User Guide

AWS CodeCommit



API Version 2015-04-13

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS CodeCommit: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

What is CodeCommit? 1
Introducing CodeCommit 1
CodeCommit, Git, and choosing the right AWS service for your needs 2
How does CodeCommit work? 5
How is CodeCommit different from file versioning in Amazon S3? 6
How do I get started with CodeCommit? 6
Where can I learn more about Git? 6
Setting up 8
View and manage your credentials 8
Setting up using Git credentials 9
Setting up using other methods 9
Compatibility for CodeCommit, Git, and other components 11
For HTTPS users using Git credentials 12
Step 1: Initial configuration for CodeCommit 12
Step 2: Install Git 13
Step 3: Create Git credentials for HTTPS connections to CodeCommit 14
Step 4: Connect to the CodeCommit console and clone the repository 15
Next steps 16
For HTTPS connections with git-remote-codecommit 17
Step 0: Install prerequisites for git-remote-codecommit 18
Step 1: Initial configuration for CodeCommit 19
Step 2: Install git-remote-codecommit 22
Step 3: Connect to the CodeCommit console and clone the repository 23
Next steps 24
For connections from development tools 25
Integrate AWS Cloud9 with AWS CodeCommit 27
Integrate Visual Studio with AWS CodeCommit 32
For SSH users not using the AWS CLI 33
Step 1: Associate your public key with your IAM user 33
Step 2: Add CodeCommit to your SSH configuration 34
Next steps 35
For SSH connections on Linux, macOS, or Unix 35
Step 1: Initial configuration for CodeCommit 35
Step 2: Install Git 36

Step 3: Configure credentials on Linux, macOS, or Unix ... 37
Step 4: Connect to the CodeCommit console and clone the repository ... 40
Next steps ... 41
For SSH connections on Windows ... 41
Step 1: Initial configuration for CodeCommit ... 42
Step 2: Install Git ... 43
Step 3: Set up the public and private keys for Git and CodeCommit ... 44
Step 4: Connect to the CodeCommit console and clone the repository ... 47
Next steps ... 48
For HTTPS connections on Linux, macOS, or Unix with the AWS CLI credential helper ... 48
Step 1: Initial configuration for CodeCommit ... 49
Step 2: Install Git ... 52
Step 3: Set up the credential helper ... 53
Step 4: Connect to the CodeCommit console and clone the repository ... 55
Next steps ... 56
For HTTPS connections on Windows with the AWS CLI credential helper ... 56
Step 1: Initial configuration for CodeCommit ... 57
Step 2: Install Git ... 60
Step 3: Set up the credential helper ... 61
Step 4: Connect to the CodeCommit console and clone the repository ... 63
Next steps ... 64
Getting started ... 65
Getting started with CodeCommit ... 65
Prerequisites ... 66
Step 1: Create a CodeCommit repository ... 66
Step 2: Add files to your repository ... 69
Step 3: Browse the contents of your repository ... 71
Step 4: Create and collaborate on a pull request ... 75
Step 5: Clean up ... 81
Step 6: Next steps ... 82
Getting started with Git and CodeCommit ... 82
Step 1: Create a CodeCommit repository ... 83
Step 2: Create a local repo ... 85
Step 3: Create your first commit ... 87
Step 4: Push your first commit ... 88
Step 5: Share the CodeCommit repository and push and pull another commit ... 88

Step 6: Create and share a branch ... 91
Step 7: Create and share a tag ... 92
Step 8: Set up access permissions ... 94
Step 9: Clean up ... 97
Product and service integrations ... 99
Integration with other AWS services ... 99
Integration examples from the community ... 106
Blog posts ... 106
Code samples ... 110
Working with repositories ... 111
Create a repository ... 112
Create a repository (console) ... 113
Create a repository (AWS CLI) ... 114
Connect to a repository ... 117
Prerequisites for connecting to a CodeCommit repository ... 117
Connect to the CodeCommit repository by cloning the repository ... 118
Connect a local repo to the CodeCommit repository ... 120
Share a repository ... 121
Choose the connection protocol to share with your users ... 122
Create IAM policies for your repository ... 123
Create an IAM group for repository users ... 125
Share the connection information with your users ... 126
Configuring notifications for repository events ... 127
Change or disable notifications ... 127
Delete notifications ... 128
Tagging a repository ... 128
Add a tag to a repository ... 129
View tags for a repository ... 131
Edit tags for a repository ... 132
Remove a tag from a repository ... 133
Manage triggers for a repository ... 135
Create the resource and add permissions for CodeCommit ... 135
Create a trigger for an Amazon SNS topic ... 136
Create a trigger for a Lambda function ... 143
Create a trigger for an existing Lambda function ... 148
Edit triggers for a repository ... 156

Test triggers for a repository ... 158
Delete triggers from a repository ... 160
Amazon CodeGuru Reviewer and CodeCommit ... 163
View repository details ... 163
View repository details (console) ... 164
View CodeCommit repository details (Git) ... 164
View CodeCommit repository details (AWS CLI) ... 166
Change repository settings ... 169
Change repository settings (console) ... 170
Change AWS CodeCommit repository settings (AWS CLI) ... 172
Sync changes between repositories ... 174
Push commits to two repositories ... 175
Configure cross-account access to a repository using roles ... 180
Cross-account repository access: Actions for the administrator in AccountA ... 181
Cross-account repository access: Actions for the administrator in AccountB ... 185
Cross-account repository access: Actions for the repository user in AccountB ... 187
Delete a repository ... 192
Delete a CodeCommit repository (console) ... 193
Delete a local repo ... 193
Delete a CodeCommit repository (AWS CLI) ... 194
Working with files ... 195
Browse files in a repository ... 196
Browse a CodeCommit repository ... 196
Create or add a file ... 197
Create or upload a file (console) ... 198
Add a file (AWS CLI) ... 199
Add a file (Git) ... 201
Edit the contents of a file ... 201
Edit a file (console) ... 201
Edit or delete a file (AWS CLI) ... 202
Edit a file (Git) ... 204
Working with pull requests ... 205
Create a pull request ... 209
Create a pull request (console) ... 209
Create a pull request (AWS CLI) ... 211
Create an approval rule ... 212

Create an approval rule for a pull request (console) ... 213
Create an approval rule for a pull request (AWS CLI) ... 215
View pull requests ... 217
View pull requests (console) ... 217
View pull requests (AWS CLI) ... 218
Review a pull request ... 222
Review a pull request (console) ... 222
Review pull requests (AWS CLI) ... 228
Update a pull request ... 233
Update a pull request (console) ... 233
Update pull requests (AWS CLI) ... 234
Edit or delete an approval rule ... 235
Edit or delete an approval rule for a pull request (console) ... 235
Edit or delete an approval rule for a pull request (AWS CLI) ... 237
Override approval rules on a pull request ... 239
Override approval rules (console) ... 239
Override approval rules (AWS CLI) ... 240
Merge a pull request ... 241
Merge a pull request (console) ... 242
Merge a pull request (AWS CLI) ... 245
Resolve conflicts in a pull request ... 251
Resolve conflicts in a pull request (console) ... 252
Resolve conflicts in a pull request (AWS CLI) ... 254
Close a pull request ... 262
Close a pull request (console) ... 263
Close a pull request (AWS CLI) ... 263
Working with approval rule templates ... 265
Create an approval rule template ... 267
Create an approval rule template (console) ... 267
Create an approval rule template (AWS CLI) ... 271
Associate an approval rule template with a repository ... 272
Associate an approval rule template (console) ... 273
Associate an approval rule template (AWS CLI) ... 273
Manage approval rule templates ... 275
Manage approval rule templates (console) ... 275
Manage approval rule templates (AWS CLI) ... 275

Disassociate an approval rule template ... 280
Disassociate an approval rule template (console) ... 280
Disassociate an approval rule template (AWS CLI) ... 280
Delete an approval rule template ... 282
Delete an approval rule template (console) ... 282
Delete an approval rule template (AWS CLI) ... 282
Working with commits ... 284
Create a commit ... 285
Create the first commit for a repository using the AWS CLI ... 285
Create a commit using a Git client ... 287
Create a commit using the AWS CLI ... 290
View commit details ... 293
Browse commits in a repository ... 294
View commit details (AWS CLI) ... 298
View commit details (Git) ... 306
Compare commits ... 308
Compare a commit to its parent ... 309
Compare any two commit specifiers ... 311
Comment on a commit ... 313
View comments on a commit in a repository ... 314
Add and reply to comments on a commit in a repository ... 314
View, add, update, and reply to commments (AWS CLI) ... 319
Create a Git tag ... 328
Use Git to create a tag ... 328
View tag details ... 329
View tag details (console) ... 329
View Git tag details (Git) ... 330
Delete a tag ... 332
Use Git to delete a Git tag ... 333
Working with branches ... 334
Create a branch ... 336
Create a branch (console) ... 336
Create a branch (Git) ... 337
Create a branch (AWS CLI) ... 338
Limit pushes and merges to branches ... 340
Configure an IAM policy to limit pushes and merges to a branch ... 340

Apply the IAM policy to an IAM group or role ... 343
Test the policy ... 343
View branch details ... 344
View branch details (console) ... 344
View branch details (Git) ... 345
View branch details (AWS CLI) ... 346
Compare and merge branches ... 347
Compare a branch to the default branch ... 348
Compare two specific branches ... 348
Merge two branches (AWS CLI) ... 349
Change branch settings ... 352
Change the default branch (console) ... 352
Change the default branch (AWS CLI) ... 353
Delete a branch ... 354
Delete a branch (console) ... 355
Delete a branch (AWS CLI) ... 355
Delete a branch (Git) ... 356
Working with user preferences ... 358
Migrate to CodeCommit ... 359
Migrate a Git repository to AWS CodeCommit ... 359
Step 0: Setup required for access to CodeCommit ... 360
Step 1: Create a CodeCommit repository ... 365
Step 2: Clone the repository and push to the CodeCommit repository ... 367
Step 3: View files in CodeCommit ... 369
Step 4: Share the CodeCommit repository ... 369
Migrate content to CodeCommit ... 372
Step 0: Setup required for access to CodeCommit ... 373
Step 1: Create a CodeCommit repository ... 378
Step 2: Migrate local content to the CodeCommit repository ... 379
Step 3: View files in CodeCommit ... 381
Step 4: Share the CodeCommit repository ... 381
Migrate a repository in increments ... 384
Step 0: Determine whether to migrate incrementally ... 384
Step 1: Install prerequisites and add the CodeCommit repository as a remote ... 385
Step 2: Create the script to use for migrating incrementally ... 387
Step 3: Run the script and migrate incrementally to CodeCommit ... 387

Appendix: Sample script `incremental-repo-migration.py` ... 389
Security ... 397
Data protection ... 397
AWS KMS and encryption ... 398
Using rotating credentials ... 400
Identity and Access Management ... 405
Audience ... 406
Authenticating with identities ... 406
Managing access using policies ... 407
Authentication and access control ... 409
How AWS CodeCommit works with IAM ... 474
CodeCommit resource-based policies ... 475
Authorization based on CodeCommit tags ... 475
CodeCommit IAM roles ... 478
Identity-based policy examples ... 479
Troubleshooting ... 482
Resilience ... 484
Infrastructure security ... 485
Monitoring CodeCommit ... 486
Monitoring CodeCommit events ... 486
referenceCreated event ... 488
referenceUpdated event ... 488
referenceDeleted event ... 489
unreferencedMergeCommitCreated event ... 490
commentOnCommitCreated event ... 491
commentOnCommitUpdated event ... 492
commentOnPullRequestCreated event ... 493
commentOnPullRequestUpdated event ... 494
pullRequestCreated event ... 495
pullRequestSourceBranchUpdated event ... 496
pullRequestStatusChanged event ... 497
pullRequestMergeStatusUpdated event ... 498
approvalRuleTemplateCreated event ... 499
approvalRuleTemplateUpdated event ... 499
approvalRuleTemplateDeleted event ... 500
approvalRuleTemplateAssociatedWithRepository event ... 501

approvalRuleTemplateDisassociatedWithRepository event ... 502
approvalRuleTemplateBatchAssociatedWithRepositories event ... 503
approvalRuleTemplateBatchDisassociatedFromRepositories event ... 503
pullRequestApprovalRuleCreated event ... 504
pullRequestApprovalRuleDeleted event ... 505
pullRequestApprovalRuleOverridden event ... 507
pullRequestApprovalStateChanged event ... 509
pullRequestApprovalRuleUpdated event ... 511
reactionCreated event ... 512
reactionUpdated event ... 513
Logging AWS CodeCommit API calls with AWS CloudTrail ... 514
CodeCommit information in CloudTrail ... 514
Understanding CodeCommit log file entries ... 515
AWS CloudFormation resources ... 523
CodeCommit and CloudFormation templates ... 523
Template examples ... 524
CloudFormation, CodeCommit, and the AWS Cloud Development Kit (AWS CDK) ... 525
Learn more about CloudFormation ... 526
Troubleshooting ... 527
Troubleshooting Git credentials (HTTPS) ... 527
Git credentials for AWS CodeCommit: I keep seeing a prompt for credentials when I connect to my CodeCommit repository at the terminal or command line ... 528
Git credentials for AWS CodeCommit: I set up Git credentials, but my system is not using them ... 528
Troubleshooting git-remote-codecommit ... 529
I see an error: git: 'remote-codecommit' is not a git command ... 529
I see an error: fatal: Unable to find remote helper for 'codecommit' ... 530
Cloning error: I cannot clone a CodeCommit repository from an IDE ... 530
Push or pull error: I cannot push or pull commits from an IDE to a CodeCommit repository ... 530
Installation error: I see an externally-managed-environment error when I try to install git-remote-codecommit ... 531
Troubleshooting SSH connections ... 531
Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems ... 532

Access error: Public key is uploaded successfully to IAM and SSH tested successfully but connection fails on Windows systems 533
Authentication challenge: Authenticity of host can't be established when connecting to a CodeCommit repository 534
IAM error: 'Invalid format' when attempting to add a public key to IAM 541
I need to access CodeCommit repositories in multiple Amazon Web Services accounts with SSH credentials 541
Git on Windows: Bash emulator or command line freezes when attempting to connect using SSH 542
Public key format requires specification in some distributions of Linux 543
Access error: SSH public key denied when connecting to a CodeCommit repository 543
Troubleshooting the credential helper (HTTPS) 543
I receive an error when running the `git config` command to configure the credential helper 544
I see an error reading the user name when attempting to clone a repository 544
I get a command not found error in Windows when using the credential helper 545
I am prompted for a user name when I connect to a CodeCommit repository 546
Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository (403) 546
Git for Windows: I installed Git for Windows, but I am denied access to my repository (403) 549
Troubleshooting Git clients 552
Git error: Error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly 552
Git error: Too many reference update commands 552
Git error: Push via HTTPS is broken in some versions of Git 553
Git error: 'gnutls_handshake() failed' 553
Git error: Git cannot find the CodeCommit repository or does not have permission to access the repository 553
Git on Windows: No supported authentication methods available (publickey) 553
Troubleshooting access errors 554
Access error: I am prompted for a user name and password when I connect to a CodeCommit repository from Windows 554
Access error: Public key denied when connecting to a CodeCommit repository 555
Access error: "Rate Exceeded" or "429" message when connecting to a CodeCommit repository 555

Troubleshooting configuration errors ... 556
Configuration error: Cannot configure AWS CLI credentials on macOS ... 557
Troubleshooting console errors ... 557
Access error: Encryption key access denied for a CodeCommit repository from the console or AWS CLI ... 555
Encryption error: Repository can't be decrypted ... 558
Console error: Cannot browse the code in a CodeCommit repository from the console ... 558
Display error: Cannot view a file or a comparison between files ... 558
Troubleshooting triggers ... 559
Trigger error: A repository trigger does not run when expected ... 559
Turn on debugging ... 559
CodeCommit reference ... 561
Regions and Git connection endpoints ... 561
Supported AWS Regions for CodeCommit ... 561
Git connection endpoints ... 563
Server fingerprints for CodeCommit ... 570
Using AWS CodeCommit with interface VPC endpoints ... 577
Availability ... 578
Create VPC endpoints for CodeCommit ... 579
Create a VPC endpoint policy for CodeCommit ... 580
Quotas ... 581
Command line reference ... 589
Basic Git commands ... 595
Configuration variables ... 595
Remote repositories ... 596
Commits ... 597
Branches ... 599
Tags ... 600
Document history ... 602
Earlier updates ... 611
AWS Glossary ... 619

What is AWS CodeCommit?

AWS CodeCommit is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud. For information about pricing for CodeCommit, see Pricing.

Note

CodeCommit is in scope with many compliance programs. For details about AWS and compliance efforts, see AWS Services In Scope by Compliance Program.
This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see HIPAA Overview.
For information about this service and ISO 27001, a security management standard that specifies security management best practices, see ISO 27001 Overview.
For information about this service and the Payment Card Industry Data Security Standard (PCI DSS), see PCI DSS Overview.
For information about this service and the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard that specifies the security requirements for cryptographic modules that protect sensitive information, see Federal Information Processing Standard (FIPS) 140-2 Overview and Git connection endpoints.

Topics

- Introducing CodeCommit
- CodeCommit, Git, and choosing the right AWS service for your needs
- How does CodeCommit work?
- How is CodeCommit different from file versioning in Amazon S3?
- How do I get started with CodeCommit?
- Where can I learn more about Git?

Introducing CodeCommit

CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. CodeCommit eliminates the need for you to manage your own source control system

or worry about scaling its infrastructure. You can use CodeCommit to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with your existing Git-based tools.

With CodeCommit, you can:

- **Benefit from a fully managed service hosted by AWS**. CodeCommit provides high service availability and durability and eliminates the administrative overhead of managing your own hardware and software. There is no hardware to provision and scale and no server software to install, configure, and update.
- **Store your code securely**. CodeCommit repositories are encrypted at rest as well as in transit.
- **Work collaboratively on code.** CodeCommit repositories support pull requests, where users can review and comment on each other's code changes before merging them to branches; notifications that automatically send emails to users about pull requests and comments; and more.
- **Easily scale your version control projects**. CodeCommit repositories can scale up to meet your development needs. The service can handle repositories with large numbers of files or branches, large file sizes, and lengthy revision histories.
- **Store anything, anytime**. CodeCommit has no limit on the size of your repositories or on the file types you can store.
- **Integrate with other AWS and third-party services**. CodeCommit keeps your repositories close to your other production resources in the AWS Cloud, which helps increase the speed and frequency of your development lifecycle. It is integrated with IAM and can be used with other AWS services and in parallel with other repositories. For more information, see Product and service integrations with AWS CodeCommit.
- **Easily migrate files from other remote repositories**. You can migrate to CodeCommit from any Git-based repository.
- **Use the Git tools you already know**. CodeCommit supports Git commands as well as its own AWS CLI commands and APIs.

CodeCommit, Git, and choosing the right AWS service for your needs

As a Git-based service, CodeCommit is well suited to most version control needs. There are no arbitrary limits on file size, file type, and repository size. However, there are inherent limitations

to Git that can negatively affect the performance of certain kinds of operations, particularly over time. You can avoid potential degradation of CodeCommit repository performance by avoiding using it for use cases where other AWS services are better suited to the task. You can also optimize Git performance for complex repositories. Here are some use cases where Git, and therefore CodeCommit, might not be the best solution for you, or where you might need to take additional steps to optimize for Git.

Use case	Description	Other services to consider
Large files that change frequently	Git uses delta encoding to store differences between versions of files. For example, if you change a few words in a document, Git will only store those changed words. If you have files or objects over 5 MB with many changes, Git might need to reconstru ct a large chain of delta differences. This can consume an increasing amount of compute resources on both your local computer and in CodeCommit as these files grow over time.	To version large files, consider Amazon Simple Storage Service (Amazon S3). For more information, see Using Versioning in the *Amazon Simple Storage Service User Guide*.
Database	Git repositories grow larger over time. Because versioning tracks all changes, any change will increase your repositor y size. In other words, as you commit data, even if you delete data in a commit, data is added to a repositor y. As there is more data to process and transmit over time, Git will slow down. This	To create and use a database with consistent performance regardless of size, consider Amazon DynamoDB. For more information, see the Amazon DynamoDB Getting Started Guide.

Use case	Description	Other services to consider
	is particularly detrimental to a database use case. Git was not designed as a database.	
Audit trails	Typically, audit trails are kept for long periods of time and are continuously generated by system processes at a very frequent cadence. Git was designed to securely store source code generated by groups of developers on a development cycle. Rapidly changing repositories that continually store programmatically-generated system changes will see performance degrade over time.	To store audit trails, consider Amazon Simple Storage Service (Amazon S3). To audit AWS activity, depending on your use case, consider using AWS CloudTrail, AWS Config, or Amazon CloudWatch.
Backups	Git was designed to version source code written by developers. You can push commits to two remote repositories, including a CodeCommit repository, as a backup strategy. However, Git was not designed to handle backups of your computer file system, database dumps, or similar backup content. Doing so might slow down your system and increase the amount of time required to clone and push a repository.	For information about backing up to the AWS Cloud, see Backup & Restore.

Use case	Description	Other services to consider

Large numbers of branches or references	When a Git client pushes or pulls repository data, the remote server must send all branches and references such as tags, even if you are only interested in a single branch. If you have thousands of branches and references, this can take time to process and send (pack negotiati on) and result in apparently slow repository response. The more branches and tags you have, the longer this process can take. We recommend using CodeCommit, but delete branches and tags that are no longer needed.	To analyze the number of references in a CodeCommit repository to determine which might not be needed, you can use one of the following commands:
		• Linux, macOS, or Unix, or Bash emulator on Windows:
		`git ls-remote | wc -l`
		• Powershell:
		`git ls-remote | Measure-Object -line`

How does CodeCommit work?

CodeCommit is familiar to users of Git-based repositories, but even those unfamiliar should find the transition to CodeCommit relatively simple. CodeCommit provides a console for the easy creation of repositories and the listing of existing repositories and branches. In a few simple steps, users can find information about a repository and clone it to their computer, creating a local repo where they can make changes and then push them to the CodeCommit repository. Users can work from the command line on their local machines or use a GUI-based editor.

1. Use the AWS CLI or the CodeCommit console to create a CodeCommit repository.
2. From your development machine, use Git to run **git clone**, specifying the name of the CodeCommit repository. This creates a local repo that connects to the CodeCommit repository.

3. Use the local repo on your development machine to modify (add, edit, and delete) files, and then run **git add** to stage the modified files locally. Run **git commit** to commit the files locally, and then run **git push** to send the files to the CodeCommit repository.
4. Download changes from other users. Run **git pull** to synchronize the files in the CodeCommit repository with your local repo. This ensures you're working with the latest version of the files.

You can use the AWS CLI or the CodeCommit console to track and manage your repositories.

How is CodeCommit different from file versioning in Amazon S3?

CodeCommit is optimized for team software development. It manages batches of changes across multiple files, which can occur in parallel with changes made by other developers. Amazon S3 versioning supports the recovery of past versions of files, but it's not focused on collaborative file tracking features that software development teams need.

How do I get started with CodeCommit?

To get started with CodeCommit:

1. Follow the steps in Setting up to prepare your development machines.
2. Follow the steps in one or more of the tutorials in Getting started.
3. Create version control projects in CodeCommit or migrate version control projects to CodeCommit.

Where can I learn more about Git?

If you don't know it already, you should learn how to use Git. Here are some helpful resources:

- Pro Git, an online version of the *Pro Git* book. Written by Scott Chacon. Published by Apress.
- Git Immersion, a try-it-yourself guided tour that walks you through the fundamentals of using Git. Published by Neo Innovation, Inc.
- Git Reference, an online quick reference that can also be used as a more in-depth Git tutorial. Published by the GitHub team.
- Git Cheat Sheet with basic Git command syntax. Published by the GitHub team.

- Git Pocket Guide. Written by Richard E. Silverman. Published by O'Reilly Media, Inc.

Setting up for AWS CodeCommit

You can sign in to the AWS Management Console and upload, add, or edit a file to a repository directly from the AWS CodeCommit console. This is a quick way to make a change. However, if you want to work with multiple files, files across branches, and so on, consider setting up your local computer to work with repositories. The easiest way to set up CodeCommit is to configure HTTPS Git credentials for AWS CodeCommit. This HTTPS authentication method:

- Uses a static user name and password.
- Works with all operating systems supported by CodeCommit.
- Is also compatible with integrated development environments (IDEs) and other development tools that support Git credentials.

You can use other methods if you do not want to or cannot use Git credentials for operational reasons. For example, if you access CodeCommit repositories using federated access, temporary credentials, or a web identity provider, you cannot use Git credentials. We recommend that you set up your local computer using the `git-remote-codecommit` command. Review these options carefully, to decide which alternative method works best for you.

- Setting up using Git credentials
- Setting up using other methods
- Compatibility for CodeCommit, Git, and other components

For information about using CodeCommit and Amazon Virtual Private Cloud, see Using AWS CodeCommit with interface VPC endpoints.

View and manage your credentials

You can view and manage your CodeCommit credentials from the AWS console through **My Security Credentials**.

> **Note**
>
> This option is not available for users using federated access, temporary credentials, or a web identity provider.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation bar on the upper right, choose your user name, and then choose **Security Credentials**.
3. Choose the **AWS CodeCommit credentials** tab.

Setting up using Git credentials

With HTTPS connections and Git credentials, you generate a static user name and password in IAM. You then use these credentials with Git and any third-party tool that supports Git user name and password authentication. This method is supported by most IDEs and development tools. It is the simplest and easiest connection method to use with CodeCommit.

- For HTTPS users using Git credentials: Follow these instructions to set up connections between your local computer and CodeCommit repositories using Git credentials.
- For connections from development tools: Follow these guidelines to set up connections between your IDE or other development tools and CodeCommit repositories using Git credentials. IDEs that support Git credentials include (but are not limited to) Visual Studio, Xcode, and IntelliJ.

Setting up using other methods

You can use the SSH protocol instead of HTTPS to connect to your CodeCommit repository. With SSH connections, you create public and private key files on your local machine that Git and CodeCommit use for SSH authentication. You associate the public key with your IAM user. You store the private key on your local machine. Because SSH requires manual creation and management of public and private key files, you might find Git credentials simpler and easier to use with CodeCommit.

Unlike Git credentials, SSH connection setup varies, depending on the operating system on your local computer.

- For SSH users not using the AWS CLI: Follow these abbreviated instructions if you already have a public-private key pair and are familiar with SSH connections on your local computer.
- For SSH connections on Linux, macOS, or Unix: Follow these instructions for a step-by-step walkthrough of creating a public-private key pair and setting up connections on Linux, macOS, or Unix operating systems.

- For SSH connections on Windows: Follow these instructions for a step-by-step walkthrough of creating public-private key pair and setting up connections on Windows operating systems.

If you are connecting to CodeCommit and AWS using federated access, an identity provider, or temporary credentials, or if you do not want to configure IAM users or Git credentials for IAM users, you can set up connections to CodeCommit repositories in one of two ways:

- Install and use **git-remote-codecommit** (recommended).
- Install and use the credential helper included in the AWS CLI.

Both methods support accessing CodeCommit repositories without requiring an IAM user, which means that you can connect to repositories using federated access and temporary credentials. The git-remote-codecommit utility is the recommended approach. It extends Git and is compatible with a variety of Git versions and credential helpers. However, not all IDEs support the clone URL format used by `git-remote-codecommit`. You might have to manually clone repositories to your local computer before you can work with them in your IDE.

- Follow the instructions in Setup Steps for HTTPS Connections to AWS CodeCommit Repositories with git-remote-codecommit to install and set up **git-remote-codecommit** on Windows, Linux, macOS, or Unix.

The credential helper included in the AWS CLI allows Git to use HTTPS and a cryptographically signed version of your IAM user credentials or Amazon EC2 instance role whenever Git needs to authenticate with AWS to interact with CodeCommit repositories. Some operating systems and Git versions have their own credential helpers, which conflict with the credential helper included in the AWS CLI. They can cause connectivity issues for CodeCommit.

- For HTTPS connections on Linux, macOS, or Unix with the AWS CLI credential helper: Follow these instructions for a step-by-step walkthrough of installing and setting up the credential helper on Linux, macOS, or Unix systems.
- For HTTPS connections on Windows with the AWS CLI credential helper: Follow these instructions for a step-by-step walkthrough of installing and setting up the credential helper on Windows systems.

If you are connecting to a CodeCommit repository that is hosted in another Amazon Web Services account, you can configure access and set up connections using roles, policies, and the credential helper included in the AWS CLI.

- Configure cross-account access to an AWS CodeCommit repository using roles: Follow these instructions for a step-by-step walkthrough of configuring cross-account access in one Amazon Web Services account to users in an IAM group in another Amazon Web Services account.

Compatibility for CodeCommit, Git, and other components

When you work with CodeCommit, you use Git. You might use other programs, too. The following table provides the latest guidance for version compatibility. As a best practice, we recommend that you use the latest versions of Git and other software.

Version compatibility information for AWS CodeCommit

Component	Version
Git	CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuri ng the branch name for initial commits. We recommend using a recent version of Git.
Curl	CodeCommit requires curl 7.33 and later. However, there is a known issue with HTTPS and curl update 7.41.0. For more information, see Troubleshooting.
Python (**git-remote-codecommit** only)	**git-remote-codecommit** requires version 3 and later.
Pip (**git-remote-codecommit** only)	**git-remote-codecommit** requires version 9.0.3 and later.
AWS CLI (**git-remote-codecommit** only)	We recommend a recent version of AWS CLI version 2 for all CodeCommit users. **git-remot e-codecommit** requires AWS CLI version 2 to support AWS SSO and connections

Component	Version
	that require temporary credentials, such as federated users.

Setup for HTTPS users using Git credentials

The simplest way to set up connections to AWS CodeCommit repositories is to configure Git credentials for CodeCommit in the IAM console, and then use those credentials for HTTPS connections. You can also use these same credentials with any third-party tool or integrated development environment (IDE) that supports HTTPS authentication using a static user name and password. For examples, see For connections from development tools.

> **Note**
>
> If you have previously configured your local computer to use the credential helper for CodeCommit, you must edit your .gitconfig file to remove the credential helper information from the file before you can use Git credentials. If your local computer is running macOS, you might need to clear cached credentials from Keychain Access.

Step 1: Initial configuration for CodeCommit

Follow these steps to set up an Amazon Web Services account, create an IAM user, and configure access to CodeCommit.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the

AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. We recommend that you create a profile for using the AWS CLI with CodeCommit. For more information, see Command line reference and Using named profiles.

Step 2: Install Git

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

Note

Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

Step 3: Create Git credentials for HTTPS connections to CodeCommit

After you have installed Git, create Git credentials for your IAM user in IAM.

To set up HTTPS Git credentials for CodeCommit

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

   Make sure to sign in as the IAM user who will create and use the Git credentials for connections to CodeCommit.

2. In the IAM console, in the navigation pane, choose **Users**, and from the list of users, choose your IAM user.

   > **Note**
   >
   > You can directly view and manage your CodeCommit credentials in **My Security Credentials**. For more information, see View and manage your credentials.

3. On the user details page, choose the **Security Credentials** tab. In the **API keys** section, choose **Generate API Key**. Select **AWS CodeCommit** as the service.

   > **Note**
   >
   > You cannot choose your own user name or password for Git credentials. For more information, see Use Git Credentials and HTTPS with CodeCommit.

4. Copy the user name and password that IAM generated for you, either by showing, copying, and then pasting this information into a secure file on your local computer, or by choosing **Download credentials** to download this information as a .CSV file. You need this information to connect to CodeCommit.
   Downloading Git credentials from the IAM console

   After you have saved your credentials, choose **Close**.

> **Important**
>
> This is your only chance to save the user name and password. If you do not save them, you can copy the user name from the IAM console, but you cannot look up the password. You must reset the password and then save it.

Step 4: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the name and connection details for the CodeCommit repository, you can skip this step and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

   - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.
   - Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
   - Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

> **Note**
>
> If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4. Open a terminal, command line, or Git shell. Run the **git clone** command with the HTTPS clone URL you copied to clone the repository. For example, to clone a repository named *MyDemoRepo* to a local repo named *my-demo-repo* in the US East (Ohio) Region:

   ```
   git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-
   demo-repo
   ```

   The first time you connect, you are prompted for the user name and password for the repository. Depending on the configuration of your local computer, this prompt either originates from a credential management system for the operating system, a credential manager utility for your version of Git (for example, the Git Credential Manager included in Git for Windows), your IDE, or Git itself. Enter the user name and password generated for Git credentials in IAM (the ones you created in Step 3: Create Git credentials for HTTPS connections to CodeCommit). Depending on your operating system and other software, this information might be saved for you in a credential store or credential management utility. If so, you should not be prompted again unless you change the password, inactivate the Git credentials, or delete the Git credentials in IAM.

   If you do not have a credential store or credential management utility configured on your local computer, you can install one. For more information about Git and how it manages credentials, see Credential Storage in the Git documentation.

   For more information, see Connect to the CodeCommit repository by cloning the repository and Create a commit.

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

To learn how to create and push your first commit, see Create a commit in AWS CodeCommit. If you're new to Git, you might also want to review the information in Where can I learn more about Git? and Getting started with Git and AWS CodeCommit.

Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit

If you want to connect to CodeCommit using a root account, federated access, or temporary credentials, you should set up access using **git-remote-codecommit**. This utility provides a simple method for pushing and pulling code from CodeCommit repositories by extending Git. It is the recommended method for supporting connections made with federated access, identity providers, and temporary credentials. To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

You can also use **git-remote-codecommit** with an IAM user. Unlike other HTTPS connection methods, **git-remote-codecommit** does not require setting up Git credentials for the user.

> **Note**
>
> Some IDEs do not support the clone URL format used by `git-remote-codecommit`. You might have to manually clone repositories to your local computer before you can work with them in your preferred IDE. For more information, see Troubleshooting git-remote-codecommit and AWS CodeCommit.

These procedures are written with the assumption that you have an Amazon Web Services account, have created at least one repository in CodeCommit, and use an IAM user with a managed policy when connecting to CodeCommit repositories. For information about how to configure access for federated users and other rotating credential types, see Connecting to AWS CodeCommit repositories with rotating credentials.

Topics

- Step 0: Install prerequisites for git-remote-codecommit
- Step 1: Initial configuration for CodeCommit
- Step 2: Install git-remote-codecommit

- Step 3: Connect to the CodeCommit console and clone the repository
- Next steps

Step 0: Install prerequisites for git-remote-codecommit

Before you can use **git-remote-codecommit**, you must install some prerequisites on your local computer. These include:

- A supported version of Python. For more information about supported versions of Python, see git-remote-codecommit.

  For more information about setting up and using git-remote-codecommit, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.
- Git

> **Note**
>
> When you install Python on Windows, make sure that you choose the option to add Python to the path.

git-remote-codecommit requires pip version 9.0.3 or later. To check your version of pip, open a terminal or command line and run the following command:

```
pip --version
```

You can run the following two commands to update your version of pip to the latest version:

```
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py --user
```

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

Step 1: Initial configuration for CodeCommit

Follow these steps to create an IAM user, configure it with the appropriate policies, obtain an access key and secret key, and install and configure the AWS CLI.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

   For more information, see Getting Set Up with the AWS Command Line Interface.

   > **Note**
   >
   > CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

   ```
   aws codecommit help
   ```

   This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as

us-east-2. When prompted for the default output format, specify json. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI
- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- ap-southeast-3
- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1

- eu-west-2
- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1
- ap-northeast-3
- af-south-1
- il-central-1

For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

Step 2: Install git-remote-codecommit

Follow these steps to install **git-remote-codecommit**.

To install git-remote-codecommit

1. At the terminal or command line, run the following command:

```
pip install git-remote-codecommit
```

> **Note**
>
> Depending on your operating system and configuration, you might need to run this command with elevated permissions, such as sudo, or use the --user parameter to install to a directory that doesn't require special privileges, such as your current user account. For example, on a computer running Linux, macOS, or Unix:
>
> ```
> sudo pip install git-remote-codecommit
> ```
>
> On a computer running Windows:
>
> ```
> pip install --user git-remote-codecommit
> ```

2. Monitor the installation process until you see a success message.

Step 3: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the clone URL to use with **git-remote-codecommit** for the CodeCommit repository, you can skip connecting to the console and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

   - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.
   - Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
   - Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

> **Note**
>
> If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4. At the terminal or command prompt, clone the repository with the **git clone** command. Use the HTTPS git-remote-codecommit URL you copied and the name of the AWS CLI profile, if you created a named profile. If you do not specify a profile, the command assumes the default profile. The local repo is created in a subdirectory of the directory where you run the command. For example, to clone a repository named *MyDemoRepo* to a local repo named *my-demo-repo*:

   ```
   git clone codecommit://MyDemoRepo my-demo-repo
   ```

   To clone the same repository using a profile named *CodeCommitProfile*:

   ```
   git clone codecommit://CodeCommitProfile@MyDemoRepo my-demo-repo
   ```

   To clone a repository in a different AWS Region than the one configured in your profile, include the AWS Region name. For example:

   ```
   git clone codecommit::ap-northeast-1://MyDemoRepo my-demo-repo
   ```

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

To learn how to create and push your first commit, see Create a commit in AWS CodeCommit. If you're new to Git, you might also want to review the information in Where can I learn more about Git? and Getting started with Git and AWS CodeCommit.

Set up connections from development tools using Git credentials

After you have configured Git credentials for AWS CodeCommit in the IAM console, you can use those credentials with any development tool that supports Git credentials. For example, you can configure access to your CodeCommit repository in AWS Cloud9, Visual Studio, Xcode, IntelliJ, or any integrated development environment (IDE) that integrates Git credentials. After you configure access, you can edit your code, commit your changes, and push directly from the IDE or other development tool.

> **Note**
>
> If you access CodeCommit repositories using federated access, temporary credentials, or a web idenity provider, you cannot use Git credentials. We recommend that you set up your local computer using the `git-remote-codecommit` command. However, not all IDEs are fully compatible with Git remote helpers such as **git-remote-codecommit**. If you encounter problems, see Troubleshooting git-remote-codecommit and AWS CodeCommit.

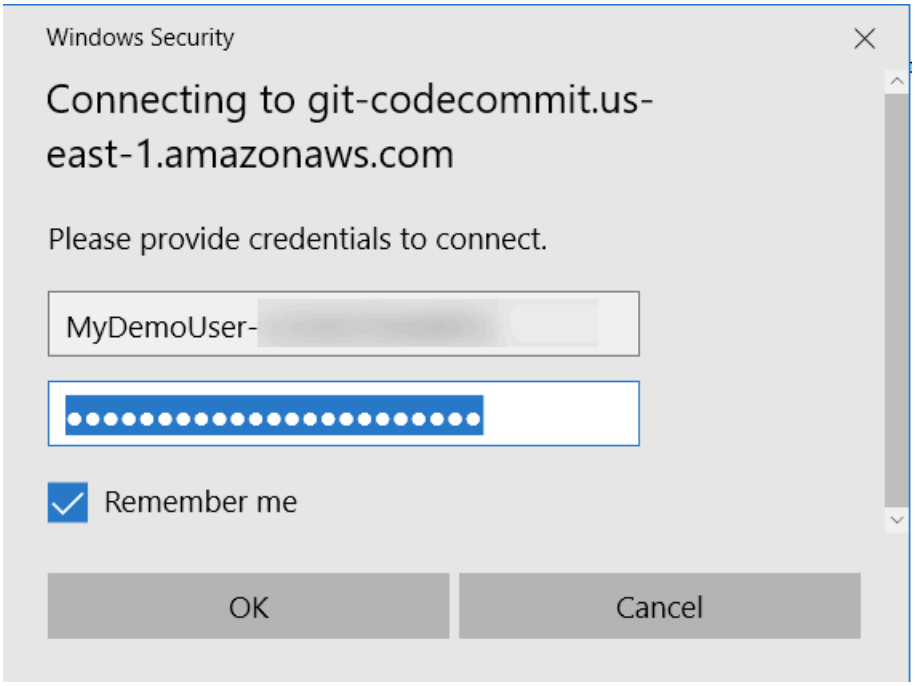
Topics

- Integrate AWS Cloud9 with AWS CodeCommit
- Integrate Visual Studio with AWS CodeCommit

When prompted by your IDE or development tool for the user name and password used to connect to the CodeCommit repository, provide the Git credentials for **User name** and **Password** you created in IAM.

For more information about AWS Regions and endpoints for CodeCommit, see Regions and Git connection endpoints.

You might also see a prompt from your operating system to store your user name and password. For example, in Windows, you would provide your Git credentials as follows:

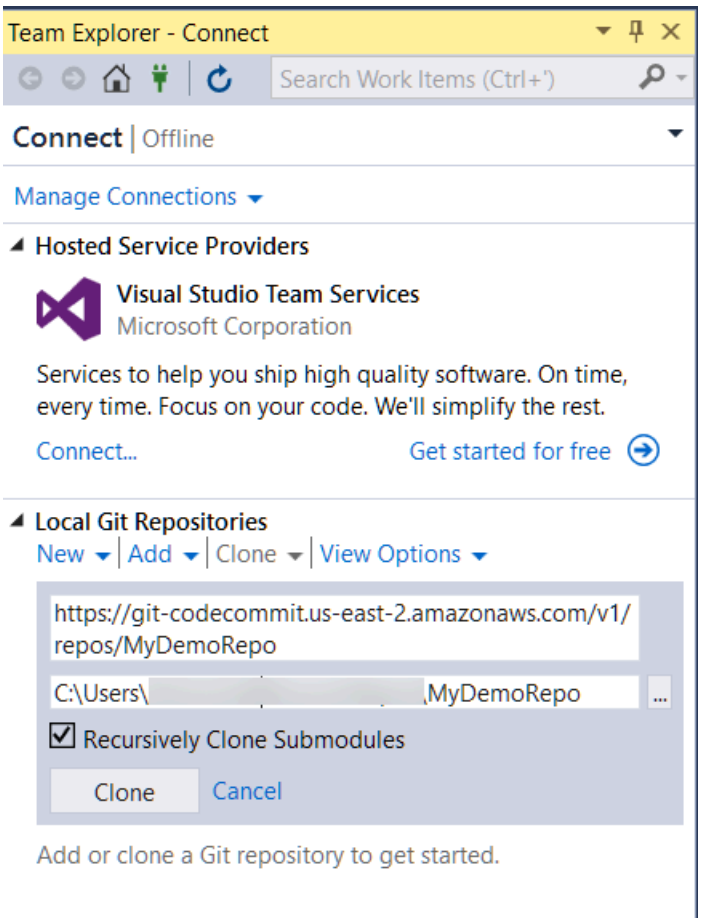


For information about configuring Git credentials for a particular software program or development tool, consult the product documentation.

The following is not a comprehensive list of IDEs. The links are provided solely to help you learn more about these tools. AWS is not responsible for the content of any of these topics.

- AWS Cloud9
- Visual Studio

  Alternatively, install the AWS Toolkit for Visual Studio. For more information, see Integrate Visual Studio with AWS CodeCommit.



- XCode

Integrate AWS Cloud9 with AWS CodeCommit

You can use AWS Cloud9 to make code changes in a CodeCommit repository. AWS Cloud9 contains a collection of tools that you can use to write code and build, run, test, debug, and release software. You can clone existing repositories, create repositories, commit and push code changes to a repository, and more, all from your AWS Cloud9 EC2 development environment. The AWS Cloud9 EC2 development environment is generally preconfigured with the AWS CLI, an Amazon EC2 role, and Git, so in most cases, you can run a few simple commands and start interacting with your repository.

To use AWS Cloud9 with CodeCommit, you need the following:

- An AWS Cloud9 EC2 development environment running on Amazon Linux.
- The AWS Cloud9 IDE open in a web browser.

- An IAM user with one of the CodeCommit managed policies and one of the AWS Cloud9 managed policies applied to it.

  For more information, see AWS managed policies for CodeCommit and Understanding and Getting Your Security Credentials.

> **Note**
>
> This topic describes setting up integration with CodeCommit and AWS Cloud9 with general access from the Internet. You can set up access to CodeCommit and AWS Cloud9 in an isolated environment, but that requires additional steps. For more information, see:
>
> - Using AWS CodeCommit with interface VPC endpoints
> - Accessing no-ingress Amazon EC2 instances with AWS Systems Manager
> - Working with Shared Environments
> - Share your VPC with other accounts
> - Blog post: Isolating network access to your AWS Cloud9 environments

Topics

- Step 1: Create an AWS Cloud9 development environment
- Step 2: Configure the AWS CLI credential helper on your AWS Cloud9 EC2 development environment
- Step 3: Clone a CodeCommit repository into your AWS Cloud9 EC2 development environment
- Next steps

Step 1: Create an AWS Cloud9 development environment

AWS Cloud9 hosts your development environment on an Amazon EC2 instance. This is the easiest way to integrate, because you can use the AWS managed temporary credentials for the instance to connect to your CodeCommit repository. If you want to use your own server instead, see the AWS Cloud9 User Guide.

To create an AWS Cloud9 environment

1. Sign in to AWS as the IAM user you've configured and open the AWS Cloud9 console.
2. In the AWS Cloud9 console, choose **Create environment**.

3. In **Step 1: Name environment**, enter a name and optional description for the environment, and then choose **Next step**.
4. In **Step 2: Configure Settings**, configure your environment as follows:
   - In **Environment type**, choose **Create a new instance for environment (EC2)**.
   - In **Instance type**, choose the appropriate instance type for your development environment. For example, if you're just exploring the service, you might choose the default of t2.micro. If you intend to use this environment for development work, choose a larger instance type.
   - Accept the other default settings unless you have reasons to choose otherwise (for example, your organization uses a specific VPC, or your Amazon Web Services account does not have any VPCs configured), and then choose **Next step**.
5. In **Step 3: Review**, review your settings. Choose **Previous step** if you want to make any changes. If not, choose **Create environment**.

   Creating an environment and connecting to it for the first time takes several minutes. If it seems to take an unusually long time, see Troubleshooting in the *AWS Cloud9 User Guide*.
6. After you are connected to your environment, check to see if Git is already installed and is a supported version by running the **git --version** command in the terminal window.

   If Git is not installed, or if it is not a supported version, install a supported version. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git. To install Git, we recommend websites such as Git Downloads.

   Tip

   Depending on the operating system of your environment, you might be able to use the **yum** command with the **sudo** option to install updates, including Git. For example, an administrative command sequence might resemble the following three commands:

   ```
   sudo yum -y update
   sudo yum -y install git
   git --version
   ```

7. Configure a user name and email to be associated with your Git commits by running the **git config** command. For example:

```
    git config --global user.name "Mary Major"
    git config --global user.email mary.major@example.com
```

Step 2: Configure the AWS CLI credential helper on your AWS Cloud9 EC2 development environment

After you've created an AWS Cloud9 environment, you can configure the AWS CLI credential helper to manage the credentials for connections to your CodeCommit repository. The AWS Cloud9 development environment comes with AWS managed temporary credentials that are associated with your IAM user. You use these credentials with the AWS CLI credential helper.

1. Open the terminal window and run the following command to verify that the AWS CLI is installed:

   ```
   aws --version
   ```

   If successful, this command returns the currently installed version of the AWS CLI. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. At the terminal, run the following commands to configure the AWS CLI credential helper for HTTPS connections:

   ```
   git config --global credential.helper '!aws codecommit credential-helper $@'
   git config --global credential.UseHttpPath true
   ```

   Tip

   The credential helper uses the default Amazon EC2 instance role for your development environment. If you intend to use the development environment to connect to repositories that are not hosted in CodeCommit, either configure SSH connections to those repositories, or configure a local `.gitconfig` file to use an alternative credential management system when connecting to those other repositories. For more information, see Git Tools - Credential Storage on the Git website.

Step 3: Clone a CodeCommit repository into your AWS Cloud9 EC2 development environment

After you've configured the AWS CLI credential helper, you can clone your CodeCommit repository onto it. Then you can start working with the code.

1. In the terminal, run the **git clone** command, specifying the HTTPS clone URL of the repository you want to clone. For example, if you want to clone a repository named MyDemoRepo in the US East (Ohio) Region, you would enter:

   ```
   git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
   ```

   > **Tip**
   >
   > You can find the Clone URL for your repository in the CodeCommit console by choosing **Clone URL**.

2. When the cloning is complete, in the side navigation, expand the folder for your repository, and choose the file you want to open for editing. Alternatively, choose **File** and then choose **New File** to create a file.
3. When you have finished editing or creating files, in the terminal window, change directories to your cloned repository and then commit and push your changes. For example, if you added a new file named *MyFile.py*:

   ```
   cd MyDemoRepo
   git commit -a MyFile.py
   git commit -m "Added a new file with some code improvements"
   git push
   ```

Next steps

For more information, see the AWS Cloud9 User Guide and CodeCommit sample for AWS Cloud9. For more information about using Git with CodeCommit, see Getting started with Git and AWS CodeCommit.

Integrate Visual Studio with AWS CodeCommit

You can use Visual Studio to make code changes in a CodeCommit repository. The AWS Toolkit for Visual Studio now includes features that make working with CodeCommit easier and more convenient when working in Visual Studio.. The Toolkit for Visual Studio integration is designed to work with Git credentials and an IAM user. You can clone existing repositories, create repositories, commit and push code changes to a repository, and more.

> **Important**
>
> The Toolkit for Visual Studio is available for installation on Windows operating systems only. If you are looking for information about working with Visual Studio Code, see AWS Toolkit for Visual Studio Code.

If you've used the Toolkit for Visual Studio before, you're probably already familiar with setting up AWS credential profiles that contain an access key and secret key. Credential profiles are used in the Toolkit for Visual Studio to enable calls to AWS service APIs (for example, to Amazon S3 to list buckets or to CodeCommit to list repositories). To pull and push code to a CodeCommit repository, you also need Git credentials. If you don't have Git credentials, the Toolkit for Visual Studio can generate and apply those credentials for you. This can save you a lot of time.

To use Visual Studio with CodeCommit, you need the following:

- An IAM user with a valid set of credentials (an access key and secret key) configured for it. This IAM user should also have:

  One of the CodeCommit managed policies and the IAMSelfManageServiceSpecificCredentials managed policy applied to it.

  OR

  If the IAM user already has Git credentials configured, one of the CodeCommit managed policies or equivalent permissions.

  For more information, see AWS managed policies for CodeCommit and Understanding and Getting Your Security Credentials.
- The AWS Toolkit for Visual Studio installed on the computer where you've installed Visual Studio. For more information, see Setting Up the AWS Toolkit for Visual Studio.

For more information on using AWS Toolkit for Visual Studio with CodeCommit, see Using AWS CodeCommit with Visual Studio Team Explorer in the *Toolkit for Visual Studio User Guide*.

Setup for SSH users not using the AWS CLI

If you want to use SSH connections for your repository, you can connect to AWS CodeCommit without installing the AWS CLI. The AWS CLI includes commands that are useful when you use and manage CodeCommit repositories, but it is not required for initial setup.

This topic assumes:

- You have set up an IAM user with the policies or permissions required for CodeCommit and the **IAMUserSSHKeys** managed policy or equivalent permissions required for uploading keys. For more information, see Using identity-based policies (IAM Policies) for CodeCommit.
- You already have, or know how to create, a public-private key pair. We strongly recommend that you use a secure passphrase for your SSH key.
- You are familiar with SSH, your Git client, and its configuration files.
- If you are using Windows, you have installed a command-line utility, such as Git Bash, that emulates the bash shell.

If you need more guidance, follow the instructions in For SSH connections on Linux, macOS, or Unix or For SSH connections on Windows.

Topics

- Step 1: Associate your public key with your IAM user
- Step 2: Add CodeCommit to your SSH configuration
- Next steps

Step 1: Associate your public key with your IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the IAM console, in the navigation pane, choose **Users**, and from the list of users, choose your IAM user.
3. On the **Security Credentials** tab, choose **Upload SSH public key**.

4. Paste the contents of your SSH public key into the field, and then choose **Upload SSH Key**.

> **Tip**
>
> The public-private key pair must be SSH-2 RSA, in OpenSSH format, and contain 2048 bits. The key looks similar to this:
>
> ```
> ssh-rsa EXAMPLE-
> AfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJB
> gNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb2
> 5zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhc
> NMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAw
> DgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAS=EXAMPLE user-
> name@ip-192-0-2-137
> ```
>
> IAM accepts public keys in the OpenSSH format only. If you provide your public key in another format, you see an error message that says the key format is not valid.

5. Copy the SSH key ID (for example, *APKAEIBAERJR2EXAMPLE*) and close the console.

Step 2: Add CodeCommit to your SSH configuration

1. At the terminal (Linux, macOS, or Unix) or bash emulator (Windows), edit your SSH configuration file by typing **cat>> ~/.ssh/config**:

> **Tip**
>
> If you have more than one SSH configuration, make sure you include the blank lines before and after the content. Save the file by pressing the `Ctrl` and d keys simultaneously.

2. Run the following command to test your SSH configuration:

   Enter the passphrase for your SSH key file when prompted. If everything is configured correctly, you should see the following success message:

```
You have successfully authenticated over SSH. You can use Git to interact with
 CodeCommit.
```

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

To connect to a repository, follow the steps in Connect to a repository. To create a repository, follow the steps in Create a repository.

Setup steps for SSH connections to AWS CodeCommit repositories on Linux, macOS, or Unix

Before you can connect to CodeCommit for the first time, you must complete some initial configuration steps. After you set up your computer and AWS profile, you can connect to a CodeCommit repository and clone that repository to your computer (also known as creating a local repo). If you're new to Git, you might also want to review the information in Where can I learn more about Git?.

Topics

- Step 1: Initial configuration for CodeCommit
- Step 2: Install Git
- Step 3: Configure credentials on Linux, macOS, or Unix
- Step 4: Connect to the CodeCommit console and clone the repository
- Next steps

Step 1: Initial configuration for CodeCommit

Follow these steps to set up an Amazon Web Services account, create an IAM user, and configure access to CodeCommit.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

> **Note**
>
> CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

> **Note**
>
> If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

Step 2: Install Git

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

Step 3: Configure credentials on Linux, macOS, or Unix

SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit

To set up the public and private keys for Git and CodeCommit

1. From the terminal on your local machine, run the **ssh-keygen** command, and follow the directions to save the file to the .ssh directory for your profile.

   > **Note**
   >
   > Be sure to check with your system administrator about where key files should be stored and which file naming pattern should be used.

   For example:

```
$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/user-name/.ssh/id_rsa): Type /home/
your-user-name/.ssh/ and a file name here, for example /home/your-user-name/.ssh/
codecommit_rsa

Enter passphrase (empty for no passphrase): <Type a passphrase, and then press
 Enter>
Enter same passphrase again: <Type the passphrase again, and then press Enter>

Your identification has been saved in /home/user-name/.ssh/codecommit_rsa.
Your public key has been saved in /home/user-name/.ssh/codecommit_rsa.pub.
```

```
The key fingerprint is:
45:63:d5:99:0e:99:73:50:5e:d4:b3:2d:86:4a:2c:14 user-name@client-name
The key's randomart image is:
+--[ RSA 2048]----+
|        E.+.o*.++|
|        .o .=.=o.|
|       . ..  *. +|
|        ..o . +..|
|        So . . . |
|          .      |
|                 |
|                 |
|                 |
+-----------------+
```

This generates:

- The *codecommit_rsa* file, which is the private key file.
- The *codecommit_rsa.pub* file, which is the public key file.

> **Tip**
>
> By default, **ssh-keygen** generates a 2048 bit key. You can use the -t and -b parameters to specify the type and length of the key. If you want a 4096 bit key in the rsa format, you would specify this by running the command with the following parameters:
>
> ```
> ssh-keygen -t rsa -b 4096
> ```
>
> For more information about the formats and lengths required for SSH keys, see Using IAM with CodeCommit.

2. Run the following command to display the value of the public key file (*codecommit_rsa.pub*):

```
cat ~/.ssh/codecommit_rsa.pub
```

Copy this value. It looks similar to the following:

```
ssh-rsa EXAMPLE-AfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJB
```

```
gNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb2
5zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhc
NMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAw
DgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAS=EXAMPLE user-name@ip-192-0-2-137
```

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

> **Note**
>
> You can directly view and manage your CodeCommit credentials in **My Security Credentials**. For more information, see View and manage your credentials.

4. In the IAM console, in the navigation pane, choose **Users**, and from the list of users, choose your IAM user.
5. On the user details page, choose the **Security Credentials** tab, and then choose **Upload SSH public key**.
6. Paste the contents of your SSH public key into the field, and then choose **Upload SSH public key**.
7. Copy or save the information in **SSH Key ID** (for example, *APKAEIBAERJR2EXAMPLE*).

> **Note**
>
> If you have more than one SSH key IDs uploaded, the keys are listed alphabetically by key ID, not by upload date. Make sure that you have copied the key ID that is associated with the correct upload date.

8. On your local machine, use a text editor to create a config file in the ~/.ssh directory, and then add the following lines to the file, where the value for *User* is the SSH key ID you copied earlier:

> **Note**
>
> If you gave your private key file a name other than *codecommit_rsa*, be sure to use it here.
> You can set up SSH access to repositories in multiple Amazon Web Services accounts, For more information, see Troubleshooting SSH connections to AWS CodeCommit.

Save and name this file `config`.

9. From the terminal, run the following command to change the permissions for the config file:

```
chmod 600 config
```

10. Run the following command to test your SSH configuration:

    After you have confirmed the connection, you should see confirmation that you have added the server to your known hosts file and a successful connection message. If you do not see a success message, check that you saved the `config` file in the ~/.ssh directory of the IAM user you configured for access to CodeCommit, and that you specified the correct private key file.

    For information to help you troubleshoot problems, run the `ssh` command with the `-v` parameter. For example:

    For information to help you troubleshoot connection problems, see Troubleshooting SSH connections to AWS CodeCommit.

Step 4: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the name and connection details for the CodeCommit repository, you can skip this step and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

    - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.

- Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
- Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

> **Note**
>
> If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4.

> **Note**
>
> If you successfully tested your connection, but the clone command fails, you might not have the required access to your config file, or another setting might be in conflict with your config file. Try connecting again, this time including the SSH key ID in the command. For example:
> For more information, see Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems.

For more information about how to connect to repositories, see Connect to the CodeCommit repository by cloning the repository.

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

Setup steps for SSH connections to AWS CodeCommit repositories on Windows

Before you can connect to AWS CodeCommit for the first time, you must complete some initial configuration steps. After you set up your computer and AWS profile, you can connect to a CodeCommit repository and clone that repository to your computer (also known as creating a local

repo). If you're new to Git, you might also want to review the information in Where can I learn more about Git?.

Topics

- Step 1: Initial configuration for CodeCommit
- Step 2: Install Git
- Step 3: Set up the public and private keys for Git and CodeCommit
- Step 4: Connect to the CodeCommit console and clone the repository
- Next steps

Step 1: Initial configuration for CodeCommit

Follow these steps to set up an Amazon Web Services account, create an IAM user, and configure access to CodeCommit.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.

7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

> **Note**
>
> If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

Step 2: Install Git

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

If the version of Git you installed does not include a Bash emulator, such as Git Bash, install one. You use this emulator instead of the Windows command line when you configure SSH connections.

Step 3: Set up the public and private keys for Git and CodeCommit

To set up the public and private keys for Git and CodeCommit on Windows

1. Open the Bash emulator.

> **Note**
>
> You might need to run the emulator with administrative permissions.

From the emulator, run the **ssh-keygen** command, and follow the directions to save the file to the .ssh directory for your profile.

For example:

```
$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/drive/Users/user-name/.ssh/id_rsa): Type a
 file name here, for example /c/Users/user-name/.ssh/codecommit_rsa

Enter passphrase (empty for no passphrase): <Type a passphrase, and then press
 Enter>
Enter same passphrase again: <Type the passphrase again, and then press Enter>

Your identification has been saved in drive/Users/user-name/.ssh/codecommit_rsa.
Your public key has been saved in drive/Users/user-name/.ssh/codecommit_rsa.pub.
The key fingerprint is:
45:63:d5:99:0e:99:73:50:5e:d4:b3:2d:86:4a:2c:14 user-name@client-name
The key's randomart image is:
+--[ RSA 2048]----+
|        E.+.o*.++|
|        .o .=.=o.|
|       . ..  *. +|
|        ..o . +..|
|        So . . . |
|          .      |
|                 |
|                 |
|                 |
```

```
+-----------------+
```

This generates:

- The *codecommit_rsa* file, which is the private key file.
- The *codecommit_rsa*.pub file, which is the public key file.

> **Tip**
>
> By default, **ssh-keygen** generates a 2048 bit key. You can use the -t and -b parameters to specify the type and length of the key. If you want a 4096 bit key in the rsa format, you would specify this by running the command with the following parameters:
>
> ```
> ssh-keygen -t rsa -b 4096
> ```
>
> For more information about the formats and lengths required for SSH keys, see Using IAM with CodeCommit.

2. Run the following commands to display the value of the public key file (*codecommit_rsa*.pub):

```
cd .ssh
notepad codecommit_rsa.pub
```

Copy the contents of the file, and then close Notepad without saving. The contents of the file look similar to the following:

```
ssh-rsa EXAMPLE-AfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJB
gNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb2
5zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wHhc
NMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAw
DgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAS=EXAMPLE user-name@computer-name
```

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

> **Note**
>
> You can directly view and manage your CodeCommit credentials in **My Security Credentials**. For more information, see View and manage your credentials.

4. In the IAM console, in the navigation pane, choose **Users**, and from the list of users, choose your IAM user.
5. On the user details page, choose the **Security Credentials** tab, and then choose **Upload SSH public key**.
6. Paste the contents of your SSH public key into the field, and then choose **Upload SSH public key**.
7. Copy or save the information in **SSH Key ID** (for example, *APKAEIBAERJR2EXAMPLE*).

> **Note**
>
> If you have more than one SSH key IDs uploaded, the keys are listed alphabetically by key ID, not by upload date. Make sure that you have copied the key ID that is associated with the correct upload date.

8. In the Bash emulator, run the following commands to create a config file in the ~/.ssh directory, or edit it if one already exists:

```
notepad ~/.ssh/config
```

9. Add the following lines to the file, where the value for *User* is the SSH key ID you copied earlier, and the value for *IdentityFile* is the path to and name of the private key file:

> **Note**
>
> If you gave your private key file a name other than *codecommit_rsa*, be sure to use it here.
> You can set up SSH access to repositories in multiple Amazon Web Services accounts, For more information, see Troubleshooting SSH connections to AWS CodeCommit.

Save the file as config (not config.txt), and then close Notepad.

> **Important**
>
> The name of the file must be `config` with no file extension. Otherwise, the SSH connections fail.

10. Run the following command to test your SSH configuration:

    After you have confirmed the connection, you should see confirmation that you have added the server to your known hosts file and a successful connection message. If you do not see a success message, double-check that you saved the `config` file in the ~/.ssh directory of the IAM user you configured for access to CodeCommit, that the `config` file has no file extension (for example, it must not be named config.txt), and that you specified the correct private key file (*`codecommit_rsa`*, not *`codecommit_rsa`*.pub).

    To troubleshoot problems, run the `ssh` command with the `-v` parameter. For example:

    For information to help you troubleshoot connection problems, see Troubleshooting SSH connections to AWS CodeCommit.

Step 4: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the name and connection details for the CodeCommit repository, you can skip this step and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

    - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.

- Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
- Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

> **Note**
>
> If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4. Alternatively, open a command prompt, and using the URL and the SSH key ID for the public key you uploaded to IAM, run the **git clone** command. The local repo is created in a subdirectory of the directory where you run the command. For example, to clone a repository named *MyDemoRepo* to a local repo named *my-demo-repo*:

   For more information, see Connect to the CodeCommit repository by cloning the repository and Create a commit.

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

Setup steps for HTTPS connections to AWS CodeCommit repositories on Linux, macOS, or Unix with the AWS CLI credential helper

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. For most users, this can be done most easily by following the steps in For HTTPS users using Git credentials. However, if you want to connect to CodeCommit using a root account, federated access, or temporary credentials, you can use the credential helper that is included in the AWS CLI.

> **Note**
>
> Although the credential helper is a supported method for connecting to CodeCommit using federated access, an identity provider, or temporary credentials, the recommended method is to install and use the **git-remote-codecommit** utility. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

Topics

- Step 1: Initial configuration for CodeCommit
- Step 2: Install Git
- Step 3: Set up the credential helper
- Step 4: Connect to the CodeCommit console and clone the repository
- Next steps

Step 1: Initial configuration for CodeCommit

Follow these steps to set up an Amazon Web Services account, create and configure an IAM user, and install the AWS CLI.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

   For more information, see Getting Set Up with the AWS Command Line Interface.

   > **Note**
   >
   > CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

```
aws codecommit help
```

This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

```
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI
- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1

- ap-southeast-2
- ap-southeast-3
- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1
- ap-northeast-3
- af-south-1
- il-central-1

For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

Step 2: Install Git

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28

supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

Step 3: Set up the credential helper

1. From the terminal, use Git to run **git config**, specifying the use of the Git credential helper with the AWS credential profile, and enabling the Git credential helper to send the path to repositories:

```
git config --global credential.helper '!aws codecommit credential-helper $@'
git config --global credential.UseHttpPath true
```

> **Tip**
>
> The credential helper uses the default AWS credential profile or the Amazon EC2 instance role. You can specify a profile to use, such as `CodeCommitProfile`, if you have created an AWS credential profile to use with CodeCommit:
>
> ```
> git config --global credential.helper '!aws --profile CodeCommitProfile
>  codecommit credential-helper $@'
> ```
>
> If your profile name contains spaces, make sure you enclose the name in quotation marks (").
> You can configure profiles per repository instead of globally by using `--local` instead of `--global`.

The Git credential helper writes the following value to `~/.gitconfig`:

```
[credential]
    helper = !aws --profile CodeCommitProfile codecommit credential-helper $@
    UseHttpPath = true
```

> **Important**
>
> If you want to use a different IAM user on the same local machine for CodeCommit, you must run the **git config** command again and specify a different AWS credential profile.

2. Run **git config --global --edit** to verify the preceding value has been written to `~/.gitconfig`. If successful, you should see the preceding value (in addition to values that might already exist in the Git global configuration file). To exit, typically you would type **`:q`**, and then press Enter.

   If you experience problems after you configure your credential helper, see Troubleshooting.

   > **Important**
   >
   > If you are using macOS, use the following steps to ensure the credential helper is configured correctly.

3. If you are using macOS, use HTTPS to connect to a CodeCommit repository. After you connect to a CodeCommit repository with HTTPS for the first time, subsequent access fails after about 15 minutes. The default Git version on macOS uses the Keychain Access utility to store credentials. For security measures, the password generated for access to your CodeCommit repository is temporary, so the credentials stored in the keychain stop working after about 15 minutes. To prevent these expired credentials from being used, you must either:

   - Install a version of Git that does not use the keychain by default.
   - Configure the Keychain Access utility to not provide credentials for CodeCommit repositories.

   1. Open the Keychain Access utility. (You can use Finder to locate it.)
   2. Search for `git-codecommit.us-east-2.amazonaws.com`. Highlight the row, open the context menu or right-click it, and then choose **Get Info**.

3. Choose the **Access Control** tab.
4. In **Confirm before allowing access**, choose `git-credential-osxkeychain`, and then choose the minus sign to remove it from the list.

> **Note**
>
> After you remove `git-credential-osxkeychain` from the list, you see a pop-up message whenever you run a Git command. Choose **Deny** to continue. If you find the pop-ups too disruptive, here are some other options:
>
> - Connect to CodeCommit using SSH instead of HTTPS. For more information, see For SSH connections on Linux, macOS, or Unix.
> - In the Keychain Access utility, on the **Access Control** tab for `git-codecommit.us-east-2.amazonaws.com`, choose the **Allow all applications to access this item (access to this item is not restricted)** option. This prevents the pop-ups, but the credentials eventually expire (on average, in about 15 minutes) and you see a 403 error message. When this happens, you must delete the keychain item to restore functionality.
> - For more information, see Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository (403).

Step 4: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the name and connection details for the CodeCommit repository, you can skip this step and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

- Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.
- Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
- Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

> **Note**
>
> If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4.

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

Setup steps for HTTPS connections to AWS CodeCommit repositories on Windows with the AWS CLI credential helper

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. For most users, this can be done most easily by following the steps in For HTTPS users using Git credentials. However, if you want to connect to CodeCommit using a root account, federated access, or temporary credentials, you can use the credential helper that is included in the AWS CLI.

> **Note**
>
> Although the credential helper is a supported method for connecting to CodeCommit using federated access, an identity provider, or temporary credentials, the recommended method

is to install and use the **git-remote-codecommit** utility. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

This topic walks you through the steps to install the AWS CLI, set up your computer and AWS profile, connect to a CodeCommit repository, and clone that repository to your computer, also known as creating a local repo. If you're new to Git, you might also want to review the information in Where can I learn more about Git?.

Topics

- Step 1: Initial configuration for CodeCommit
- Step 2: Install Git
- Step 3: Set up the credential helper
- Step 4: Connect to the CodeCommit console and clone the repository
- Next steps

Step 1: Initial configuration for CodeCommit

Follow these steps to set up an Amazon Web Services account, create and configure an IAM user, and install the AWS CLI. The AWS CLI includes a credential helper that you configure for HTTPS connections to your CodeCommit repositories.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

> **Note**
>
> CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

   For more information, see Getting Set Up with the AWS Command Line Interface.

   > **Note**
   >
   > CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

```
aws codecommit help
```

This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

```
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI
- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1

- ap-southeast-2
- ap-southeast-3
- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1
- ap-northeast-3
- af-south-1
- il-central-1

For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

Step 2: Install Git

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28

supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git for Windows. If you use this link to install Git, you can accept all of the installation default settings except for the following:

- When prompted during the **Adjusting your PATH environment** step, choose the option to use Git from the command line.
- (Optional) If you intend to use HTTPS with the credential helper that is included in the AWS CLI instead of configuring Git credentials for CodeCommit, on the **Configuring extra options** page, make sure the **Enable Git Credential Manager** option is cleared. The Git Credential Manager is only compatible with CodeCommit if IAM users configure Git credentials. For more information, see For HTTPS users using Git credentials and Git for Windows: I installed Git for Windows, but I am denied access to my repository (403).

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

Step 3: Set up the credential helper

The AWS CLI includes a Git credential helper you can use with CodeCommit. The Git credential helper requires an AWS *credential profile*, which stores a copy of an IAM user's AWS access key ID and AWS secret access key (along with a default AWS Region name and default output format). The Git credential helper uses this information to automatically authenticate with CodeCommit so you don't need to enter this information every time you use Git to interact with CodeCommit.

1. Open a command prompt and use Git to run **git config**, specifying the use of the Git credential helper with the AWS credential profile, which enables the Git credential helper to send the path to repositories:

   ```
   git config --global credential.helper "!aws codecommit credential-helper $@"
   git config --global credential.UseHttpPath true
   ```

   The Git credential helper writes the following to the .gitconfig file:

```
[credential]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true
```

Important

- If you are using a Bash emulator instead of the Windows command line, you must use single quotes instead of double quotes.
- The credential helper uses the default AWS profile or the Amazon EC2 instance role. If you have created an AWS credential profile to use, such as *CodeCommitProfile*, you can modify the command as follows to use it instead:

  ```
  git config --global credential.helper "!aws codecommit credential-helper
   --profile CodeCommitProfile $@"
  ```

  This writes the following to the .gitconfig file:

  ```
  [credential]
      helper = !aws codecommit credential-helper --profile=CodeCommitProfile
   $@
      UseHttpPath = true
  ```

- If your profile name contains spaces, you must edit your .gitconfig file after you run this command to enclose it in single quotation marks ('). Otherwise, the credential helper does not work.
- If your installation of Git for Windows included the Git Credential Manager utility, you see 403 errors or prompts to provide credentials into the Credential Manager utility after the first few connection attempts. The most reliable way to solve this problem is to uninstall and then reinstall Git for Windows without the option for the Git Credential Manager utility, because it is not compatible with CodeCommit. If you want to keep the Git Credential Manager utility, you must perform additional configuration steps to also use CodeCommit, including manually modifying the .gitconfig file to specify the use of the credential helper for AWS CodeCommit when connecting to CodeCommit. Remove any stored credentials from the Credential Manager utility (you can find this utility in Control Panel). After you

have removed any stored credentials, add the following to your .gitconfig file, save it, and then try connecting again from a new command prompt window:

```
[credential "https://git-codecommit.us-east-2.amazonaws.com"]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true

[credential "https://git-codecommit.us-east-1.amazonaws.com"]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true
```

You might also have to reconfigure your **git config** settings by specifying **--system** instead of **--global** or **--local** before all connections work as expected.

- If you want to use different IAM users on the same local machine for CodeCommit, you should specify **git config --local** instead of **git config --global**, and run the configuration for each AWS credential profile.

2. Run **git config --global --edit** to verify the preceding values have been written to the .gitconfig file for your user profile (by default, `%HOME%\.gitconfig` or *`drive`*`:\Users\`*`UserName`*`\.gitconfig`). If successful, you should see the preceding values (in addition to values that might already exist in the Git global configuration file). To exit, typically you would type **`:q`** and then press Enter.

Step 4: Connect to the CodeCommit console and clone the repository

If an administrator has already sent you the name and connection details for the CodeCommit repository, you can skip this step and clone the repository directly.

To connect to a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.

3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

   - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.
   - Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
   - Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

   > **Note**
   >
   > If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4. On some versions of Windows, you might see a pop-up message asking for your user name and password. This is the built-in credential management system for Windows, but it is not compatible with the credential helper for AWS CodeCommit. Choose **Cancel**.

Next steps

You have completed the prerequisites. Follow the steps in Getting started with CodeCommit to start using CodeCommit.

Getting started

The easiest way to get started with CodeCommit is to follow the steps in Getting started with CodeCommit . If you are new to Git and CodeCommit, you should also consider following the steps in Getting started with Git and CodeCommit. This helps you familiarize yourself with CodeCommit and the basics of using Git when interacting with your CodeCommit repositories.

You can also follow the tutorial in Simple Pipeline Walkthrough with CodePipeline and CodeCommit to learn how to use your CodeCommit repository as part of a continuous delivery pipeline.

The tutorials in this section are written with the assumption that you have completed the prerequisites and setup, including:

- Assigning permissions to the IAM user.
- Setting up credential management for HTTPS or SSH connections on the local machine you are using for this tutorial.
- Configuring the AWS CLI if you want to use the command line or terminal for all operations, including creating the repository.

Topics

- Getting started with AWS CodeCommit
- Getting started with Git and AWS CodeCommit

Getting started with AWS CodeCommit

This tutorial shows you how to use some key CodeCommit features. First, you create a repository and commit some changes to it. Then, you browse the files and view the changes. You can also create a pull request so other users can review and comment on changes to your code.

If you want to migrate existing code to CodeCommit, see Migrate to AWS CodeCommit.

If you are unfamiliar with Git, consider completing Getting started with Git and CodeCommit too. After you complete these tutorials, you should have enough practice to start using CodeCommit for your own projects and in team environments.

The CodeCommit console includes helpful information in a collapsible panel that you can open from the information icon

()

or any **Info** link on the page. You can close this panel at any time.

The CodeCommit console also provides a way to quickly search for your resources, such as repositories, build projects, deployment applications, and pipelines. Choose **Go to resource** or press the / key, and then type the name of the resource. Any matches appear in the list. Searches are case insensitive. You only see resources that you have permissions to view. For more information, see Viewing resources in the console.

Prerequisites

Before you begin, you must complete the prerequisites and setup procedure, including:

- Assigning permissions to the IAM user.
- Setting up credential management for HTTPS or SSH connections on the local machine you use for this tutorial.
- Configuring the AWS CLI if you want to use the command line or terminal for all operations, including to create the repository.

Topics

- Step 1: Create a CodeCommit repository
- Step 2: Add files to your repository
- Step 3: Browse the contents of your repository
- Step 4: Create and collaborate on a pull request
- Step 5: Clean up
- Step 6: Next steps

Step 1: Create a CodeCommit repository

You can use the CodeCommit console to create a CodeCommit repository. If you already have a repository you want to use for this tutorial, you can skip this step.

> **Note**
>
> Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the CodeCommit product information page.

To create the CodeCommit repository

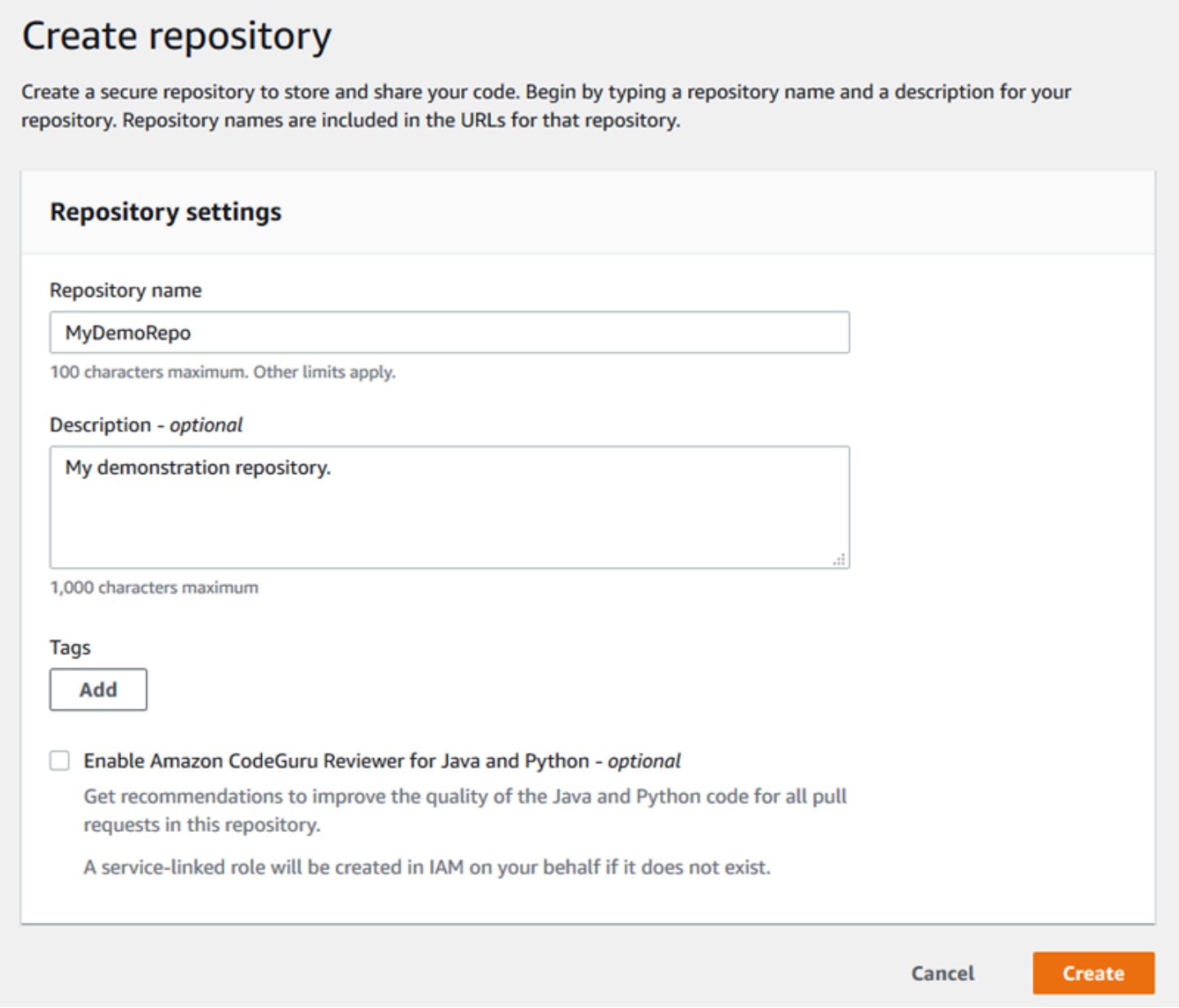
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Use the Region selector to choose the AWS Region where you want to create the repository. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose **Create repository**.
4. On the **Create repository** page, in **Repository name**, enter a name for your repository (for example, **`MyDemoRepo`**).

   > **Note**
   >
   > Repository names are case sensitive and can be no longer than 100 characters. For more information, see Limits.

5. (Optional) In **Description**, enter a description (for example, **`My demonstration repository`**). This can help you and other users identify the purpose of the repository.
6. (Optional) Choose **Add tag** to add one or more repository tags (a custom attribute label that helps you organize and manage your AWS resources) to your repository. For more information, see Tagging repositories in AWS CodeCommit.
7. (Optional) Expand **Additional configuration** to specify whether to use the default AWS managed key or your own customer managed key for encrypting and decrypting the data in this repository. If you choose to use your own customer managed key, you must ensure that it is available in the AWS Region where you are creating the repository, and that the key is active. For more information, see AWS Key Management Service and encryption for AWS CodeCommit repositories.
8. (Optional) Select **Enable Amazon CodeGuru Reviewer for Java and Python** if this repository will contain Java or Python code, and you want to have CodeGuru Reviewer analyze that code. CodeGuru Reviewer uses multiple machine learning models to find code defects and to

automatically suggest improvements and fixes in pull requests. For more information, see the Amazon CodeGuru Reviewer User Guide.

9. Choose **Create**.



Note

If you use a name other than `MyDemoRepo` for your repository, be sure to use it in the remaining steps.

When the repository opens, you see information about how to add files directly from the CodeCommit console.

Step 2: Add files to your repository

You can add files to your repository by:

- Creating a file in the CodeCommit console. If you create the first file for a repository in the console, a branch is created for you named *main*. This branch is the default branch for your repository.
- Uploading a file from your local computer using the CodeCommit console. If you upload the first file for a repository from the console, a branch is created for you named *main*. This branch is the default branch for your repository.
- Using a Git client to clone the repository to your local computer, and then adding, committing, and pushing files to the CodeCommit repository. A branch is created for you as part of that first commit from Git, and it is set as the default branch for your repository. The name of the branch is the default choice of your Git client. Consider configuring your Git client to use *main* as the name for the initial branch.

> **Note**
>
> You can create branches and change the default branch for a repository at any time. For more information, see Working with branches in AWS CodeCommit repositories.

The simplest way to get started is to open the CodeCommit console and add a file. By doing so, you also create a default branch for your repository named *main*. For instructions about how to add a file and create a first commit to a repository using the AWS CLI, see Create the first commit for a repository using the AWS CLI.

To add a file to the repository

1. In the navigation bar for the repository, choose **Code**.
2. Choose **Add file**, and then choose whether to create a file or upload a file from your computer. This tutorial shows you how to do both.
3. To add a file, do the following:

a. In the drop-down list of branches, choose the branch where you want to add the file. The default branch is selected automatically for you. In the example shown here, the default branch is named *main*. If you want to add the file to a different branch, choose a different branch.

b. In **File name**, enter a name for the file. In the code editor, enter the code for the file.

c. In **Author name**, enter the name you want displayed to other repository users.

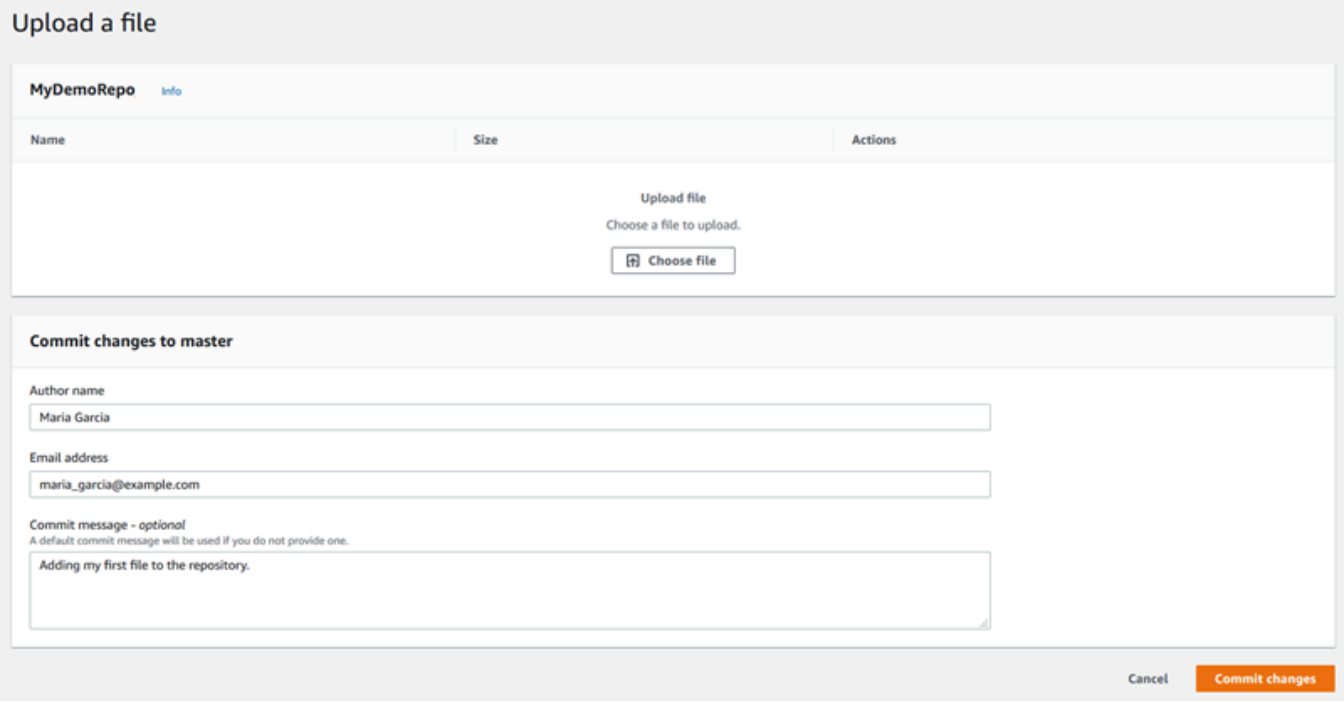
d. In **Email address**, enter an email address.

e. (Optional) In **Commit message**, enter a brief message. Although this is optional, we recommend that you add a commit message to help your team members understand why you added this file. If you do not enter a commit message, a default message is used.

f. Choose **Commit changes**.

To upload a file, do the following:

- If you're uploading a file, choose the file you want to upload.



- In **Author name**, enter the name you want displayed to other repository users.
- In **Email address**, enter an email address.

- (Optional) In **Commit message**, enter a brief message. Although this is optional, we recommend that you add a commit message to help your team members understand why you added this file. If you do not enter a commit message, a default message is used.
- Choose **Commit changes**.

For more information, see Working with files in AWS CodeCommit repositories.

To use a Git client to clone the repository, install Git on your local computer, and then clone the CodeCommit repository. Add some files to the local repo and push them to the CodeCommit repository. For an in-depth introduction, try the Getting started with Git and CodeCommit. If you are familiar with Git, but are not sure how to do this with a CodeCommit repository, you can view examples and instructions in Create a commit, Step 2: Create a local repo, or Connect to a repository.

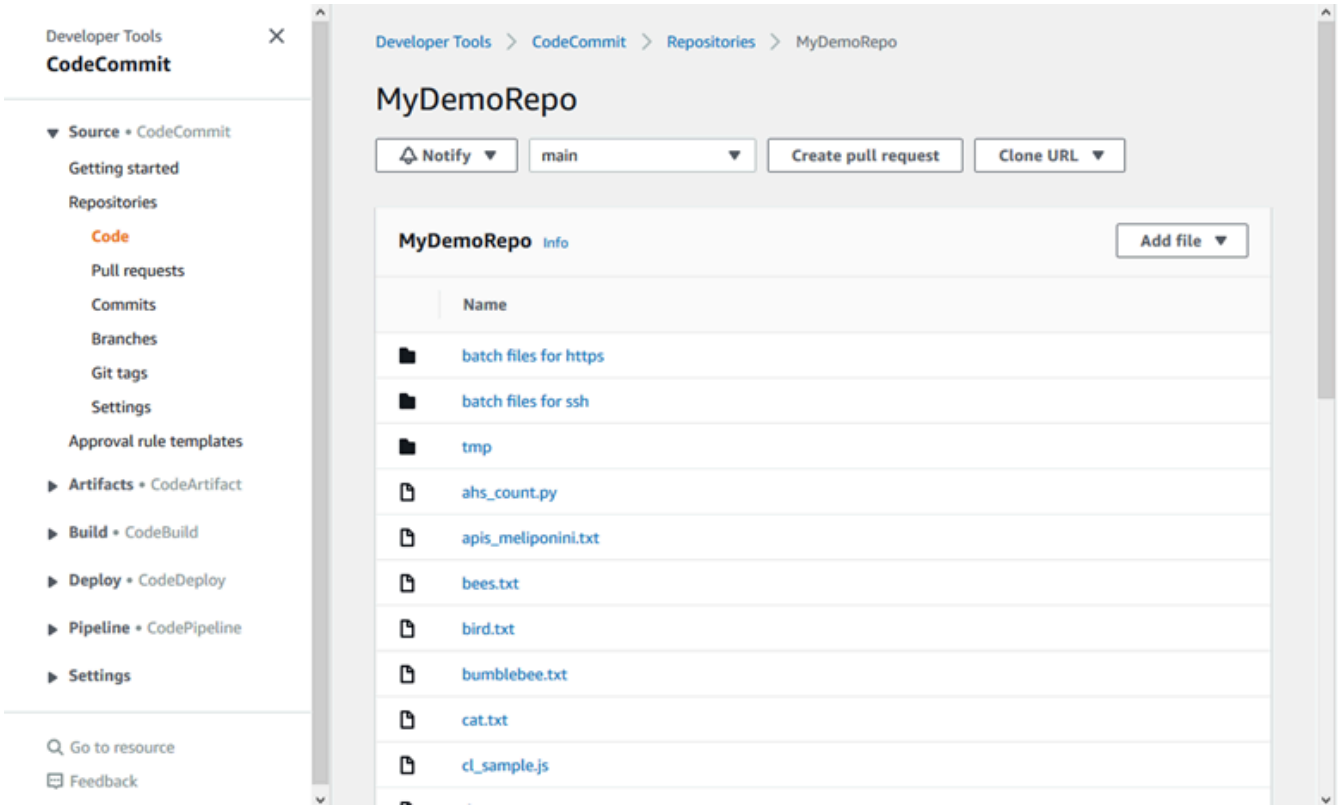
After you have added some files to the CodeCommit repository, you can view them in the console.

Step 3: Browse the contents of your repository

You can use the CodeCommit console to review the files in a repository or quickly read the contents of a file. This helps you determine which branch to check out or whether to create a local copy of a repository.

To browse the repository

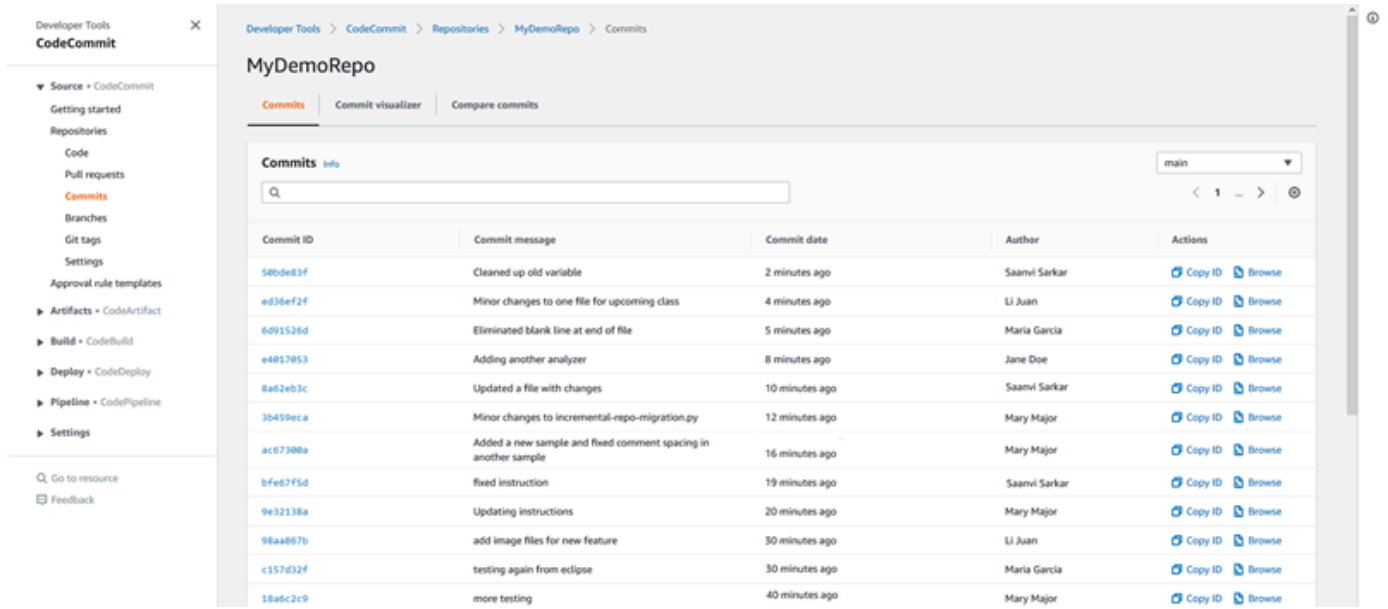
1. From **Repositories**, choose MyDemoRepo.
2. The page displays the contents in the default branch of your repository. To view another branch or to view the code at a specific tag, choose the branch or tag you want to view from the list. In the following screenshot, the view is set to the **main** branch.



3. To view the contents of a file in your repository, choose the file from the list. To change the color of the displayed code, choose the settings icon.

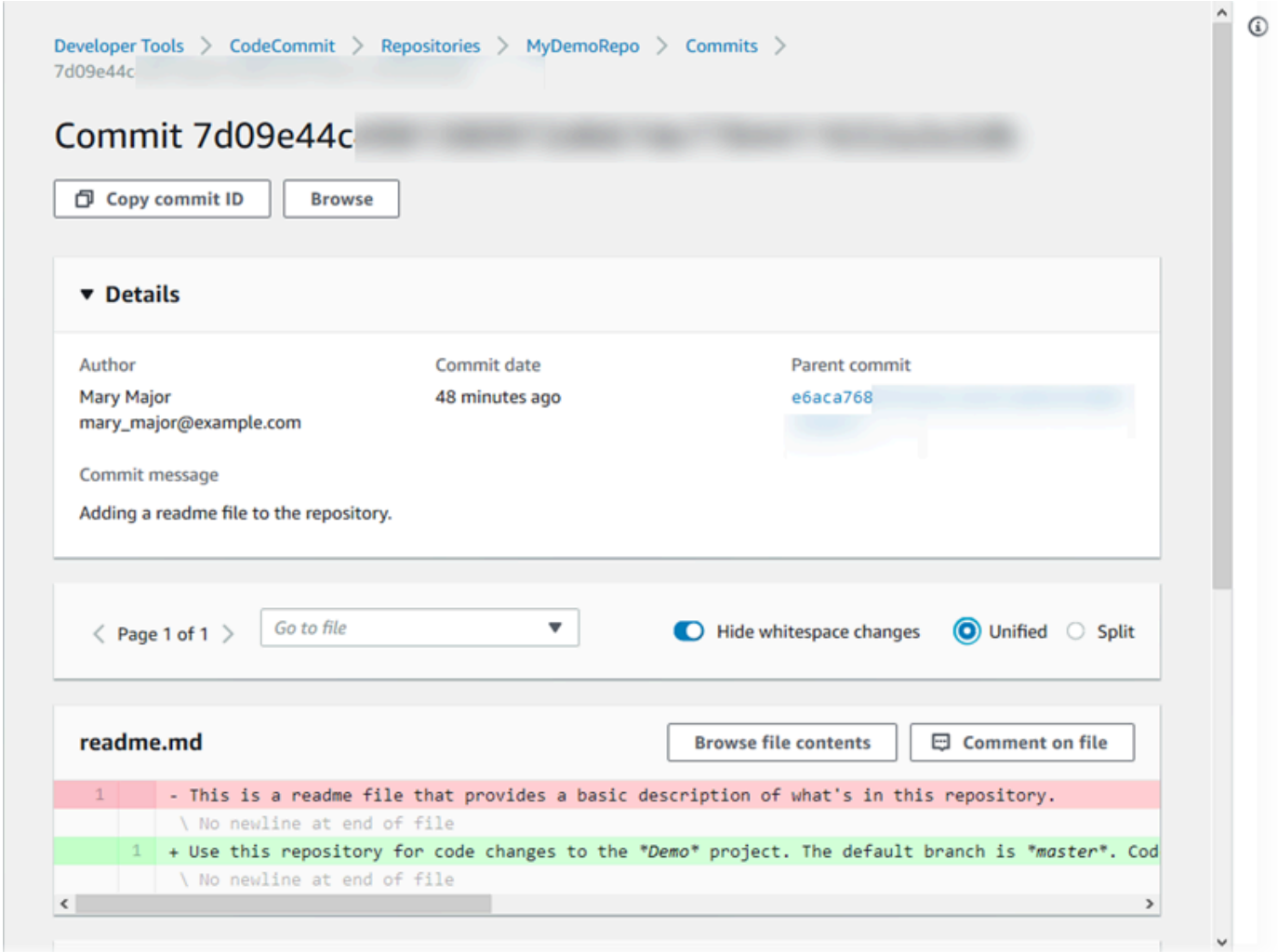
   For more information, see Browse files in a repository.

4. To browse the commit history of the repository, choose **Commits**. The console displays the commit history for the default branch, in reverse chronological order. Review the commit details by author, date, and more.



5. To view the commit history by branch or by Git tag, choose the branch or tag you want to view from the list.
6. To view the differences between a commit and its parent commit, choose the abbreviated commit ID. You can choose how the changes are displayed, including showing or hiding white space changes, and whether to view changes inline (**Unified** view) or side by side (**Split** view).

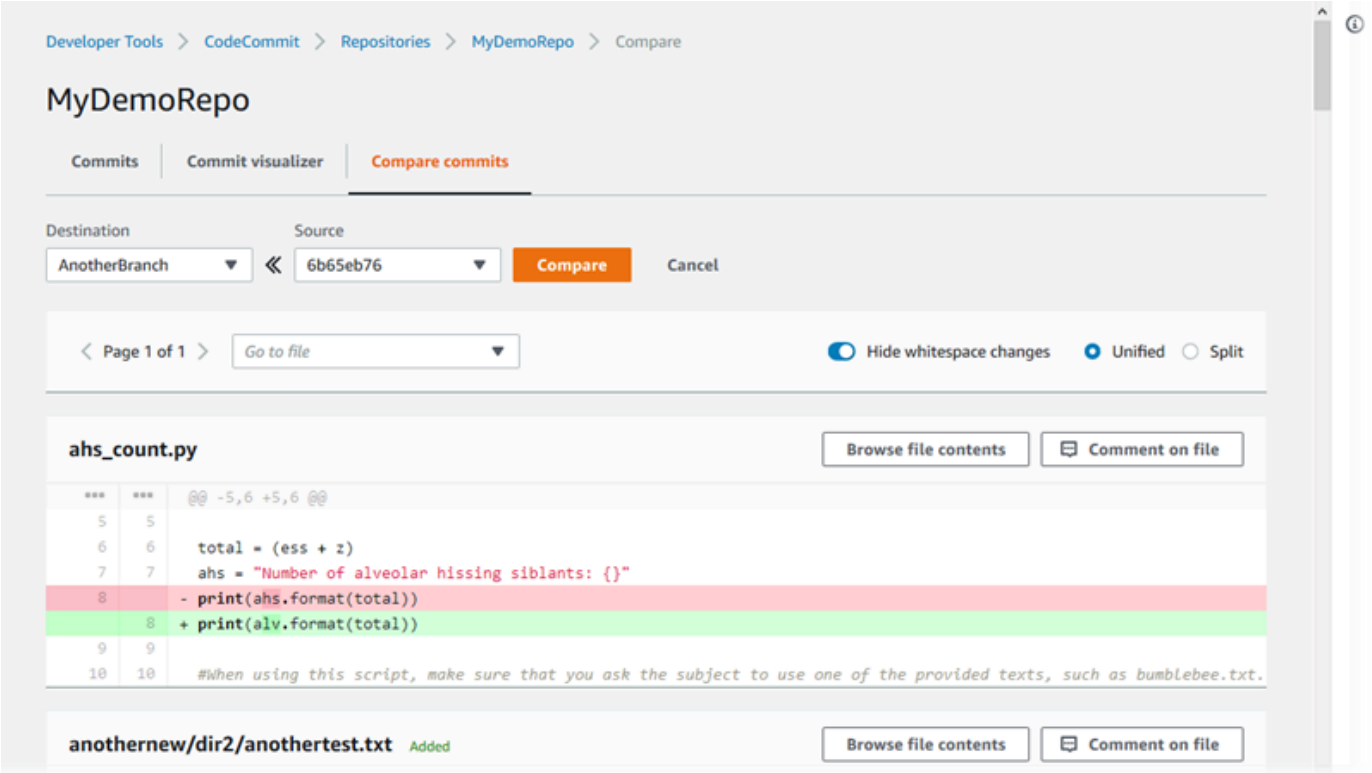
> **Note**
>
> Your preferences for viewing code and other console settings are saved as browser cookies whenever you change them. For more information, see Working with user preferences.



7. To view all comments on a commit, choose the commit and then scroll through the changes to view them inline. You can also add your own comments and reply to the comments made by others.

   For more information, see Comment on a commit.

8. To view the differences between any two commits specifiers, including tags, branches, and commit IDs, in the navigation pane, choose **Commits**, and then choose **Compare commits**.



For more information, see Browse the commit history of a repository and Compare commits.

9. In **Commits**, choose the **Commit visualizer** tab.

   The commit graph is displayed, with the subject line for each commit shown next to its point in the graph. The subject line display is limited to 80 characters.

10. To see more details about a commit, choose its abbreviated commit ID. To render the graph from a specific commit, choose that point in the graph. For more information, see View a graph of the commit history of a repository.

Step 4: Create and collaborate on a pull request

When you work in a repository with other users, you can collaborate on code and review changes. You can create a pull request so that other users can review and comment on your code changes in a branch. You can also create one or more approval rules for the pull request. For example, you can create an approval rule that requires at least two other users to approve the pull request before it can be merged. After the pull request is approved, you can merge those changes into its destination branch. If you set up notifications for your repository, repository users can receive emails about repository events (for example, for pull requests or when someone comments on

code). For more information, see Configuring notifications for events in an AWS CodeCommit repository.

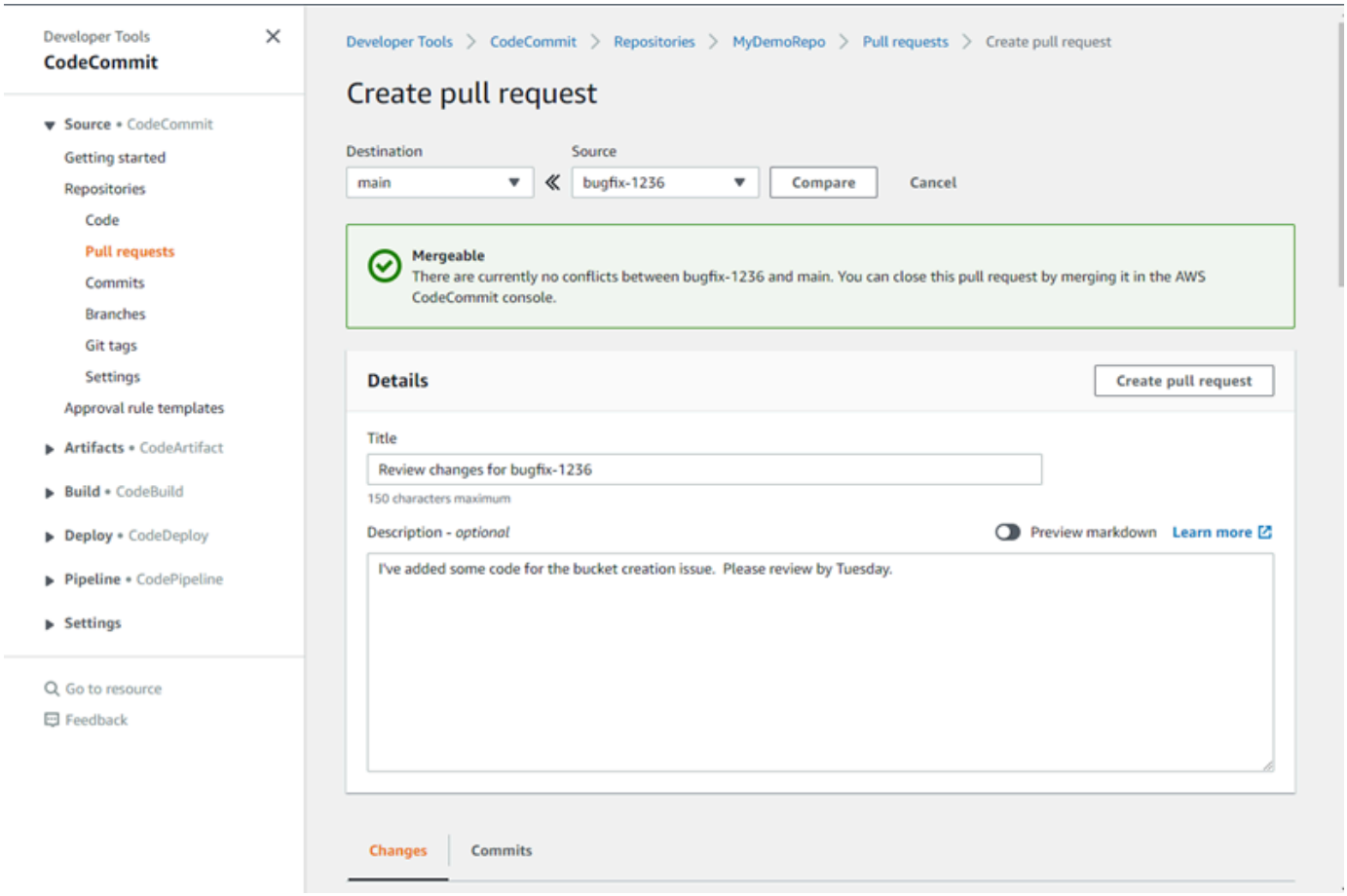
> **Important**
>
> Before you can create a pull request, you must create a branch that contains the code changes you want to review. For more information, see Create a branch.

To create and collaborate on a pull request

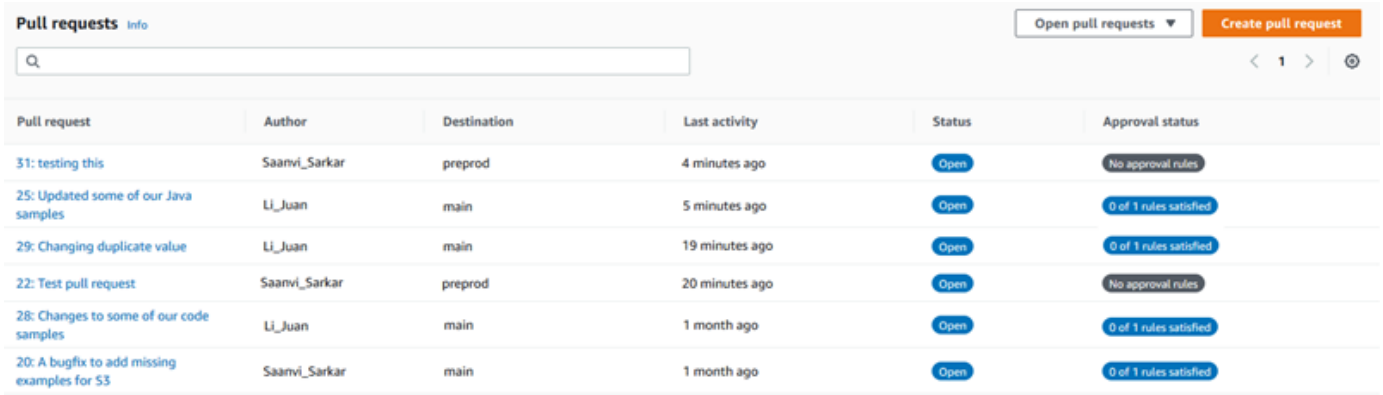
1. In the navigation pane, choose **Pull requests**.
2. In **Pull request**, choose **Create pull request**.

   > **Tip**
   >
   > You can also create pull requests from **Branches** and **Code**.

   In **Create pull request**, in **Source**, choose the branch that contains the changes you want reviewed. In **Destination**, choose the branch where you want the reviewed code to be merged when the pull request is closed. Choose **Compare**.
3. Review the merge details and changes to confirm that the pull request contains the changes and commits you want reviewed. If so, in **Title**, enter a title for this review. This is the title that appears in the list of pull requests for the repository. In **Description**, enter details about what this review is about and any other useful information for reviewers. Choose **Create**.

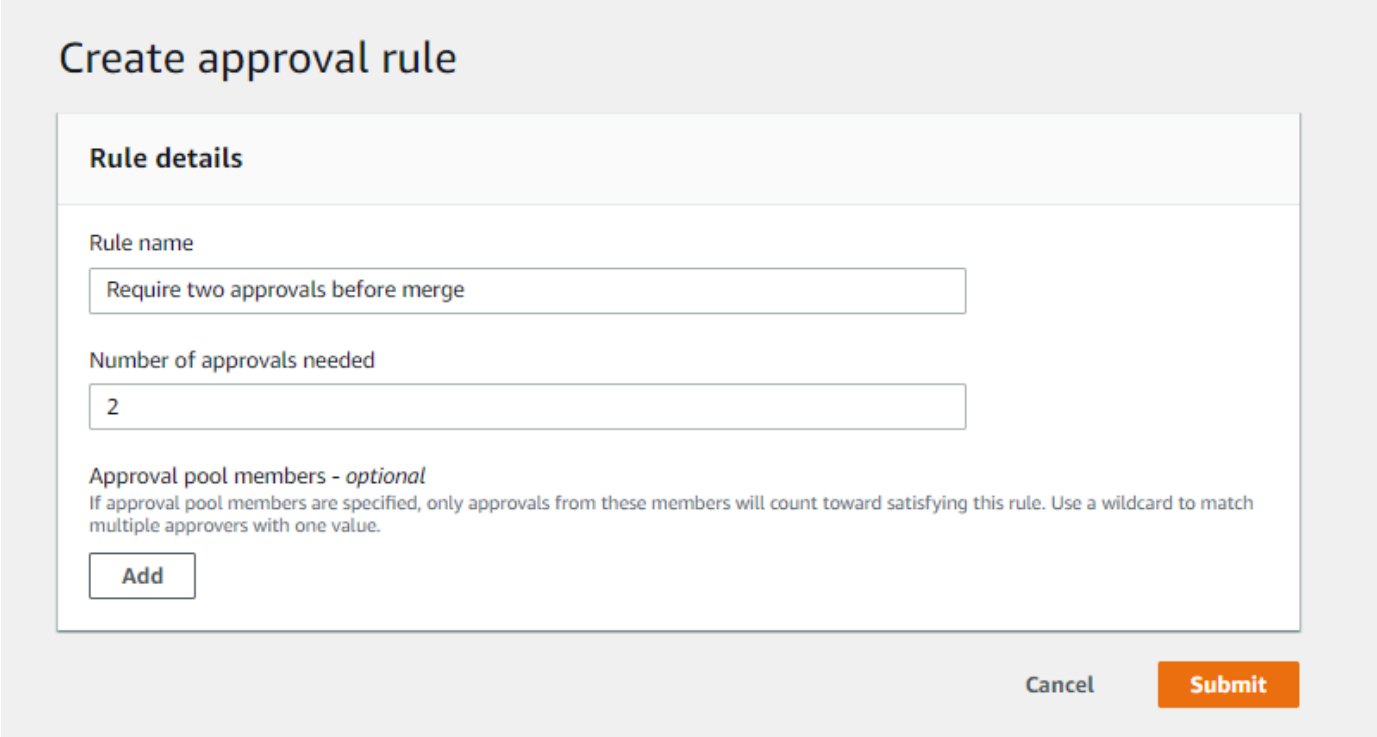


4. Your pull request appears in the list of pull requests for the repository. You can filter the view to show only open requests, closed requests, requests that you created, and more.

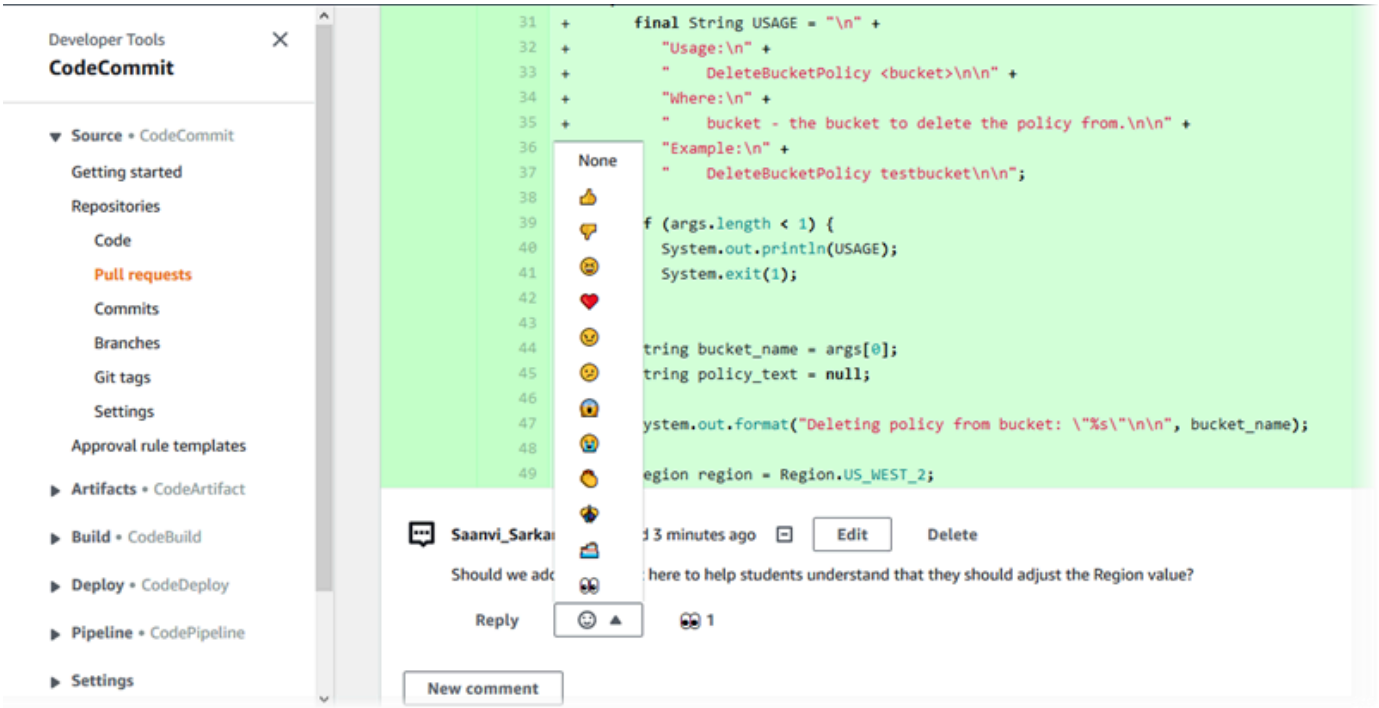


5. You can add an approval rule to your pull request to ensure that certain conditions are met before it can be merged. To add an approval rule to your pull request, choose the pull request from the list. On the **Approvals** tab, choose **Create approval rule**.
6. In **Rule name**, give the rule a descriptive name. For example, if you want to require two people to approve a pull request before it can be merged, you might name the rule **`Require two approvals before merge`**. In **Number of approvals needed**, enter **2**, the number you

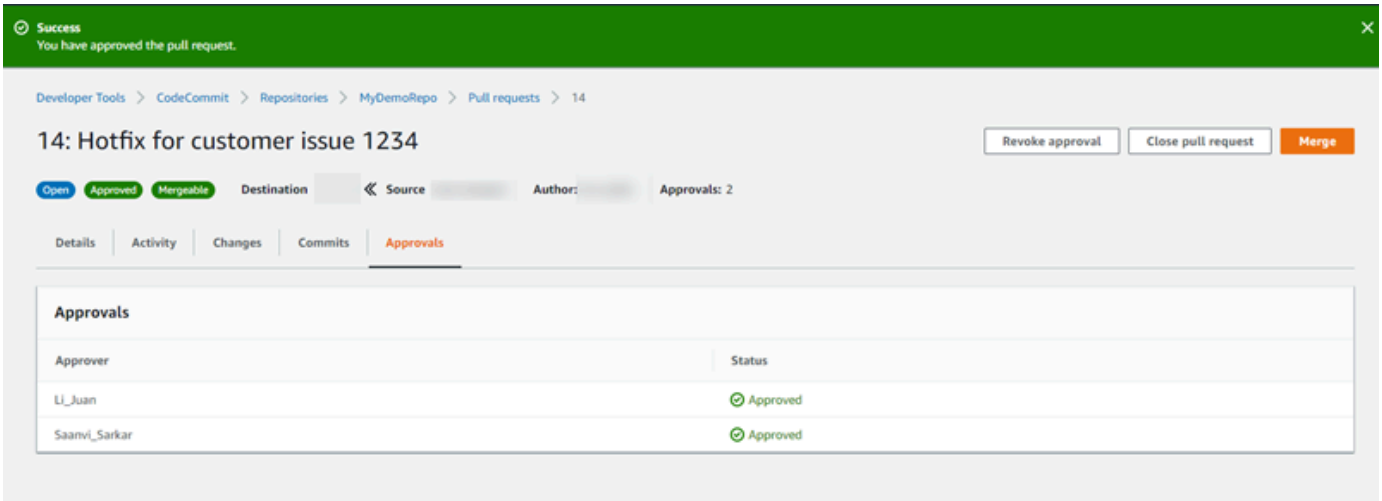
want. The default is 1. Choose **Submit**. To learn more about approval rules and approval pool members, see Create an approval rule for a pull request.



7. If you configured notifications for your repository and chose to notify users of pull request events, users receive email about your new pull request. Users can view the changes and comment on specific lines of code, files, and the pull request itself. They can also reply to comments with text and emojis. If necessary, you can push changes to the pull request branch, which updates the pull request.



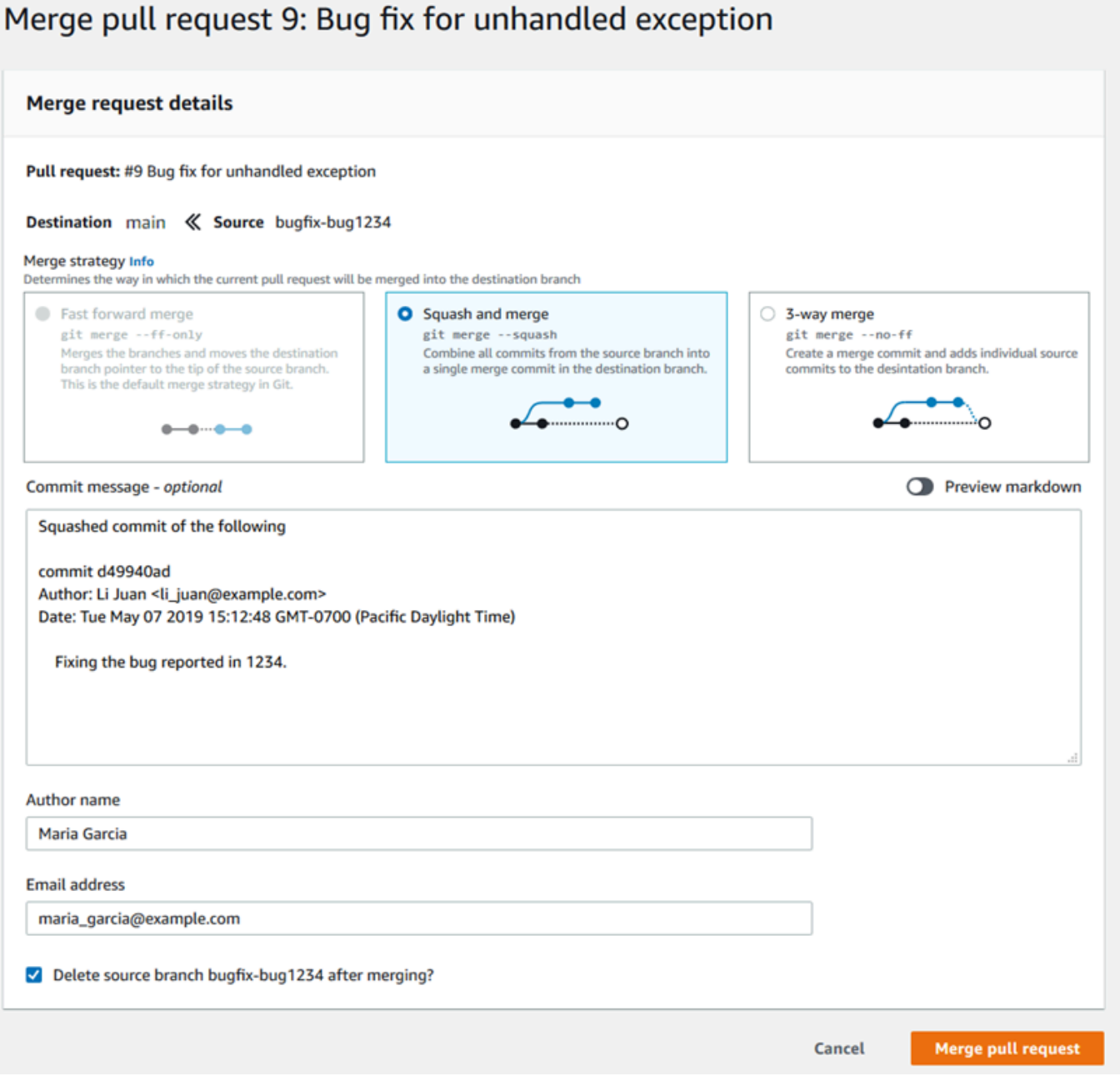
8. If you are satisfied about the changes made in the request, choose **Approve**. You can choose to approve a pull request even if no approval rules are configured for that pull request. This provides a clear record of your having reviewed the pull request and your approval of the changes. You can also choose to revoke your approval if you change your mind.



> **Note**
>
> You cannot approve a pull request if you created it.

9. When you are satisfied that all the code changes have been reviewed and agreed to, from the pull request, do one of the following:

- If you want to close the pull request without merging branches, choose **Close pull request**.
- If you want to merge the branches and close the pull request, choose **Merge**. You can choose between the merge strategies available for your code, which depend on the differences between the source and destination branches, and whether to automatically delete the source branch after the merge is complete. After you have made your choices, choose **Merge pull request** to complete the merge.



- If there are merge conflicts in the branches that cannot be resolved automatically, you can resolve them in the CodeCommit console, or you can use your local Git client to merge the

branches and then push the merge. For more information, see Resolve conflicts in a pull request in an AWS CodeCommit repository.

> **Note**
>
> You can always manually merge branches, including pull request branches, by using the **git merge** command in your local repo and pushing your changes.

For more information, see Working with pull requests and Working with approval rule templates.

Step 5: Clean up

If you no longer need the CodeCommit repository, you should delete the CodeCommit repository and other resources you used in this exercise so you won't continue to be charged for the storage space.

> **Important**
>
> This action cannot be undone. After you delete this repository, you can no longer clone it to any local repo or shared repo. You also can no longer pull data from or push data to it, or perform any Git operations, from any local repo or shared repo.
> If you configured notifications for your repository, deleting the repository also deletes the Amazon CloudWatch Events rule created for the repository. It does not delete the Amazon SNS topic used as a target for that rule.
> If you configured triggers for your repository, deleting the repository does not delete the Amazon SNS topics or Lambda functions you configured as the targets of those triggers. Be sure to delete those resources if you don't need them. For more information, see Delete triggers from a repository.

To delete the CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository you want to delete. If you followed the naming convention in this topic, it is named **MyDemoRepo**.

3. In the navigation pane, choose **Settings**.
4. On the **Settings** page, in **Delete repository**, choose **Delete repository**.
5. Type **`delete`**, and then choose **Delete**. The repository is permanently deleted.

Step 6: Next steps

Now that you have familiarized yourself with CodeCommit and some of its features, consider doing the following:

- If you are new to Git and CodeCommit or want to review examples of using Git with CodeCommit, continue to the Getting started with Git and CodeCommit tutorial.
- If you want to work with others in a CodeCommit repository, see Share a repository. (If you want to share your repository with users in another Amazon Web Services account, see Configure cross-account access to an AWS CodeCommit repository using roles.)
- If you want to migrate a repository to CodeCommit, follow the steps in Migrate to CodeCommit.
- If you want to add your repository to a continuous delivery pipeline, follow the steps in Simple Pipeline Walkthrough.
- If you want to learn more about products and services that integrate with CodeCommit, including examples from the community, see Product and service integrations.

Getting started with Git and AWS CodeCommit

If you are new to Git and CodeCommit, this tutorial helps you learn some simple commands to get you started. If you are already familiar with Git, you can skip this tutorial and go to Getting started with CodeCommit .

In this tutorial, you create a repository that represents a local copy of the CodeCommit repository, which we refer to as a *local repo*.

After you create the local repo, you make some changes to it. Then you send (push) your changes to the CodeCommit repository.

You also simulate a team environment where two users independently commit changes to their local repo and push those changes to the CodeCommit repository. The users then pull the changes from the CodeCommit repository to their own local repo to see the changes the other user made.

You also create branches and tags and manage some access permissions in the CodeCommit repository.

After you complete this tutorial, you should have enough practice with the core Git and CodeCommit concepts to use them for your own projects.

Complete the prerequisites and setup, including:

- Assign permissions to the IAM user.
- Set up CodeCommit to connect to a repository using HTTPS, SSH, or **git-remote-codecommit**. For more information about these choices, see Setting up for AWS CodeCommit.
- Configure the AWS CLI if you want to use the command line or terminal for all operations, including creating the repository.

Topics

- Step 1: Create a CodeCommit repository
- Step 2: Create a local repo
- Step 3: Create your first commit
- Step 4: Push your first commit
- Step 5: Share the CodeCommit repository and push and pull another commit
- Step 6: Create and share a branch
- Step 7: Create and share a tag
- Step 8: Set up access permissions
- Step 9: Clean up

Step 1: Create a CodeCommit repository

In this step, you use the CodeCommit console to create the repository.

You can skip this step if you already have a CodeCommit repository you want to use.

> **Note**
>
> Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the CodeCommit product information page.

To create the CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Use the Region selector to choose the AWS Region where you want to create the repository. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose **Create repository**.
4. On the **Create repository** page, in **Repository name**, enter a name for your repository (for example, **`MyDemoRepo`**).

   > **Note**
   >
   > Repository names are case sensitive and can be no longer than 100 characters. For more information, see Limits.

5. (Optional) In **Description**, enter a description (for example, **`My demonstration repository`**). This can help you and other users identify the purpose of the repository.
6. (Optional) Choose **Add tag** to add one or more repository tags (a custom attribute label that helps you organize and manage your AWS resources) to your repository. For more information, see Tagging repositories in AWS CodeCommit.
7. (Optional) Expand **Additional configuration** to specify whether to use the default AWS managed key or your own customer managed key for encrypting and decrypting the data in this repository. If you choose to use your own customer managed key, you must ensure that it is available in the AWS Region where you are creating the repository, and that the key is active. For more information, see AWS Key Management Service and encryption for AWS CodeCommit repositories.
8. (Optional) Select **Enable Amazon CodeGuru Reviewer for Java and Python** if this repository will contain Java or Python code, and you want to have CodeGuru Reviewer analyze that code. CodeGuru Reviewer uses multiple machine learning models to find code defects and to automatically suggest improvements and fixes in pull requests. For more information, see the Amazon CodeGuru Reviewer User Guide.
9. Choose **Create**.

> **Note**
>
> The remaining steps in this tutorial use `MyDemoRepo` for the name of the CodeCommit repository. If you choose a different name, be sure to use it throughout this tutorial.

For more information about creating repositories, including how to create a repository from the terminal or command line, see Create a repository.

Step 2: Create a local repo

In this step, you set up a local repo on your local machine to connect to your repository. To do this, you select a directory on your local machine that represents the local repo. You use Git to clone and initialize a copy of your empty CodeCommit repository inside of that directory. Then you specify the Git user name and email address used to annotate your commits.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. Find the repository you want to connect to from the list and choose it. Choose **Clone URL**, and then choose the protocol you want to use when cloning or connecting to the repository. This copies the clone URL.

   - Copy the HTTPS URL if you are using either Git credentials with your IAM user or the credential helper included with the AWS CLI.
   - Copy the HTTPS (GRC) URL if you are using the **git-remote-codecommit** command on your local computer.
   - Copy the SSH URL if you are using an SSH public/private key pair with your IAM user.

   > **Note**
   >
   > If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account in the AWS Region where you are signed in. To

create a repository, see the section called "Create a repository" or follow the steps in the Getting started with Git and CodeCommit tutorial.

4. (Optional) We recommend that you configure your local Git client to use **`main`** as the name for the default branch for your repository. This is the name used for the default branch in all examples in this guide. It is also the same default branch name CodeCommit uses if you make your first commit in the console. Run the following command to configure the default branch name globally for your system:

```
git config --global init.defaultBranch main
```

   If you prefer to use a different default branch name for all your repositories, replace **`main`** with your preferred name. This tutorial assumes that your default branch is named *main*.

   If you want to use different default branch names for different repositories, you can set this attribute locally (**--local**) instead of globally (**--global**).

5. At the terminal or command prompt, clone the repository with the **git clone** command and provide the clone URL you copied in step 3. Your clone URL depends on which protocol and configuration you use. For example, if you are using HTTPS with Git credentials to clone a repository named *MyDemoRepo* in the US East (Ohio) Region:

```
git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-
demo-repo
```

   If you are using HTTPS with **git-remote-codecommit**:

```
git clone codecommit://MyDemoRepo my-demo-repo
```

   If you are using SSH:

```
git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-
repo
```

> **Note**
>
> If you see an error when attempting to clone a repository, you might not have completed the setup necessary for your local computer. For more information, see Setting up for AWS CodeCommit.

Step 3: Create your first commit

In this step, you create your first commit in your local repo. To do this, you create two example files in your local repo. You use Git to stage the change to, and then commit the change to, your local repo.

1. Use a text editor to create the following two example text files in your directory. Name the files `cat.txt` and `dog.txt`:

```
cat.txt
-------
The domestic cat (Felis catus or Felis silvestris catus) is a small, usually furry,
 domesticated, and carnivorous mammal.
```

```
dog.txt
-------
The domestic dog (Canis lupus familiaris) is a canid that is known as man's best
 friend.
```

2. Run **git config** to add your user name and email address represented by placeholders *`your-user-name`* and *`your-email-address`* to your local repo. This makes it easier to identify the commits you make:

```
git config --local user.name "your-user-name"
git config --local user.email your-email-address
```

3. If you did not set your default branch name globally when you created the local repo, run the following command to set the default branch name to **`main`**:

```
git config --local init.defaultBranch main
```

4. Run **git add** to stage the change:

```
git add cat.txt dog.txt
```

5. Run **git commit** to commit the change:

```
git commit -m "Added cat.txt and dog.txt"
```

> **Tip**
>
> To see details about the commit you just made, run **git log**.

Step 4: Push your first commit

In this step, you push the commit from your local repo to your CodeCommit repository.

Run **git push** to push your commit through the default remote name Git uses for your CodeCommit repository (`origin`), from the default branch in your local repo (`main`):

```
git push -u origin main
```

> **Tip**
>
> After you have pushed files to your CodeCommit repository, you can use the CodeCommit console to view the contents. For more information, see Browse files in a repository.

Step 5: Share the CodeCommit repository and push and pull another commit

In this step, you share information about the CodeCommit repository with a fellow team member. The team member uses this information to get a local copy, make some changes to it, and then push the modified local copy to your CodeCommit repository. You then pull the changes from the CodeCommit repository to your local repo.

In this tutorial, you simulate the fellow user by having Git create a directory separate from the one you created in step 2. (Typically, this directory is on a different machine.) This new directory is a

copy of your CodeCommit repository. Any changes you make to the existing directory or this new directory are made independently. The only way to identify changes to these directories is to pull from the CodeCommit repository.

Even though they're on the same local machine, we call the existing directory your *local repo* and the new directory the *shared repo*.

From the new directory, you get a separate copy of the CodeCommit repository. You then add a new example file, commit the changes to the shared repo, and then push the commit from the shared repo to your CodeCommit repository.

Lastly, you pull the changes from your repository to your local repo and then browse it to see the changes committed by the other user.

1. Switch to the `/tmp` directory or the `c:\temp` directory.
2. Run **git clone** to pull down a copy of the repository into the shared repo:

   For HTTPS:

   ```
   git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
    shared-demo-repo
   ```

   For HTTPS with **git-remote-codecommit**:

   ```
   git clone codecommit://MyDemoRepo shared-demo-repo
   ```

   For SSH:

   ```
   git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo shared-
   demo-repo
   ```

   Note

   When you clone a repository using SSH on Windows operating systems, you might need to add the SSH key ID to the connection string as follows:

   ```
   git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/
   repos/MyDemoRepo my-demo-repo
   ```

For more information, see For SSH connections on Windows.

In this command, `MyDemoRepo` is the name of your CodeCommit repository. `shared-demo-repo` is the name of the directory Git creates in the `/tmp` directory or the `c:\temp` directory. After Git creates the directory, Git pulls down a copy of your repository into the `shared-demo-repo` directory.

3. Switch to the `shared-demo-repo` directory:

```
(For Linux, macOS, or Unix) cd /tmp/shared-demo-repo
(For Windows) cd c:\temp\shared-demo-repo
```

4. Run **git config** to add another user name and email address represented by placeholders *other-user-name* and *other-email-address*. This makes it easier to identify the commits the other user makes:

```
git config --local user.name "other-user-name"
git config --local user.email other-email-address
```

5. Use a text editor to create the following example text file in the `shared-demo-repo` directory. Name the file `horse.txt`:

```
horse.txt
-------
The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus.
```

6. Run **git add** to stage the change to the shared repo:

```
git add horse.txt
```

7. Run **git commit** to commit the change to the shared repo:

```
git commit -m "Added horse.txt"
```

8. Run **git push** to push your initial commit through the default remote name Git uses for your CodeCommit repository (`origin`), from the default branch in your local repo (`main`):

```
git push -u origin main
```

9. Switch to your local repo and run **git pull** to pull into your local repo the commit the shared repo made to the CodeCommit repository. Then run **git log** to see the commit that was initiated from the shared repo.

Step 6: Create and share a branch

In this step, you create a branch in your local repo, make a few changes, and then push the branch to your CodeCommit repository. You then pull the branch to the shared repo from your CodeCommit repository.

A *branch* allows you to independently develop a different version of the repository's contents (for example, to work on a new software feature without affecting the work of your team members). When that feature is stable, you merge the branch into a more stable branch of the software.

You use Git to create the branch and then point it to the first commit you made. You use Git to push the branch to the CodeCommit repository. You then switch to your shared repo and use Git to pull the new branch into your shared local repo and explore the branch.

1. From your local repo, run **git checkout**, specifying the name of the branch (for example, `MyNewBranch`) and the ID of the first commit you made in the local repo.

   If you don't know the commit ID, run **git log** to get it. Make sure the commit has your user name and email address, not the user name and email address of the other user. This is to simulate that `main` is a stable version of the CodeCommit repository and the `MyNewBranch` branch is for some new, relatively unstable feature:

   ```
   git checkout -b MyNewBranch commit-ID
   ```

2. Run **git push** to send the new branch from the local repo to the CodeCommit repository:

   ```
   git push origin MyNewBranch
   ```

3. Now, pull the branch into the shared repo and check your results:

   1. Switch to the shared repo directory (shared-demo-repo).
   2. Pull in the new branch (**git fetch origin**).
   3. Confirm that the branch has been pulled in (**git branch --all** displays a list of all branches for the repository).

4. Switch to the new branch (**git checkout MyNewBranch**).
5. Confirm that you have switched to the `MyNewBranch` branch by running **git status** or **git branch**. The output shows which branch you are on. In this case, it should be `MyNewBranch`.
6. View the list of commits in the branch (**git log**).

   Here's the list of Git commands to call:

   ```
   git fetch origin
   git branch --all
   git checkout MyNewBranch
   git branch or git status
   git log
   ```

4. Switch back to the `main` branch and view its list of commits. The Git commands should look like this:

   ```
   git checkout main
   git log
   ```

5. Switch to the `main` branch in your local repo. You can run **git status** or **git branch**. The output shows which branch you are on. In this case, it should be `main`. The Git commands should look like this:

   ```
   git checkout main
   git branch or git status
   ```

Step 7: Create and share a tag

In this step, you create two tags in your local repo, associate the tags with commits, and then push the tags to your CodeCommit repository. You then pull the changes from the CodeCommit repository to the shared repo.

A *tag* is used to give a human-readable name to a commit (or branch or even another tag). You would do this, for example, if you want to tag a commit as `v2.1`. A commit, branch, or tag can have any number of tags associated with it, but an individual tag can be associated with only one commit, branch, or tag. In this tutorial, you tag one commit as `release` and one as `beta`.

You use Git to create the tags, pointing the `release` tag to the first commit you made and the `beta` tag to the commit made by the other user. You then use Git to push the tags to the CodeCommit repository. Then you switch to your shared repo and use Git to pull the tags into your shared local repo and explore the tags.

1. From your local repo, run **git tag**, specifying the name of the new tag (`release`) and the ID of the first commit you made in the local repo.

   If you don't know the commit ID, run **git log** to get it. Make sure the commit has your user name and email address, not the user name and email address of the other user. This is to simulate that your commit is a stable version of the CodeCommit repository:

   ```
   git tag release commit-ID
   ```

   Run **git tag** again to tag the commit from the other user with the `beta` tag. This is to simulate that the commit is for some new, relatively unstable feature:

   ```
   git tag beta commit-ID
   ```

2. Run **git push --tags** to send the tags to the CodeCommit repository.
3. Now pull the tags into the shared repo and check your results:

   1. Switch to the shared repo directory (shared-demo-repo).
   2. Pull in the new tags (**git fetch origin**).
   3. Confirm that the tags have been pulled in (**git tag** displays a list of tags for the repository).
   4. View information about each tag (**git log release** and **git log beta**).

   Here's the list of Git commands to call:

   ```
   git fetch origin
   git tag
   git log release
   git log beta
   ```

4. Try this out in the local repo, too:

   ```
   git log release
   ```

```
git log beta
```

Step 8: Set up access permissions

In this step, you give a user permission to synchronize the shared repo with the CodeCommit repository. This is an optional step. It's recommended for users who are interested in learning about how to control access to CodeCommit repositories when users use Git credentials or SSH key pairs are used with IAM users for access to CodeCommit repositories.

> **Note**
>
> If you are using federated access, temporary credentials, or a web identity provider such as IAM Identity Center, set up users, access, and permissions for your identity provider, and then use **git-remote-codecommit**. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit and Connecting to AWS CodeCommit repositories with rotating credentials.

To do this, you use the IAM console to create a user, who, by default, does not have permissions to synchronize the shared repo with the CodeCommit repository. You can run **git pull** to verify this. If the new user doesn't have permission to synchronize, the command doesn't work. Then you go back to the IAM console and apply a policy that allows the user to use **git pull**. Again, you can run **git pull** to verify this.

This step is written with the assumption you have permissions to create IAM users in your Amazon Web Services account. If you don't have these permissions, you can't perform the procedures in this step. Skip ahead to Step 9: Clean up to clean up the resources you used for your tutorial.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

   Be sure to sign in with the same user name and password you used in Setting up .
2. In the navigation pane, choose **Users**, and then choose **Create New Users**.
3. In the first **Enter User Names** box, enter an example user name (for example, **`JaneDoe-CodeCommit`**). Select the **Generate an access key for each user** box, and then choose **Create**.
4. Choose **Show User Security Credentials**. Make a note of the access key ID and secret access key or choose **Download Credentials**.

5. Follow the instructions in For HTTPS users using Git credentials to generate and supply the credentials of the IAM user.

   If you want to use SSH, follow the instructions in SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit or Step 3: Set up the public and private keys for Git and CodeCommit to set up the user with public and private keys.

6. Run **git pull**. The following error should appear:

   For HTTPS:

   ```
   fatal: unable to access 'https://git-codecommit.us-
   east-2.amazonaws.com/v1/repos/repository-name/': The requested URL
   returned error: 403.
   ```

   For SSH:

   ```
   fatal: unable to access 'ssh://git-codecommit.us-east-2.amazonaws.com/
   v1/repos/repository-name/': The requested URL returned error: 403.
   ```

   The error appears because the new user doesn't have permission to synchronize the shared repo with the CodeCommit repository.

7. Return to the IAM console. In the navigation pane, choose **Policies**, and then choose **Create Policy**. (If a **Get Started** button appears, choose it, and then choose **Create Policy**.)
8. Next to **Create Your Own Policy**, choose **Select**.
9. In the **Policy Name** box, enter a name (for example, **CodeCommitAccess-GettingStarted**).
10. In the **Policy Document** box, enter the following, which allows an IAM user to pull from any repository associated with the IAM user:

    JSON

    ```
    {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "codecommit:GitPull"
          ],
    ```

```
      "Resource": "*"
    }
  ]
}
```

Tip

If you want the IAM user to be able to push commits to any repository associated with the IAM user, enter this instead:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull",
        "codecommit:GitPush"
      ],
      "Resource": "*"
    }
  ]
}
```

For information about other CodeCommit action and resource permissions you can give to users, see Authentication and access control for AWS CodeCommit.

11. In the navigation pane, choose **Users**.
12. Choose the example user name (for example, **`JaneDoe-CodeCommit`**) to which you want to attach the policy.
13. Choose the **Permissions** tab.
14. In **Managed Policies**, choose **Attach Policy**.
15. Select the **`CodeCommitAccess-GettingStarted`** policy you just created, and then choose **Attach Policy**.

16. Run **git pull**. This time the command should work and an `Already up-to-date` message should appear.
17. If you are using HTTPS, switch to your original Git credentials or, if using **git-remote-codecommit**, your usual profile. For more information, see the instructions in Setup for HTTPS users using Git credentials or Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

    If you are using SSH, switch to your original keys. For more information, see SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit or Step 3: Set up the public and private keys for Git and CodeCommit .

You've reached the end of this tutorial.

Step 9: Clean up

In this step, you delete the CodeCommit repository you used in this tutorial, so you won't continue to be charged for the storage space.

You also remove the local repo and shared repo on your local machine because they won't be needed after you delete the CodeCommit repository.

> **Important**
>
> After you delete this repository, you won't be able to clone it to any local repo or shared repo. You also won't be able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

To delete the CodeCommit repository (console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. On the **Dashboard** page, in the list of repositories, choose **MyDemoRepo**.
3. In the navigation pane, choose **Settings**.
4. On the **Settings** page, in **Delete repository**, choose **Delete repository**.
5. In the box next to **Type the name of the repository to confirm deletion**, enter **MyDemoRepo**, and then choose **Delete**.

To delete the CodeCommit repository (AWS CLI)

Run the delete-repository command:

```
aws codecommit delete-repository --repository-name MyDemoRepo
```

To delete the local repo and shared repo

For Linux, macOS, or Unix:

```
cd /tmp
rm -rf /tmp/my-demo-repo
rm -rf /tmp/shared-demo-repo
```

For Windows:

```
cd c:\temp
rd /s /q c:\temp\my-demo-repo
rd /s /q c:\temp\shared-demo-repo
```

Product and service integrations with AWS CodeCommit

By default, CodeCommit is integrated with a number of AWS services. You can also use CodeCommit with products and services outside of AWS. The following information can help you configure CodeCommit to integrate with the products and services you use.

Topics

- Integration with other AWS services
- Integration examples from the community

Integration with other AWS services

CodeCommit is integrated with the following AWS services:

AWS Amplify	AWS Amplify makes it easy to create, configure, and implement scalable mobile applications powered by AWS. Amplify seamlessly provisions and manages your mobile backend and provides a simple framework to easily integrate your backend with your iOS, Android, Web, and React Native frontends. Amplify also automates the application release process of both your frontend and backend, which makes it possible for you to deliver features faster. You can connect your CodeCommit repository in the Amplify console. After you authorize the Amplify console, Amplify fetches an access token from the repository provider, but it doesn't store the token on the AWS servers. Amplify accesses your repository using deploy keys installed in a specific repository only. Learn more:

	• AWS Amplify User Guide • Getting Started
AWS Cloud9	AWS Cloud9 contains a collection of tools that you use to code, build, run, test, debug, and release software in the cloud. This collectio n of tools is referred to as the AWS Cloud9 integrated development environment, or IDE. You access the AWS Cloud9 IDE through a web browser. The IDE offers a rich code-editing experience with support for several programming languages and runtime debuggers, and a built-in terminal. Learn more: • AWS Cloud9 User Guide • AWS CodeCommit Sample for AWS Cloud9 • Integrate AWS Cloud9 with AWS CodeCommit
CloudFormation	CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications. You create a template that describes resources, including a CodeCommit repository, and CloudFormation takes care of provisioning and configuring those resources for you. Learn more: • AWS CodeCommit Repository resource page

AWS CloudTrail	CloudTrail captures AWS API calls and related events made by or on behalf of an Amazon Web Services account and delivers log files to an Amazon S3 bucket that you specify. You can configure CloudTrail to capture API calls from the AWS CodeCommit console, CodeCommit commands from the AWS CLI, the local Git client, and from the CodeCommit API. Learn more: • Logging AWS CodeCommit API calls with AWS CloudTrail

Amazon CloudWatch Events

CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. CloudWatch Events responds to these operational changes and takes action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information.

You can configure CloudWatch Events to monitor CodeCommit repositories and respond to repository events by targeting streams, functions, tasks, or other processes in other AWS services, such as Amazon Simple Queue Service, Amazon Kinesis, AWS Lambda, and many more.

Learn more:

- CloudWatch Events User Guide
- AWS CodeCommit Events
- Blog post: Build Serverless AWS CodeCommit Workflows using Amazon CloudWatch Events and JGit

AWS CodeBuild	CodeBuild is a fully managed build service in the cloud that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can store the source code to be built and the build specification in a CodeCommit repository. You can use CodeBuild directly with CodeCommit, or you can incorporate both CodeBuild and CodeCommit in a continuous delivery pipeline with CodePipeline. Learn more: • Plan a Build • Create a Build Project • Use CodePipeline with AWS CodeBuild to Run Builds
Amazon CodeGuru Reviewer	Amazon CodeGuru Reviewer is an automated code review service that uses program analysis and machine learning to detect common issues and recommend fixes in your Java or Python code. You can associate repositories in your Amazon Web Services account with CodeGuru Reviewer. When you do, CodeGuru Reviewer creates a service-linked role that allows CodeGuru Reviewer to analyze code in all pull requests created after the association is made. Learn more: • Amazon CodeGuru Reviewer and AWS CodeCommit • Amazon CodeGuru Reviewer User Guide

AWS CodePipeline	CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can configure CodePipeline to use a CodeCommit repository as a source action in a pipeline, and automate building, testing, and deploying your changes. Learn more: • Simple Pipeline Walkthrough with CodePipeline and AWS CodeCommit • Migrate to Amazon CloudWatch Events Change Detection for Pipelines with a CodeCommit Repository • Change-Detection Methods Used to Start Pipelines Automatically
AWS Elastic Beanstalk	Elastic Beanstalk is a managed service that makes it easy to deploy and manage applications in the AWS cloud without worrying about the infrastructure that runs those applications. You can use the Elastic Beanstalk command line interface (EB CLI) to deploy your application directly from a new or existing CodeCommit repository. Learn more: • Using the EB CLI with AWS CodeCommit • Using an Existing AWS CodeCommit Repository • eb codesource (EB CLI command)

AWS Key Management Service	AWS KMS is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. By default, CodeCommit uses AWS KMS to encrypt repositories. Learn more: • AWS KMS and encryption
AWS Lambda	Lambda lets you run code without provision ing or managing servers. You can configure triggers for CodeCommit repositories that invoke Lambda functions in response to repository events. Learn more: • Create a trigger for a Lambda function • AWS Lambda Developer Guide
Amazon Simple Notification Service	Amazon SNS is a web service that enables applications, end users, and devices to instantly send and receive notifications from the cloud. You can configure triggers for CodeCommit repositories that send Amazon SNS notifications in response to repository events. You can also use Amazon SNS notificat ions to integrate with other AWS services. For example, you can use an Amazon SNS notificat ion to send messages to an Amazon Simple Queue Service queue. Learn more: • Create a trigger for an Amazon SNS topic • Amazon Simple Notification Service Developer Guide

Integration examples from the community

The following sections provide links to blog posts, articles, and community-provided examples.

> **Note**
>
> These links are provided for informational purposes only, and should not be considered either a comprehensive list or an endorsement of the content of the examples. AWS is not responsible for the content or accuracy of external content.

Topics

- Blog posts
- Code samples

Blog posts

- **Integrating SonarQube as a Pull Request Approver on AWS CodeCommit**

  Learn how to create a CodeCommit repository that requires a successful SonarQube quality analysis before pull requests can be merged.

  Published December 12, 2019
- **Migration to AWS CodeCommit, AWS CodePipeline, and AWS CodeBuild From GitLab**

  Learn how to migrate multiple repositories to AWS CodeCommit from GitLab and set up a CI/CD pipeline using AWS CodePipeline and AWS CodeBuild.

  Published November 22, 2019
- **Implementing GitFlow Using AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, and AWS CodeDeploy**

  Learn how to implement GitFlow using AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, and AWS CodeDeploy.

  Published February 22, 2019
- **Using Git with AWS CodeCommit Across Multiple AWS Accounts**

Learn how to manage your Git configuration across multiple Amazon Web Services accounts.

Published February 12, 2019

- **Validating AWS CodeCommit Pull Requests with AWS CodeBuildand AWS Lambda**

  Learn how to validate pull requests with AWS CodeCommit, AWS CodeBuild, and AWS Lambda. By running tests against the proposed changes prior to merging them into the default branch, you can help ensure a high level of quality in pull requests, catch any potential issues, and boost the confidence of the developer in relation to their changes.

  Published February 11, 2019

- **Using Federated Identities with AWS CodeCommit**

  Learn how to access repositories in AWS CodeCommit using the identities used in your business.

  Published October 5, 2018

- **Refining Access to Branches in AWS CodeCommit**

  Learn how to restrict commits to repository branches by creating and applying an IAM policy that uses a context key.

  Published May 16, 2018

- **Replicate AWS CodeCommit Repositories Between Regions Using AWS Fargate**

  Learn how to set up continuous replication of a CodeCommit repository from one AWS region to another using a serverless architecture.

  Published April 11, 2018

- **Distributing Your AWS OpsWorks for Chef Automate Infrastructure**

  Learn how to use CodePipeline, CodeCommit, CodeBuild, and AWS Lambda to ensure that cookbooks and other configurations are consistently deployed across two or more Chef Servers residing in one or more AWS Regions.

  Published March 9, 2018

- **Peanut Butter and Chocolate: Azure Functions CI/CD Pipeline with AWS CodeCommit**

  Learn how to create a PowerShell-based Azure Functions CI/CD pipeline where the code is stored in a CodeCommit repository.

Published February 19, 2018

- **Continuous Deployment to Kubernetes Using AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, Amazon ECR, and AWS Lambda**

  Learn how to use Kubernetes and AWS together to create a fully managed, continuous deployment pipeline for container based applications.

  Published January 11, 2018

- **Use AWS CodeCommit Pull Requests to Request Code Reviews and Discuss Code**

  Learn how to use pull requests to review, comment upon, and interactively iterate on code changes in a CodeCommit repository.

  Published November 20, 2017

- **Build Serverless AWS CodeCommit Workflows Using Amazon CloudWatch Events and JGit**

  Learn how to create CloudWatch Events rules that process changes in a repository using CodeCommit repository events and target actions in other AWS services. Examples include AWS Lambda functions that enforce Git commit message policies on commits, replicate a CodeCommit repository, and backing up a CodeCommit repository to Amazon S3.

  Published August 3, 2017

- **Migrating to AWS CodeCommit**

  Learn how to push code to two repositories as part of migrating from using another Git repository to CodeCommit when using SourceTree.

  Published September 6, 2016

- **Set Up Continuous Testing with Appium, AWS CodeCommit, Jenkins, and AWS Device Farm**

  Learn how to create a continuous testing process for mobile devices using Appium, CodeCommit, Jenkins, and Device Farm.

  Published February 2, 2016

- **Using AWS CodeCommit with Git Repositories in Multiple Amazon Web Services accounts**

  Learn how to clone your CodeCommit repository and, in one command, configure the credential helper to use a specific IAM role for connections to that repository.

Published November 2015

- **Integrating OpsWorks and AWS CodeCommit**

  Learn how OpsWorks can automatically fetch Apps and Chef cookbooks from CodeCommit.

  Published August 25, 2015

- **Using AWS CodeCommit and GitHub Credential Helpers**

  Learn how to configure your gitconfig file to work with both CodeCommit and GitHub credential helpers.

  Published September 2015

- **Using AWS CodeCommit from Eclipse**

  Learn how to use the EGit tools in Eclipse to work with CodeCommit.

  Published August 2015

- **AWS CodeCommit with Amazon EC2 Role Credentials**

  Learn how to use an instance profile for Amazon EC2 when configuring automated agent access to a CodeCommit repository.

  Published July 2015

- **Integrating AWS CodeCommit with Jenkins**

  Learn how to use CodeCommit and Jenkins to support two simple continuous integration (CI) scenarios.

  Published July 2015

- **Integrating AWS CodeCommit with Review Board**

  Learn how to integrate CodeCommit into a development workflow using the Review Board code review system.

  Published July 2015

Code samples

The following are code samples that might be of interest to CodeCommit users.

- **Mac OS X Script to Periodically Delete Cached Credentials in the OS X Certificate Store**

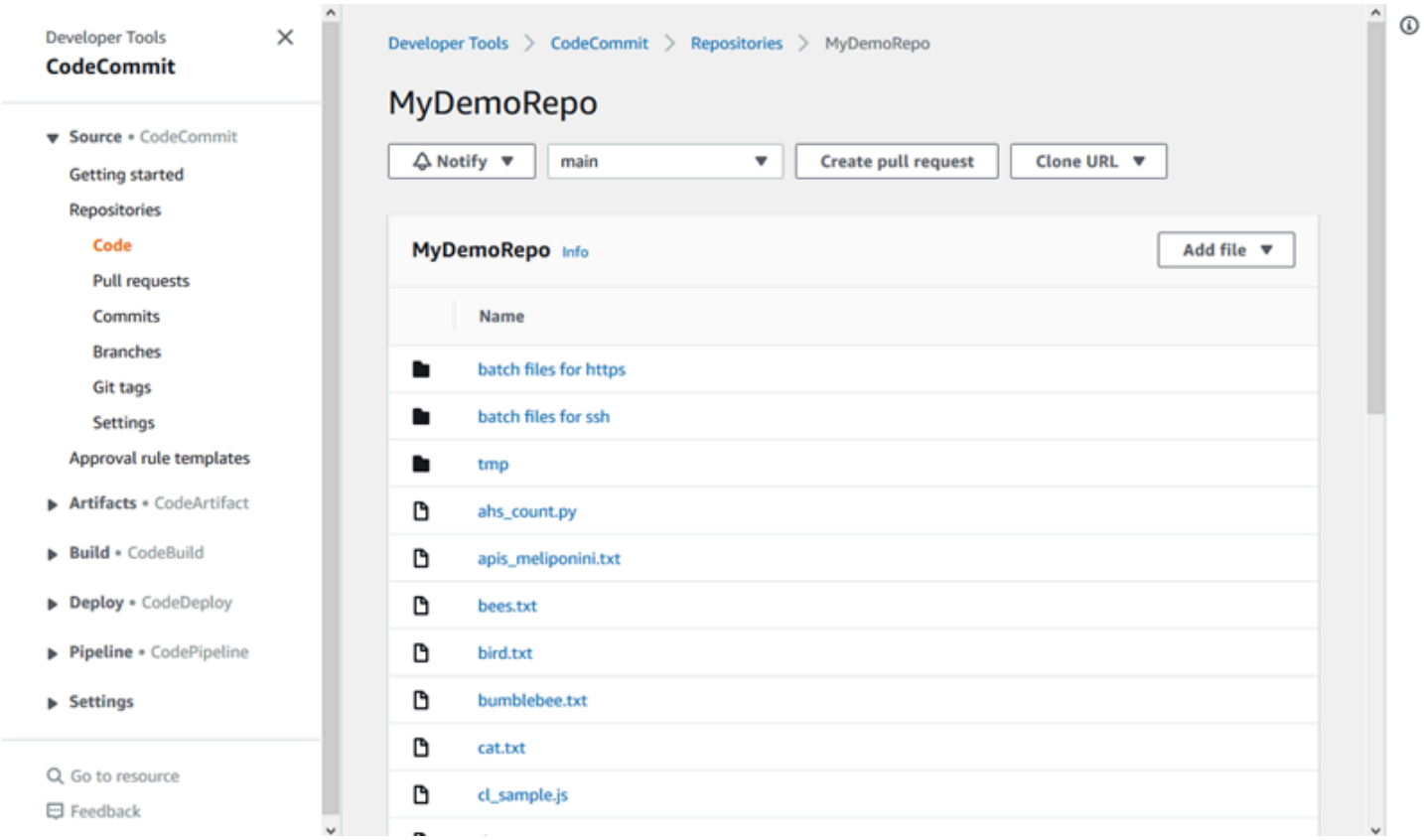
  If you use the credential helper for CodeCommit on Mac OS X, you are likely familiar with the problem with cached credentials. This script demonstrate one solution.

  Author: Nico Coetzee

  Published February 2016

Working with repositories in AWS CodeCommit

A repository is the fundamental version control object in CodeCommit. It's where you securely store code and files for your project. It also stores your project history, from the first commit through the latest changes. You can share your repository with other users so you can work together on a project. If you add AWS tags to repositories, you can set up notifications so that repository users receive email about events (for example, another user commenting on code). You can also change the default settings for your repository, browse its contents, and more. You can create triggers for your repository so that code pushes or other events trigger actions, such as emails or code functions. You can even configure a repository on your local computer (a local repo) to push your changes to more than one repository.



Before you can push changes to a CodeCommit repository, you must configure an IAM user in your Amazon Web Services account, or set up access for federated access or temporary credentials. For more information, see Step 1: Initial configuration for CodeCommit and Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

For information about working with other aspects of your repository in CodeCommit, see Working with files, Working with pull requests, Working with commits, Working with branches, and

Working with user preferences. For information about migrating to CodeCommit, see Migrate to CodeCommit.

Topics

- Create an AWS CodeCommit repository
- Connect to an AWS CodeCommit repository
- Share a AWS CodeCommit repository
- Configuring notifications for events in an AWS CodeCommit repository
- Tagging repositories in AWS CodeCommit
- Manage triggers for an AWS CodeCommit repository
- Amazon CodeGuru Reviewer and AWS CodeCommit
- View CodeCommit repository details
- Change AWS CodeCommit repository settings
- Synchronize changes between a local repo and an AWS CodeCommit repository
- Push commits to an additional Git repository
- Configure cross-account access to an AWS CodeCommit repository using roles
- Delete an AWS CodeCommit repository

Create an AWS CodeCommit repository

Use the AWS CodeCommit console or the AWS Command Line Interface (AWS CLI) to create an empty CodeCommit repository. To add tags to a repository after you create it, see Add a tag to a repository.

These instructions assume that you have completed the steps in Setting up .

> **Note**
>
> Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the CodeCommit product information page.

Topics

- Create a repository (console)
- Create a repository (AWS CLI)

Create a repository (console)

To create a CodeCommit repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where you want to create the repository. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose **Create repository**.
4. On the **Create repository** page, in **Repository name**, enter a name for the repository.

   > **Note**
   >
   > Repository names are case sensitive. The name must be unique in the AWS Region for your Amazon Web Services account.

5. (Optional) In **Description**, enter a description for the repository. This can help you and other users identify the purpose of the repository.

   > **Note**
   >
   > The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

6. (Optional) Choose **Add tag** to add one or more repository tags (a custom attribute label that helps you organize and manage your AWS resources) to your repository. For more information, see Tagging repositories in AWS CodeCommit.
7. (Optional) Expand **Additional configuration** to specify whether to use the default AWS managed key or your own customer managed key for encrypting and decrypting data in this repository. If you choose to use your own customer managed key, you must ensure that it is available in the AWS Region where you are creating the repository, and that the key is active. For more information, see AWS Key Management Service and encryption for AWS CodeCommit repositories.
8. (Optional) Select **Enable Amazon CodeGuru Reviewer for Java and Python** if this repository contains Java or Python code, and you want CodeGuru Reviewer to analyze it. CodeGuru

Reviewer uses multiple machine learning models to find code defects and to suggest improvements and fixes in pull requests. For more information, see the *Amazon CodeGuru Reviewer User Guide*.

9. Choose **Create**.

After you create a repository, you can connect to it and start adding code either through the CodeCommit console or a local Git client, or by integrating your CodeCommit repository with your favorite IDE. For more information, see Setting up for AWS CodeCommit. You can also add your repository to a continuous delivery pipeline. For more information, see Simple Pipeline Walkthrough.

To get information about the new CodeCommit repository, such as the URLs to use when cloning the repository, choose the repository's name from the list, or just choose the connection protocol you want to use next to the repository's name.

To share this repository with others, you must send them the HTTPS or SSH link to use to clone the repository. Make sure they have the permissions required to access the repository. For more information, see Share a repository and Authentication and access control for AWS CodeCommit.

Create a repository (AWS CLI)

You can use the AWS CLI to create a CodeCommit repository. Unlike the console, you can add tags to a repository if you create it using the AWS CLI.

1. Make sure that you have configured the AWS CLI with the AWS Region where the repository exists. To verify the Region, run the following command at the command line or terminal and review the information for default region name.

   ```
   aws configure
   ```

   The default region name must match the AWS Region for the repository in CodeCommit. For more information, see Regions and Git connection endpoints.

2. Run the **create-repository** command, specifying:

   - A name that uniquely identifies the CodeCommit repository (with the `--repository-name` option).

> **Note**
>
> This name must be unique across an Amazon Web Services account.

- An optional comment about the CodeCommit repository (with the `--repository-description` option).
- An optional key-value pair or pairs to use as tags for the CodeCommit repository (with the `--tags` option).
- An optional customer managed key to use when encrypting and decrypting this repository. All repositories are encrypted in transit and at rest using a key in AWS KMS. If no key is specified, the default AWS managed key `aws/codecommit` is used.

For example, to create a CodeCommit repository named `MyDemoRepo` with the description `"My demonstration repository"` and a tag with a key named *`Team`* with the value of *`Saanvi`*, use this command.

```
aws codecommit create-repository --repository-name MyDemoRepo --repository-
description "My demonstration repository" --tags Team=Saanvi
```

> **Note**
>
> The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

3. If successful, this command outputs a `repositoryMetadata` object with the following information:

   - The description (`repositoryDescription`).
   - The unique, system-generated ID (`repositoryId`).
   - The name (`repositoryName`).
   - The ID of the Amazon Web Services account associated with the CodeCommit repository (`accountId`).

The following is example output, based on the preceding example command.

```
{
    "repositoryMetadata": {
        "repositoryName": "MyDemoRepo",
        "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo",
        "lastModifiedDate": 1446071622.494,
        "repositoryDescription": "My demonstration repository",
        "cloneUrlHttp": "https://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo",
        "defaultBranch": main,
        "kmsKeyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "creationDate": 1446071622.494,
        "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
        "Arn": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
        "accountId": "111111111111"
    }
}
```

Note

Tags that were added when the repository was created are not returned in the output. To view a list of tags associated with a repository, run the **list-tags-for-resource** command.

4. Make a note of the name and ID of the CodeCommit repository. You need them to monitor and change information about the CodeCommit repository, especially if you use AWS CLI.

   If you forget the name or ID, follow the instructions in View CodeCommit repository details (AWS CLI).

After you create a repository, you can connect to it and start adding code. For more information, see Connect to a repository. You can also add your repository to a continuous delivery pipeline. For more information, see Simple Pipeline Walkthrough.

Connect to an AWS CodeCommit repository

When you connect to a CodeCommit repository for the first time, you typically clone its contents to your local machine. You can also add files to and edit files in a repository directly from the CodeCommit console. Alternatively, if you already have a local repo, you can add a CodeCommit repository as a remote. This topic provides instructions for connecting to a CodeCommit repository. If you want to migrate an existing repository to CodeCommit, see Migrate to CodeCommit.

> **Note**
>
> Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the CodeCommit product information page.

Topics

- Prerequisites for connecting to a CodeCommit repository
- Connect to the CodeCommit repository by cloning the repository
- Connect a local repo to the CodeCommit repository

Prerequisites for connecting to a CodeCommit repository

Before you can clone a CodeCommit repository or connect a local repo to an CodeCommit repository:

- You must have configured your local computer with the software and settings required to connect to CodeCommit. This includes installing and configuring Git. For more information, see Setting up and Getting started with Git and AWS CodeCommit.
- You must have the clone URL of the CodeCommit repository to which you want to connect. For more information, see View repository details.

  If you have not yet created a CodeCommit repository, follow the instructions in Create a repository, copy the clone URL of the CodeCommit repository, and return to this page.

  If you have a CodeCommit repository but you do not know its name, follow the instructions in View repository details.
- You must have a location on your local machine to store a local copy of the CodeCommit repository you connect to. (This local copy of the CodeCommit repository is known as a *local*

repo.) You then switch to and run Git commands from that location. For example, you could use `/tmp` (for Linux, macOS, or Unix) or `c:\temp` (for Windows) if you are making a temporary clone for testing purposes. That is the directory path used in these examples.

> **Note**
>
> You can use any directory you want. If you are cloning a repository for long-term use, consider creating the clone from a working directory and not one used for temporary files. If you are using a directory different from `/tmp` or `c:\temp`, be sure to substitute that directory for ours when you follow these instructions.

Connect to the CodeCommit repository by cloning the repository

If you do not already have a local repo, follow the steps in this procedure to clone the CodeCommit repository to your local machine.

1. Complete the prerequisites, including Setting up .

   > **Important**
   >
   > If you have not completed setup, you cannot connect to or clone the repository.

2. From the `/tmp` directory or the `c:\temp` directory, use Git to run the **clone** command. The following examples show how to clone a repository named *MyDemoRepo* in the US East (Ohio) Region.

   For HTTPS using Git credentials or the credential helper included with the AWS CLI:

   ```
   git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   For HTTPS using **git-remote-codecommit**, assuming the default profile and AWS Region configured in the AWS CLI:

   ```
   git clone codecommit://MyDemoRepo my-demo-repo
   ```

   For SSH:

```
git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

In this example, `git-codecommit.us-east-2.amazonaws.com` is the Git connection point for the US East (Ohio) Region where the repository exists, `MyDemoRepo` represents the name of your CodeCommit repository, and `my-demo-repo` represents the name of the directory Git creates in the `/tmp` directory or the `c:\temp` directory. For more information about the AWS Regions that support CodeCommit and the Git connections for those AWS Regions, see Regions and Git connection endpoints.

Note

When you use SSH on Windows operating systems to clone a repository, you might need to add the SSH key ID to the connection string as follows:

```
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

For more information, see For SSH connections on Windows and Troubleshooting.

After Git creates the directory, it pulls down a copy of your CodeCommit repository into the newly created directory.

If the CodeCommit repository is new or otherwise empty, you see a message that you are cloning an empty repository. This is expected.

Note

If you receive an error that Git can't find the CodeCommit repository or that you don't have permission to connect to the CodeCommit repository, make sure you completed the prerequisites, including assigning permissions to the IAM user and setting up your IAM user credentials for Git and CodeCommit on the local machine. Also, make sure you specified the correct repository name.

After you successfully connect your local repo to your CodeCommit repository, you are now ready to start running Git commands from the local repo to create commits, branches, and tags and push to and pull from the CodeCommit repository.

Connect a local repo to the CodeCommit repository

Complete the following steps if you already have a local repo and want to add a CodeCommit repository as the remote repository. If you already have a remote repository and want to push your commits to CodeCommit and that other remote repository, follow the steps in Push commits to two repositories.

1. Complete the prerequisites.
2. From the command prompt or terminal, switch to your local repo directory and run the **git remote add** command to add the CodeCommit repository as a remote repository for your local repo.

   For example, the following command adds the remote nicknamed **`origin`** to https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo:

   For HTTPS:

   ```
   git remote add origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/
   MyDemoRepo
   ```

   For SSH:

   ```
   git remote add origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/
   MyDemoRepo
   ```

   This command returns nothing.
3. To verify that you have added the CodeCommit repository as a remote for your local repo, run the **git remote -v** command , which should create output similar to the following:

   For HTTPS:

   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   ```

```
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

For SSH:

```
origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)

origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

After you successfully connect your local repo to your CodeCommit repository, you are ready to start running Git commands from the local repo to create commits, branches, and tags, and to push to and pull from the CodeCommit repository.

Share a AWS CodeCommit repository

After you have created a CodeCommit repository, you can share it with other users. First, determine whether you will use a federated access, temporary credentials, or a web identity provider such as IAM Identity Center when accessing CodeCommit, or if you want to use Git credentials or SSH key pairs with IAM users. If you're using the former, you will need to set up users, access, and permissions for your identity provider, and then provide instructions for your users to use **git-remote-codecommit**. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit and Connecting to AWS CodeCommit repositories with rotating credentials.

You cannot use Git credentials or SSH key pairs with federated access or identity providers, but many IDEs work best with these credentials. In this case, decide which protocol (HTTPS or SSH) to recommend to users when cloning and using a Git client or an IDE to connect to your repository. Then send the URL and connection information to the users with whom you want to share the repository. Depending on your security requirements, sharing a repository might also require creating an IAM group, applying managed policies to that group, and editing IAM policies to refine access, or creating and using IAM roles.

> **Note**
>
> After you have granted users console access to the repository, they can add or edit files directly in the console without having to set up a Git client or other connection. For more

information, see Create or add a file to an AWS CodeCommit repository and Edit the contents of a file in an AWS CodeCommit repository.

These instructions are written with the assumption that you have already completed the steps in Setting up and Create a repository.

> **Note**
>
> Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the CodeCommit product information page.

Topics

- Choose the connection protocol to share with your users
- Create IAM policies for your repository
- Create an IAM group for repository users
- Share the connection information with your users

Choose the connection protocol to share with your users

When you create a repository in CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active regardless of which protocol you recommend to your users.

HTTPS connections require either:

- Git credentials, which IAM users can generate for themselves in IAM. Git credentials are the easiest method for users of your repository to set up and use.
- An AWS access key or role to assume, which your repository users must configure in their credential profile. You can configure **git-remote-codecommit** (recommended) or the credential helper included in the AWS CLI. These are the only methods available for root account or federated users.

SSH connections require your users to:

- Generate a public-private key pair.
- Store the public key.
- Associate the public key with their IAM user.
- Configure their known hosts file on their local computer.
- Create and maintain a config file on their local computers.

Because this is a more complex configuration process, we recommend that you choose HTTPS and Git credentials for connections to CodeCommit.

For more information about HTTPS, SSH, Git, **git-remote-codecommit**, and remote repositories, see Setting up , Connecting to AWS CodeCommit repositories with rotating credentials, or consult your Git documentation.

> **Note**
>
> Although Git supports a variety of connection protocols, CodeCommit does not support connections with unsecured protocols, such as the local protocol or generic HTTP.

Create IAM policies for your repository

AWS provides three managed policies in IAM for CodeCommit. These policies cannot be edited and apply to all repositories associated with your Amazon Web Services account. However, you can use these policies as templates to create your own custom managed policies that apply only to the repository you want to share. Your customer managed policy can apply specifically to the repository you want to share. For more information, see Managed Policies and IAM Users and Groups.

> **Tip**
>
> For more fine-grained control over access to your repository, you can create more than one customer managed policy and apply the policies to different IAM users and groups.

For information about reviewing the contents of managed policies and using policies to create and apply permissions, see Authentication and access control for AWS CodeCommit.

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page,, choose **Import managed policy**.
4. On the **Import managed policies** page, in **Filter policies**, enter **`AWSCodeCommitPowerUser`**. Choose the button next to the policy name and then choose **Import**.
5. On the **Create policy** page, choose **JSON**. Replace the "*" portion of the `Resource` line for CodeCommit actions with the Amazon Resource Name (ARN) of the CodeCommit repository, as shown here:

```
"Resource": [
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo"
 ]
```

> **Tip**
>
> To find the ARN for the CodeCommit repository, go to the CodeCommit console, choose the repository name from the list, and then choose **Settings**. For more information, see View repository details.

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown here:

```
"Resource": [
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
 "arn:aws:codecommit:us-east-2:111111111111:MyOtherDemoRepo"
 ]
```

When you are finished editing, choose **Review policy**.

6. On the **Review Policy** page, in **Name**, enter a new name for the policy (for example, *`AWSCodeCommitPowerUser-MyDemoRepo`*). Optionally provide a description for this policy.
7. Choose **Create Policy**.

Create an IAM group for repository users

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step. Alternatively, you can create a role with an attached customer managed policy, and have users assume that role.

If you use SSH, you must attach another managed policy to the IAMUserSSHKeys group, the IAM managed policy that allows users to upload their SSH public key and associate it with the IAM user they use to connect to CodeCommit.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Groups**, and then choose **Create New Group**.
3. On the **Set Group Name** page, in **Group Name**, enter a name for the group (for example, *MyDemoRepoGroup*), and then choose **Next Step**. Consider including the repository name as part of the group name.

   > **Note**
   >
   > This name must be unique across an Amazon Web Services account.

4. Select the box next to the customer managed policy you created in the previous section (for example, **AWSCodeCommitPowerUser-MyDemoRepo**).
5. On the **Review** page, choose **Create Group**. IAM creates this group with the specified policies already attached. The group appears in the list of groups associated with your Amazon Web Services account.
6. Choose your group from the list.
7. On the group summary page, choose the **Users** tab, and then choose **Add Users to Group**. On the list that shows all users associated with your Amazon Web Services account, select the boxes next to the users to whom you want to allow access to the CodeCommit repository, and then choose **Add Users**.

   > **Tip**
   >
   > You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

Share the connection information with your users

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose the repository you want to share.
4. In **Clone URL**, choose the protocol that you want your users to use. This copies the clone URL for the connection protocol.
5. Send your users the clone URL along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, HTTPS).

The following example email provides information for users connecting to the MyDemoRepo repository with the HTTPS connection protocol and Git credentials in the US East (Ohio) (us-east-2) Region. This email is written with the assumption the user has already installed Git and is familiar with using it.

```
I've created a CodeCommit repository for us to use while working on our project.
The name of the repository is MyDemoRepo, and
it is in the  US East (Ohio) (us-east-2) region.
Here's what you need to do in order to get started using it:

1. Make sure that your version of Git on your local computer is 1.7.9 or later.
2. Generate Git credentials for your IAM user by signing into the IAM console
 here: https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/  .
Switch to the Security credentials tab for your IAM user and choose the Generate button
 in HTTPS Git credentials for CodeCommit.
Make sure to save your credentials in a secure location!
3. Switch to a directory of your choice and clone the CodeCommit repository to your
 local machine by running the following command:
    git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-
demo-repo
4. When prompted for user name and password, use the Git credentials you just saved.
```

```
That's it!  If you'd like to learn more about using CodeCommit, you can start with the
 tutorial here.
```

You can find complete setup instructions in Setting up .

Configuring notifications for events in an AWS CodeCommit repository

Topics

- Change or disable notifications
- Delete notifications

Change or disable notifications

To change notification settings

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**. If you see a banner informing you that you have notifications instead of notification rules, choose **Manage existing notifications**.
4. Choose **Edit**.
5. Make your changes, and then choose **Save**.

Disabling notifications is an easy way to temporarily prevent users from receiving emails about repository events.

To disable notifications

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to disable notifications.

3. In the navigation pane, choose **Settings**, and then choose **Notifications**. Choose **Manage existing notifications**.
4. Choose **Edit**, and in **Event status**, use the slider to turn off **Enable notifications**. Choose **Save**.
5. The event status changes to **Disabled**. No emails about events are sent. When you disable notifications, the CloudWatch Events rule for the repository is disabled automatically. Do not manually change its status in the CloudWatch Events console.

Delete notifications

To delete notification settings

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**. If you see a banner informing you that you have notifications instead of notification rules, choose **Manage existing notifications**.
4. In **CloudWatch event rule**, copy the name of the rule that was created for the notification.
5. Sign in to the AWS Management Console and open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
6. In **Events**, choose **Rules**. In **Name**, paste the name of the rule created for the notification. Choose the rule, and in **Actions**, choose **Delete**.
7. (Optional) To change or delete the Amazon SNS topic used for notifications after you delete notification settings, go to the Amazon SNS console at https://eusc-de-east-1.console.amazonaws-eusc.eu/sns/v3/home. For more information, see Clean Up in Amazon Simple Notification Service Developer Guide.

Tagging repositories in AWS CodeCommit

A *tag* is a custom attribute label that you or AWS assigns to an AWS resource. AWS tags are different from Git tags, which can be applied to commits. Each AWS tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, `Project`, or `Secret`). Tag keys are case sensitive.

- An optional field known as a *tag value* (for example, `111122223333`, `Production`, or a team name). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Together these are known as key-value pairs. For limits on the number of tags you can have on a repository and restrictions on tag keys and values, see Limits.

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to a CodeCommit repository that you assign to an Amazon S3 bucket. For more information about tagging strategies, see Tagging AWS Resources.

In CodeCommit, the primary resource is a repository. You can use the CodeCommit console, the AWS CLI, CodeCommit APIs, or AWS SDKs to add, manage, and remove tags for a repository. In addition to identifying, organizing, and tracking your repository with tags, you can use tags in IAM policies to help control who can view and interact with your repository. For examples of tag-based access policies, see Example 5: Deny or allow actions on repositories with tags.

Topics

- Add a tag to a repository
- View tags for a repository
- Edit tags for a repository
- Remove a tag from a repository

Add a tag to a repository

Adding tags to a repository can help you identify and organize your AWS resources and manage access to them. First, you add one or more tags (key-value pairs) to a repository. Keep in mind that there are limits on the number of tags you can have on a repository. There are restrictions on the characters you can use in the key and value fields. For more information, see Limits. After you have tags, you can create IAM policies to manage access to the repository based on these tags. You can use the CodeCommit console or the AWS CLI to add tags to a repository.

> **Important**
>
> Adding tags to a repository can impact access to that repository. Before you add a tag to a repository, make sure to review any IAM policies that might use tags to control access to

resources such as repositories. For examples of tag-based access policies, see Example 5: Deny or allow actions on repositories with tags.

For more information about adding tags to a repository when you create it, see Create a repository (console).

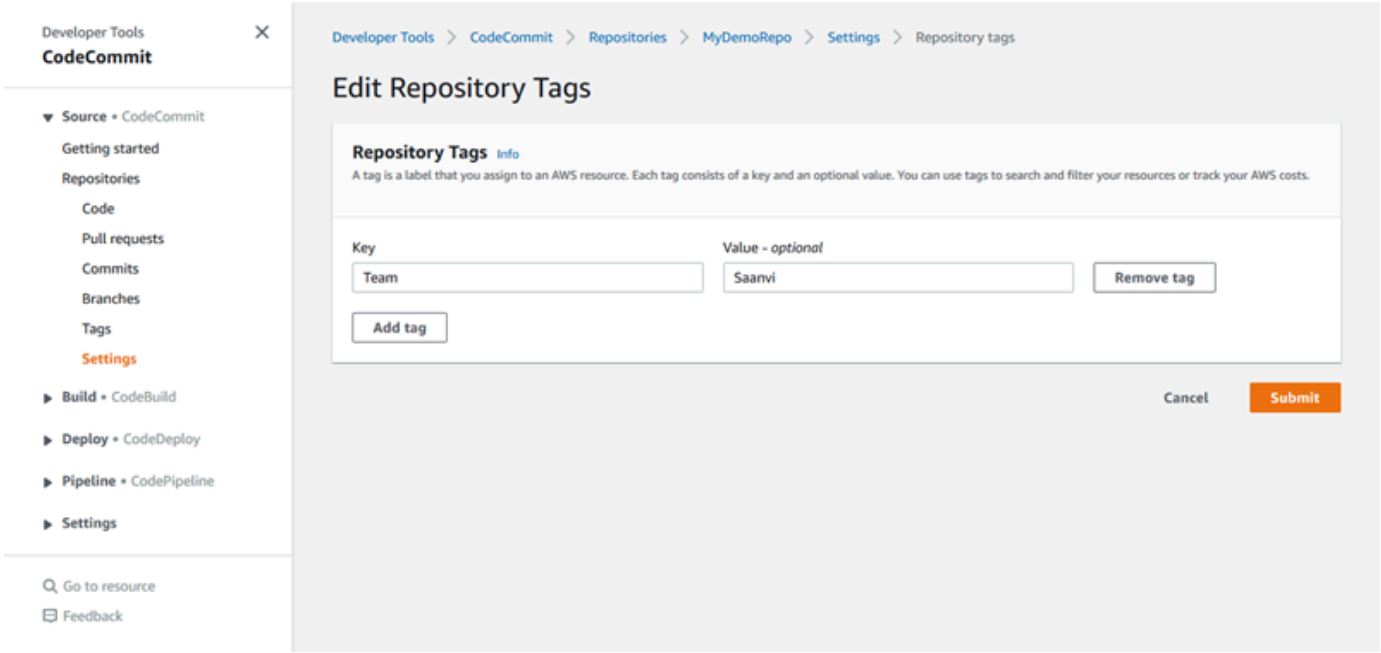
Topics

- Add a tag to a repository (console)
- Add a tag to a repository (AWS CLI)

Add a tag to a repository (console)

You can use the CodeCommit console to add one or more tags to a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to add tags.
3. In the navigation pane, choose **Settings**. Choose **Repository tags**.
4. If no tags have been added to the repository, choose **Add tag**. Otherwise, choose **Edit**, and then choose **Add tag**.
5. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.



6. (Optional) To add another tag, choose **Add tag** again.
7. When you have finished adding tags, choose **Submit**.

Add a tag to a repository (AWS CLI)

Follow these steps to use the AWS CLI to add a tag to a CodeCommit repository. To add a tag to a repository when you create it, see Create a repository (AWS CLI).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the **tag-resource** command, specifying the Amazon Resource Name (ARN) of the repository where you want to add tags and the key and value of the tag you want to add. You can add more than one tag to a repository. For example, to tag a repository named *MyDemoRepo* with two tags, a tag key named *Status* with the tag value of *Secret*, and a tag key named *Team* with the tag value of *Saanvi*:

```
aws codecommit tag-resource --resource-arn arn:aws:codecommit:us-
west-2:111111111111:MyDemoRepo --tags Status=Secret,Team=Saanvi
```

If successful, this command returns nothing.

View tags for a repository

Tags can help you identify and organize your AWS resources and manage access to them. For more information about tagging strategies, see Tagging AWS Resources. For examples of tag-based access policies, see Example 5: Deny or allow actions on repositories with tags.

View tags for a repository (console)

You can use the CodeCommit console to view the tags associated with a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to view tags.
3. In the navigation pane, choose **Settings**. Choose **Repository tags**.

View tags for a repository (AWS CLI)

Follow these steps to use the AWS CLI to view the AWS tags for a CodeCommit repository. If no tags have been added, the returned list is empty.

At the terminal or command line, run the **list-tags-for-resource** command. For example, to view a list of tag keys and tag values for a repository named *MyDemoRepo* with the ARN *arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo*:

```
aws codecommit list-tags-for-resource --resource-arn arn:aws:codecommit:us-
west-2:111111111111:MyDemoRepo
```

If successful, this command returns information similar to the following:

```
{
    "tags": {
        "Status": "Secret",
        "Team": "Saanvi"
    }
}
```

Edit tags for a repository

You can change the value for a tag associated with a repository. You can also change the name of the key, which is equivalent to removing the current tag and adding a different one with the new name and the same value as the other key. Keep in mind that there are limits on the characters you can use in the key and value fields. For more information, see Limits.

> **Important**
>
> Editing tags for a repository can impact access to that repository. Before you edit the name (key) or value of a tag for a repository, make sure to review any IAM policies that might use the key or value for a tag to control access to resources such as repositories. For examples of tag-based access policies, see Example 5: Deny or allow actions on repositories with tags.

Edit a tag for a repository (console)

You can use the CodeCommit console to edit the tags associated with a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to edit tags.
3. In the navigation pane, choose **Settings**. Choose **Repository tags**.
4. Choose **Edit**.
5. Do one of the following:
   - To change the tag, enter a new name in **Key**. Changing the name of the tag is the equivalent of removing a tag and adding a new tag with the new key name.
   - To change the value of a tag, enter a new value. If you want to change the value to nothing, delete the current value and leave the field blank.
6. When you have finished editing tags, choose **Submit**.

Edit tags for a repository (AWS CLI)

Follow these steps to use the AWS CLI to update a tag for a CodeCommit repository. You can change the value for an existing key, or add another key.

At the terminal or command line, run the **tag-resource** command, specifying the Amazon Resource Name (ARN) of the repository where you want to update a tag and specify the tag key and tag value:

```
aws codecommit tag-resource --resource-arn arn:aws:codecommit:us-
west-2:111111111111:MyDemoRepo --tags Team=Li
```

Remove a tag from a repository

You can remove one or more tags associated with a repository. Removing a tag does not delete the tag from other AWS resources that are associated with that tag.

> **Important**
>
> Removing tags for a repository can impact access to that repository. Before you remove a tag from a repository, make sure to review any IAM policies that might use the key or value for a tag to control access to resources such as repositories. For examples of tag-based access policies, see Example 5: Deny or allow actions on repositories with tags.

Remove a tag from a repository (console)

You can use the CodeCommit console to remove the association between a tag and a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to remove tags.
3. In the navigation pane, choose **Settings**. Choose **Repository tags**.
4. Choose **Edit**.
5. Find the tag you want to remove, and then choose **Remove tag**.
6. When you have finished removing tags, choose **Submit**.

Remove a tag from a repository (AWS CLI)

Follow these steps to use the AWS CLI to remove a tag from a CodeCommit repository. Removing a tag does not delete it, but simply removes the association between the tag and the repository.

> **Note**
>
> If you delete a CodeCommit repository, all tag associations are removed from the deleted repository. You do not have to remove tags before you delete a repository.

At the terminal or command line, run the **untag-resource** command, specifying the Amazon Resource Name (ARN) of the repository where you want to remove tags and the tag key of the tag you want to remove. For example, to remove a tag on a repository named *MyDemoRepo* with the tag key *Status*:

```
aws codecommit untag-resource --resource-arn arn:aws:codecommit:us-
west-2:111111111111:MyDemoRepo --tag-keys Status
```

If successful, this command returns nothing. To verify the tags associated with the repository, run the **list-tags-for-resource** command.

Manage triggers for an AWS CodeCommit repository

You can configure a CodeCommit repository so that code pushes or other events trigger actions, such as sending a notification from Amazon Simple Notification Service (Amazon SNS) or invoking a function in AWS Lambda. You can create up to 10 triggers for each CodeCommit repository.

Triggers are commonly configured to:

- Send emails to subscribed users every time someone pushes to the repository.
- Notify an external build system to start a build after someone pushes to the main branch of the repository.

Scenarios like notifying an external build system require writing a Lambda function to interact with other applications. The email scenario simply requires creating an Amazon SNS topic.

This topic shows you how to set permissions that allow CodeCommit to trigger actions in Amazon SNS and Lambda. It also includes links to examples for creating, editing, testing, and deleting triggers.

Topics

- Create the resource and add permissions for CodeCommit
- Example: Create an AWS CodeCommit trigger for an Amazon SNS topic
- Example: Create an AWS CodeCommit trigger for an AWS Lambda function
- Example: Create a trigger in AWS CodeCommit for an existing AWS Lambda function
- Edit triggers for an AWS CodeCommit repository
- Test triggers for an AWS CodeCommit repository
- Delete triggers from an AWS CodeCommit repository

Create the resource and add permissions for CodeCommit

You can integrate Amazon SNS topics and Lambda functions with triggers in CodeCommit, but you must first create and then configure resources with a policy that grants CodeCommit the permissions to interact with those resources. You must create the resource in the same AWS Region as the CodeCommit repository. For example, if the repository is in US East (Ohio) (us-east-2), the Amazon SNS topic or Lambda function must be in US East (Ohio).

- For Amazon SNS topics, you do not need to configure additional IAM policies or permissions if the Amazon SNS topic is created using the same account as the CodeCommit repository. You can create the CodeCommit trigger as soon as you have created and subscribed to the Amazon SNS topic.
  - For more information about creating topics in Amazon SNS, see the Amazon SNS documentation.
  - For information about using Amazon SNS to send messages to Amazon SQS queues, see Sending Messages to Amazon SQS Queues in the *Amazon SNS Developer Guide*.
  - For information about using Amazon SNS to invoke a Lambda function, see Invoking Lambda Functions in the *Amazon SNS Developer Guide*.
- You can configure Lambda functions by creating the trigger in the Lambda console as part of the function. This is the simplest method, because triggers created in the Lambda console automatically include the permissions required for CodeCommit to invoke the Lambda function. If you create the trigger in CodeCommit, you must include a policy to allow CodeCommit to invoke the function. For more information, see Create a trigger for an existing Lambda function.

Example: Create an AWS CodeCommit trigger for an Amazon SNS topic

You can create a trigger for a CodeCommit repository so that events in that repository trigger notifications from an Amazon Simple Notification Service (Amazon SNS) topic. You might want to create a trigger to an Amazon SNS topic to enable users to subscribe to notifications about repository events, such as the deletion of branches. You can also take advantage of the integration of Amazon SNS topics with other services, such as Amazon Simple Queue Service (Amazon SQS) and AWS Lambda.

> **Note**
>
> - You must point the trigger to an existing Amazon SNS topic that is the action taken in response to repository events. For more information about creating and subscribing to Amazon SNS topics, see Getting Started with Amazon Simple Notification Service.
> - Amazon SNS FIFO (first in, first out) topics are not supported for CodeCommit triggers.

Topics

- Create a trigger to an Amazon SNS topic for a CodeCommit repository (console)

- Create a trigger to an Amazon SNS topic for a CodeCommit repository (AWS CLI)

Create a trigger to an Amazon SNS topic for a CodeCommit repository (console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to create triggers for repository events.
3. In the navigation pane for the repository, choose **Settings**, and then choose **Triggers**.
4. Choose **Create trigger**, and then do the following:.

   - In **Trigger name**, enter a name for the trigger (for example, `MyFirstTrigger`).
   - In **Events**, choose the repository events that trigger the Amazon SNS topic to send notifications.

     If you choose **All repository events**, you cannot choose any other events. To choose a subset of events, remove **All repository events**, and then choose one or more events from the list. For example, if you want the trigger to run only when a user creates a branch or tag in the CodeCommit repository, remove **All repository events**, and then choose **Create branch or tag**.
   - If you want the trigger to apply to all branches of the repository, in **Branches**, leave the selection blank, as this default option applies the trigger to all branches automatically. If you want this trigger to apply to specific branches only, choose up to 10 branch names from the list of repository branches.
   - In **Choose the service to use**, choose **Amazon SNS**.
   - In **Amazon SNS**, choose a topic name from the list or enter the ARN for the topic.

     > **Note**
     >
     > Amazon SNS FIFO (first in, first out) topics are not supported for CodeCommit triggers. You must choose an Amazon SNS topic that has its type set to Standard.

   - In **Custom data**, provide any optional information you want included in the notification sent by the Amazon SNS topic (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters.

5. (Optional) Choose **Test trigger**. This step helps you confirm have correctly configured access between CodeCommit and the Amazon SNS topic. It uses the Amazon SNS topic to send a test notification using data from your repository, if available. If no real data is available, the test notification contains sample data.
6. Choose **Create trigger** to finish creating the trigger.

Create a trigger to an Amazon SNS topic for a CodeCommit repository (AWS CLI)

You can also use the command line to create a trigger for an Amazon SNS topic in response to CodeCommit repository events, such as when someone pushes a commit to your repository.

To create a trigger for an Amazon SNS topic

1. Open a plain-text editor and create a JSON file that specifies:

   - The Amazon SNS topic name.

     > **Note**
     >
     > Amazon SNS FIFO (first in, first out) topics are not supported for CodeCommit triggers. You must choose an Amazon SNS topic that has its type set to Standard.

   - The repository and branches you want to monitor with this trigger. (If you do not specify any branches, the trigger applies to all branches in the repository.)
   - The events that activate this trigger.

   Save the file.

   For example, to create a trigger for a repository named *MyDemoRepo* that publishes all repository events to an Amazon SNS topic named *MySNSTopic* for two branches, *main* and *preprod*:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "name": "MyFirstTrigger",
            "destinationArn": "arn:aws:sns:us-east-2:111122223333:MySNSTopic",
            "customData": "",
```

```
            "branches": [
                "main", "preprod"
            ],
            "events": [
                "all"
            ]
        }
    ]
}
```

There must be a trigger block in the JSON for each trigger for a repository. To create more than one trigger for the repository, include more than one trigger block in the JSON. Remember that all triggers created in this file are for the specified repository. You cannot create triggers for multiple repositories in a single JSON file. For example, if you wanted to create two triggers for a repository, you can create a JSON file with two trigger blocks. In the following example, no branches are specified for the second trigger, so that trigger applies to all branches:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "name": "MyFirstTrigger",
            "destinationArn": "arn:aws:sns:us-east-2:111122223333:MySNSTopic",
            "customData": "",
            "branches": [
                "main", "preprod"
            ],
            "events": [
                "all"
            ]
        },
        {
            "name": "MySecondTrigger",
            "destinationArn": "arn:aws:sns:us-east-2:111122223333:MySNSTopic2",
            "customData": "",
            "branches": [],
            "events": [
                "updateReference", "deleteReference"
            ]
        }
```

```
    ]
}
```

You can create triggers for events you specify, such as when a commit is pushed to a repository. Event types include:

- `all` for all events in the specified repository and branches.
- `updateReference` for when commits are pushed to the specified repository and branches.
- `createReference` for when a new branch or tag is created in the specified repository.
- `deleteReference` for when a branch or tag is deleted in the specified repository.

> **Note**
>
> You can use more than one event type in a trigger. However, if you specify `all`, you cannot specify other events.

To see the full list of valid event types, at the terminal or command prompt, enter **aws codecommit put-repository-triggers help**.

In addition, you can include a string in `customData` (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters. This string is appended as an attribute to the CodeCommit JSON returned in response to the trigger.

2. (Optional) At a terminal or command prompt, run the **test-repository-triggers** command. This test uses sample data from the repository (or generates sample data if no data is available) to send a notification to the subscribers of the Amazon SNS topic. For example, the following is used to test that the JSON in the trigger file named *`trigger.json`* is valid and that CodeCommit can publish to the Amazon SNS topic:

```
aws codecommit test-repository-triggers --cli-input-json file://trigger.json
```

If successful, this command returns information similar to the following:

```
{
    "successfulExecutions": [
        "MyFirstTrigger"
```

```
    ],
    "failedExecutions": []
}
```

3. At a terminal or command prompt, run the **put-repository-triggers** command to create the trigger in CodeCommit. For example, to use a JSON file named *trigger.json* to create the trigger:

   ```
   aws codecommit put-repository-triggers --cli-input-json
   file://trigger.json
   ```

   This command returns a configuration ID, similar to the following:

   ```
   {
       "configurationId": "0123456-I-AM-AN-EXAMPLE"
   }
   ```

4. To view the configuration of the trigger, run the **get-repository-triggers** command, specifying the name of the repository:

   ```
   aws codecommit get-repository-triggers --repository-name MyDemoRepo
   ```

   This command returns the structure of all triggers configured for the repository, similar to the following:

   ```
   {
       "configurationId": "0123456-I-AM-AN-EXAMPLE",
       "triggers": [
           {
               "events": [
                   "all"
               ],
               "destinationArn": "arn:aws:sns:us-east-2:111122223333:MySNSTopic",
               "branches": [
                   "main",
                   "preprod"
               ],
               "name": "MyFirstTrigger",
               "customData": "Project ID 12345"
           }
       ]
   }
   ```

5. To test the functionality of the trigger itself, make and push a commit to the repository where you configured the trigger. You should see a response from the Amazon SNS topic. For example, if you configured the Amazon SNS topic to send an email, you should see an email from Amazon SNS in the email account subscribed to the topic.

   The following is example output from an email sent from Amazon SNS in response to a push to a CodeCommit repository:

```
{
  "Records":[
     {
        "awsRegion":"us-east-2",
        "codecommit":{
           "references" : [
              {
                    "commit":"317f8570EXAMPLE",
                    "created":true,
                    "ref":"refs/heads/NewBranch"
              },
              {
                    "commit":"4c925148EXAMPLE",
                    "ref":"refs/heads/preprod",
              }
            ]
          },
        "eventId":"11111-EXAMPLE-ID",
        "eventName":"ReferenceChange",
        "eventPartNumber":1,
        "eventSource":"aws:codecommit",
        "eventSourceARN":"arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
        "eventTime":"2016-02-09T00:08:11.743+0000",
        "eventTotalParts":1,
        "eventTriggerConfigId":"0123456-I-AM-AN-EXAMPLE",
        "eventTriggerName":"MyFirstTrigger",
        "eventVersion":"1.0",
        "customData":"Project ID 12345",
        "userIdentityARN":"arn:aws:iam::111122223333:user/JaneDoe-CodeCommit",
     }
  ]
}
```

Example: Create an AWS CodeCommit trigger for an AWS Lambda function

You can create a trigger for a CodeCommit repository so that events in the repository invoke a Lambda function. In this example, you create a Lambda function that returns the URL used to clone the repository to an Amazon CloudWatch log.

Topics

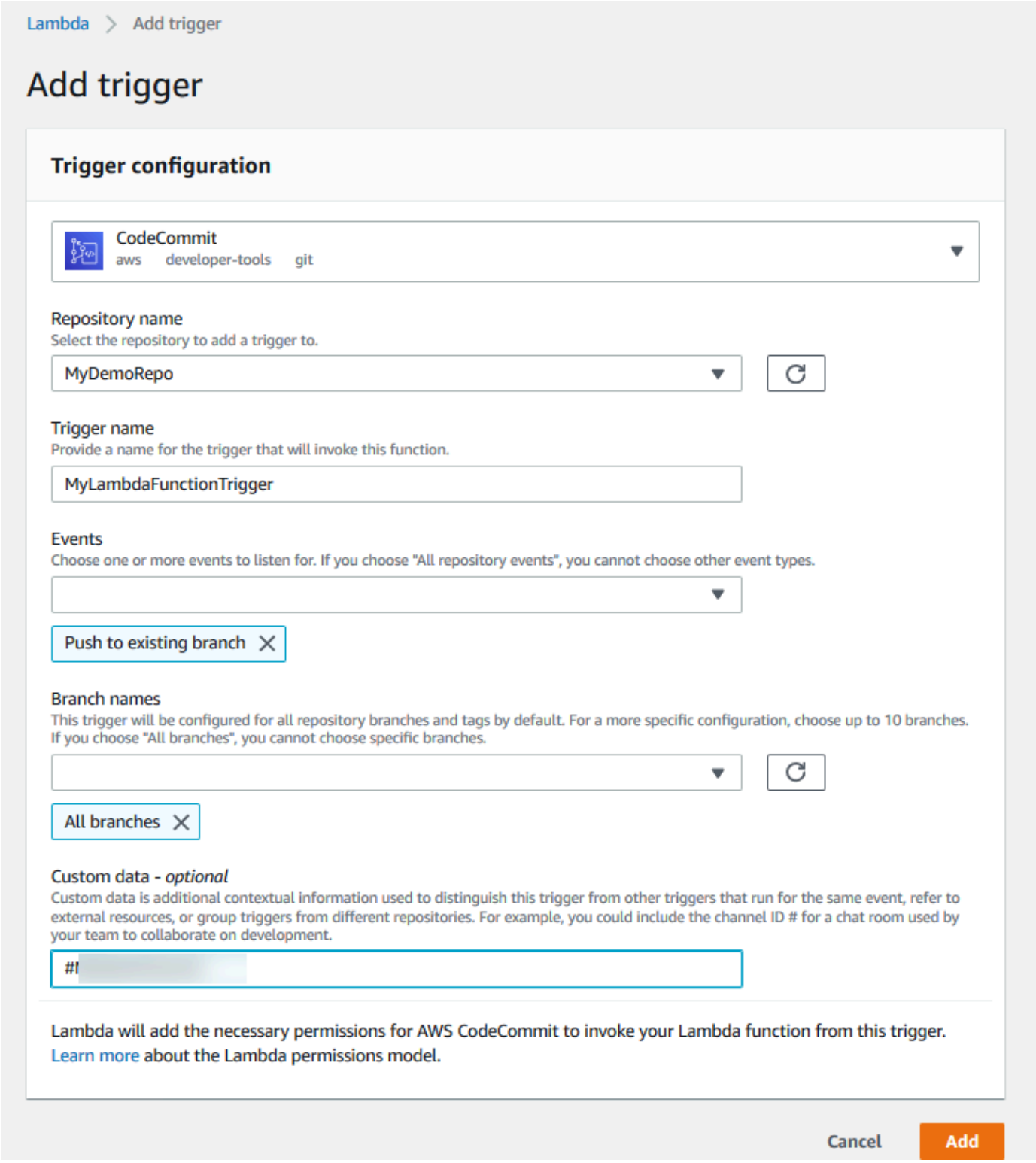
- Create the Lambda function
- View the trigger for the Lambda function in the AWS CodeCommit repository

Create the Lambda function

When you use the Lambda console to create the function, you can also create a CodeCommit trigger for the Lambda function. The following steps include a sample Lambda function. The sample is available in two languages: JavaScript and Python. The function returns the URLs used for cloning a repository to a CloudWatch log.

To create a Lambda function using a Lambda blueprint

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. On the **Lambda Functions** page, choose **Create function**. (If you have not used Lambda before, choose **Get Started Now**.)
3. On the **Create function** page, choose **Author from scratch**. In **Function Name**, provide a name for the function, for example `MyLambdaFunctionforCodeCommit`. In **Runtime**, choose the language you want to use to write your function, and then choose **Create function**.
4. On the **Configuration** tab, choose **Add trigger**.
5. In **Trigger configuration**, choose **CodeCommit** from the services drop-down list.



- In **Repository name**, choose the name of the repository where you want to configure a trigger that uses the Lambda function in response to repository events.

- In **Trigger name**, enter a name for the trigger (for example, *MyLambdaFunctionTrigger*).
- In **Events**, choose the repository events that trigger the Lambda function. If you choose **All repository events**, you cannot choose any other events. If you want to choose a subset of events, clear **All repository events**, and then choose the events you want from the list. For example, if you want the trigger to run only when a user creates a tag or a branch in the AWS CodeCommit repository, remove **All repository events**, and then choose **Create branch or tag**.
- If you want the trigger to apply to all branches of the repository, in **Branches**, choose **All branches**. Otherwise, choose **Specific branches**. The default branch for the repository is added by default. You can keep or delete this branch from the list. Choose up to 10 branch names from the list of repository branches.
- (Optional) In **Custom data**, enter information you want included in the Lambda function (for example, the name of the IRC channel used by developers to discuss development in the repository). This field is a string. It cannot be used to pass any dynamic parameters.

Choose **Add**.

6. On the **Configuration** page, in **Function Code**, in Code entry type, choose Edit code inline.. In **Runtime**, choose **Node.js**. If you want to create a sample Python function, choose **Python**.
7. In **Code entry type**, choose **Edit code inline**, and then replace the hello world code with one of the two following samples.

   For Node.js:

```
import {
  CodeCommitClient,
  GetRepositoryCommand,
} from "@aws-sdk/client-codecommit";

const codecommit = new CodeCommitClient({ region: "your-region" });

/**
 * @param {{ Records: { codecommit: { references: { ref: string }[] },
 eventSourceARN: string  }[]} event
 */
export const handler = async (event) => {
  // Log the updated references from the event
  const references = event.Records[0].codecommit.references.map(
    (reference) => reference.ref,
```

```
  );
  console.log("References:", references);

  // Get the repository from the event and show its git clone URL
  const repository = event.Records[0].eventSourceARN.split(":")[5];
  const params = {
    repositoryName: repository,
  };

  try {
    const data = await codecommit.send(new GetRepositoryCommand(params));
    console.log("Clone URL:", data.repositoryMetadata.cloneUrlHttp);
    return data.repositoryMetadata.cloneUrlHttp;
  } catch (error) {
    console.error("Error:", error);
    throw new Error(
      `Error getting repository metadata for repository ${repository}`,
    );
  }
};
```

For Python:

```
import json
import boto3

codecommit = boto3.client("codecommit")


def lambda_handler(event, context):
    # Log the updated references from the event
    references = {
        reference["ref"]
        for reference in event["Records"][0]["codecommit"]["references"]
    }
    print("References: " + str(references))

    # Get the repository from the event and show its git clone URL
    repository = event["Records"][0]["eventSourceARN"].split(":")[5]
    try:
        response = codecommit.get_repository(repositoryName=repository)
        print("Clone URL: " + response["repositoryMetadata"]["cloneUrlHttp"])
```

```
        return response["repositoryMetadata"]["cloneUrlHttp"]
    except Exception as e:
        print(e)
        print(
            "Error getting repository {}. Make sure it exists and that your
repository is in the same region as this function.".format(
                repository
            )
        )
        raise e
```

8. In the **Permissions** tab, in **Execution role**, choose the role to open it in the IAM console. Edit the attached policy to add `GetRepository` permission for the repository you want to use the trigger.

View the trigger for the Lambda function in the AWS CodeCommit repository

After you have created the Lambda function, you can view and test the trigger in AWS CodeCommit. Testing the trigger runs the function in response to the repository events you specify.

To view and test the trigger for the Lambda function

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to view triggers.
3. In the navigation pane for the repository, choose **Settings**, and then choose **Triggers**.
4. Review the list of triggers for the repository. You should see the trigger you created in the Lambda console. Choose it from the list and then choose **Test trigger**. This option attempts to invoke the function with sample data about your repository, including the most recent commit ID for the repository. (If no commit history exists, sample values consisting of zeroes are generated instead.) This helps you confirm that you have correctly configured access between AWS CodeCommit and the Lambda function.
5. To further verify the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the **Monitoring** tab for that function in the Lambda console. From the **Monitoring** tab, choose **View logs in CloudWatch**. The CloudWatch console opens in a new tab and displays events

for your function. Select the log stream from the list that corresponds to the time you pushed your commit. You should see event data similar to the following:

```
START RequestId: 70afdc9a-EXAMPLE Version: $LATEST
2015-11-10T18:18:28.689Z 70afdc9a-EXAMPLE References: [ 'refs/heads/main' ]
2015-11-10T18:18:29.814Z 70afdc9a-EXAMPLE Clone URL: https://git-codecommit.us-
east-2.amazonaws.com/v1/repos/MyDemoRepo
END RequestId: 70afdc9a-EXAMPLE
REPORT RequestId: 70afdc9a-EXAMPLE Duration: 1126.87 ms Billed Duration: 1200 ms
 Memory Size: 128 MB Max Memory Used: 14 MB
```

Example: Create a trigger in AWS CodeCommit for an existing AWS Lambda function

The easiest way to create a trigger that invokes a Lambda function is to create that trigger in the Lambda console. This built-in integration ensures that CodeCommit has the permissions required to run the function. To add a trigger for an existing Lambda function, go to the Lambda console, and choose the function. On the **Triggers** tab for the function, follow the steps in **Add trigger**. These steps are similar to the ones in Create the Lambda function.

You can also create a trigger for a Lambda function in a CodeCommit repository. Doing so requires that you choose an existing Lambda function to invoke. It also requires that you manually configure the permissions required for CodeCommit to run the function.

Topics

- Manually configure permissions to allow CodeCommit to run a Lambda function
- Create a trigger for the Lambda function in a CodeCommit repository (console)
- Create a trigger to a Lambda function for a CodeCommit repository (AWS CLI)

Manually configure permissions to allow CodeCommit to run a Lambda function

If you create a trigger in CodeCommit that invokes a Lambda function, you must manually configure the permissions that allow CodeCommit to run the Lambda function. To avoid this manual configuration, consider creating the trigger for the function in the Lambda console instead.

To allow CodeCommit to run a Lambda function

1. Open a plain-text editor and create a JSON file that specifies the Lambda function name, the details of the CodeCommit repository, and the actions you want to allow in Lambda, similar to the following:

```
{
    "FunctionName": "MyCodeCommitFunction",
    "StatementId": "1",
    "Action": "lambda:InvokeFunction",
    "Principal": "codecommit.amazonaws.com",
    "SourceArn": "arn:aws:codecommit:us-east-1:111122223333:MyDemoRepo",
    "SourceAccount": "111122223333"
}
```

2. Save the file as a JSON file with a name that is easy for you to remember (for example, *AllowAccessfromMyDemoRepo*.json).

3. Using the JSON file you just created, at the terminal (Linux, macOS, or Unix) or command line (Windows), run the **aws lambda add-permissions** command to add a permission to the resource policy associated with your Lambda function:

```
aws lambda add-permission --cli-input-json file://AllowAccessfromMyDemoRepo.json
```

   This command returns the JSON of the policy statement you just added, similar to the following:

```
{
    "Statement": "{\"Condition\":{\"StringEquals\":{\"AWS:SourceAccount
\":\"111122223333\"},\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:codecommit:us-east-1:111122223333:MyDemoRepo\"}},\"Action
\":[\"lambda:InvokeFunction\"],\"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:MyCodeCommitFunction\",\"Effect\":\"Allow\",
\"Principal\":{\"Service\":\"codecommit.amazonaws.com\"},\"Sid\":\"1\"}"
}
```

   For more information about resource policies for Lambda functions, see AddPermission and The Pull/Push Event Models in the *AWS Lambda User Guide*.

4. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

5. In the **Dashboard** navigation pane, choose **Roles**, and in the list of roles, select *lambda_basic_execution*.
6. On the summary page for the role, choose the **Permissions** tab, and in **Inline Policies**, choose **Create Role Policy**.
7. On the **Set Permissions** page, choose **Policy Generator**, and then choose **Select**.
8. On the **Edit Permissions** page, do the following:

   - In **Effect**, choose **Allow**.
   - In **AWS Service**, choose **AWS CodeCommit**.
   - In **Actions**, select **GetRepository**.
   - In **Amazon Resource Name (ARN)**, enter the ARN for the repository (for example, `arn:aws:codecommit:us-east-1:`*111122223333*`:`*MyDemoRepo*).

   Choose **Add Statement**, and then choose **Next Step**.
9. On the **Review Policy** page, choose **Apply Policy**.

   Your policy statement should look similar to the following example:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt11111111",
            "Effect": "Allow",
            "Action": [
                "codecommit:GetRepository"
            ],
            "Resource": [
                "arn:aws-eusc:codecommit:eusc-de-
east-1:111122223333:MyDemoRepo"
            ]
        }
    ]
}
```

Create a trigger for the Lambda function in a CodeCommit repository (console)

After you have created the Lambda function, you can create a trigger in CodeCommit that runs the function in response to the repository events you specify.

> **Note**
>
> Before you can successfully test or run the trigger for the example, you must configure the policies that allow CodeCommit to invoke the function and the Lambda function to get information about the repository. For more information, see To allow CodeCommit to run a Lambda function.

To create a trigger for a Lambda function

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to create triggers for repository events.
3. In the navigation pane for the repository, choose **Settings**, and then choose **Triggers**.
4. Choose **Create trigger**.
5. In **Create trigger**, do the following:

   - In **Trigger name**, enter a name for the trigger (for example, `MyLambdaFunctionTrigger`).
   - In **Events**, choose the repository events that trigger the Lambda function.

     If you choose **All repository events**, you cannot choose any other events. If you want to choose a subset of events, clear **All repository events**, and then choose the events you want from the list. For example, if you want the trigger to run only when a user creates a tag or a branch in the CodeCommit repository, remove **All repository events**, and then choose **Create branch or tag**.
   - If you want the trigger to apply to all branches of the repository, in **Branches**, leave the selection blank, because this default option applies the trigger to all branches automatically. If you want this trigger to apply to specific branches only, choose up to 10 branch names from the list of repository branches.
   - In **Choose the service to use**, choose **AWS Lambda**.

- In **Lambda function**, choose the function name from the list, or enter the ARN for the function.
- (Optional) In **Custom data**, enter information you want included in the Lambda function (for example, the name of the IRC channel used by developers to discuss development in the repository). This field is a string. It cannot be used to pass any dynamic parameters.

6. (Optional) Choose **Test trigger**. This option attempts to invoke the function with sample data about your repository, including the most recent commit ID for the repository. (If no commit history exists, sample values consisting of zeroes are generated instead.) This helps you confirm that you have correctly configured access between CodeCommit and the Lambda function.
7. Choose **Create trigger** to finish creating the trigger.
8. To verify the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the **Monitoring** tab for that function in the Lambda console.

Create a trigger to a Lambda function for a CodeCommit repository (AWS CLI)

You can also use the command line to create a trigger for a Lambda function in response to CodeCommit repository events, such as when someone pushes a commit to your repository.

To create a trigger for an Lambda function

1. Open a plain-text editor and create a JSON file that specifies:

   - The Lambda function name.
   - The repository and branches you want to monitor with this trigger. (If you do not specify any branches, the trigger applies to all branches in the repository.)
   - The events that activate this trigger.

   Save the file.

   For example, if you want to create a trigger for a repository named *MyDemoRepo* that publishes all repository events to a Lambda function named *MyCodeCommitFunction* for two branches, *main* and *preprod*:

   ```
   {
   ```

```
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "name": "MyLambdaFunctionTrigger",
            "destinationArn": "arn:aws:lambda:us-
east-1:111122223333:function:MyCodeCommitFunction",
            "customData": "",
            "branches": [
                "main", "preprod"
            ],
            "events": [
                "all"
            ]
        }
    ]
}
```

There must be a trigger block in the JSON for each trigger for a repository. To create more than one trigger for a repository, include additional blocks in the JSON. Remember that all triggers created in this file are for the specified repository. You cannot create triggers for multiple repositories in a single JSON file. For example, if you wanted to create two triggers for a repository, you can create a JSON file with two trigger blocks. In the following example, no branches are specified in the second trigger block, so that trigger applies to all branches:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "name": "MyLambdaFunctionTrigger",
            "destinationArn": "arn:aws:lambda:us-
east-1:111122223333:function:MyCodeCommitFunction",
            "customData": "",
            "branches": [
                "main", "preprod"
            ],
            "events": [
                "all"
            ]
        },
        {
            "name": "MyOtherLambdaFunctionTrigger",
```

```
            "destinationArn": "arn:aws:lambda:us-
east-1:111122223333:function:MyOtherCodeCommitFunction",
            "customData": "",
            "branches": [],
            "events": [
                "updateReference", "deleteReference"
            ]
        }
    ]
}
```

You can create triggers for events you specify, such as when a commit is pushed to a repository. Event types include:

- `all` for all events in the specified repository and branches.
- `updateReference` for when commits are pushed to the specified repository and branches.
- `createReference` for when a new branch or tag is created in the specified repository.
- `deleteReference` for when a branch or tag is deleted in the specified repository.

> **Note**
>
> You can use more than one event type in a trigger. However, if you specify `all`, you cannot specify other events.

To see the full list of valid event types, at the terminal or command prompt, enter **aws codecommit put-repository-triggers help**.

In addition, you can include a string in `customData` (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters. This string is appended as an attribute to the CodeCommit JSON returned in response to the trigger.

2. (Optional) At a terminal or command prompt, run the **test-repository-triggers** command. For example, the following is used to test that the JSON file named *`trigger.json`* is valid and that CodeCommit can trigger the Lambda function. This test uses sample data to trigger the function if no real data is available.

```
aws codecommit test-repository-triggers --cli-input-json file://trigger.json
```

If successful, this command returns information similar to the following:

```
{
    "successfulExecutions": [
        "MyLambdaFunctionTrigger"
    ],
    "failedExecutions": []
}
```

3. At a terminal or command prompt, run the **put-repository-triggers** command to create the trigger in CodeCommit. For example, to use a JSON file named *trigger.json* to create the trigger:

   ```
   aws codecommit put-repository-triggers --cli-input-json
   file://trigger.json
   ```

   This command returns a configuration ID, similar to the following:

   ```
   {
       "configurationId": "0123456-I-AM-AN-EXAMPLE"
   }
   ```

4. To view the configuration of the trigger, run the **get-repository-triggers** command, specifying the name of the repository:

   ```
   aws codecommit get-repository-triggers --repository-name MyDemoRepo
   ```

   This command returns the structure of all triggers configured for the repository, similar to the following:

   ```
   {
       "configurationId": "0123456-I-AM-AN-EXAMPLE",
       "triggers": [
           {
               "events": [
                   "all"
               ],
   ```

```
            "destinationArn": "arn:aws:lambda:us-
east-1:111122223333:MyCodeCommitFunction",
            "branches": [
                "main",
                "preprod"
            ],
            "name": "MyLambdaFunctionTrigger",
            "customData": "Project ID 12345"
        }
    ]
}
```

5. To test the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the **Monitoring** tab for that function in the Lambda console.

Edit triggers for an AWS CodeCommit repository

You can edit the triggers that have been created for a CodeCommit repository. You can change the events and branches for the trigger, the action taken in response to the event, and other settings.

Topics

- Edit a trigger for a repository (console)
- Edit a trigger for a repository (AWS CLI)

Edit a trigger for a repository (console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to edit a trigger for repository events.
3. In the navigation pane for the repository, choose **Settings**, and then choose **Triggers**.
4. From the list of triggers for the repository, choose the trigger you want to edit, and then choose **Edit**.
5. Make the changes you want to the trigger, and then choose **Save**.

Edit a trigger for a repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the **get-repository-triggers** command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named *MyTriggers.json* with the structure of all of the triggers configured for a repository named *MyDemoRepo*:

   ```
   aws codecommit get-repository-triggers --repository-name MyDemoRepo
    >MyTriggers.json
   ```

   This command returns nothing, but a file named *MyTriggers.json* is created in the directory where you ran the command.

2. Edit the JSON file in a plain-text editor and make changes to the trigger block of the trigger you want to edit. Replace the `configurationId` pair with a `repositoryName` pair. Save the file.

   For example, if you want to edit a trigger named *MyFirstTrigger* in the repository named *MyDemoRepo* so that it applies to all branches, replace `configurationId` with `repositoryName`, and remove the specified `main` and `preprod` branches in *red italic text*. By default, if no branches are specified, the trigger applies to all branches in the repository:

   ```
   {
       "repositoryName": "MyDemoRepo",
       "triggers": [
           {
               "destinationArn": "arn:aws:sns:us-
   east-2:111122223333:MyCodeCommitTopic",
               "branches": [
                   "main",
                   "preprod"
               ],
               "name": "MyFirstTrigger",
               "customData": "",
               "events": [
                   "all"
               ]
           }
       ]
   ```

```
}
```

3. At the terminal or command line, run the **put-repository-triggers** command. This updates all triggers for the repository, including the changes you made to the *MyFirstTrigger* trigger:

```
aws codecommit put-repository-triggers --repository-name MyDemoRepo
 file://MyTriggers.json
```

This command returns a configuration ID, similar to the following:

```
{
    "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

Test triggers for an AWS CodeCommit repository

You can test the triggers that have been created for a CodeCommit repository. Testing involves running the trigger with sample data from your repository, including the most recent commit ID. If no commit history exists for the repository, sample values consisting of zeroes are generated instead. Testing triggers helps you confirm you have correctly configured access between CodeCommit and the target of the trigger, whether that is an AWS Lambda function or an Amazon Simple Notification Service notification.

Topics

- Test a trigger for a repository (console)
- Test a trigger for a repository (AWS CLI)

Test a trigger for a repository (console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to test a trigger for repository events.
3. In the navigation pane for the repository, choose **Settings**, and then choose **Triggers**.
4. Choose the trigger you want to test, and then choose **Test trigger**. You should see a success or failure message. If successful, you should also see a corresponding action response from the Lambda function or the Amazon SNS topic.

Test a trigger for a repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the **get-repository-triggers** command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named *TestTrigger.json* with the structure of all of the triggers configured for a repository named MyDemoRepo:

   ```
   aws codecommit get-repository-triggers --repository-name MyDemoRepo
    >TestTrigger.json
   ```

   This command creates a file named *TestTriggers.json* in the directory where you ran the command.

2. Edit the JSON file in a plain-text editor and make the changes to the trigger statement. Replace the `configurationId` pair with a `repositoryName` pair. Save the file.

   For example, if you want to test a trigger named *MyFirstTrigger* in the repository named *MyDemoRepo* so that it applies to all branches, replace the `configurationId` with `repositoryName` and then save a file that looks similar to the following as *TestTrigger.json*:

   ```
   {
       "repositoryName": "MyDemoRepo",
       "triggers": [
           {
               "destinationArn": "arn:aws:sns:us-
   east-2:111122223333:MyCodeCommitTopic",
               "branches": [
                   "main",
                   "preprod"
               ],
               "name": "MyFirstTrigger",
               "customData": "",
               "events": [
                   "all"
               ]
           }
       ]
   }
   ```

3. At the terminal or command line, run the **test-repository-triggers** command. This updates all triggers for the repository, including the changes you made to the *MyFirstTrigger* trigger:

```
aws codecommit test-repository-triggers --cli-input-json file://TestTrigger.json
```

This command returns a response similar to the following:

```
{
    "successfulExecutions": [
        "MyFirstTrigger"
    ],
    "failedExecutions": []
}
```

Delete triggers from an AWS CodeCommit repository

You might want to delete triggers if they are no longer being used. You cannot undo the deletion of a trigger, but you can create one again.

> **Note**
>
> If you configured one or more triggers for your repository, deleting the repository does not delete the Amazon SNS topics or Lambda functions you configured as the targets of those triggers. Be sure to delete those resources, too, if they are no longer needed.

Topics

- Delete a trigger from a repository (console)
- Delete a trigger from a repository (AWS CLI)

Delete a trigger from a repository (console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to delete triggers for repository events.
3. In the navigation pane for the repository, choose **Settings**. In **Settings**, choose **Triggers**.

4. Choose the trigger you want to delete from the list of triggers, and then choose **Delete**.
5. In the dialog box, type **delete** to confirm.

Delete a trigger from a repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the **get-repository-triggers** command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named *MyTriggers.json* with the structure of all of the triggers configured for a repository named MyDemoRepo:

```
aws codecommit get-repository-triggers --repository-name MyDemoRepo
 >MyTriggers.json
```

   This command creates a file named *MyTriggers.json* in the directory where you ran the command.

2. Edit the JSON file in a plain-text editor and remove the trigger block for the trigger you want to delete. Replace the `configurationId` pair with a `repositoryName` pair. Save the file.

   For example, if you want to remove a trigger named *MyFirstTrigger* from the repository named *MyDemoRepo*, you would replace `configurationId` with `repositoryName`, and remove the statement in *red italic text*:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "destinationArn": "arn:aws:sns:us-
east-2:111122223333:MyCodeCommitTopic",
            "branches": [
                "main",
                "preprod"
            ],
            "name": "MyFirstTrigger",
            "customData": "",
            "events": [
                "all"
            ]
        },
        {
```

```
            "destinationArn": "arn:aws:lambda:us-
east-2:111122223333:function:MyCodeCommitJSFunction",
            "branches": [],
            "name": "MyLambdaTrigger",
            "events": [
                "all"
            ]
        }
    ]
}
```

3. At the terminal or command line, run the **put-repository-triggers** command. This updates the triggers for the repository and deletes the *MyFirstTrigger* trigger:

```
aws codecommit put-repository-triggers --repository-name MyDemoRepo
 file://MyTriggers.json
```

This command returns a configuration ID, similar to the following:

```
{
    "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

Note

To delete all triggers for a repository named *MyDemoRepo*, your JSON file would look similar to this:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": []
}
```

Amazon CodeGuru Reviewer and AWS CodeCommit

> **Important**
>
> As of November 7, 2025, Amazon CodeGuru Reviewer is in maintenance mode. You cannot create new repository associations with CodeGuru Reviewer. Existing associations continue to function, but AWS will not enhance or add functionality to the service.

If you have an existing CodeGuru Reviewer association with a CodeCommit repository, it will continue to analyze pull requests as before. To disassociate a repository from CodeGuru Reviewer, choose **Settings**, choose **Amazon CodeGuru Reviewer**, and then choose **Disassociate repository**.

For code review and security scanning capabilities, consider the following alternatives:

- **Amazon Q Developer** — Provides code review including static analysis, secrets detection, and dependency scanning. Works in your IDE and with GitHub and GitLab repositories. For more information, see Reviewing code with Amazon Q Developer.
- **Amazon Inspector** — Automatically discovers and scans code repositories for software vulnerabilities. For more information, see Amazon Inspector Code Security.

For more information about the CodeGuru Reviewer availability change, see Amazon CodeGuru Reviewer availability change.

View CodeCommit repository details

You can use the AWS CodeCommit console, AWS CLI, or Git from a local repo connected to the CodeCommit repository to view information about available repositories.

Before you follow these instructions, complete the steps in Setting up .

Topics

- View repository details (console)
- View CodeCommit repository details (Git)
- View CodeCommit repository details (AWS CLI)

View repository details (console)

Use the AWS CodeCommit console to quickly view all repositories created with your Amazon Web Services account.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, view the details about the repositories in the AWS Region where you are signed in. Use the Region selector to choose a different AWS Region to view repositories in that Region.
3. Choose the name of the repository for which you want to view more details, and then do one of the following:
   - To view the URL for cloning the repository, choose **Clone URL**, and then choose the protocol you want to use when cloning the repository. This copies the clone URL. To review it, paste it into a plain-text editor.
   - To view configurable options for the repository as well as details such as the repository ARN and repository ID, in the navigation pane, choose **Settings**.

> **Note**
>
> If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with user preferences.

View CodeCommit repository details (Git)

To use Git from a local repo to view details about CodeCommit repositories, run the **git remote show** command.

Before you perform these steps, connect the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. Run the **git remote show *`remote-name`*** command, where *`remote-name`* is the alias of the CodeCommit repository (by default, `origin`).

> **Tip**
>
> To get a list of CodeCommit repository names and their URLs, run the **git remote -v** command.

For example, to view details about the CodeCommit repository with the alias `origin`:

```
git remote show origin
```

2. For HTTPS:

```
* remote origin
  Fetch URL: https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  Push  URL: https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  HEAD branch: (unknown)
  Remote branches:
    MyNewBranch tracked
    main tracked
  Local ref configured for 'git pull':
    MyNewBranch merges with remote MyNewBranch (up to date)
  Local refs configured for 'git push':
    MyNewBranch pushes to MyNewBranch (up to date)
    main pushes to main (up to date)
```

For SSH:

```
* remote origin
  Fetch URL: ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  Push  URL: ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  HEAD branch: (unknown)
  Remote branches:
    MyNewBranch tracked
    main tracked
  Local ref configured for 'git pull':
    MyNewBranch merges with remote MyNewBranch (up to date)
  Local refs configured for 'git push':
    MyNewBranch pushes to MyNewBranch (up to date)
    main pushes to main (up to date)
```

> **Tip**
>
> To look up the SSH key ID for your IAM user, open the IAM console and expand **Security Credentials** on the IAM user details page. The SSH key ID can be found in **SSH Keys for AWS CodeCommit**.

For more options, see your Git documentation.

View CodeCommit repository details (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to view repository details, run the following commands:

- To view a list of CodeCommit repository names and their corresponding IDs, run list-repositories.
- To view information about a single CodeCommit repository, run get-repository.
- To view information about multiple repositories in CodeCommit, run batch-get-repositories.

To view a list of CodeCommit repositories

1. Run the **list-repositories** command:

   ```
   aws codecommit list-repositories
   ```

   You can use the optional `--sort-by` or `--order` options to change the order of returned information.

2. If successful, this command outputs a `repositories` object that contains the names and IDs of all repositories in CodeCommit associated with the Amazon Web Services account.

   Here is some example output based on the preceding command:

   ```
   {
       "repositories": [
           {
               "repositoryName": "MyDemoRepo",
   ```

```
            "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE"
        },
        {
            "repositoryName": "MyOtherDemoRepo",
            "repositoryId": "cfc29ac4-b0cb-44dc-9990-f6f51EXAMPLE"
        }
    ]
}
```

To view details about a single CodeCommit repository

1. Run the **get-repository** command, specifying the name of the CodeCommit repository with the `--repository-name` option.

   > **Tip**
   >
   > To get the name of the CodeCommit repository, run the list-repositories command.

   For example, to view details about a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit get-repository --repository-name MyDemoRepo
   ```

2. If successful, this command outputs a `repositoryMetadata` object with the following information:

   - The repository's name (`repositoryName`).
   - The repository's description (`repositoryDescription`).
   - The repository's unique, system-generated ID (`repositoryId`).
   - The ID of the Amazon Web Services account associated with the repository (`accountId`).

   Here is some example output, based on the preceding example command:

   ```
   {
           "repositoryMetadata": {
               "creationDate": 1429203623.625,
               "defaultBranch": "main",
               "repositoryName": "MyDemoRepo",
   ```

```
            "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo",
            "lastModifiedDate": 1430783812.0869999,
            "repositoryDescription": "My demonstration repository",
            "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo",
            "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
            "Arn": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
            "accountId": "111111111111"
        }
}
```

To view details about multiple CodeCommit repositories

1. Run the **batch-get-repositories** command with the `--repository-names` option. Add a space between each CodeCommit repository name.

   > **Tip**
   >
   > To get the names of the repositories in CodeCommit, run the list-repositories command.

   For example, to view details about two CodeCommit repositories named `MyDemoRepo` and `MyOtherDemoRepo`:

   ```
   aws codecommit batch-get-repositories --repository-names MyDemoRepo MyOtherDemoRepo
   ```

2. If successful, this command outputs an object with the following information:

   - A list of any CodeCommit repositories that could not be found (`repositoriesNotFound`).
   - A list of CodeCommit repositories (`repositories`). Each CodeCommit repository name is followed by:
     - The repository's description (`repositoryDescription`).
     - The repository's unique, system-generated ID (`repositoryId`).
     - The ID of the Amazon Web Services account associated with the repository (`accountId`).

   Here is some example output, based on the preceding example command:

```
{
        "repositoriesNotFound": [],
        "repositories": [
             {
                "creationDate": 1429203623.625,
                "defaultBranch": "main",
                "repositoryName": "MyDemoRepo",
                "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo",
                "lastModifiedDate": 1430783812.0869999,
                "repositoryDescription": "My demonstration repository",
                "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo",
                "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
                "Arn": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
                "accountId": "111111111111"
            },
            {
                "creationDate": 1429203623.627,
                "defaultBranch": "main",
                "repositoryName": "MyOtherDemoRepo",
                "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyOtherDemoRepo",
                "lastModifiedDate": 1430783812.0889999,
                "repositoryDescription": "My other demonstration repository",
                "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/
repos/MyOtherDemoRepo",
                "repositoryId": "cfc29ac4-b0cb-44dc-9990-f6f51EXAMPLE",
                "Arn": "arn:aws:codecommit:us-east-2:111111111111:MyOtherDemoRepo",
                "accountId": "111111111111"
            }
        ],
        "repositoriesNotFound": []
    }
```


Change AWS CodeCommit repository settings

You can use the AWS CLI and the AWS CodeCommit console to change the settings of an CodeCommit repository, such as its description or name.

> **Important**
>
> Changing a repository's name may break any local repos that use the old name in their remote URL. Run the **git remote set-url** command to update the remote URL to use the new repository's name.

Topics

- Change repository settings (console)
- Change AWS CodeCommit repository settings (AWS CLI)

Change repository settings (console)

To use the AWS CodeCommit console to change a CodeCommit repository's settings in AWS CodeCommit, follow these steps.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to change settings.
3. In the navigation pane, choose **Settings**.
4. To change the name of the repository, in **Repository name**, enter a new name in the **Name** text box and choose **Save**. When prompted, verify your choice.

   > **Important**
   >
   > Changing the name of the AWS CodeCommit repository will change the SSH and HTTPS URLs that users need to connect to the repository. Users will not be able to connect to this repository until they update their connection settings. Also, because the repository's ARN will change, changing the repository name will invalidate any IAM user policies that rely on this repository's ARN.
   >
   > To connect to the repository after the name is changed, each user must use the **git remote set-url** command and specify the new URL to use. For example, if you changed the name of the repository from MyDemoRepo to MyRenamedDemoRepo, users who use HTTPS to connect to the repository would run the following Git command:

```
git remote set-url origin https://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyRenamedDemoRepo
```

Users who use SSH to connect to the repository would run the following Git command:

```
git remote set-url origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyRenamedDemoRepo
```

For more options, see your Git documentation.

5. To change the repository's description, modify the text in the **Description** text box, and then choose **Save**.

> **Note**
>
> The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

6. To change the default branch, in **Default branch**, choose the branch drop-down list and choose a different branch. Choose **Save**.
7. To change the AWS KMS encryption key used to encrypt and decrypt data in the repository, in **Repository encryption key**, choose either **AWS managed key** or **Customer managed key** to specify the type of key to use. If choosing a customer managed key, enter the ARN of the key. Choose **Save**.
8. To delete the repository, choose **Delete repository**. In the box next to **Type the name of the repository to confirm deletion**, enter **`delete`**, and then choose **Delete**.

> **Important**
>
> After you delete this repository in AWS CodeCommit, you will no longer be able to clone it to any local repo or shared repo. You will also no longer be able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

Change AWS CodeCommit repository settings (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use AWS CLI to change a CodeCommit repository's settings in AWS CodeCommit, run one or more of the following commands:

- update-repository-description to change the description of an CodeCommit repository.
- update-repository-name to change the name of an CodeCommit repository.

To change a CodeCommit repository's description

1. Run the **update-repository-description** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).

     > **Tip**
     >
     > To get the name of the CodeCommit repository, run the **list-repositories** command.

   - The new repository description (with the `--repository-description` option).

     > **Note**
     >
     > The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

   For example, to change the description for the CodeCommit repository named `MyDemoRepo` to `This description was changed`:

```
aws codecommit update-repository-description --repository-name MyDemoRepo --
repository-description "This description was changed"
```

This command produces output only if there are errors.

2. To verify the changed description, run the **get-repository** command, specifying the name of the CodeCommit repository whose description you changed with the `--repository-name` option.

   The output of the command shows the changed text in `repositoryDescription`.

To change a CodeCommit repository's name

1. Run the **update-repository-name** command, specifying:

   - The current name of the CodeCommit repository (with the `--old-name` option).

     > **Tip**
     >
     > To get the CodeCommit repository's name, run the list-repositories command.

   - The new name of the CodeCommit repository (with the `--new-name` option).

   For example, to change the repository named `MyDemoRepo` to `MyRenamedDemoRepo`:

   ```
   aws codecommit update-repository-name --old-name MyDemoRepo --new-name
    MyRenamedDemoRepo
   ```

   This command produces output only if there are errors.

   > **Important**
   >
   > Changing the name of the AWS CodeCommit repository changes the SSH and HTTPS URLs that users need to connect to the repository. Users cannot connect to this repository until they update their connection settings. Also, because the repository's ARN changes, changing the repository name invalidates any IAM user policies that rely on this repository's ARN.

2. To verify the changed name, run the **list-repositories** command and review the list of repository names.

Synchronize changes between a local repo and an AWS CodeCommit repository

You use Git to synchronize changes between a local repo and the CodeCommit repository connected to the local repo.

To push changes from the local repo to the CodeCommit repository, run **git push *remote-name branch-name***.

To pull changes to the local repo from the CodeCommit repository, run **git pull *remote-name branch-name***.

For both pushing and pulling, *remote-name* is the nickname the local repo uses for the CodeCommit repository. *branch-name* is the name of the branch on the CodeCommit repository to push to or pull from.

> **Tip**
>
> To get the nickname the local repo uses for the CodeCommit repository, run **git remote**. To get a list of branch names, run **git branch**. An asterisk (*) appears next to the name of the current branch. (You can also run **git status** to show the current branch name.)

> **Note**
>
> If you cloned the repository, from the perspective of the local repo, *remote-name* is not the name of the CodeCommit repository. When you clone a repository, *remote-name* is set automatically to `origin`.

For example, to push changes from the local repo to the `main` branch in the CodeCommit repository with the nickname `origin`:

```
git push origin main
```

Similarly, to pull changes to the local repo from the `main` branch in the CodeCommit repository with the nickname `origin`:

```
git pull origin main
```

> **Tip**
>
> If you add the `-u` option to **git push**, you set upstream tracking information. For example, if you run **git push -u origin main**), in the future you can run **git push** and **git pull** without *`remote-name branch-name`*. To get upstream tracking information, run **git remote show *`remote-name`*** (for example, **git remote show origin**).

For more options, see your Git documentation.

Push commits to an additional Git repository

You can configure your local repo to push changes to two remote repositories. For example, you might want to continue using your existing Git repository solution while you try out AWS CodeCommit. Follow these basic steps to push changes in your local repo to CodeCommit and a separate Git repository.

> **Tip**
>
> If you do not have a Git repository, you can create an empty one on a service other than CodeCommit and then migrate your CodeCommit repository to it. You should follow steps similar to the ones in Migrate to CodeCommit.

1. From the command prompt or terminal, switch to your local repo directory and run the **git remote -v** command. You should see output similar to the following:

   For HTTPS:

   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

   For SSH:

```
origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

2. Run the **git remote set-url --add --push origin *git-repository-name*** command where *git-repository-name* is the URL and name of the Git repository where you want to host your code. This changes the push destination of `origin` to that Git repository.

   > **Note**
   >
   > **git remote set-url --add --push** overrides the default URL for pushes, so you must run this command twice, as demonstrated in later steps.

   For example, the following command changes the push of origin to *some-URL*/MyDestinationRepo:

   ```
   git remote set-url --add --push origin some-URL/MyDestinationRepo
   ```

   This command returns nothing.

   > **Tip**
   >
   > If you are pushing to a Git repository that requires credentials, make sure you configure those credentials in a credential helper or in the configuration of the *some-URL* string. Otherwise, your pushes to that repository fail.

3. Run the **git remote -v** command again, which should create output similar to the following:

   For HTTPS:

   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  some-URL/MyDestinationRepo (push)
   ```

   For SSH:

   ```
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  some-URL/MyDestinationRepo (push)
   ```

4. Now add the CodeCommit repository. Run **git remote set-url --add --push origin** again, this time with the URL and repository name of your CodeCommit repository.

   For example, the following command adds the push of **origin** to https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo:

   For HTTPS:

   ```
   git remote set-url --add --push origin https://git-codecommit.us-
   east-2.amazonaws.com/v1/repos/MyDemoRepo
   ```

   For SSH:

   ```
   git remote set-url --add --push origin ssh://git-codecommit.us-
   east-2.amazonaws.com/v1/repos/MyDemoRepo
   ```

   This command returns nothing.

5. Run the **git remote -v** command again, which should create output similar to the following:

   For HTTPS:

   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  some-URL/MyDestinationRepo (push)
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

   For SSH:

   ```
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  some-URL/MyDestinationRepo (push)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

   You now have two Git repositories as the destination for your pushes, but your pushes go to *some-URL*/MyDestinationRepo first. If the push to that repository fails, your commits are not pushed to either repository.

> **Tip**
>
> If the other repository requires credentials you want to enter manually, consider changing the order of the pushes so that you push to CodeCommit first. Run **git remote set-url --delete** to delete the repository that is pushed to first, and then run **git remote set-url --add** to add it again so that it becomes the second push destination in the list.
> For more options, see your Git documentation.

6. To verify you are now pushing to both remote repositories, use a text editor to create the following text file in your local repo:

```
bees.txt
-------
Bees are flying insects closely related to wasps and ants, and are known for their
 role in pollination and for producing honey and beeswax.
```

7. Run **git add** to stage the change in your local repo:

```
git add bees.txt
```

8. Run **git commit** to commit the change in your local repo:

```
git commit -m "Added bees.txt"
```

9. To push the commit from the local repo to your remote repositories, run **git push -u *remote-name branch-name*** where *remote-name* is the nickname the local repo uses for the remote repositories and *branch-name* is the name of the branch to push to the repository.

> **Tip**
>
> You only have to use the `-u` option the first time you push. Then the upstream tracking information is set.

For example, running **git push -u origin main** would show the push went to both remote repositories in the expected branches, with output similar to the following:

For HTTPS:

```
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To some-URL/MyDestinationRepo
   a5ba4ed..250f6c3  main -> main
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
To https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
   a5ba4ed..250f6c3  main -> main
```

For SSH:

```
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To some-URL/MyDestinationRepo
   a5ba4ed..250f6c3  main -> main
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
To ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
   a5ba4ed..250f6c3  main -> main
```

For more options, see your Git documentation.

Configure cross-account access to an AWS CodeCommit repository using roles

You can configure access to CodeCommit repositories for IAM users and groups in another AWS account. This is often referred to as *cross-account access*. This section provides examples and step-by-step instructions for configuring cross-account access for a repository named *MySharedDemoRepo* in the US East (Ohio) Region in an AWS account (referred to as AccountA) to IAM users who belong to an IAM group named *DevelopersWithCrossAccountRepositoryAccess* in another AWS account (referred to as AccountB).

This section is divided into three parts:

- Actions for the Administrator in AccountA.
- Actions for the Administrator in AccountB.
- Actions for the repository user in AccountB.

To configure cross-account access:

- The administrator in AccountA signs in as an IAM user with the permissions required to create and manage repositories in CodeCommit and create roles in IAM. If you are using managed policies, apply IAMFullAccess and AWSCodeCommitFullAccess to this IAM user.

  The example account ID for AccountA is *111122223333*.
- The administrator in AccountB signs in as an IAM user with the permissions required to create and manage IAM users and groups, and to configure policies for users and groups. If you are using managed policies, apply IAMFullAccess to this IAM user.

  The example account ID for AccountB is *888888888888*.
- The repository user in AccountB, to emulate the activities of a developer, signs in as an IAM user who is a member of the IAM group created to allow access to the CodeCommit repository in AccountA. This account must be configured with:
  - AWS Management Console access.
  - An access key and secret key to use when connecting to AWS resources and the ARN of the role to assume when accessing repositories in AccountA.

- The **git-remote-codecommit** utility on the local computer where the repository is cloned. This utility requires Python and its installer, pip. You can download the utility from **git-remote-codecommit** on the Python Package Index website.

For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit and IAM users.

Topics

- Cross-account repository access: Actions for the administrator in AccountA
- Cross-account repository access: Actions for the administrator in AccountB
- Cross-account repository access: Actions for the repository user in AccountB

Cross-account repository access: Actions for the administrator in AccountA

To allow users or groups in AccountB to access a repository in AccountA, an AccountA administrator must:

- Create a policy in AccountA that grants access to the repository.
- Create a role in AccountA that can be assumed by IAM users and groups in AccountB.
- Attach the policy to the role.

The following sections provide steps and examples.

Topics

- Step 1: Create a policy for repository access in AccountA
- Step 2: Create a role for repository access in AccountA

Step 1: Create a policy for repository access in AccountA

You can create a policy in AccountA that grants access to the repository in AccountA to users in AccountB. Depending on the level of access you want to allow, do one of the following:

- Configure the policy to allow AccountB users access to a specific repository, but do not allow them to view a list of all repositories in AccountA.

- Configure additional access to allow AccountB users to choose the repository from a list of all repositories in AccountA.

To create a policy for repository access

1. Sign in to the AWS Management Console as an IAM user with permissions to create policies in AccountA.
2. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
3. In the navigation pane, choose **Policies**.
4. Choose **Create policy**.
5. Choose the **JSON** tab, and paste the following JSON policy document into the JSON text box. Replace *us-east-2* with the AWS Region for the repository, *111122223333* with the account ID for AccountA, and *MySharedDemoRepo* with the name for your CodeCommit repository in AccountA:

   JSON

```
{
"Version":"2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "codecommit:BatchGet*",
            "codecommit:Create*",
            "codecommit:DeleteBranch",
            "codecommit:Get*",
            "codecommit:List*",
            "codecommit:Describe*",
            "codecommit:Put*",
            "codecommit:Post*",
            "codecommit:Merge*",
            "codecommit:Test*",
            "codecommit:Update*",
            "codecommit:GitPull",
            "codecommit:GitPush"
        ],
        "Resource": [
            "arn:aws-eusc:codecommit:us-east-2:111122223333:MySharedDemoRepo"
```

```
        ]
    }
]
}
```

If you want users who assume this role to be able to view a list of repositories on the CodeCommit console home page, add an additional statement to the policy, as follows:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGet*",
                "codecommit:Create*",
                "codecommit:DeleteBranch",
                "codecommit:Get*",
                "codecommit:List*",
                "codecommit:Describe*",
                "codecommit:Put*",
                "codecommit:Post*",
                "codecommit:Merge*",
                "codecommit:Test*",
                "codecommit:Update*",
                "codecommit:GitPull",
                "codecommit:GitPush"
            ],
            "Resource": [
                "arn:aws-eusc:codecommit:us-
east-2:111122223333:MySharedDemoRepo"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "codecommit:ListRepositories",
            "Resource": "*"
        }
    ]
```

```
}
```

This access makes it easier for users who assume this role with this policy to find the repository to which they have access. They can choose the name of the repository from the list and be directed to the home page of the shared repository (Code). Users cannot access any of the other repositories they see in the list, but they can view the repositories in AccountA on the **Dashboard** page.

If you do not want to allow users who assume the role to be able to view a list of all repositories in AccountA, use the first policy example, but make sure that you send those users a direct link to the home page of the shared repository in the CodeCommit console.

6. Choose **Review policy**. The policy validator reports syntax errors (for example, if you forget to replace the example Amazon Web Services account ID and repository name with your Amazon Web Services account ID and repository name).
7. On the **Review policy** page, enter a name for the policy (for example, *CrossAccountAccessForMySharedDemoRepo*). You can also provide an optional description for this policy. Choose **Create policy**.

Step 2: Create a role for repository access in AccountA

After you have configured a policy, create a role that IAM users and groups in AccountB can assume, and attach the policy to that role.

To create a role for repository access

1. In the IAM console, choose **Roles**.
2. Choose **Create role**.
3. Choose **Another Amazon Web Services account**.
4. In **Account ID**, enter the Amazon Web Services account ID for AccountB (for example, *888888888888*). Choose **Next: Permissions**.
5. In **Attach permissions policies**, select the policy you created in the previous procedure (*CrossAccountAccessForMySharedDemoRepo*). Choose **Next: Review**.
6. In **Role name**, enter a name for the role (for example, *MyCrossAccountRepositoryContributorRole*). You can also enter an optional description to help others understand the purpose of the role.

7. Choose **Create role**.
8. Open the role you just created, and copy the role ARN (for example, `arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole`). You need to provide this ARN to the AccountB administrator.

Cross-account repository access: Actions for the administrator in AccountB

To allow users or groups in AccountB to access a repository in AccountA, the AccountB administrator must create a group in AccountB. This group must be configured with a policy that allows group members to assume the role created by the AccountA administrator.

The following sections provide steps and examples.

Topics

- Step 1: Create an IAM group for repository access for AccountB users
- Step 2: Create a policy and add users to the IAM group

Step 1: Create an IAM group for repository access for AccountB users

The simplest way to manage which IAM users in AccountB can access the AccountA repository is to create an IAM group in AccountB that has permission to assume the role in AccountA, and then add the IAM users to that group.

To create a group for cross-account repository access

1. Sign in to the AWS Management Console as an IAM user with the permissions required to create IAM groups and policies and manage IAM users in AccountB.
2. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
3. In the IAM console, choose **Groups**.
4. Choose **Create New Group**.
5. In **Group Name**, enter a name for the group (for example, *DevelopersWithCrossAccountRepositoryAccess*). Choose **Next Step**.
6. In **Attach Policy**, choose **Next Step**. You create the cross-account policy in the next procedure. Finish creating the group.

Step 2: Create a policy and add users to the IAM group

Now that you have a group, create the policy that allows members of this group to assume the role that gives them access to the repository in AccountA. Then add to the group the IAM users in AccountB that you want to allow access in AccountA.

To create a policy for the group and add users to it

1. In the IAM console, choose **Groups**, and then choose the name of the group you just created (for example, *DevelopersWithCrossAccountRepositoryAccess*).
2. Choose the **Permissions** tab. Expand **Inline Policies**, and then choose the link to create an inline policy. (If you are configuring a group that already has an inline policy, choose **Create Group Policy**.)
3. Choose **Custom Policy**, and then choose **Select**.
4. In **Policy Name**, enter a name for the policy (for example, *AccessPolicyForSharedRepository*).
5. In **Policy Document**, paste the following policy. In `Resource`, replace the ARN with the ARN of the policy created by the administrator in AccountA (for example, arn:aws:iam::*111122223333*:role/*MyCrossAccountRepositoryContributorRole*), and then choose **Apply Policy**. For more information about the policy created by the administrator in AccountA, see Step 1: Create a policy for repository access in AccountA.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": {
       "Effect": "Allow",
       "Action": "sts:AssumeRole",
       "Resource": "arn:aws-
   eusc:iam::111122223333:role/MyCrossAccountRepositoryContributorRole"
     }
   }
   ```

6. Choose the **Users** tab. Choose **Add Users to Group**, and then add the AccountB IAM users. For example, you might add an IAM user with the user name *Saanvi_Sarkar* to the group.

> **Note**
>
> Users in AccountB must have programmatic access, including an access key and secret key, to configure their local computers for access to the shared CodeCommit repository. If you are creating IAM users, be sure to save the access key and secret key. To ensure the security of your AWS account, the secret access key is accessible only at the time you create it.

Cross-account repository access: Actions for the repository user in AccountB

To access the repository in AccountA, users in the AccountB group must configure their local computers for repository access. The following sections provide steps and examples.

Topics

- Step 1: Configure the AWS CLI and Git for an AccountB user to access the repository in AccountA
- Step 2: Clone and access the CodeCommit repository in AccountA

Step 1: Configure the AWS CLI and Git for an AccountB user to access the repository in AccountA

You cannot use SSH keys or Git credentials to access repositories in another Amazon Web Services account. AccountB users must configure their computers to use either **git-remote-codecommit** (recommended) or the credential helper to access the shared CodeCommit repository in AccountA. However, you can continue to use SSH keys or Git credentials when accessing repositories in AccountB.

Follow these steps to configure access using **git-remote-codecommit**. If you have not already installed **git-remote-codecommit**, download it from **git-remote-codecommit**on the Python Package Index website.

To configure the AWS CLI and Git for cross-account access

1. Install the AWS CLI on the local computer. See instructions for your operating system in Installing the AWS CLI.

2. Install Git on the local computer. To install Git, we recommend websites such as Git Downloads or Git for Windows.

   > **Note**
   >
   > CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git. Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

3. From the terminal or command line, at the directory location where you want to clone the repository, run the **git config --local user.name** and **git config --local user.email** commands to set the user name and email for the commits you will make to the repository. For example:

   ```
   git config --local user.name "Saanvi Sarkar"
   git config --local user.email saanvi_sarkar@example.com
   ```

   These commands return nothing, but the email and user name you specify is associated with the commits you make to the repository in AccountA.

4. Run the **aws configure --profile** command to configure a default profile to use when connecting to resources in AccountB. When prompted, provide the access key and secret key for your IAM user.

   > **Note**
   >
   > If you have already installed the AWS CLI and configured a profile, you can skip this step.

   For example, run the following command to create a default AWS CLI profile that you use to access AWS resources in AccountB in US East (Ohio) (us-east-2):

   ```
   aws configure
   ```

   When prompted, provide the following information:

```
AWS Access Key ID [None]: Your-IAM-User-Access-Key
AWS Secret Access Key ID [None]: Your-IAM-User-Secret-Access-Key
Default region name ID [None]: us-east-2
Default output format [None]: json
```

5. Run the **aws configure --profile** command again to configure a named profile to use when connecting to the repository in AccountA. When prompted, provide the access key and secret key for your IAM user. For example, run the following command to create an AWS CLI profile named *MyCrossAccountAccessProfile* that you use to access a repository in AccountA in US East (Ohio) (us-east-2):

   ```
   aws configure --profile MyCrossAccountAccessProfile
   ```

   When prompted, provide the following information:

   ```
   AWS Access Key ID [None]: Your-IAM-User-Access-Key
   AWS Secret Access Key ID [None]: Your-IAM-User-Secret-Access-Key
   Default region name ID [None]: us-east-2
   Default output format [None]: json
   ```

6. In a plain-text editor, open the `config` file, also known as the AWS CLI configuration file. Depending on your operating system, this file might be located at `~/.aws/config` on Linux, macOS, or Unix, or at *drive*:\Users*USERNAME*\.aws\config on Windows.

7. In the file, find the entry that corresponds to the default profile you configured for access to repositories in AccountB. It should look similar to the following:

   ```
   [default]
   region = us-east-2
   output = json
   ```

   Add `account` to the profile configuration. Provide the AWS account ID of AccountB. For example:

   ```
   [default]
   account = 888888888888
   region = us-east-2
   output = json
   ```

8. In the file, find the entry that corresponds to the *MyCrossAccountAccessProfile* profile you just created. It should look similar to the following:

```
[profile MyCrossAccountAccessProfile]
region = us-east-2
output = json
```

Add `account`, `role_arn` and `source_profile` to the profile configuration. Provide the Amazon Web Services account ID of AccountA, the ARN of the role in AccountA that you assume to access the repository in the other account, and the name of your default AWS CLI profile in AccountB. For example:

```
[profile MyCrossAccountAccessProfile]
region = us-east-2
account = 111122223333
role_arn = arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole
source_profile = default
output = json
```

Save your changes, and close the plain-text editor.

Step 2: Clone and access the CodeCommit repository in AccountA

Run **git clone**, **git push**, and **git pull** to clone, push to, and pull from, the cross-account CodeCommit repository. You can also sign in to the AWS Management Console, switch roles, and use the CodeCommit console to interact with the repository in the other account.

> **Note**
>
> Depending on how the IAM role was configured, you might be able to view repositories on the default page for CodeCommit. If you cannot view the repositories, ask the repository administrator to email you a URL link to the **Code** page for the shared repository in the CodeCommit console. The URL is similar to the following:
>
> ```
> https://console.aws.amazon.com/codecommit/home?region=us-east-2#/
> repository/MySharedDemoRepo/browse/HEAD/--/
> ```

To clone the cross-account repository to your local computer

1. At the command line or terminal, in the directory where you want to clone the repository, run the **git clone** command with the HTTPS (GRC) clone URL. For example:

   ```
   git clone codecommit://MyCrossAccountAccessProfile@MySharedDemoRepo
   ```

   Unless you specify otherwise, the repository is cloned into a subdirectory with the same name as the repository.

2. Change directories to the cloned repository, and either add or make a change to a file. For example, you can add a file named *NewFile.txt*.

3. Add the file to the tracked changes for the local repo, commit the change, and push the file to the CodeCommit repository. For example:

   ```
   git add NewFile.txt
   git commit -m "Added a file to test cross-account access to this repository"
   git push
   ```

   For more information, see Getting started with Git and AWS CodeCommit.

Now that you've added a file, go to the CodeCommit console to view your commit, review other users' changes to the repo, participate in pull requests, and more.

To access the cross-account repository in the CodeCommit console

1. Sign in to the AWS Management Console in AccountB (*888888888888*) as the IAM user who has been granted cross-account access to the repository in AccountA.

2. Choose your user name on the navigation bar, and in the drop-down list, choose **Switch Role**.

   > **Note**
   >
   > If this is the first time you have selected this option, review the information on the page, and then choose **Switch Role** again.

3. On the **Switch Role** page, do the following:

   - In **Account**, enter the account ID for AccountA (for example, *111122223333*).

- In **Role**, enter the name of the role you want to assume for access to the repository in AccountA (for example, *MyCrossAccountRepositoryContributorRole*).
- In **Display Name**, enter a friendly name for this role. This name appears in the console when you are assuming this role. It also appears in the list of assumed roles the next time you want to switch roles in the console.
- (Optional) In **Color**, choose a color label for the display name.
- Choose **Switch Role**.

For more information, see Switching to a Role (AWS Management Console).

4. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.

   If the assumed role has permission to view the names of repositories in AccountA, you see a list of repositories and an error message that informs you that you do not have permissions to view their status. This is expected behavior. Choose the name of the shared repository from the list.

   If the assumed role does not have permission to view the names of repositories in AccountA, you see an error message and a blank list with no repositories. Paste the URL link to the repository or modify the console link and change `/list` to the name of the shared repository (for example, */MySharedDemoRepo*).

5. In **Code**, find the name of the file you added from your local computer. Choose it to browse the code in the file, and then browse the rest of the repository and start using its features.

   For more information, see Getting started with AWS CodeCommit.

Delete an AWS CodeCommit repository

You can use the CodeCommit console or the AWS CLI to delete a CodeCommit repository.

> **Note**
>
> Deleting a repository does not delete any local copies of that repository (local repos). To delete a local repo, use your local machine's directory and file management tools.

Topics

- Delete a CodeCommit repository (console)
- Delete a local repo
- Delete a CodeCommit repository (AWS CLI)

Delete a CodeCommit repository (console)

Follow these steps to use the CodeCommit console to delete a CodeCommit repository.

> **Important**
>
> After you delete a CodeCommit repository, you are no longer able to clone it to any local repo or shared repo. You are also no longer able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository you want to delete.
3. In the navigation pane, choose **Settings**.
4. On the **General** tab, in **Delete repository**, choose **Delete repository**. Enter **`delete`**, and then choose **Delete**. The repository is permanently deleted.

   > **Note**
   >
   > Deleting the repository in CodeCommit does not delete any local repos.

Delete a local repo

Use your local machine's directory and file management tools to delete the directory that contains the local repo.

Deleting a local repo does not delete any CodeCommit repository to which it might be connected.

Delete a CodeCommit repository (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to delete a CodeCommit repository, run the **delete-repository** command, specifying the name of the CodeCommit repository to delete (with the `--repository-name` option).

Important

After you delete a CodeCommit repository, you are no longer able to clone it to any local repo or shared repo. You are also no longer able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

Tip

To get the name of the CodeCommit repository, run the list-repositories command.

For example, to delete a repository named `MyDemoRepo`:

```
aws codecommit delete-repository --repository-name MyDemoRepo
```

If successful, the ID of the CodeCommit repository that was permanently deleted appears in the output:

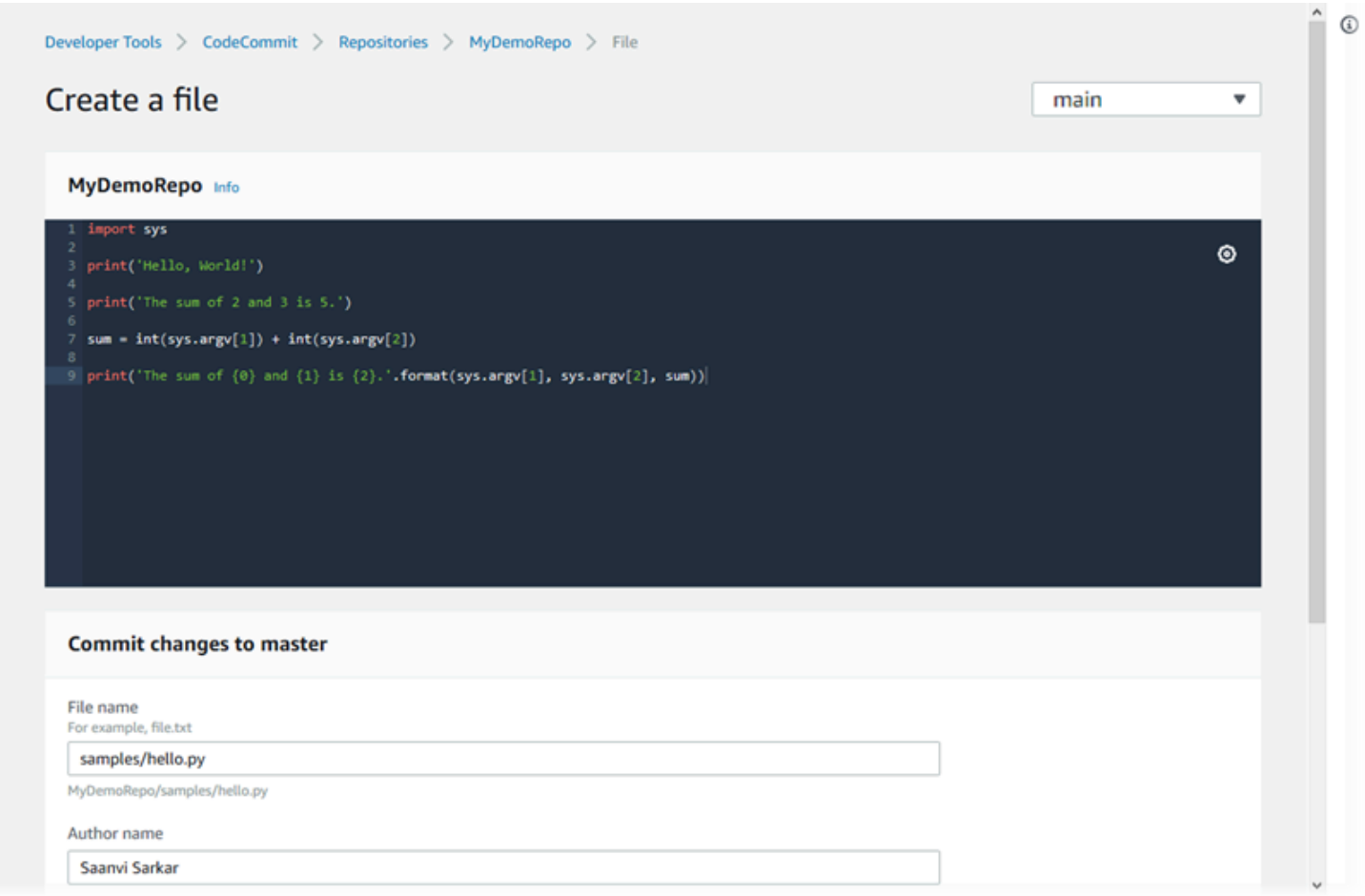
```
{
    "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE"
}
```

Deleting a CodeCommit repository does not delete any local repos that might be connected to it.

Working with files in AWS CodeCommit repositories

In CodeCommit, a file is a version-controlled, self-contained piece of information available to you and other users of the repository and branch where the file is stored. You can organize your repository files with a directory structure, just as you would on a computer. Unlike your computer, CodeCommit automatically tracks every change to a file. You can compare versions of a file and store different versions of a file in different repository branches.

To add or edit a file in a repository, you can use a Git client. You can also use the CodeCommit console, the AWS CLI, or the CodeCommit API.



For information about working with other aspects of your repository in CodeCommit, see Working with repositories, Working with pull requests, Working with branches, Working with commits, and Working with user preferences.

Topics

- Browse files in an AWS CodeCommit repository

- Create or add a file to an AWS CodeCommit repository
- Edit the contents of a file in an AWS CodeCommit repository

Browse files in an AWS CodeCommit repository

After you connect to a CodeCommit repository, you can clone it to a local repo or use the CodeCommit console to browse its contents. This topic describes how to use the CodeCommit console to browse the content of a CodeCommit repository.

> **Note**
>
> For active CodeCommit users, there is no charge for browsing code from the CodeCommit console. For information about when charges might apply, see Pricing.

Browse a CodeCommit repository

You can use the CodeCommit console to review the files contained in a repository or to quickly read the contents of a file.

To browse the content of a repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. On the **Repositories** page, from the list of repositories, choose the repository you want to browse.
3. In the **Code** view, browse the contents of the default branch for your repository.

   To change the view to a different branch or tag, choose the view selector button. Either choose a branch or tag name from the drop-down list, or in the filter box, enter the name of the branch or tag, and then choose it from the list.
4. Do one of the following:

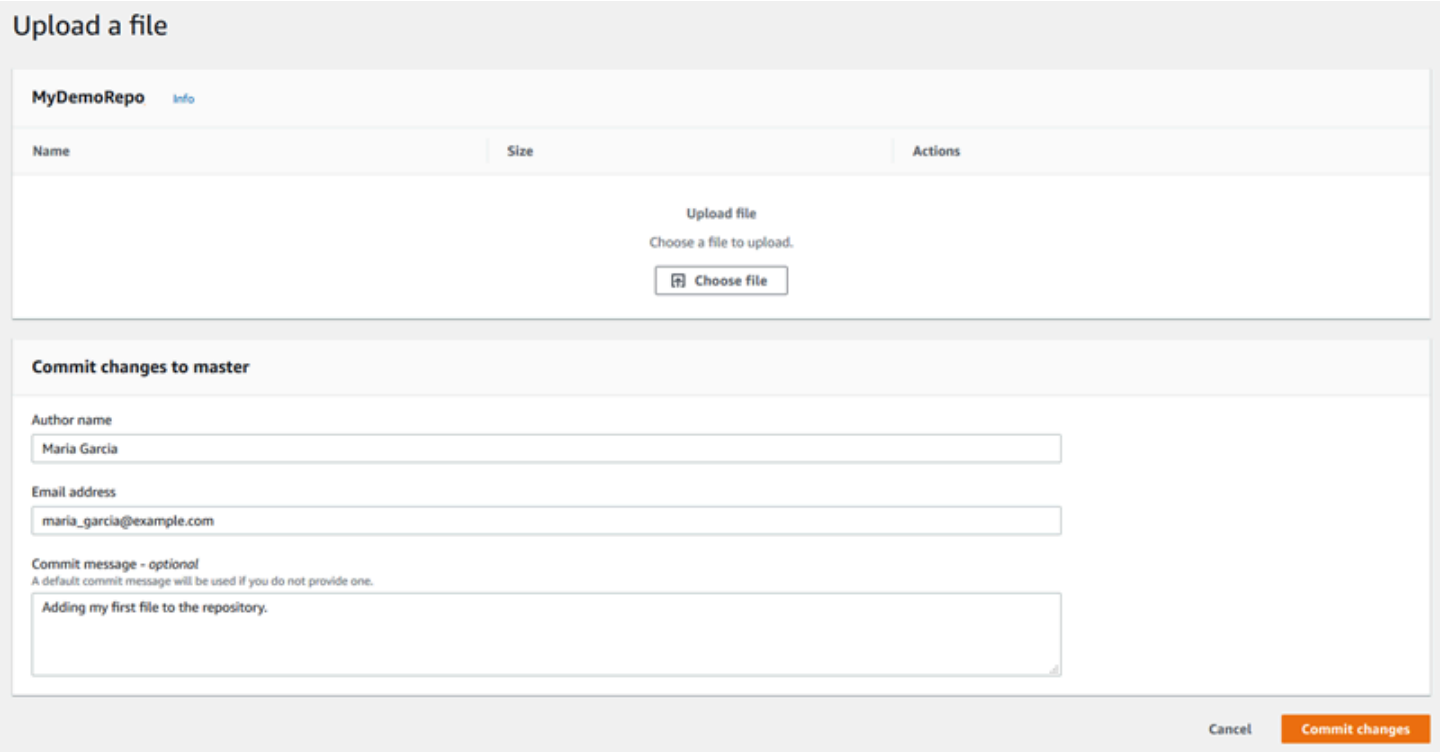
   - To view the contents of a directory, choose it from the list. You can choose any of the directories in the navigation list to return to that directory view. You can also use the up arrow at the top of the directory list.

- To view the contents of a file, choose it from the list. If the file is larger than the commit object limit, it cannot be displayed in the console and must be viewed in a local repo instead. For more information, see Quotas. To exit the file view, from the code navigation bar, choose the directory you want to view.

> **Note**
>
> Not all binary files are viewable in the console. If you choose a binary file and it is potentially viewable, a warning message appears, asking you to confirm that you want to display the contents. To view the file, choose **Show file contents**. If you do not want to view the file, from the code navigation bar, choose the directory you want to view. If you choose a markdown file (.md), use the **Rendered Markdown** and **Markdown Source** buttons to toggle between the rendered and syntax views. For more information, see Using Markdown in the console.

Create or add a file to an AWS CodeCommit repository

You can use the CodeCommit console, AWS CLI, or a Git client to add a file to a repository. You can upload a file from your local computer to the repository, or you can use the code editor in the console to create the file. The editor is a quick and easy way to add a simple file, such as a readme.md file, to a branch in a repository.



Topics

- Create or upload a file (console)
- Add a file (AWS CLI)
- Add a file (Git)

Create or upload a file (console)

You can use the CodeCommit console to create a file and add it to a branch in a CodeCommit repository. As part of creating the file, you can provide your user name and an email address. You can also add a commit message so other users understand who added the file and why. You can also upload a file directly from your local computer to a branch in a repository.

To add a file to a repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to add a file.
3. In the **Code** view, choose the branch where you want to add the file. By default, the contents of the default branch are shown when you open the **Code** view.

To change the view to a different branch, choose the view selector button. Either choose a branch name from the drop-down list, or in the filter box, enter the name of the branch, and then choose it from the list.

4. Choose **Add file**, and then choose one of the following options:

   - To use the code editor to create the contents of a file and add it to the repository, choose **Create file**.
   - To upload a file from your local computer to the repository, choose **Upload file**.

5. Provide information to other users about who added this file to the repository and why.

   - In **Author name**, enter your name. This name is used as both the author name and the committer name in the commit information. CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.
   - In **Email address**, enter an email address so that other repository users can contact you about this change.
   - In **Commit message**, enter a brief description. This is optional, but highly recommended. Otherwise, a default commit message is used.

6. Do one of the following:

   - If you are uploading a file, choose the file from your local computer.
   - If you are creating a file, enter the content you want to add in the code editor, and provide a name for the file.

7. Choose **Commit changes**.

Add a file (AWS CLI)

You can use the AWS CLI and the **put-file** command to add a file in an CodeCommit repository. You can also use the **put-file** command to add a directory or path structure for the file.

> **Note**
>
> To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To add a file to a repository

1. On your local computer, create the file you want to add to the CodeCommit repository.
2. At the terminal or command line, run the **put-file** command, specifying:

   - The repository where you want to add the file.
   - The branch where you want to add the file.
   - The full commit ID of the most recent commit made to that branch, also known as the tip or head commit.
   - The local location of the file. The syntax used for this location varies, depending on your local operating system.
   - The name of the file you want to add, including the path where the updated file is stored in the repository, if any.
   - The user name and email you want associated with this file.
   - A commit message that explains why you added this file.

   The user name, email address, and commit message are optional, but help other users know who made the change and why. If you do not supply a user name, CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.

   For example, to add a file named *ExampleSolution.py* to a repository named *MyDemoRepo* to a branch named *feature-randomizationfeature* whose most recent commit has an ID of *4c925148EXAMPLE*:

```
aws codecommit put-file --repository-name MyDemoRepo --branch-name feature-
randomizationfeature --file-content file://MyDirectory/ExampleSolution.py --file-
path /solutions/ExampleSolution.py --parent-commit-id 4c925148EXAMPLE --name "María
 García" --email "maría_garcía@example.com" --commit-message "I added a third
 randomization routine."
```

   Note

   When you add binary files, make sure that you use `fileb://` to specify the local location of the file.

If successful, this command returns output similar to the following:

```
{
   "blobId": "2eb4af3bEXAMPLE",
   "commitId": "317f8570EXAMPLE",
   "treeId": "347a3408EXAMPLE"
}
```

Add a file (Git)

You can add files in a local repo and push your changes to a CodeCommit repository. For more information, see Getting started with Git and AWS CodeCommit.

Edit the contents of a file in an AWS CodeCommit repository

You can use the CodeCommit console, AWS CLI, or a Git client to edit the contents of a file in a CodeCommit repository.

Topics

- Edit a file (console)
- Edit or delete a file (AWS CLI)
- Edit a file (Git)

Edit a file (console)

You can use the CodeCommit console to edit a file that has been added to a branch in a CodeCommit repository. As part of editing the file, you can provide your user name and an email address. You can also add a commit message so other users understand who made the change and why.

To edit a file in a repository

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to edit a file.

3. In the **Code** view, choose the branch where you want to edit the file. By default, the contents of the default branch are shown when you open the **Code** view.

   To change the view to a different branch, choose the view selector button. Either choose a branch name from the drop-down list, or in the filter box, enter the name of the branch, and then choose it from the list.

4. Navigate the contents of the branch and choose the file you want to edit. In the file view, choose **Edit**.

   > **Note**
   >
   > If you choose a binary file, a warning message appears asking you to confirm that you want to display the contents. You should not use the CodeCommit console to edit binary files.

5. Edit the file, and provide information to other users about who made this change and why.

   - In **Author name**, enter your name. This name is used as both the author name and the committer name in the commit information. CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.
   - In **Email address**, enter an email address so that other repository users can contact you about this change.
   - In **Commit message**, enter a brief description of your changes.

6. Choose **Commit changes** to save your changes to the file and commit the changes to the repository.

Edit or delete a file (AWS CLI)

You can use the AWS CLI and the **put-file** command to make changes to a file in a CodeCommit repository. You can also use the **put-file** command to add a directory or path structure for the changed file, if you want to store the changed file in a location different from the original. If you want to delete a file entirely, you can use the **delete-file** command.

> **Note**
>
> To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To edit a file in a repository

1. Using a local copy of the file, make the changes you want to add to the CodeCommit repository.
2. At the terminal or command line, run the **put-file** command, specifying:

   - The repository where you want to add the edited file.
   - The branch where you want to add the edited file.
   - The full commit ID of the most recent commit made to that branch, also known as the tip or head commit.
   - The local location of the file.
   - The name of the updated file you want to add, including the path where the updated file is stored in the repository, if any.
   - The user name and email you want associated with this file change.
   - A commit message that explains the change you made.

   The user name, email address, and commit message are optional, but help other users know who made the change and why. If you do not supply a user name, CodeCommit defaults to using your IAM user name or a derivation of your console login.

   For example, to add edits made to a file named *ExampleSolution.py* to a repository named *MyDemoRepo* to a branch named *feature-randomizationfeature* whose most recent commit has an ID of *4c925148EXAMPLE*:

```
aws codecommit put-file --repository-name MyDemoRepo --branch-name feature-
randomizationfeature --file-content file://MyDirectory/ExampleSolution.py --file-
path /solutions/ExampleSolution.py --parent-commit-id 4c925148EXAMPLE --name "María
 García" --email "maría_garcía@example.com" --commit-message "I fixed the bug Mary
 found."
```

> **Note**
>
> If you want to add a changed binary file, make sure to use `--file-content` with the notation **`fileb://`***`MyDirectory`*`/`*`MyFile`*`.raw`.

If successful, this command returns output similar to the following:

```
{
   "blobId": "2eb4af3bEXAMPLE",
   "commitId": "317f8570EXAMPLE",
   "treeId": "347a3408EXAMPLE"
}
```

To delete a file, use the **delete-file** command. For example, to delete a file named *README.md* in a branch named *main* with a most recent commit ID of *c5709475EXAMPLE* in a repository named *MyDemoRepo*:

```
aws codecommit delete-file --repository-name MyDemoRepo --branch-name main --file-
path README.md --parent-commit-id c5709475EXAMPLE
```

If successful, this command returns output similar to the following:

```
{
  "blobId":"559b44fEXAMPLE",
  "commitId":"353cf655EXAMPLE",
  "filePath":"README.md",
  "treeId":"6bc824cEXAMPLE"
}
```

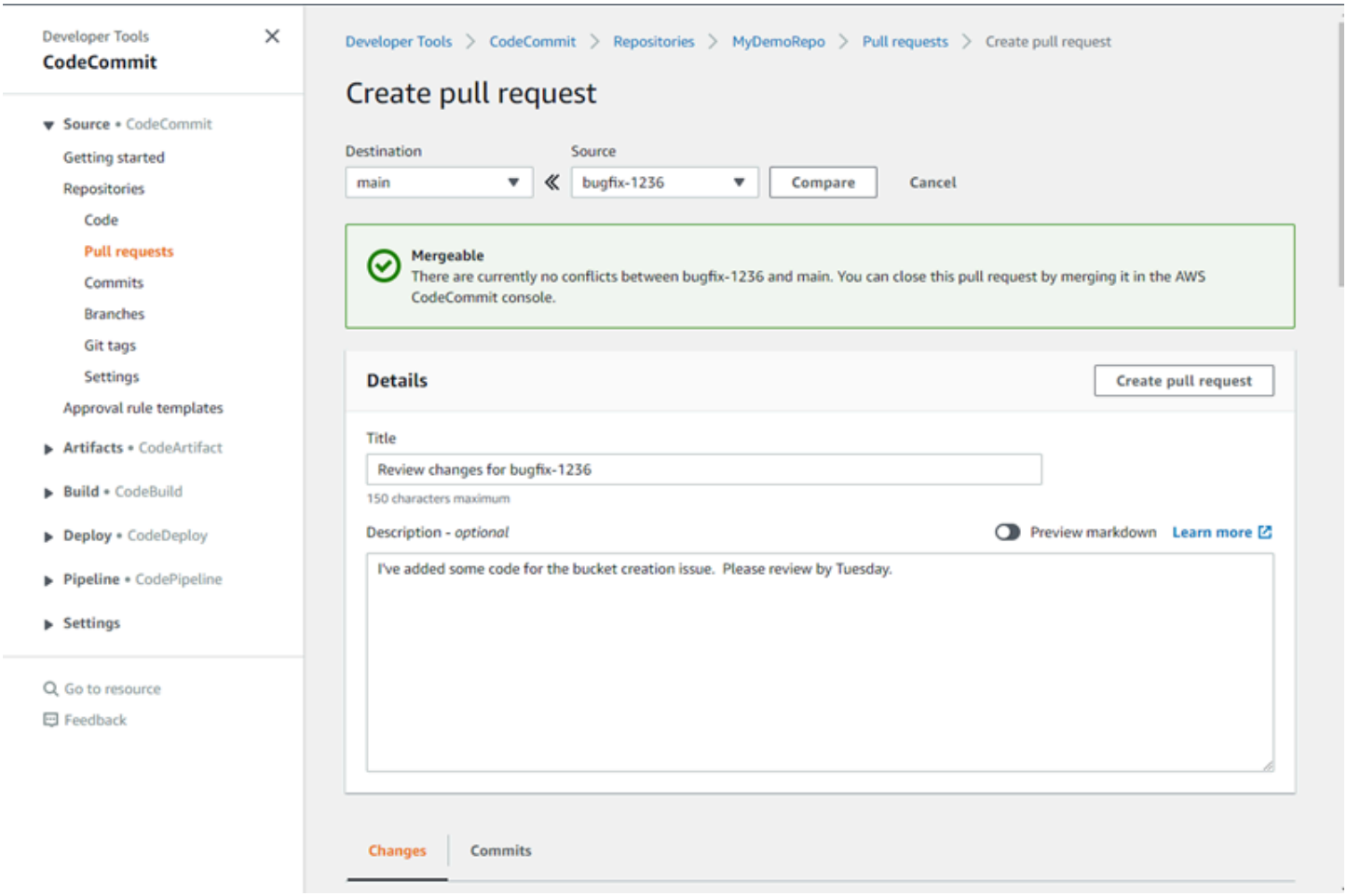
Edit a file (Git)

You can edit files in a local repo and push your changes to a CodeCommit repository. For more information, see Getting started with Git and AWS CodeCommit.

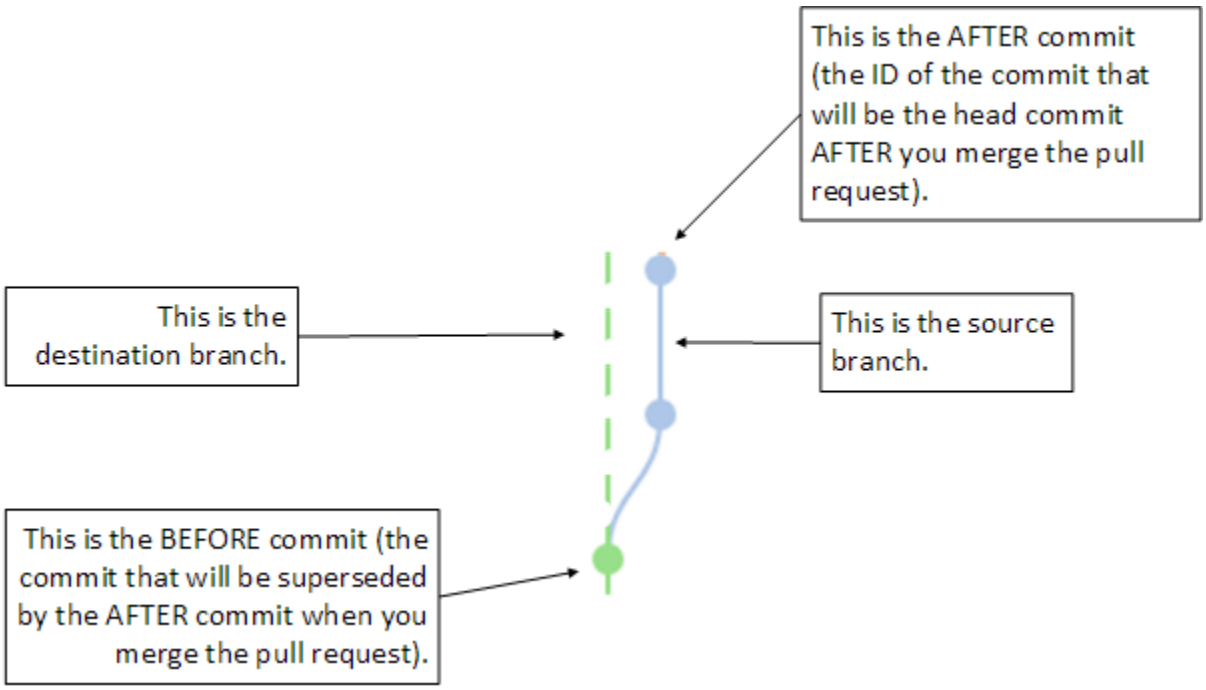
Working with pull requests in AWS CodeCommit repositories

A pull request is the primary way you and other repository users can review, comment on, and merge code changes from one branch to another. You can use pull requests to collaboratively review code changes for minor changes or fixes, major feature additions, or new versions of your released software. Here is one possible workflow for a pull request:

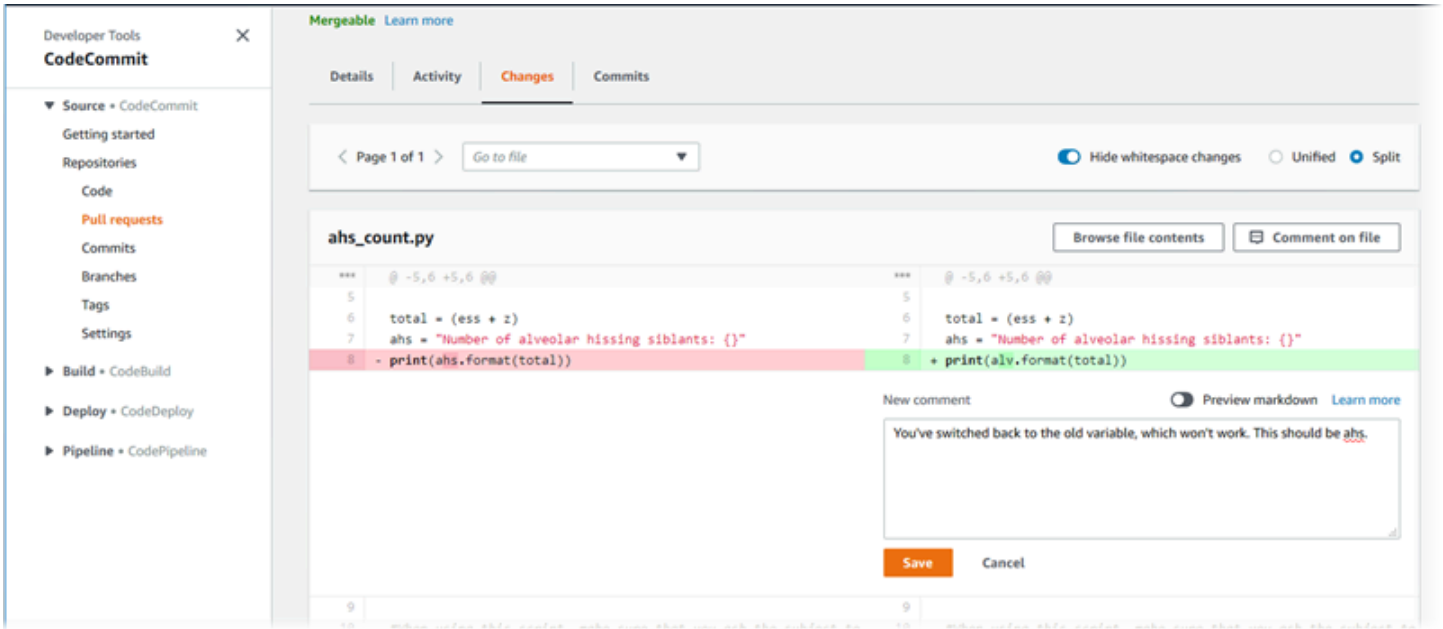
Li Juan, a developer working in a repo named MyDemoRepo, wants to work on a new feature for an upcoming version of a product. To keep her work separate from production-ready code, she creates a branch off of the default branch and names it *`feature-randomizationfeature`*. She writes code, makes commits, and pushes the new feature code into this branch. She wants other repository users to review the code for quality before she merges her changes into the default branch. To do this, she creates a pull request. The pull request contains the comparison between her working branch and the branch of the code where she intends to merge her changes (in this case, the default branch). She can also create an approval rule that requires a specified number of users to approve her pull request. She can even specify an approval pool of users. Other users review her code and changes, adding comments and suggestions. She might update her working branch multiple times with code changes in response to comments. Her changes are incorporated into the pull request every time she pushes them to that branch in CodeCommit. She might also incorporate changes that have been made in the intended destination branch while the pull request is open, so users can be sure they're reviewing all of the proposed changes in context. When she and her reviewers are satisfied, and the conditions for approval rules (if any) have been satisfied, she or one of her reviewers merges her code and closes the pull request.



Pull requests require two branches: a source branch that contains the code you want reviewed, and a destination branch, where you merge the reviewed code. The source branch contains the AFTER commit, which is the commit that contains the changes you want to merge into the destination branch. The destination branch contains the BEFORE commit, which represents the state of the code before the pull request branch is merged into the destination branch. The choice of merge strategy affects the details of how commits are merged between the source and destination branches in the CodeCommit console. For more information about merge strategies in CodeCommit, see Merge a pull request (console).



The pull request displays the differences between the tip of the source branch and the latest commit on the destination branch when the pull request is created, so users can view and comment on the changes. You can update the pull request in response to comments by committing and pushing changes to the source branch.



When your code has been reviewed, and the approval rule requirements (if any) have been satisfied, you can close the pull request in one of several ways:

- Merge the branches locally and push your changes. This closes the request automatically if the fast-forward merge strategy is used and there are no merge conflicts.
- Use the AWS CodeCommit console to close the pull request without merging, resolve conflicts in a merge, or, if there are no conflicts, close and merge the branches using one of the available merge strategies.
- Use the AWS CLI.

Before you create a pull request:

- Make sure that you have committed and pushed the code changes you want reviewed to a branch (the source branch).
- Set up notifications for your repository, so other users can be notified about the pull request and changes to it. (This step is optional, but recommended.)
- Create and associate approval rule templates with your repository, so that approval rules are automatically created for pull requests to help ensure code quality. For more information, see Working with approval rule templates.

Pull requests are more effective when you've set up IAM users for your repository users in your Amazon Web Services account. It's easier to identify which user made which comment. The other advantage is that IAM users can use Git credentials for repository access. For more information, see Step 1: Initial configuration for CodeCommit. You can use pull requests with other kinds of users, including federated access users.

For information about working with other aspects of your repository in CodeCommit, see Working with repositories, Working with approval rule templates, Working with files, Working with commits, Working with branches, and Working with user preferences.

Topics

- Create a pull request
- Create an approval rule for a pull request
- View pull requests in an AWS CodeCommit repository
- Review a pull request
- Update a pull request
- Edit or delete an approval rule for a pull request

- Override approval rules on a pull request
- Merge a pull request in an AWS CodeCommit repository
- Resolve conflicts in a pull request in an AWS CodeCommit repository
- Close a pull request in an AWS CodeCommit repository

Create a pull request

Creating pull requests helps other users see and review your code changes before you merge them into another branch. First, you create a branch for your code changes. This is referred to as the source branch for a pull request. After you commit and push changes to the repository, you can create a pull request that compares the contents of that branch (the source branch) to the branch where you want to merge your changes after the pull request is closed (the destination branch).

You can use the AWS CodeCommit console or the AWS CLI to create pull requests for your repository.

Topics

- Create a pull request (console)
- Create a pull request (AWS CLI)

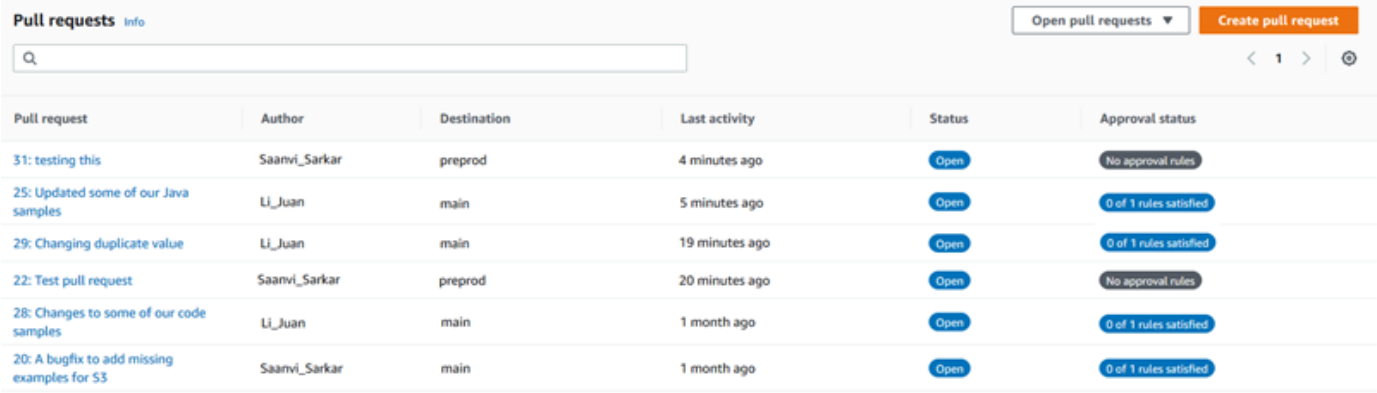
Create a pull request (console)

You can use the CodeCommit console to create a pull request in a CodeCommit repository. If your repository is configured with notifications, subscribed users receive an email when you create a pull request.

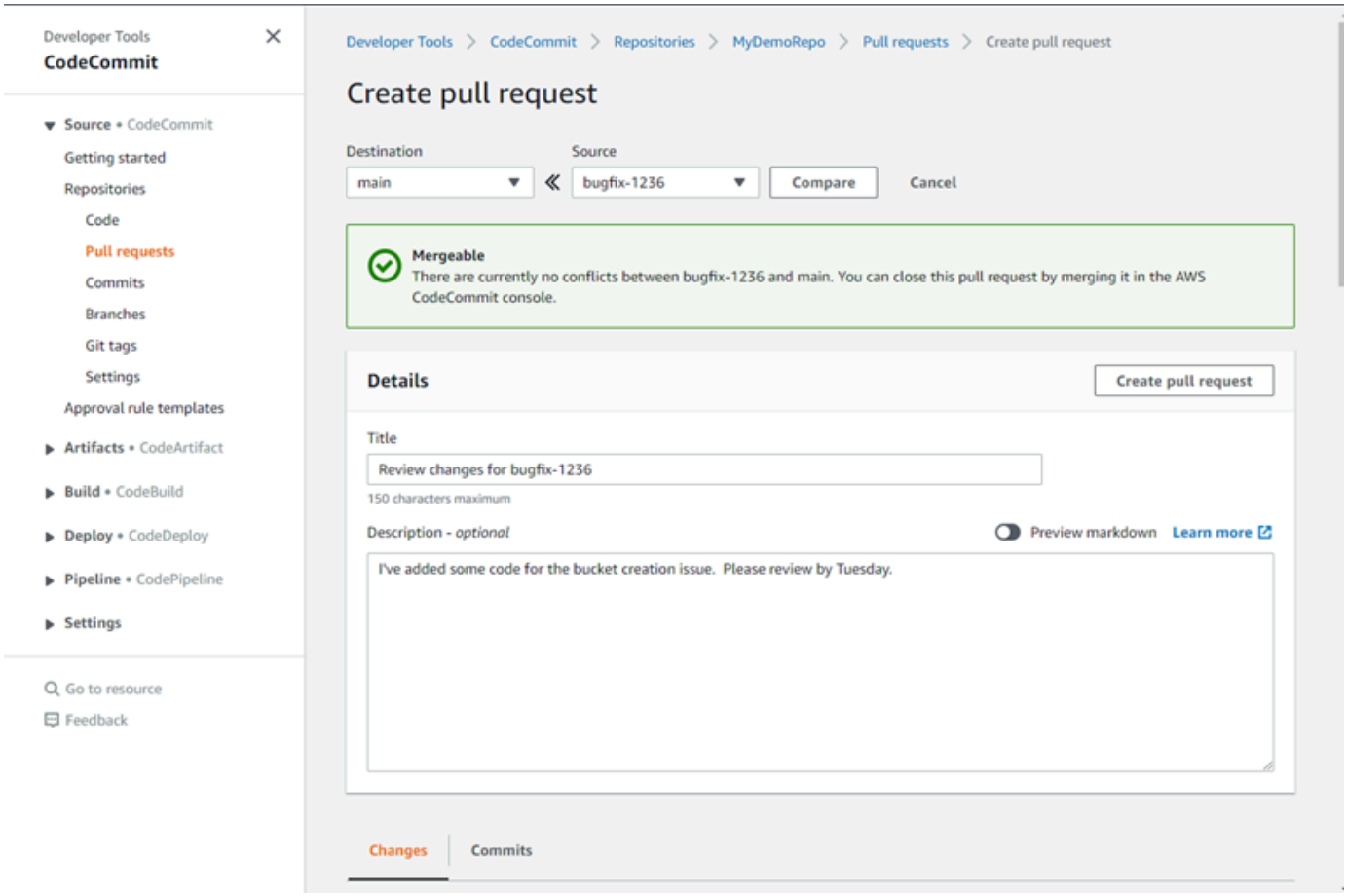
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to create a pull request.
3. In the navigation pane, choose **Pull Requests**.

   > **Tip**
   >
   > You can also create pull requests from **Branches** and **Code**.

4. Choose **Create pull request**.



5. In **Create pull request**, in **Source**, choose the branch that contains the changes you want reviewed.
6. In **Destination**, choose the branch where you intend to merge your code changes when the pull request is closed.
7. Choose **Compare**. A comparison runs on the two branches, and the differences between them are displayed. An analysis is also performed to determine whether the two branches can be merged automatically when the pull request is closed.
8. Review the comparison details and the changes to be sure that the pull request contains the changes and commits you want reviewed. If not, adjust your choices for source and destination branches, and choose **Compare** again.
9. When you are satisfied with the comparison results for the pull request, in **Title**, enter a short but descriptive title for this review. This is the title that appears in the list of pull requests for the repository.
10. (Optional) In **Description**, enter details about this review and any other useful information for reviewers.
11. Choose **Create**.



Your pull request appears in the list of pull requests for the repository. If you configured notifications, subscribers to the Amazon SNS topic receive an email to inform them of the newly created pull request.

Create a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to create a pull request in a CodeCommit repository

- Run the **create-pull-request** command, specifying:
  - The name of the pull request (with the **--title** option).
  - The description of the pull request (with the **--description** option).
  - A list of targets for the **create-pull-request** command, including:

- The name of the CodeCommit repository where the pull request is created (with the **repositoryName** attribute).
- The name of the branch that contains the code changes you want reviewed, also known as the source branch (with the **sourceReference** attribute).
- (Optional) The name of the branch where you intend to merge your code changes, also known as the destination branch, if you do not want to merge to the default branch (with the **destinationReference** attribute).

- A unique, client-generated idempotency token (with the **--client-request-token** option).

This example creates a pull request named *Pronunciation difficulty analyzer* with a description of *Please review these changes by Tuesday* that targets the *jane-branch* source branch. The pull request is to be merged into the default branch *main* in a CodeCommit repository named MyDemoRepo:

```
aws codecommit create-pull-request --title "Pronunciation difficulty analyzer"
 --description "Please review these changes by Tuesday" --client-request-token
 123Example --targets repositoryName=MyDemoRepo,sourceReference=jane-branch
```

Create an approval rule for a pull request

Creating approval rules for your pull requests helps ensure the quality of your code by requiring users to approve the pull request before the code can be merged into the destination branch. You can specify the number of users who must approve a pull request. You can also specify an approval pool of users for the rule. If you do so, only approvals from those users count toward the number of required approvals for the rule.

> **Note**
>
> You can also create approval rule templates, which can help you automate the creation of approval rules across repositories that will apply to every pull request. For more information, see Working with approval rule templates.

You can use the AWS CodeCommit console or the AWS CLI to create approval rules for your repository.

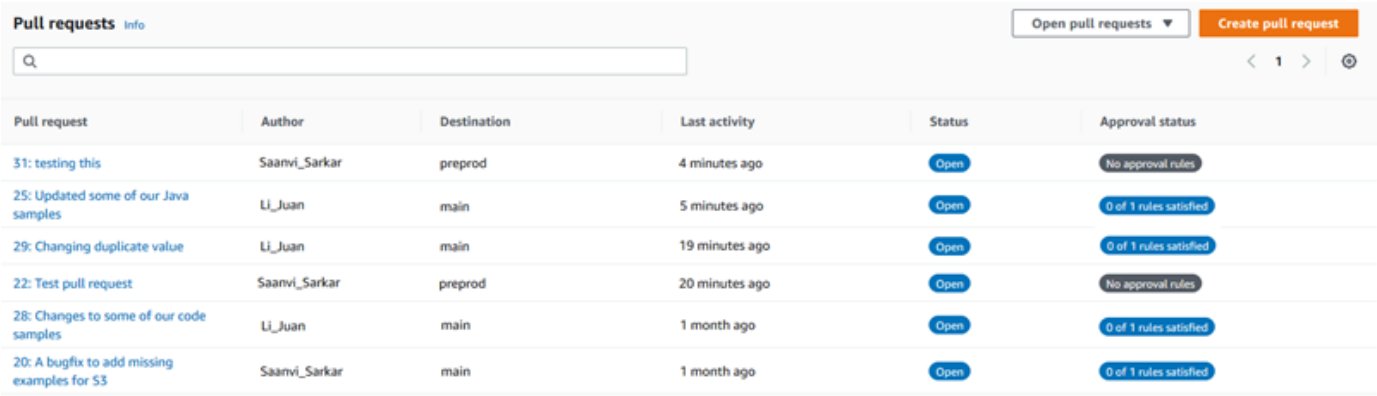
Topics

- Create an approval rule for a pull request (console)
- Create an approval rule for a pull request (AWS CLI)

Create an approval rule for a pull request (console)

You can use the CodeCommit console to create an approval rule for a pull request in a CodeCommit repository.

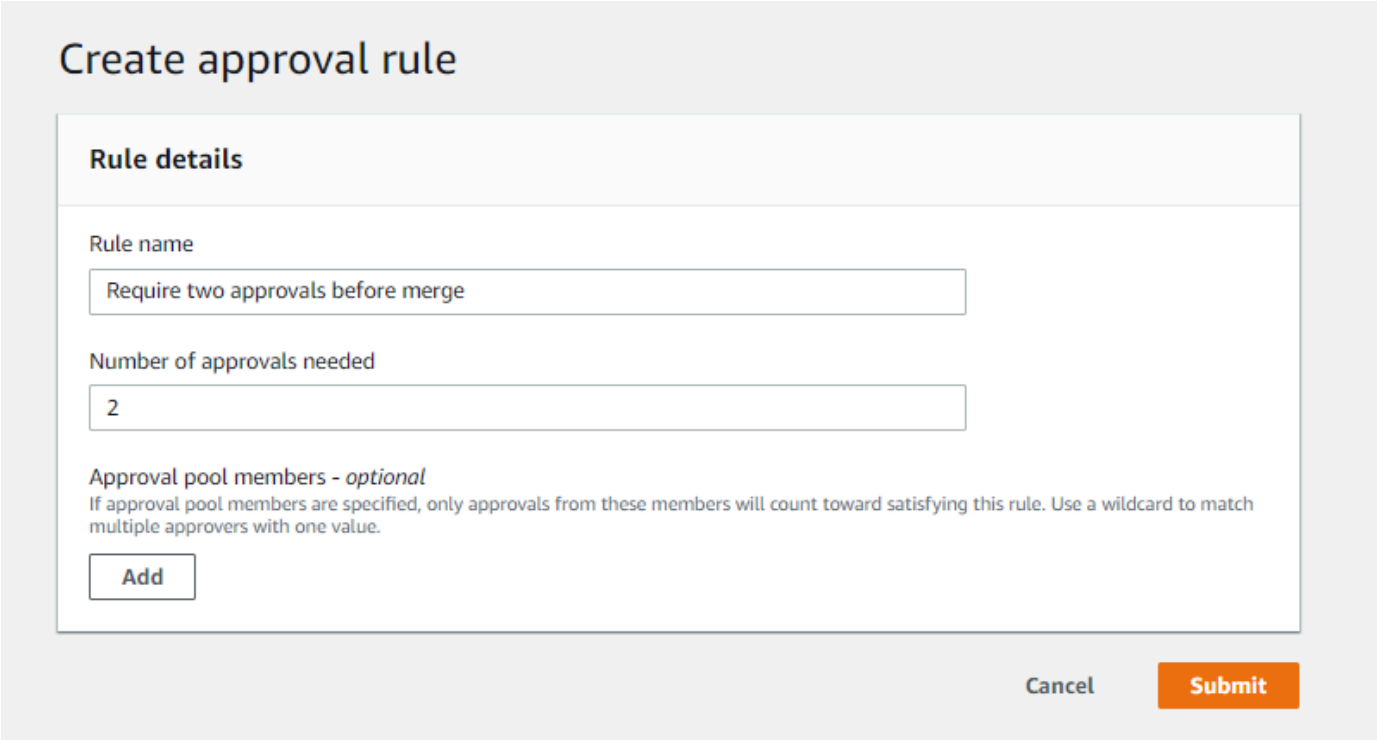
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to create an approval rule for a pull request.
3. In the navigation pane, choose **Pull Requests**.
4. Choose the pull request for which you want to create an approval rule from the list. You can only create approval rules for open pull requests.



5. In the pull request, choose **Approvals**, and then choose **Create approval rule**.
6. In **Rule name**, give the rule a descriptive name so you know what it is for. For example, if you want to require two people to approve a pull request before it can be merged, you might name the rule **`Require two approvals before merge`**.

> **Note**
>
> You cannot change the name of an approval rule after you create it.

In **Number of approvals needed**, enter the number you want. The default is 1.



7. (Optional) If you want to require that the approvals for a pull request come from a specific group of users, in **Approval rule members**, choose **Add**. In **Approver type**, choose one of the following:

   - **IAM user name or assumed role**: This option prepopulates the AWS account ID with the account you used to sign in, and only requires a name. It can be used for both IAM users and federated access users whose name matches the provided name. This is a very powerful option that offers a great deal of flexibility. For example, if you are signed in with the Amazon Web Services account 123456789012 and choose this option, and you specify **`Mary_Major`**, all of the following are counted as approvals coming from that user:
     - An IAM user in the account (`arn:aws:iam::123456789012:user/Mary_Major`)
     - A federated user identified in IAM as Mary_Major (`arn:aws:sts::123456789012:federated-user/Mary_Major`)

     This option would not recognize an active session of someone assuming the role of **`CodeCommitReview`** with a role session name of Mary_Major (`arn:aws:sts::123456789012:assumed-role/CodeCommitReview/Mary_Major`) unless you include a wildcard (`*Mary_Major`). You can also specify the role name explicitly (`CodeCommitReview/Mary_Major`).

- **Fully qualified ARN**: This option allows you to specify the fully qualified Amazon Resource Name (ARN) of the IAM user or role. This option also supports assumed roles used by other AWS services, such as AWS Lambda and AWS CodeBuild. For assumed roles, the ARN format should be `arn:aws:sts::AccountID:assumed-role/RoleName` for roles and `arn:aws:sts::AccountID:assumed-role/FunctionName` for functions.

If you chose **IAM user name or assumed role** as the approver type, in **Value**, enter the name of the IAM user or role or the fully qualified ARN of the user or role. Choose **Add** again to add more users or roles, until you have added all the users or roles whose approvals count toward the number of required approvals.

Both approver types allow you to use wildcards (*) in their values. For example, if you choose the **IAM user name or assumed role** option, and you specify **`CodeCommitReview/*`**, all users who assume the role of **`CodeCommitReview`** are counted in the approval pool. Their individual role session names count toward the required number of approvers. In this way, both Mary_Major and Li_Juan are counted as approvals when signed in and assuming the role of `CodeCommitReview`. For more information about IAM ARNs, wildcards, and formats, see IAM Identifiers.

> **Note**
>
> Approval rules do not support cross-account approvals.

8. When you have finished configuring the approval rule, choose **Submit**.

Create an approval rule for a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To create an approval rule for a pull request in a CodeCommit repository

- Run the **create-pull-request-approval-rule** command, specifying:
  - The ID of the pull request (with the **--id** option).
  - The name of the approval rule (with the **--approval-rule-name** option).
  - The content of the approval rule (with the **--approval-rule-content** option).

When you create the approval rule, you can specify approvers in an approval pool in one of two ways:

- **CodeCommitApprovers**: This option only requires an Amazon Web Services account and a resource. It can be used for both IAM users and federated access users whose name matches the provided resource name. This is a very powerful option that offers a great deal of flexibility. For example, if you specify the Amazon Web Services account 123456789012 and **`Mary_Major`**, all of the following are counted as approvals coming from that user:
  - An IAM user in the account (`arn:aws:iam::123456789012:user/Mary_Major`)
  - A federated user identified in IAM as Mary_Major (`arn:aws:sts::123456789012:federated-user/Mary_Major`)

  This option would not recognize an active session of someone assuming the role of **`CodeCommitReview`** with a role session name of Mary_Major (`arn:aws:sts::123456789012:assumed-role/CodeCommitReview/Mary_Major`) unless you include a wildcard (`*Mary_Major`).
- **Fully qualified ARN**: This option allows you to specify the fully qualified Amazon Resource Name (ARN) of the IAM user or role.

For more information about IAM ARNs, wildcards, and formats, see IAM Identifiers.

The following example creates an approval rule named `Require two approved approvers` for a pull request with the ID of 27. The rule specifies two approvals are required from an approval pool. The pool includes all users who access CodeCommit and assume the role of **`CodeCommitReview`** in the `123456789012` Amazon Web Services account. It also includes either an IAM user or federated user named `Nikhil_Jayashankar` in the same Amazon Web Services account:

```
aws codecommit create-pull-request-approval-rule --pull-request-id 27
 --approval-rule-name "Require two approved approvers" --approval-
rule-content "{\"Version\": \"2018-11-08\",\"Statements\": [{\"Type\":
 \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers
\": [\"CodeCommitApprovers:123456789012:Nikhil_Jayashankar\",
 \"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}"
```

View pull requests in an AWS CodeCommit repository

You can use the AWS CodeCommit console or the AWS CLI to view pull requests for your repository. By default, you see only open pull requests, but you can change the filter to view all pull requests, only closed requests, only pull requests that you created, and more.

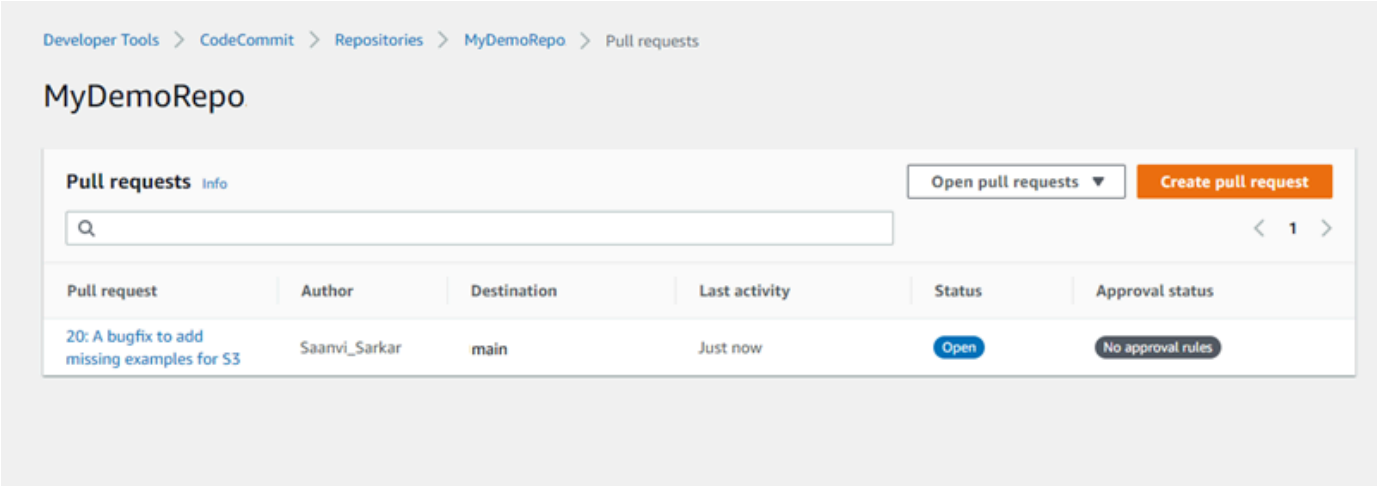
Topics

- View pull requests (console)
- View pull requests (AWS CLI)

View pull requests (console)

You can use the AWS CodeCommit console to view a list of pull requests in a CodeCommit repository. By changing the filter, you can change the list display to only show you a certain set of pull requests. For example, you can view a list of pull requests you created with a status of **Open**, or you can choose a different filter and view pull requests you created with a status of **Closed**.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to view pull requests.
3. In the navigation pane, choose **Pull Requests**.
4. By default, a list of all open pull requests is displayed.



5. To change the display filter, choose from the list of available filters:
   - **Open pull requests** (default): Displays all pull requests with a status of **Open**.

- **All pull requests**: Displays all pull requests.
- **Closed pull requests**: Displays all pull requests with a status of **Closed**.
- **My pull requests**: Displays all pull requests that you created, regardless of the status. It does not display reviews that you have commented on or otherwise participated in.
- **My open pull requests**: Displays all pull requests that you created with a status of **Open**.
- **My closed pull requests**: Displays all pull requests that you created with a status of **Closed**.

6. When you find a pull request in the displayed list that you would like to view, choose it.

View pull requests (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

Follow these steps to use the AWS CLI to view pull requests in an CodeCommit repository.

1. To view a list of pull requests in a repository, run the **list-pull-requests** command, specifying:

   - The name of the CodeCommit repository where you want to view pull requests (with the **--repository-name** option).
   - (Optional) The status of the pull request (with the **--pull-request-status** option).
   - (Optional) The Amazon Resource Name (ARN) of the IAM user who created the pull request (with the **--author-arn** option).
   - (Optional) An enumeration token that can be used to return batches of results (with the **--next-token** option)
   - (Optional) A limit on the number of returned results per request (with the **--max-results** option).

   For example, to list pull requests created by an IAM user with the ARN *arn:aws:iam::111111111111:user/Li_Juan* and the status of *CLOSED* in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit list-pull-requests --author-arn arn:aws:iam::111111111111:user/
   Li_Juan --pull-request-status CLOSED --repository-name MyDemoRepo
   ```

   If successful, this command produces output similar to the following:

```
{
   "nextToken": "",
   "pullRequestIds": ["2","12","16","22","23","35","30","39","47"]
}
```

Pull request IDs are displayed in the order of most recent activity.

2. To view details of a pull request, run the **get-pull-request** command with the **--pull-request-id** option, specifying the ID of the pull request. For example, to view information about a pull request with the ID of *27*:

```
aws codecommit get-pull-request --pull-request-id 27
```

If successful, this command produces output similar to the following:

```
{
    "pullRequest": {
        "approvalRules": [
            {
                "approvalRuleContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
                "approvalRuleId": "dd8b17fe-EXAMPLE",
                "approvalRuleName": "2-approver-rule-for-main",
                "creationDate": 1571356106.936,
                "lastModifiedDate": 571356106.936,
                "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
                "ruleContentSha256": "4711b576EXAMPLE"
            }
        ],
        "lastActivityDate": 1562619583.565,
        "pullRequestTargets": [
            {
                "sourceCommit": "ca45e279EXAMPLE",
                "sourceReference": "refs/heads/bugfix-1234",
                "mergeBase": "a99f5ddbEXAMPLE",
                "destinationReference": "refs/heads/main",
                "mergeMetadata": {
                    "isMerged": false
                },
                "destinationCommit": "2abfc6beEXAMPLE",
```

```
                "repositoryName": "MyDemoRepo"
            }
        ],
        "revisionId": "e47def21EXAMPLE",
        "title": "Quick fix for bug 1234",
        "authorArn": "arn:aws:iam::123456789012:user/Nikhil_Jayashankar",
        "clientRequestToken": "d8d7612e-EXAMPLE",
        "creationDate": 1562619583.565,
        "pullRequestId": "27",
        "pullRequestStatus": "OPEN"
    }
}
```

3. To view approvals on a pull request, run the **get-pull-request-approval-state** command, specifying:

   - The ID of the pull request (using the **--pull-request-id** option).
   - The revision ID of the pull request (using the **--revision-id option)**. You can get the current revision ID for a pull request by using the get-pull-request command.

   For example, to view approvals on a pull request with an ID of *8* and a revision ID of *9f29d167EXAMPLE*:

   ```
   aws codecommit get-pull-request-approval-state --pull-request-id 8 --revision-
   id 9f29d167EXAMPLE
   ```

   If successful, this command produces output similar to the following:

   ```
   {
       "approvals": [
           {
               "userArn": "arn:aws:iam::123456789012:user/Mary_Major",
               "approvalState": "APPROVE"
           }
       ]
   }
   ```

4. To view events in a pull request, run the **describe-pull-request-events** command with the **--pull-request-id** option, specifying the ID of the pull request. For example, to view the events for a pull request with the ID of *8*:

```
aws codecommit describe-pull-request-events --pull-request-id 8
```

If successful, this command produces output similar to the following:

```
{
    "pullRequestEvents": [
        {
            "pullRequestId": "8",
            "pullRequestEventType": "PULL_REQUEST_CREATED",
            "eventDate": 1510341779.53,
            "actor": "arn:aws:iam::111111111111:user/Zhang_Wei"
        },
        {
            "pullRequestStatusChangedEventMetadata": {
                "pullRequestStatus": "CLOSED"
            },
            "pullRequestId": "8",
            "pullRequestEventType": "PULL_REQUEST_STATUS_CHANGED",
            "eventDate": 1510341930.72,
            "actor": "arn:aws:iam::111111111111:user/Jane_Doe"
        }
    ]
}
```

5. To view whether there are any merge conflicts for a pull request, run the **get-merge-conflicts** command, specifying:

   - The name of the CodeCommit repository (with the **--repository-name** option).
   - The branch, tag, HEAD, or other fully qualified reference for the source of the changes to use in the merge evaluation (with the **--source-commit-specifier** option).
   - The branch, tag, HEAD, or other fully qualified reference for the destination of the changes to use in the merge evaluation (with the **--destination-commit-specifier** option).
   - The merge option to use (with the **--merge-option** option)

   For example, to view whether there are any merge conflicts between the tip of a source branch named *my-feature-branch* and a destination branch named *main* in a repository named MyDemoRepo:

```
aws codecommit get-merge-conflicts --repository-name MyDemoRepo --source-commit-
specifier my-feature-branch --destination-commit-specifier main --merge-option
 FAST_FORWARD_MERGE
```

If successful, this command returns output similar to the following:

```
{
    "destinationCommitId": "fac04518EXAMPLE",
    "mergeable": false,
    "sourceCommitId": "16d097f03EXAMPLE"
}
```

Review a pull request

You can use the AWS CodeCommit console to review the changes included in a pull request. You can add comments to the request, files, and individual lines of code. You can also reply to comments made by other users. If your repository is configured with notifications, you receive emails when users reply to your comments or when users comment on a pull request.

You can use the AWS CLI to comment on a pull request and reply to comments. To review the changes, you must use the CodeCommit console, the **git diff** command, or a diff tool.

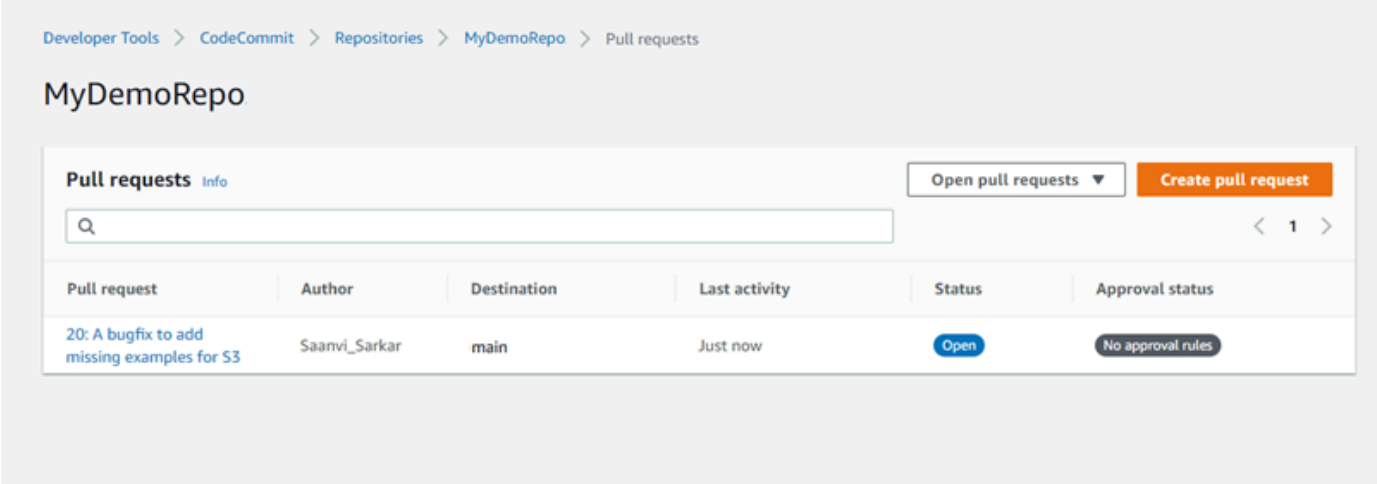
Topics

- Review a pull request (console)
- Review pull requests (AWS CLI)

Review a pull request (console)

You can use the CodeCommit console to review a pull request in a CodeCommit repository.

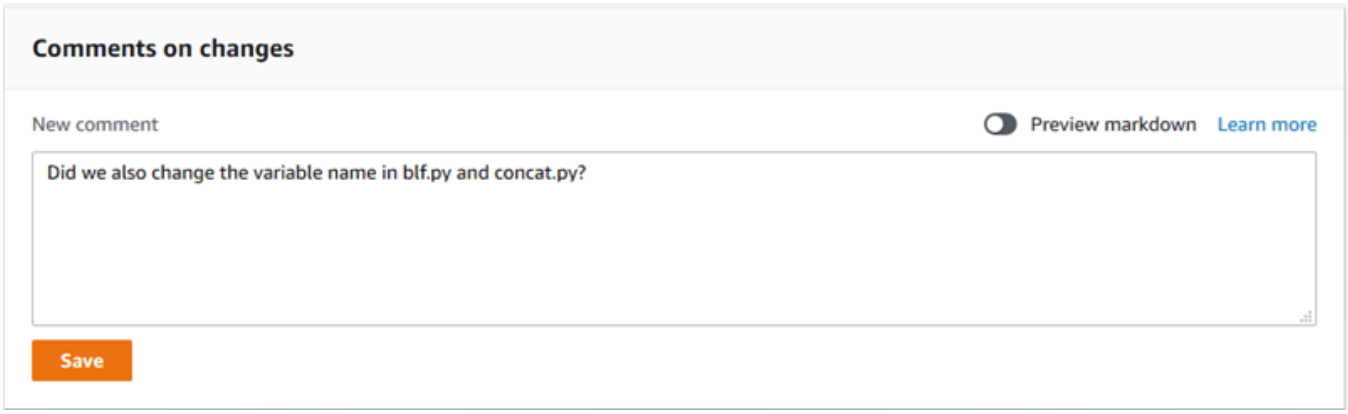
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository.
3. In the navigation pane, choose **Pull requests**.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to review.



> **Note**
>
> You can comment on a closed or merged pull request, but you cannot merge or reopen it.

5. In the pull request, choose **Changes**.
6. Do one of the following:

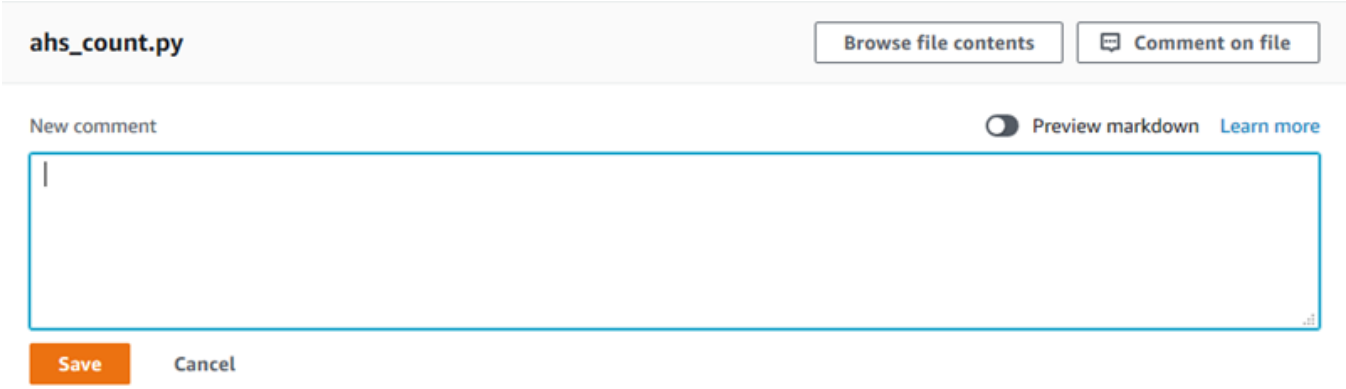
   - To add a general comment for the entire pull request, in **Comments on changes**, in **New comment**, enter a comment, and then choose **Save**. You can use Markdown, or you can enter your comment in plaintext.

   - To add a comment to a file in the commit, in **Changes**, find the name of the file. Choose the comment icon

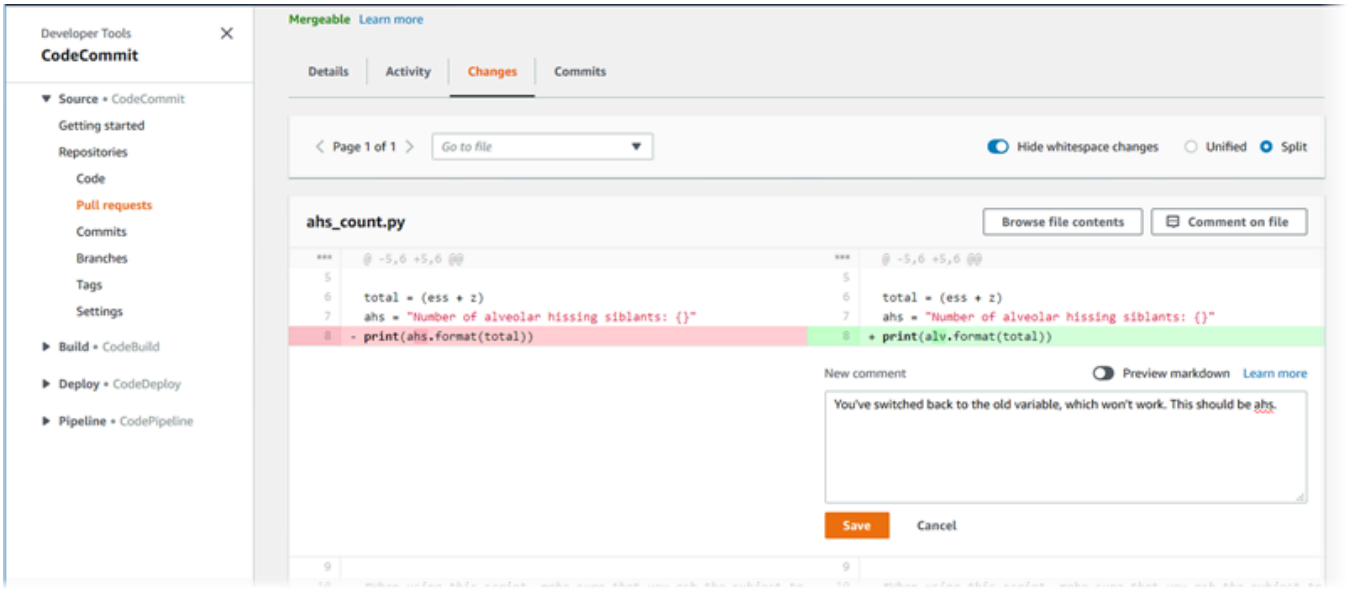
     

     that appears next to the file name, enter a comment, and then choose **Save**.

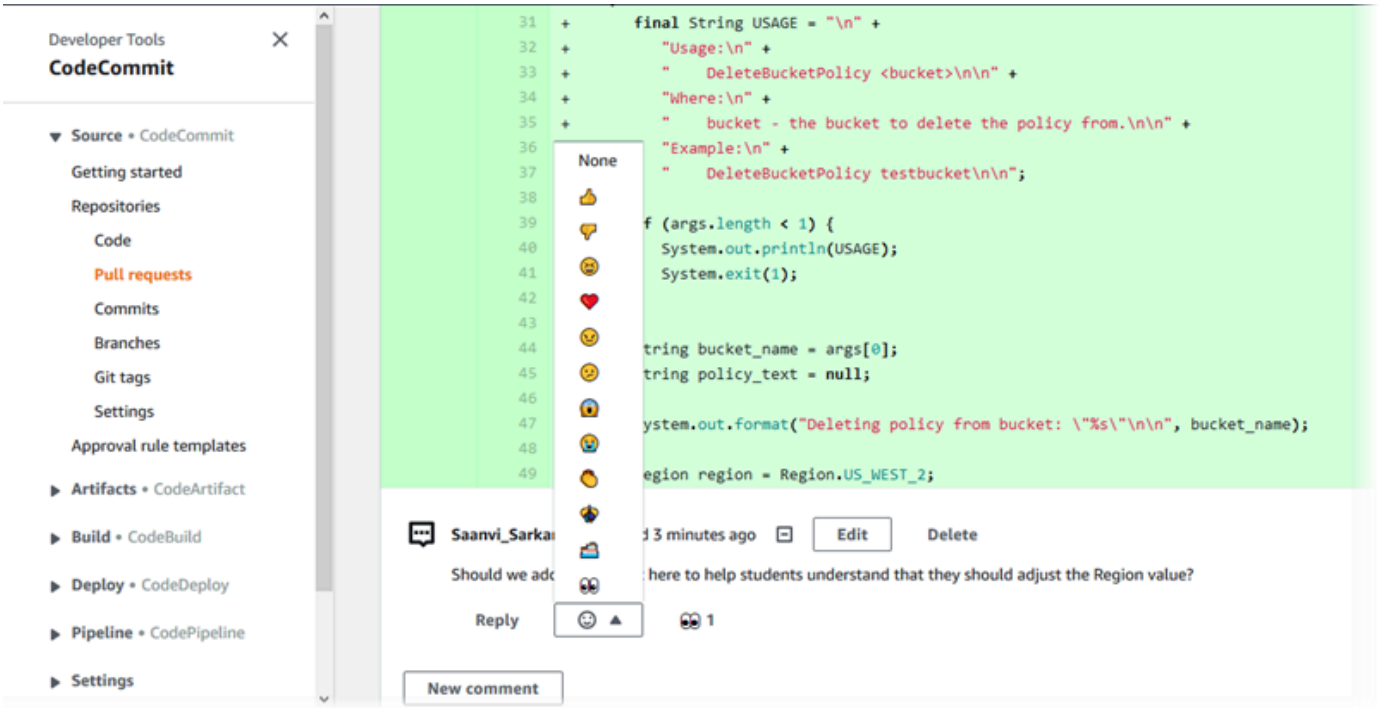


- To add a comment to a changed line in the pull request, in **Changes**, go to the line you want to comment on. Choose the comment icon

  that appears for that line, enter a comment, and then choose **Save**.

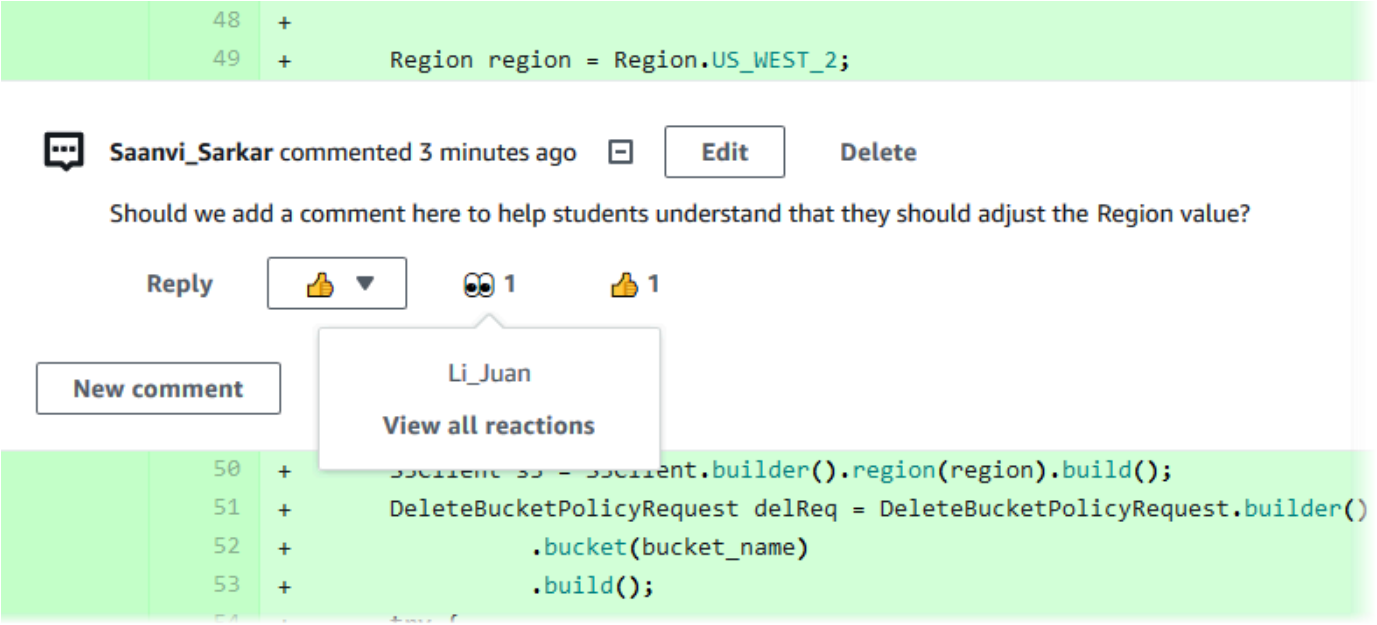


7. To reply to comments on a commit, in **Changes** or **Activity**, choose **Reply**. You can reply with text and with emojis.



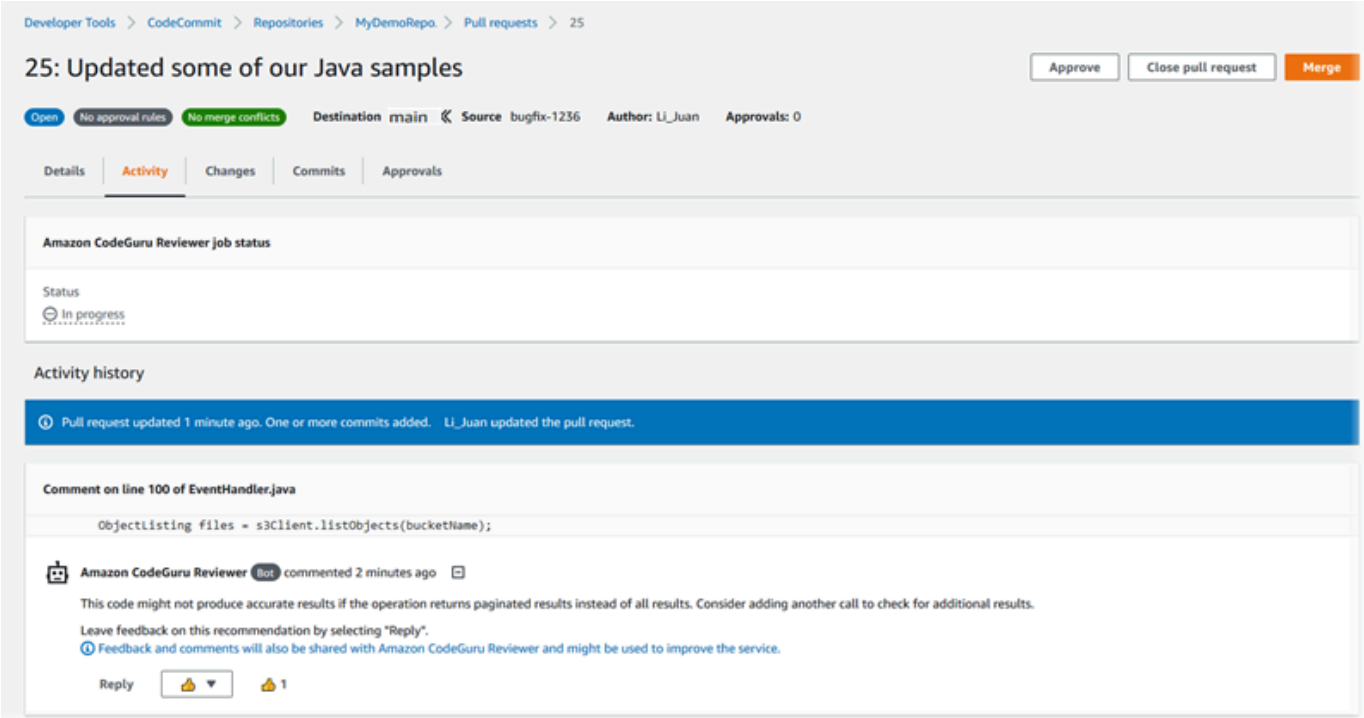
You can view the names of those who responded with a particular emoji reaction reply by choosing it. To view all emoji reactions and information about who responded with what emojis, choose **View all reactions**. If you've responded with an emoji to a comment, your response is shown in the icon for the emoji reaction button.

> **Note**
>
> Reaction counts displayed in the console are accurate as of the time the page was loaded. For the most current information about emoji reaction counts, either refresh the page, or choose **View all reactions**.



8. (Optional) To reply to a recommendation created by Amazon CodeGuru Reviewer, including providing feedback on the recommendation's quality, choose **Reply**. Use the reaction buttons to provide general information about whether you approve or disapprove the recommendation. Use the comment field to provide more details about your reaction.

> **Note**
>
> Amazon CodeGuru Reviewer is an automated code review service that uses program analysis and machine learning to detect common issues and recommend fixes in your Java or Python code.
>
> - You only see Amazon CodeGuru Reviewer comments if you have associated the repository with Amazon CodeGuru Reviewer, if the analysis is complete, and if the code in the pull request is Java or Python code. For more information, see Amazon CodeGuru Reviewer and AWS CodeCommit.
> - Amazon CodeGuru Reviewer comments only appear in the **Changes** tab if the comments were made on the most recent revision of the pull request. They always appear in the **Activity** tab.
> - While you can respond with any of the available emoji reactions to Amazon CodeGuru Reviewer recommendations, only thumbs up and thumbs down emoji reactions are used to evaluate the usefulness of the recommendation.

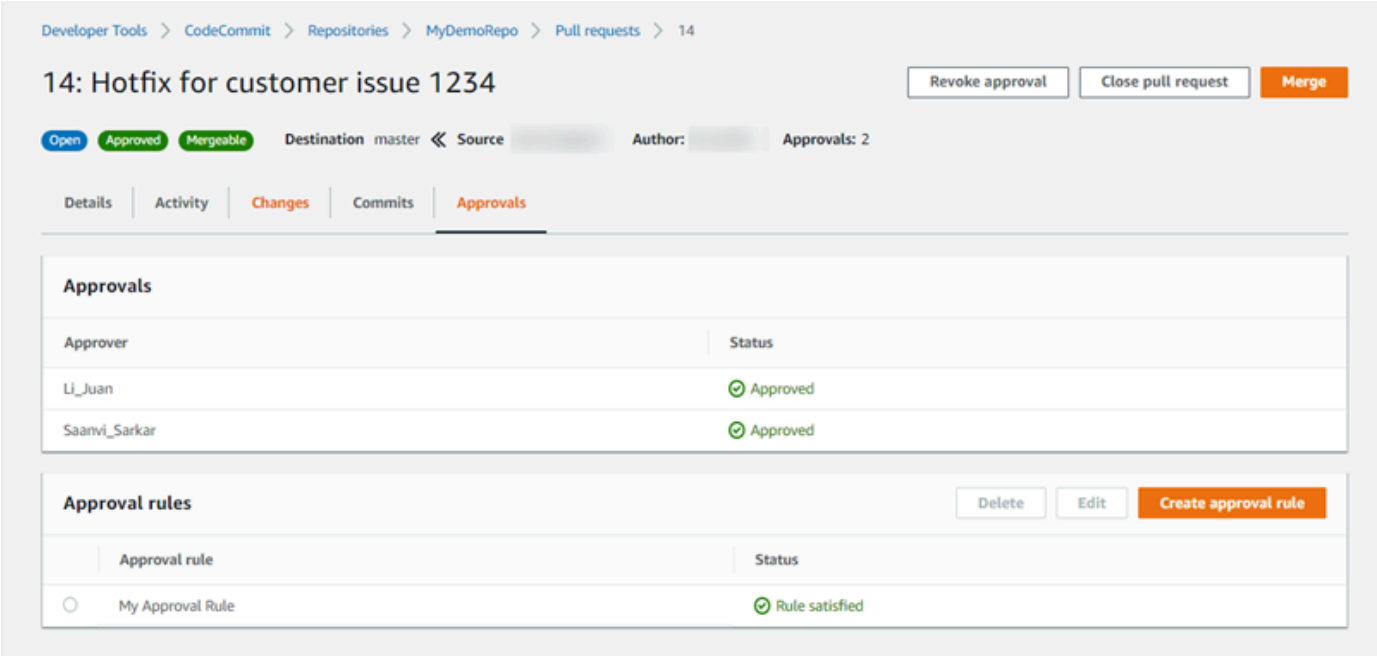


9. To approve the changes made in a pull request, choose **Approve**.

> **Note**
>
> You cannot approve a pull request that you created.

You can view approvals, approval rules for a pull request, and approval rules created by approval rule templates in **Approvals**. If you decide you do not want to approve the pull request after all, you can choose **Revoke approval**.

> **Note**
>
> You can only approve or revoke approval on an open pull request. You cannot approve or revoke approval on a pull request whose status is Merged or Closed.



Review pull requests (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

You can review pull requests with the following AWS CLI commands:

- **post-comment-for-pull-request**, to add a comment to a pull request
- **get-comments-for-pull-request**, to view comments left on a pull request
- **update-pull-request-approval-state**, to approve or revoke approval for a pull request
- **post-comment-reply**, to reply to a comment in a pull request

You can also use emojis with comments in a pull request with the following commands:

- To reply to a comment with an emoji, run **put-comment-reaction**.
- To view emoji reactions to a comment, run **get-comment-reactions**.

To use the AWS CLI to review pull requests in an CodeCommit repository

1. To add a comment to a pull request in a repository, run the **post-comment-for-pull-request** command, specifying:

   - The ID of the pull request (with the **--pull-request-id** option).
   - The name of the repository that contains the pull request (with the **--repository-name** option).
   - The full commit ID of the commit in the destination branch where the pull request is merged (with the **--before-commit-id** option).
   - The full commit ID of the commit in the source branch that is the current tip of the branch for the pull request when you post the comment (with the **--after-commit-id** option).
   - A unique, client-generated idempotency token (with the **--client-request-token** option).
   - The content of your comment (with the **--content** option).
   - A list of location information about where to place the comment, including:
     - The name of the file being compared, including its extension and subdirectory, if any (with the **filePath** attribute).
     - The line number of the change in a compared file (with the **filePosition** attribute).
     - Whether the comment on the change is "before" or "after" in the comparison between the source and destination branches (with the **relativeFileVersion** attribute).

   For example, use this command to add the comment *"These don't appear to be used anywhere. Can we remove them?"* on the change to the *ahs_count.py* file in a pull request with the ID of *47* in a repository named *MyDemoRepo*.

   ```
   aws codecommit post-comment-for-pull-request --pull-request-id "47" --
   repository-name MyDemoRepo --before-commit-id 317f8570EXAMPLE --after-
   commit-id 5d036259EXAMPLE --client-request-token 123Example --content
    "These don't appear to be used anywhere. Can we remove them?" --location
    filePath=ahs_count.py,filePosition=367,relativeFileVersion=AFTER
   ```

   If successful, this command produces output similar to the following.

   ```
   {
           "afterBlobId": "1f330709EXAMPLE",
           "afterCommitId": "5d036259EXAMPLE",
           "beforeBlobId": "80906a4cEXAMPLE",
           "beforeCommitId": "317f8570EXAMPLE",
   ```

```
        "comment": {
              "authorArn": "arn:aws:iam::111111111111:user/Saanvi_Sarkar",
              "clientRequestToken": "123Example",
              "commentId": "abcd1234EXAMPLEb5678efgh",
              "content": "These don't appear to be used anywhere. Can we remove
them?",
              "creationDate": 1508369622.123,
              "deleted": false,
              "lastModifiedDate": 1508369622.123,
              "callerReactions": [],
              "reactionCounts": []
           }
            "location": {
              "filePath": "ahs_count.py",
              "filePosition": 367,
              "relativeFileVersion": "AFTER"
            },
        "repositoryName": "MyDemoRepo",
        "pullRequestId": "47"
}
```

2. To view comments for a pull request, run the **get-comments-for-pull-request** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The system-generated ID of the pull request (with the `--pull-request-id` option).
   - (Optional) An enumeration token to return the next batch of the results (with the `--next-token` option).
   - (Optional) A non-negative integer to limit the number of returned results (with the `--max-results` option).

   For example, use this command to view comments for a pull request with an ID of 42.

   ```
   aws codecommit get-comments-for-pull-request --pull-request-id 42
   ```

   If successful, this command produces output similar to the following.

   ```
   {
      "commentsForPullRequestData": [
         {
   ```

```
        "afterBlobId": "1f330709EXAMPLE",
        "afterCommitId": "5d036259EXAMPLE",
        "beforeBlobId": "80906a4cEXAMPLE",
        "beforeCommitId": "317f8570EXAMPLE",
        "comments": [
           {
              "authorArn": "arn:aws:iam::111111111111:user/Saanvi_Sarkar",
              "clientRequestToken": "",
              "commentId": "abcd1234EXAMPLEb5678efgh",
              "content": "These don't appear to be used anywhere. Can we remove
them?",
              "creationDate": 1508369622.123,
              "deleted": false,
              "lastModifiedDate": 1508369622.123,
              "callerReactions": [],
              "reactionCounts":
               {
                 "THUMBSUP" : 6,
                 "CONFUSED" : 1
               }
           },
           {
              "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
              "clientRequestToken": "",
              "commentId": "442b498bEXAMPLE5756813",
              "content": "Good catch. I'll remove them.",
              "creationDate": 1508369829.104,
              "deleted": false,
              "lastModifiedDate": 150836912.273,
              "callerReactions": ["THUMBSUP"]
              "reactionCounts":
               {
                 "THUMBSUP" : 14
               }
            }
        ],
        "location": {
           "filePath": "ahs_count.py",
           "filePosition": 367,
           "relativeFileVersion": "AFTER"
        },
        "repositoryName": "MyDemoRepo",
        "pullRequestId": "42"
     }
```

```
    ],
    "nextToken": "exampleToken"
}
```

3. To approve or revoke approval for a pull request, run the **update-pull-request-approval-state** command, specifying:

   - The ID of the pull request (using the **--pull-request-id** option).
   - The revision ID of the pull request (using the **--revision-id option)**. You can get the current revision ID for a pull request by using the get-pull-request command.
   - The approval state you want to apply (using the **--approval-state**) option. Valid approval states include APPROVE and REVOKE.

   For example, use this command to approve a pull request with the ID of *27* and a revision ID of *9f29d167EXAMPLE*.

   ```
   aws codecommit update-pull-request-approval-state --pull-request-id 27 --revision-id 9f29d167EXAMPLE --approval-state "APPROVE"
   ```

   If successful, this command returns nothing.

4. To post a reply to a comment in a pull request, run the **post-comment-reply** command, specifying:

   - The system-generated ID of the comment to which you want to reply (with the **--in-reply-to** option).
   - A unique, client-generated idempotency token (with the **--client-request-token** option).
   - The content of your reply (with the **--content** option).

   For example, use this command to add the reply *"Good catch. I'll remove them."* to the comment with the system-generated ID of *abcd1234EXAMPLEb5678efgh*.

   ```
   aws codecommit post-comment-reply --in-reply-to abcd1234EXAMPLEb5678efgh --content "Good catch. I'll remove them." --client-request-token 123Example
   ```

   If successful, this command produces output similar to the following.

```
{
    "comment": {
        "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
        "clientRequestToken": "123Example",
        "commentId": "442b498bEXAMPLE5756813",
        "content": "Good catch. I'll remove them.",
        "creationDate": 1508369829.136,
        "deleted": false,
        "lastModifiedDate": 150836912.221,
        "callerReactions": [],
        "reactionCounts": []
    }
}
```

Update a pull request

You can update a pull request with further code changes by pushing commits to the source branch of an open pull request. For more information, see Create a commit in AWS CodeCommit.

You can use the AWS CodeCommit console or the AWS CLI to update the title or description of a pull request. You might want to update the pull request title or description because:

- Other users don't understand the description, or the original title is misleading.
- You want the title or description to reflect changes made to the source branch of an open pull request.

Update a pull request (console)

You can use the CodeCommit console to update the title and description of a pull request in an CodeCommit repository. To update the code in the pull request, push commits to the source branch of an open pull request.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to update a pull request.
3. In the navigation pane, choose **Pull requests**.

4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to update.
5. In the pull request, choose **Details**, and then choose **Edit details** to edit the title or description.

> **Note**
>
> You cannot update the title or description of a closed or merged pull request.

Update pull requests (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

You might also be interested in the following commands:

- **update-pull-request-approval-state**, to approve or revoke approval on a pull request.
- **create-pull-request-approval-rule**, to create an approval rule for a pull request.
- **delete-pull-request-approval-rule**, to delete an approval rule for a pull request.
- Create a commit using the AWS CLI or Create a commit using a Git client, to create and push additional code changes to the source branch of an open pull request.

To use the AWS CLI to update pull requests in a CodeCommit repository

1. To update the title of a pull request in a repository, run the **update-pull-request-title** command, specifying:

   - The ID of the pull request (with the **--pull-request-id** option).
   - The title of the pull request (with the **--title** option).

   For example, to update the title of a pull request with the ID of *47*:

   ```
   aws codecommit update-pull-request-title --pull-request-id 47 --title
    "Consolidation of global variables - updated review"
   ```

2. To update the description of a pull request, run the **update-pull-request-description** command, specifying:

- The ID of the pull request (with the **--pull-request-id** option).
- The description (with the **--description** option).

For example, to update the description of a pull request with the ID of *47* :

```
aws codecommit update-pull-request-description --pull-request-id 47 --description
 "Updated the pull request to remove unused global variable."
```

Edit or delete an approval rule for a pull request

When you have an approval rule on a pull request, you cannot merge that pull request until its conditions have been met. You can change the approval rules for pull requests to make it easier to satisfy their conditions, or to increase the rigor of reviews. You can change the number of users who must approve a pull request. You can also add, remove, or change the membership in an approval pool of users for the rule. Lastly, if you no longer want to use an approval rule for a pull request, you can delete it.

> **Note**
>
> You can also override approval rules for a pull request. For more information, see Override approval rules on a pull request.

You can use the AWS CodeCommit console or the AWS CLI to edit and delete approval rules for your repository.

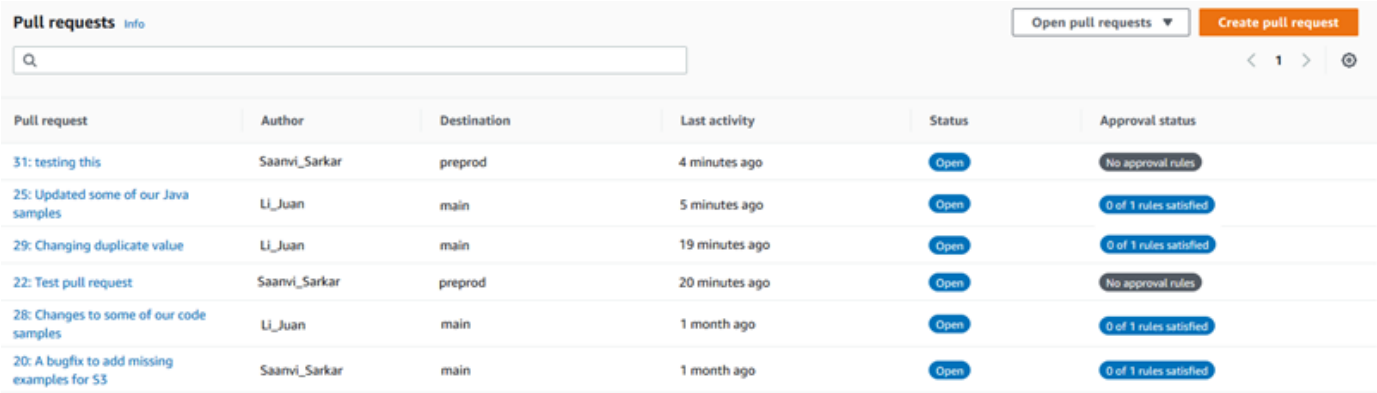
Topics

- Edit or delete an approval rule for a pull request (console)
- Edit or delete an approval rule for a pull request (AWS CLI)

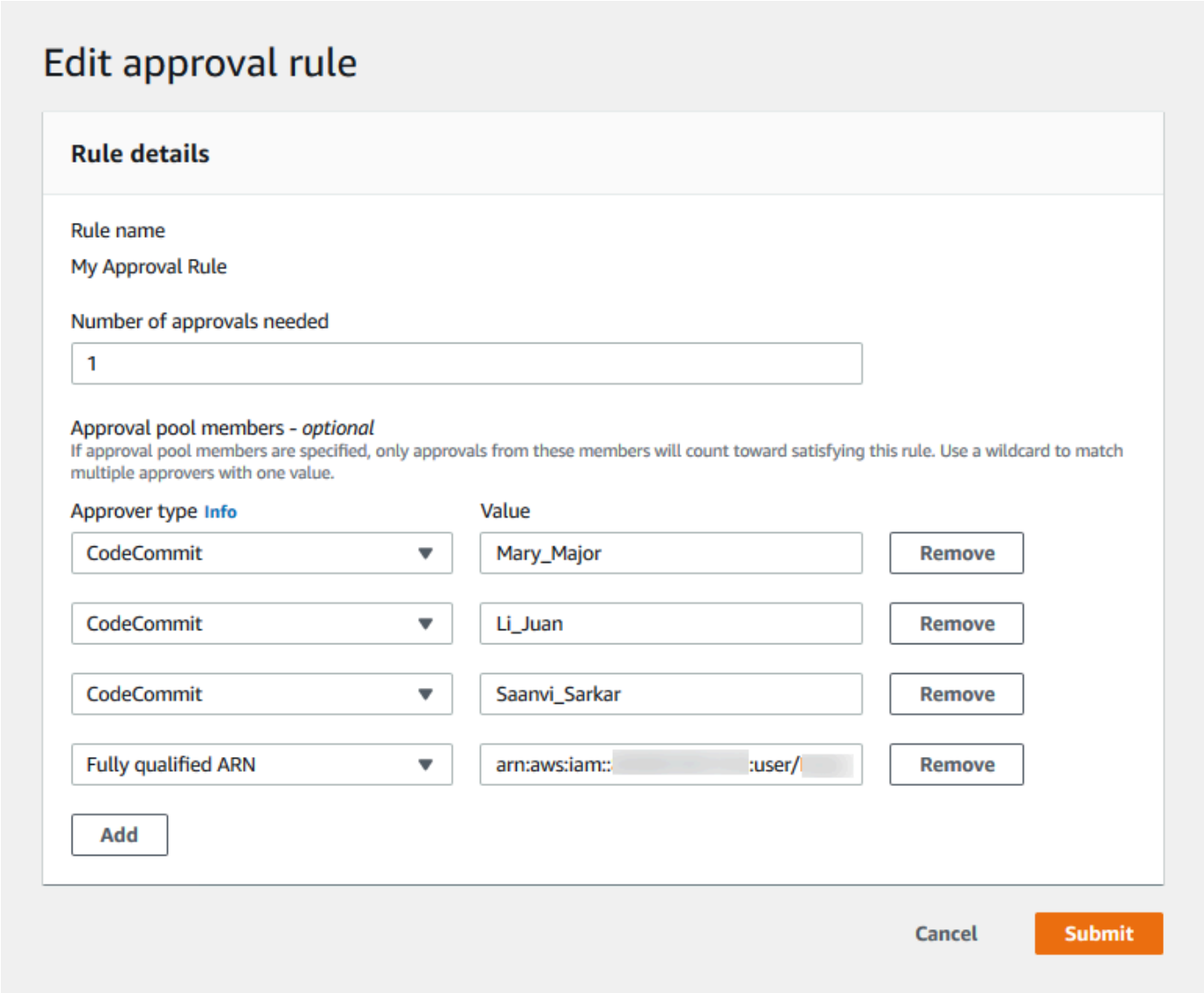
Edit or delete an approval rule for a pull request (console)

You can use the CodeCommit console to edit or delete an approval rule for a pull request in a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to edit or delete an approval rule for a pull request.
3. In the navigation pane, choose **Pull Requests**.
4. Choose the pull request where you want to edit or delete an approval rule. You can only edit and delete approval rules for open pull requests.



5. In the pull request, choose **Approvals**, and then choose the rule you want to edit or delete from the list. Do one of the following:
   - If you want to edit the rule, choose **Edit**.
   - If you want to delete the rule, choose **Delete**, and then follow the instructions for verifying the deletion of the rule.
6. In **Edit approval rule**, make the changes you want to the rule, and then choose **Submit**.



7. When you have finished configuring the approval rule, choose **Submit**.

Edit or delete an approval rule for a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

You can use the AWS CLI to edit the content of an approval rule and to delete an approval rule.

> **Note**
>
> You might also be interested in the following commands:

- update-pull-request-approval-state, to approve or revoke approval on a pull request.
- get-pull-request-approval-states, to view the approvals on the pull request.
- evaluate-pull-request-approval-rules, to determine whether approval rules for a pull request have had their conditions satisifed.

To use the AWS CLI to edit or delete an approval rule for a pull request in a CodeCommit repository

1. To edit an approval rule, run the **update-pull-request-approval-rule-content** command, specifying:

   - The ID of the pull request (with the **--id** option).
   - The name of the approval rule (with the **--approval-rule-name** option).
   - The content of the approval rule (with the **--approval-rule-content** option).

   This example updates an approval rule named *Require two approved approvers* for a pull request with the ID of *27*. The rule requires one user approval from an approval pool that includes any IAM user in the *123456789012* Amazon Web Services account:

   ```
   aws codecommit update-pull-request-approval-rule-content --pull-request-id 27
    --approval-rule-name "Require two approved approvers" --approval-rule-content
    "{Version: 2018-11-08, Statements: [{Type: \"Approvers\", NumberOfApprovalsNeeded:
    1, ApprovalPoolMembers:[\"CodeCommitApprovers:123456789012:user/*\"]}]}"
   ```

2. To delete an approval rule, run the **delete-pull-request-approval-rule** command, specifying:

   - The ID of the pull request (with the **--id** option).
   - The name of the approval rule (with the **--approval-rule-name** option).

   For example, to delete an approval rule with the name *My Approval Rule* for a pull request with the ID of *15*:

   ```
   aws codecommit delete-pull-request-approval-rule --pull-request-id 15 --approval-
   rule-name "My Approval Rule"
   ```

Override approval rules on a pull request

In the normal course of development, you want users to meet the conditions of approval rules before you merge pull requests. However, there might be times when you need to expedite merging a pull request. For example, you might want to put a bug fix into production, but no one in the approval pool is available to approve the pull request. In cases like these, you can choose to override the approval rules on a pull request. You can override all approval rules for a pull request, including those created specifically for the pull request and generated from an approval rule template. You cannot selectively override a specific approval rule, just all rules. After you have set aside the approval rule requirements by overriding the rules, you can merge the pull request to its destination branch.

When you override approval rules on a pull request, information about the user who overrode the rules is recorded in the activity for the pull request. This way you can go back into the history of a pull request and review who overrode the rules. You can also choose to revoke the override if the pull request is still open. After the pull request has been merged, you can no longer revoke the override.

Topics

- Override approval rules (console)
- Override approval rules (AWS CLI)

Override approval rules (console)

You can override the requirements of approval rules on a pull request in the console, as part of reviewing a pull request. If you change your mind, you can revoke your override, and the approval rule requirements are reapplied. You can only override approval rules or revoke an override if the pull request is still open. If it is merged or closed, you cannot change its override state.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository.
3. In the navigation pane, choose **Pull requests**. Choose the pull request where you want to override approval rule requirements, or revoke an override.

4. On the **Approvals** tab, choose **Override approval rules**. The requirements are set aside, and the button text changes to **Revoke override**. To reapply the approval rule requirements, choose **Revoke override**.

Override approval rules (AWS CLI)

You can use the AWS CLI to override approval rule requirements. You can also use it to view the override status for a pull request.

To override approval rule requirements on a pull request

1. At a terminal or command line, run the **override-pull-request-approval-rules** command, specifying:

   - The system-generated ID of the pull request.
   - The latest revision ID of the pull request. To view this information, use **get-pull-request**.
   - The status you want for the override, `OVERRIDE` or `REVOKE`. The `REVOKE` status removes the `OVERRIDE` status but is not saved.

   For example, to override approval rules on a pull request with an ID of **34** and a revision ID of **927df8d8EXAMPLE**:

   ```
   aws codecommit override-pull-request-approval-rules --pull-request-id 34 --
   revision-id 927df8d8dEXAMPLE --override-status OVERRIDE
   ```

2. If successful, this command returns nothing.
3. To revoke the override on a pull request with an ID of **34** and a revision ID of **927df8d8EXAMPLE**:

   ```
   aws codecommit override-pull-request-approval-rules --pull-request-id 34 --
   revision-id 927df8d8dEXAMPLE --override-status REVOKE
   ```

To get information about the override status of a pull request

1. At a terminal or command line, run the **get-pull-request-override-state** command, specifying:

   - The system-generated ID of the pull request.

- The latest revision ID of the pull request. To view this information, use **get-pull-request**.

For example, to view the override state for a pull request with an ID of **34** and a revision ID of **927df8d8EXAMPLE**:

```
aws codecommit get-pull-request-override-state --pull-request-id 34 --revision-
id 927df8d8dEXAMPLE
```

2. If successful, this command produces output similar to the following:

```
{
    "overridden": true,
    "overrider": "arn:aws:iam::123456789012:user/Mary_Major"
}
```

Merge a pull request in an AWS CodeCommit repository

After your code has been reviewed and all approval rules (if any) on the pull request have been satisfied, you can merge a pull request in one of several ways:

- You can use the console to merge your source branch to the destination branch using one of the available merge strategies, which also closes the pull request. You can also resolve any merge conflicts in the console. The console displays a message that indicates if the pull request is mergeable or if conflicts must be resolved. When all conflicts are resolved and you choose **Merge**, the merge is performed using the merge strategy that you choose. Fast-forward is the default merge strategy, which is the default option for Git. Depending on the state of the code in the source and destination branches, that strategy might not be available, but other options might be, such as squash or 3-way.
- You can use the AWS CLI to merge and close the pull request using the fast-forward, squash, or 3-way merge strategy.
- On your local computer, you can use the **git merge** command to merge the source branch into the destination branch, and then push your merged code to the destination branch. This approach has drawbacks that you should carefully consider. It merges the pull request regardless if the requirements for approval rules on the pull request have been satisfied, circumventing those controls. Merging and pushing the destination branch also closes the pull

request automatically if the pull request is merged using the fast-forward merge strategy. One advantage of this approach is that the **git merge** command allows you to choose merge options or strategies that are not available in the CodeCommit console. For more information about **git merge** and merge options, see git-merge or your Git documentation.

CodeCommit closes a pull request automatically if either the source or destination branch of the pull request is deleted.

Topics

- Merge a pull request (console)
- Merge a pull request (AWS CLI)

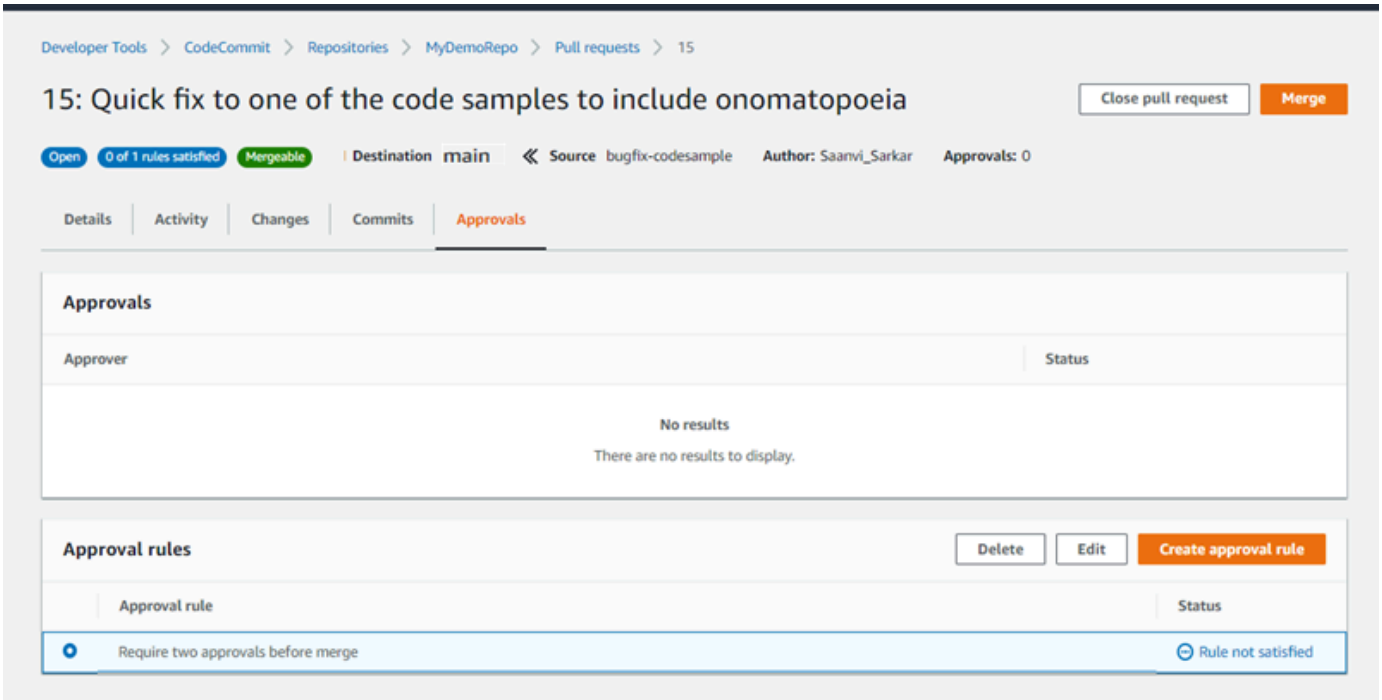
Merge a pull request (console)

You can use the CodeCommit console to merge a pull request in a CodeCommit repository. After the status of a pull request is changed to **Merged**, it no longer appears in the list of open pull requests. A merged pull request is categorized as closed. It cannot be changed back to **Open**, but users can still comment on the changes and reply to comments. After a pull request is merged or closed, you cannot approve it, revoke approval for it, or override the approval rules applied to the pull request.

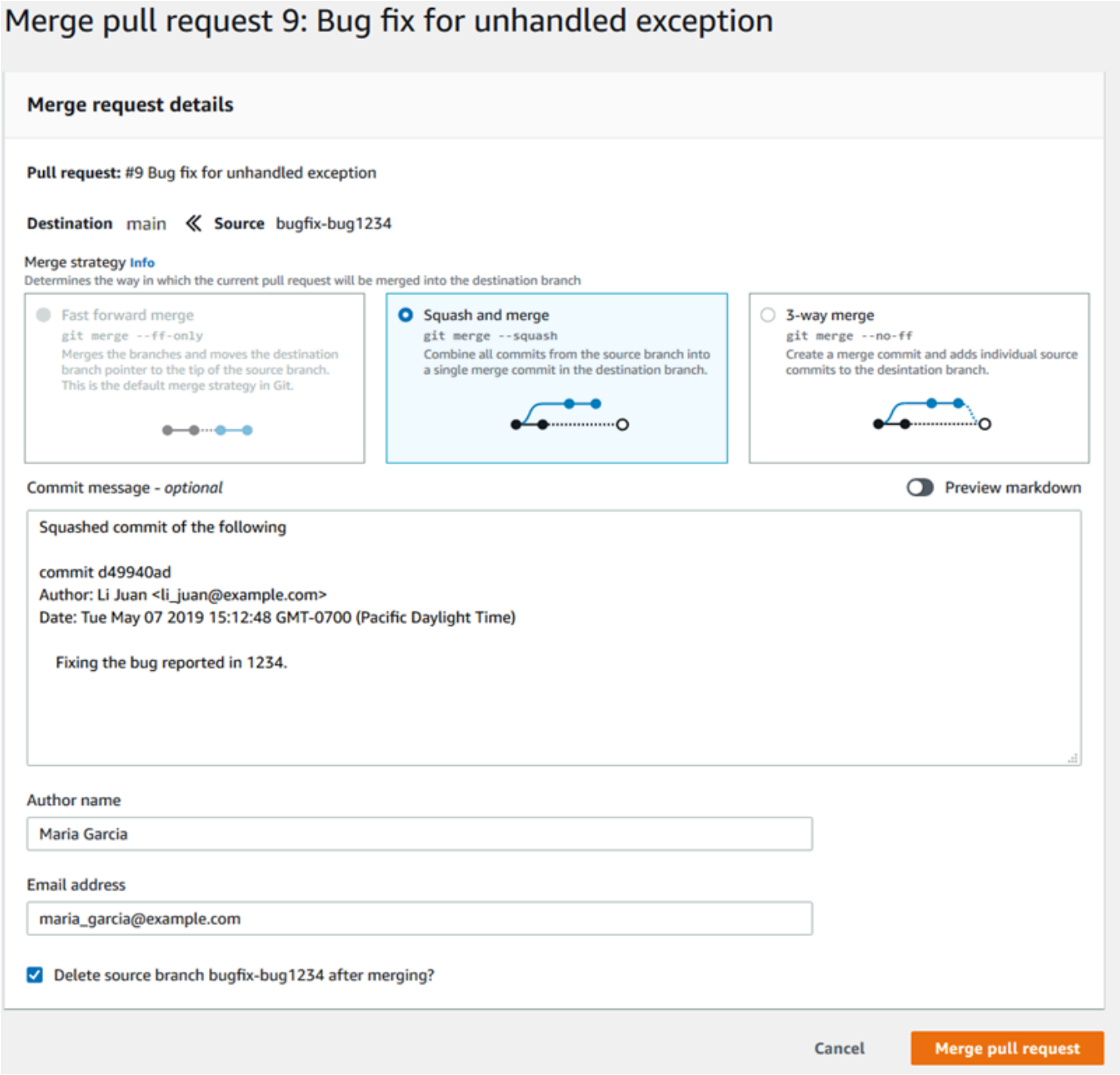
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository.
3. In the navigation pane, choose **Pull requests**.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to merge.
5. In the pull request, choose **Approvals**. Review the list of approvers, and verify that all approval rules (if any) have had their conditions satisfied. You cannot merge a pull request if one or more approval rules have the status of **Rule not satisfied**. If no one has approved the pull request, consider whether you want to merge it, or whether you want to wait for approvals.

> **Note**
>
> If an approval rule was created for a pull request, you can edit it or delete it to unblock the merge. If the approval rule was created with an approval rule template, you

cannot edit or delete it. You can only choose to override the requirements. For more information, see Override approval rules on a pull request.



6. Choose **Merge**.
7. In the pull request, choose between the available merge strategies. Merge strategies that cannot be applied appear greyed out. If no merge strategies are available, you can choose to manually resolve conflicts in the CodeCommit console, or you can resolve them locally using your Git client. For more information, see Resolve conflicts in a pull request in an AWS CodeCommit repository.



- A fast-forward merge moves the reference for the destination branch forward to the most recent commit of the source branch. This is the default behavior of Git when possible. No merge commit is created, but all commit history from the source branch is retained as if it had occurred in the destination branch. Fast-forward merges do not appear as a branch merge in the commit visualizer view of the destination branch's history because no merge commit is created. The tip of the source branch is fast-forwarded to the tip of the destination branch.

- A squash merge creates one commit that contains the changes in the source branch and applies that single squashed commit to the destination branch. By default, the commit message for that squash commit contains all the commit messages of the changes in the source branch. No individual commit history of the branch changes is retained. This can help simplify your repository history while still retaining a graphical representation of the merge in the commit visualizer view of the destination branch's history.
- A three-way merge creates a merge commit for the merge in the destination branch, but also retains the individual commits made in the source branch as part of the history of the destination branch. This can help maintain a complete history of changes to your repository.

8. If you choose the squash or 3-way merge strategy, review the automatically generated commit message and modify it if you want to change the information. Add your name and email address for the commit history.
9. (Optional) Clear the option to delete the source branch as part of the merge. The default is to delete the source branch when a pull request is merged.
10. Choose **Merge pull request** to complete the merge.

Merge a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to merge pull requests in a CodeCommit repository

1. To evaluate whether a pull request has had all of its approval rules satisfied and is ready to be merged, run the **evaluate-pull-request-approval-rules** command, specifying:

   - The ID of the pull request (using the **--pull-request-id** option).
   - The revision ID of the pull request (using the **--revision-id option)**. You can get the current revision ID for a pull request by using the **get-pull-request** command.

   For example, to evaluate the state of approval rules on a pull request with an ID of *27* and a revision ID of *9f29d167EXAMPLE*:

   ```
   aws codecommit evaluate-pull-request-approval-rules --pull-request-id 27 --
   revision-id 9f29d167EXAMPLE
   ```

If successful, this command produces output similar to the following:

```
{
    "evaluation": {
        "approved": false,
        "approvalRulesNotSatisfied": [
            "Require two approved approvers"
        ],
        "overridden": false,
        "approvalRulesSatisfied": []
    }
}
```

> **Note**
>
> This output indicates that a pull request is not mergeable because the requirements of an approval rule have not been satisfied. To merge this pull request, you can have reviewers approve it to meet the conditions of the rule. Depending on your permissions and how the rule was created, you might also be able to edit, override, or delete the rule. For more information, see Review a pull request, Override approval rules on a pull request, and Edit or delete an approval rule for a pull request.

2. To merge and close a pull request using the fast-forward merge strategy, run the **merge-pull-request-by-fast-forward** command, specifying:

   - The ID of the pull request (with the **--pull-request-id** option).
   - The full commit ID of the tip of the source branch (with the **--source-commit-id** option).
   - The name of the repository (with the **--repository-name** option).

   For example, to merge and close a pull request with the ID of *47* and a source commit ID of *99132ab0EXAMPLE* in a repository named *MyDemoRepo*:

   ```
   aws codecommit merge-pull-request-by-fast-forward --pull-request-id 47 --source-commit-id 99132ab0EXAMPLE --repository-name MyDemoRepo
   ```

   If successful, this command produces output similar to the following:

```
{
    "pullRequest": {
        "approvalRules": [
            {
                "approvalRuleContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 1,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
                "approvalRuleId": "dd8b17fe-EXAMPLE",
                "approvalRuleName": "I want one approver for this pull request",
                "creationDate": 1571356106.936,
                "lastModifiedDate": 571356106.936,
                "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
                "ruleContentSha256": "4711b576EXAMPLE"
            }
        ],
        "authorArn": "arn:aws:iam::123456789012:user/Li_Juan",
        "clientRequestToken": "",
        "creationDate": 1508530823.142,
        "description": "Review the latest changes and updates to the global
 variables",
        "lastActivityDate": 1508887223.155,
        "pullRequestId": "47",
        "pullRequestStatus": "CLOSED",
        "pullRequestTargets": [
            {
                "destinationCommit": "9f31c968EXAMPLE",
                "destinationReference": "refs/heads/main",
                "mergeMetadata": {
                    "isMerged": true,
                    "mergedBy": "arn:aws:iam::123456789012:user/Mary_Major"
                },
                "repositoryName": "MyDemoRepo",
                "sourceCommit": "99132ab0EXAMPLE",
                "sourceReference": "refs/heads/variables-branch"
            }
        ],
        "title": "Consolidation of global variables"
    }
}
```


3. To merge and close a pull request using the squash merge strategy, run the **merge-pull-request-by-squash** command, specifying:

- The ID of the pull request (with the **--pull-request-id** option).
- The full commit ID of the tip of the source branch (with the **--source-commit-id** option).
- The name of the repository (with the **--repository-name** option).
- The level of conflict detail you want to use (with the **--conflict-detail-level** option). If unspecified, the default **FILE_LEVEL** is used.
- The conflict resolution strategy you want to use (with the **--conflict-resolution-strategy** option). If unspecified, this defaults to NONE, and conflicts must be resolved manually.
- The commit message to include (with the **--commit-message** option).
- The name to use for the commit (with the **--author-name** option).
- The email address to use for the commit (with the **--email** option).
- Whether to keep any empty folders (with the **--keep-empty-folders** option).

The following example merges and closes a pull request with the ID of *47* and a source commit ID of *99132ab0EXAMPLE* in a repository named *MyDemoRepo*. It uses the conflict detail of LINE_LEVEL and the conflict resolution strategy of ACCEPT_SOURCE:

```
aws codecommit merge-pull-request-by-squash --pull-request-id 47 --source-commit-
id 99132ab0EXAMPLE --repository-name MyDemoRepo --conflict-detail-level LINE_LEVEL
 --conflict-resolution-strategy ACCEPT_SOURCE --author-name "Jorge Souza" --email
 "jorge_souza@example.com" --commit-message "Merging pull request 47 by squash and
 accepting source in merge conflicts"
```

If successful, this command produces the same kind of output as merging by fast-forward, output similar to the following:

```
{
    "pullRequest": {
        "approvalRules": [
            {
                "approvalRuleContent": "{\"Version\": \"2018-11-08\",
\"DestinationReferences\": [\"refs/heads/main\"],\"Statements\": [{\"Type
\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers\":
 [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
                "approvalRuleId": "dd8b17fe-EXAMPLE",
                "approvalRuleName": "2-approver-rule-for-main",
                "creationDate": 1571356106.936,
```

```
                "lastModifiedDate": 571356106.936,
                "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
                "originApprovalRuleTemplate": {
                    "approvalRuleTemplateId": "dd8b17fe-EXAMPLE",
                    "approvalRuleTemplateName": "2-approver-rule-for-main"
                },
                "ruleContentSha256": "4711b576EXAMPLE"
            }
        ],
        "authorArn": "arn:aws:iam::123456789012:user/Li_Juan",
        "clientRequestToken": "",
        "creationDate": 1508530823.142,
        "description": "Review the latest changes and updates to the global
 variables",
        "lastActivityDate": 1508887223.155,
        "pullRequestId": "47",
        "pullRequestStatus": "CLOSED",
        "pullRequestTargets": [
            {
                "destinationCommit": "9f31c968EXAMPLE",
                "destinationReference": "refs/heads/main",
                "mergeMetadata": {
                    "isMerged": true,
                    "mergedBy": "arn:aws:iam::123456789012:user/Mary_Major"
                },
                "repositoryName": "MyDemoRepo",
                "sourceCommit": "99132ab0EXAMPLE",
                "sourceReference": "refs/heads/variables-branch"
            }
        ],
        "title": "Consolidation of global variables"
    }
}
```

4. To merge and close a pull request using the three-way merge strategy, run the **merge-pull-request-by-three-way** command, specifying:

   - The ID of the pull request (with the **--pull-request-id** option).
   - The full commit ID of the tip of the source branch (with the **--source-commit-id** option).
   - The name of the repository (with the **--repository-name** option).
   - The level of conflict detail you want to use (with the **--conflict-detail-level** option). If unspecified, the default **`FILE_LEVEL`** is used.

- The conflict resolution strategy you want to use (with the **--conflict-resolution-strategy** option). If unspecified, this defaults to NONE, and conflicts must be resolved manually.
- The commit message to include (with the **--commit-message** option).
- The name to use for the commit (with the **--author-name** option).
- The email address to use for the commit (with the **--email** option).
- Whether to keep any empty folders (with the **--keep-empty-folders** option).

The following example merges and closes a pull request with the ID of *47* and a source commit ID of *99132ab0EXAMPLE* in a repository named *MyDemoRepo*. It uses the default options for conflict detail and conflict resolution strategy:

```
aws codecommit merge-pull-request-by-three-way --pull-request-id 47 --source-
commit-id 99132ab0EXAMPLE --repository-name MyDemoRepo --author-name "Maria Garcia"
 --email "maria_garcia@example.com" --commit-message "Merging pull request 47 by
 three-way with default options"
```

If successful, this command produces the same kind of output as merging by fast-forward, similar to the following:

```
{
    "pullRequest": {
        "approvalRules": [
            {
                "approvalRuleContent": "{\"Version\": \"2018-11-08\",
\"DestinationReferences\": [\"refs/heads/main\"],\"Statements\": [{\"Type
\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers\":
 [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
                "approvalRuleId": "dd8b17fe-EXAMPLE",
                "approvalRuleName": "2-approver-rule-for-main",
                "creationDate": 1571356106.936,
                "lastModifiedDate": 571356106.936,
                "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
                "originApprovalRuleTemplate": {
                    "approvalRuleTemplateId": "dd8b17fe-EXAMPLE",
                    "approvalRuleTemplateName": "2-approver-rule-for-main"
                },
                "ruleContentSha256": "4711b576EXAMPLE"
            }
        ],
```

```
        "authorArn": "arn:aws:iam::123456789012:user/Li_Juan",
        "clientRequestToken": "",
        "creationDate": 1508530823.142,
        "description": "Review the latest changes and updates to the global
 variables",
        "lastActivityDate": 1508887223.155,
        "pullRequestId": "47",
        "pullRequestStatus": "CLOSED",
        "pullRequestTargets": [
            {
                "destinationCommit": "9f31c968EXAMPLE",
                "destinationReference": "refs/heads/main",
                "mergeMetadata": {
                    "isMerged": true,
                    "mergedBy": "arn:aws:iam::123456789012:user/Mary_Major"
                },
                "repositoryName": "MyDemoRepo",
                "sourceCommit": "99132ab0EXAMPLE",
                "sourceReference": "refs/heads/variables-branch"
            }
        ],
        "title": "Consolidation of global variables"
    }
}
```


Resolve conflicts in a pull request in an AWS CodeCommit repository

If your pull request has conflicts and cannot be merged, you can try to resolve the conflicts in one of several ways:

- On your local computer, you can use the **git diff** command to find the conflicts between the two branches and make changes to resolve them. You can also use a difference tool or other software to help you find and resolve differences. Once you have resolved them to your satisfaction, you can push your source branch with the changes that contain the resolved conflicts, which updates the pull request. For more information about **git diff** and **git difftool**, see your Git documentation.

- In the console, you can choose **Resolve conflicts**. This opens a plain-text editor that shows conflicts in a similar way as the **git diff** command. You can manually review the conflicts in each file that contain them, make changes, and then update the pull request with your changes.
- In the AWS CLI, you can use the AWS CLI to get information about merge conflicts and create an unreferenced merge commit to test a merge.

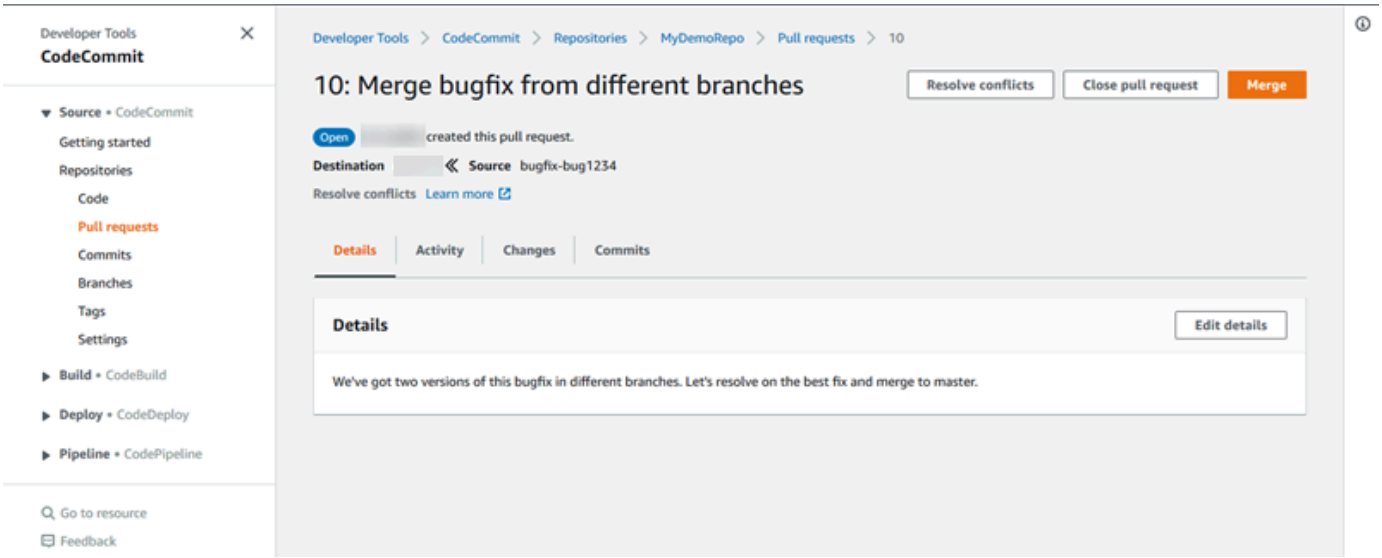
Topics

- Resolve conflicts in a pull request (console)
- Resolve conflicts in a pull request (AWS CLI)

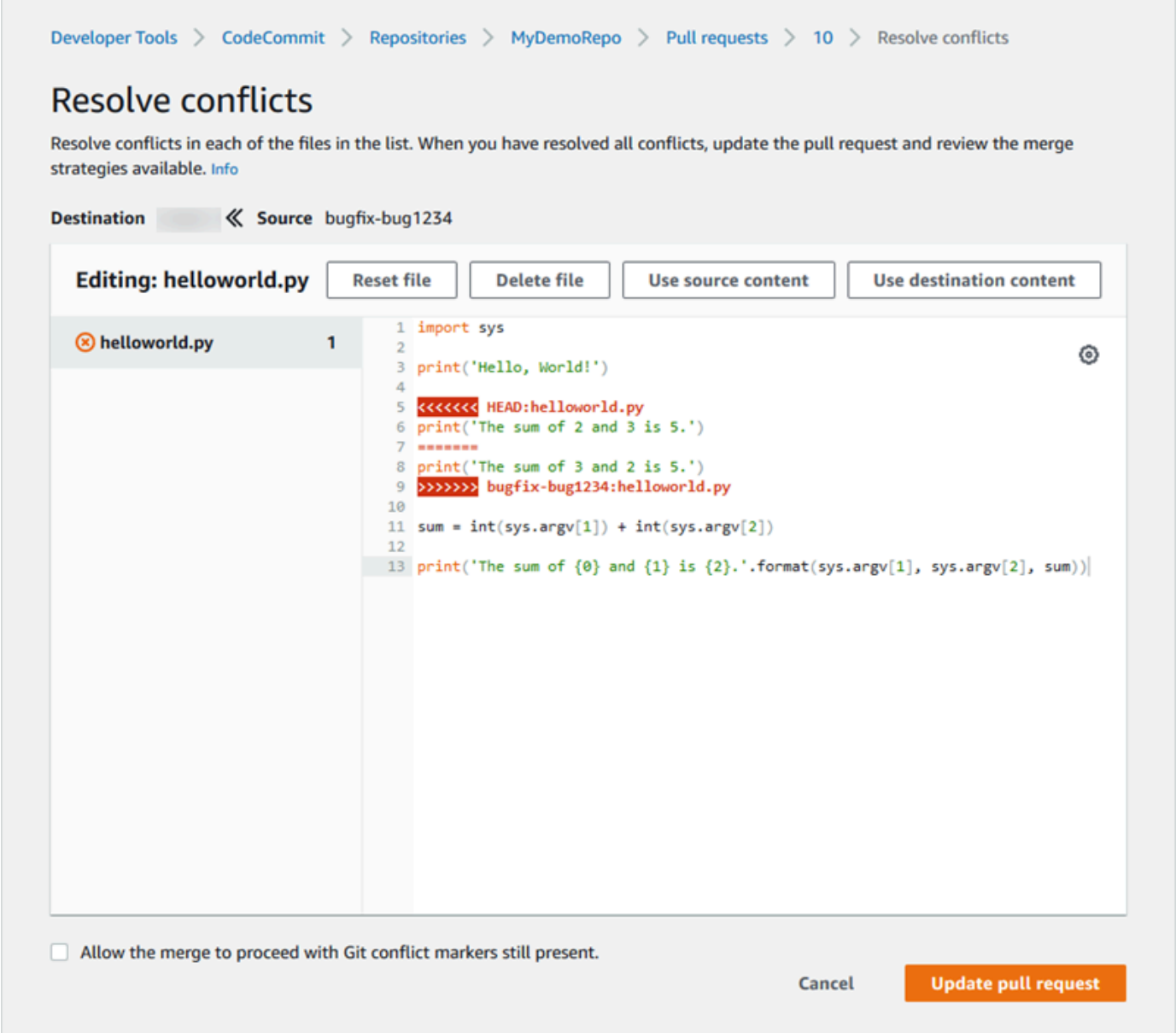
Resolve conflicts in a pull request (console)

You can use the CodeCommit console to resolve conflicts in a pull request in a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository.
3. In the navigation pane, choose **Pull requests**.
4. By default, a list of all open pull requests is displayed. Choose the open pull request that you want to merge but it contains conflicts.
5. In the pull request, choose **Resolve conflicts**. This option only appears if there are conflicts that must be resolved before the pull request can be merged.



6. A conflict resolution window opens listing each file that has conflicts that must be resolved. Choose each file in the list to review the conflicts, and make any necessary changes until all conflicts have been resolved.



- You can choose to use the source file contents, the destination file contents, or if the file is not a binary file, to manually edit the contents of a file so it contains only the changes you want. Standard git diff markers are used to show the conflicts between the destination (HEAD) and source branches in the file.
- If a file is a binary file, a Git submodule, or if there is a file/folder name conflict, you must choose to use the source file or the destination file to resolve the conflicts. You cannot view or edit binary files in the CodeCommit console.

- If there are file mode conflicts, you see the option to resolve that conflict by choosing between the file mode of the source file and the file mode of the destination file.
- If you decide you want to discard your changes for a file and restore it to its conflicted state, choose **Reset file**. This allows you to resolve the conflicts in a different way.

7. When you are satisfied with your changes, choose **Update pull request**.

> **Note**
>
> You must resolve all conflicts in all files before you can successfully update the pull request with your changes.

8. The pull request is updated with your changes and mergeable. You see the merge page. You can choose to merge the pull request at this time, or you can return to the list of pull requests.

Resolve conflicts in a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

No single AWS CLI command enables you to resolve conflicts in a pull request and merge that request. However, you can use individual commands to discover conflicts, attempt to resolve them, and test whether a pull request is mergeable. You can use:

- **get-merge-options**, to find out what merge options are available for a merge between two commit specifiers.
- **get-merge-conflicts**, to return a list of files with merge conflicts in a merge between two commit specifiers.
- **batch-describe-merge-conflicts**, to get information about all merge conflicts in files in a merge between two commits using a specified merge strategy.
- **describe-merge-conflicts**, to get detailed information about merge conflicts for a specific file between two commits using a specified merge strategy.
- **create-unreferenced-merge-commit**, to test the result of merging two commit specifiers using a specified merge strategy.

1.

To discover what merge options are available for a merge between two commit specifiers, run the **get-merge-options** command, specifying:

- A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
- A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).
- The name of the repository (with the **--repository-name** option).
- (Optional) A conflict resolution strategy to use (with the **--conflict-resolution-strategy** option).
- (Optional) The level of detail you want about any conflicts (with the **--conflict-detail-level** option).

For example, to determine the merge strategies available for merging a source branch named *bugfix-1234* with a destination branch named *main* in a repository named *MyDemoRepo*:

```
aws codecommit get-merge-options --source-commit-specifier bugfix-1234 --
destination-commit-specifier main --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "mergeOptions": [
        "FAST_FORWARD_MERGE",
        "SQUASH_MERGE",
        "THREE_WAY_MERGE"
    ],
    "sourceCommitId": "d49940adEXAMPLE",
    "destinationCommitId": "86958e0aEXAMPLE",
    "baseCommitId": "86958e0aEXAMPLE"
}
```

2. To get a list of files that contain merge conflicts in a merge between two commit specifiers, run the **get-merge-conflicts** command, specifying:

- A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
- A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).

- The name of the repository (with the **--repository-name** option).
- The merge option you want to use (with the **--merge-option** option).
- (Optional) The level of detail you want about any conflicts (with the **--conflict-detail-level** option).
- (Optional) A conflict resolution strategy to use (with the **--conflict-resolution-strategy** option).
- (Optional) The maximum number of files with conflicts to return (with the **--max-conflict-files** option).

For example, to get a list of files that contain conflicts in a merge between a source branch named feature-randomizationfeature and a destination branch named main using the three-way merge strategy in a repository named MyDemoRepo:

```
aws codecommit get-merge-conflicts --source-commit-specifier feature-
randomizationfeature --destination-commit-specifier main --merge-option
 THREE_WAY_MERGE --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "mergeable": false,
    "destinationCommitId": "86958e0aEXAMPLE",
    "sourceCommitId": "6ccd57fdEXAMPLE",
    "baseCommitId": "767b6958EXAMPLE",
    "conflictMetadataList": [
        {
            "filePath": "readme.md",
            "fileSizes": {
                "source": 139,
                "destination": 230,
                "base": 85
            },
            "fileModes": {
                "source": "NORMAL",
                "destination": "NORMAL",
                "base": "NORMAL"
            },
            "objectTypes": {
                "source": "FILE",
```

```
                "destination": "FILE",
                "base": "FILE"
            },
            "numberOfConflicts": 1,
            "isBinaryFile": {
                "source": false,
                "destination": false,
                "base": false
            },
            "contentConflict": true,
            "fileModeConflict": false,
            "objectTypeConflict": false,
            "mergeOperations": {
                "source": "M",
                "destination": "M"
            }
        }
    ]
}
```

3. To get information about merge conflicts in all files or a subset of files in a merge between two commit specifiers, run the **batch-describe-merge-conflicts** command, specifying:

- A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
- A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).
- The merge option you want to use (with the **--merge-option** option).
- The name of the repository (with the **--repository-name** option).
- (Optional) A conflict resolution strategy to use (with the **--conflict-resolution-strategy** option).
- (Optional) The level of detail you want about any conflicts (with the **--conflict-detail-level** option).
- (Optional) The maximum number of merge hunks to return (with the **--max-merge-hunks** option).
- (Optional) The maximum number of files with conflicts to return (with the **--max-conflict-files** option).
- (Optional) The path of target files to use to describe the conflicts (with the **--file-paths** option).

For example, to determine the merge conflicts for merging a source branch named *feature-randomizationfeature* with a destination branch named *main* using the *THREE_WAY_MERGE* strategy in a repository named *MyDemoRepo*:

```
aws codecommit batch-describe-merge-conflicts --source-commit-specifier feature-
randomizationfeature --destination-commit-specifier main --merge-
option THREE_WAY_MERGE --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "conflicts": [
        {
            "conflictMetadata": {
                "filePath": "readme.md",
                "fileSizes": {
                    "source": 139,
                    "destination": 230,
                    "base": 85
                },
                "fileModes": {
                    "source": "NORMAL",
                    "destination": "NORMAL",
                    "base": "NORMAL"
                },
                "objectTypes": {
                    "source": "FILE",
                    "destination": "FILE",
                    "base": "FILE"
                },
                "numberOfConflicts": 1,
                "isBinaryFile": {
                    "source": false,
                    "destination": false,
                    "base": false
                },
                "contentConflict": true,
                "fileModeConflict": false,
                "objectTypeConflict": false,
                "mergeOperations": {
                    "source": "M",
```

```
                    "destination": "M"
                }
            },
            "mergeHunks": [
                {
                    "isConflict": true,
                    "source": {
                        "startLine": 0,
                        "endLine": 3,
                        "hunkContent": "VGhpcyBpEXAMPLE=="
                    },
                    "destination": {
                        "startLine": 0,
                        "endLine": 1,
                        "hunkContent": "VXNlIHRoEXAMPLE="
                    }
                }
            ]
        }
    ],
    "errors": [],
    "destinationCommitId": "86958e0aEXAMPLE",
    "sourceCommitId": "6ccd57fdEXAMPLE",
    "baseCommitId": "767b6958EXAMPLE"
}
```

4. To get detailed information about any merge conflicts for a specific file in a merge between two commit specifiers, run the **describe-merge-conflicts** command, specifying:

   - A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
   - A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).
   - The merge option you want to use (with the **--merge-option** option).
   - The path of target file to use to describe the conflicts (with the **--file-path** option).
   - The name of the repository (with the **--repository-name** option).
   - (Optional) A conflict resolution strategy to use (with the **--conflict-resolution-strategy** option).
   - (Optional) The level of detail you want about any conflicts (with the **--conflict-detail-level** option).

- (Optional) The maximum number of merge hunks to return (with the **--max-merge-hunks** option).
- (Optional) The maximum number of files with conflicts to return (with the **--max-conflict-files** option).

For example, to determine the merge conflicts for a file named *readme.md* in a source branch named *feature-randomizationfeature* with a destination branch named *main* using the *THREE_WAY_MERGE* strategy in a repository named *MyDemoRepo*:

```
aws codecommit describe-merge-conflicts --source-commit-specifier feature-
randomizationfeature --destination-commit-specifier main --merge-
option THREE_WAY_MERGE --file-path readme.md --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "conflictMetadata": {
        "filePath": "readme.md",
        "fileSizes": {
            "source": 139,
            "destination": 230,
            "base": 85
        },
        "fileModes": {
            "source": "NORMAL",
            "destination": "NORMAL",
            "base": "NORMAL"
        },
        "objectTypes": {
            "source": "FILE",
            "destination": "FILE",
            "base": "FILE"
        },
        "numberOfConflicts": 1,
        "isBinaryFile": {
            "source": false,
            "destination": false,
            "base": false
        },
        "contentConflict": true,
```

```
        "fileModeConflict": false,
        "objectTypeConflict": false,
        "mergeOperations": {
            "source": "M",
            "destination": "M"
        }
    },
    "mergeHunks": [
        {
            "isConflict": true,
            "source": {
                "startLine": 0,
                "endLine": 3,
                "hunkContent": "VGhpcyBpEXAMPLE=="
            },
            "destination": {
                "startLine": 0,
                "endLine": 1,
                "hunkContent": "VXNlIHRoEXAMPLE="
            }
        }
    ],
    "destinationCommitId": "86958e0aEXAMPLE",
    "sourceCommitId": "6ccd57fdEXAMPLE",
    "baseCommitId": "767b69580EXAMPLE"
}
```

5. To create an unreferenced commit that represents the result of merging two commit specifiers, run the **create-unreferenced-merge-commit** command, specifying:

- A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
- A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).
- The merge option you want to use (with the **--merge-option** option).
- The name of the repository (with the **--repository-name** option).
- (Optional) A conflict resolution strategy to use (with the **--conflict-resolution-strategy** option).
- (Optional) The level of detail you want about any conflicts (with the **--conflict-detail-level** option).
- (Optional) The commit message to include (with the **--commit-message** option).

- (Optional) The name to use for the commit (with the **--name** option).
- (Optional) The email address to use for the commit (with the **--email** option).
- (Optional) Whether to keep any empty folders (with the **--keep-empty-folders** option).

For example, to determine the merge conflicts for merging a source branch named *bugfix-1234* with a destination branch named *main* using the ACCEPT_SOURCE strategy in a repository named *MyDemoRepo*:

```
aws codecommit create-unreferenced-merge-commit --source-commit-
specifier bugfix-1234 --destination-commit-specifier main --merge-option
 THREE_WAY_MERGE --repository-name MyDemoRepo --name "Maria Garcia" --email
 "maria_garcia@example.com" --commit-message "Testing the results of this merge."
```

If successful, this command produces output similar to the following:

```
{
    "commitId": "4f178133EXAMPLE",
    "treeId": "389765daEXAMPLE"
}
```

Close a pull request in an AWS CodeCommit repository

If you want to close a pull request without merging the code, you can do so in one of several ways:

- In the console, you can close a pull request without merging the code. You might want to do this if you want to use the **git merge** command to merge the branches manually, or if the code in the pull request source branch isn't code you want merged into the destination branch.
- You can delete the source branch specified in the pull request. CodeCommit closes a pull request automatically if either the source or destination branch of the pull request is deleted.
- In the AWS CLI, you can update the status of a pull request from `OPEN` to `CLOSED`. This closes the pull request without merging the code.

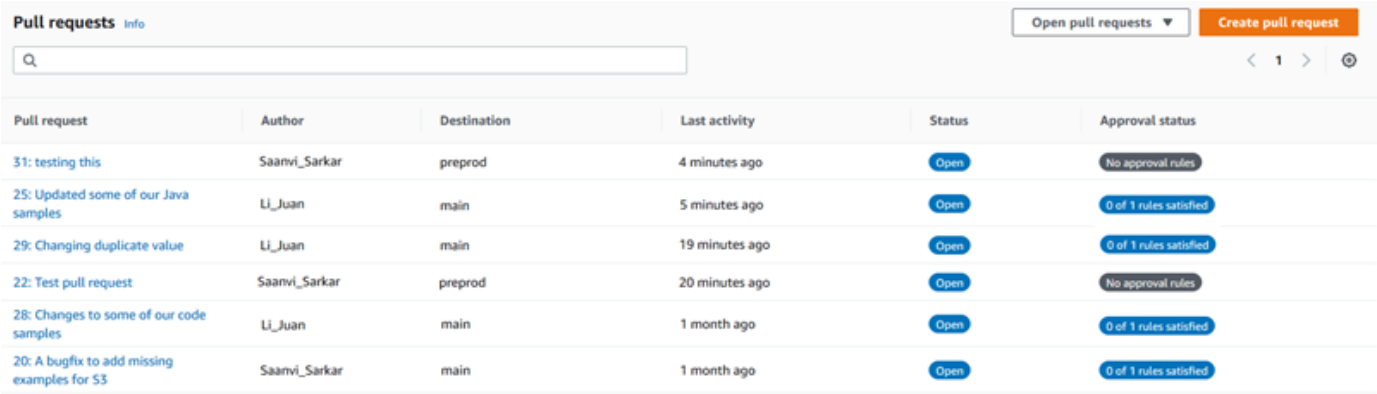
Topics

- Close a pull request (console)
- Close a pull request (AWS CLI)

Close a pull request (console)

You can use the CodeCommit console to close a pull request in a CodeCommit repository. After the status of a pull request is changed to **Closed**, it cannot be changed back to **Open**, but users can still comment on the changes and reply to comments.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository.
3. In the navigation pane, choose **Pull requests**.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to close.



5. In the pull request, choose **Close pull request**. This option closes the pull request without attempting to merge the source branch into the destination branch. This option does not provide a way to delete the source branch as part of closing the pull request, but you can do it yourself after the request is closed.

Close a pull request (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to close pull requests in a CodeCommit repository

- To update the status of a pull request in a repository from `OPEN` to `CLOSED`, run the **update-pull-request-status** command, specifying:
  - The ID of the pull request (with the **--pull-request-id** option).

- The status of the pull request (with the **--pull-request-status** option).

For example, to update the status of a pull request with the ID of *42* to a status of *CLOSED* in a CodeCommit repository named MyDemoRepo:

```
aws codecommit update-pull-request-status --pull-request-id 42 --pull-request-
status CLOSED
```

If successful, this command produces output similar to the following:

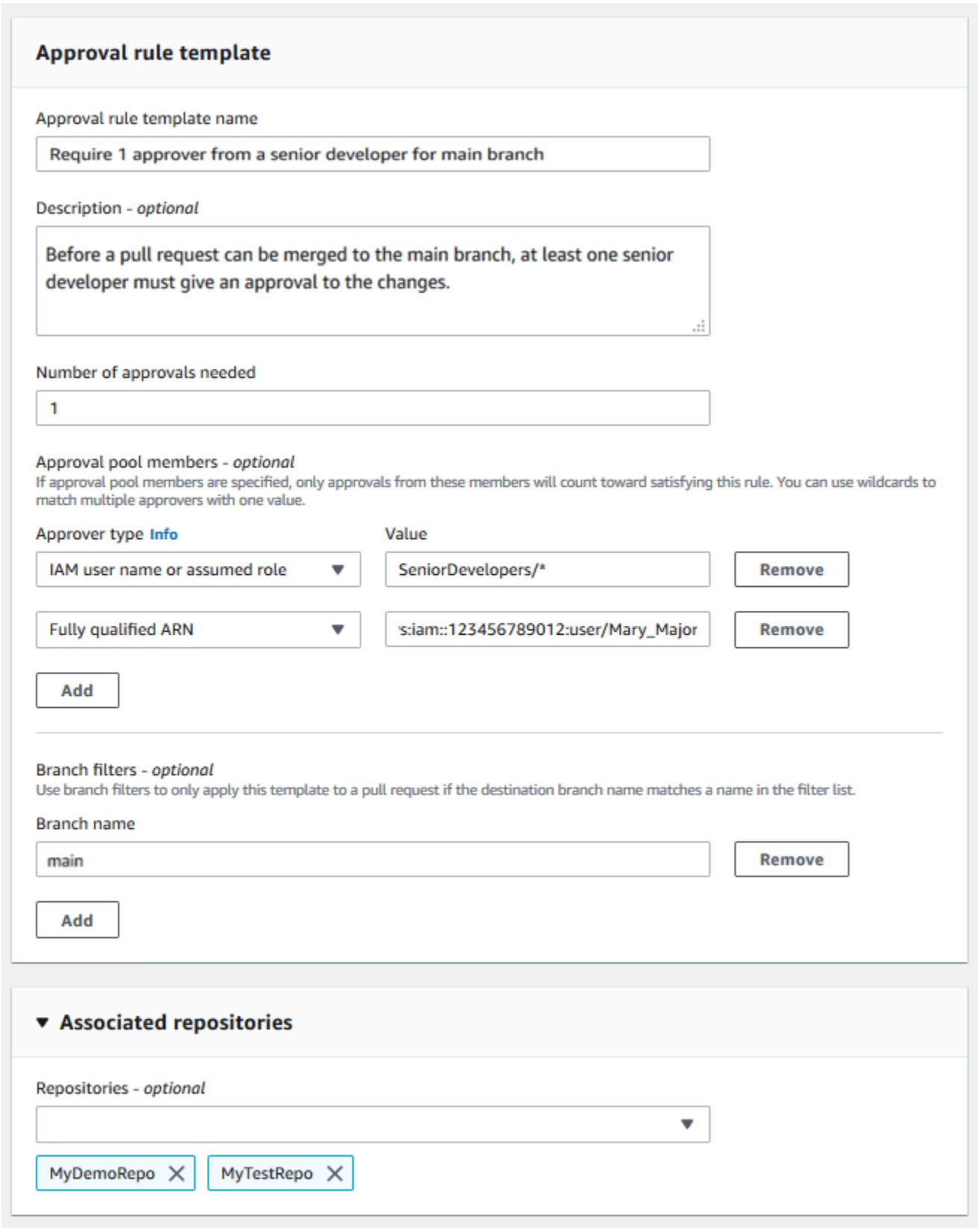
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codecommit:GetRepository",
            "Resource": [
                "arn:aws-eusc:codecommit:us-
east-2:111122223333:MySharedDemoRepo"
            ]
        }
    ]
}
```

Working with approval rule templates

You can create approval rules for pull requests. To automatically apply approval rules to some or all of the pull requests created in repositories, use approval rule templates. Approval rule templates help you customize your development workflows across repositories so that different branches have appropriate levels of approvals and control. You can define different rules for production and development branches. Those rules are applied every time a pull request that matches the rule conditions is created. For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

You can associate an approval rule template with one or more repositories in the AWS Region where they are created. When a template is associated with a repository, it automatically creates approval rules for pull requests in that repository as part of creating the pull request. Just like a single approval rule, an approval rule template defines an approval rule structure, including the number of required approvals and an optional pool of users from which approvals must come. Unlike an approval rule, you can also define destination references (the branch or branches), also known as *branch filters*. If you define destination references, then only pull requests whose destination branch names match the specified branch names (destination references) in the template have rules created for them. So, for example, if you specify **`refs/heads/main`** as a destination reference, the approval rule defined in the template is only applied to pull requests if the destination branch is `main`.


Approval rule template
Approval rule template name
Require 1 approver from a senior developer for main branch
Description - optional
Before a pull request can be merged to the main branch, at least one senior developer must give an approval to the changes.
Number of approvals needed
1
Approval pool members - optional
If approval pool members are specified, only approvals from these members will count toward satisfying this rule. You can use wildcards to match multiple approvers with one value.
Approver type Info
Value
IAM user name or assumed role
SeniorDevelopers/*
Remove
Fully qualified ARN
s:iam::123456789012:user/Mary_Major
Remove
Add
Branch filters - optional
Use branch filters to only apply this template to a pull request if the destination branch name matches a name in the filter list.
Branch name
main
Remove
Add
Associated repositories
Repositories - optional
MyDemoRepo
MyTestRepo

Topics

- Create an approval rule template
- Associate an approval rule template with a repository
- Manage approval rule templates
- Disassociate an approval rule template
- Delete an approval rule template

Create an approval rule template

You can create one or more approval rule templates to help you customize your development workflows across repositories. By creating multiple templates, you can configure the automatic creation of approval rules so that different branches have appropriate levels of approvals and control. For example, you can create different templates for production and development branches and apply these templates to one or more repositories. When users create pull requests in those repositories, the request is evaluated against those templates. If the request matches the conditions in the applied templates, approval rules are created for the pull request.

You can use the console or AWS CLI to create approval rule templates. For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

Topics

- Create an approval rule template (console)
- Create an approval rule template (AWS CLI)

Create an approval rule template (console)

Approval rule templates are not associated with any repository by default. You can make an association between a template and one or more repositories when you create the template, or you can add the associations at a later time.

To create an approval rule template (Console)

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Choose **Approval rule templates**, and then choose **Create template**.

3. In **Approval rule template name**, give the template a descriptive name so you know what it is for. For example, if you want to require one person from a set of senior developers to approve a pull request before it can be merged, you could name the rule **`Require 1 approver from a senior developer`**.
4. (Optional) In **Description**, provide a description of the purpose of this template. This can help others decide whether this template is appropriate for their repositories.
5. In **Number of approvals needed**, enter the number you want. The default is 1.
6. (Optional) If you want to require that the approvals for a pull request come from a specific group of users, in **Approval rule members**, choose **Add**. In **Approver type**, choose one of the following:

   - **IAM user name or assumed role**: This option prepopulates the Amazon Web Services account ID for the account you used to sign in, and only requires a name. It can be used for both IAM users and federated access users whose name matches the provided name. This is a very powerful option that offers a great deal of flexibility. For example, if you choose this option and are signed in with the Amazon Web Services account 123456789012, and you specify **`Mary_Major`**, all of the following are counted as approvals coming from that user:
     - An IAM user in the account (`arn:aws:iam::123456789012:user/Mary_Major`)
     - A federated user identified in IAM as Mary_Major (`arn:aws:sts::123456789012:federated-user/Mary_Major`)

     This option does not recognize an active session of someone assuming the role of **`CodeCommitReview`** with a role session name of Mary_Major (`arn:aws:sts::123456789012:assumed-role/CodeCommitReview/Mary_Major`) unless you include a wildcard (`*Mary_Major`). You can also specify the role name explicitly (`CodeCommitReview/Mary_Major`).
   - **Fully qualified ARN**: This option allows you to specify the fully qualified Amazon Resource Name (ARN) of the IAM user or role. This option also supports assumed roles used by other AWS services, such as AWS Lambda and AWS CodeBuild. For assumed roles, the ARN format should be `arn:aws:sts::`*`AccountID`*`:assumed-role/`*`RoleName`* for roles and `arn:aws:sts::`*`AccountID`*`:assumed-role/`*`FunctionName`* for functions.

   If you chose **IAM user name or assumed role** as the approver type, in **Value**, enter the name of the IAM user or role or the fully qualified ARN of the user or role. Choose **Add** again to add

more users or roles, until you have added all the users or roles whose approvals count toward the number of required approvals.

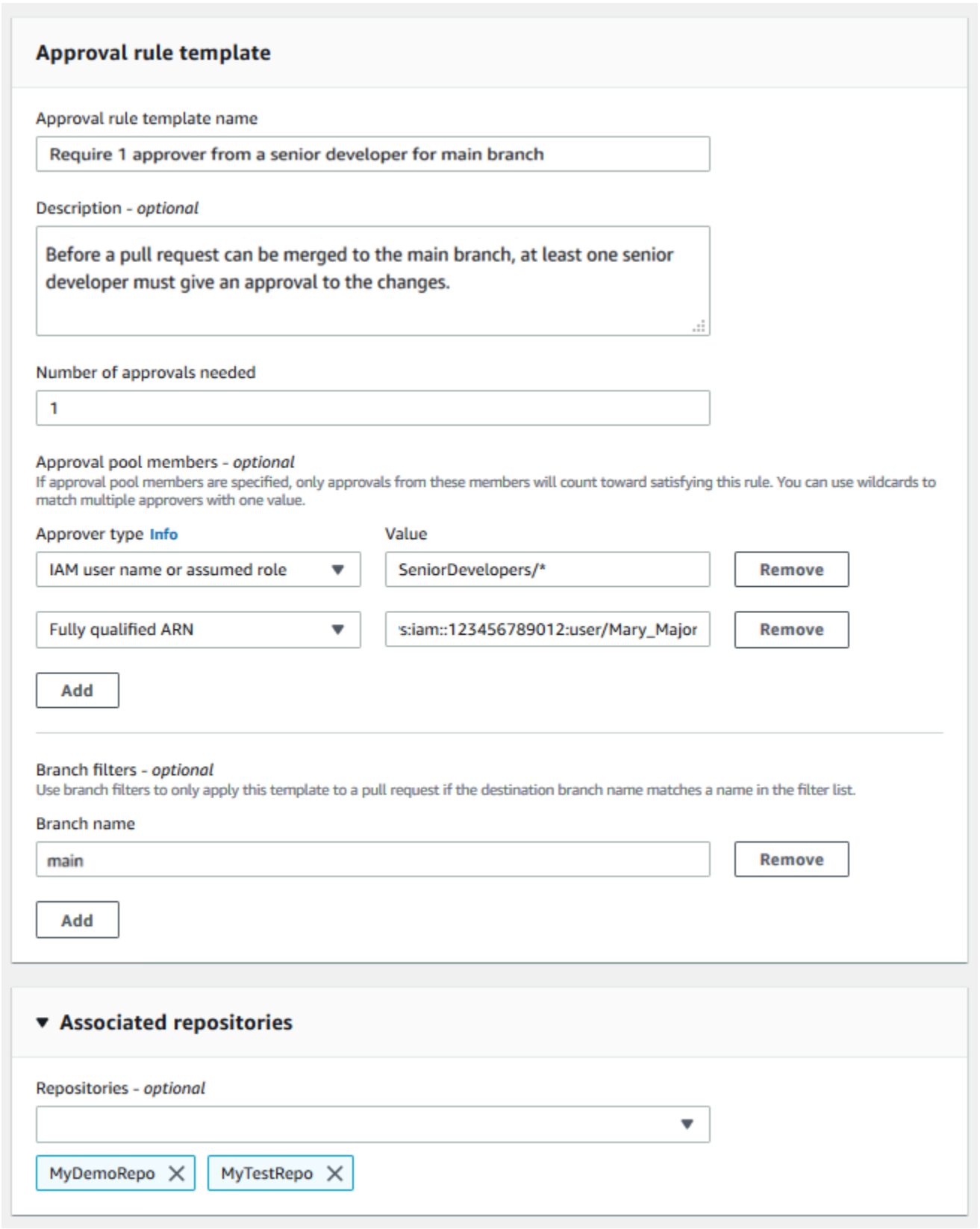
Both approver types allow you to use wildcards (*) in their values. For example, if you choose the **IAM user name or assumed role** option, and you specify **`CodeCommitReview/*`**, all users who assume the role of **`CodeCommitReview`** are counted in the approval pool. Their individual role session names count toward the required number of approvers. In this way, both Mary_Major and Li_Juan count as approvals when signed in and assuming the role of `CodeCommitReview`. For more information about IAM ARNs, wildcards, and formats, see IAM Identifiers.

> **Note**
>
> Approval rules do not support cross-account approvals.

7. (Optional) In **Branch filters**, enter destination branch names to use to filter the creation of approval rules. For example, if you specify *`main`*, an approval rule is created for pull requests in associated repositories only if the destination branch for the pull request is a branch named *`main`*. You can use wildcards (*) in branch names to apply approval rules to all branch names that match the wildcard cases. However, you cannot use a wildcard at the beginning of a branch name. You can specify up to 100 branch names. If you do not specify any filters, the template applies to all branches in an associated repository.
8. (Optional) In **Associated repositories**, in the **Repositories** list, choose the repositories in this AWS Region that you want to associate with this approval rule.

> **Note**
>
> You can choose to associate repositories after creating the template. For more information, see Associate an approval rule template with a repository.

9. Choose **Create**.


Approval rule template
Approval rule template name
Require 1 approver from a senior developer for main branch
Description - optional
Before a pull request can be merged to the main branch, at least one senior developer must give an approval to the changes.
Number of approvals needed
1
Approval pool members - optional
If approval pool members are specified, only approvals from these members will count toward satisfying this rule. You can use wildcards to match multiple approvers with one value.
Approver type Info
Value
IAM user name or assumed role
SeniorDevelopers/*
Remove
Fully qualified ARN
s:iam::123456789012:user/Mary_Major
Remove
Add
Branch filters - optional
Use branch filters to only apply this template to a pull request if the destination branch name matches a name in the filter list.
Branch name
main
Remove
Add
Associated repositories
Repositories - optional
MyDemoRepo
MyTestRepo

Create an approval rule template (AWS CLI)

You can use the AWS CLI to create approval rule templates. When you use the AWS CLI, you can specify destination references for the template, so that it applies only to pull requests whose destination branches match those in the template.

To create an approval rule template (AWS CLI)

1. At a terminal or command line, run the **create-approval-rule-template** command, specifying:

   - The name for the approval rule template. Consider using a name that describes its purpose.
   - A description of the approval rule template. As with the name, consider providing a detailed description.
   - The JSON structure of the approval rule template. This structure can include requirements for destination references, which are the destination branches for pull requests for which the approval rule is applied, and approval pool members, who are users whose approvals count toward the number of required approvals.

   When creating the content of the approval rule, you can specify approvers in an approval pool in one of two ways:

   - **CodeCommitApprovers**: This option only requires an Amazon Web Services account and a resource. It can be used for both IAM users and federated access users whose name matches the provided resource name. This is a very powerful option that offers a great deal of flexibility. For example, if you specify the AWS account 123456789012 and **`Mary_Major`**, all of the following are counted as approvals coming from that user:
     - An IAM user in the account (`arn:aws:iam::123456789012:user/Mary_Major`)
     - A federated user identified in IAM as Mary_Major (`arn:aws:sts::123456789012:federated-user/Mary_Major`)

     This option does not recognize an active session of someone assuming the role of *`SeniorDevelopers`* with a role session name of *`Mary_Major`* (`arn:aws:sts::123456789012:assumed-role/`*`SeniorDevelopers`*`/`*`Mary_Major`*) unless you include a wildcard (`*Mary_Major`).
   - **Fully qualified ARN**: This option allows you to specify the fully qualified Amazon Resource Name (ARN) of the IAM user or role.

For more information about IAM ARNs, wildcards, and formats, see IAM Identifiers.

The following example creates an approval rule template named **2-approver-rule-for-main** and a description of **Requires two developers from the team to approve the pull request if the destination branch is main**. The template requires two users who assume the role of **CodeCommitReview** to approve any pull request before it can be merged to the **main** branch:

```
aws codecommit create-approval-rule-template --approval-rule-template-name 2-
approver-rule-for-main --approval-rule-template-description "Requires two
 developers from the team to approve the pull request if the destination branch
 is main" --approval-rule-template-content "{\"Version\": \"2018-11-08\",
\"DestinationReferences\": [\"refs/heads/main\"],\"Statements\": [{\"Type
\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers\":
 [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}"
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplate": {
        "approvalRuleTemplateName": "2-approver-rule-for-main",
        "creationDate": 1571356106.936,
        "approvalRuleTemplateId": "dd8b17fe-EXAMPLE",
        "approvalRuleTemplateContent": "{\"Version\": \"2018-11-08\",
\"DestinationReferences\": [\"refs/heads/main\"],\"Statements\": [{\"Type
\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers\":
 [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
        "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
        "approvalRuleTemplateDescription": "Requires two developers from the team
 to approve the pull request if the destination branch is main",
        "lastModifiedDate": 1571356106.936,
        "ruleContentSha256": "4711b576EXAMPLE"
    }
}
```

Associate an approval rule template with a repository

Approval rule templates are created in a specific AWS Region, but they do not affect any repositores in that AWS Region until they are associated. To apply a template to one or more

repositories, you must associate the template with the repository or repositories. You can apply a single template to multiple repositories in an AWS Region. This helps you automate and standardize the development workflow in your repositories by creating consistent conditions for approving and merging pull requests.

You can only associate an approval rule template with repositories in the AWS Region where the approval rule template was created.

For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

Topics

- Associate an approval rule template (console)
- Associate an approval rule template (AWS CLI)

Associate an approval rule template (console)

You might have associated repositories with an approval rule template when you created it. (That step is optional.) You can add or remove associations by editing the template.

To associate an approval rule template with repositories

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Choose **Approval rule templates**. Choose the template, and then choose **Edit**.
3. In **Associated Repositories**, choose the repositories from the **Repositories** list. Each associated repository appears under the list box.
4. Choose **Save**. Approval rules are now applied to any pull requests created in those associated repositories.

Associate an approval rule template (AWS CLI)

You can use the AWS CLI to associate an approval rule template with one or more repositories.

To associate a template with a single repository

1. At the terminal or command line, run the **associate-approval-rule-template-with-repository** command, specifying:

- The name of the approval rule template you want to associate with a repository.
- The name of the repository to be associated with the approval rule template.

For example, to associate an approval rule template named *2-approver-rule-for-main* with a repository named *MyDemoRepo*:

```
aws codecommit associate-approval-rule-template-with-repository --repository-
name MyDemoRepo --approval-rule-template-name 2-approver-rule-for-main
```

2. If successful, this command returns nothing.

To associate a template with multiple repositories

1. At the terminal or command line, run the **batch-associate-approval-rule-template-with-repositories** command, specifying:

   - The name of the approval rule template you want to associate with a repository.
   - The names of the repositories to be associated with the approval rule template.

   For example, to associate an approval rule template named **2-approver-rule-for-main** with a repository named **MyDemoRepo** and **MyOtherDemoRepo**:

```
aws codecommit batch-associate-approval-rule-template-with-repositories --
repository-names "MyDemoRepo", "MyOtherDemoRepo" --approval-rule-template-name 2-
approver-rule-for-main
```

2. If successful, this command returns output similar to the following:

```
{
    "associatedRepositoryNames": [
        "MyDemoRepo",
        "MyOtherDemoRepo"
    ],
    "errors": []
}
```

Manage approval rule templates

You can manage the approval rule templates in an AWS Region to help understand how they are being used and what they are for. For example, you can edit the names and descriptions of approval rule templates to help others understand their purpose. You can list all the approval rule templates in an AWS Region, and get information about the content and structure of a template. You can review which templates are associated with a repository, and which repositories are associated with a template.

For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

Manage approval rule templates (console)

You can view and manage your approval rule templates in the CodeCommit console.

To manage approval rule templates

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Choose **Approval rule templates** to view the list of approval rule templates in the AWS Region where you are signed in.

   > **Note**
   >
   > Approval rule templates are only available in the AWS Region where they were created.

3. If you want to make changes to a template, choose it from the list, and then choose **Edit**.
4. Make your changes, and then choose **Save**.

Manage approval rule templates (AWS CLI)

You can manage your approval rule templates with the following AWS CLI commands:

- **list-approval-rule-templates**, to view a list of all approval rule templates in an AWS Region
- **get-approval-rule-template**, to view the content of an approval rule template
- **update-approval-rule-template-content**, to change the content of an approval rule template
- **update-approval-rule-template-name**, to change the name of an approval rule template

- update-approval-rule-template-description, to change the description of an approval rule template
- list-repositories-for-approval-rule-template, to view all repositories associated with an approval rule template
- list-associated-approval-rule-templates-for-repository, to view all approval rule templates associated with a repository

To list all approval rule templates in an AWS Region

1. At the terminal or command line, run the **list-approval-rule-templates** command. For example, to list all approval rule templates in the US East (Ohio) Region:

```
aws codecommit list-approval-rule-templates --region us-east-2
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplateNames": [
        "2-approver-rule-for-main",
        "1-approver-rule-for-all-pull-requests"
    ]
}
```

To get the content of an approval rule template

1. At the terminal or command line, run the **get-approval-rule-template** command, specifying the name of the approval rule template:

```
aws codecommit get-approval-rule-template --approval-rule-template-name 1-approver-
rule-for-all-pull-requests
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplate": {
        "approvalRuleTemplateContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 1,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
        "ruleContentSha256": "621181bbEXAMPLE",
```

```
        "lastModifiedDate": 1571356106.936,
        "creationDate": 1571356106.936,
        "approvalRuleTemplateName": "1-approver-rule-for-all-pull-requests",
        "lastModifiedUser": "arn:aws:iam::123456789012:user/Li_Juan",
        "approvalRuleTemplateId": "a29abb15-EXAMPLE",
        "approvalRuleTemplateDescription": "All pull requests must be approved by
 one developer on the team."
    }
}
```

To update the content of an approval rule template

1. At the terminal or command prompt, run the **update-approval-rule-template-content** command, specifying the name of the template and the changed content. For example, to change the content of an approval rule template named **`1-approver-rule`** to redefine the approval pool to users who assume the role of **`CodeCommitReview`**:

```
aws codecommit update-approval-rule-template-content --approval-rule-template-
name 1-approver-rule --new-rule-content "{\"Version\": \"2018-11-08\",
\"DestinationReferences\": [\"refs/heads/main\"],\"Statements\": [{\"Type
\": \"Approvers\",\"NumberOfApprovalsNeeded\": 2,\"ApprovalPoolMembers\":
 [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}"
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplate": {
        "creationDate": 1571352720.773,
        "approvalRuleTemplateDescription": "Requires 1 approval for all pull
 requests from the CodeCommitReview pool",
        "lastModifiedDate": 1571358728.41,
        "approvalRuleTemplateId": "41de97b7-EXAMPLE",
        "approvalRuleTemplateContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 1,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
        "approvalRuleTemplateName": "1-approver-rule-for-all-pull-requests",
        "ruleContentSha256": "2f6c21a5EXAMPLE",
        "lastModifiedUser": "arn:aws:iam::123456789012:user/Li_Juan"
    }
}
```

To update the name of an approval rule template

1. At the terminal or command prompt, run the **update-approval-rule-template-name** command, specifying the current name and the name you want to change it to. For example, to change the name of an approval rule template from **`1-approver-rule`** to **`1-approver-rule-for-all-pull-requests`**:

```
aws codecommit update-approval-rule-template-name --old-approval-rule-template-name
 "1-approver-rule" --new-approval-rule-template-name "1-approver-rule-for-all-pull-
requests"
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplate": {
        "approvalRuleTemplateName": "1-approver-rule-for-all-pull-requests",
        "lastModifiedDate": 1571358241.619,
        "approvalRuleTemplateId": "41de97b7-EXAMPLE",
        "approvalRuleTemplateContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 1,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
        "creationDate": 1571352720.773,
        "lastModifiedUser": "arn:aws:iam::123456789012:user/Mary_Major",
        "approvalRuleTemplateDescription": "All pull requests must be approved by
 one developer on the team.",
        "ruleContentSha256": "2f6c21a5cEXAMPLE"
    }
}
```

To update the description of an approval rule template

1. At the terminal or command line, run the **update-approval-rule-template-description** command, specifying the name of the approval rule template and the new description:

```
aws codecommit update-approval-rule-template-description --approval-rule-template-
name "1-approver-rule-for-all-pull-requests" --approval-rule-template-description
 "Requires 1 approval for all pull requests from the CodeCommitReview pool"
```

2. If successful, this command produces output similar to the following:

```
{
```

```
    "approvalRuleTemplate": {
        "creationDate": 1571352720.773,
        "approvalRuleTemplateDescription": "Requires 1 approval for all pull
 requests from the CodeCommitReview pool",
        "lastModifiedDate": 1571358728.41,
        "approvalRuleTemplateId": "41de97b7-EXAMPLE",
        "approvalRuleTemplateContent": "{\"Version\": \"2018-11-08\",\"Statements
\": [{\"Type\": \"Approvers\",\"NumberOfApprovalsNeeded\": 1,\"ApprovalPoolMembers
\": [\"arn:aws:sts::123456789012:assumed-role/CodeCommitReview/*\"]}]}",
        "approvalRuleTemplateName": "1-approver-rule-for-all-pull-requests",
        "ruleContentSha256": "2f6c21a5EXAMPLE",
        "lastModifiedUser": "arn:aws:iam::123456789012:user/Li_Juan"
    }
}
```

To list all repositories associated with a template

1. At the command line or terminal, run the **list-repositories-for-approval-rule-template** command, specifying the name of the template:

```
aws codecommit list-repositories-for-approval-rule-template --approval-rule-
template-name 2-approver-rule-for-main
```

2. If successful, this command returns output similar to the following:

```
{
    "repositoryNames": [
        "MyDemoRepo",
        "MyClonedRepo"
    ]
}
```

To list all templates associated with a repository

1. At the command line or terminal, run the **list-associated-approval-rule-templates-for-repository** command, specifying the name of the repository:

```
aws codecommit list-associated-approval-rule-templates-for-repository --repository-
name MyDemoRepo
```

2. If successful, this command returns output similar to the following:

```
{
    "approvalRuleTemplateNames": [
        "2-approver-rule-for-main",
        "1-approver-rule-for-all-pull-requests"
    ]
}
```

Disassociate an approval rule template

If the approval rules generated by an approval rule template no longer make sense for your team's workflow in a repository, you can disassociate the template from that repository. Disassociating a template does not remove any approval rules created while the template was associated with the repository.

For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

Disassociate an approval rule template (console)

You can use the console to remove the association between a repository and an approval rule template.

To disassociate an approval rule template from repositories

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Choose **Approval rule templates**. Choose the template you want to disassociate from a repository or repositories, and then choose **Edit**.
3. In **Associated repositories**, choose the **X** next to the repositories you want to disassociate. The repository names no longer appear.
4. Choose **Save**. Approval rules are not applied to pull requests created in those repositories. The rules are still applied to pull requests that were made while the association was in place.

Disassociate an approval rule template (AWS CLI)

You can use the AWS CLI to disassociate one or more repositories from an approval rule template.

To disassociate an approval rule template from a repository

1. At the terminal or command line, run the **disassociate-approval-rule-template-from-repository** command, specifying:

   - The name of the approval rule template.
   - The name of the repository.

   For example, to disassociate an approval rule template named **`1-approver-rule-for-all-pull-requests`** from a repository named **`MyDemoRepo`**:

   ```
   aws codecommit disassociate-approval-rule-template-from-repository --repository-name MyDemoRepo --approval-rule-template-name 1-approver-rule-for-all-pull-requests
   ```

2. If successful, this command returns nothing.

To disassociate an approval rule template from multiple repositories

1. At the terminal or command line, run the **batch-disassociate-approval-rule-template-from-repositories** command, specifying:

   - The name of the approval rule template.
   - The names of the repositories.

   For example, to disassociate an approval rule template named **`1-approver-rule-for-all-pull-requests`** from a repository named **`MyDemoRepo`** and a repository named **`MyOtherDemoRepo`**:

   ```
   aws codecommit batch-disassociate-approval-rule-template-from-repositories --repository-names "MyDemoRepo", "MyOtherDemoRepo" --approval-rule-template-name 1-approver-rule-for-all-pull-requests
   ```

2. If successful, this command returns output similar to the following:

   ```
   {
       "disassociatedRepositoryNames": [
           "MyDemoRepo",
           "MyOtherDemoRepo"
       ],
   ```

```
    "errors": []
}
```

Delete an approval rule template

You can delete an approval rule template if you are not using it in any repositories. Deleting unused approval rule templates helps keep your templates organized and makes it easier to find templates that make sense for your workflows.

For more information about managed policies and permissions for approval rule templates, see Permissions for actions on approval rule templates and AWS managed policies for CodeCommit.

Topics

- Delete an approval rule template (console)
- Delete an approval rule template (AWS CLI)

Delete an approval rule template (console)

You can delete an approval rule template if it is no longer relevant to your development work. When you use the console to delete an approval rule template, it is disassociated from any repositories during the deletion process.

To delete an approval rule template

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. Choose **Approval rule templates**. Choose the template you want to delete, and then choose **Delete**.

Delete an approval rule template (AWS CLI)

You can use the AWS CLI to delete an approval rule if it has been disassociated from all repositories. For more information, see Disassociate an approval rule template (AWS CLI).

To delete an approval rule template

1. At a terminal or command line, run the **delete-approval-rule-template** command, specifying the name of the approval rule template you want to delete:

```
aws codecommit delete-approval-rule-template --approval-rule-template-name 1-approver-for-all-pull-requests
```

2. If successful, this command returns output similar to the following. If the approval rule template has already been deleted, this command returns nothing.

```
{
    "approvalRuleTemplateId": "41de97b7-EXAMPLE"
}
```

Working with commits in AWS CodeCommit repositories

Commits are snapshots of the contents and changes to the contents of your repository. Every time a user commits and pushes a change, that information is saved and stored. So, too, is information that includes who committed the change, the date and time of the commit, and the changes made as part of the commit. You can also add tags to commits, to easily identify specific commits. In CodeCommit, you can:

- Review commits.
- View the history of commits in a graph.
- Compare a commit to its parent or to another specifier.
- Add comments to your commits and reply to comments made by others.



Before you can push commits to a CodeCommit repository, you must set up your local computer to connect to the repository. For the simplest method, see For HTTPS users using Git credentials.

For information about working with other aspects of your repository in CodeCommit, see Working with repositories, Working with files, Working with pull requests, Working with branches, and Working with user preferences.

Topics

- Create a commit in AWS CodeCommit
- View commit details in AWS CodeCommit
- Compare commits in AWS CodeCommit
- Comment on a commit in AWS CodeCommit
- Create a Git tag in AWS CodeCommit
- View Git tag details in AWS CodeCommit
- Delete a Git tag in AWS CodeCommit

Create a commit in AWS CodeCommit

When you create the first commit for a new repository, you use the AWS CLI and the **put-file** command. This creates the first commit and it allows you to create and specify the default branch for your new repository. You can use Git or the AWS CLI to create a commit in a CodeCommit repository. If the local repo is connected to a CodeCommit repository, you use Git to push the commit from the local repo to the CodeCommit repository. To create a commit directly in the CodeCommit console, see Create or add a file to an AWS CodeCommit repository and Edit the contents of a file in an AWS CodeCommit repository.

> **Note**
>
> As a best practice, we recommend that you use the latest supported versions of the AWS CLI, Git, and other software. If you use the AWS CLI, make sure that you have a recent version installed to ensure that you are using a version that contains the `create-commit` command.

Topics

- Create the first commit for a repository using the AWS CLI
- Create a commit using a Git client
- Create a commit using the AWS CLI

Create the first commit for a repository using the AWS CLI

You can use the AWS CLI and the `put-file` command to create your first commit for a repository. Using **put-file** creates a first commit that adds a file to your empty repository, and it creates a

branch with the name you specify. It designates the new branch as the default branch for your repository.

> **Note**
>
> To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To create the first commit for a repository using the AWS CLI

1. On your local computer, create the file you want to add as the first file to the CodeCommit repository. A common practice is to create a `README.md` markdown file that explains the purpose of this repository to other repository users. If you include a `README.md` file, the content of the file is displayed automatically at the bottom of the **Code** page for your repository in the CodeCommit console.
2. At the terminal or command line, run the **put-file** command, specifying:

   - The name of the repository where you want to add the first file.
   - The name of the branch you want to create as the default branch.
   - The local location of the file. The syntax used for this location varies, depending on your local operating system.
   - The name of the file you want to add, including the path where the updated file is stored in the repository.
   - The user name and email you want to associate with this file.
   - A commit message that explains why you added this file.

   The user name, email address, and commit message are optional, but can help other users know who made the change and why. If you do not supply a user name, CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.

   For example, to add a file named *README.md* with example base 6 encoded file content to a repository named *MyDemoRepo* to a branch named *development*:

```
aws codecommit put-file --repository-name MyDemoRepo --branch-name development --
file-path README.md --file-content "EXAMPLE0123456789example0123456789EXAMPLE1"
```

```
--name "Mary Major" --email "mary_major@example.com" --commit-message "I added a
quick readme for our new team repository."
```

If successful, this command returns output similar to the following:

```
{
    "commitId": "724caa36EXAMPLE",
    "blobId": "a8a94062EXAMPLE",
    "treeId": "08b2fc73EXAMPLE"
}
```

Create a commit using a Git client

You can create commits using a Git client installed on your local computer, and then push those commits to your CodeCommit repository.

1. Complete the prerequisites, including Setting up .

   Important

   If you have not completed setup, you cannot connect or commit to the repository using Git.

2. Make sure you are creating a commit in the correct branch. To see a list of available branches and find out which branch you are currently set to use, run **git branch**. All branches are displayed. An asterisk (*) appears next to your current branch. To switch to a different branch, run **git checkout *branch-name***. If this is your first commit, run the **git config** command to configure your Git client to create an initial branch with the name you want to use for that branch. For example, if you want your default branch to have the name *development*:

   ```
   git config --local init.defaultBranch development
   ```

   Tip

   This command is only available in Git v.2.28 and later.
   You can also run this command to set your default branch name to **development** for all newly-created repositories:

```
git config --global init.defaultBranch development
```

3. Make a change to the branch (such as adding, modifying, or deleting a file).

   For example, in the local repo, create a file named `bird.txt` with the following text:

```
bird.txt
--------
Birds (class Aves or clade Avialae) are feathered, winged, two-legged, warm-
blooded, egg-laying vertebrates.
```

4. Run **git status**, which should indicate that `bird.txt` has not yet been included in any pending commit:

```
...
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        bird.txt
```

5. Run **git add bird.txt** to include the new file in the pending commit.

6. If you run **git status** again, you should see output similar to the following. It indicates that `bird.txt` is now part of the pending commit or staged for commit:

```
...
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   bird.txt
```

7. To finalize the commit, run **git commit** with the `-m` option (for example, **git commit -m "*Adding bird.txt to the repository.*"**) The `-m` option creates the commit message.

8. If you run **git status** again, you should see output similar to the following. It indicates that the commit is ready to be pushed from the local repo to the CodeCommit repository:

```
...
nothing to commit, working directory clean
```

9. Before you push the finalized commit from the local repo to the CodeCommit repository, you can see what you are pushing by running **git diff --stat *remote-name*/*branch-name***, where *remote-name* is the nickname the local repo uses for the CodeCommit repository and *branch-name* is the name of the branch to compare.

   > **Tip**
   >
   > To get the nickname, run **git remote**. To get a list of branch names, run **git branch**. An asterisk (*) appears next to the current branch. You can also run **git status** to get the current branch name.

   > **Note**
   >
   > If you cloned the repository, from the perspective of the local repo, *remote-name* is not the name of the CodeCommit repository. When you clone a repository, *remote-name* is set automatically to `origin`.

   For example, **git diff --stat origin/main** would show output similar to the following:

   ```
   bird.txt | 1 +
   1 file changed, 1 insertion(+)
   ```

   The output assumes you have already connected the local repo to the CodeCommit repository. (For instructions, see Connect to a repository.)

10. When you're ready to push the commit from the local repo to the CodeCommit repository, run **git push *remote-name branch-name***, where *remote-name* is the nickname the local repo uses for the CodeCommit repository and *branch-name* is the name of the branch to push to the CodeCommit repository.

    For example, running **git push origin main** would show output similar to the following:

    For HTTPS:

    ```
    Counting objects: 7, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (4/4), done.
    ```

```
Writing objects: 100% (5/5), 516 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote:
To https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
    b9e7aa6..3dbf4dd main -> main
```

For SSH:

```
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 516 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote:
To ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
    b9e7aa6..3dbf4dd main -> main
```

Tip

If you add the `-u` option to **git push** (for example, **git push -u origin main**), then you only need to run **git push** in the future because upstream tracking information has been set. To get upstream tracking information, run **git remote show *remote-name*** (for example, **git remote show origin**).

For more options, see your Git documentation.

Create a commit using the AWS CLI

You can use the AWS CLI and the `create-commit` command to create a commit for a repository on the tip of a specified branch. You can also create an unreferenced merge commit to represent the results of merging two commit specifiers. For more information, see Create an unreferenced commit.

Note

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To create a commit

1. On your local computer, make the changes you want committed to the CodeCommit repository.
2. At the terminal or command line, run the **create-commit** command, specifying:

   - The repository where you want to commit the changes.
   - The branch where you want to commit the changes.
   - The full commit ID of the most recent commit made to that branch, also known as the tip or head commit or the parent commit ID.
   - Whether to keep any empty folders if the changes you made delete the content of those folders. By default, this value is false.
   - The information about the files you want added, changed, or deleted.
   - The user name and email you want associated with these changes.
   - A commit message that explains why you made these changes.

   The user name, email address, and commit message are optional, but help other users know who made the changes and why. If you do not supply a user name, CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.

   For example, to create a commit for a repository that adds a `README.md` file to a repository named *`MyDemoRepo`* in the *`main`* branch. The content of the file is in Base64 and reads "Welcome to our team repository!":

   ```
   aws codecommit create-commit --repository-name MyDemoRepo --
   branch-name main --parent-commit-id 4c925148EXAMPLE --put-files
    "filePath=README.md,fileContent=V2VsY29tZSB0byBvdXIgdGVhbSByZXBvc2l0b3J5IQo="
   ```

   > **Tip**
   >
   > To get the parent commit ID, run the get-branch command.

   If successful, this command returns output similar to the following:

   ```
   {
   ```

```
    "commitId": "4df8b524-EXAMPLE",
    "treeId": "55b57003-EXAMPLE",
    "filesAdded": [
        {
            "blobId": "5e1c309dEXAMPLE",
            "absolutePath": "meeting.md",
            "fileMode": "NORMAL"
        }
    ],
    "filesDeleted": [],
    "filesUpdated": []
}
```

To create a commit that makes changes to files named *file1.py* and *file2.txt*, renames a file from *picture.png* to *image1.png* and moves it from a directory named *pictures* to a directory named, *images*, and deletes a file named *ExampleSolution.py* in a repository named *MyDemoRepo* on a branch named *MyFeatureBranch* whose most recent commit has an ID of *4c925148EXAMPLE*:

```
aws codecommit create-commit --repository-name MyDemoRepo --branch-
name MyFeatureBranch --parent-commit-id 4c925148EXAMPLE --author-name "Saanvi
 Sarkar"
 --email "saanvi_sarkar@example.com" --commit-message "I'm creating this commit to
 update a variable name in a number of files."
 --keep-empty-folders false  --put-files '{"filePath": "file1.py", "fileMode":
 "EXECUTABLE", "fileContent": "bucket_name = sys.argv[1] region = sys.argv[2]"}'
'{"filePath": "file2.txt", "fileMode": "NORMAL", "fileContent": "//Adding a comment
 to explain the variable changes in file1.py"}' '{"filePath": "images/image1.png",
"fileMode": "NORMAL", "sourceFile": {"filePath": "pictures/picture.png", "isMove":
 true}}' --delete-files filePath="ExampleSolution.py"
```

> **Note**
>
> The syntax for the **--put-files** segment varies depending on your operating system. The above example is optimized for Linux, macOS, or Unix users and Windows users with a Bash emulator. Windows users at the command line or in Powershell should use syntax appropriate for those systems.

If successful, this command returns output similar to the following:

```
{
   "commitId": "317f8570EXAMPLE",
   "treeId": "347a3408EXAMPLE",
   "filesAdded": [
        {
        "absolutePath": "images/image1.png",
        "blobId": "d68ba6ccEXAMPLE",
        "fileMode": "NORMAL"
        }
    ],
    "filesUpdated": [
        {
        "absolutePath": "file1.py",
        "blobId": "0a4d55a8EXAMPLE",
        "fileMode": "EXECUTABLE"
        },
        {
        "absolutePath": "file2.txt",
        "blobId": "915766bbEXAMPLE",
        "fileMode": "NORMAL"
        }
    ],
    "filesDeleted": [
        {
        "absolutePath": "ExampleSolution.py",
        "blobId": "4f9cebe6aEXAMPLE",
        "fileMode": "EXECUTABLE"
        },
        {
        "absolutePath": "pictures/picture.png",
        "blobId": "fb12a539EXAMPLE",
        "fileMode": "NORMAL"
        }
    ]
}
```

View commit details in AWS CodeCommit

You can use the AWS CodeCommit console to browse the history of commits in a repository. This can help you identify changes made in a repository, including:

- When and by whom the changes were made.
- When specific commits were merged into a branch.

Viewing the history of commits for a branch might also help you understand the difference between branches. If you use tagging, you can also quickly view the commit that was labeled with a tag and the parents of that tagged commit. At the command line, you can use Git to view details about the commits in a local repo or a CodeCommit repository.

Browse commits in a repository

You can use the AWS CodeCommit console to browse the history of commits to a repository. You can also view a graph of the commits in the repository and its branches over time. This can help you understand the history of the repository, including when changes were made.

> **Note**
>
> Using the **git rebase** command to rebase a repository changes the history of a repository, which might cause commits to appear out of order. For more information, see Git Branching-Rebasing or your Git documentation.

Topics

- Browse the commit history of a repository
- View a graph of the commit history of a repository

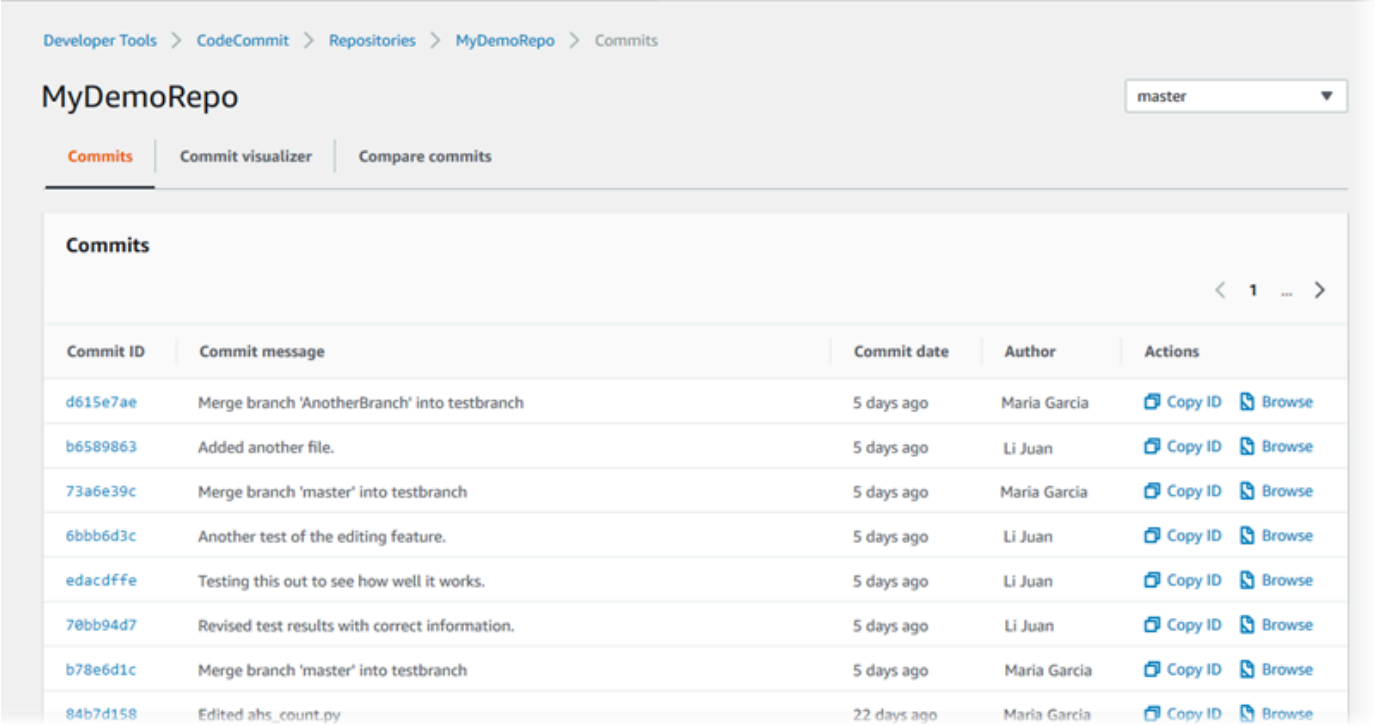
Browse the commit history of a repository

You can browse the commit history for a specific branch or tag of the repository, including information about the committer and the commit message. You can also view the code for a commit.

To browse the history of commits

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository for which you want to review the commit history.

3. In the navigation pane, choose **Commits**. In the commit history view, a history of commits for the repository in the default branch is displayed, in reverse chronological order of the commit date. Date and time are in coordinated universal time (UTC). You can view the commit history of a different branch by choosing the view selector button and then choosing a branch from the list. If you are using tags in your repository, you can view a commit with a specific tag and its parents by choosing that tag in the view selector button.



4. To view the difference between a commit and its parent, and to see any comments on the changes, choose the abbreviated commit ID. For more information, see Compare a commit to its parent and Comment on a commit. To view the difference between a commit and any other commit specifier, including a branch, tag, or commit ID, see Compare any two commit specifiers.

5. Do one or more of the following:

   - To view the date and time a change was made, hover over the commit date.
   - To view the full commit ID, copy and then paste it into a text editor or other location. To copy it, choose **Copy ID**.
   - To view the code as it was at the time of a commit, choose **Browse**. The contents of the repository as they were at the time of that commit is displayed in the **Code** view. The view selector button displays the abbreviated commit ID instead of a branch or tag.

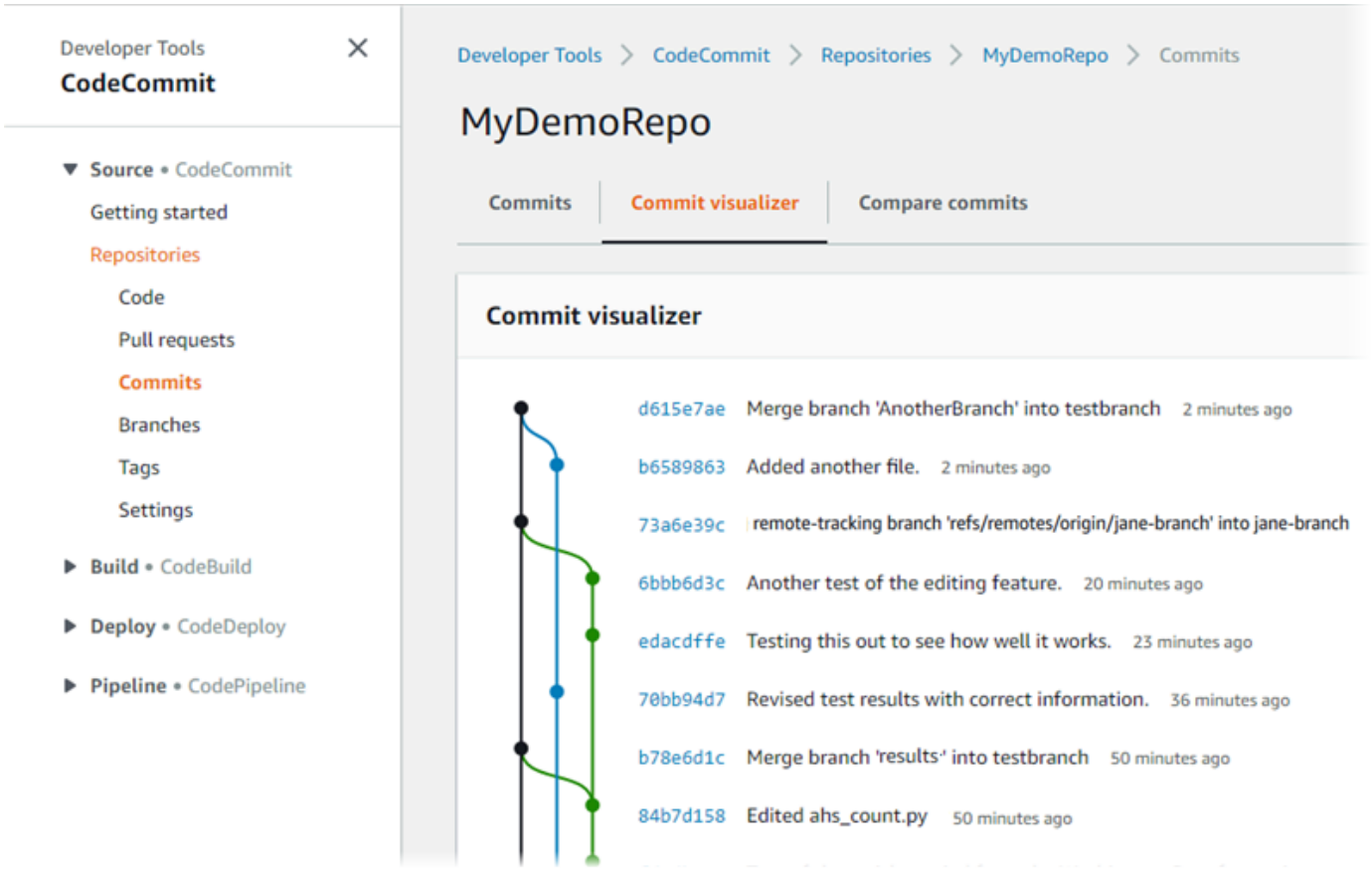
View a graph of the commit history of a repository

You can view a graph of the commits made to a repository. The **Commit Visualizer** view is a directed acyclic graph (DAG) representation of all the commits made to a branch of the repository. This graphical representation can help you understand when commits and associated features were added or merged. It can also help you pinpoint when a change was made in relation to other changes.

> **Note**
>
> Commits that are merged using the fast-forward method do not appear as separate lines in the graph of commits.

To view a graph of commits

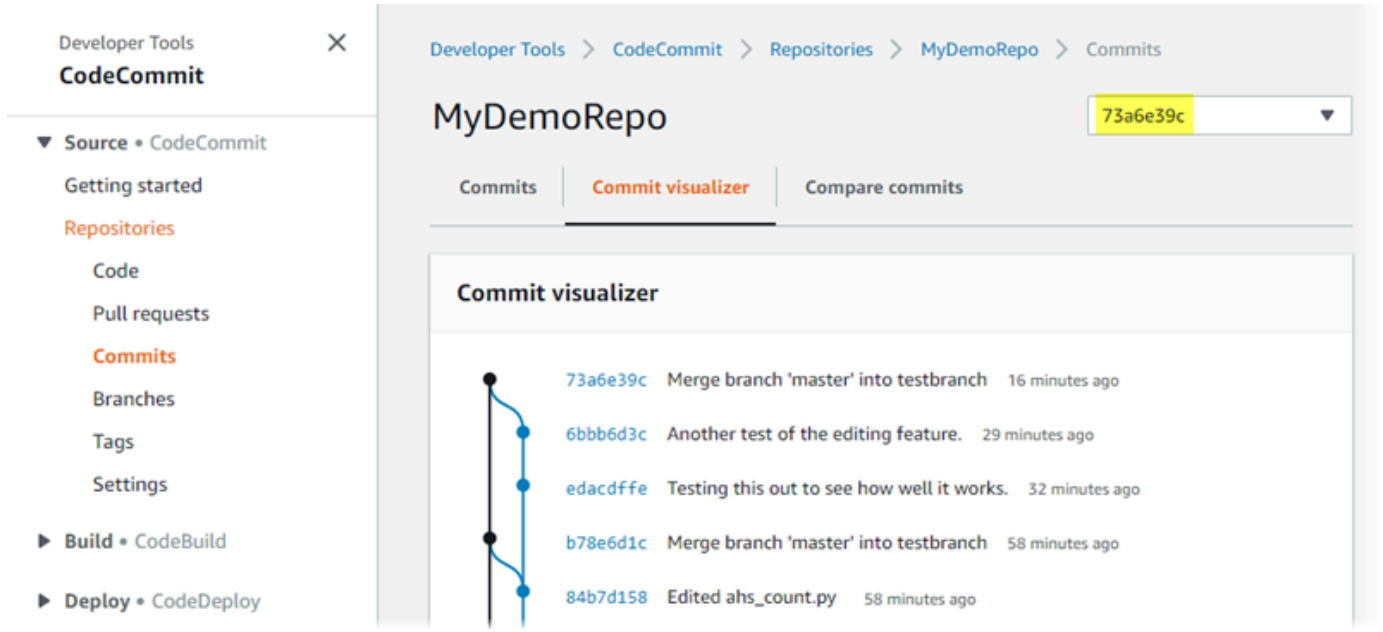
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository for which you want to view a commit graph.
3. In the navigation pane, choose **Commits**, and then choose the **Commit visualizer** tab.



In the commit graph, the abbreviated commit ID and the subject for each commit message appears next to that point in the graph.

> **Note**
>
> The graph can display up to 35 branches on a page. If there are more than 35 branches, the graph is too complex to display. You can simplify the view in two ways:
>
> - By using the view selector button to show the graph for a specific branch.
> - By pasting a full commit ID into the search box to render the graph from that commit.

4. To render a new graph from a commit, choose the point in the graph that corresponds to that commit. The view selector button changes to the abbreviated commit ID.



View commit details (AWS CLI)

Git lets you view details about commits. You can also use the AWS CLI to view details about the commits in a local repo or in a CodeCommit repository by running the following commands:

- To view information about a commit, run **aws codecommit get-commit**.
- To view information about multiple commits, run **aws codecommit batch-get-commits**.
- To view information about a merge commit, run **aws codecommit get-merge-commit**.
- To view information about changes for a commit specifier (branch, tag, HEAD, or other fully qualified references, such as commit IDs), run **aws codecommit get-differences**.
- To view the base64-encoded content of a Git blob object in a repository, run **aws codecommit get-blob**.

To view information about a commit

1. Run the **aws codecommit get-commit** command, specifying:
   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The full commit ID.

For example, to view information about a commit with the ID `317f8570EXAMPLE` in a CodeCommit repository named `MyDemoRepo`:

```
aws codecommit get-commit  --repository-name MyDemoRepo  --commit-id
 317f8570EXAMPLE
```

2. If successful, the output of this command includes the following:

- Information about the author of the commit (as configured in Git), including the date in timestamp format and the coordinated universal time (UTC) offset.
- Information about the committer (as configured in Git) including the date in timestamp format and the UTC offset.
- The ID of the Git tree where the commit exists.
- The commit ID of the parent commit.
- The commit message.

Here is some example output, based on the preceding example command:

```
{
    "commit": {
        "additionalData": "",
        "committer": {
            "date": "1484167798 -0800",
            "name": "Mary Major",
            "email": "mary_major@example.com"
        },
        "author": {
            "date": "1484167798 -0800",
            "name": "Mary Major",
            "email": "mary_major@example.com"
        },
        "treeId": "347a3408EXAMPLE",
        "parents": [
            "4c925148EXAMPLE"
        ],
        "message": "Fix incorrect variable name"
    }
}
```

To view information about a merge commit

1. Run the **get-merge-commit** command, specifying:

   - A commit specifier for the source of the merge (with the **--source-commit-specifier** option).
   - A commit specifier for the destination for the merge (with the **--destination-commit-specifier** option).
   - The merge option you want to use (with the **--merge-option** option).
   - The name of the repository (with the **--repository-name** option).

   For example, to view information about a merge commit for the source branch named *bugfix-bug1234* with a destination branch named *main* using the *THREE_WAY_MERGE* strategy in a repository named *MyDemoRepo*:

   ```
   aws codecommit get-merge-commit --source-commit-specifier bugfix-bug1234 --
   destination-commit-specifier main --merge-option THREE_WAY_MERGE --repository-
   name MyDemoRepo
   ```

2. If successful, the output of this command returns information similar to the following:

   ```
   {
       "sourceCommitId": "c5709475EXAMPLE",
       "destinationCommitId": "317f8570EXAMPLE",
       "baseCommitId": "fb12a539EXAMPLE",
       "mergeCommitId": "ffc4d608eEXAMPLE"
   }
   ```

To view information about multiple commits

1. Run the **batch-get-commits** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - A list of full commit IDs for every commit about which you want to view information.

   For example, to view information about commits with the IDs `317f8570EXAMPLE` and `4c925148EXAMPLE` in a CodeCommit repository named `MyDemoRepo`:

```
aws codecommit batch-get-commits  --repository-name MyDemoRepo  --commit-ids
 317f8570EXAMPLE 4c925148EXAMPLE
```

2. If successful, the output of this command includes the following:

   - Information about the authors of the commits (as configured in Git), including the date in timestamp format and the coordinated universal time (UTC) offset.
   - Information about the committers (as configured in Git) including the date in timestamp format and the UTC offset.
   - The IDs of the Git tree where the commit exists.
   - The commit IDs of the parent commit.
   - The commit messages.

   Here is some example output, based on the preceding example command:

```
{
    "commits": [
      {
        "additionalData": "",
        "committer": {
            "date": "1508280564 -0800",
            "name": "Mary Major",
            "email": "mary_major@example.com"
        },
        "author": {
            "date": "1508280564 -0800",
            "name": "Mary Major",
            "email": "mary_major@example.com"
        },
        "commitId": "317f8570EXAMPLE",
        "treeId": "1f330709EXAMPLE",
        "parents": [
            "6e147360EXAMPLE"
        ],
        "message": "Change variable name and add new response element"
    },
    {
        "additionalData": "",
        "committer": {
```

```
            "date": "1508280542 -0800",
            "name": "Li Juan",
            "email": "li_juan@example.com"
        },
        "author": {
            "date": "1508280542 -0800",
            "name": "Li Juan",
            "email": "li_juan@example.com"
        },
        "commitId": "4c925148EXAMPLE",
        "treeId": "1f330709EXAMPLE",
        "parents": [
            "317f8570EXAMPLE"
        ],
        "message": "Added new class"
    }
}
```

To view information about the changes for a commit specifier

1. Run the **aws codecommit get-differences** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The commit specifiers you want to get information about. Only `--after-commit-specifier` is required. If you do not specify `--before-commit-specifier`, all files current as of the `--after-commit-specifier` are shown.

   For example, to view information about the differences between commits with the IDs `317f8570EXAMPLE` and `4c925148EXAMPLE` in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit get-differences  --repository-name MyDemoRepo  --before-commit-
   specifier 317f8570EXAMPLE --after-commit-specifier 4c925148EXAMPLE
   ```

2. If successful, the output of this command includes the following:

   - A list of differences, including the change type (A for added, D for deleted, or M for modified).
   - The mode of the file change type.

- The ID of the Git blob object that contains the change.

Here is some example output, based on the preceding example command:

```
{
    "differences": [
        {
            "afterBlob": {
                "path": "blob.txt",
                "blobId": "2eb4af3bEXAMPLE",
                "mode": "100644"
            },
            "changeType": "M",
            "beforeBlob": {
                "path": "blob.txt",
                "blobId": "bf7fcf28fEXAMPLE",
                "mode": "100644"
            }
        }
    ]
}
```

To view information about a Git blob object

1. Run the **aws codecommit get-blob** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The ID of the Git blob (with the `--blob-id` option).

   For example, to view information about a Git blob with the ID of `2eb4af3bEXAMPLE` in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit get-blob  --repository-name MyDemoRepo  --blob-id 2eb4af3bEXAMPLE
   ```

2. If successful, the output of this command includes the following:

   - The base64-encoded content of the blob, usually a file.

For example, the output of the previous command might be similar to the following:

```
{
    "content": "QSBCaW5hcnkgTGFyToEXAMPLE="
}
```

To view a line-level diff between two blobs

Use **get-blob-differences** to display a line-level diff between two file versions. The command returns an ordered list of hunks—contiguous runs of changed lines with surrounding context—that you can use to render side-by-side or unified diff views in your tooling.

1. Run the **get-blob-differences** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The ID of the after blob—the version to compare to (with the `--after-blob-id` option).
   - Optionally, the ID of the before blob (with the `--before-blob-id` option). If you do not specify a before blob, the command computes the diff against an empty before-state, which is the same as treating the file as newly added.
   - Optionally, the number of unchanged context lines per hunk (with the `--context-lines` option, integer between 0 and 20; defaults to 3).
   - Optionally, whether to ignore whitespace-only changes (with the `--ignore-whitespace` option).
   - Optionally, the maximum total number of diff hunks to return in the command's output (with the `--max-items` option, any positive integer). The AWS Command Line Interface (AWS CLI) might make multiple internal service calls to collect results. If more results exist beyond `--max-items`, the output includes a `NextToken` pagination token.
   - Optionally, the number of results per service call (with the `--page-size` option, integer between 1 and 1000; defaults to 100).
   - Optionally, a pagination token from a previous response (with the `--starting-token` option). Use this value to retrieve the next page when the output includes a `NextToken`.

   For example, to view the line-level diff between two blob versions of a file in a CodeCommit repository named `MyDemoRepo`, with three lines of context:

```
aws codecommit get-blob-differences --repository-name MyDemoRepo --before-blob-id
 bf7fcf28fEXAMPLE --after-blob-id 2eb4af3bEXAMPLE
```

2. If successful, the output of this command includes the following:

   - The list of diff hunks, each with start lines and counts for both versions and an ordered list of line-level changes (`CONTEXT`, `ADD`, or `DELETE`). `ADD` changes omit `beforeLineNumber`, and `DELETE` changes omit `afterLineNumber`.
   - Whether the content was treated as binary (`isBinary`).
   - The size of each blob in bytes (`beforeBlobSize`, `afterBlobSize`).
   - If the diff spans multiple pages, an opaque `NextToken` that you can pass to a follow-up **get-blob-differences** call (with the `--starting-token` option) to retrieve the next page. When the response contains the final page, `NextToken` is omitted.

   For example, the output of the previous command might be similar to the following:

```
{
    "hunks": [
        {
            "beforeStartLine": 1,
            "beforeLineCount": 4,
            "afterStartLine": 1,
            "afterLineCount": 5,
            "changes": [
                { "type": "CONTEXT", "beforeLineNumber": 1, "afterLineNumber": 1,
 "content": "import os" },
                { "type": "DELETE",  "beforeLineNumber": 2, "content": "DEBUG =
 False" },
                { "type": "ADD",     "afterLineNumber": 2, "content": "DEBUG =
 True" },
                { "type": "ADD",     "afterLineNumber": 3, "content": "VERBOSE =
 True" },
                { "type": "CONTEXT", "beforeLineNumber": 3, "afterLineNumber": 4,
 "content": "" },
                { "type": "CONTEXT", "beforeLineNumber": 4, "afterLineNumber": 5,
 "content": "def main():" }
            ]
        }
    ],
```

```
    "isBinary": false,
    "beforeBlobSize": 142,
    "afterBlobSize": 168
}
```

View commit details (Git)

Before you follow these steps, you should have already connected the local repo to the CodeCommit repository and committed changes. For instructions, see Connect to a repository.

To show the changes for the most recent commit to a repository, run the **git show** command.

```
git show
```

The command produces output similar to the following:

```
commit 4f8c6f9d
Author: Mary Major <mary.major@example.com>
Date:   Mon May 23 15:56:48 2016 -0700

    Added bumblebee.txt

diff --git a/bumblebee.txt b/bumblebee.txt
new file mode 100644
index 0000000..443b974
--- /dev/null
+++ b/bumblebee.txt
@@ -0,0 +1 @@
+A bumblebee, also written bumble bee, is a member of the bee genus Bombus, in the
 family Apidae.
\ No newline at end of file
```

> **Note**
>
> In this and the following examples, commit IDs have been abbreviated. The full commit IDs are not shown.

To view the changes that occurred, use the **git show** command with the commit ID:

```
git show 94ba1e60

commit 94ba1e60
Author: John Doe <johndoe@example.com>
Date:   Mon May 23 15:39:14 2016 -0700

    Added horse.txt

diff --git a/horse.txt b/horse.txt
new file mode 100644
index 0000000..080f68f
--- /dev/null
+++ b/horse.txt
@@ -0,0 +1 @@
+The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus.
```

To see the differences between two commits, run the **git diff** command and include the two commit IDs.

```
git diff ce22850d 4f8c6f9d
```

In this example, the difference between the two commits is that two files were added. The command produces output similar to the following:

```
diff --git a/bees.txt b/bees.txt
new file mode 100644
index 0000000..cf57550
--- /dev/null
+++ b/bees.txt
@@ -0,0 +1 @@
+Bees are flying insects closely related to wasps and ants, and are known for their
 role in pollination and for producing honey and beeswax.
diff --git a/bumblebee.txt b/bumblebee.txt
new file mode 100644
index 0000000..443b974
--- /dev/null
+++ b/bumblebee.txt
@@ -0,0 +1 @@
+A bumblebee, also written bumble bee, is a member of the bee genus Bombus, in the
 family Apidae.
\ No newline at end of file
```

To use Git to view details about the commits in a local repo, run the **git log** command:

```
git log
```

If successful, this command produces output similar to the following:

```
commit 94ba1e60
Author: John Doe <johndoe@example.com>
Date:   Mon May 23 15:39:14 2016 -0700

    Added horse.txt

commit 4c925148
Author: Jane Doe <janedoe@example.com>
Date:   Mon May 22 14:54:55 2014 -0700

    Added cat.txt and dog.txt
```

To show only commit IDs and messages, run the **git log --pretty=oneline** command:

```
git log --pretty=oneline
```

If successful, this command produces output similar to the following:

```
94ba1e60 Added horse.txt
4c925148 Added cat.txt and dog.txt
```

For more options, see your Git documentation.

Compare commits in AWS CodeCommit

You can use the CodeCommit console to view the differences between commit specifiers in a CodeCommit repository. You can quickly view the difference between a commit and its parent. You can also compare any two references, including commit IDs.

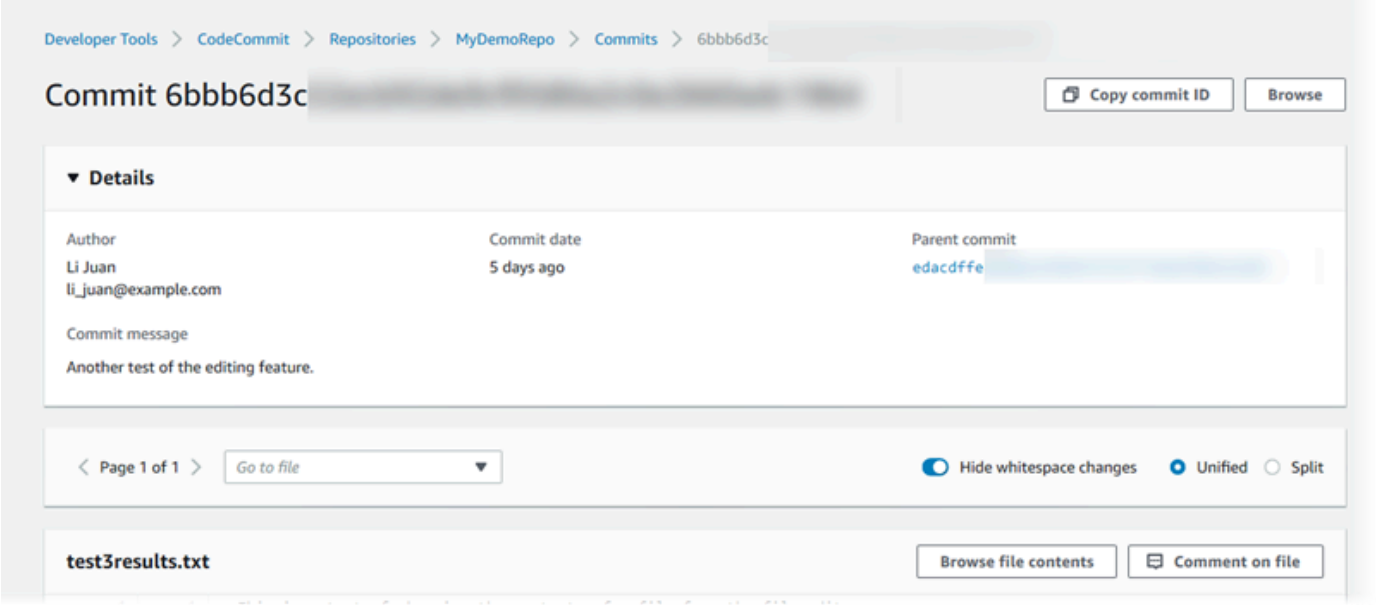
Topics

- Compare a commit to its parent
- Compare any two commit specifiers

Compare a commit to its parent

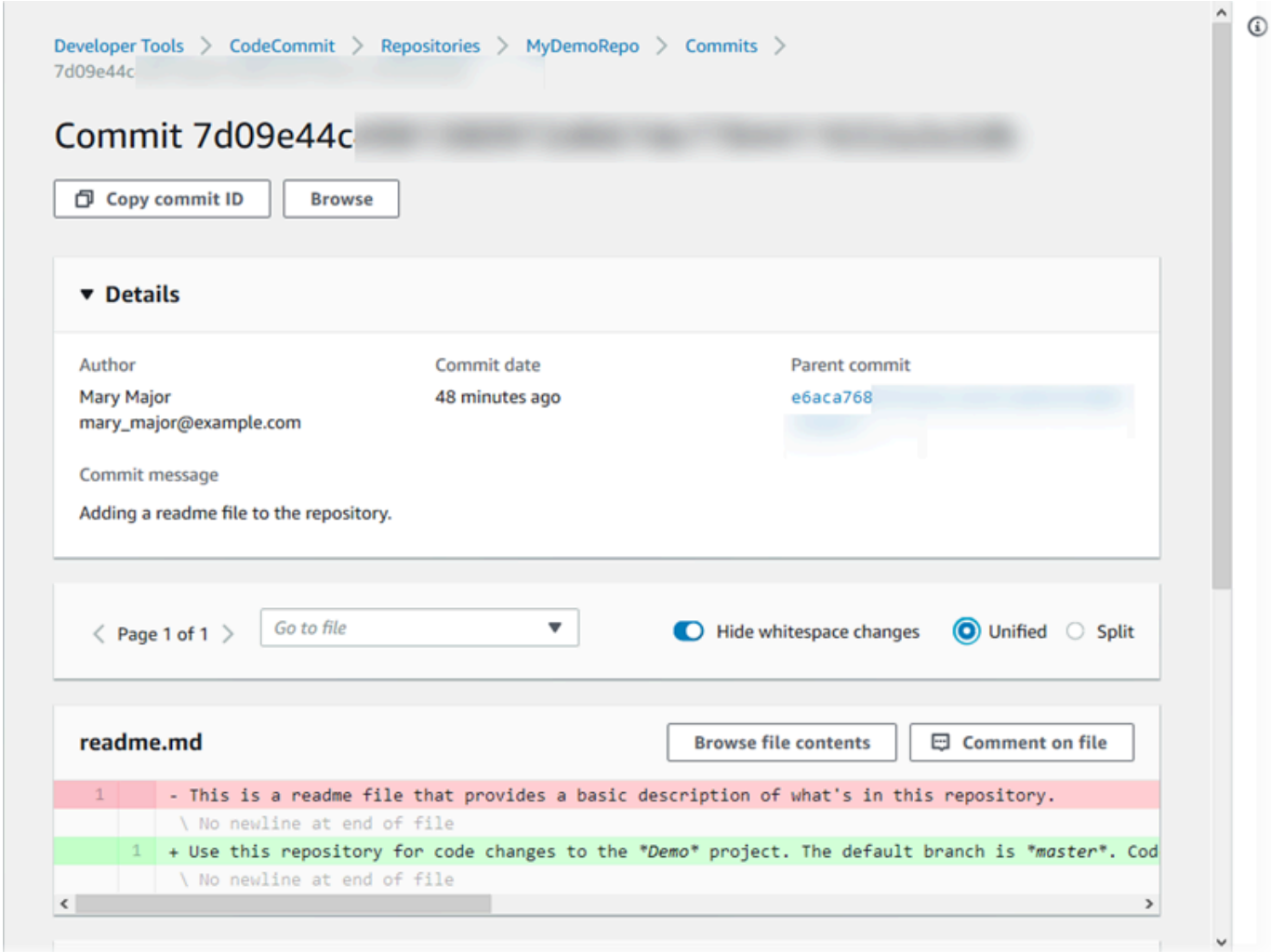
You can quickly view the difference between a commit and its parent to review the commit message, the committer, and what changed.

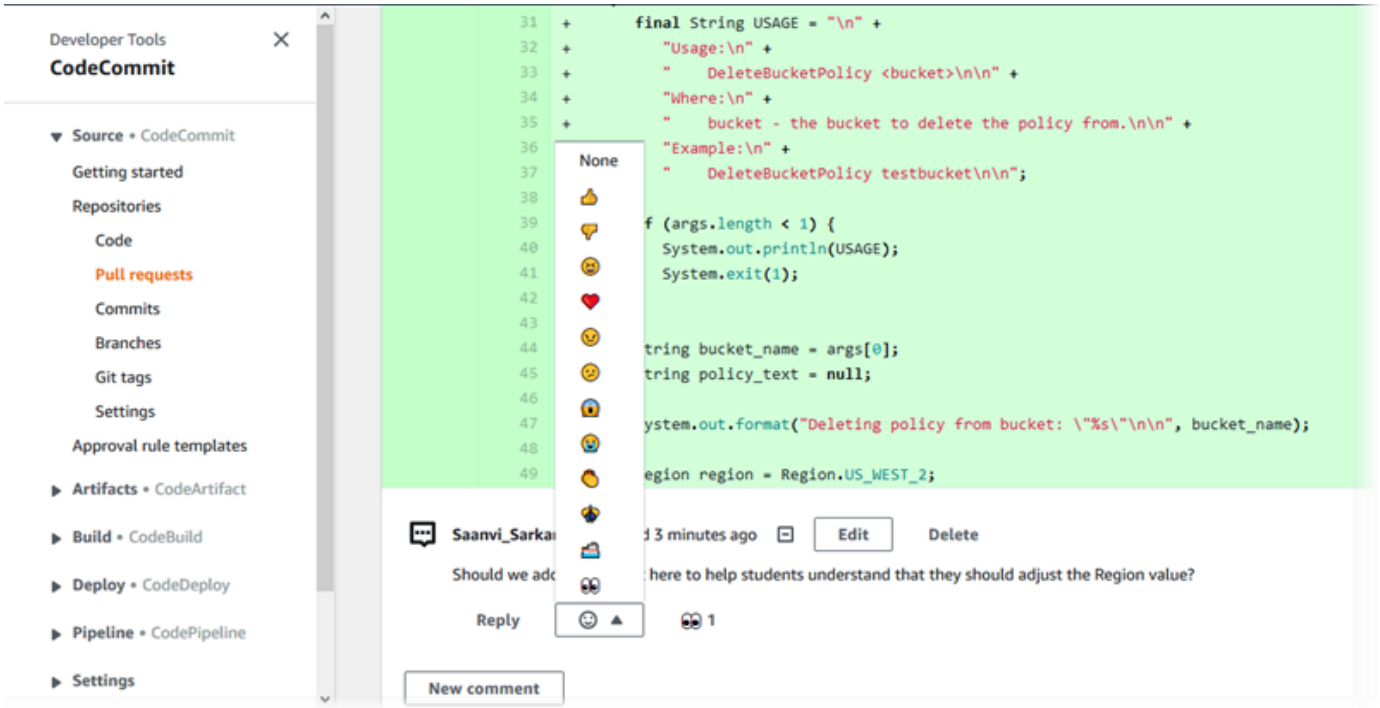
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. On the **Repositories** page, choose the repository where you want to view the difference between a commit and its parent.
3. In the navigation pane, choose **Commits**.
4. Choose the abbreviated commit ID of any commit in the list. The view changes to show details for this commit, including the differences between it and its parent commit.



You can show changes side by side (**Split** view) or inline (**Unified** view). You can also hide or show white space changes. You can also add comments. For more information, see Comment on a commit.

> **Note**
>
> Your preferences for viewing code and other console settings are saved as browser cookies whenever you change them. For more information, see Working with user preferences.


Developer Tools > CodeCommit > Repositories > MyDemoRepo > Commits >
7d09e44c
Commit 7d09e44c
Copy commit ID
Browse
Details
Author
Mary Major
mary_major@example.com
Commit date
48 minutes ago
Parent commit
e6aca768
Commit message
Adding a readme file to the repository.
Page 1 of 1
Go to file
Hide whitespace changes
Unified
Split
readme.md
Browse file contents
Comment on file
1 - This is a readme file that provides a basic description of what's in this repository.
\ No newline at end of file
1 + Use this repository for code changes to the *Demo* project. The default branch is *master*. Cod
\ No newline at end of file



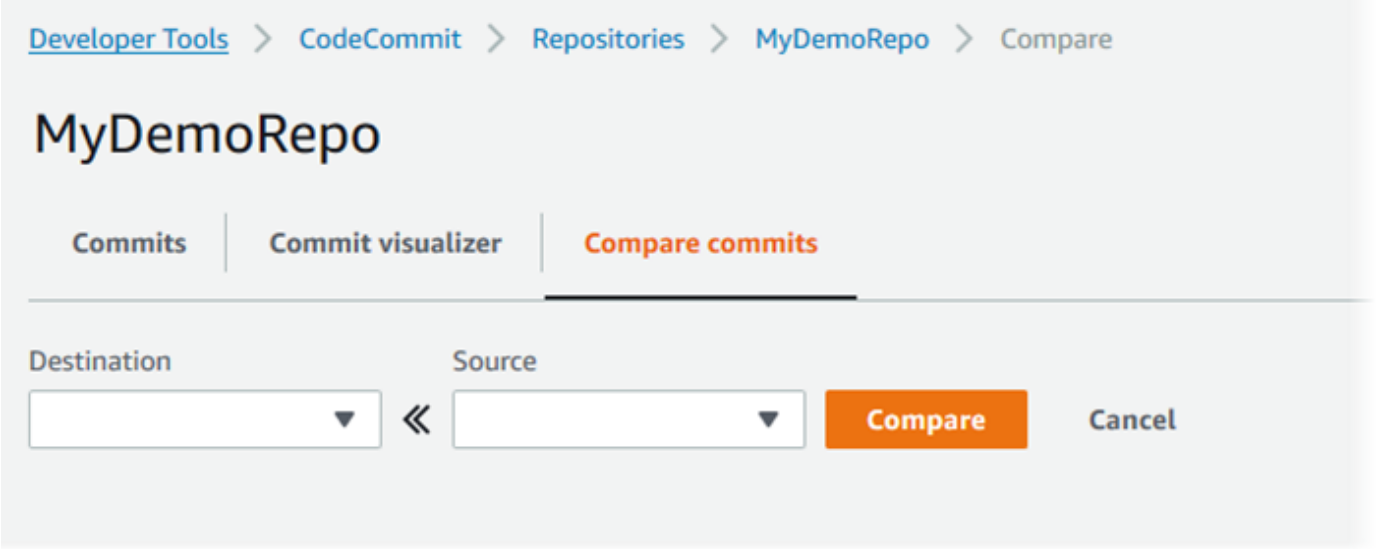
> **Note**
>
> Depending on line ending style, your code editor, and other factors, you might see entire lines added or deleted instead of specific changes in a line. The level of detail matches what's returned in the **git show** or **git diff** commands.

5. To compare a commit to its parent, from the **Commit visualizer** tab, choose the abbreviated commit ID. The commit details, including the changes between the commit and its parent, are displayed.

Compare any two commit specifiers

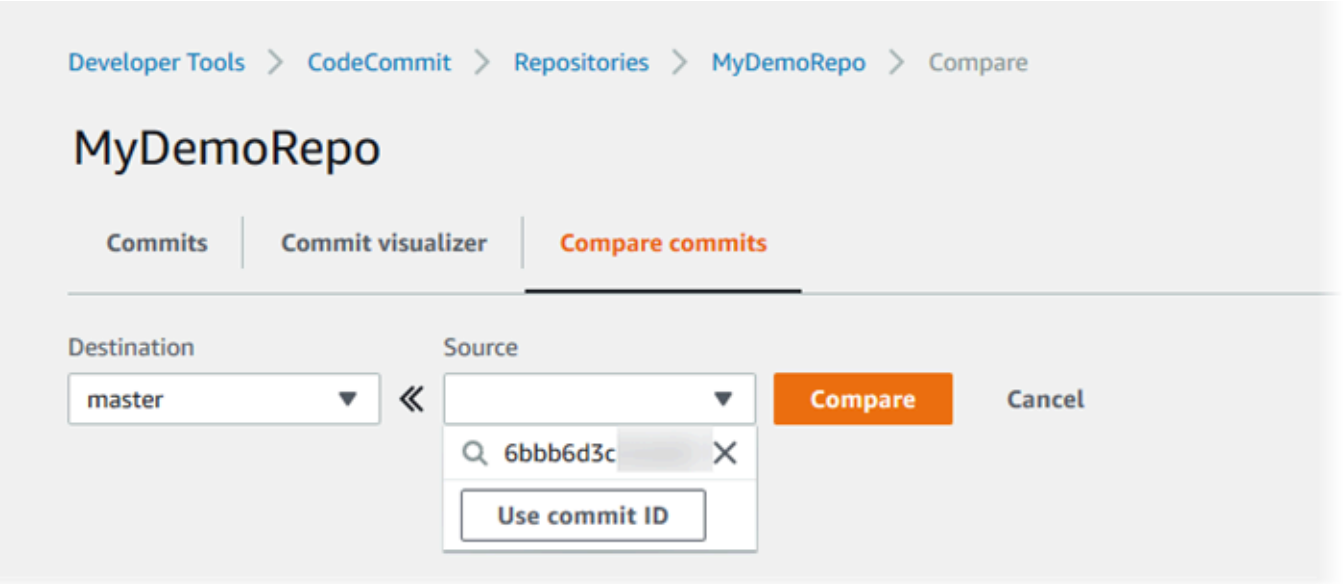
You can view the differences between any two commit specifiers in the CodeCommit console. Commit specifiers are references, such as branches, tags, and commit IDs.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. On the **Repositories** page, choose the repository where you want to compare commits, branches, or tagged commits.
3. In the navigation pane, choose **Commits**, and then choose **Compare commits**.

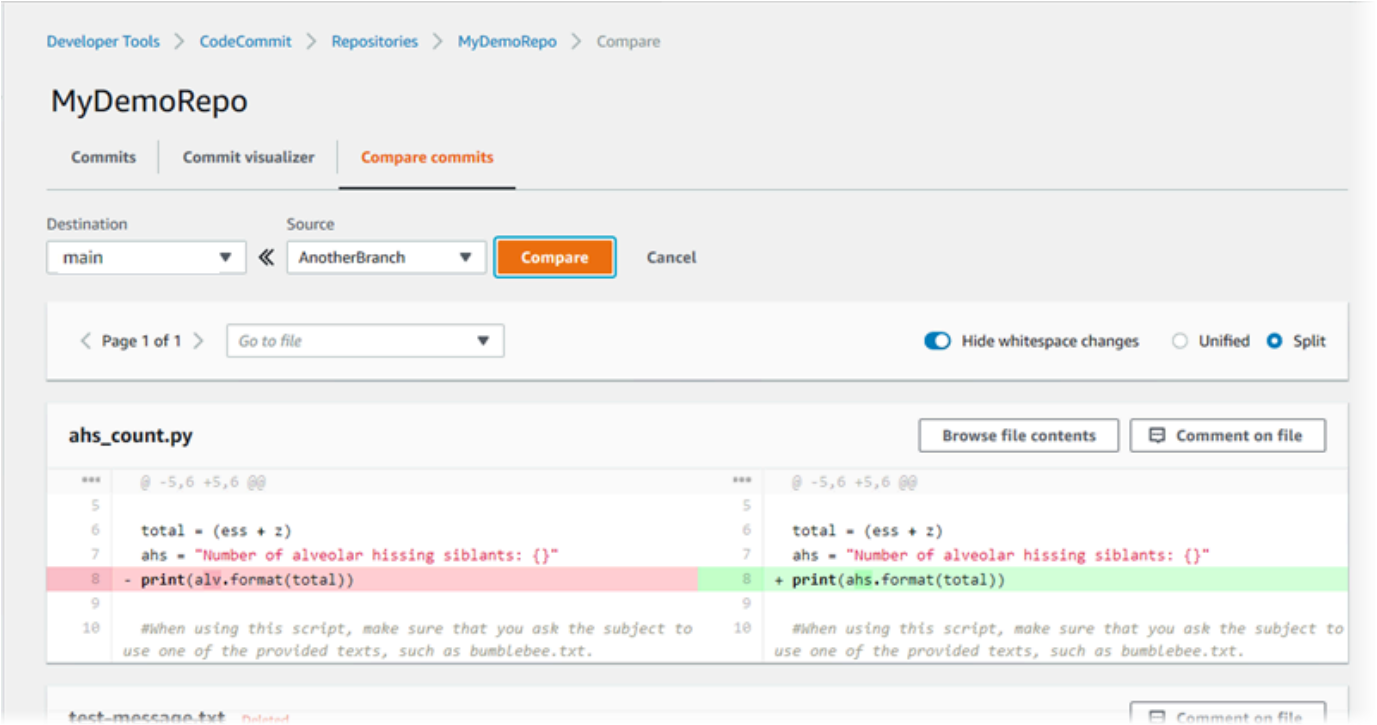


4. Use the boxes to compare two commit specifiers.

   - To compare the tip of a branch, choose the branch name from the list. This selects the most recent commit from that branch for the comparison.
   - To compare a commit with a specific tag associated with it, choose the tag name from the list, if any. This selects the tagged commit for the comparison.
   - To compare a specific commit, enter or paste the commit ID in the box. To get the full commit ID, choose **Commits** in the navigation bar, and copy the commit ID from the list. On the **Compare commits** page, paste the full commit ID in the text box, and choose **Use commit ID**.



5. After you have selected the specifiers, choose **Compare**.



You can show differences side by side (**Split** view) or inline (**Unified** view). You can also hide or show white space changes.

6. To clear your comparison choices, choose **Cancel**.

Comment on a commit in AWS CodeCommit

You can use the CodeCommit console to comment on commits in a repository, and view and reply to other users' comments on commits. This can help you discuss changes made in a repository, including:

- Why changes were made.
- Whether more changes are required.
- Whether changes should be merged into another branch.

You can comment on an overall commit, a file in a commit, or a specific line or change in a file. You can also link to a line of code by selecting the line and then copying the resulting URL in your browser.

> **Note**
>
> For best results, use commenting when you are signed in as an IAM user. The commenting functionality is not optimized for users who sign in with root account credentials, federated access, or temporary credentials.

Topics

- View comments on a commit in a repository
- Add and reply to comments on a commit in a repository
- View, add, update, and reply to commments (AWS CLI)

View comments on a commit in a repository

You can use the CodeCommit console to view comments on a commit.

To view comments on a commit

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository for which you want to review comments on commits.
3. In the navigation pane, choose **Commits**. Choose the commit ID of the commit where you want to view any comments.

   The page for that commit is displayed, along with any comments.

Add and reply to comments on a commit in a repository

You can use the CodeCommit console to add comments to the comparison of a commit and a parent, or to the comparison between two specified commits. You can also reply to comments with emojis, with your own comments, or both.

Add and reply to comments on a commit (console)

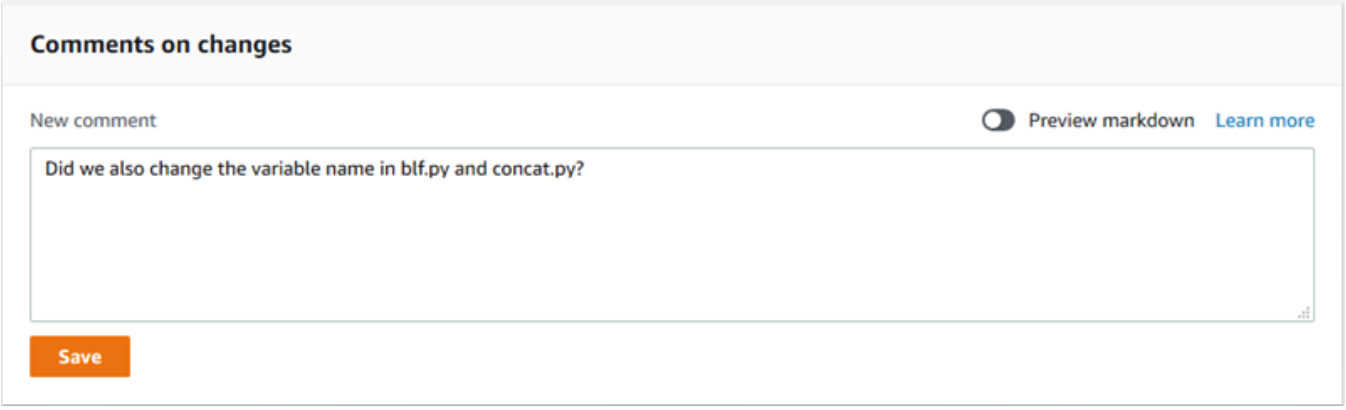
You can add and reply to comments to a commit with text and with emojis. Your comments and emojis are marked as belonging to the IAM user or role you used to sign in to the console.

To add and reply to comments on a commit

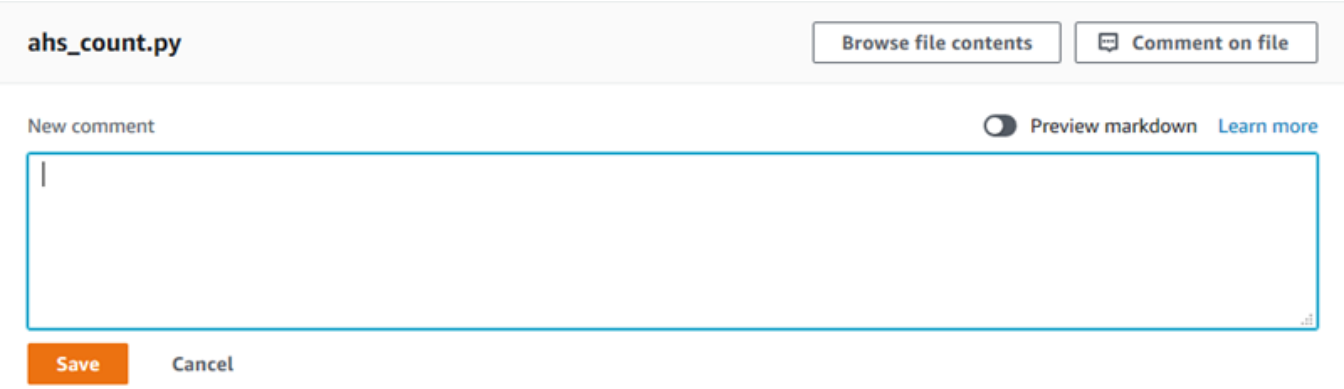
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to comment on commits.
3. In the navigation pane, choose **Commits**. Choose the commit ID of the commit where you want to add or reply to comments.

   The page for that commit is displayed, along with any comments.
4. To add a comment, do one of the following:

   - To add a general comment, in **Comments on changes**, enter your comment, and then choose **Save**. You can use Markdown, or you can enter your comment in plaintext.



   - To add a comment to a file in the commit, find the name of the file. Choose **Comment on file**, enter your comment, and then choose **Save**.



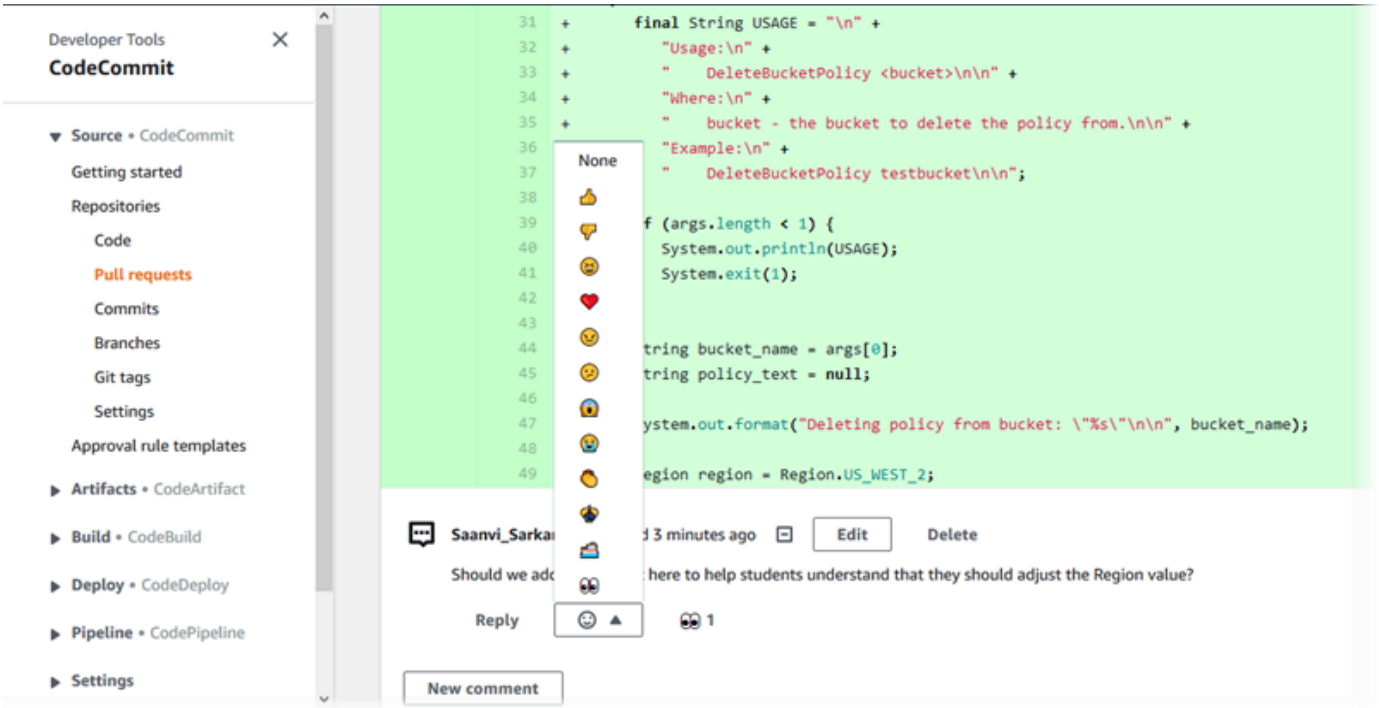
   - To add a comment to a changed line in the commit, go to the line where the change appears. Choose the comment bubble

enter your comment, and then choose **Save**.



> **Note**
>
> You can edit your comment after you have saved it. You can also delete its contents. The comment will remain with a message saying that the contents have been deleted. Consider using the **Preview markdown** mode for your comment before you save it.

5. To reply to comments on a commit, choose **Reply**. To reply to a comment with an emoji, choose the emoji you want from the list. You can only choose one emoji per comment. If you want to change your emoji reaction, choose a different one from the list, or choose **None** to remove your reaction.

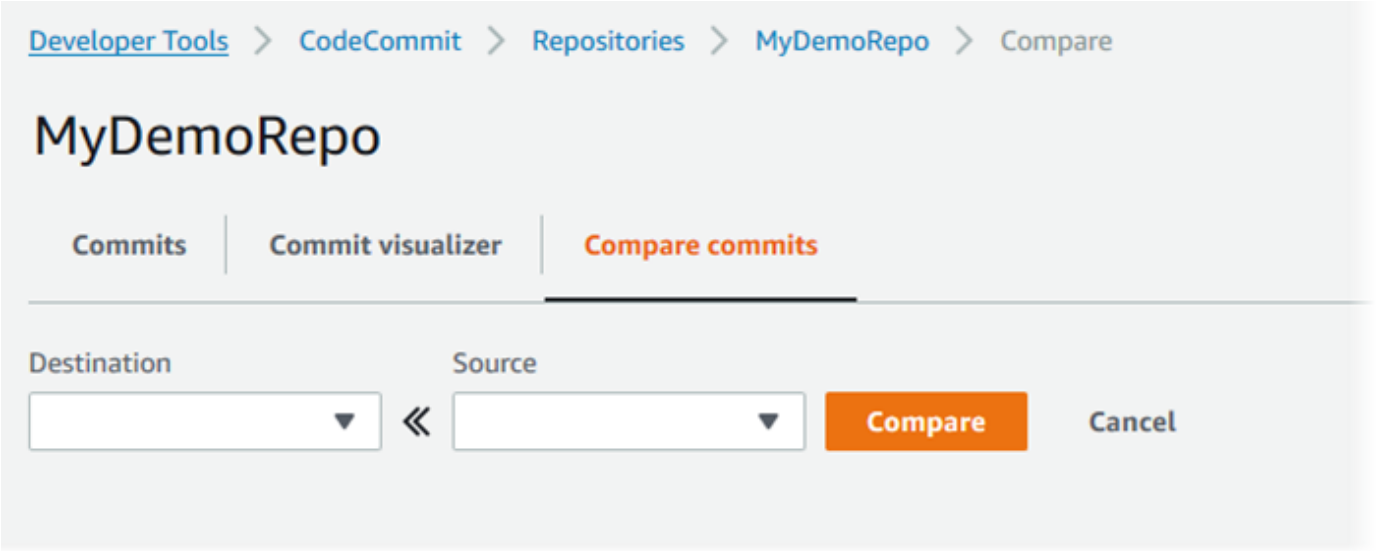


Add and reply to comments when comparing two commit specifiers

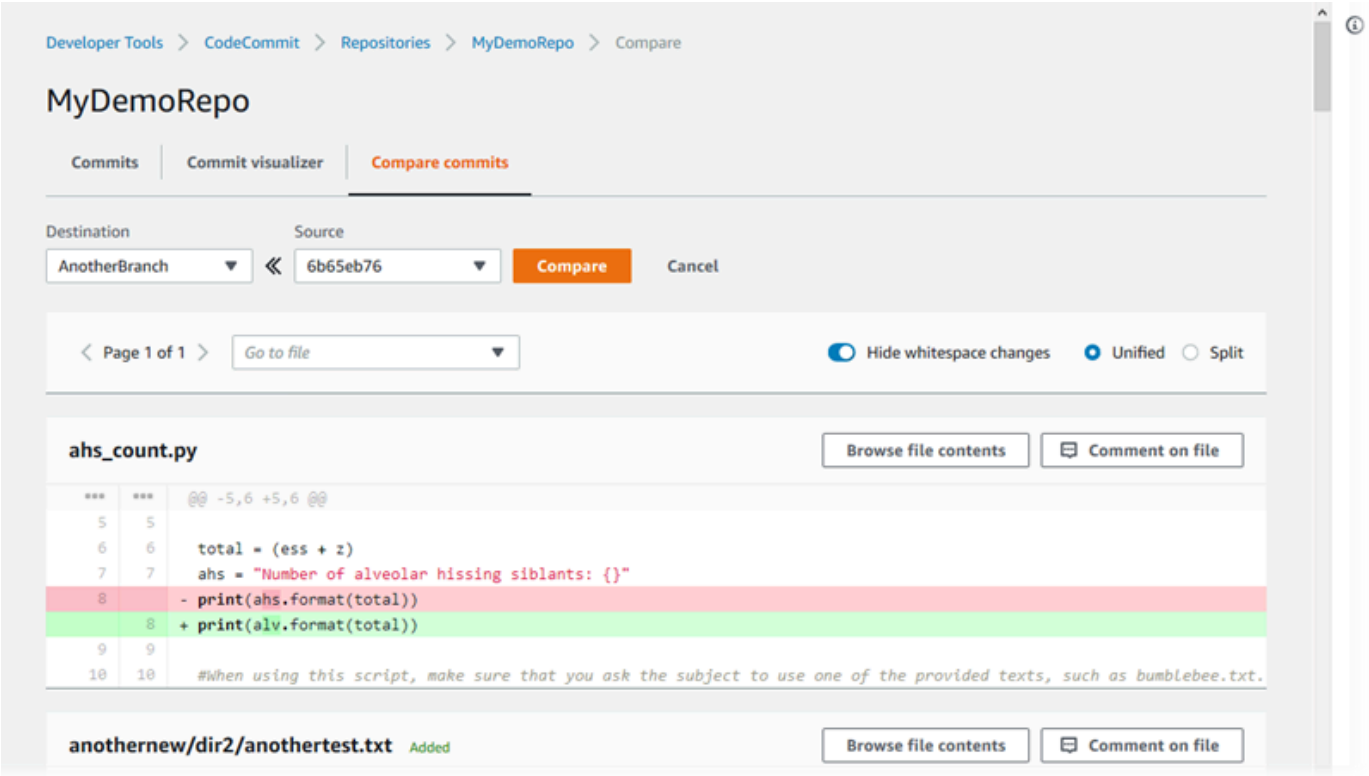
You can add comments to a comparison between branches, tags, or commits.

To add or reply to comments when comparing commit specifiers

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the repository where you want to compare commits, branches, or tagged commits.
3. In the navigation pane, choose **Commits**, and then choose the **Compare commits** tab.



4. Use the **Destination** and **Source** fields to compare two commit specifiers. Use the drop-down lists or paste in commit IDs. Choose **Compare**.



5. Do one or more of the following:

   - To add comments to files or lines, choose the comment bubble

     .
   - To add general comments on the compared changes, go to **Comments on changes**.

View, add, update, and reply to commments (AWS CLI)

You can view, add, reply, update, and delete the contents of a comment by running the following commands:

- To view the comments on the comparison between two commits, run **get-comments-for-compared-commit**.
- To view details on a comment, run **get-comment**.
- To delete the contents of a comment that you created, run **delete-comment-content**.
- To create a comment on the comparison between two commits, run **post-comment-for-compared-commit**.
- To update a comment, run **update-comment**.
- To reply to a comment, run **post-comment-reply**.
- To reply to a comment with an emoji, run **put-comment-reaction**.
- To view emoji reactions to a comment, run **get-comment-reactions**.

To view comments on a commit

1. Run the **get-comments-for-compared-commit** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).
   - The full commit ID of the after commit, to establish the directionality of the comparison (with the `--after-commit-id option`).
   - The full commit ID of the before commit, to establish the directionality of the comparison (with the `--before-commit-id` option).
   - (Optional) An enumeration token to return the next batch of the results (with the `--next-token` option).
   - (Optional) A non-negative integer to limit the number of returned results (with the `--max-results` option).

   For example, to view comments made on the comparison between two commits in a repository named *MyDemoRepo*:

```
aws codecommit get-comments-for-compared-commit --repository-name MyDemoRepo --
before-commit-id 6e147360EXAMPLE --after-commit-id 317f8570EXAMPLE
```

2. If successful, this command produces output similar to the following:

```
{
   "commentsForComparedCommitData": [
      {
         "afterBlobId": "1f330709EXAMPLE",
         "afterCommitId": "317f8570EXAMPLE",
         "beforeBlobId": "80906a4cEXAMPLE",
         "beforeCommitId": "6e147360EXAMPLE",
         "comments": [
            {
               "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
               "clientRequestToken": "123Example",
               "commentId": "ff30b348EXAMPLEb9aa670f",
               "content": "Whoops - I meant to add this comment to the line, not
 the file, but I don't see how to delete it.",
               "creationDate": 1508369768.142,
               "deleted": false,
               "CommentId": "123abc-EXAMPLE",
               "lastModifiedDate": 1508369842.278,
               "callerReactions": [],
               "reactionCounts":
                {
                  "SMILE" : 6,
                  "THUMBSUP" : 1
                }
            },
            {
               "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
               "clientRequestToken": "123Example",
               "commentId": "553b509bEXAMPLE56198325",
               "content": "Can you add a test case for this?",
               "creationDate": 1508369612.240,
               "deleted": false,
               "commentId": "456def-EXAMPLE",
               "lastModifiedDate": 1508369612.240,
               "callerReactions": [],
               "reactionCounts":
                {
```

```
                  "THUMBSUP" : 2
               }
            }
         ],
         "location": {
            "filePath": "cl_sample.js",
            "filePosition": 1232,
            "relativeFileVersion": "after"
         },
         "repositoryName": "MyDemoRepo"
      }
   ],
   "nextToken": "exampleToken"
}
```

To view details of a comment on a commit

1. Run the **get-comment** command, specifying the system-generated comment ID. For example:

```
aws codecommit get-comment --comment-id ff30b348EXAMPLEb9aa670f
```

2. If successful, this command returns output similar to the following:

```
{
   "comment": {
      "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
      "clientRequestToken": "123Example",
      "commentId": "ff30b348EXAMPLEb9aa670f",
      "content": "Whoops - I meant to add this comment to the line, but I don't see
 how to delete it.",
      "creationDate": 1508369768.142,
      "deleted": false,
      "commentId": "",
      "lastModifiedDate": 1508369842.278,
      "callerReactions": [],
      "reactionCounts":
         {
           "SMILE" : 6,
           "THUMBSUP" : 1
         }
   }
```

```
}
```

To delete the contents of a comment on a commit

1. Run the **delete-comment-content** command, specifying the system-generated comment ID. For example:

```
aws codecommit delete-comment-content --comment-id ff30b348EXAMPLEb9aa670f
```

> **Note**
>
> You can only delete the content of a comment if you have the AWSCodeCommitFullAccess policy applied, or if you have the `DeleteCommentContent` permission set to **Allow**.

2. If successful, this command produces output similar to the following:

```
{
   "comment": {
      "creationDate": 1508369768.142,
      "deleted": true,
      "lastModifiedDate": 1508369842.278,
      "clientRequestToken": "123Example",
      "commentId": "ff30b348EXAMPLEb9aa670f",
      "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
      "callerReactions": [],
      "reactionCounts":
         {
           "CLAP" : 1
         }
   }
}
```

To create a comment on a commit

1. Run the **post-comment-for-compared-commit** command, specifying:

   - The name of the CodeCommit repository (with the `--repository-name` option).

- The full commit ID of the after commit, to establish the directionality of the comparison (with the `--after-commit-id` option).
- The full commit ID of the before commit, to establish the directionality of the comparison (with the `--before-commit-id` option).
- A unique, client-generated idempotency token (with the **--client-request-token** option).
- The content of your comment (with the **--content** option).
- A list of location information about where to place the comment, including:
  - The name of the file being compared, including its extension and subdirectory, if any (with the **filePath** attribute).
  - The line number of the change within a compared file (with the **filePosition** attribute).
  - Whether the comment on the change is before or after in the comparison between the source and destination branches (with the **relativeFileVersion** attribute).

For example, to add the comment *"Can you add a test case for this?"* on the change to the *cl_sample.js* file in the comparison between two commits in a repository named *MyDemoRepo*:

```
aws codecommit post-comment-for-compared-commit --repository-name MyDemoRepo
 --before-commit-id 317f8570EXAMPLE --after-commit-id 5d036259EXAMPLE --client-
request-token 123Example --content "Can you add a test case for this?" --location
 filePath=cl_sample.js,filePosition=1232,relativeFileVersion=AFTER
```

2. If successful, this command produces output similar to the following:

```
{
        "afterBlobId": "1f330709EXAMPLE",
        "afterCommitId": "317f8570EXAMPLE",
        "beforeBlobId": "80906a4cEXAMPLE",
        "beforeCommitId": "6e147360EXAMPLE",
        "comment": {
              "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
              "clientRequestToken": "",
              "commentId": "553b509bEXAMPLE56198325",
              "content": "Can you add a test case for this?",
              "creationDate": 1508369612.203,
              "deleted": false,
              "commentId": "abc123-EXAMPLE",
              "lastModifiedDate": 1508369612.203,
```

```
            "callerReactions": [],
            "reactionCounts": []
          },
          "location": {
            "filePath": "cl_sample.js",
            "filePosition": 1232,
            "relativeFileVersion": "AFTER"
          },
        "repositoryName": "MyDemoRepo"
}
```

To update a comment on a commit

1. Run the **update-comment** command, specifying the system-generated comment ID and the content to replace any existing content.

   For example, to add the content *"Fixed as requested. I'll update the pull request."* to a comment with an ID of *442b498bEXAMPLE5756813*:

   ```
   aws codecommit update-comment --comment-id 442b498bEXAMPLE5756813 --content "Fixed
    as requested. I'll update the pull request."
   ```

2. If successful, this command produces output similar to the following:

   ```
   {
       "comment": {
           "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
           "clientRequestToken": "",
           "commentId": "442b498bEXAMPLE5756813",
           "content": "Fixed as requested. I'll update the pull request.",
           "creationDate": 1508369929.783,
           "deleted": false,
           "lastModifiedDate": 1508369929.287,
           "callerReactions": [],
           "reactionCounts":
             {
               "THUMBSUP" : 2
             }
       }
   }
   ```

To reply to a comment on a commit

1. To post a reply to a comment in a pull request, run the **post-comment-reply** command, specifying:

   - The system-generated ID of the comment to which you want to reply (with the **--in-reply-to** option).
   - A unique, client-generated idempotency token (with the **--client-request-token** option).
   - The content of your reply (with the **--content** option).

   For example, to add the reply *"Good catch. I'll remove them."* to the comment with the system-generated ID of *abcd1234EXAMPLEb5678efgh*:

   ```
   aws codecommit post-comment-reply --in-reply-to abcd1234EXAMPLEb5678efgh --
   content "Good catch. I'll remove them." --client-request-token 123Example
   ```

2. If successful, this command produces output similar to the following:

   ```
   {
       "comment": {
           "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
           "clientRequestToken": "123Example",
           "commentId": "442b498bEXAMPLE5756813",
           "content": "Good catch. I'll remove them.",
           "creationDate": 1508369829.136,
           "deleted": false,
           "CommentId": "abcd1234EXAMPLEb5678efgh",
           "lastModifiedDate": 150836912.221,
           "callerReactions": [],
           "reactionCounts": []
       }
    }
   ```

To reply to a comment on a commit with an emoji

1. To reply to a comment in a pull request with an emoji, or to change the value of your emoji reaction, run the **put-comment-reaction** command, specifying:

   - The system-generated ID of the comment to which you want to reply with an emoji.

- The value of the reaction you want to add or update. Acceptable values include supported emojis, shortcodes, and Unicode values.

The following values are supported for emojis in CodeCommit:

Emoji	Shortcode	Unicode
#	:thumbsup:	U+1F44D
#	:thumbsdown:	U+1F44E
#	:smile:	U+1F604
❤	:heart:	U+2764
#	:angry:	U+1F620
#	:confused:	U+1F615
#	:scream:	U+1F631
#	:sob:	U+1F62D
#	:clap:	U+1F44F
#	:confetti_ball:	U+1F38A
#	:ship:	U+1F6A2
#	:eyes:	U+1F440
	none	U+0000

For example, to add the emoji *:thumbsup:* to the comment with the system-generated ID of *abcd1234EXAMPLEb5678efgh*:

```
aws codecommit put-comment-reaction --comment-id abcd1234EXAMPLEb5678efgh --
reaction-value :thumbsup:
```

2. If successful, this command produces no output.

To view emoji reactions to a comment

1. To view emoji reactions to a comment, including the users who reacted with those emojis, run the **get-comment-reactions** command, specifying the system-generated ID of the comment.

   For example, to view emoji reactions to the comment with the system-generated ID of *abcd1234EXAMPLEb5678efgh*:

```
aws codecommit get-comment-reactions --comment-id abcd1234EXAMPLEb5678efgh
```

2. If successful, this command produces output similar to the following:

```
{
    "reactionsForComment": {
        [
           {
                "reaction": {
                    "emoji:"#",
                    "shortCode": "thumbsup",
                    "unicode": "U+1F44D"
                },
                "users": [
                    "arn:aws:iam::123456789012:user/Li_Juan",
                    "arn:aws:iam::123456789012:user/Mary_Major",
                    "arn:aws:iam::123456789012:user/Jorge_Souza"
                ]
            },
            {
                "reaction": {
                    "emoji": "#",
                    "shortCode": "thumbsdown",
                    "unicode": "U+1F44E"
                },
                "users": [
                    "arn:aws:iam::123456789012:user/Nikhil_Jayashankar"
                ]
            },
            {
                "reaction": {
                    "emoji": "#",
```

```
                    "shortCode": "confused",
                    "unicode": "U+1F615"
                },
                "users": [
                    "arn:aws:iam::123456789012:user/Saanvi_Sarkar"
                ]
            }
        ]
    }
}
```

Create a Git tag in AWS CodeCommit

You can use a Git tag to mark a commit with a label that helps other repository users understand its importance. To create a Git tag in a CodeCommit repository, you can use Git from a local repo connected to the CodeCommit repository. After you have created a Git tag in the local repo, you can use **git push --tags** to push it to the CodeCommit repository.

For more information, see View tag details.

Use Git to create a tag

Follow these steps to use Git from a local repo to create a Git tag in a CodeCommit repository.

In these steps, we assume that you have already connected the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. Run the **git tag *new-tag-name commit-id*** command, where *new-tag-name* is the new Git tag's name and *commit-id* is the ID of the commit to associate with the Git tag.

   For example, the following command creates a Git tag named `beta` and associates it with the commit ID `dc082f9a...af873b88`:

   ```
   git tag beta dc082f9a...af873b88
   ```

2. To push the new Git tag from the local repo to the CodeCommit repository, run the **git push *remote-name new-tag-name*** command, where *remote-name* is the name of the CodeCommit repository and *new-tag-name* is the name of the new Git tag.

   For example, to push a new Git tag named `beta` to a CodeCommit repository named `origin`:

```
git push origin beta
```

> **Note**
>
> To push all new Git tags from your local repo to the CodeCommit repository, run **git push --tags**.
> To ensure your local repo is updated with all of the Git tags in the CodeCommit repository, run **git fetch** followed by **git fetch --tags**.

For more options, see your Git documentation.

View Git tag details in AWS CodeCommit

In Git, a tag is a label you can apply to a reference like a commit to mark it with information that might be important to other repository users. For example, you might tag the commit that was the beta release point for a project with the tag **`beta`**. For more information, see Use Git to create a tag. Git tags are different from repository tags. For more information about how to use repository tags, see Add a tag to a repository.

You can use the AWS CodeCommit console to view information about Git tags in your repository, including the date and commit message of the commit referenced by each Git tag. From the console, you can compare the commit referenced by the tag with the head of the default branch of your repository. Like any other commit, you can also view the code at the point of that Git tag.

You can also use Git from your terminal or command line to view details about Git tags in a local repo.

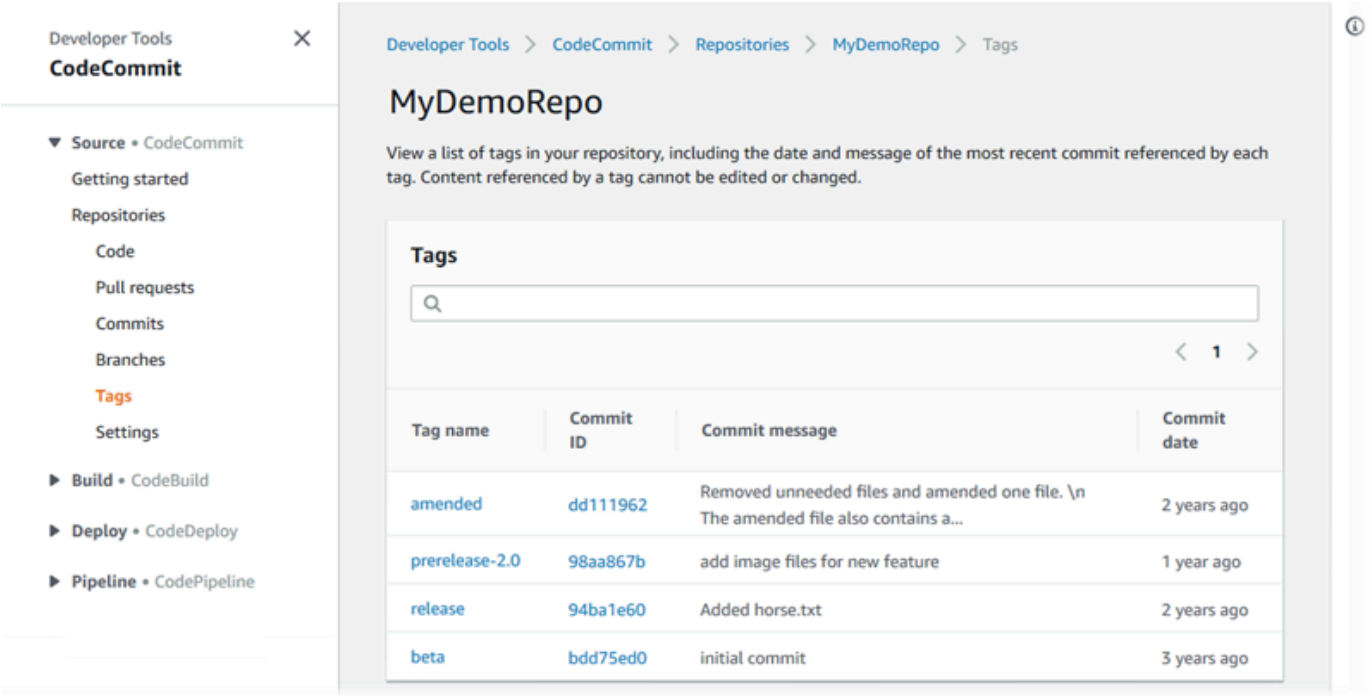
Topics

- View tag details (console)
- View Git tag details (Git)

View tag details (console)

Use the AWS CodeCommit console to quickly view a list of Git tags for your repository and details about the commits referenced by the Git tags.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to view tags.
3. In the navigation pane, choose **Git tags**.



4. Do one of the following:
   - To view the code as it was at that commit, choose the Git tag name.
   - To view details of the commit, including the full commit message, committer, and author, choose the abbreviated commit ID.

View Git tag details (Git)

To use Git to view details about Git tags in a local repo, run one of the following commands:

- git tag to view a list of Git tag names.
- git show to view information about a specific Git tag.
- git ls-remote to view information about Git tags in a CodeCommit repository.

> **Note**
>
> To ensure that your local repo is updated with all of the Git tags in the CodeCommit repository, run **git fetch** followed by **git fetch --tags**.

In the following steps, we assume that you have already connected the local repo to a CodeCommit repository. For instructions, see Connect to a repository.

To view a list of Git tags in a local repo

1. Run the **git tag** command:

```
git tag
```

2. If successful, this command produces output similar to the following:

```
beta
release
```

> **Note**
>
> If no tags have been defined, **git tag** returns nothing.

For more options, see your Git documentation.

To view information about a Git tag in a local repo

1. Run the **git show *tag-name*** command. For example, to view information about a Git tag named `beta`, run:

```
git show beta
```

2. If successful, this command produces output similar to the following:

```
commit 317f8570...ad9e3c09
Author: John Doe <johndoe@example.com>
Date:   Tue Sep 23 13:49:51 2014 -0700
```

```
    Added horse.txt

diff --git a/horse.txt b/horse.txt
new file mode 100644
index 0000000..df42ff1
--- /dev/null
+++ b/horse.txt
@@ -0,0 +1 @@
+The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus
\ No newline at end of file
```

> **Note**
>
> To exit the output of the Git tag information, type **:q**.

For more options, see your Git documentation.

To view information about Git tags in a CodeCommit repository

1. Run the **git ls-remote --tags** command.

```
git ls-remote --tags
```

2. If successful, this command produces as output a list of the Git tags in the CodeCommit repository:

```
129ce87a...70fbffba    refs/tags/beta
785de9bd...59b402d8    refs/tags/release
```

   If no Git tags have been defined, **git ls-remote --tags** returns a blank line.

For more options, see your Git documentation.

Delete a Git tag in AWS CodeCommit

To delete a Git tag in a CodeCommit repository, use Git from a local repo connected to the CodeCommit repository. .

Use Git to delete a Git tag

Follow these steps to use Git from a local repo to delete a Git tag in a CodeCommit repository.

These steps are written with the assumption that you have already connected the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. To delete the Git tag from the local repo, run the **git tag -d *tag-name*** command where *tag-name* is the name of the Git tag you want to delete.

   > **Tip**
   >
   > To get a list of Git tag names, run **git tag**.

   For example, to delete a Git tag in the local repo named `beta`:

   ```
   git tag -d beta
   ```

2. To delete the Git tag from the CodeCommit repository, run the **git push *remote-name* --delete *tag-name*** command where *remote-name* is the nickname the local repo uses for the CodeCommit repository and *tag-name* is the name of the Git tag you want to delete from the CodeCommit repository.

   > **Tip**
   >
   > To get a list of CodeCommit repository names and their URLs, run the **git remote -v** command.

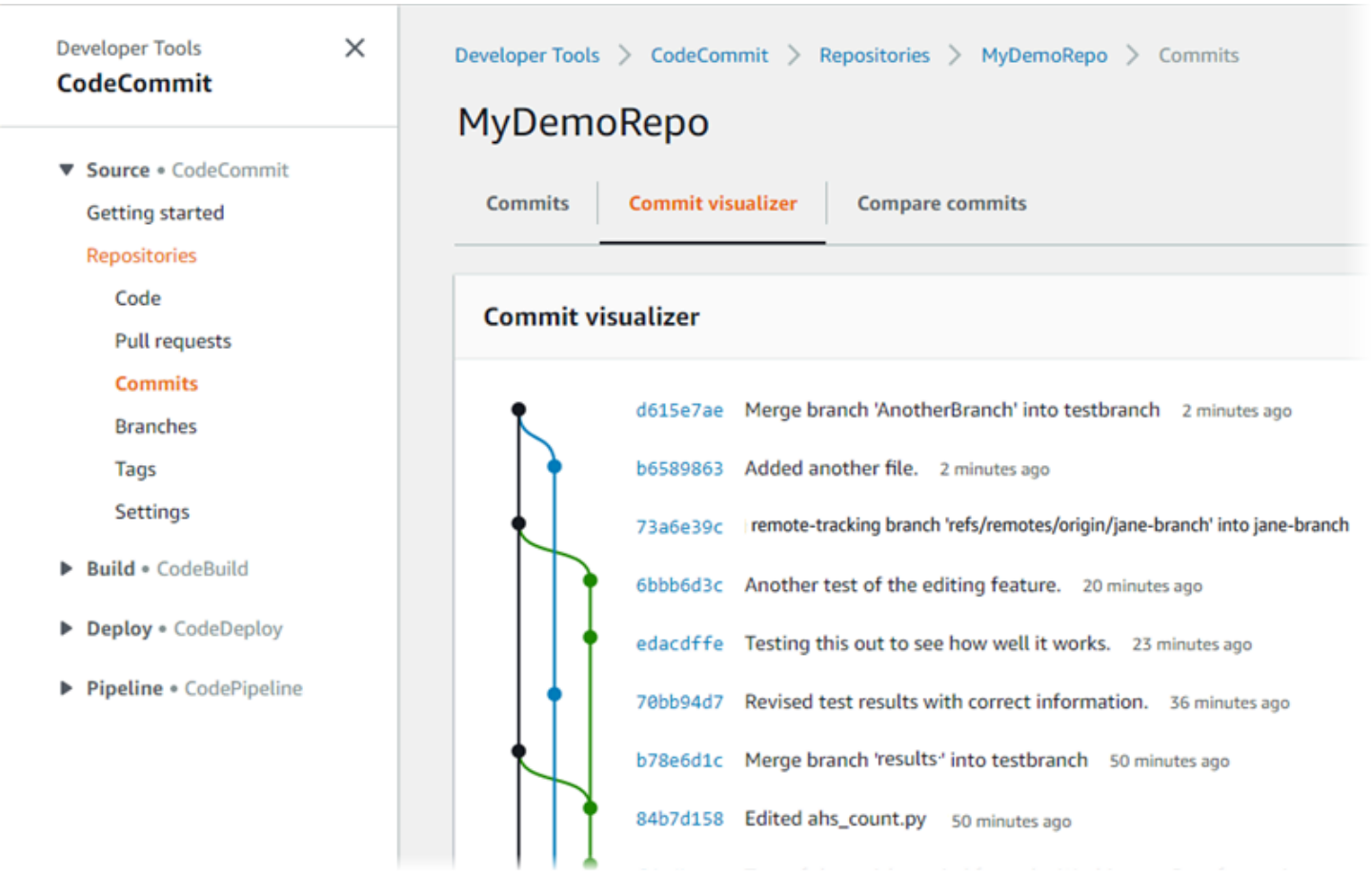
   For example, to delete a Git tag named `beta` in the CodeCommit repository named `origin`:

   ```
   git push origin --delete beta
   ```

Working with branches in AWS CodeCommit repositories

What is a branch? In Git, branches are pointers or references to a commit. In development, they're a convenient way to organize your work. You can use branches to separate work on a new or different version of files without affecting work in other branches. You can use branches to develop new features, store a specific version of your project from a particular commit, and more. When you create your first commit, a *default branch* is created for you. This default branch is the one used as the base or default branch in local repositories (repos) when users clone the repository. The name of that default branch varies depending on how you create your first commit. If you add the first file to your repository by using the CodeCommit console, the AWS CLI, or one of the SDKs, the name of that default branch is *main*. This is the default branch name used in the examples in this guide. If you push your first commit using a Git client, the name of the default branch is what the Git client specifies as its default. Consider configuring your Git client to use *main* as the name for the initial branch.

In CodeCommit, you can change the default branch for your repository. You can also create and delete branches and view details about a branch. You can quickly compare differences between a branch and the default branch (or any two branches). To view the history of branches and merges in your repository, you can use the Commit visualizer, which is shown in the following graphic.



For information about working with other aspects of your repository in CodeCommit, see Working with repositories, Working with files, Working with pull requests, Working with commits, and Working with user preferences.

Topics

- Create a branch in AWS CodeCommit
- Limit pushes and merges to branches in AWS CodeCommit
- View branch details in AWS CodeCommit
- Compare and merge branches in AWS CodeCommit
- Change branch settings in AWS CodeCommit
- Delete a branch in AWS CodeCommit

Create a branch in AWS CodeCommit

You can use the CodeCommit console or the AWS CLI to create branches for your repository. This is a quick way to separate work on a new or different version of files without impacting work in the default branch. After you create a branch in the CodeCommit console, you must pull that change to your local repo. Alternatively, you can create a branch locally and then use Git from a local repo connected to the CodeCommit repository to push that change.

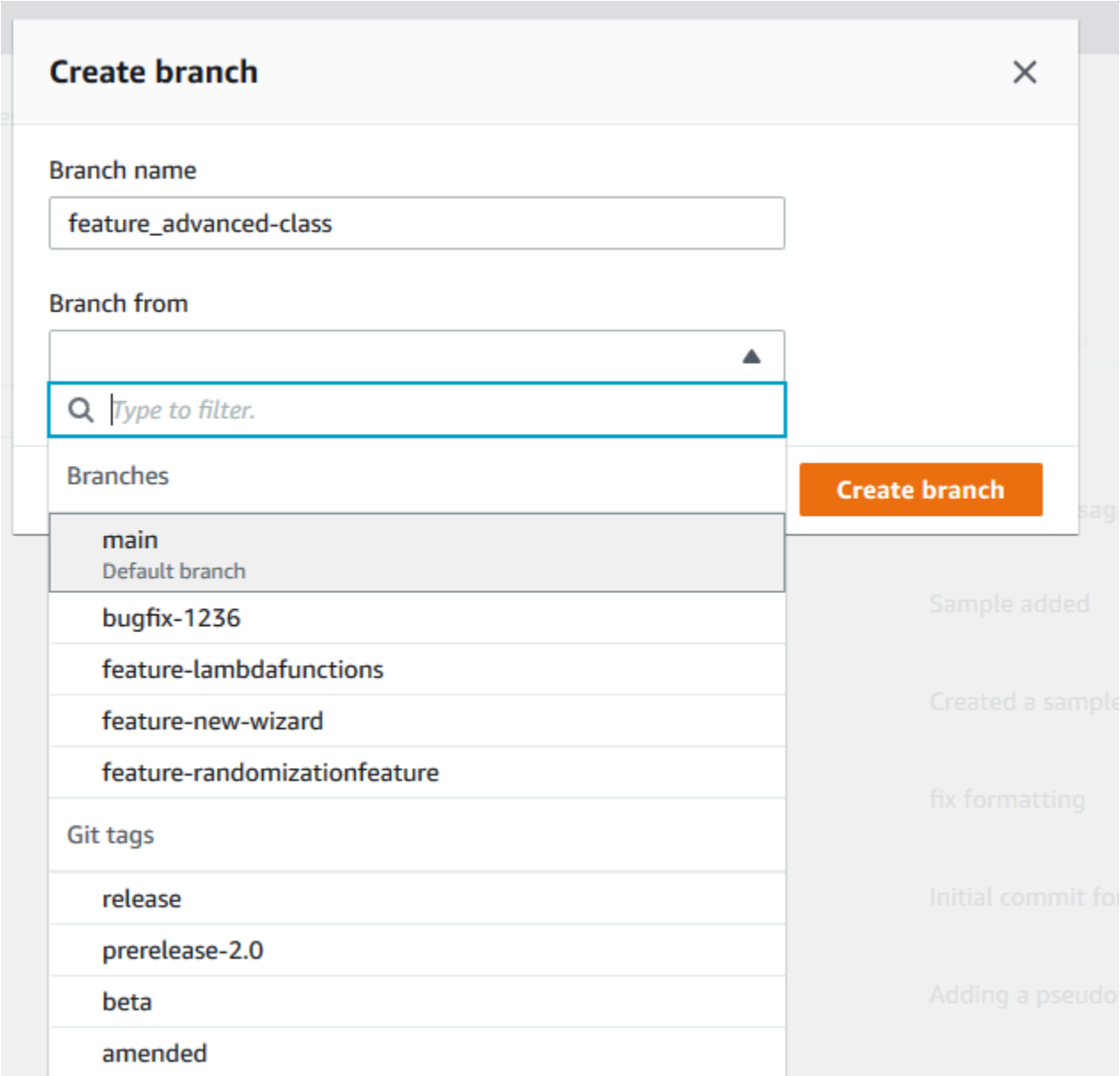
Topics

- Create a branch (console)
- Create a branch (Git)
- Create a branch (AWS CLI)

Create a branch (console)

You can use the CodeCommit console to create a branch in a CodeCommit repository. The next time users pull changes from the repository, they see the new branch.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to create a branch.
3. In the navigation pane, choose **Branches**.
4. Choose **Create branch**.



In **Branch name**, enter a name for the branch. In **Branch from**, choose a branch or tag from the list, or paste a commit ID. Choose **Create branch**.

Create a branch (Git)

Follow these steps to use Git from a local repo to create a branch in a local repo and then push that branch to the CodeCommit repository.

These steps are written with the assumption that you have already connected the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. Create a branch in your local repo by running the **git checkout -b *new-branch-name*** command, where *new-branch-name* is the name of the new branch.

   For example, the following command creates a branch named `MyNewBranch` in the local repo:

   ```
   git checkout -b MyNewBranch
   ```

2. To push the new branch from the local repo to the CodeCommit repository, run the **git push** command, specifying both the ***remote-name*** and the ***new-branch-name***.

   For example, to push a new branch in the local repo named `MyNewBranch` to the CodeCommit repository with the nickname `origin`:

   ```
   git push origin MyNewBranch
   ```

> **Note**
>
> If you add the `-u` option to **git push** (for example, **git push -u origin main**), then in the future you can run **git push** without *remote-name branch-name*. Upstream tracking information is set. To get upstream tracking information, run **git remote show *remote-name*** (for example, **git remote show origin**).
> To see a list of all of your local and remote tracking branches, run **git branch --all**.
> To set up a branch in the local repo that is connected to a branch in the CodeCommit repository, run **git checkout *remote-branch-name***.

For more options, see your Git documentation.

Create a branch (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

Follow these steps to use the AWS CLI to create a branch in a CodeCommit repository and then push that branch to the CodeCommit repository. For steps to create an initial commit and specify the name of the default branch for an empty repository, see Create the first commit for a repository using the AWS CLI.

1. Run the **create-branch** command, specifying:

   - The name of the CodeCommit repository where the branch is created (with the **--repository-name** option).

     > **Note**
     >
     > To get the name of the CodeCommit repository, run the list-repositories command.

   - The name of the new branch (with the **--branch-name** option).
   - The ID of the commit to which the new branch points (with the **--commit-id** option).

   For example, to create a branch named `MyNewBranch` that points to commit ID `317f8570EXAMPLE` in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit create-branch --repository-name MyDemoRepo --branch-name MyNewBranch
    --commit-id 317f8570EXAMPLE
   ```

   This command produces output only if there are errors.

2. To update the list of available CodeCommit repository branches in your local repo with the new remote branch name, run **git remote update *remote-name***.

   For example, to update the list of available branches for the CodeCommit repository with the nickname `origin`:

   ```
   git remote update origin
   ```

   > **Note**
   >
   > Alternatively, you can run the **git fetch** command. You can also view all remote branches by running **git branch --all**, but until you update the list of your local repo, the remote branch you created does not appear in the list.
   > For more options, see your Git documentation.

3. To set up a branch in the local repo that is connected to the new branch in the CodeCommit repository, run **git checkout *remote-branch-name***.

> **Note**
>
> To get a list of CodeCommit repository names and their URLs, run the **git remote -v** command.

Limit pushes and merges to branches in AWS CodeCommit

By default, any CodeCommit repository user who has sufficient permissions to push code to the repository can contribute to any branch in that repository. This is true no matter how you add a branch to the repository: by using the console, the command line, or Git. However, you might want to configure a branch so that only some repository users can push or merge code to that branch. For example, you might want to configure a branch used for production code so that only a subset of senior developers can push or merge changes to that branch. Other developers can still pull from the branch, make their own branches, and create pull requests, but they cannot push or merge changes to that branch. You can configure this access by creating a conditional policy that uses a context key for one or more branches in IAM.

> **Note**
>
> To complete some of the procedures in this topic, you must sign in with an adminstrative user that has sufficient permissions to configure and apply IAM policies. For more information, see Creating an IAM Admin User and Group.

Topics

- Configure an IAM policy to limit pushes and merges to a branch
- Apply the IAM policy to an IAM group or role
- Test the policy

Configure an IAM policy to limit pushes and merges to a branch

You can create a policy in IAM that prevents users from updating a branch, including pushing commits to a branch and merging pull requests to a branch. To do this, your policy uses a conditional statement, so that the effect of the Deny statement applies only if the condition is met. The APIs you include in the Deny statement determine which actions are not allowed. You

can configure this policy to apply to only one branch in a repository, a number of branches in a repository, or to all branches that match the criteria across all repositories in an Amazon Web Services account.

To create a conditional policy for branches

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. Choose **JSON**, and then paste the following example policy. Replace the value of `Resource` with the ARN of the repository that contains the branch for which you want to restrict access. Replace the value of `codecommit:References` with a reference to the branch or branches to which you want to restrict access. For example, this policy denies pushing commits, merging branches, deleting branches, deleting files, merging pull requests, and adding files to a branch named *`main`* and a branch named *`prod`* in a repository named *`MyDemoRepo`*:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "codecommit:GitPush",
                "codecommit:DeleteBranch",
                "codecommit:DeleteFile",
                "codecommit:PutFile",
                "codecommit:MergeBranchesByFastForward",
                "codecommit:MergeBranchesBySquash",
                "codecommit:MergeBranchesByThreeWay",
                "codecommit:MergePullRequestByFastForward",
                "codecommit:MergePullRequestBySquash",
                "codecommit:MergePullRequestByThreeWay"
            ],
            "Resource": "arn:aws-eusc:codecommit:us-
east-2:111111111111:MyDemoRepo",
            "Condition": {
                "StringEqualsIfExists": {
```

```
                    "codecommit:References": [
                        "refs/heads/main",
                        "refs/heads/prod"
                     ]
                },
                "Null": {
                    "codecommit:References": "false"
                }
            }
        }
    ]
}
```

Branches in Git are simply pointers (references) to the SHA-1 value of the head commit, which is why the condition uses `References`. The `Null` statement is required in any policy whose effect is `Deny` and where `GitPush` is one of the actions. This is required because of the way Git and `git-receive-pack` work when pushing changes from a local repo to CodeCommit.

> **Tip**
>
> To create a policy that applies to all branches named main in all repositories in an Amazon Web Services account, change the value of `Resource` from a repository ARN to an asterisk (*).

5. Choose **Review policy**. Correct any errors in your policy statement, and then continue to **Create policy**.
6. When the JSON is validated, the **Create policy** page is displayed. A warning appears in the **Summary** section, advising you that this policy does not grant permissions. This is expected.
   - In **Name**, enter a name for this policy, such as **`DenyChangesToMain`**.
   - In **Description**, enter a description of the policy's purpose. This is optional, but recommended.
   - Choose **Create policy**.

Apply the IAM policy to an IAM group or role

You've created a policy that limits pushes and merges to a branch, but the policy has no effect until you apply it to an IAM user, group, or role. As a best practice, consider applying the policy to an IAM group or role. Applying policies to individual IAM users does not scale well.

To apply the conditional policy to a group or role

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, if you want to apply the policy to an IAM group, choose **Groups**.If you want to apply the policy to a role that users assume, choose **Role**. Choose the name of the group or role.
3. On the **Permissions** tab, choose **Attach Policy**.
4. Select the conditional policy you created from the list of policies, and then choose **Attach policy**.

For more information, see Attaching and Detatching IAM Policies.

Test the policy

You should test the effects of the policy you've applied on the group or role to ensure that it acts as expected. There are many ways you can do this. For example, to test a policy similar to the one shown above, you can:

- Sign in to the CodeCommit console with an IAM user who is either a member of an IAM group that has the policy applied, or assumes a role that has the policy applied. In the console, add a file on the branch where the restrictions apply. You should see an error message when you attempt to save or upload a file to that branch. Add a file to a different branch. The operation should succeed.
- Sign in to the CodeCommit console with an IAM user who is either a member of an IAM group that has the policy applied, or assumes a role that has the policy applied. Create a pull request that merges to the branch where the restrictions apply. You should be able to create the pull request, but get an error if you try to merge it.
- From the terminal or command line, create a commit on the branch where the restrictions apply, and then push that commit to the CodeCommit repository. You should see an error message. Commits and pushes made from other branches should work as usual.

View branch details in AWS CodeCommit

You can use the CodeCommit console to view details about the branches in a CodeCommit repository. You can view the date of the last commit to a branch, the commit message, and more. You can also use the AWS CLI or Git from a local repo connected to the CodeCommit repository.

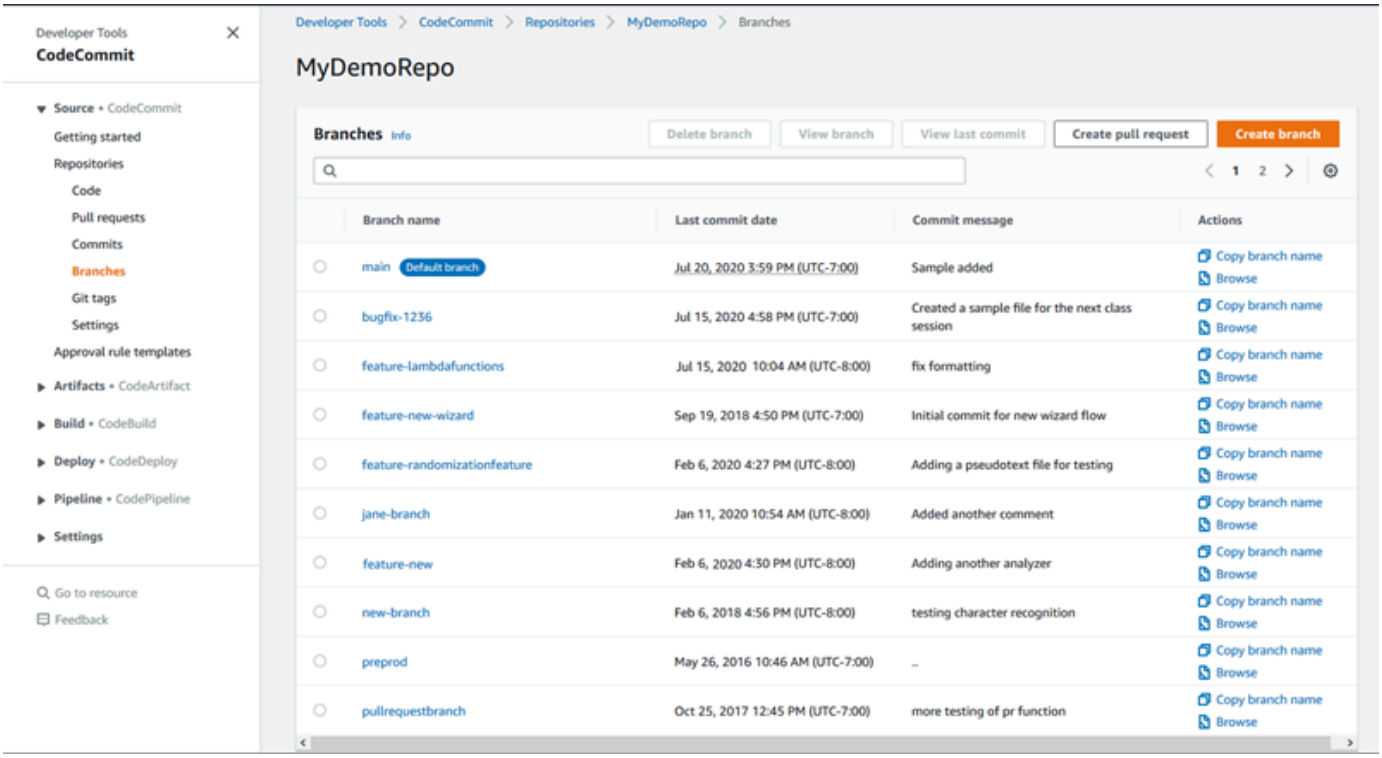
Topics

- View branch details (console)
- View branch details (Git)
- View branch details (AWS CLI)

View branch details (console)

Use the CodeCommit console to quickly view a list of branches for your repository and details about the branches.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to view branch details.
3. In the navigation pane, choose **Branches**.



4. The name of the branch used as the default for the repository is displayed next to **Default branch**. To view details about the most recent commit to a branch, choose the branch, and then choose **View last commit**. To view the files and code in a branch, choose the branch name.

View branch details (Git)

To use Git from a local repo to view details about both the local and remote tracking branches for a CodeCommit repository, run the **git branch** command.

The following steps are written with the assumption that you have already connected the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. Run the **git branch** command, specifying the **--all** option:

```
git branch --all
```

2. If successful, this command returns output similar to the following:

```
  MyNewBranch
* main
  remotes/origin/MyNewBranch
  remotes/origin/main
```

The asterisk (*) appears next to the currently open branch. The entries after that are remote tracking references.

> **Tip**
>
> **git branch** shows local branches.
> **git branch -r** shows remote branches.
> **git checkout *existing-branch-name*** switches to the specified branch name and, if **git branch** is run immediately afterward, displays it with an asterisk (*).
> **git remote update *remote-name*** updates your local repo with the list of available CodeCommit repository branches. (To get a list of CodeCommit repository names and their URLs, run the **git remote -v** command.)

For more options, see your Git documentation.

View branch details (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to view details about the branches in a CodeCommit repository, run one or more of the following commands:

- To view a list of branch names, run list-branches.
- To view information about a specific branch, run get-branch.

To view a list of branch names

1. Run the **list-branches** command, specifying the name of the CodeCommit repository (with the `--repository-name` option).

   > **Tip**
   >
   > To get the name of the CodeCommit repository, run the list-repositories command.

   For example, to view details about the branches in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit list-branches --repository-name MyDemoRepo
   ```

2. If successful, this command outputs a `branchNameList` object, with an entry for each branch.

   Here is some example output based on the preceding example command:

   ```
   {
       "branches": [
           "MyNewBranch",
           "main"
       ]
   }
   ```

To view information about a branch

1. Run the **get-branch** command, specifying:

   - The repository name (with the **--repository-name** option).
   - The branch name (with the **--branch-name** option).

   For example, to view information about a branch named `MyNewBranch` in a CodeCommit repository named `MyDemoRepo`:

   ```
   aws codecommit get-branch --repository-name MyDemoRepo --branch-name MyNewBranch
   ```

2. If successful, this command outputs the name of the branch and the ID of the last commit made to the branch.

   Here is some example output based on the preceding example command:

   ```
   {
       "branch": {
             "branchName": "MyNewBranch",
             "commitID": "317f8570EXAMPLE"
       }
   }
   ```

Compare and merge branches in AWS CodeCommit

You can use the CodeCommit console to compare branches in a CodeCommit repository. Comparing branches helps you quickly view the differences between a branch and the default branch, or view the differences between any two branches.

Topics

- Compare a branch to the default branch
- Compare two specific branches
- Merge two branches (AWS CLI)

Compare a branch to the default branch

Use the CodeCommit console to quickly view the differences between a branch and the default branch for your repository.

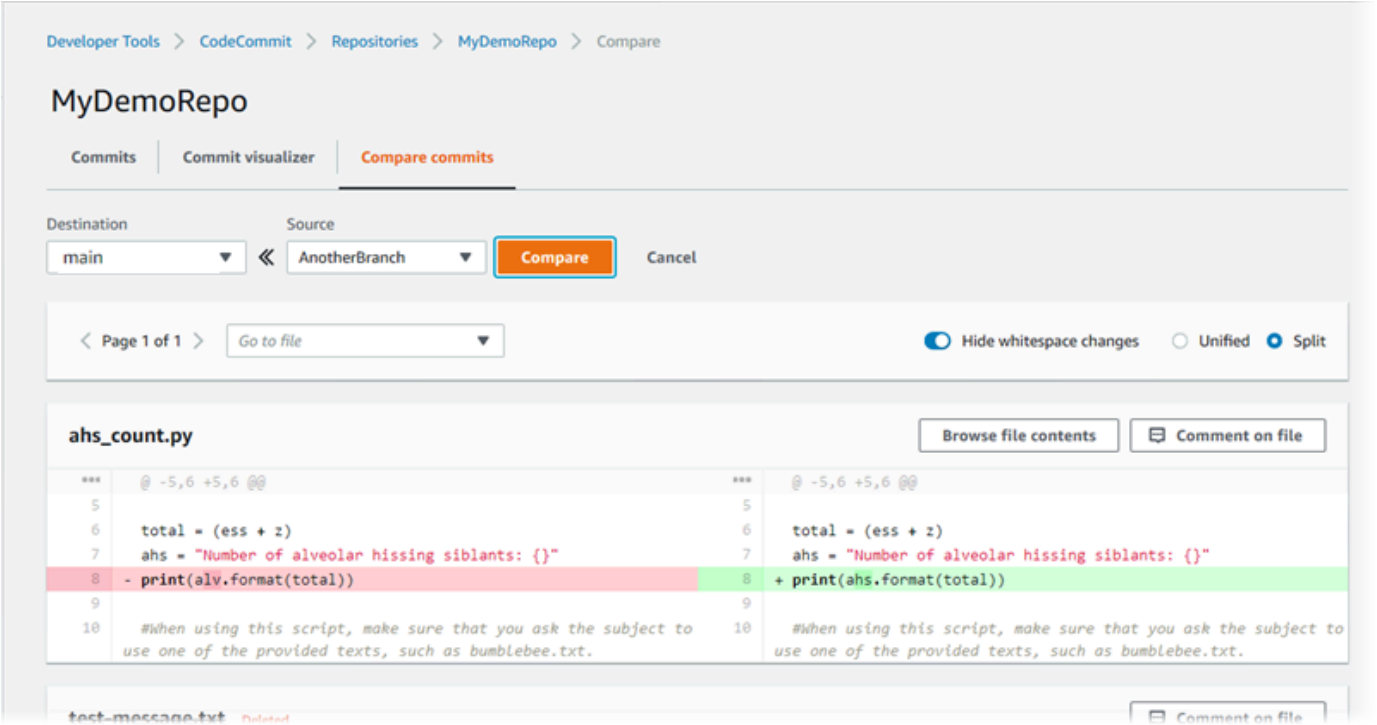
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to compare branches.
3. In the navigation pane, choose **Commits**, and then choose the **Compare commits** tab.
4. In **Destination**, choose the name of the default branch. In **Source**, choose the branch you want to compare to the default branch. Choose **Compare**.

Compare two specific branches

Use the CodeCommit console to view the differences between two branches that you want to compare.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to compare branches.
3. In the navigation pane, choose **Commits**, and then choose the **Compare commits** tab.
4. In **Destination** and **Source**, choose the two branches to compare, and then choose **Compare**. To view the list of changed files, expand the changed files list. You can view changes in files side by side (Split view) or inline (Unified view).

> **Note**
>
> If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with user preferences.



Merge two branches (AWS CLI)

You can merge two branches in a CodeCommit repository using the AWS CLI using one of the available merge strategies by running one of the following commands:

- To merge two branches using the fast-forward merge strategy, run the **merge-branches-by-fast-forward** command.
- To merge two branches using the squash merge strategy, run the **merge-branches-by-squash** command.
- To merge two branches using the three-way merge strategy, run the **merge-branches-by-three-way** command.

You can also test merges by running the **create-unreferenced-merge-commit** command. For more information, see Resolve Conflicts in a Pull Request.

> **Note**
>
> To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to merge two branches in a CodeCommit repository

1. To merge two branches using the fast-forward merge strategy, run the **merge-branches-by-fast-forward** command, specifying:

   - The name of the source branch that contains the changes you want to merge (with the **--source-commit-specifier** option).
   - The name of the destination branch where you want to merge your changes (with the **--destination-commit-specifier** option).
   - The name of the repository (with the **--repository-name** option).

   For example, to merge a source branch named *bugfix-1234* into a destination branch named *preprod* in a repository named *MyDemoRepo*:

   ```
   aws codecommit merge-branches-by-fast-forward --source-commit-specifier bugfix-bug1234 --destination-commit-specifier preprod --repository-name MyDemoRepo
   ```

   If successful, this command produces output similar to the following:

   ```
   {
       "commitId": "4f178133EXAMPLE",
       "treeId": "389765daEXAMPLE"
   }
   ```

2. To merge two branches using the squash merge strategy, run the **merge-branches-by-squash** command, specifying:

   - The name of the source branch that contains the changes you want to merge (with the **--source-commit-specifier** option).

- The name of the destination branch where you want to merge your changes (with the **--destination-commit-specifier** option).
- The name of the repository (with the **--repository-name** option).
- The commit message to include (with the **--commit-message** option).
- The name to use for the commit (with the **--name** option).
- The email address to use for the commit (with the **--email** option).

For example, to merge a source branch named *bugfix-bug1234* with a destination branch named *bugfix-quarterly* in a repository named *MyDemoRepo*:

```
aws codecommit merge-branches-by-squash --source-commit-specifier bugfix-bug1234 --
destination-commit-specifier bugfix-quarterly --author-name "Maria Garcia" --email
 "maria_garcia@example.com" --commit-message "Merging in fix branches to prepare
 for a general patch." --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "commitId": "4f178133EXAMPLE",
    "treeId": "389765daEXAMPLE"
}
```

3. To merge two branches using the three-way merge strategy, run the **merge-branches-by-three-way** command, specifying:

- The name of the source branch that contains the changes you want to merge (with the **--source-commit-specifier** option).
- The name of the destination branch where you want to merge your changes (with the **--destination-commit-specifier** option).
- The name of the repository (with the **--repository-name** option).
- The commit message to include (with the **--commit-message** option).
- The name to use for the commit (with the **--name** option).
- The email address to use for the commit (with the **--email** option).

For example, to merge a source branch named *main* with a destination branch named *bugfix-1234* in a repository named *MyDemoRepo*:

```
aws codecommit merge-branches-by-three-way --source-commit-specifier main --
destination-commit-specifier bugfix-bug1234 --author-name "Jorge Souza" --email
 "jorge_souza@example.com" --commit-message "Merging changes from main to bugfix
 branch before additional testing."  --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "commitId": "4f178133EXAMPLE",
    "treeId": "389765daEXAMPLE"
}
```

Change branch settings in AWS CodeCommit

You can change which branch to use as the default branch in the AWS CodeCommit console or with the AWS CLI. For example, if you created your first commit using a Git client that set the default branch to *master*, you could create a branch named *main*, and then change the branch settings so that the new branch is set as the default branch for the repository. To change other branch settings, you can use Git from a local repo connected to the CodeCommit repository.

Topics

- Change the default branch (console)
- Change the default branch (AWS CLI)

Change the default branch (console)

You can specify which branch is the default branch in a CodeCommit repository in the AWS CodeCommit console.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to change settings.

3. In the navigation pane, choose **Settings**.
4. In **Default branch**, choose the branch drop-down list and choose a different branch. Choose **Save**.

> **Tip**
>
> - If you do not see another branch in the drop-down list, you have not created any additional branches. You cannot change the default branch of a repository if the repository has only one branch. For more information, see Create a branch in AWS CodeCommit.
> - If you do not see the **Default branch** section but instead see items for notification rules and connections, you are in the general settings menu for the console. The settings menu for repositories is listed under **Repositories** at the same level as **Code** and **Pull requests**.

Change the default branch (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see Command line reference.

To use the AWS CLI to change a repository's branch settings in a CodeCommit repository, run the following command:

- update-default-branch to change the default branch.

To change the default branch

1. Run the **update-default-branch** command, specifying:
   - The name of the CodeCommit repository where the default branch is updated (with the **--repository-name** option).

     > **Tip**
     >
     > To get the name of the CodeCommit repository, run the list-repositories command.

   - The name of the new default branch (with the **--default-branch-name** option).

> **Tip**
>
> To get the name of the branch, run the list-branches command.

2. For example, to change the default branch to `MyNewBranch` in a CodeCommit repository named `MyDemoRepo`:

```
aws codecommit update-default-branch --repository-name MyDemoRepo --default-branch-name MyNewBranch
```

   This command produces output only if there are errors.

For more options, see your Git documentation.

Delete a branch in AWS CodeCommit

You can use the CodeCommit console to delete a branch in a repository. Deleting a branch in CodeCommit does not delete that branch in a local repo, so users might continue to have copies of that branch until the next time they pull changes. To delete a branch locally and push that change to the CodeCommit repository, use Git from a local repo connected to the CodeCommit repository.

Deleting a branch does not delete any commits, but it does delete all references to the commits in that branch. If you delete a branch that contains commits that have not been merged into another branch in the repository, you cannot retrieve those commits unless you have their full commit IDs.

> **Note**
>
> You cannot use the instructions in this topic to delete a repository's default branch. If you want to delete the default branch, you must create a branch, make the new branch the default branch, and then delete the old branch. For more information, see Create a branch and Change branch settings.

Topics

- Delete a branch (console)
- Delete a branch (AWS CLI)

- Delete a branch (Git)

Delete a branch (console)

You can use the CodeCommit console to delete a branch in a CodeCommit repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository where you want to delete a branch.
3. In the navigation pane, choose **Branches**.
4. Find the name of the branch that you want to delete, choose **Delete branch**, and confirm your choice.

Delete a branch (AWS CLI)

You can use the AWS CLI to delete a branch in a CodeCommit repository, if that branch is not the default branch for the repository. For more information about installing and using the AWS CLI, see Command line reference.

1. At the terminal or command line, run the **delete-branch** command, specifying:
   - The name of the CodeCommit repository where the branch is to deleted (with the **--repository-name** option).

     > **Tip**
     >
     > To get the name of the CodeCommit repository, run the list-repositories command.

   - The name of the branch to delete (with the **branch-name** option).

     > **Tip**
     >
     > To get the name of the branch, run the list-branches command.

2. For example, to delete a branch named `MyNewBranch` in an CodeCommit repository named `MyDemoRepo`:

```
aws codecommit delete-branch --repository-name MyDemoRepo --branch-name MyNewBranch
```

This command returns information about the deleted branch, including the name of the deleted branch and the full commit ID of the commit that was the head of the branch. For example:

```
"deletedBranch": {
    "branchName": "MyNewBranch",
    "commitId": "317f8570EXAMPLE"
}
```

Delete a branch (Git)

Follow these steps to use Git from a local repo to delete a branch in a CodeCommit repository.

These steps are written with the assumption that you have already connected the local repo to the CodeCommit repository. For instructions, see Connect to a repository.

1. To delete the branch from the local repo, run the **git branch -D *branch-name*** command where *branch-name* is the name of the branch you want to delete.

   > **Tip**
   >
   > To get a list of branch names, run **git branch --all**.

   For example, to delete a branch in the local repo named `MyNewBranch`:

   ```
   git branch -D MyNewBranch
   ```

2. To delete the branch from the CodeCommit repository, run the **git push *remote-name* --delete *branch-name*** command where *remote-name* is the nickname the local repo uses for the CodeCommit repository and *branch-name* is the name of the branch you want to delete from the CodeCommit repository.

> **Tip**
>
> To get a list of CodeCommit repository names and their URLs, run the **git remote -v** command.

For example, to delete a branch named `MyNewBranch` in the CodeCommit repository named `origin`:

```
git push origin --delete MyNewBranch
```

> **Tip**
>
> This command does not delete a branch if it is the default branch.

For more options, see your Git documentation.

Working with user preferences

You can use the AWS CodeCommit console to configure some default settings. For example, you can change your preferences for viewing code changes either inline or in a split view. When you make a change to one of these settings, the AWS CodeCommit console sets a cookie in your browser that stores and applies your choices every time you use the console. These preferences are applied to all repositories in all regions any time you access the AWS CodeCommit console using that browser. These setting preferences are not repository-specific or region-specific. They do not have any effect on your interactions with the AWS CLI, AWS CodeCommit API, or other services that interact with AWS CodeCommit.

> **Note**
>
> User preference cookies are specific to a browser. If you clear the cookies from your browser, your preferences are cleared. Similarly, if you use a different browser to access a repository, that browser has no access to the other browser's cookies. Your preferences are not retained.

User preferences include:

- When viewing changes in code, whether to use **Unified** or **Split** view, and whether to show or hide whitespace changes.
- When viewing, editing, or authoring code, whether to use a light background or a dark background in the code editor window.

There is no one page for setting your preferences. Instead, wherever you change a preference in the console, such as how to view code changes, that change is saved and applied wherever appropriate.

Migrate to AWS CodeCommit

You can migrate a Git repository to a CodeCommit repository in a number of ways: by cloning it, mirroring it, migrating all or just some of the branches, and so on. You can also migrate local, unversioned content on your computer to CodeCommit.

The following topics show you some of the ways you can migrate a repository. Your steps might vary, depending on the type, style, or complexity of your repository and the decisions you make about what and how you want to migrate. For very large repositories, you might want to consider migrating incrementally.

> **Note**
>
> You can migrate to CodeCommit from other version control systems, such as Perforce, Subversion, or TFS, but you must first migrate to Git.
> For more options, see your Git documentation.
> Alternatively, you can review the information about migrating to Git in the *Pro Git* book by Scott Chacon and Ben Straub.

Topics

- Migrate a Git repository to AWS CodeCommit
- Migrate local or unversioned content to AWS CodeCommit
- Migrate a repository incrementally

Migrate a Git repository to AWS CodeCommit

You can migrate an existing Git repository to a CodeCommit repository. The procedures in this topic show you how to migrate a project hosted on another Git repository to CodeCommit. As part of this process, you:

- Complete the initial setup required for CodeCommit.
- Create a CodeCommit repository.
- Clone the repository and push it to CodeCommit.
- View files in the CodeCommit repository.

- Share the CodeCommit repository with your team.

Topics

- Step 0: Setup required for access to CodeCommit
- Step 1: Create a CodeCommit repository
- Step 2: Clone the repository and push to the CodeCommit repository
- Step 3: View files in CodeCommit
- Step 4: Share the CodeCommit repository

Step 0: Setup required for access to CodeCommit

Before you can migrate a repository to CodeCommit, you must create and configure an IAM user for CodeCommit and configure your local computer for access. You should also install the AWS CLI to manage CodeCommit. Although you can perform most CodeCommit tasks without it, the AWS CLI offers flexibility when working with Git at the command line or terminal.

If you are already set up for CodeCommit, you can skip ahead to Step 1: Create a CodeCommit repository.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

   For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

   For more information, see Getting Set Up with the AWS Command Line Interface.

   > **Note**
   >
   > CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

   ```
   aws codecommit help
   ```

   This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

```
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI
- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- ap-southeast-3

- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1
- ap-northeast-3
- af-south-1
- il-central-1

For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

Next, you must install Git.

- **For Linux, macOS, or Unix**:

To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28

supports configuring the branch name for initial commits. We recommend using a recent version of Git.

To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

- **For Windows:**

  To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

  To install Git, we recommend websites such as Git for Windows. If you use this link to install Git, you can accept all of the installation default settings except for the following:

  - When prompted during the **Adjusting your PATH environment** step, choose the option to use Git from the command line.
  - (Optional) If you intend to use HTTPS with the credential helper that is included in the AWS CLI instead of configuring Git credentials for CodeCommit, on the **Configuring extra options** page, make sure the **Enable Git Credential Manager** option is cleared. The Git Credential Manager is only compatible with CodeCommit if IAM users configure Git credentials. For more information, see For HTTPS users using Git credentials and Git for Windows: I installed Git for Windows, but I am denied access to my repository (403).

  > **Note**
  >
  > Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

CodeCommit supports both HTTPS and SSH authentication. To complete setup, you must configure Git credentials for CodeCommit (HTTPS, recommended for most users), an SSH key pair to use

when accessing CodeCommit (SSH), **git-remote-codecommit** (recommended for users who use federated access), or the credential helper included in the AWS CLI (HTTPS).

- For Git credentials on all supported operating systems, see Step 3: Create Git credentials for HTTPS connections to CodeCommit.
- For SSH on Linux, macOS, or Unix, see SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit.
- For SSH on Windows, see Step 3: Set up the public and private keys for Git and CodeCommit .
- For **git-remote-codecommit**, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.
- For the credential helper on Linux, macOS, or Unix, see Set Up the Credential Helper (Linux, macOS, or Unix).
- For the credential helper on Windows, see Set Up the Credential Helper (Windows).

Step 1: Create a CodeCommit repository

In this section, you use the CodeCommit console to create the CodeCommit repository you use for the rest of this tutorial. To use the AWS CLI to create the repository, see Create a repository (AWS CLI).

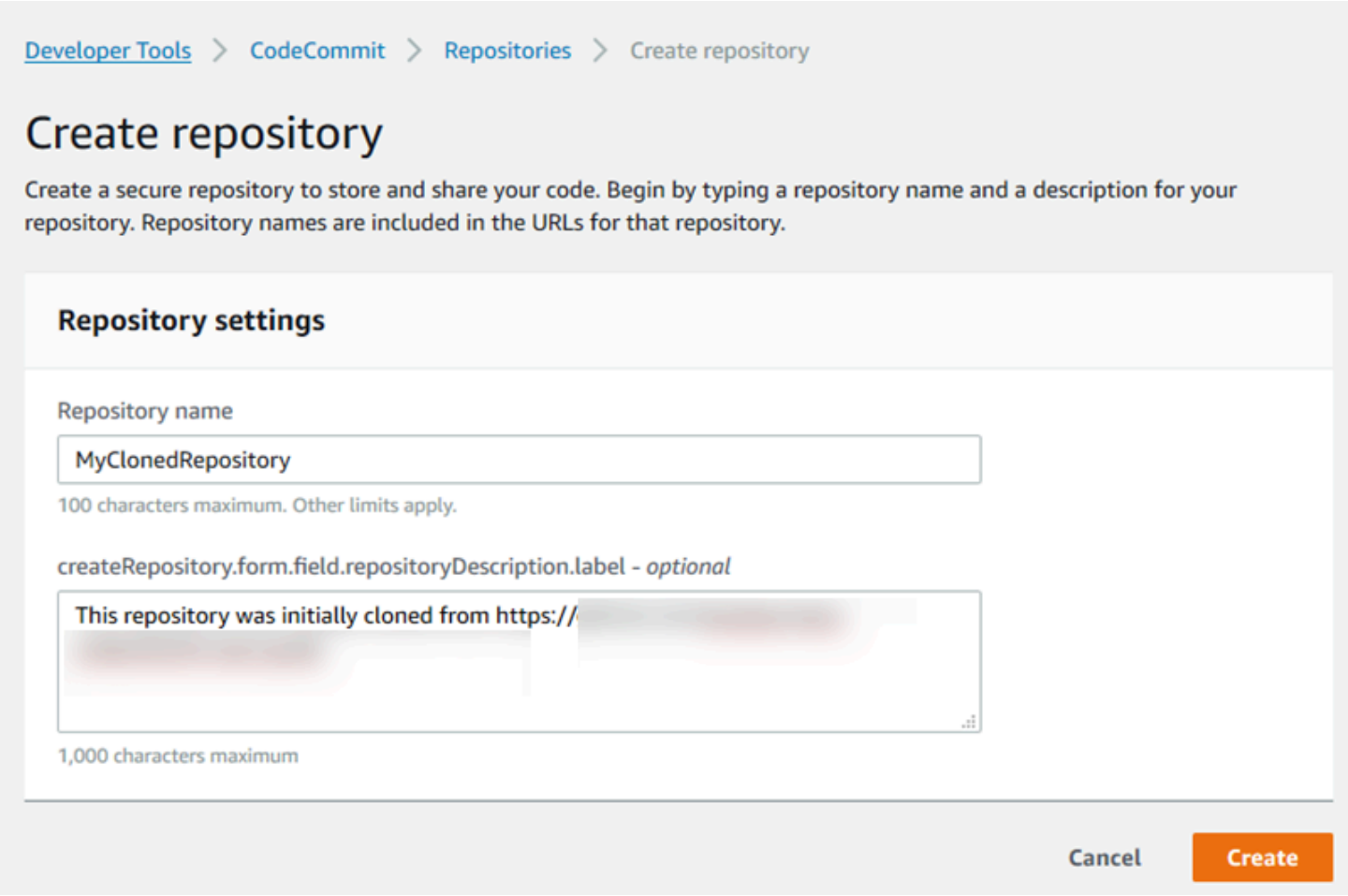
1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where you want to create the repository. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose **Create repository**.
4. On the **Create repository** page, in **Repository name**, enter a name for the repository.

   > **Note**
   >
   > Repository names are case sensitive. The name must be unique in the AWS Region for your Amazon Web Services account.

5. (Optional) In **Description**, enter a description for the repository. This can help you and other users identify the purpose of the repository.

> **Note**
>
> The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

6. (Optional) Choose **Add tag** to add one or more repository tags (a custom attribute label that helps you organize and manage your AWS resources) to your repository. For more information, see Tagging repositories in AWS CodeCommit.
7. (Optional) Expand **Additional configuration** to specify whether to use the default AWS managed key or your own customer managed key for encrypting and decrypting data in this repository. If you choose to use your own customer managed key, you must ensure that it is available in the AWS Region where you are creating the repository, and that the key is active. For more information, see AWS Key Management Service and encryption for AWS CodeCommit repositories.
8. (Optional) Select **Enable Amazon CodeGuru Reviewer for Java and Python** if this repository contains Java or Python code, and you want CodeGuru Reviewer to analyze it. CodeGuru Reviewer uses multiple machine learning models to find code defects and to suggest improvements and fixes in pull requests. For more information, see the *Amazon CodeGuru Reviewer User Guide*.
9. Choose **Create**.



After it is created, the repository appears in the **Repositories** list. In the URL column, choose the copy icon, and then choose the protocol (SSH or HTTPS) to be used to connect to CodeCommit. Copy the URL.

For example, if you named your repository *MyClonedRepository* and you are using Git credentials with HTTPS in the US East (Ohio) Region, the URL looks like the following:

```
https://git-codecommit.us-east-2.amazonaws.com/MyClonedRepository
```

You need this URL later in Step 2: Clone the repository and push to the CodeCommit repository.

Step 2: Clone the repository and push to the CodeCommit repository

In this section, you clone a Git repository to your local computer, creating what is called a local repo. You then push the contents of the local repo to the CodeCommit repository you created earlier.

1. From the terminal or command prompt on your local computer, run the **git clone** command with the `--mirror` option to clone a bare copy of the remote repository into a new folder named *aws-codecommit-demo*. This is a bare repo meant only for migration. It is not the local repo for interacting with the migrated repository in CodeCommit. You can create that later, after the migration to CodeCommit is complete.

   The following example clones a demo application hosted on GitHub (*https://github.com/awslabs/aws-demo-php-simple-app.git*) to a local repo in a directory named *aws-codecommit-demo*.

   ```
   git clone --mirror https://github.com/awslabs/aws-demo-php-simple-app.git aws-codecommit-demo
   ```

2. Change directories to the directory where you made the clone.

   ```
   cd aws-codecommit-demo
   ```

3. Run the **git push** command, specifying the URL and name of the destination CodeCommit repository and the **--all** option. (This is the URL you copied in Step 1: Create a CodeCommit repository).

   For example, if you named your repository *MyClonedRepository* and you are set up to use HTTPS, you would run the following command:

   ```
   git push https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository
    --all
   ```

   > **Note**
   >
   > The **--all** option only pushes all branches for the repository. It does not push other references, such as tags. If you want to push tags, wait until the initial push is complete, and then push again, this time using the **--tags** option:
   >
   > ```
   > git push ssh://git-codecommit.us-east-2.amazonaws.com/v1/
   > repos/MyClonedRepository --tags
   > ```

> For more information, see Git push on the Git website. For information about pushing large repositories, especially when pushing all references at once (for example, with the **--mirror** option), see Migrate a repository in increments.

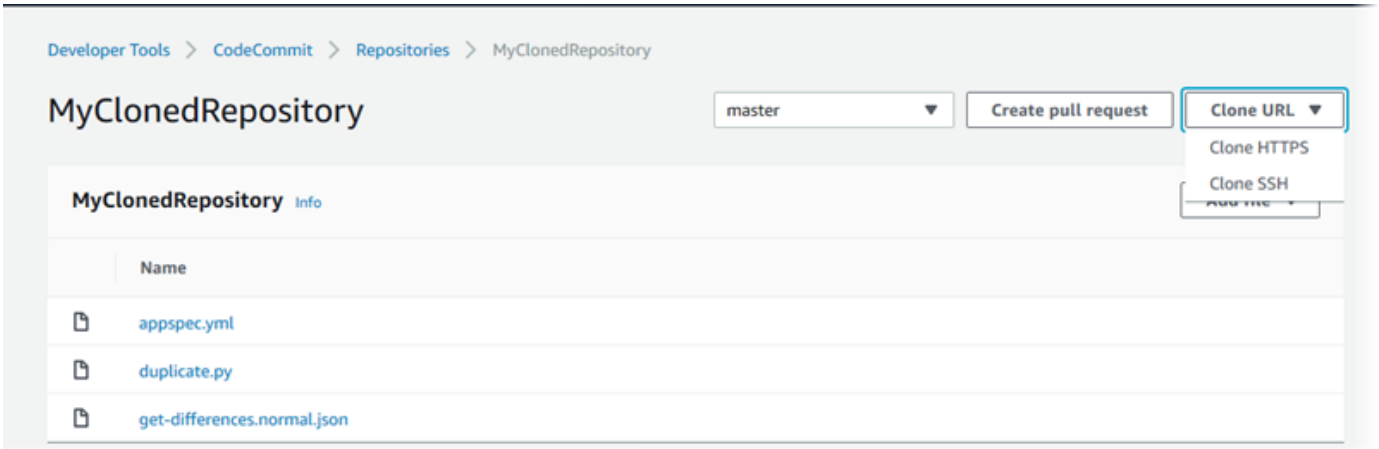
You can delete the *aws-codecommit-demo* folder and its contents after you have migrated the repository to CodeCommit. To create a local repo with all the correct references for working with the repository in CodeCommit, run the `git clone` command without the `--mirror` option:

```
git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository
```

Step 3: View files in CodeCommit

After you have pushed the contents of your directory, you can use the CodeCommit console to quickly view all of the files in that repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository (for example, *MyClonedRepository*).
3. View the files in the repository for the branches, the clone URLs, the settings, and more.



Step 4: Share the CodeCommit repository

When you create a repository in CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active no matter which protocol you

recommend to your users. Before you can share your repository with others, you must create IAM policies that allow other users access to your repository. Provide those access instructions to your users.

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page,, choose **Import managed policy**.
4. On the **Import managed policies** page, in **Filter policies**, enter **`AWSCodeCommitPowerUser`**. Choose the button next to the policy name and then choose **Import**.
5. On the **Create policy** page, choose **JSON**. Replace the "*" portion of the `Resource` line for CodeCommit actions with the Amazon Resource Name (ARN) of the CodeCommit repository, as shown here:

```
"Resource": [
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo"
 ]
```

> **Tip**
>
> To find the ARN for the CodeCommit repository, go to the CodeCommit console, choose the repository name from the list, and then choose **Settings**. For more information, see View repository details.

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown here:

```
"Resource": [
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
 "arn:aws:codecommit:us-east-2:111111111111:MyOtherDemoRepo"
 ]
```

When you are finished editing, choose **Review policy**.

6. On the **Review Policy** page, in **Name**, enter a new name for the policy (for example, *AWSCodeCommitPowerUser-MyDemoRepo*). Optionally provide a description for this policy.
7. Choose **Create Policy**.

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step. Attach any other policies required for access, such as IAMUserSSHKeys or IAMSelfManageServiceSpecificCredentials.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Groups**, and then choose **Create New Group**.
3. On the **Set Group Name** page, in **Group Name**, enter a name for the group (for example, *MyDemoRepoGroup*), and then choose **Next Step**. Consider including the repository name as part of the group name.

   > **Note**
   >
   > This name must be unique across an Amazon Web Services account.

4. Select the box next to the customer managed policy you created in the previous section (for example, **AWSCodeCommitPowerUser-MyDemoRepo**).
5. On the **Review** page, choose **Create Group**. IAM creates this group with the specified policies already attached. The group appears in the list of groups associated with your Amazon Web Services account.
6. Choose your group from the list.
7. On the group summary page, choose the **Users** tab, and then choose **Add Users to Group**. On the list that shows all users associated with your Amazon Web Services account, select the boxes next to the users to whom you want to allow access to the CodeCommit repository, and then choose **Add Users**.

   > **Tip**
   >
   > You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

After you have created an IAM user to access CodeCommit using the policy group and policies you configured, send that user the information required to connect to the repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose the repository you want to share.
4. In **Clone URL**, choose the protocol that you want your users to use. This copies the clone URL for the connection protocol.
5. Send your users the clone URL along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, HTTPS).

Migrate local or unversioned content to AWS CodeCommit

The procedures in this topic show you how to migrate an existing project or local content on your computer to a CodeCommit repository. As part of this process, you:

- Complete the initial setup required for CodeCommit.
- Create a CodeCommit repository.
- Place a local folder under Git version control and push the contents of that folder to the CodeCommit repository.
- View files in the CodeCommit repository.
- Share the CodeCommit repository with your team.

Topics

- Step 0: Setup required for access to CodeCommit
- Step 1: Create a CodeCommit repository
- Step 2: Migrate local content to the CodeCommit repository
- Step 3: View files in CodeCommit
- Step 4: Share the CodeCommit repository

Step 0: Setup required for access to CodeCommit

Before you can migrate local content to CodeCommit, you must create and configure an IAM user for CodeCommit and configure your local computer for access. You should also install the AWS CLI to manage CodeCommit. Although you can perform most CodeCommit tasks without it, the AWS CLI offers flexibility when working with Git.

If you are already set up for CodeCommit, you can skip ahead to Step 1: Create a CodeCommit repository.

To create and configure an IAM user for accessing CodeCommit

1. Create an Amazon Web Services account by going to http://aws.amazon.com and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your Amazon Web Services account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your Amazon Web Services account.

   > **Note**
   >
   > CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by CodeCommit. For more information, see AWS KMS and encryption.

3. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
4. In the IAM console, in the navigation pane, choose **Users**, and then choose the IAM user you want to configure for CodeCommit access.
5. On the **Permissions** tab, choose **Add Permissions**.
6. In **Grant permissions**, choose **Attach existing policies directly**.
7. From the list of policies, select **AWSCodeCommitPowerUser** or another managed policy for CodeCommit access. For more information, see AWS managed policies for CodeCommit.

   After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies to attach to the IAM user. If the list is correct, choose **Add permissions**.

For more information about CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a repository and Authentication and access control for AWS CodeCommit.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

   For more information, see Getting Set Up with the AWS Command Line Interface.

   > **Note**
   >
   > CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

   ```
   aws codecommit help
   ```

   This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI
- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- ap-southeast-3
- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2

- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1
- ap-northeast-3
- af-south-1
- il-central-1

  For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

Next, you must install Git.

- **For Linux, macOS, or Unix**:

  To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

  To install Git, we recommend websites such as Git Downloads.

> **Note**
>
> Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

- **For Windows:**

  To work with files, commits, and other information in CodeCommit repositories, you must install Git on your local machine. CodeCommit supports Git versions 1.7.9 and later. Git version 2.28 supports configuring the branch name for initial commits. We recommend using a recent version of Git.

  To install Git, we recommend websites such as Git for Windows. If you use this link to install Git, you can accept all of the installation default settings except for the following:

  - When prompted during the **Adjusting your PATH environment** step, choose the option to use Git from the command line.
  - (Optional) If you intend to use HTTPS with the credential helper that is included in the AWS CLI instead of configuring Git credentials for CodeCommit, on the **Configuring extra options** page, make sure the **Enable Git Credential Manager** option is cleared. The Git Credential Manager is only compatible with CodeCommit if IAM users configure Git credentials. For more information, see For HTTPS users using Git credentials and Git for Windows: I installed Git for Windows, but I am denied access to my repository (403).

  > **Note**
  >
  > Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with CodeCommit. If you encounter issues with a specific version of Git and CodeCommit, review the information in Troubleshooting.

CodeCommit supports both HTTPS and SSH authentication. To complete setup, you must configure Git credentials for CodeCommit (HTTPS, recommended for most users), an SSH key pair (SSH) to use when accessing CodeCommit, **git-remote-codecommit** (recommended for users who use federated access), or the credential helper included in the AWS CLI.

- For Git credentials on all supported operating systems, see Step 3: Create Git credentials for HTTPS connections to CodeCommit.
- For SSH on Linux, macOS, or Unix, see SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit.
- For SSH on Windows, see Step 3: Set up the public and private keys for Git and CodeCommit .
- For **git-remote-codecommit**, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.
- For the credential helper on Linux, macOS, or Unix, see Set Up the Credential Helper (Linux, macOS, or Unix).
- For the credential helper on Windows, see Set Up the Credential Helper (Windows).

Step 1: Create a CodeCommit repository

In this section, you use the CodeCommit console to create the CodeCommit repository you use for the rest of this tutorial. To use the AWS CLI to create the repository, see Create a repository (AWS CLI).

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where you want to create the repository. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose **Create repository**.
4. On the **Create repository** page, in **Repository name**, enter a name for the repository.

   > **Note**
   >
   > Repository names are case sensitive. The name must be unique in the AWS Region for your Amazon Web Services account.

5. (Optional) In **Description**, enter a description for the repository. This can help you and other users identify the purpose of the repository.

   > **Note**
   >
   > The description field displays Markdown in the console and accepts all HTML characters and valid Unicode characters. If you are an application developer who is

using the `GetRepository` or `BatchGetRepositories` APIs and you plan to display the repository description field in a web browser, see the CodeCommit API Reference.

6. (Optional) Choose **Add tag** to add one or more repository tags (a custom attribute label that helps you organize and manage your AWS resources) to your repository. For more information, see Tagging repositories in AWS CodeCommit.
7. (Optional) Expand **Additional configuration** to specify whether to use the default AWS managed key or your own customer managed key for encrypting and decrypting data in this repository. If you choose to use your own customer managed key, you must ensure that it is available in the AWS Region where you are creating the repository, and that the key is active. For more information, see AWS Key Management Service and encryption for AWS CodeCommit repositories.
8. (Optional) Select **Enable Amazon CodeGuru Reviewer for Java and Python** if this repository contains Java or Python code, and you want CodeGuru Reviewer to analyze it. CodeGuru Reviewer uses multiple machine learning models to find code defects and to suggest improvements and fixes in pull requests. For more information, see the *Amazon CodeGuru Reviewer User Guide*.
9. Choose **Create**.

After it is created, the repository appears in the **Repositories** list. In the URL column, choose the copy icon, and then choose the protocol (HTTPS or SSH) to be used to connect to CodeCommit. Copy the URL.

For example, if you named your repository *MyFirstRepo* and you are using HTTPS, the URL would look like the following:

```
https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyFirstRepo
```

You need this URL later in Step 2: Migrate local content to the CodeCommit repository.

Step 2: Migrate local content to the CodeCommit repository

Now that you have a CodeCommit repository, you can choose a directory on your local computer to convert into a local Git repository. The **git init** command can be used to either convert existing, unversioned content to a Git repository or, if you do not yet have files or content, to initialize a new, empty repository.

1. From the terminal or command line on your local computer, change directories to the directory you want to use as the source for your repository.
2. Run the following command to configure Git to use a default branch named **`main`**:

   ```
   git config --local init.defaultBranch main
   ```

   You can also run this command to set your default branch name to **`main`** for all newly-created repositories:

   ```
   git config --global init.defaultBranch main
   ```
3. Run the **git init** command to initialize Git version control in the directory. This creates a .git subdirectory in the root of the directory that enables version control tracking. The .git folder also contains all of the required metadata for the repository.

   ```
   git init
   ```
4. Check the status of the initialized directory by running the following command:

   ```
   git status
   ```

   Add the files you want to add to version control. In this tutorial, you run the `git add` command with the `.` specifier to add all of the files in this directory. For other options, consult your Git documentation.

   ```
   git add .
   ```
5. Create a commit for the added files with a commit message.

   ```
   git commit -m "Initial commit"
   ```
6. Run the **git push** command, specifying the URL and name of the destination CodeCommit repository and the `--all` option. (This is the URL you copied in Step 1: Create a CodeCommit repository.)

   For example, if you named your repository *`MyFirstRepo`* and you are set up to use HTTPS, you would run the following command:

```
git push https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyFirstRepo --all
```

Step 3: View files in CodeCommit

After you have pushed the contents of your directory, you can use the CodeCommit console to quickly view all of the files in the repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In **Repositories**, choose the name of the repository (for example, *MyFirstRepository*) from the list.
3. View the files in the repository for the branches, clone URLs, settings, and more.

Step 4: Share the CodeCommit repository

When you create a repository in CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active no matter which protocol you recommend to your users. Before you can share your repository with others, you must create IAM policies that allow other users access to your repository. Provide those access instructions to your users.

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page,, choose **Import managed policy**.
4. On the **Import managed policies** page, in **Filter policies**, enter **`AWSCodeCommitPowerUser`**. Choose the button next to the policy name and then choose **Import**.
5. On the **Create policy** page, choose **JSON**. Replace the "*" portion of the `Resource` line for CodeCommit actions with the Amazon Resource Name (ARN) of the CodeCommit repository, as shown here:

```
"Resource": [
```

```
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo"
 ]
```

> **Tip**
>
> To find the ARN for the CodeCommit repository, go to the CodeCommit console, choose the repository name from the list, and then choose **Settings**. For more information, see View repository details.

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown here:

```
"Resource": [
 "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
 "arn:aws:codecommit:us-east-2:111111111111:MyOtherDemoRepo"
 ]
```

When you are finished editing, choose **Review policy**.

6. On the **Review Policy** page, in **Name**, enter a new name for the policy (for example, *AWSCodeCommitPowerUser-MyDemoRepo*). Optionally provide a description for this policy.
7. Choose **Create Policy**.

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step. Attach any other policies required for access, such as `IAMSelfManageServiceSpecificCredentials` or `IAMUserSSHKeys`.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the **Dashboard** navigation area, choose **Groups**, and then choose **Create New Group**.
3. On the **Set Group Name** page, in **Group Name**, enter a name for the group (for example, *MyDemoRepoGroup*), and then choose **Next Step**. Consider including the repository name as part of the group name.

> **Note**
>
> This name must be unique across an Amazon Web Services account.

4. Select the box next to the customer managed policy you created in the previous section (for example, **AWSCodeCommitPowerUser-MyDemoRepo**).
5. On the **Review** page, choose **Create Group**. IAM creates this group with the specified policies already attached. The group appears in the list of groups associated with your Amazon Web Services account.
6. Choose your group from the list.
7. On the group summary page, choose the **Users** tab, and then choose **Add Users to Group**. On the list that shows all users associated with your Amazon Web Services account, select the boxes next to the users to whom you want to allow access to the CodeCommit repository, and then choose **Add Users**.

> **Tip**
>
> You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

After you have created an IAM user to be used to access CodeCommit using the policy group and policies you configured, send that user the information required to connect to the repository.

1. Open the CodeCommit console at https://console.aws.amazon.com/codesuite/codecommit/home.
2. In the Region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see Regions and Git connection endpoints.
3. On the **Repositories** page, choose the repository you want to share.
4. In **Clone URL**, choose the protocol that you want your users to use. This copies the clone URL for the connection protocol.
5. Send your users the clone URL along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, HTTPS).

Migrate a repository incrementally

When migrating to AWS CodeCommit, consider pushing your repository in increments or chunks to reduce the chances an intermittent network issue or degraded network performance causes the entire push to fail. By using incremental pushes with a script like the one included here, you can restart the migration and push only those commits that did not succeed on the earlier attempt.

The procedures in this topic show you how to create and run a script that migrates your repository in increments and repushes only those increments that did not succeed until the migration is complete.

These instructions are written with the assumption that you have already completed the steps in Setting up and Create a repository.

Topics

- Step 0: Determine whether to migrate incrementally
- Step 1: Install prerequisites and add the CodeCommit repository as a remote
- Step 2: Create the script to use for migrating incrementally
- Step 3: Run the script and migrate incrementally to CodeCommit
- Appendix: Sample script incremental-repo-migration.py

Step 0: Determine whether to migrate incrementally

There are several factors to consider to determine the overall size of your repository and whether to migrate incrementally. The most obvious is the overall size of the artifacts in the repository. Factors such as the accumulated history of the repository can also contribute to size. A repository with years of history and branches can be very large, even though the individual assets are not. There are a number of strategies you can pursue to make migrating these repositories simpler and more efficient. For example, you can use a shallow clone strategy when cloning a repository with a long history of development, or you can turn off delta compression for large binary files. You can research options by consulting your Git documentation, or you can choose to set up and configure incremental pushes for migrating your repository using the sample script included in this topic, `incremental-repo-migration.py`.

You might want to configure incremental pushes if one or more of the following conditions is true:

- The repository you want to migrate has more than five years of history.

- Your internet connection is subject to intermittent outages, dropped packets, slow response, or other interruptions in service.
- The overall size of the repository is larger than 2 GB and you intend to migrate the entire repository.
- The repository contains large artifacts or binaries that do not compress well, such as large image files with more than five tracked versions.
- You have previously attempted a migration to CodeCommit and received an "Internal Service Error" message.

Even if none of the above conditions are true, you can still choose to push incrementally.

Step 1: Install prerequisites and add the CodeCommit repository as a remote

You can create your own custom script, which has its own prerequisites. If you use the sample included in this topic, you must:

- Install its prerequisites.
- Clone the repository to your local computer.
- Add the CodeCommit repository as a remote for the repository you want to migrate.

Set up to run incremental-repo-migration.py

1. On your local computer, install Python 2.6 or later. For more information and the latest versions, see the Python website.
2. On the same computer, install GitPython, which is a Python library used to interact with Git repositories. For more information, see the GitPython documentation.
3. Use the **git clone --mirror** command to clone the repository you want to migrate to your local computer. From the terminal (Linux, macOS, or Unix) or the command prompt (Windows), use the **git clone --mirror** command to create a local repo for the repository, including the directory where you want to create the local repo. For example, to clone a Git repository named *MyMigrationRepo* with a URL of *https://example.com/my-repo/* to a directory named *my-repo*:

   ```
   git clone --mirror https://example.com/my-repo/MyMigrationRepo.git my-repo
   ```

You should see output similar to the following, which indicates the repository has been cloned into a bare local repo named my-repo:

```
Cloning into bare repository 'my-repo'...
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 20 (delta 5), reused 15 (delta 3)
Unpacking objects: 100% (20/20), done.
Checking connectivity... done.
```

4. Change directories to the local repo for the repository you just cloned (for example, *my-repo*). From that directory, use the **git remote add** ***DefaultRemoteName RemoteRepositoryURL*** command to add the CodeCommit repository as a remote repository for the local repo.

> **Note**
>
> When pushing large repositories, consider using SSH instead of HTTPS. When you push a large change, a large number of changes, or a large repository, long-running HTTPS connections are often terminated prematurely due to networking issues or firewall settings. For more information about setting up CodeCommit for SSH, see For SSH connections on Linux, macOS, or Unix or For SSH connections on Windows.

For example, use the following command to add the SSH endpoint for a CodeCommit repository named MyDestinationRepo as a remote repository for the remote named `codecommit`:

```
git remote add codecommit ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDestinationRepo
```

> **Tip**
>
> Because this is a clone, the default remote name (`origin`) is already in use. You must use another remote name. Although the example uses `codecommit`, you can use any name you want. Use the **git remote show** command to review the list of remotes set for your local repo.

5. Use the **git remote -v** command to display the fetch and push settings for your local repo and confirm they are set correctly. For example:

```
codecommit  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDestinationRepo
 (fetch)
codecommit  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDestinationRepo
 (push)
```

> **Tip**
>
> If you still see fetch and push entries for a different remote repository (for example, entries for origin), use the **git remote set-url --delete** command to remove them.

Step 2: Create the script to use for migrating incrementally

These steps are written with the assumption that you are using the `incremental-repo-migration.py` sample script.

1. Open a text editor and paste the contents of the sample script into an empty document.
2. Save the document in a documents directory (not the working directory of your local repo) and name it `incremental-repo-migration.py`. Make sure the directory you choose is one configured in your local environment or path variables, so you can run the Python script from a command line or terminal.

Step 3: Run the script and migrate incrementally to CodeCommit

Now that you have created your `incremental-repo-migration.py` script, you can use it to incrementally migrate a local repo to a CodeCommit repository. By default, the script pushes commits in batches of 1,000 commits and attempts to use the Git settings for the directory from which it is run as the settings for the local repo and remote repository. You can use the options included in `incremental-repo-migration.py` to configure other settings, if necessary.

1. From the terminal or command prompt, change directories to the local repo you want to migrate.
2. From that directory, run the following command:

```
python incremental-repo-migration.py
```

3. The script runs and shows progress at the terminal or command prompt. Some large repositories are slow to show progress. The script stops if a single push fails three times. You can then rerun the script, and it starts from the batch that failed. You can rerun the script until all pushes succeed and the migration is complete.

Tip

You can run `incremental-repo-migration.py` from any directory as long as you use the `-l` and `-r` options to specify the local and remote settings to use. For example, to use the script from any directory to migrate a local repo located at /tmp/*my-repo* to a remote nicknamed *codecommit*:

```
python incremental-repo-migration.py -l "/tmp/my-repo" -r "codecommit"
```

You might also want to use the `-b` option to change the default batch size used when pushing incrementally. For example, if you are regularly pushing a repository with very large binary files that change often and are working from a location that has restricted network bandwidth, you might want to use the `-b` option to change the batch size to 500 instead of 1,000. For example:

```
python incremental-repo-migration.py -b 500
```

This pushes the local repo incrementally in batches of 500 commits. If you decide to change the batch size again when you migrate the repository (for example, if you decide to decrease the batch size after an unsuccessful attempt), remember to use the `-c` option to remove the batch tags before resetting the batch size with `-b`:

```
python incremental-repo-migration.py -c
python incremental-repo-migration.py -b 250
```

> **Important**
>
> Do not use the `-c` option if you want to rerun the script after a failure. The `-c` option removes the tags used to batch the commits. Use the `-c` option only if you want to change the batch size and start again, or if you decide you no longer want to use the script.

Appendix: Sample script `incremental-repo-migration.py`

For your convenience, we have developed a sample Python script, `incremental-repo-migration.py`, for pushing a repository incrementally. This script is an open source code sample and provided as-is.

```
# Copyright 2015 Amazon.com, Inc. or its affiliates. All Rights Reserved. Licensed
 under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License. A copy of the
 License is located at
#    http://aws.amazon.com/asl/
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 KIND, express or implied. See the License for
# the specific language governing permissions and limitations under the License.

#!/usr/bin/env python

import os
import sys
from optparse import OptionParser
from git import Repo, TagReference, RemoteProgress, GitCommandError


class PushProgressPrinter(RemoteProgress):
    def update(self, op_code, cur_count, max_count=None, message=""):
        op_id = op_code & self.OP_MASK
        stage_id = op_code & self.STAGE_MASK
        if op_id == self.WRITING and stage_id == self.BEGIN:
            print("\tObjects: %d" % max_count)


class RepositoryMigration:
    MAX_COMMITS_TOLERANCE_PERCENT = 0.05
    PUSH_RETRY_LIMIT = 3
    MIGRATION_TAG_PREFIX = "codecommit_migration_"
```

```
    def migrate_repository_in_parts(
        self, repo_dir, remote_name, commit_batch_size, clean
    ):
        self.next_tag_number = 0
        self.migration_tags = []
        self.walked_commits = set()
        self.local_repo = Repo(repo_dir)
        self.remote_name = remote_name
        self.max_commits_per_push = commit_batch_size
        self.max_commits_tolerance = (
            self.max_commits_per_push * self.MAX_COMMITS_TOLERANCE_PERCENT
        )

        try:
            self.remote_repo = self.local_repo.remote(remote_name)
            self.get_remote_migration_tags()
        except (ValueError, GitCommandError):
            print(
                "Could not contact the remote repository. The most common reasons for
this error are that the name of the remote repository is incorrect, or that you do not
have permissions to interact with that remote repository."
            )
            sys.exit(1)

        if clean:
            self.clean_up(clean_up_remote=True)
            return

        self.clean_up()

        print("Analyzing repository")
        head_commit = self.local_repo.head.commit
        sys.setrecursionlimit(max(sys.getrecursionlimit(), head_commit.count()))

        # tag commits on default branch
        leftover_commits = self.migrate_commit(head_commit)
        self.tag_commits([commit for (commit, commit_count) in leftover_commits])

        # tag commits on each branch
        for branch in self.local_repo.heads:
            leftover_commits = self.migrate_commit(branch.commit)
            self.tag_commits([commit for (commit, commit_count) in leftover_commits])
```

```
        # push the tags
        self.push_migration_tags()

        # push all branch references
        for branch in self.local_repo.heads:
            print("Pushing branch %s" % branch.name)
            self.do_push_with_retries(ref=branch.name)

        # push all tags
        print("Pushing tags")
        self.do_push_with_retries(push_tags=True)

        self.get_remote_migration_tags()
        self.clean_up(clean_up_remote=True)

        print("Migration to CodeCommit was successful")

    def migrate_commit(self, commit):
        if commit in self.walked_commits:
            return []

        pending_ancestor_pushes = []
        commit_count = 1

        if len(commit.parents) > 1:
            # This is a merge commit
            # Ensure that all parents are pushed first
            for parent_commit in commit.parents:
                pending_ancestor_pushes.extend(self.migrate_commit(parent_commit))
        elif len(commit.parents) == 1:
            # Split linear history into individual pushes
            next_ancestor, commits_to_next_ancestor = self.find_next_ancestor_for_push(
                commit.parents[0]
            )
            commit_count += commits_to_next_ancestor
            pending_ancestor_pushes.extend(self.migrate_commit(next_ancestor))

        self.walked_commits.add(commit)

        return self.stage_push(commit, commit_count, pending_ancestor_pushes)

    def find_next_ancestor_for_push(self, commit):
        commit_count = 0
```

```
        # Traverse linear history until we reach our commit limit, a merge commit, or
an initial commit
        while (
            len(commit.parents) == 1
            and commit_count < self.max_commits_per_push
            and commit not in self.walked_commits
        ):
            commit_count += 1
            self.walked_commits.add(commit)
            commit = commit.parents[0]

        return commit, commit_count

    def stage_push(self, commit, commit_count, pending_ancestor_pushes):
        # Determine whether we can roll up pending ancestor pushes into this push
        combined_commit_count = commit_count + sum(
            ancestor_commit_count
            for (ancestor, ancestor_commit_count) in pending_ancestor_pushes
        )

        if combined_commit_count < self.max_commits_per_push:
            # don't push anything, roll up all pending ancestor pushes into this
pending push
            return [(commit, combined_commit_count)]

        if combined_commit_count <= (
            self.max_commits_per_push + self.max_commits_tolerance
        ):
            # roll up everything into this commit and push
            self.tag_commits([commit])
            return []

        if commit_count >= self.max_commits_per_push:
            # need to push each pending ancestor and this commit
            self.tag_commits(
                [
                    ancestor
                    for (ancestor, ancestor_commit_count) in pending_ancestor_pushes
                ]
            )
            self.tag_commits([commit])
            return []

        # push each pending ancestor, but roll up this commit
```

```
        self.tag_commits(
            [ancestor for (ancestor, ancestor_commit_count) in pending_ancestor_pushes]
        )
        return [(commit, commit_count)]

    def tag_commits(self, commits):
        for commit in commits:
            self.next_tag_number += 1
            tag_name = self.MIGRATION_TAG_PREFIX + str(self.next_tag_number)

            if tag_name not in self.remote_migration_tags:
                tag = self.local_repo.create_tag(tag_name, ref=commit)
                self.migration_tags.append(tag)
            elif self.remote_migration_tags[tag_name] != str(commit):
                print(
                    "Migration tags on the remote do not match the local tags. Most
likely your batch size has changed since the last time you ran this script. Please run
this script with the --clean option, and try again."
                )
                sys.exit(1)

    def push_migration_tags(self):
        print("Will attempt to push %d tags" % len(self.migration_tags))
        self.migration_tags.sort(
            key=lambda tag: int(tag.name.replace(self.MIGRATION_TAG_PREFIX, ""))
        )
        for tag in self.migration_tags:
            print(
                "Pushing tag %s (out of %d tags), commit %s"
                % (tag.name, self.next_tag_number, str(tag.commit))
            )
            self.do_push_with_retries(ref=tag.name)

    def do_push_with_retries(self, ref=None, push_tags=False):
        for i in range(0, self.PUSH_RETRY_LIMIT):
            if i == 0:
                progress_printer = PushProgressPrinter()
            else:
                progress_printer = None

            try:
                if push_tags:
                    infos = self.remote_repo.push(tags=True, progress=progress_printer)
                elif ref is not None:
```

```
                    infos = self.remote_repo.push(
                        refspec=ref, progress=progress_printer
                    )
                else:
                    infos = self.remote_repo.push(progress=progress_printer)

                success = True
                if len(infos) == 0:
                    success = False
                else:
                    for info in infos:
                        if (
                            info.flags & info.UP_TO_DATE
                            or info.flags & info.NEW_TAG
                            or info.flags & info.NEW_HEAD
                        ):
                            continue
                        success = False
                        print(info.summary)

                if success:
                    return
            except GitCommandError as err:
                print(err)

        if push_tags:
            print("Pushing all tags failed after %d attempts" %
(self.PUSH_RETRY_LIMIT))
        elif ref is not None:
            print("Pushing %s failed after %d attempts" % (ref, self.PUSH_RETRY_LIMIT))
            print(
                "For more information about the cause of this error, run the following
command from the local repo: 'git push %s %s'"
                % (self.remote_name, ref)
            )
        else:
            print(
                "Pushing all branches failed after %d attempts"
                % (self.PUSH_RETRY_LIMIT)
            )
        sys.exit(1)

    def get_remote_migration_tags(self):
        remote_tags_output = self.local_repo.git.ls_remote(
```

```
            self.remote_name, tags=True
        ).split("\n")
        self.remote_migration_tags = dict(
            (tag.split()[1].replace("refs/tags/", ""), tag.split()[0])
            for tag in remote_tags_output
            if self.MIGRATION_TAG_PREFIX in tag
        )

    def clean_up(self, clean_up_remote=False):
        tags = [
            tag
            for tag in self.local_repo.tags
            if tag.name.startswith(self.MIGRATION_TAG_PREFIX)
        ]

        # delete the local tags
        TagReference.delete(self.local_repo, *tags)

        # delete the remote tags
        if clean_up_remote:
            tags_to_delete = [":" + tag_name for tag_name in
 self.remote_migration_tags]
            self.remote_repo.push(refspec=tags_to_delete)


parser = OptionParser()
parser.add_option(
    "-l",
    "--local",
    action="store",
    dest="localrepo",
    default=os.getcwd(),
    help="The path to the local repo. If this option is not specified, the script will
 attempt to use current directory by default. If it is not a local git repo, the script
 will fail.",
)
parser.add_option(
    "-r",
    "--remote",
    action="store",
    dest="remoterepo",
    default="codecommit",
```

```
    help="The name of the remote repository to be used as the push or migration
 destination. The remote must already be set in the local repo ('git remote add ...').
 If this option is not specified, the script will use 'codecommit' by default.",
)
parser.add_option(
    "-b",
    "--batch",
    action="store",
    dest="batchsize",
    default="1000",
    help="Specifies the commit batch size for pushes. If not explicitly set, the
 default is 1,000 commits.",
)
parser.add_option(
    "-c",
    "--clean",
    action="store_true",
    dest="clean",
    default=False,
    help="Remove the temporary tags created by migration from both the local repo
 and the remote repository. This option will not do any migration work, just cleanup.
 Cleanup is done automatically at the end of a successful migration, but not after a
 failure so that when you re-run the script, the tags from the prior run can be used to
 identify commit batches that were not pushed successfully.",
)

(options, args) = parser.parse_args()

migration = RepositoryMigration()
migration.migrate_repository_in_parts(
    options.localrepo, options.remoterepo, int(options.batchsize), options.clean
)
```

Security in AWS CodeCommit

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the .
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using CodeCommit. The following topics show you how to configure CodeCommit to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CodeCommit resources.

Topics

- Data protection in AWS CodeCommit
- Identity and Access Management for AWS CodeCommit
- Resilience in AWS CodeCommit
- Infrastructure security in AWS CodeCommit

Data protection in AWS CodeCommit

As a managed service, is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

CodeCommit repositories are automatically encrypted at rest. No customer action is required. CodeCommit also encrypts repository data in transit. You can use either the HTTPS protocol, the SSH protocol, or both with CodeCommit repositories. For more information, see Setting up for AWS CodeCommit. You can also configure cross-account access to CodeCommit repositories.

Topics

- AWS Key Management Service and encryption for AWS CodeCommit repositories
- Connecting to AWS CodeCommit repositories with rotating credentials

AWS Key Management Service and encryption for AWS CodeCommit repositories

Data in CodeCommit repositories is encrypted in transit and at rest. When data is pushed into a CodeCommit repository (for example, by calling **git push**), CodeCommit encrypts the received data as it is stored in the repository. When data is pulled from a CodeCommit repository (for example, by calling **git pull**), CodeCommit decrypts the data and then sends it to the caller. This assumes the IAM user associated with the push or pull request has been authenticated by AWS. Data sent or received is transmitted using the HTTPS or SSH encrypted network protocols.

You can use either an AWS managed key or a customer managed key for encrypting and decrypting the data in your repository. For more information about the differences between customer managed keys and AWS managed keys, see Customer managed keys and AWS managed keys. If you don't specify a customer managed key, CodeCommit will use an AWS managed key for encrypting and decrypting the data in your repository. This AWS managed key is created automatically for you in your AWS account. The first time you create a CodeCommit repository in a new AWS Region in your Amazon Web Services account, if you don't specify a customer managed key, CodeCommit creates an AWS managed key (the `aws/codecommit` key) in that same AWS Region in AWS Key Management Service (AWS KMS). This `aws/codecommit` key is used only by CodeCommit. It is stored in your Amazon Web Services account. Depending on what you specify, CodeCommit either uses the customer managed key or the AWS managed key to encrypt and decrypt the data in the repository.

Important

CodeCommit performs the following AWS KMS actions against the AWS KMS key used to encrypt and decrypt data in a repository. If you're using an AWS managed key, a user does not need explicit permissions for these actions, but the user must not have any attached policies that deny these actions for the `aws/codecommit` key. If you are using a customer managed key that has an AWS account ID set as a policy principal for that key, these permissions must be explicitly set to `allow`. Specifically, when you create your first repository, and if you update keys for your repository, you must not have any of the following permissions set to `deny` if you are using a AWS managed key, and must be set to `allow` if you are using a customer managed key with a policy principal:

- `"kms:Encrypt"`
- `"kms:Decrypt"`
- `"kms:ReEncrypt"` (depending on context, this could require `kms:ReEncryptFrom`, `kms:ReEncryptTo`, or `kms:ReEncrypt*` not set to deny)
- `"kms:GenerateDataKey"`
- `"kms:GenerateDataKeyWithoutPlaintext"`
- `"kms:DescribeKey"`

If you want to use your own customer managed key, the key must be available in the AWS Region where the repository exists. CodeCommit supports using both single and multi-Region customer managed keys. While all key material origin types are supported, we recommend using the default **KMS** option. Customers using the **External key store** option might experience delays from their store provider. In addition. CodeCommit has the following requirements for customer managed keys:

- CodeCommit only supports using symmetric keys.
- The key usage type must be set to **Encrypt and decrypt**.

For more information on creating customer managed keys, see Concepts and Creating keys.

To see information about the AWS managed key generated by CodeCommit, do the following:

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at https://eusc-de-east-1.console.amazonaws-eusc.eu/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the service navigation pane, choose **AWS managed keys**. Make sure that you are signed in to the AWS Region where you want to review keys.
4. In the list of encryption keys, choose the AWS managed key with the alias **aws/codecommit**. Basic information about the AWS owned key is displayed.

You cannot change or delete this AWS managed key.

How encryption algorithms are used to encrypt repository data

CodeCommit uses two different approaches for encrypting data. Individual Git objects under 6 MB are encrypted using AES-GCM-256, which provides data integrity validation. Objects between 6 MB and the maximum 2 GB for a single blob are encrypted using AES-CBC-256. CodeCommit always validates the encryption context.

Encryption context

Each service integrated with AWS KMS specifies an encryption context for both the encryption and decryption operations. The encryption context is additional authenticated information AWS KMS uses to check for data integrity. When specified for the encryption operation, it must also be specified in the decryption operation. Otherwise, decryption fails. CodeCommit uses the CodeCommit repository ID for the encryption context. You can use the **get-repository** command or the CodeCommit console to find the repository ID. Search for the CodeCommit repository ID in AWS CloudTrail logs to understand which encryption operations were taken on which key in AWS KMS to encrypt or decrypt data in the CodeCommit repository.

For more information about AWS KMS, see the AWS Key Management Service Developer Guide.

Connecting to AWS CodeCommit repositories with rotating credentials

You can give users access to your AWS CodeCommit repositories without configuring IAM users for them or using an access key and secret key. To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider

(federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*. You can also configure role-based access for IAM users to access CodeCommit repositories in separate Amazon Web Services accounts (a technique known as *cross-account access*). For a walkthrough of configuring cross-account access to a repository, see Configure cross-account access to an AWS CodeCommit repository using roles.

You can configure access for users who want or must authenticate through methods such as:

- Security Assertion Markup Language (SAML)
- Multi-factor authentication (MFA)
- Federation
- Login with Amazon
- Amazon Cognito
- Facebook
- Google
- OpenID Connect (OIDC)-compatible identity provider

> **Note**
>
> The following information applies only to the use of **git-remote-codecommit** or the AWS CLI credential helper to connect to CodeCommit repositories. Because the recommended approach for temporary or federated access to CodeCommit is to set up **git-remote-codecommit**, this topic provides examples using that utility. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit. You cannot use SSH or Git credentials and HTTPS to connect to CodeCommit repositories with rotating or temporary access credentials.

You do not need to complete these steps if all of the following requirements are true:

- You are signed in to an Amazon EC2 instance.
- You are using Git and HTTPS with the AWS CLI credential helper to connect from the Amazon EC2 instance to CodeCommit repositories.

- The Amazon EC2 instance has an attached IAM instance profile that contains the access permissions described in For HTTPS connections on Linux, macOS, or Unix with the AWS CLI credential helper or For HTTPS connections on Windows with the AWS CLI credential helper.
- You have installed and configured the Git credential helper on the Amazon EC2 instance, as described in For HTTPS connections on Linux, macOS, or Unix with the AWS CLI credential helper or For HTTPS connections on Windows with the AWS CLI credential helper.

Amazon EC2 instances that meet the preceding requirements are already set up to communicate temporary access credentials to CodeCommit on your behalf.

> **Note**
>
> You can configure and use **git-remote-codecommit** on Amazon EC2 instances.

To give users temporary access to your CodeCommit repositories, complete the following steps.

Step 1: Complete the prerequisites

Complete the setup steps to provide a user with access to your CodeCommit repositories using rotating credentials:

- For cross-account access, see Walkthrough: Delegating Access Across Amazon Web Services accounts Using IAM Roles and Configure cross-account access to an AWS CodeCommit repository using roles.
- For SAML and federation, see Using Your Organization's Authentication System to Grant Access to AWS Resources and About AWS STS SAML 2.0-based Federation.
- For MFA, see Using Multi-Factor Authentication (MFA) Devices with AWS and Creating Temporary Security Credentials to Enable Access for IAM Users.
- For Login with Amazon, Amazon Cognito, Facebook, Google, or any OIDC-compatible identity provider, see About AWS STS Web Identity Federation.

Use the information in Authentication and access control for AWS CodeCommit to specify the CodeCommit permissions you want to grant the user.

Step 2: Get role name or access credentials

If you want your users to access repositories by assuming a role, provide your users with the Amazon Resource Name (ARN) of that role. Otherwise, depending on the way you set up access, your user can get rotating credentials in one of the following ways:

- For cross-account access, call the AWS CLI assume-role command or call the AWS STS AssumeRole API.
- For SAML, call the AWS CLI assume-role-with-saml command or the AWS STS AssumeRoleWithSAML API.
- For federation, call the AWS CLI assume-role or get-federation-token commands or the AWS STS AssumeRole or GetFederationToken APIs.
- For MFA, call the AWS CLI get-session-token command or the AWS STS GetSessionToken API.
- For Login with Amazon, Amazon Cognito, Facebook, Google, or any OIDC-compatible identity provider, call the AWS CLI assume-role-with-web-identity command or the AWS STS AssumeRoleWithWebIdentity API.

Step 3: Install git-remote-codecommit and configure the AWS CLI

You must configure your local computer to use the access credentials by installing **git-remote-codecommit** and configuring a profile in the AWS CLI.

1. Follow the instructions in Setting up to set up the AWS CLI. Use the **aws configure** command to configure one or more profiles. Consider creating a named profile to use when you connect to CodeCommit repositories using rotating credentials.
2. You can associate the credentials with the user's AWS CLI named profile in one of the following ways.

   - If you are assuming a role to access CodeCommit, configure a named profile with the information required to assume that role. For example, if you want to assume a role named *CodeCommitAccess* in the Amazon Web Services account 111111111111, you can configure a default profile to use when working with other AWS resources and a named profile to use when assuming that role. The following commands create a named profile named *CodeAccess* that assumes a role named *CodeCommitAccess*. The user name *Maria_Garcia* is associated with the session and the default profile is set as the source of its AWS credentials:

```
aws configure set role_arn arn:aws:iam::111111111111:role/CodeCommitAccess --
profile CodeAccess
aws configure set source_profile default --profile CodeAccess
aws configure set role_session_name "Maria_Garcia" --profile CodeAccess
```

If you want to verify the changes, manually view or edit the `~/.aws/config` file (for Linux) or the `%UserProfile%.aws\config` file (for Windows) and review the information under the named profile. For example, your file might look similar to the following:

```
[default]
region = us-east-1
output = json

[profile CodeAccess]
source_profile = default
role_session_name = Maria_Garcia
role_arn = arn:aws:iam::111111111111:role/CodeCommitAccess
```

After you have configured your named profile, you can then clone CodeCommit repositories with the **git-remote-codecommit** utility using the named profile. For example, to clone a repository named *MyDemoRepo*:

```
git clone codecommit://CodeAccess@MyDemoRepo
```

- If you are using web identity federation and OpenID Connect (OIDC), configure a named profile that makes the AWS Security Token Service (AWS STS) `AssumeRoleWithWebIdentity` API call on your behalf to refresh temporary credentials. Use the **aws configure set** command or manually edit the `~/.aws/credentials` file (for Linux) or the `%UserProfile%.aws\credentials` file (for Windows) to add an AWS CLI named profile with the required setting values. For example, to create a profile that assumes the *CodeCommitAccess* role and uses a web identity token file ~/*my-credentials/my-token-file*:

  ```
  [CodeCommitWebIdentity]
  role_arn = arn:aws:iam::111111111111:role/CodeCommitAccess
  web_identity_token_file=~/my-credentials/my-token-file
  role_session_name = Maria_Garcia
  ```

For more information, see Configuring the AWS Command Line Interface and Using an IAM Role in the AWS CLI in the *AWS Command Line Interface User Guide*.

Step 4: Access the CodeCommit repositories

Assuming your user has followed the instructions in Connect to a repository to connect to the CodeCommit repositories, the user then uses the extended functionality provided by **git-remote-codecommit** and Git to call **git clone**, **git push**, and **git pull** to clone, push to, and pull from, the CodeCommit repositories to which they have access. For example, to clone a repository:

```
git clone codecommit://CodeAccess@MyDemoRepo
```

Git commit, push, and pull commands use regular Git syntax.

When the user uses the AWS CLI and specifies the AWS CLI named profile associated with the rotating access credentials, results scoped to that profile are returned.

Identity and Access Management for AWS CodeCommit

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CodeCommit resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- Authentication and access control for AWS CodeCommit
- How AWS CodeCommit works with IAM
- CodeCommit resource-based policies
- Authorization based on CodeCommit tags
- CodeCommit IAM roles

- AWS CodeCommit identity-based policy examples
- Troubleshooting AWS CodeCommit identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS CodeCommit identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS CodeCommit works with IAM)
- **IAM administrator** - write policies to manage access (see AWS CodeCommit identity-based policy examples)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more

information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between

managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

Authentication and access control for AWS CodeCommit

Access to AWS CodeCommit requires credentials. Those credentials must have permissions to access AWS resources, such as CodeCommit repositories, and your IAM user, which you use to manage your Git credentials or the SSH public key that you use for making Git connections. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and CodeCommit to help secure access to your resources:

- Authentication
- Access control

Authentication

Because CodeCommit repositories are Git-based and support the basic functionality of Git, including Git credentials, we recommend that you use an IAM user when working with CodeCommit. You can access CodeCommit with other identity types, but the other identity types are subject to limitations, as described below.

Identity types:

- **IAM user** – An IAM user is an identity within your Amazon Web Services account that has specific custom permissions. For example, an IAM user can have permissions to create and manage Git credentials for accessing CodeCommit repositories. **This is the recommended user type for working with CodeCommit.** You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  You can generate Git credentials or associate SSH public keys with your IAM user, or you can install and configure **git-remote-codecommit**. These are the easiest ways to set up Git to work with your CodeCommit repositories. With Git credentials, you generate a static user name and password in IAM. You then use these credentials for HTTPS connections with Git and any third-party tool that supports Git user name and password authentication. With SSH connections, you create public and private key files on your local machine that Git and CodeCommit use for SSH

authentication. You associate the public key with your IAM user, and you store the private key on your local machine. **git-remote-codecommit** extends Git itself, and does not require setting up Git credentials for the user.

In addition, you can generate access keys for each user. Use access keys when you access AWS services programmatically, either through one of the AWS SDKs or by using the AWS Command Line Interface (AWS CLI). The SDK and CLI tools use the access keys to cryptographically sign your requests. If you don't use the AWS tools, you must sign the requests yourself. CodeCommit supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **Amazon Web Services account root user** – When you sign up for AWS, you provide an email address and password that is associated with your Amazon Web Services account. These are your *root credentials*, and they provide complete access to all of your AWS resources. Some CodeCommit features are not available for root account users. In addition, the only way to use Git with your root account is to either install and configure **git-remote-codecommit** (recommended) or to configure the AWS credential helper, which is included with the AWS CLI. You cannot use Git credentials or SSH public-private key pairs with your root account user. For these reasons, we do not recommend using your root account user when interacting with CodeCommit.

  > **Important**
  >
  > For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.

- **IAM Identity Center and users in IAM Identity Center** – AWS IAM Identity Center expands the capabilities of AWS Identity and Access Management to provide a central place that brings together administration of users and their access to AWS accounts and cloud applications. While recommended as a best practice for most users working with AWS, IAM Identity Center does not currently provide mechanisms for Git credentials or SSH key pairs. These users can install and configure **git-remote-codecommit** to locally clone CodeCommit repositories, but not all integrated development environments (IDEs) support cloning, pushing, or pulling with **git-remote-codecommit**.

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

- **IAM role** – Like an IAM user, an IAM role is an IAM identity that you can create in your account to grant specific permissions.

  An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

  IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

> **Note**
>
> You cannot use Git credentials or SSH public-private key pairs with federated users. In addition, user preferences are not available for federated users. For information about how to set up connections using federated access, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

Access control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access CodeCommit resources. For example, you must have permissions to view repositories, push code, create and manage Git credentials, and so on.

The following sections describe how to manage permissions for CodeCommit. We recommend that you read the overview first.

- Overview of managing access permissions to your CodeCommit resources

- Using identity-based policies (IAM Policies) for CodeCommit
- CodeCommit permissions reference

Overview of managing access permissions to your CodeCommit resources

Every AWS resource is owned by an Amazon Web Services account. Permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services, such as AWS Lambda, also support attaching permissions policies to resources.

> **Note**
>
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who gets the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

- CodeCommit resources and operations
- Understanding resource ownership
- Managing access to resources
- Resource scoping in CodeCommit
- Specifying policy elements: resources, actions, effects, and principals
- Specifying conditions in a policy

CodeCommit resources and operations

In CodeCommit, the primary resource is a repository. Each resource has a unique Amazon Resource Names (ARN) associated with it. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the *Amazon Web Services General Reference*. CodeCommit does not currently support other resource types, which are referred to as subresources.

The following table describes how to specify CodeCommit resources.

Resource Type	ARN Format
Repository	arn:aws:codecommit:*region*:*account-id* :*repository-name*
All CodeCommit repositories	arn:aws:codecommit:*
All CodeCommit repositories owned by the specified account in the specified AWS Region	arn:aws:codecommit:*region*:*account-id* :*

> **Note**
>
> Most AWS services treat a colon (:) or a forward slash (/) in ARNs as the same character. However, CodeCommit requires an exact match in resource patterns and rules. When creating event patterns, be sure to use the correct ARN characters so that they match the ARN syntax in the resource.

For example, you can indicate a specific repository (*MyDemoRepo*) in your statement using its ARN as follows:

```
"Resource": "arn:aws:codecommit:us-west-2:111111111111:MyDemoRepo"
```

To specify all repositories that belong to a specific account, use the wildcard character (*) as follows:

```
"Resource": "arn:aws:codecommit:us-west-2:111111111111:*"
```

To specify all resources, or if a specific API action does not support ARNs, use the wildcard character (*) in the `Resource` element as follows:

```
"Resource": "*"
```

You can also use the wildcard character(*) to specify all resources that match part of a repository name. For example, the following ARN specifies any CodeCommit repository that begins with the name `MyDemo` and that is registered to the Amazon Web Services account `111111111111` in the `us-east-2` AWS Region:

```
arn:aws:codecommit:us-east-2:111111111111:MyDemo*
```

For a list of available operations that work with the CodeCommit resources, see CodeCommit permissions reference.

Understanding resource ownership

The Amazon Web Services account owns the resources that are created in the account, regardless of who created them. Specifically, the resource owner is the Amazon Web Services account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you create an IAM user in your Amazon Web Services account and grant permissions to create CodeCommit resources to that user, the user can create CodeCommit resources. However, your Amazon Web Services account, to which the user belongs, owns the CodeCommit resources.
- If you use the root account credentials of your Amazon Web Services account to create a rule, your Amazon Web Services account is the owner of the CodeCommit resource.
- If you create an IAM role in your Amazon Web Services account with permissions to create CodeCommit resources, anyone who can assume the role can create CodeCommit resources. Your Amazon Web Services account, to which the role belongs, owns the CodeCommit resources.

Managing access to resources

To manage access to AWS resources, you use permissions policies. A *permissions policy* describes who has access to what. The following section explains the options for creating permissions policies.

> **Note**
>
> This section discusses using IAM in the context of CodeCommit. It doesn't provide detailed information about the IAM service. For more information about IAM, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see IAM Policy Reference in the *IAM User Guide*.

Permissions policies that are attached to an IAM identity are referred to as identity-based policies (IAM policies). Permissions policies that are attached to a resource are referred to as resource-based policies. Currently, CodeCommit supports only identity-based policies (IAM policies).

Topics

- Identity-based policies (IAM policies)
- Resource-based policies

Identity-based policies (IAM policies)

To manage access to AWS resources, you attach permissions policies to IAM identities. In CodeCommit, you use identity-based policies to control access to repositories. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to view CodeCommit resources in the CodeCommit console, attach an identity-based permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (to grant cross-account permissions)** – Delegation, such as when you want to grant cross-account access, involves setting up a trust between the account that owns the resource (the trusting account), and the account that contains the users who need to access the resource (the trusted account). A permissions policy grants the user of a role the needed permissions to carry out the intended tasks on the resource. A trust policy specifies which trusted accounts are allowed to grant its users permissions to assume the role. For more information, see IAM Terms and Concepts.

  To grant cross-account permissions, attach an identity-based permissions policy to an IAM role. For example, the administrator in Account A can create a role to grant cross-account permissions to another Amazon Web Services account (for example, Account B) or an AWS service as follows:

  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. If you want to grant an AWS service permission to assume the role, the principal in the trust

policy can also be an AWS service principal. For more information, see Delegation in IAM Terms and Concepts.

For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following example policy allows a user to create a branch in a repository named *MyDemoRepo*:

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codecommit:CreateBranch"
      ],
      "Resource" : "arn:aws-eusc:codecommit:us-east-2:111122223333:MyDemoRepo"
    }
  ]
}
```

To restrict the calls and resources that users in your account have access to, create specific IAM policies, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for CodeCommit, see Customer managed identity policy examples.

Resource-based policies

Some services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a resource-based policy to an S3 bucket to manage access permissions to that bucket. CodeCommit doesn't support resource-based policies, but you can use tags to identify resources, which you can then use in IAM policies. For an example of a tag-based policy, see Identity-based policies (IAM policies).

Resource scoping in CodeCommit

In CodeCommit, you can scope identity-based policies and permissions to resources, as described in CodeCommit resources and operations. However, you cannot scope the `ListRepositories` permission to a resource. Instead, you must scope it to all resources (using the wildcard `*`). Otherwise, the action fails.

All other CodeCommit permissions can be scoped to resources.

Specifying policy elements: resources, actions, effects, and principals

You can create policies to allow or deny users access to resources, or allow or deny users to take specific actions on those resources. CodeCommit defines a set of public API operations that define how users work with the service, whether that is through the CodeCommit console, the SDKs, the AWS CLI, or by directly calling those APIs. To grant permissions for these API operations, CodeCommit defines a set of actions that you can specify in a policy.

Some API operations can require permissions for more than one action. For more information about resources and API operations, see CodeCommit resources and operations and CodeCommit permissions reference.

The following are the basic elements of a policy:

- **Resource** – To identify the resource that the policy applies to, you use an Amazon Resource Name (ARN). For more information, see CodeCommit resources and operations.
- **Action** – To identify resource operations that you want to allow or deny, you use action keywords. For example, depending on the specified `Effect`, the `codecommit:GetBranch` permission either allows or denies the user to perform the `GetBranch` operation, which gets details about a branch in a CodeCommit repository.
- **Effect** – You specify the effect, either allow or deny, that takes place when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the only type of policies that CodeCommit supports, the user that the policy is attached to is the implicit principal.

To learn more about IAM policy syntax, see IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the CodeCommit API actions and the resources that they apply to, see CodeCommit permissions reference.

Specifying conditions in a policy

When you grant permissions, you use the access policy language for IAM to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition and Policy Grammar in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to CodeCommit. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

Using identity-based policies (IAM Policies) for CodeCommit

The following examples of identity-based policies demonstrate how an account administrator can attach permissions policies to IAM identities (users, groups, and roles) to grant permissions to perform operations on CodeCommit resources.

> **Important**
>
> We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your CodeCommit resources. For more information, see Overview of managing access permissions to your CodeCommit resources.

Topics

- Permissions required to use the CodeCommit console
- Viewing resources in the console
- AWS managed policies for CodeCommit
- Customer managed policy examples

The following is an example of an identity-based permissions policy:

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codecommit:BatchGetRepositories"
      ],
      "Resource" : [
        "arn:aws-eusc:codecommit:us-east-2:111122223333:MyDemoRepo",
        "arn:aws-eusc:codecommit:us-east-2:111122223333:MyDemo*"
      ]
    }
  ]
}
```

This policy has one statement that allows a user to get information about the CodeCommit repository named `MyDestinationRepo` and all CodeCommit repositories that start with the name `MyDemo` in the **`us-east-2`** Region.

Permissions required to use the CodeCommit console

To see the required permissions for each CodeCommit API operation, and for more information about CodeCommit operations, see CodeCommit permissions reference.

To allow users to use the CodeCommit console, the administrator must grant them permissions for CodeCommit actions. For example, you could attach the AWSCodeCommitPowerUser managed policy or its equivalent to a user or group.

In addition to permissions granted to users by identity-based policies, CodeCommit requires permissions for AWS Key Management Service (AWS KMS) actions. An IAM user does not need explicit `Allow` permissions for these actions, but the user must not have any policies attached that set the following permissions to `Deny`:

```
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey",
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:DescribeKey"
```

For more information about encryption and CodeCommit, see AWS KMS and encryption.

Viewing resources in the console

The CodeCommit console requires the `ListRepositories` permission to display a list of repositories for your Amazon Web Services account in the AWS Region where you are signed in. The console also includes a **Go to resource** function to quickly perform a case insensitive search for resources. This search is performed in your Amazon Web Services account in the AWS Region where you are signed in. The following resources are displayed across the following services:

- AWS CodeBuild: Build projects
- AWS CodeCommit: Repositories
- AWS CodeDeploy: Applications
- AWS CodePipeline: Pipelines

To perform this search across resources in all services, you must have the following permissions:

- CodeBuild: `ListProjects`
- CodeCommit: `ListRepositories`
- CodeDeploy: `ListApplications`
- CodePipeline: `ListPipelines`

Results are not returned for a service's resources if you do not have permissions for that service. Even if you have permissions for viewing resources, specific resources will not be returned if there is an explicit `Deny` to view those resources.

AWS managed policies for CodeCommit

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when

a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant required permissions for common use cases. The managed policies for CodeCommit also provide permissions to perform operations in other services, such as IAM, Amazon SNS, and Amazon CloudWatch Events, as required for the responsibilities for the users who have been granted the policy in question. For example, the AWSCodeCommitFullAccess policy is an administrative-level user policy that allows users with this policy to create and manage CloudWatch Events rules for repositories (rules whose names are prefixed with `codecommit`) and Amazon SNS topics for notifications about repository-related events (topics whose names are prefixed with `codecommit`), as well as administer repositories in CodeCommit.

The following AWS managed policies, which you can attach to users in your account, are specific to CodeCommit.

Topics

- AWS managed policy: AWSCodeCommitFullAccess
- AWS managed policy: AWSCodeCommitPowerUser
- AWS managed policy: AWSCodeCommitReadOnly
- CodeCommit managed policies and notifications
- AWS CodeCommit managed policies and Amazon CodeGuru Reviewer
- CodeCommit updates to AWS managed policies

AWS managed policy: AWSCodeCommitFullAccess

You can attach the `AWSCodeCommitFullAccess` policy to your IAM identities. This policy grants full access to CodeCommit. Apply this policy only to administrative-level users to whom you want to grant full control over CodeCommit repositories and related resources in your Amazon Web Services account, including the ability to delete repositories.

The AWSCodeCommitFullAccess policy contains the following policy statement:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchEventsCodeCommitRulesAccess",
      "Effect": "Allow",
      "Action": [
        "events:DeleteRule",
        "events:DescribeRule",
        "events:DisableRule",
        "events:EnableRule",
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:ListTargetsByRule"
      ],
      "Resource": "arn:aws-eusc:events:*:*:rule/codecommit*"
    },
    {
      "Sid": "SNSTopicAndSubscriptionAccess",
      "Effect": "Allow",
      "Action": [
        "sns:CreateTopic",
        "sns:DeleteTopic",
        "sns:Subscribe",
        "sns:Unsubscribe",
        "sns:SetTopicAttributes"
      ],
      "Resource": "arn:aws-eusc:sns:*:*:codecommit*"
    },
    {
      "Sid": "SNSTopicAndSubscriptionReadAccess",
```

```
      "Effect": "Allow",
      "Action": [
        "sns:ListTopics",
        "sns:ListSubscriptionsByTopic",
        "sns:GetTopicAttributes"
      ],
      "Resource": "*"
    },
    {
      "Sid": "LambdaReadOnlyListAccess",
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMReadOnlyListAccess",
      "Effect": "Allow",
      "Action": [
        "iam:ListUsers"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMReadOnlyConsoleAccess",
      "Effect": "Allow",
      "Action": [
        "iam:ListAccessKeys",
        "iam:ListSSHPublicKeys",
        "iam:ListServiceSpecificCredentials"
      ],
      "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
    },
    {
      "Sid": "IAMUserSSHKeys",
      "Effect": "Allow",
      "Action": [
        "iam:DeleteSSHPublicKey",
        "iam:GetSSHPublicKey",
        "iam:ListSSHPublicKeys",
        "iam:UpdateSSHPublicKey",
        "iam:UploadSSHPublicKey"
      ],
```

```
            "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
          },
          {
            "Sid": "IAMSelfManageServiceSpecificCredentials",
            "Effect": "Allow",
            "Action": [
              "iam:CreateServiceSpecificCredential",
              "iam:UpdateServiceSpecificCredential",
              "iam:DeleteServiceSpecificCredential",
              "iam:ResetServiceSpecificCredential"
            ],
            "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
          },
          {
            "Sid": "CodeStarNotificationsReadWriteAccess",
            "Effect": "Allow",
            "Action": [
              "codestar-notifications:CreateNotificationRule",
              "codestar-notifications:DescribeNotificationRule",
              "codestar-notifications:UpdateNotificationRule",
              "codestar-notifications:DeleteNotificationRule",
              "codestar-notifications:Subscribe",
              "codestar-notifications:Unsubscribe"
            ],
            "Resource": "*",
            "Condition": {
              "ArnLike": {
                "codestar-notifications:NotificationsForResource": "arn:aws-
eusc:iam::*:role/Service*"
              }
            }
          },
          {
            "Sid": "CodeStarNotificationsListAccess",
            "Effect": "Allow",
            "Action": [
              "codestar-notifications:ListNotificationRules",
              "codestar-notifications:ListTargets",
              "codestar-notifications:ListTagsforResource",
              "codestar-notifications:ListEventTypes"
            ],
            "Resource": "*"
          },
          {
```

```
            "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
            "Effect": "Allow",
            "Action": [
              "sns:CreateTopic",
              "sns:SetTopicAttributes"
            ],
            "Resource": "arn:aws-eusc:sns:*:*:codestar-notifications*"
          },
          {
            "Sid": "AmazonCodeGuruReviewerFullAccess",
            "Effect": "Allow",
            "Action": [
              "codeguru-reviewer:AssociateRepository",
              "codeguru-reviewer:DescribeRepositoryAssociation",
              "codeguru-reviewer:ListRepositoryAssociations",
              "codeguru-reviewer:DisassociateRepository",
              "codeguru-reviewer:DescribeCodeReview",
              "codeguru-reviewer:ListCodeReviews"
            ],
            "Resource": "*"
          },
          {
            "Sid": "AmazonCodeGuruReviewerSLRCreation",
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "arn:aws-eusc:iam::*:role/aws-service-role/codeguru-
reviewer.amazonaws.com/AWSServiceRoleForAmazonCodeGuruReviewer",
            "Condition": {
              "StringLike": {
                "iam:AWSServiceName": "codeguru-reviewer.amazonaws.com"
              }
            }
          },
          {
            "Sid": "CloudWatchEventsManagedRules",
            "Effect": "Allow",
            "Action": [
              "events:PutRule",
              "events:PutTargets",
              "events:DeleteRule",
              "events:RemoveTargets"
            ],
            "Resource": "*",
            "Condition": {
```

```
            "StringEquals": {
              "events:ManagedBy": "codeguru-reviewer.amazonaws.com"
            }
          }
        },
        {
          "Sid": "CodeStarNotificationsChatbotAccess",
          "Effect": "Allow",
          "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
          "Resource": "*"
        },
        {
            "Sid": "CodeStarConnectionsReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "codestar-connections:ListConnections",
                "codestar-connections:GetConnection"
            ],
            "Resource": "arn:aws-eusc:codestar-connections:*:*:connection/*"
        }
      ]
    }
```

AWS managed policy: AWSCodeCommitPowerUser

You can attach the `AWSCodeCommitPowerUser` policy to your IAM identities. This policy allows users access to all of the functionality of CodeCommit and repository-related resources, except it does not allow them to delete CodeCommit repositories or create or delete repository-related resources in other AWS services, such as Amazon CloudWatch Events. We recommend that you apply this policy to most users.

The AWSCodeCommitPowerUser policy contains the following policy statement:

JSON

```
    {
      "Version":"2012-10-17",
      "Statement": [
```

```
        {
          "Effect": "Allow",
          "Action": [
            "codecommit:AssociateApprovalRuleTemplateWithRepository",
            "codecommit:BatchAssociateApprovalRuleTemplateWithRepositories",
            "codecommit:BatchDisassociateApprovalRuleTemplateFromRepositories",
            "codecommit:BatchGet*",
            "codecommit:BatchDescribe*",
            "codecommit:Create*",
            "codecommit:DeleteBranch",
            "codecommit:DeleteFile",
            "codecommit:Describe*",
            "codecommit:DisassociateApprovalRuleTemplateFromRepository",
            "codecommit:EvaluatePullRequestApprovalRules",
            "codecommit:Get*",
            "codecommit:List*",
            "codecommit:Merge*",
            "codecommit:OverridePullRequestApprovalRules",
            "codecommit:Put*",
            "codecommit:Post*",
            "codecommit:TagResource",
            "codecommit:Test*",
            "codecommit:UntagResource",
            "codecommit:Update*",
            "codecommit:GitPull",
            "codecommit:GitPush"
          ],
          "Resource": "*"
        },
        {
          "Sid": "CloudWatchEventsCodeCommitRulesAccess",
          "Effect": "Allow",
          "Action": [
            "events:DeleteRule",
            "events:DescribeRule",
            "events:DisableRule",
            "events:EnableRule",
            "events:PutRule",
            "events:PutTargets",
            "events:RemoveTargets",
            "events:ListTargetsByRule"
          ],
          "Resource": "arn:aws-eusc:events:*:*:rule/codecommit*"
        },
```

```
        {
          "Sid": "SNSTopicAndSubscriptionAccess",
          "Effect": "Allow",
          "Action": [
            "sns:Subscribe",
            "sns:Unsubscribe"
          ],
          "Resource": "arn:aws-eusc:sns:*:*:codecommit*"
        },
        {
          "Sid": "SNSTopicAndSubscriptionReadAccess",
          "Effect": "Allow",
          "Action": [
            "sns:ListTopics",
            "sns:ListSubscriptionsByTopic",
            "sns:GetTopicAttributes"
          ],
          "Resource": "*"
        },
        {
          "Sid": "LambdaReadOnlyListAccess",
          "Effect": "Allow",
          "Action": [
            "lambda:ListFunctions"
          ],
          "Resource": "*"
        },
        {
          "Sid": "IAMReadOnlyListAccess",
          "Effect": "Allow",
          "Action": [
            "iam:ListUsers"
          ],
          "Resource": "*"
        },
        {
          "Sid": "IAMReadOnlyConsoleAccess",
          "Effect": "Allow",
          "Action": [
            "iam:ListAccessKeys",
            "iam:ListSSHPublicKeys",
            "iam:ListServiceSpecificCredentials"
          ],
          "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
```

```
        },
        {
          "Sid": "IAMUserSSHKeys",
          "Effect": "Allow",
          "Action": [
            "iam:DeleteSSHPublicKey",
            "iam:GetSSHPublicKey",
            "iam:ListSSHPublicKeys",
            "iam:UpdateSSHPublicKey",
            "iam:UploadSSHPublicKey"
          ],
          "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
        },
        {
          "Sid": "IAMSelfManageServiceSpecificCredentials",
          "Effect": "Allow",
          "Action": [
            "iam:CreateServiceSpecificCredential",
            "iam:UpdateServiceSpecificCredential",
            "iam:DeleteServiceSpecificCredential",
            "iam:ResetServiceSpecificCredential"
          ],
          "Resource": "arn:aws-eusc:iam::*:user/${aws:username}"
        },
        {
          "Sid": "CodeStarNotificationsReadWriteAccess",
          "Effect": "Allow",
          "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
          ],
          "Resource": "*",
          "Condition": {
            "ArnLike": {
              "codestar-notifications:NotificationsForResource": "arn:aws-
eusc:iam::*:role/Service*"
            }
          }
        },
        {
          "Sid": "CodeStarNotificationsListAccess",
```

```
          "Effect": "Allow",
          "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
          ],
          "Resource": "*"
        },
        {
          "Sid": "AmazonCodeGuruReviewerFullAccess",
          "Effect": "Allow",
          "Action": [
            "codeguru-reviewer:AssociateRepository",
            "codeguru-reviewer:DescribeRepositoryAssociation",
            "codeguru-reviewer:ListRepositoryAssociations",
            "codeguru-reviewer:DisassociateRepository",
            "codeguru-reviewer:DescribeCodeReview",
            "codeguru-reviewer:ListCodeReviews"
          ],
          "Resource": "*"
        },
        {
          "Sid": "AmazonCodeGuruReviewerSLRCreation",
          "Action": "iam:CreateServiceLinkedRole",
          "Effect": "Allow",
          "Resource": "arn:aws-eusc:iam::*:role/aws-service-role/codeguru-
reviewer.amazonaws.com/AWSServiceRoleForAmazonCodeGuruReviewer",
          "Condition": {
            "StringLike": {
              "iam:AWSServiceName": "codeguru-reviewer.amazonaws.com"
            }
          }
        },
        {
          "Sid": "CloudWatchEventsManagedRules",
          "Effect": "Allow",
          "Action": [
            "events:PutRule",
            "events:PutTargets",
            "events:DeleteRule",
            "events:RemoveTargets"
          ],
          "Resource": "*",
```

```
        "Condition": {
          "StringEquals": {
            "events:ManagedBy": "codeguru-reviewer.amazonaws.com"
          }
        }
      },
      {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
          "chatbot:DescribeSlackChannelConfigurations",
          "chatbot:ListMicrosoftTeamsChannelConfigurations"
        ],
        "Resource": "*"
      },
      {
          "Sid": "CodeStarConnectionsReadOnlyAccess",
          "Effect": "Allow",
          "Action": [
              "codestar-connections:ListConnections",
              "codestar-connections:GetConnection"
          ],
          "Resource": "arn:aws-eusc:codestar-connections:*:*:connection/*"
      }
    ]
  }
```

AWS managed policy: AWSCodeCommitReadOnly

You can attach the `AWSCodeCommitReadOnly` policy to your IAM identities. This policy grants read-only access to CodeCommit and repository-related resources in other AWS services, as well as the ability to create and manage their own CodeCommit-related resources (such as Git credentials and SSH keys for their IAM user to use when accessing repositories). Apply this policy to users to whom you want to grant the ability to read the contents of a repository, but not make any changes to its contents.

The AWSCodeCommitReadOnly policy contains the following policy statement:

JSON

```
  {
```

```
        "Version":"2012-10-17",
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "codecommit:BatchGet*",
                 "codecommit:BatchDescribe*",
                 "codecommit:Describe*",
                 "codecommit:EvaluatePullRequestApprovalRules",
                 "codecommit:Get*",
                 "codecommit:List*",
                 "codecommit:GitPull"
              ],
              "Resource":"*"
           },
           {
              "Sid":"CloudWatchEventsCodeCommitRulesReadOnlyAccess",
              "Effect":"Allow",
              "Action":[
                 "events:DescribeRule",
                 "events:ListTargetsByRule"
              ],
              "Resource":"arn:aws-eusc:events:*:*:rule/codecommit*"
           },
           {
              "Sid":"SNSSubscriptionAccess",
              "Effect":"Allow",
              "Action":[
                 "sns:ListTopics",
                 "sns:ListSubscriptionsByTopic",
                 "sns:GetTopicAttributes"
              ],
              "Resource":"*"
           },
           {
              "Sid":"LambdaReadOnlyListAccess",
              "Effect":"Allow",
              "Action":[
                 "lambda:ListFunctions"
              ],
              "Resource":"*"
           },
           {
              "Sid":"IAMReadOnlyListAccess",
```

```
            "Effect":"Allow",
            "Action":[
               "iam:ListUsers"
            ],
            "Resource":"*"
         },
         {
            "Sid":"IAMReadOnlyConsoleAccess",
            "Effect":"Allow",
            "Action":[
               "iam:ListAccessKeys",
               "iam:ListSSHPublicKeys",
               "iam:ListServiceSpecificCredentials",
               "iam:GetSSHPublicKey"
            ],
            "Resource":"arn:aws-eusc:iam::*:user/${aws:username}"
         },
         {
            "Sid":"CodeStarNotificationsReadOnlyAccess",
            "Effect":"Allow",
            "Action":[
               "codestar-notifications:DescribeNotificationRule"
            ],
            "Resource":"*",
            "Condition":{
               "ArnLike":{
                  "codestar-notifications:NotificationsForResource":"arn:aws-
eusc:codecommit:us-east-2:111122223333:*"
               }
            }
         },
         {
            "Sid":"CodeStarNotificationsListAccess",
            "Effect":"Allow",
            "Action":[
               "codestar-notifications:ListNotificationRules",
               "codestar-notifications:ListEventTypes",
               "codestar-notifications:ListTargets"
            ],
            "Resource":"*"
         },
         {
            "Sid": "AmazonCodeGuruReviewerReadOnlyAccess",
            "Effect": "Allow",
```

```
            "Action": [
               "codeguru-reviewer:DescribeRepositoryAssociation",
               "codeguru-reviewer:ListRepositoryAssociations",
               "codeguru-reviewer:DescribeCodeReview",
               "codeguru-reviewer:ListCodeReviews"
            ],
            "Resource": "*"
         },
         {
           "Sid": "CodeStarConnectionsReadOnlyAccess",
           "Effect": "Allow",
           "Action": [
               "codestar-connections:ListConnections",
               "codestar-connections:GetConnection"
           ],
           "Resource": "arn:aws-eusc:codestar-connections:*:*:connection/*"
       }
    ]
}
```

CodeCommit managed policies and notifications

AWS CodeCommit supports notifications, which can notify users of important changes to repositories. Managed policies for CodeCommit include policy statements for notification functionality. For more information, see What are notifications?.

Permissions related to notifications in full access managed policies

The `AWSCodeCommitFullAccess` managed policy includes the following statements to allow full access to notifications. Users with this managed policy applied can also create and manage Amazon SNS topics for notifications, subscribe and unsubscribe users to topics, list topics to choose as targets for notification rules, and list Amazon Q Developer in chat applications clients configured for Slack.

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
```

```
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codecommit:*"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource,"
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

Permissions related to notifications in read-only managed policies

The AWSCodeCommitReadOnlyAccess managed policy includes the following statements to allow read-only access to notifications. Users with this managed policy applied can view notifications for resources, but cannot create, manage, or subscribe to them.

```
  {
       "Sid": "CodeStarNotificationsPowerUserAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:DescribeNotificationRule"
       ],
       "Resource": "*",
       "Condition" : {
           "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codecommit:*"}
       }
   },
   {
       "Sid": "CodeStarNotificationsListAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:ListNotificationRules",
           "codestar-notifications:ListEventTypes",
           "codestar-notifications:ListTargets"
       ],
       "Resource": "*"
   }
```

Permissions related to notifications in other managed policies

The AWSCodeCommitPowerUser managed policy includes the following statements to allow users to create, edit, and subscribe to notifications. Users cannot delete notification rules or manage tags for resources.

```
   {
       "Sid": "CodeStarNotificationsReadWriteAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:CreateNotificationRule",
           "codestar-notifications:DescribeNotificationRule",
           "codestar-notifications:UpdateNotificationRule",
```

```
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codecommit*"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

For more information about IAM and notifications, see Identity and Access Management for AWS CodeStar Notifications.

AWS CodeCommit managed policies and Amazon CodeGuru Reviewer

CodeCommit supports Amazon CodeGuru Reviewer, an automated code review service that uses program analysis and machine learning to detect common issues and recommend fixes in your Java or Python code. Managed policies for CodeCommit include policy statements for CodeGuru Reviewer functionality. For more information, see What Is Amazon CodeGuru Reviewer.

Permissions related to CodeGuru Reviewer in AWSCodeCommitFullAccess

The `AWSCodeCommitFullAccess` managed policy includes the following statements to allow CodeGuru Reviewer to be associated and disassociated with CodeCommit repositories. Users with this managed policy applied can also view the association status between CodeCommit repositories and CodeGuru Reviewer and view the status of review jobs for pull requests.

```
    {
      "Sid": "AmazonCodeGuruReviewerFullAccess",
      "Effect": "Allow",
      "Action": [
        "codeguru-reviewer:AssociateRepository",
        "codeguru-reviewer:DescribeRepositoryAssociation",
        "codeguru-reviewer:ListRepositoryAssociations",
        "codeguru-reviewer:DisassociateRepository",
        "codeguru-reviewer:DescribeCodeReview",
        "codeguru-reviewer:ListCodeReviews"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AmazonCodeGuruReviewerSLRCreation",
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/codeguru-
reviewer.amazonaws.com/AWSServiceRoleForAmazonCodeGuruReviewer",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "codeguru-reviewer.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchEventsManagedRules",
      "Effect": "Allow",
      "Action": [
```

```
        "events:PutRule",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "events:ManagedBy": "codeguru-reviewer.amazonaws.com"
        }
      }
    }
```

Permissions related to CodeGuru Reviewer in AWSCodeCommitPowerUser

The `AWSCodeCommitPowerUser` managed policy includes the following statements to allow users to associate and disassociate repositories with CodeGuru Reviewer, view association status, and view the status of review jobs for pull requests.

```
    {
      "Sid": "AmazonCodeGuruReviewerFullAccess",
      "Effect": "Allow",
      "Action": [
        "codeguru-reviewer:AssociateRepository",
        "codeguru-reviewer:DescribeRepositoryAssociation",
        "codeguru-reviewer:ListRepositoryAssociations",
        "codeguru-reviewer:DisassociateRepository",
        "codeguru-reviewer:DescribeCodeReview",
        "codeguru-reviewer:ListCodeReviews"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AmazonCodeGuruReviewerSLRCreation",
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/codeguru-
reviewer.amazonaws.com/AWSServiceRoleForAmazonCodeGuruReviewer",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "codeguru-reviewer.amazonaws.com"
        }
      }
```

```
    },
    {
      "Sid": "CloudWatchEventsManagedRules",
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "events:ManagedBy": "codeguru-reviewer.amazonaws.com"
        }
      }
    }
```

Permissions related to CodeGuru Reviewer in AWSCodeCommitReadOnly

The `AWSCodeCommitReadOnlyAccess` managed policy includes the following statements to allow read-only access to CodeGuru Reviewer association status and view the status of review jobs for pull requests. Users with this managed policy applied cannot associate or disassociate repositories.

```
     {
      "Sid": "AmazonCodeGuruReviewerReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
            "codeguru-reviewer:DescribeRepositoryAssociation",
            "codeguru-reviewer:ListRepositoryAssociations",
            "codeguru-reviewer:DescribeCodeReview",
            "codeguru-reviewer:ListCodeReviews"
      ],
      "Resource": "*"
    }
```

Amazon CodeGuru Reviewer service-linked role

When you associate a repository with CodeGuru Reviewer, a service-linked role is created so that CodeGuru Reviewer can detect issues and recommend fixes for Java or Python code in pull

requests. The service-linked role is named AWSServiceRoleForAmazonCodeGuruReviewer. For more information, see Using Service-Linked Roles for Amazon CodeGuru Reviewer.

For more information, see AWS Managed Policies in the *IAM User Guide*.

CodeCommit updates to AWS managed policies

View details about updates to AWS managed policies for CodeCommit since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on AWS CodeCommit User Guide document history.

Change	Description	Date
AWS managed policy: AWSCodeCommitFullAccess and AWS managed policy: AWSCodeCommitPowerUser – Update to existing policies	CodeCommit added a permission to these policies to support an additiona l notification type using Amazon Q Developer in chat applications. The AWSCodeCommitPower User and AWSCodeCo mmitFullAccess policies have been changed to add a permission, `chatbot:L istMicrosoftTeamsC hannelConfiguratio ns` .	May 16, 2023
AWS managed policy: AWSCodeCommitReadOnly – Update to an existing policy	CodeCommit removed a duplicate permission from the policy. The AWSCodeCommitReadO nly has been changed to remove a duplicate permissio n, `"iam:ListAccessKey s"` .	August 18, 2021

Change	Description	Date
CodeCommit started tracking changes	CodeCommit started tracking changes for its AWS managed policies.	August 18, 2021

Customer managed policy examples

You can create your own custom IAM policies to allow permissions for CodeCommit actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions. You can also create your own custom IAM policies for integration between CodeCommit and other AWS services.

Topics

- Customer managed identity policy examples

Customer managed identity policy examples

The following example IAM policies grant permissions for various CodeCommit actions. Use them to limit CodeCommit access for your IAM users and roles. These policies control the ability to perform actions with the CodeCommit console, API, AWS SDKs, or the AWS CLI.

> **Note**
>
> All examples use the US West (Oregon) Region (us-west-2) and contain fictitious account IDs.

Examples

- Example 1: Allow a user to perform CodeCommit operations in a single AWS Region
- Example 2: Allow a user to use Git for a single repository
- Example 3: Allow a user connecting from a specified IP address range access to a repository
- Example 4: Deny or allow actions on branches
- Example 5: Deny or allow actions on repositories with tags

Example 1: Allow a user to perform CodeCommit operations in a single AWS Region

The following permissions policy uses a wildcard character (`"codecommit:*"`) to allow users to perform all CodeCommit actions in the us-east-2 Region and not from other AWS Regions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codecommit:*",
            "Resource": "arn:aws-eusc:codecommit:us-east-2:111111111111:*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": "us-east-2"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "codecommit:ListRepositories",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": "us-east-2"
                }
            }
        }
    ]
}
```

Example 2: Allow a user to use Git for a single repository

In CodeCommit, the `GitPull` IAM policy permissions apply to any Git client command where data is retrieved from CodeCommit, including **git fetch**, **git clone**, and so on. Similarly, the `GitPush` IAM policy permissions apply to any Git client command where data is sent to CodeCommit. For example, if the `GitPush` IAM policy permission is set to `Allow`, a user can push the deletion of a branch using the Git protocol. That push is unaffected by any permissions applied to the

`DeleteBranch` operation for that IAM user. The `DeleteBranch` permission applies to actions performed with the console, the AWS CLI, the SDKs, and the API, but not the Git protocol.

The following example allows the specified user to pull from, and push to, the CodeCommit repository named `MyDemoRepo`:

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codecommit:GitPull",
        "codecommit:GitPush"
      ],
      "Resource" : "arn:aws-eusc:codecommit:us-east-2:111122223333:MyDemoRepo"
    }
  ]
}
```

Example 3: Allow a user connecting from a specified IP address range access to a repository

You can create a policy that only allows users to connect to a CodeCommit repository if their IP address is within a certain IP address range. There are two equally valid approaches to this. You can create a `Deny` policy that disallows CodeCommit operations if the IP address for the user is not within a specific block, or you can create an `Allow` policy that allows CodeCommit operations if the IP address for the user is within a specific block.

You can create a `Deny` policy that denies access to all users who are not within a certain IP range. For example, you could attach the AWSCodeCommitPowerUser managed policy and a customer-managed policy to all users who require access to your repository. The following example policy denies all CodeCommit permissions to users whose IP addresses are not within the specified IP address block of 203.0.113.0/16:

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": [
            "codecommit:*"
         ],
         "Resource": "*",
         "Condition": {
            "NotIpAddress": {
               "aws:SourceIp": [
                  "203.0.113.0/16"
               ]
            }
         }
      }
   ]
}
```

The following example policy allows the specified user to access a CodeCommit repository named MyDemoRepo with the equivalent permissions of the AWSCodeCommitPowerUser managed policy only if their IP address is within the specified address block of 203.0.113.0/16:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codecommit:BatchGetRepositories",
            "codecommit:CreateBranch",
            "codecommit:CreateRepository",
            "codecommit:Get*",
            "codecommit:GitPull",
            "codecommit:GitPush",
            "codecommit:List*",
            "codecommit:Put*",
            "codecommit:Post*",
            "codecommit:Merge*",
```

```
            "codecommit:TagResource",
            "codecommit:Test*",
            "codecommit:UntagResource",
            "codecommit:Update*"
         ],
         "Resource": "arn:aws-eusc:codecommit:us-east-2:111122223333:MyDemoRepo",
         "Condition": {
            "IpAddress": {
               "aws:SourceIp": [
                  "203.0.113.0/16"
               ]
            }
         }
      }
   ]
}
```

Example 4: Deny or allow actions on branches

You can create a policy that denies users permissions to actions you specify on one or more branches. Alternatively, you can create a policy that allows actions on one or more branches that they might not otherwise have in other branches of a repository. You can use these policies with the appropriate managed (predefined) policies. For more information, see Limit pushes and merges to branches in AWS CodeCommit.

For example, you can create a `Deny` policy that denies users the ability to make changes to a branch named main, including deleting that branch, in a repository named *MyDemoRepo*. You can use this policy with the **AWSCodeCommitPowerUser** managed policy. Users with these two policies applied would be able to create and delete branches, create pull requests, and all other actions as allowed by **AWSCodeCommitPowerUser**, but they would not be able to push changes to the branch named *main*, add or edit a file in the *main* branch in the CodeCommit console, or merge branches or a pull request into the *main* branch. Because `Deny` is applied to `GitPush`, you must include a `Null` statement in the policy, to allow initial `GitPush` calls to be analyzed for validity when users make pushes from their local repos.

Tip

If you want to create a policy that applies to all branches named *main* in all repositories in your Amazon Web Services account, for `Resource`, specify an asterisk (*) instead of a repository ARN.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "codecommit:GitPush",
                "codecommit:DeleteBranch",
                "codecommit:PutFile",
                "codecommit:Merge*"
            ],
            "Resource": "arn:aws-eusc:codecommit:us-
east-2:111111111111:MyDemoRepo",
            "Condition": {
                "StringEqualsIfExists": {
                    "codecommit:References": [
                        "refs/heads/main"
                    ]
                },
                "Null": {
                    "codecommit:References": "false"
                }
            }
        }
    ]
}
```

The following example policy allows a user to make changes to a branch named main in all repositories in an Amazon Web Services account. It does not allow changes to any other branches. You might use this policy with the AWSCodeCommitReadOnly managed policy to allow automated

pushes to the repository in the main branch. Because the Effect is `Allow`, this example policy would not work with managed policies such as AWSCodeCommitPowerUser.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:GitPush",
                "codecommit:Merge*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "codecommit:References": [
                        "refs/heads/main"
                    ]
                }
            }
        }
    ]
}
```

Example 5: Deny or allow actions on repositories with tags

You can create a policy that allows or denies actions on repositories based on the AWS tags associated with those repositories, and then apply those policies to the IAM groups you configure for managing IAM users. For example, you can create a policy that denies all CodeCommit actions on any repositories with the AWS tag key *Status* and the key value of *Secret*, and then apply that policy to the IAM group you created for general developers (*`Developers`*). You then need to make sure that the developers working on those tagged repositories are not members of that general *`Developers`* group, but belong instead to a different IAM group that does not have the restrictive policy applied (*SecretDevelopers*).

The following example denies all CodeCommit actions on repositories tagged with the key *Status* and the key value of *Secret*:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codecommit:Associate*",
        "codecommit:Batch*",
        "codecommit:CancelUploadArchive",
        "codecommit:CreateBranch",
        "codecommit:CreateCommit",
        "codecommit:CreatePullRequest*",
        "codecommit:CreateRepository",
        "codecommit:CreateUnreferencedMergeCommit",
        "codecommit:DeleteBranch",
        "codecommit:DeleteCommentContent",
        "codecommit:DeleteFile",
        "codecommit:DeletePullRequest*",
        "codecommit:DeleteRepository",
        "codecommit:Describe*",
        "codecommit:DisassociateApprovalRuleTemplateFromRepository",
        "codecommit:EvaluatePullRequestApprovalRules",
        "codecommit:GetBlob",
        "codecommit:GetBranch",
        "codecommit:GetComment*",
        "codecommit:GetCommit",
        "codecommit:GetDifferences*",
        "codecommit:GetFile",
        "codecommit:GetFolder",
        "codecommit:GetMerge*",
        "codecommit:GetObjectIdentifier",
        "codecommit:GetPullRequest*",
        "codecommit:GetReferences",
        "codecommit:GetRepository*",
        "codecommit:GetTree",
        "codecommit:GetUploadArchiveStatus",
        "codecommit:Git*",
        "codecommit:ListAssociatedApprovalRuleTemplatesForRepository",
        "codecommit:ListBranches",
        "codecommit:ListPullRequests",
        "codecommit:ListTagsForResource",
```

```
        "codecommit:Merge*",
        "codecommit:OverridePullRequestApprovalRules",
        "codecommit:Post*",
        "codecommit:Put*",
        "codecommit:TagResource",
        "codecommit:TestRepositoryTriggers",
        "codecommit:UntagResource",
        "codecommit:UpdateComment",
        "codecommit:UpdateDefaultBranch",
        "codecommit:UpdatePullRequest*",
        "codecommit:UpdateRepository*",
        "codecommit:UploadArchive"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Status": "Secret"
        }
      }
    }
  ]
}
```

You can further refine this strategy by specifying specific repositories, rather than all repositories, as resources. You can also create policies that allow CodeCommit actions on all repositories that are not tagged with specific tags. For example, the following policy allows the equivalent of **AWSCodeCommitPowerUser** permissions for CodeCommit actions, except that it only allows CodeCommit actions on repositories not tagged with the specified tags:

> **Note**
>
> This policy example only includes actions for CodeCommit. It does not include actions for other AWS services that are included in the **AWSCodeCommitPowerUser** managed policy. For more information, see .AWS managed policy: AWSCodeCommitPowerUser.

JSON

```
{
  "Version":"2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "codecommit:Associate*",
      "codecommit:Batch*",
      "codecommit:CancelUploadArchive",
      "codecommit:CreateBranch",
      "codecommit:CreateCommit",
      "codecommit:CreatePullRequest*",
      "codecommit:CreateRepository",
      "codecommit:CreateUnreferencedMergeCommit",
      "codecommit:DeleteBranch",
      "codecommit:DeleteCommentContent",
      "codecommit:DeleteFile",
      "codecommit:DeletePullRequest*",
      "codecommit:Describe*",
      "codecommit:DisassociateApprovalRuleTemplateFromRepository",
      "codecommit:EvaluatePullRequestApprovalRules",
      "codecommit:GetBlob",
      "codecommit:GetBranch",
      "codecommit:GetComment*",
      "codecommit:GetCommit",
      "codecommit:GetDifferences*",
      "codecommit:GetFile",
      "codecommit:GetFolder",
      "codecommit:GetMerge*",
      "codecommit:GetObjectIdentifier",
      "codecommit:GetPullRequest*",
      "codecommit:GetReferences",
      "codecommit:GetRepository*",
      "codecommit:GetTree",
      "codecommit:GetUploadArchiveStatus",
      "codecommit:Git*",
      "codecommit:ListAssociatedApprovalRuleTemplatesForRepository",
      "codecommit:ListBranches",
      "codecommit:ListPullRequests",
      "codecommit:ListTagsForResource",
      "codecommit:Merge*",
      "codecommit:OverridePullRequestApprovalRules",
      "codecommit:Post*",
      "codecommit:Put*",
      "codecommit:TagResource",
      "codecommit:TestRepositoryTriggers",
```

```
        "codecommit:UntagResource",
        "codecommit:UpdateComment",
        "codecommit:UpdateDefaultBranch",
        "codecommit:UpdatePullRequest*",
        "codecommit:UpdateRepository*",
        "codecommit:UploadArchive"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/Status": "Secret",
          "aws:ResourceTag/Team": "Saanvi"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:CreateApprovalRuleTemplate",
        "codecommit:GetApprovalRuleTemplate",
        "codecommit:ListApprovalRuleTemplates",
        "codecommit:ListRepositories",
        "codecommit:ListRepositoriesForApprovalRuleTemplate",
        "codecommit:UpdateApprovalRuleTemplateContent",
        "codecommit:UpdateApprovalRuleTemplateDescription",
        "codecommit:UpdateApprovalRuleTemplateName"
      ],
      "Resource": "*"
    }
  ]
}
```

CodeCommit permissions reference

The following tables list each CodeCommit API operation, the corresponding actions for which you can grant permissions, and the format of the resource ARN to use for granting permissions. The CodeCommit APIs are grouped into tables based on the scope of the actions allowed by that API. Refer to it when setting up Access control and writing permissions policies that you can attach to an IAM identity (identity-based policies).

When you create a permissions policy, you specify the actions in the policy's `Action` field. You specify the resource value in the policy's `Resource` field as an ARN, with or without a wildcard character (*).

To express conditions in your CodeCommit policies, use AWS-wide condition keys. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*. For complete information about actions, resources, and condition keys for CodeCommit in IAM policies, see Actions, resources, and condition keys for AWS CodeCommit.

> **Note**
>
> To specify an action, use the `codecommit:` prefix followed by the API operation name (for example, `codecommit:GetRepository` or `codecommit:CreateRepository`.

Using Wildcards

To specify multiple actions or resources, use a wildcard character (*) in your ARN. For example, `codecommit:*` specifies all CodeCommit actions and `codecommit:Get*` specifies all CodeCommit actions that begin with the word `Get`. The following example grants access to all repositories with names that begin with `MyDemo`.

```
arn:aws:codecommit:us-west-2:111111111111:MyDemo*
```

You can use wildcards only with the *`repository-name`* resources listed in the following table. You can't use wildcards with *`region`* or *`account-id`* resources. For more information about wildcards, see IAM Identifiers in *IAM User Guide*.

Topics

- Required permissions for Git client commands
- Permissions for actions on branches
- Permissions for actions on merges
- Permissions for actions on pull requests
- Permissions for actions on approval rule templates
- Permissions for actions on individual files

- Permissions for actions on comments
- Permissions for actions on committed code
- Permissions for actions on repositories
- Permissions for actions on tags
- Permissions for actions on triggers
- Permissions for actions on CodePipeline integration

Required permissions for Git client commands

In CodeCommit, the `GitPull` IAM policy permissions apply to any Git client command where data is retrieved from CodeCommit, including **git fetch**, **git clone**, and so on. Similarly, the `GitPush` IAM policy permissions apply to any Git client command where data is sent to CodeCommit. For example, if the `GitPush` IAM policy permission is set to `Allow`, a user can push the deletion of a branch using the Git protocol. That push is unaffected by any permissions applied to the `DeleteBranch` operation for that IAM user. The `DeleteBranch` permission applies to actions performed with the console, the AWS CLI, the SDKs, and the API, but not the Git protocol.

`GitPull` and `GitPush` are IAM policy permissions. They are not API actions.

CodeCommit Required Permissions for Actions for Git Client Commands

GitPull

Action(s): `codecommit:GitPull`

Required to pull information from a CodeCommit repository to a local repo. This is an IAM policy permission only, not an API action.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GitPush

Action(s): `codecommit:Git Push`

Required to push information from a local repo to a CodeCommit repository. This is an IAM policy permission only, not an API action.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on branches

The following permissions allow or deny actions on branches in CodeCommit repositories. These permissions pertain only to actions performed in the CodeCommit console and with the CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For example, the **git show-branch -r** command displays a list of remote branches for a repository and its commits using the Git protocol. It's not affected by any permissions for the CodeCommit `ListBranches` operation.

For more information about policies for branches, see Limit pushes and merges to branches in AWS CodeCommit and Customer managed policy examples.

CodeCommit API Operations and Required Permissions for Actions on Branches

CreateBranch

Action(s): `codecommit:CreateBranch`

Required to create a branch in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

DeleteBranch

Action(s): `codecommit:DeleteBranch`

Required to delete a branch from a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetBranch

Action(s): `codecommit:GetBranch`

Required to get details about a branch in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

ListBranches

Action(s): `codecommit:ListBranches`

Required to get a list of branches in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

MergeBranchesByFastForward

Action(s): `codecommit:MergeBranchesByFastForward`

Required to merge two branches using the fast-forward merge strategy in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

MergeBranchesBySquash

Action(s): `codecommit:ListBranches`

Required to merge two branches using the squash merge strategy in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

MergeBranchesByThreeWay

Action(s): `codecommit:ListBranches`

Required to merge two branches using the three-way merge strategy in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdateDefaultBranch

Action(s): `codecommit:UpdateDefaultBranch`

Required to change the default branch in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

Permissions for actions on merges

The following permissions allow or deny actions on merges in CodeCommit repositories. These permissions pertain to actions performed with the CodeCommit console and the CodeCommit API, and commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For related permissions on branches, see Permissions for actions on branches. For related permissions on pull requests, see Permissions for actions on pull requests.

CodeCommit API Operations and Required Permissions for Actions for Merge Commands

BatchDescribeMergeConflicts

Action(s): `codecommit:BatchDescribeMergeConflicts`

Required to return information about conflicts in a merge between commits in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CreateUnreferencedMergeCommit

Action(s): `codecommit:CreateUnreferencedMergeCommit`

Required to create an unreferenced commit between two branches or commits in a CodeCommit repository for the purpose of comparing them and identifying any potential conflicts.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

DescribeMergeConflicts

Action(s): `codecommit:DescribeMergeConflicts`

Required to return information about merge conflicts between the base, source, and destination versions of a file in a potential merge in an CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetMergeCommit

Action(s): `codecommit:GetMergeCommit`

Required to return information about the merge between a source and destination commit in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetMergeOptions

Action(s): `codecommit:GetMergeOptions`

Required to return information about the available merge options between two branches or commit specifiers in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on pull requests

The following permissions allow or deny actions on pull requests in CodeCommit repositories. These permissions pertain to actions performed with the CodeCommit console and the CodeCommit API, and commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For related permissions on comments, see Permissions for actions on comments.

CodeCommit API Operations and Required Permissions for Actions on Pull Requests

BatchGetPullRequests

Action(s): `codecommit:BatchGetPullRequests`

Required to return information about one or more pull requests in a CodeCommit repository. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CreatePullRequest

Action(s): `codecommit:CreatePullRequest`

Required to create a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CreatePullRequestApprovalRule

Action(s): `codecommit:CreatePullRequestApprovalRule`

Required to create an approval rule for a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

DeletePullRequestApprovalRule

Action(s): `codecommit:DeletePullRequestApprovalRule`

Required to delete an approval rule for a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

DescribePullRequestEvents

Action(s): `codecommit:DescribePullRequestEvents`

Required to return information about one or more pull request events in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

EvaluatePullRequestApprovalRules

Action(s): `codecommit:EvaluatePullRequestApprovalRules`

Required to evaluate whether a pull request has met all the conditions specified in its associated approval rules in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetCommentsForPullRequest

Action(s): `codecommit:GetCommentsForPullRequest`

Required to return comments made on a pull request.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetCommitsFromMergeBase

Action(s): `codecommit:GetCommitsFromMergeBase`

Required to return information about the difference between commits in the context of a potential merge. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetMergeConflicts

Action(s): `codecommit:GetMergeConflicts`

Required to return information about merge conflicts between the source and destination branch in a pull request.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetPullRequest

Action(s): `codecommit:GetPullRequest`

Required to return information about a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetPullRequestApprovalStates

Action(s): `codecommit:GetPullRequestApprovalStates`

Required to return information about the approval states for a specified pull request.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetPullRequestOverrideState

Action(s): `codecommit:GetPullRequestOverrideState`

Required to return information about whether approval rules have been set aside (overridden) for a pull request, and if so, the Amazon Resource Name (ARN) of the user or identity that overrode the rules and their requirements for the pull request.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

ListPullRequests

Action(s): `codecommit:ListPullRequests`

Required to list pull requests in a repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

MergePullRequestByFastForward

Action(s): `codecommit:MergePullRequestByFastForward`

Required to close a pull request and attempt to merge the source branch into the destination branch of a pull request using the fast-forward merge strategy.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

MergePullRequestBySquash

Action(s): `codecommit:MergePullRequestBySquash`

Required to close a pull request and attempt to merge the source branch into the destination branch of a pull request using the squash merge strategy.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

MergePullRequestByThreeWay

Action(s): `codecommit:MergePullRequestByThreeWay`

Required to close a pull request and attempt to merge the source branch into the destination branch of a pull request using the three-way merge strategy.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

OverridePullRequestApprovalRules

Action(s): `codecommit:OverridePullRequestApprovalRules`

Required to set aside all approval rule requirements for a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

PostCommentForPullRequest

Action(s): `codecommit:PostCommentForPullRequest`

Required to post a comment on a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdatePullRequestApprovalRuleContent

Action(s): `codecommit:UpdatePullRequestApprovalRuleContent`

Required to change the structure of an approval rule for a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdatePullRequestApprovalState

Action(s): `codecommit:UpdatePullRequestApprovalState`

Required to update the state of an approval on a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdatePullRequestDescription

Action(s): `codecommit:UpdatePullRequestDescription`

Required to change the description of a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdatePullRequestStatus

Action(s): `codecommit:UpdatePullRequestStatus`

Required to change the status of a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

UpdatePullRequestTitle

Action(s): `codecommit:UpdatePullRequestTitle`

Required to change the title of a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

Permissions for actions on approval rule templates

The following permissions allow or deny actions on approval rule templates in CodeCommit repositories. These permissions pertain only to actions performed in the CodeCommit console, the CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For related permissions on pull requests, see Permissions for actions on pull requests.

CodeCommit API Operations and Required Permissions for Actions on Approval Rule Templates

AssociateApprovalRuleTemplateWithRepository

Action(s): `codecommit:AssociateApprovalRuleTemplateWithRepository`

Required to associate a template with a specified repository in an Amazon Web Services account. Once associated, this automatically creates approval rules that match the template conditions on every pull request created in the specified repository.

Resource: *

BatchAssociateApprovalRuleTemplateWithRepositories

Action(s): `codecommit:BatchAssociateApprovalRuleTemplateWithRepositories`

Required to associate a template with one or more specified repositories in an Amazon Web Services account.

Resource: *

BatchDisassociateApprovalRuleTemplateFromRepositories

Action(s): `codecommit:BatchDisassociateApprovalRuleTemplateFromRepositories`

Required to disassociate a template from one or more specified repositories in an Amazon Web Services account.

Resource: *

CreateApprovalRuleTemplate

Action(s): `codecommit:CreateApprovalRuleTemplate`

Required to create a template for approval rules that can then be associated with one or more repositories in your Amazon Web Services account.

Resource: *

DeleteApprovalRuleTemplate

Action(s): `codecommit:DeleteApprovalRuleTemplate`

Required to delete an approval rule template from an AWS account.

Resource: *

DisassociateApprovalRuleTemplateFromRepository

Action(s): `codecommit:DisassociateApprovalRuleTemplateFromRepository`

Required to disassociate the specified template from a repository in an Amazon Web Services account. It does not remove approval rules on pull requests already created with the template.

Resource: *

GetApprovalRuleTemplate

Action(s): `codecommit:GetApprovalRuleTemplate`

Required to return information about an approval rule template in an Amazon Web Services account.

Resource: *

ListApprovalRuleTemplates

Action(s): `codecommit:ListApprovalRuleTemplates`

Required to list approval rule templates in an Amazon Web Services account.

Resource: *

ListAssociatedApprovalRuleTemplatesForRepository

Action(s): `codecommit:ListAssociatedApprovalRuleTemplatesForRepository`

Required to list all approval rule templates that are associated with a specified repository in an Amazon Web Services account.

Resource: *

ListRepositoriesForApprovalRuleTemplate

Action(s): `codecommit:ListRepositoriesForApprovalRuleTemplate`

Required to list all repositories that are associated with a specified approval rule template in an Amazon Web Services account.

Resource: *

UpdateApprovalRuleTemplateContent

Action(s): `codecommit:UpdateApprovalRuleTemplateContent`

Required to update the content of an approval rule template in an Amazon Web Services account.

Resource: *

UpdateApprovalRuleTemplateDescription

Action(s): `codecommit:UpdateApprovalRuleTemplateDescription`

Required to update the description of an approval rule template in an Amazon Web Services account.

Resource: *

UpdateApprovalRuleTemplateName

Action(s): `codecommit:UpdateApprovalRuleTemplateName`

Required to update the name of an approval rule template in an AWS account.

Resource: *

Permissions for actions on individual files

The following permissions allow or deny actions on individual files in CodeCommit repositories. These permissions pertain only to actions performed in the CodeCommit console, the CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For example, the `git push` command pushes new and changed files to a CodeCommit repository by using the Git protocol. It's not affected by any permissions for the CodeCommit `PutFile` operation.

CodeCommit API Operations and Required Permissions for Actions on Individual Files

DeleteFile

Action(s): `codecommit:DeleteFile`

Required to delete a specified file from a specified branch in a CodeCommit repository from the CodeCommit console.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetBlob

Action(s): `codecommit:GetBlob`

Required to view the encoded content of an individual file in a CodeCommit repository from the CodeCommit console.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetBlobDifferences

Action(s): `codecommit:GetBlobDifferences`

Required to view a structured, line-level diff between two blob versions of a file in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetFile

Action(s): `codecommit:GetFile`

Required to view the encoded content of a specified file and its metadata in a CodeCommit repository from the CodeCommit console.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetFolder

Action(s): `codecommit:GetFolder`

Required to view the contents of a specified folder in a CodeCommit repository from the CodeCommit console.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

PutFile

Action(s): `codecommit:PutFile`

Required to add a new or modified file to a CodeCommit repository from the CodeCommit console, CodeCommit API, or the AWS CLI.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

Permissions for actions on comments

The following permissions allow or deny actions on comments in CodeCommit repositories. These permissions pertain to actions performed with the CodeCommit console and the CodeCommit API, and to commands performed using the AWS CLI. For related permissions on comments in pull requests, see Permissions for actions on pull requests.

CodeCommit API Operations and Required Permissions for Actions on Repositories

DeleteCommentContent

Action(s): `codecommit:DeleteCommentContent`

Required to delete the content of a comment made on a change, file, or commit in a repository. Comments cannot be deleted, but the content of a comment can be removed if the user has this permission.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetComment

Action(s): `codecommit:GetComment`

Required to return information about a comment made on a change, file, or commit in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetCommentReactions

Action(s): `codecommit:GetCommentReactions`

Required to return information about emoji reactions to a comment made on a change, file, or commit in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

GetCommentsForComparedCommit

Action(s): `codecommit:GetCommentsForComparedCommit`

Required to return information about comments made on the comparison between two commits in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

PostCommentForComparedCommit

Action(s): `codecommit:PostCommentForComparedCommit`

Required to comment on the comparison between two commits in a CodeCommit repository.

Resource: `arn:aws:codecommit:region:account-id:repository-name`

PostCommentReply

Action(s): `codecommit:PostCommentReply`

Required to create a reply to a comment on a comparison between commits or on a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

PutCommentReaction

Action(s): `codecommit:PutCommentReaction`

Required to reply to a comment with an emoji on a commit or on a pull request in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

UpdateComment

Action(s): `codecommit:UpdateComment`

Required to edit a comment on a comparison between commits or on a pull request. Comments can only be edited by the comment author.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on committed code

The following permissions allow or deny actions on code committed to CodeCommit repositories. These permissions pertain to actions performed with the CodeCommit console and the CodeCommit API, and commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For example, the **git commit** command creates a commit for a branch in a repository using the Git protocol. It's not affected by any permissions for the CodeCommit `CreateCommit` operation.

Explicitly denying some of these permissions might result in unexpected consequences in the CodeCommit console. For example, setting `GetTree` to `Deny` prevents users from navigating the contents of a repository in the console, but does not block users from viewing the contents of a file in the repository (if they are sent a link to the file in email, for example). Setting `GetBlob` to `Deny` prevents users from viewing the contents of files, but does not block users from browsing

the structure of a repository. Setting `GetCommit` to `Deny` prevents users from retrieving details about commits. Setting `GetObjectIdentifier` to `Deny` blocks most of the functionality of code browsing. If you set all three of these actions to `Deny` in a policy, a user with that policy cannot browse code in the CodeCommit console.

CodeCommit API Operations and Required Permissions for Actions on Committed Code

BatchGetCommits

Action(s): `codecommit:BatchGetCommits`

Required to return information about one or more commits in a CodeCommit repository. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CreateCommit

Action(s): `codecommit:CreateCommit`

Required to create a commit.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetCommit

Action(s): `codecommit:GetCommit`

Required to return information about a commit.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetCommitHistory

Action(s): `codecommit:GetCommitHistory`

Required to return information about the history of commits in a repository. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetDifferences

Action(s): `codecommit:GetDifferences`

Required to return information about the differences in a commit specifier (such as a branch, tag, HEAD, commit ID, or other fully qualified reference).

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetObjectIdentifier

Action(s): `codecommit:GetObjectIdentifier`

Required to resolve blobs, trees, and commits to their identifier. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetReferences

Action(s): `codecommit:GetReferences`

Required to return all references, such as branches and tags. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetTree

Action(s): `codecommit:GetTree`

Required to view the contents of a specified tree in a CodeCommit repository from the CodeCommit console. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on repositories

The following permissions allow or deny actions on CodeCommit repositories. These permissions pertain to actions performed with the CodeCommit console and the CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol.

CodeCommit API Operations and Required Permissions for Actions on Repositories

BatchGetRepositories

Action(s): `codecommit:BatchGetRepositories`

Required to get information about multiple CodeCommit repositories in that are in an Amazon Web Services account. In `Resource`, you must specify the names of all of the CodeCommit repositories for which a user is allowed (or denied) information.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CreateRepository

Action(s): `codecommit:CreateRepository`

Required to create a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

DeleteRepository

Action(s): `codecommit:DeleteRepository`

Required to delete a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetRepository

Action(s): `codecommit:GetRepository`

Required to get information about a single CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

ListRepositories

Action(s): `codecommit:ListRepositories`

Required to get a list of the names and system IDs of multiple CodeCommit repositories for an Amazon Web Services account. The only allowed value for `Resource` for this action is all repositories (*).

Resource: *

UpdateRepositoryDescription

Action(s): `codecommit:UpdateRepositoryDescription`

Required to change the description of a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

UpdateRepositoryName

Action(s): `codecommit:UpdateRepositoryName`

Required to change the name of a CodeCommit repository. In `Resource`, you must specify both the CodeCommit repositories that are allowed to be changed and the new repository names.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on tags

The following permissions allow or deny actions on AWS tags for CodeCommit resources.

CodeCommit API Operations and Required Permissions for Actions on Tags

ListTagsForResource

Action(s): `codecommit:ListTagsForResource`

Required to return information about AWS tags configured on a resource in CodeCommit.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

TagResource

Action(s): `codecommit:TagResource`

Required to add or edit AWS tags for a repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

UntagResource

Action(s): `codecommit:UntagResource`

Required to remove AWS tags from a resource in CodeCommit.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on triggers

The following permissions allow or deny actions on triggers for CodeCommit repositories.

CodeCommit API Operations and Required Permissions for Actions on Triggers

GetRepositoryTriggers

Action(s): `codecommit:GetRepositoryTriggers`

Required to return information about triggers configured for a repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

PutRepositoryTriggers

Action(s): `codecommit:PutRepositoryTriggers`

Required to create, edit, or delete triggers for a repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

TestRepositoryTriggers

Action(s): `codecommit:TestRepositoryTriggers`

Required to test the functionality of a repository trigger by sending data to the topic or function configured for the trigger.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

Permissions for actions on CodePipeline integration

In order for CodePipeline to use a CodeCommit repository in a source action for a pipeline, you must grant all of the permissions listed in the following table to the service role for CodePipeline. If these permissions are not set in the service role or are set to **Deny**, the pipeline does not run automatically when a change is made to the repository, and changes cannot be released manually.

CodeCommit API Operations and Required Permissions for Actions on CodePipeline Integration

GetBranch

Action(s): `codecommit:GetBranch`

Required to get details about a branch in a CodeCommit repository.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetCommit

Action(s): `codecommit:GetCommit`

Required to return information about a commit.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

UploadArchive

Action(s): `codecommit:UploadArchive`

Required to allow the service role for CodePipeline to upload repository changes into a pipeline. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

GetUploadArchiveStatus

Action(s): `codecommit:GetUploadArchiveStatus`

Required to determine the status of an archive upload: whether it is in progress, complete, cancelled, or if an error occurred. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

CancelUploadArchive

Action(s): `codecommit:CancelUploadArchive`

Required to cancel the uploading of an archive to a pipeline. This is an IAM policy permission only, not an API action that you can call.

Resource: `arn:aws:codecommit:`*`region`*`:`*`account-id`*`:`*`repository-name`*

How AWS CodeCommit works with IAM

Before you use IAM to manage access to CodeCommit, you should understand what IAM features are available to use with CodeCommit. To get a high-level view of how CodeCommit and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

- Condition keys
- Examples

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

CodeCommit defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Some CodeCommit actions support the `codecommit:References` condition key. For an example policy that uses this key, see Example 4: Deny or allow actions on branches.

To see a list of CodeCommit condition keys, see Condition Keys for AWS CodeCommit in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS CodeCommit.

Examples

To view examples of CodeCommit identity-based policies, see AWS CodeCommit identity-based policy examples.

CodeCommit resource-based policies

CodeCommit does not support resource-based policies.

Authorization based on CodeCommit tags

You can attach tags to CodeCommit resources or pass tags in a request to CodeCommit. To control access based on tags, you provide tag information in the condition element of a policy using the `codecommit:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging CodeCommit resources, see Example 5: Deny or allow actions on repositories with tags. For more information about tagging strategies, see Tagging AWS Resources.

CodeCommit also supports policies based on session tags. For more information, see Session Tags.

Using tags to provide identity information in CodeCommit

CodeCommit supports the use of session tags, which are key-value pair attributes that you pass when you assume an IAM role, use temporary credentials, or federate a user in AWS Security Token Service (AWS STS). You can also associate tags with an IAM user. You can use the information provided in these tags to make it easier to identify who made a change or caused an event. CodeCommit includes the values for tags with the following key names in CodeCommit events:

Key name	Value
`displayName`	The human-readable name to display and associate with the user (for example, Mary Major or Saanvi Sarkar).
`emailAddress`	The email address you want displayed for and associated with the user (for example, mary_major@example.com or saanvi_sarkar@example.com).

If this information is provided, CodeCommit includes it in events sent to Amazon EventBridge and Amazon CloudWatch Events. For more information, see Monitoring CodeCommit events in Amazon EventBridge and Amazon CloudWatch Events.

To use session tagging, roles must have policies that include the `sts:TagSession` permission set to `Allow`. If you are using federated access, you can configure display name and email tag information as part of your setup. For example, if you're using Azure Active Directory, you might provide the following claim information:

Claim name	Value
`https://aws.amazon.com/SAML/Attributes/PrincipalTag:displayNam e`	`user.displayname`
`https://aws.amazon.com/SAML/Attributes/PrincipalTag:emailAddress`	`user.mail`

You can use the AWS CLI to pass session tags for `displayName` and `emailAddress` using **AssumeRole**. For example, a user who wants to assume a role named *`Developer`* who wants to associate her name *`Mary Major`* might use the **assume-role** command similar to the following:

```
aws sts assume-role \
--role-arn arn:aws:iam::123456789012:role/Developer \
--role-session-name Mary-Major \
--tags Key=displayName,Value="Mary Major"
 Key=emailAddress,Value="mary_major@example.com" \
--external-id Example987
```

For more information, see AssumeRole.

You can use the `AssumeRoleWithSAML` operation to return a set of temporary credentials that include `displayName` and `emailAddress` tags. You can use these tags when you access CodeCommit repositories. This requires that your company or group has already integrated your third-party SAML solution with AWS. If so, you can pass SAML attributes as session tags. For example, if you wanted to pass identity attributes for a display name and email address for a user named *`Saanvi Sarkar`* as session tags:

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:displayName">
  <AttributeValue>Saanvi Sarkar</AttributeValue>
</Attribute>
<Attribute Name="https://aws.amazon.com/SAML/Attributes/PrincipalTag:emailAddress">
  <AttributeValue>saanvi_sarkar@example.com</AttributeValue>
</Attribute>
```

For more information, see Passing Session Tags using AssumeRoleWithSAML.

You can use the `AssumeRoleWithIdentity` operation to return a set of temporary credentials that include `displayName` and `emailAddress` tags. You can use these tags when you access CodeCommit repositories. To pass session tags from OpenID Connect (OIDC), you must include the session tags in the JSON Web Token (JWT). For example, the decoded JWP token used to call `AssumeRoleWithWebIdentity` that includes the `displayName` and `emailAddress` session tags for a user named *`Li Juan`*:

```
{
    "sub": "lijuan",
    "aud": "ac_oic_client",
    "jti": "ZYUCeREXAMPLE",
```

```
    "iss": "https://xyz.com",
    "iat": 1566583294,
    "exp": 1566583354,
    "auth_time": 1566583292,
    "https://aws.amazon.com/tags": {
        "principal_tags": {
            "displayName": ["Li Juan"],
            "emailAddress": ["li_juan@example.com"],
        },
        "transitive_tag_keys": [
            "displayName",
            "emailAddress"
        ]
    }
}
```

For more information, see Passing Session Tags using AssumeRoleWithWebIdentity.

You can use the `GetFederationToken` operation to return a set of temporary credentials that include `displayName` and `emailAddress` tags. You can use these tags when you access CodeCommit repositories. For example, to use the AWS CLI to get a federation token that includes the `displayName` and `emailAddress` tags:

```
aws sts get-federation-token \
--name my-federated-user \
--tags key=displayName,value="Nikhil Jayashankar"
 key=emailAddress,value=nikhil_jayashankar@example.com
```

For more information, see Passing Session Tags using GetFederationToken.

CodeCommit IAM roles

An IAM role is an entity within your Amazon Web Services account that has specific permissions.

Using temporary credentials with CodeCommit

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

CodeCommit supports using temporary credentials. For more information, see Connecting to AWS CodeCommit repositories with rotating credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

CodeCommit does not use service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

CodeCommit does not use service roles.

AWS CodeCommit identity-based policy examples

By default, IAM users and roles don't have permission to create or modify CodeCommit resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

For examples of policies, see the following:

- Example 1: Allow a user to perform CodeCommit operations in a single AWS Region
- Example 2: Allow a user to use Git for a single repository
- Example 3: Allow a user connecting from a specified IP address range access to a repository
- Example 4: Deny or allow actions on branches
- Example 5: Deny or allow actions on repositories with tags
- Configure cross-account access to an AWS CodeCommit repository using roles

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

- Policy best practices

- Using the CodeCommit console
- Allow users to view their own permissions
- Viewing CodeCommit repositories based on tags

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CodeCommit resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the CodeCommit console

To access the AWS CodeCommit console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CodeCommit resources in your Amazon Web Services account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the CodeCommit console, also attach the following AWS managed policy to the entities. For more information, see Adding Permissions to a User in the *IAM User Guide*:

For more information, see Using identity-based policies (IAM Policies) for CodeCommit.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws-eusc:iam::*:user/${aws:username}"]
```

```
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Viewing CodeCommit *repositories* based on tags

You can use conditions in your identity-based policy to control access to CodeCommit resources based on tags. For an example policy that demonstrates how to do this, see Example 5: Deny or allow actions on repositories with tags.

For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Troubleshooting AWS CodeCommit identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CodeCommit and IAM.

Topics

- I Am not authorized to perform an action in CodeCommit
- I Am not authorized to perform iam:PassRole
- I want to view my access keys
- I'm an administrator and want to allow others to access CodeCommit
- I want to allow people outside of my Amazon Web Services account to access my CodeCommit resources

I Am not authorized to perform an action in CodeCommit

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

For more information, see Permissions required to use the CodeCommit console

I Am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CodeCommit.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CodeCommit. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-eusc:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
>
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your AWS account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access CodeCommit

To allow others to access CodeCommit, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in CodeCommit. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM in the *IAM User Guide*.

I want to allow people outside of my Amazon Web Services account to access my CodeCommit resources

For more information, see Configure cross-account access to an AWS CodeCommit repository using roles.

Resilience in AWS CodeCommit

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you

can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

A CodeCommit repository or CodeCommit approval rule template exists in the AWS Region where it was created. For more information, see Regions and Git connection endpoints for AWS CodeCommit. For resiliency in repositories, you can configure your Git client to push to two repositories at once. For more information, see Push commits to an additional Git repository.

Infrastructure security in AWS CodeCommit

As a managed service, AWS CodeCommit is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access CodeCommit through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but CodeCommit does support restrictions based on the source IP address. You can also use CodeCommit policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given CodeCommit resource from only the specific VPC in the AWS network.

For more information, see the following:

- Example 1: Allow a user to perform CodeCommit operations in a single AWS Region
- Example 3: Allow a user connecting from a specified IP address range access to a repository
- Using AWS CodeCommit with interface VPC endpoints

Monitoring AWS CodeCommit

Monitoring is an important part of maintaining the reliability, availability, and performance of CodeCommit and your other AWS solutions. AWS provides the following monitoring tools to watch CodeCommit, report when something is wrong, and take automatic actions when appropriate:

- Amazon EventBridge can be used to automate your AWS services and respond automatically to system events, such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and which automated actions to take when an event matches a rule. For more information, see Amazon EventBridge User Guide and Monitoring CodeCommit events in Amazon EventBridge and Amazon CloudWatch Events.
- Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources. CloudWatch Events enables automated event-driven computing, as you can write rules that watch for certain events and trigger automated actions in other AWS services when these events happen. For more information, see the Amazon CloudWatch Events User Guide and Monitoring CodeCommit events in Amazon EventBridge and Amazon CloudWatch Events.
- Amazon CloudWatch Logs can be used to monitor, store, and access your log files from CloudTrail and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- AWS CloudTrail captures API calls and related events made by or on behalf of your Amazon Web Services account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide and Logging AWS CodeCommit API calls with AWS CloudTrail.

Monitoring CodeCommit events in Amazon EventBridge and Amazon CloudWatch Events

You can monitor AWS CodeCommit events in EventBridge, which delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services. EventBridge routes that data to targets such as AWS Lambda and Amazon Simple Notification

Service. These events are the same as those that appear in Amazon CloudWatch Events, which delivers a near real-time stream of system events that describe changes in AWS resources.

The following examples show events for CodeCommit.

> **Note**
>
> CodeCommit supports providing `displayName` and `emailAddress` information included in session tags in events, if that information is available. For more information, see Session Tags and Using tags to provide identity information in CodeCommit.

Topics

- referenceCreated event
- referenceUpdated event
- referenceDeleted event
- unreferencedMergeCommitCreated event
- commentOnCommitCreated event
- commentOnCommitUpdated event
- commentOnPullRequestCreated event
- commentOnPullRequestUpdated event
- pullRequestCreated event
- pullRequestSourceBranchUpdated event
- pullRequestStatusChanged event
- pullRequestMergeStatusUpdated event
- approvalRuleTemplateCreated event
- approvalRuleTemplateUpdated event
- approvalRuleTemplateDeleted event
- approvalRuleTemplateAssociatedWithRepository event
- approvalRuleTemplateDisassociatedWithRepository event
- approvalRuleTemplateBatchAssociatedWithRepositories event
- approvalRuleTemplateBatchDisassociatedFromRepositories event

- pullRequestApprovalRuleCreated event
- pullRequestApprovalRuleDeleted event
- pullRequestApprovalRuleOverridden event
- pullRequestApprovalStateChanged event
- pullRequestApprovalRuleUpdated event
- reactionCreated event
- reactionUpdated event

referenceCreated event

In this example event, a branch named `myBranch` has been created in a repository named `MyDemoRepo`.

```
{
   "version": "0",
   "id": "01234567-EXAMPLE",
   "detail-type": "CodeCommit Repository State Change",
   "source": "aws.codecommit",
   "account": "123456789012",
   "time": "2019-06-12T10:23:43Z",
   "region": "us-east-2",
   "resources": [
     "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
   ],
   "detail": {
     "event": "referenceCreated",
     "repositoryName": "MyDemoRepo",
     "repositoryId": "12345678-1234-5678-abcd-12345678abcd",
     "referenceType": "branch",
     "referenceName": "myBranch",
     "referenceFullName": "refs/heads/myBranch",
     "commitId": "3e5983DESTINATION"
   }
 }
```

referenceUpdated event

In this example event, a branch named `myBranch` has been updated by a merge in a repository named `MyDemoRepo`.

```
{
   "version": "0",
   "id": "01234567-EXAMPLE",
   "detail-type": "CodeCommit Repository State Change",
   "source": "aws.codecommit",
   "account": "123456789012",
   "time": "2019-06-12T10:23:43Z",
   "region": "us-east-2",
   "resources": [
     "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
   ],
   "detail": {
     "event": "referenceUpdated",
     "repositoryName": "MyDemoRepo",
     "repositoryId": "12345678-1234-5678-abcd-12345678abcd",
     "referenceType": "branch",
     "referenceName": "myBranch",
     "referenceFullName": "refs/heads/myBranch",
     "commitId": "7f0103fMERGE",
     "oldCommitId": "3e5983DESTINATION",
     "baseCommitId": "3e5a9bf1BASE",
     "sourceCommitId": "26a8f2SOURCE",
     "destinationCommitId": "3e5983DESTINATION",
     "mergeOption": "THREE_WAY_MERGE",
     "conflictDetailsLevel": "LINE_LEVEL",
     "conflictResolutionStrategy": "AUTOMERGE"
   }
}
```

referenceDeleted event

In this example event, a branch named `myBranch` has been deleted in a repository named `MyDemoRepo`.

```
{
  "version": "0",
  "id": "01234567-EXAMPLE",
  "detail-type": "CodeCommit Repository State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-06-12T10:23:43Z",
  "region": "us-east-2",
```

```
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "event": "referenceDeleted",
    "repositoryName": "MyDemoRepo",
    "repositoryId": "12345678-1234-5678-abcd-12345678abcd",
    "referenceType": "branch",
    "referenceName": "myBranch",
    "referenceFullName": "refs/heads/myBranch",
    "oldCommitId": "26a8f2EXAMPLE"
  }
}
```

unreferencedMergeCommitCreated event

In this example event, an unreferenced merge commit has been created in a repository named MyDemoRepo.

```
{
  "version": "0",
  "id": "01234567-EXAMPLE",
  "detail-type": "CodeCommit Repository State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-06-12T10:23:43Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "event": "unreferencedMergeCommitCreated",
    "repositoryName": "MyDemoRepo",
    "repositoryId": "12345678-1234-5678-abcd-12345678abcd",
    "commitId": "7f0103fMERGE",
    "baseCommitId": "3e5a9bf1BASE",
    "sourceCommitId": "26a8f2SOURCE",
    "destinationCommitId": "3e5983DESTINATION",
    "mergeOption": "SQUASH_MERGE",
    "conflictDetailsLevel": "LINE_LEVEL",
    "conflictResolutionStrategy": "AUTOMERGE"
  }
```

```
}
```

commentOnCommitCreated event

In this example event, a federated user named `Mary_Major` commented on a commit. In this example, their federated identity provider configured session tags for `displayName` and `emailAddress`. That information is included in the event.

```
{
  "version": "0",
  "id": "e9dce2e9-EXAMPLE",
  "detail-type": "CodeCommit Comment on Commit",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-09-29T20:20:39Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "beforeCommitId": "3c5dEXAMPLE",
    "repositoryId": "7dd1EXAMPLE...",
    "inReplyTo": "695bEXAMPLE...",
    "notificationBody": "A comment event occurred in the following repository:
 MyDemoRepo. The display name for the user is Mary Major. The email address for
 the user is mary_major@example.com. The user arn:aws:sts::123456789012:federated-
user/Mary_Major made a comment. The comment was made on the following comment ID:
 463bEXAMPLE.... For more information, go to the AWS CodeCommit console at https://us-
east-2.console.aws.amazon.com/codecommit/home?region=us-east-2#/repository/MyDemoRepo/
compare/3c5dEXAMPLE...f4d5EXAMPLE#463bEXAMPLE....",
    "commentId": "463bEXAMPLE...",
    "afterCommitId": "f4d5EXAMPLE",
    "event": "commentOnCommitCreated",
    "repositoryName": "MyDemoRepo",
    "callerUserArn": "arn:aws:sts::123456789012:federated-user/Mary_Major",
    "displayName": "Mary Major",
    "emailAddress": "mary_major@example.com"
  }
}
```

commentOnCommitUpdated event

In this example event, a user who assumed a role named `Admin` with a session name of `Mary_Major` edited a comment on a commit. In this example, the role included configured session tags for `displayName` and `emailAddress`. That information is included in the event.


```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Comment on Commit",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "afterCommitId": "53812581",
    "beforeCommitId": "03314446",
    "callerUserArn": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "commentId": "a7e5471e-EXAMPLE",
    "event": "commentOnCommitUpdated",
    "inReplyTo": "bdb07d47-EXAMPLE",
    "notificationBody": "A comment event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. The display name for the user is Mary
 Major. The email address for the user is mary_major@example.com. The user
 arn:aws:sts::123456789012:federated-user/Mary_Major updated a comment or
 replied to a comment. The comment was made on the following comment ID:
 bdb07d47-6fe9-47b0-a839-b93cc743b2ac:468cd1cb-2dfb-4f68-9636-8de52431d1d6.
 For more information, go to the AWS CodeCommit console https://us-
east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/
compare/033144464617842958996982309670958225176 8/.../53812581502937833614716802771360172913 82?
region\u003dus-east-2",
    "repositoryId": "12345678-1234-1234-1234-123456789012",
    "repositoryName": "MyDemoRepo",
    "displayName": "Mary Major",
    "emailAddress": "mary_major@example.com"
  }
}
```

commentOnPullRequestCreated event

In this example event, a federated user named `Saanvi_Sarkar` commented on a pull request. In this example, their federated identity provider configured session tags for `displayName` and `emailAddress`. That information is included in the event.

```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Comment on Pull Request",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "beforeCommitId": "3c5dEXAMPLE",
    "repositoryId": "7dd1EXAMPLE...",
    "inReplyTo": "695bEXAMPLE...",
    "notificationBody": "A comment event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. The display name for the user is Saanvi
 Sarkar. The email address for the user is saanvi_sarkar@example.com. The user
 arn:aws:sts::123456789012:federated-user/Saanvi_Sarkar made a comment. The comment
 was made on the following Pull Request: 201. For more information, go to the AWS
 CodeCommit console https://us-east-2.console.aws.amazon.com/codecommit/home?region=us-
east-2#/repository/MyDemoRepo/pull-request/201/activity#3276EXAMPLE...",
    "commentId": "463bEXAMPLE...",
    "afterCommitId": "f4d5EXAMPLE",
    "event": "commentOnPullRequestCreated",
    "repositoryName": "MyDemoRepo",
    "callerUserArn": "arn:aws:sts::123456789012:federated-user/Saanvi_Sarkar",
    "pullRequestId": "201",
    "displayName": "Saanvi Sarkar",
    "emailAddress": "saanvi_sarkar@example.com"
  }
}
```

commentOnPullRequestUpdated event

In this example event, a federated user named `Saanvi_Sarkar` edited a comment on a pull request. In this example, their federated identity provider configured session tags for `displayName` and `emailAddress`. That information is included in the event.


```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Comment on Pull Request",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "afterCommitId": "96814774EXAMPLE",
    "beforeCommitId": "6031971EXAMPLE",
    "callerUserArn": "arn:aws:sts::123456789012:federated-user/Saanvi_Sarkar",
    "commentId": "40cb52f0-EXAMPLE",
    "event": "commentOnPullRequestUpdated",
    "inReplyTo": "1285e713-EXAMPLE",
    "notificationBody": "A comment event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. The display name for the user is Saanvi
 Sarkar. The email address for the user is saanvi_sarkar@example.com. The user
 arn:aws:sts::123456789012:federated-user/Saanvi_Sarkar updated a comment or
 replied to a comment. The comment was made on the following Pull Request:
 1. For more information, go to the AWS CodeCommit console https://us-
east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/1/activity#40cb52f0-aac7-4c43-b771-601eff02EXAMPLE",
    "pullRequestId": "1",
    "repositoryId": "12345678-1234-1234-1234-123456789012",
    "repositoryName": "MyDemoRepo"
  }
}
```

pullRequestCreated event

In this example event, a pull request was created in a repository named `MyDemoRepo` by a user who assumed a role named `Admin` with a session name of `Mary_Major`. No session tag information was provided, so that information is not included in the event.


```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Pull Request State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "author": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "callerUserArn": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "creationDate": "Tue Feb 9 2019 10:18:42 PDT ",
    "description": "An example description.",
    "destinationCommit": "12241970EXAMPLE",
    "destinationReference": "refs/heads/main",
    "event": "pullRequestCreated",
    "isMerged": "False",
    "lastModifiedDate": "Tue Feb 9 2019 10:18:42 PDT",
    "notificationBody": "A pull request event occurred in the following AWS CodeCommit
 repository: MyDemoRepo. User: arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major.
 Event: Created. The pull request was created with the following information: Pull
 Request ID as 1 and title as My Example Pull Request. For more information, go to the
 AWS CodeCommit console https://us-east-2.console.aws.amazon.com/codesuite/codecommit/
repositories/MyDemoRepo/pull-requests/1",
    "pullRequestId": "1",
    "pullRequestStatus": "Open",
    "repositoryNames": ["MyDemoRepo"],
    "revisionId": "bdc0cb9bEXAMPLE",
    "sourceCommit": "2774290EXAMPLE",
    "sourceReference": "refs/heads/test-branch",
    "title": "My Example Pull Request"
  }
}
```

pullRequestSourceBranchUpdated event

In this example event, a user who assumed a role named `Admin` with a session name of `Mary_Major` updated the source branch named `test-branch` for a pull request with the ID of 1.


```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Pull Request State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "author": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "callerUserArn": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "creationDate": "Tue Feb 9 2019 10:18:42 PDT",
    "description": "An example description.",
    "destinationCommit": "7644990EXAMPLE",
    "destinationReference": "refs/heads/main",
    "event": "pullRequestSourceBranchUpdated",
    "isMerged": "False",
    "lastModifiedDate": "Tue Feb 9 2019 10:18:42 PDT",
    "notificationBody": "A pull request event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. User: arn:aws:sts::123456789012:assumed-role/
Admin/Mary_Major. Event: Updated.  The user updated the following pull request:
 1. The pull request was updated with one or more commits to the source branch:
 test-branch. For more information, go to the AWS CodeCommit console https://us-
east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/1?region\u003dus-east-2",
    "pullRequestId": "1",
    "pullRequestStatus": "Open",
    "repositoryNames": ["MyDemoRepo"],
    "revisionId": "bdc0cb9b4EXAMPLE",
    "sourceCommit": "64875001EXAMPLE",
    "sourceReference": "refs/heads/test-branch",
    "title": "My Example Pull Request"
  }
}
```

pullRequestStatusChanged event

In this example event, a user who assumed a role named `Admin` with a session name of `Mary_Major` closed a pull request with the ID of 1. The pull request was not merged.


```
{
  "version": "0",
  "id": "98377d67-EXAMPLE",
  "detail-type": "CodeCommit Pull Request State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-02-09T07:15:16Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "author": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "callerUserArn": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "creationDate": "Tue Jun 18 10:34:20 PDT 2019",
    "description": "An example description.",
    "destinationCommit": "95149731EXAMPLE",
    "destinationReference": "refs/heads/main",
    "event": "pullRequestStatusChanged",
    "isMerged": "False",
    "lastModifiedDate": "Tue Jun 18 10:34:20 PDT 2019",
    "notificationBody": "A pull request event occurred in the following AWS CodeCommit
 repository: MyDemoRepo. arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major
 updated the following PullRequest 1. The pull request status has been updated. The
 status is closed. For more information, go to the AWS CodeCommit console https://
us-east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/1?region\u003dus-east-2",
    "pullRequestId": "1",
    "pullRequestStatus": "Closed",
    "repositoryNames": ["MyDemoRepo"],
    "revisionId": "bdc0cb9bEXAMPLE",
    "sourceCommit": "4409936EXAMPLE",
    "sourceReference": "refs/heads/test-branch",
    "title": "My Example Pull Request"
  }
}
```

pullRequestMergeStatusUpdated event

In this example event, a user who assumed a role named `Admin` with a session name of `Mary_Major` merged a pull request with the ID of 1.


```
{
  "version": "0",
  "id": "01234567-0123-0123-0123-012345678901",
  "detail-type": "CodeCommit Pull Request State Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-06-12T10:23:43Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
  ],
  "detail": {
    "author": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "callerUserArn": "arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major",
    "creationDate": "Mon Mar 11 14:42:31 PDT 2019",
    "description": "An example description.",
    "destinationCommit": "4376719EXAMPLE",
    "destinationReference": "refs/heads/main",
    "event": "pullRequestMergeStatusUpdated",
    "isMerged": "True",
    "lastModifiedDate": "Mon Mar 11 14:42:31 PDT 2019",
    "mergeOption": "FAST_FORWARD_MERGE",
    "notificationBody": "A pull request event occurred in the following AWS CodeCommit
 repository: MyDemoRepo. arn:aws:sts::123456789012:assumed-role/Admin/Mary_Major
 updated the following PullRequest 1. The pull request merge status has been updated.
 The status is merged. For more information, go to the AWS CodeCommit console https://
us-east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/1?region\u003dus-east-2",
    "pullRequestId": "1",
    "pullRequestStatus": "Closed",
    "repositoryNames": ["MyDemoRepo"],
    "revisionId": "bdc0cb9beEXAMPLE",
    "sourceCommit": "0701696EXAMPLE",
    "sourceReference": "refs/heads/test-branch",
    "title": "My Example Pull Request"
  }
}
```

approvalRuleTemplateCreated event

In this example event, a user with an IAM user name of `Mary_Major` created an approval rule template named `2-approvers-required-for-main`.

```
{
    "version": "0",
    "id": "f7702227-EXAMPLE",
    "detail-type": "CodeCommit Approval Rule Template Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:02:27Z",
    "region": "us-east-2",
    "resources": [],
    "detail": {
        "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
        "approvalRuleTemplateId": "d7385967-EXAMPLE",
        "approvalRuleTemplateName": "2-approvers-required-for-main",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:02:14 UTC 2019",
        "event": "approvalRuleTemplateCreated",
        "lastModifiedDate": "Wed Nov 06 19:02:14 UTC 2019",
        "notificationBody": "A approval rule template event occurred in the following
 AWS CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user/Mary_Major.
  Additional information: An approval rule template with the following name has been
 created: 2-approvers-required-for-main. The ID of the created template is: d7385967-
EXAMPLE. For more information, go to the AWS CodeCommit console.",
        "repositories": {}
    }
}
```

approvalRuleTemplateUpdated event

In this example event, a user with an IAM user name of `Mary_Major` edited an approval rule template named `2-approvers-required-for-main`. The approval rule template is not associated with any repositories.

```
{
  "version": "0",
  "id": "66403118-EXAMPLE",
  "detail-type": "CodeCommit Approval Rule Template Change",
```

```
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-11-12T23:03:30Z",
  "region": "us-east-2",
  "resources": [

  ],
  "detail": {
    "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
    "approvalRuleTemplateId": "c9d2b844-EXAMPLE",
    "approvalRuleTemplateName": "2-approvers-required-for-main",
    "callerUserArn": "arn:aws:iam::123456789012:user\Mary_Major",
    "creationDate": "Tue Nov 12 23:03:06 UTC 2019",
    "event": "approvalRuleTemplateDeleted",
    "lastModifiedDate": "Tue Nov 12 23:03:20 UTC 2019",
    "notificationBody": "A approval rule template event occurred in the following AWS
 CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user\Mary_Major.
 Additional information: An approval rule template with the following name has been
 deleted: 2-approvers-required-for-main. The ID of the updated template is: c9d2b844-
EXAMPLE. For more information, go to the AWS CodeCommit console.",
    "repositories": {}
  }
}
```

approvalRuleTemplateDeleted event

In this example event, a user with an IAM user name of `Mary_Major` deleted an approval rule template named `2-approvers-required-for-main`. The approval rule template is not associated with any repositories.

```
{
  "version": "0",
  "id": "66403118-EXAMPLE",
  "detail-type": "CodeCommit Approval Rule Template Change",
  "source": "aws.codecommit",
  "account": "123456789012",
  "time": "2019-11-12T23:03:30Z",
  "region": "us-east-2",
  "resources": [],
  "detail": {
    "approvalRuleTemplateContentSha256": "4f3de6632EXAMPLE",
    "approvalRuleTemplateId": "c9d2b844-EXAMPLE",
    "approvalRuleTemplateName": "2-approvers-required-for-main",
```

```
    "callerUserArn": "arn:aws:iam::123456789012:user\Mary_Major",
    "creationDate": "Tue Nov 12 23:03:06 UTC 2019",
    "event": "approvalRuleTemplateUpdated",
    "lastModifiedDate": "Tue Nov 12 23:03:20 UTC 2019",
    "notificationBody": "A approval rule template event occurred in the following AWS
 CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user\Mary_Major.
 Additional information: An approval rule template with the following name has
 been updated: 2-approvers-required-for-main. The ID of the updated template is:
 c9d2b844-EXAMPLE. The after rule template content SHA256 is 4f3de663EXAMPLE. For more
 information, go to the AWS CodeCommit console.",
    "repositories": {}
  }
}
```

approvalRuleTemplateAssociatedWithRepository event

In this example event, a user with an IAM user name of `Mary_Major` associated an approval rule template named `2-approvers-required-for-main` with a repository named `MyDemoRepo`.

```
{
    "version": "0",
    "id": "ea1c6d73-EXAMPLE",
    "detail-type": "CodeCommit Approval Rule Template Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:02:27Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
        "approvalRuleTemplateId": "d7385967-EXAMPLE",
        "approvalRuleTemplateName": "2-approvers-required-for-main",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:02:14 UTC 2019",
        "event": "approvalRuleTemplateAssociatedWithRepository",
        "lastModifiedDate": "Wed Nov 06 19:02:14 UTC 2019",
        "notificationBody": "A approval rule template event occurred in the following
 AWS CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user/Mary_Major.
 Additional information: An approval rule template has been associated with the
 following repository: [MyDemoRepo]. For more information, go to the AWS CodeCommit
 console.",
```

```
        "repositories": {
            "MyDemoRepo": "92ca7bf2-d878-49ed-a994-336a6cc7c574"
        }
    }
}
```

approvalRuleTemplateDisassociatedWithRepository event

In this example event, a user with an IAM user name of `Mary_Major` disassociated an approval rule template named `2-approvers-required-for-main` from a repository named `MyDemoRepo`.

```
{
    "version": "0",
    "id": "ea1c6d73-EXAMPLE",
    "detail-type": "CodeCommit Approval Rule Template Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:02:27Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
        "approvalRuleTemplateId": "d7385967-EXAMPLE",
        "approvalRuleTemplateName": "2-approvers-required-for-main",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:02:14 UTC 2019",
        "event": "approvalRuleTemplateDisassociatedFromRepository",
        "lastModifiedDate": "Wed Nov 06 19:02:14 UTC 2019",
        "notificationBody": "A approval rule template event occurred in the following
 AWS CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user/Mary_Major.
 Additional information: An approval rule template has been disassociated from the
 following repository: [MyDemoRepo]. For more information, go to the AWS CodeCommit
 console.",
        "repositories": {
            "MyDemoRepo": "92ca7bf2-d878-49ed-a994-336a6cc7c574"
        }
    }
}
```

approvalRuleTemplateBatchAssociatedWithRepositories event

In this example event, a user with an IAM user name of `Mary_Major` batch associated an approval rule template named `2-approvers-required-for-main` with a repository named `MyDemoRepo` and a repository named `MyTestRepo`.

```
{
    "version": "0",
    "id": "0f861e5b-EXAMPLE",
    "detail-type": "CodeCommit Approval Rule Template Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-12T23:39:09Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
        "approvalRuleTemplateId": "c71c1fe0-EXAMPLE",
        "approvalRuleTemplateName": "2-approvers-required-for-main",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Tue Nov 12 23:38:57 UTC 2019",
        "event": "batchAssociateApprovalRuleTemplateWithRepositories",
        "lastModifiedDate": "Tue Nov 12 23:38:57 UTC 2019",
        "notificationBody": "A approval rule template event occurred in the following
 AWS CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user\Mary_Major.
 Additional information: An approval rule template has been batch associated with the
 following repository names: [MyDemoRepo, MyTestRepo]. For more information, go to the
 AWS CodeCommit console.",
        "repositories": {
            "MyDemoRepo": "MyTestRepo"
        }
    }
}
```

approvalRuleTemplateBatchDisassociatedFromRepositories event

In this example event, a user with an IAM user name of `Mary_Major` batch disassociated an approval rule template named `2-approvers-required-for-main` from a repository named `MyDemoRepo` and a repository named `MyTestRepo`.

```
{
    "version": "0",
    "id": "e08fc996-EXAMPLE",
    "detail-type": "CodeCommit Approval Rule Template Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-12T23:39:09Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleTemplateContentSha256": "f742eebbEXAMPLE",
        "approvalRuleTemplateId": "c71c1fe0-ff91-4db4-9a45-a86a7b6c474f",
        "approvalRuleTemplateName": "2-approvers-required-for-main",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Tue Nov 12 23:38:57 UTC 2019",
        "event": "batchDisassociateApprovalRuleTemplateFromRepositories",
        "lastModifiedDate": "Tue Nov 12 23:38:57 UTC 2019",
        "notificationBody": "A approval rule template event occurred in the following
 AWS CodeCommit account: 123456789012. User: arn:aws:iam::123456789012:user/Mary_Major.
 Additional information: An approval rule template has been batch disassociated from
 the following repository names: [MyDemoRepo, MyTestRepo]. For more information, go to
 the AWS CodeCommit console.",
        "repositories": {
            "MyDemoRepo": "MyTestRepo"
        }
    }
}
```

pullRequestApprovalRuleCreated event

In this example event, a user with an IAM user name of `Mary_Major` created an approval rule named `1-approver-needed` for a pull request with the ID of 227.

```
{
    "version": "0",
    "id": "ad860f12-EXAMPLE",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
```

```
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleContentSha256": "f742eebbEXAMPLE",
        "approvalRuleId": "0a9b5dfc-EXAMPLE",
        "approvalRuleName": "1-approver-needed",
        "author": "arn:aws:iam::123456789012:user/Mary_Major",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalRuleCreated",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/Mary_Major.
 Event: Updated. Pull request: 227. Additional information: An approval rule has been
 created with the following name: 1-approver-needed. For more information, go to the
 AWS CodeCommit console https://us-east-2.console.aws.amazon.com/codesuite/codecommit/
repositories/MyDemoRepo/pull-requests/227?region=us-east-2",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
        ],
        "revisionId": "3b8cecab3EXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

pullRequestApprovalRuleDeleted event

In this example event, a user with an IAM user name of `Mary_Major` deleted an approval rule named `1-approver-needed` for a pull request with the ID of 227. An IAM user with the name `Saanvi_Sarkar` originally authored the approval rule.

```
{
    "version": "0",
```

```
    "id": "c1c3509d-EXAMPLE",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleContentSha256": "f742eebbEXAMPLE",
        "approvalRuleId": "0a9b5dfc-EXAMPLE",
        "approvalRuleName": "1-approver-needed",
        "author": "arn:aws:iam::123456789012:user/Saanvi_Sarkar",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalRuleDeleted",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/Mary_Major.
 Event: Created. Pull request: 227. Additional information: An approval rule has been
 deleted: 1-approver-needed was deleted. For more information, go to the AWS CodeCommit
 console https://us-east-2.console.aws.amazon.com/codesuite/codecommit/repositories/
MyDemoRepo/pull-requests/227?region=us-east-2",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
        ],
        "revisionId": "3b8cecabEXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

pullRequestApprovalRuleOverridden event

In this example event, the approval rule requirements for a pull request have been set aside (OVERRIDE) by a user with an IAM user name of `Mary_Major`. The pull request was authored by a user with an IAM user name of `Li_Juan`.

```
{
    "version": "0",
    "id": "52d2cb73-EXAMPLE",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "author": "arn:aws:iam::123456789012:user/Li_Juan",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalRuleOverridden",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/Mary_Major.
 Event: Updated. Pull request name: 227. Additional information: An override
 event has occurred for the approval rules for this pull request. Override status:
 OVERRIDE. For more information, go to the AWS CodeCommit console https://us-
east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/227?region=us-east-2",
        "overrideStatus": "OVERRIDE",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
        ],
        "revisionId": "3b8cecabEXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
```

```
        "title": "My example pull request"
    }
}
```

In this example event, the approval rule requirements for a pull request have been reinstated (REVOKE).

```
{
    "version": "0",
    "id": "2895482d-13eb-b783-270d-76588e6029fa",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "author": "arn:aws:iam::123456789012:user/Li_Juan",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalRuleOverridden",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following
 AWS CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/
Mary_Major. Event: Updated. Pull request name: 227. Additional information: An
 override event has occurred for the approval rules for this pull request. Override
 status: REVOKE. For more information, go to the AWS CodeCommit console https://
us-east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/227?region=us-east-2",
        "overrideStatus": "REVOKE",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
        ],
        "revisionId": "3b8cecabEXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
```

```
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

pullRequestApprovalStateChanged event

In this example event, a pull request has been approved by a user with an IAM user name of `Mary_Major`.

```
{
    "version": "0",
    "id": "53e5d7e9-986c-1ebf-9d8b-ebef5596da0e",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalStatus": "APPROVE",
        "author": "arn:aws:iam::123456789012:user/Li_Juan",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalStateChanged",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following
 AWS CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/
Mary_Major. Event: Updated. Pull request name: 227. Additional information:
 A user has changed their approval state for the pull request. State change:
 APPROVE. For more information, go to the AWS CodeCommit console https://us-
east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/227?region=us-east-2",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
```

```
        ],
        "revisionId": "3b8cecabEXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

In this example event, an approval for a pull request has been revoked by a user with an IAM user name of `Mary_Major`.

```
{
    "version": "0",
    "id": "25e183d7-d01a-4e07-2bd9-b2d56ebecc81",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalStatus": "REVOKE",
        "author": "arn:aws:iam::123456789012:user/Li_Juan",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalStateChanged",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following AWS
 CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/Mary_Major.
 Event: Updated. Pull request name: 227. Additional information: A user has changed
 their approval state for the pull request. State change: REVOKE. For more information,
 go to the AWS CodeCommit console https://us-east-2.console.aws.amazon.com/codesuite/
codecommit/repositories/MyDemoRepo/pull-requests/227?region=us-east-2",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
```

```
        ],
        "revisionId": "3b8cecabEXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

pullRequestApprovalRuleUpdated event

In this example event, an approval rule for a pull request has been edited by a user with an IAM user name of `Mary_Major`. This user also authored the pull request.

```
{
    "version": "0",
    "id": "21b1c819-2889-3528-1cb8-3861aacf9d42",
    "detail-type": "CodeCommit Pull Request State Change",
    "source": "aws.codecommit",
    "account": "123456789012",
    "time": "2019-11-06T19:12:19Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
    ],
    "detail": {
        "approvalRuleContentSha256": "f742eebbEXAMPLE",
        "approvalRuleId": "0a9b5dfc-EXAMPLE",
        "approvalRuleName": "1-approver-needed",
        "author": "arn:aws:iam::123456789012:user/Mary_Major",
        "callerUserArn": "arn:aws:iam::123456789012:user/Mary_Major",
        "creationDate": "Wed Nov 06 19:10:58 UTC 2019",
        "description": "An example description.",
        "destinationCommit": "194fdf00EXAMPLE",
        "destinationReference": "refs/heads/main",
        "event": "pullRequestApprovalRuleUpdated",
        "isMerged": "False",
        "lastModifiedDate": "Wed Nov 06 19:10:58 UTC 2019",
        "notificationBody": "A pull request event occurred in the following
 AWS CodeCommit repository: MyDemoRepo. User: arn:aws:iam::123456789012:user/
Mary_Major. Event: Updated. Pull request name: 227. The content of an approval
 rule has been updated for the pull request. The name of the updated rule is: 1-
approver-needed. For more information, go to the AWS CodeCommit console https://
```

```
us-east-2.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/pull-
requests/227?region=us-east-2",
        "pullRequestId": "227",
        "pullRequestStatus": "Open",
        "repositoryNames": [
            "MyDemoRepo"
        ],
        "revisionId": "3b8cecab3EXAMPLE",
        "sourceCommit": "29964a17EXAMPLE",
        "sourceReference": "refs/heads/test-branch",
        "title": "My example pull request"
    }
}
```

reactionCreated event

In this example event, a reaction to a comment has been added by a user with an IAM user name of `Mary_Major`.

```
{
   "version":"0",
   "id":"59fcccd8-217a-32ce-2b05-561ed68a1c42",
   "detail-type":"CodeCommit Comment Reaction Change",
   "source":"aws.codecommit",
   "account":"123456789012",
   "time":"2020-04-14T00:49:03Z",
   "region":"us-east-2",
   "resources":[
      "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
   ],
   "detail":{
      "callerUserArn":"arn:aws:iam::123456789012:user/Mary_Major",
      "commentId":"28930161-EXAMPLE",
      "event":"commentReactionCreated",
      "notificationBody":"A comment reaction event occurred in the following AWS
 CodeCommit Repository: MyDemoRepo. The user: arn:aws:iam::123456789012:user/Mary_Major
 made a comment reaction # to the comment with comment ID: 28930161-EXAMPLE",
      "reactionEmojis":["#"],
      "reactionShortcodes":[":thumbsdown:"],
      "reactionUnicodes":["U+1F44E"],
      "repositoryId":"12345678-1234-5678-abcd-12345678abcd",
      "repositoryName":"MyDemoRepo"
   }
```

```
}
```

reactionUpdated event

In this example event, a reaction to a comment has been updated by a user with an IAM user name of `Mary_Major`. Users can only update their own reactions.

```
{
   "version":"0",
   "id":"0844ed99-a53f-3bdb-6048-4de315516889",
   "detail-type":"CodeCommit Comment Reaction Change",
   "source":"aws.codecommit",
   "account":"123456789012",
   "time":"2020-04-22T23:19:42Z",
   "region":"us-east-2",
   "resources":[
      "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo"
   ],
   "detail":{
      "callerUserArn":"arn:aws:iam::123456789012:user/Mary_Major",
      "commentId":"28930161-EXAMPLE",
      "event":"commentReactionUpdated",
      "notificationBody":"A comment reaction event occurred in the following AWS
 CodeCommit Repository: MyDemoRepo. The user: arn:aws:iam::123456789012:user/Mary_Major
 updated a reaction :smile: to the comment with comment ID: 28930161-EXAMPLE",
      "reactionEmojis":[
         "#"
      ],
      "reactionShortcodes":[
         ":smile:"
      ],
      "reactionUnicodes":[
         "U+1F604"
      ],
      "repositoryId":"12345678-1234-5678-abcd-12345678abcd",
      "repositoryName":"MyDemoRepo"
   }
}
```

Logging AWS CodeCommit API calls with AWS CloudTrail

CodeCommit is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in CodeCommit. CloudTrail captures all API calls for CodeCommit as events, including calls from the CodeCommit console, your Git client, and from code calls to the CodeCommit APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for CodeCommit. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to CodeCommit, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

CodeCommit information in CloudTrail

CloudTrail is enabled on your Amazon Web Services account when you create the account. When activity occurs in CodeCommit, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your Amazon Web Services account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your Amazon Web Services account, including events for CodeCommit, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

When CloudTrail logging is enabled in your Amazon Web Services account, API calls made to CodeCommit actions are tracked in CloudTrail log files, where they are written with other AWS

service records. CloudTrail determines when to create and write to a new file based on a time period and file size.

All CodeCommit actions are logged by CloudTrail, including some (such as `GetObjectIdentifier`) that are not currently documented in the AWS CodeCommit API Reference but are instead referenced as access permissions and documented in CodeCommit permissions reference. For example, calls to the `ListRepositories` (in the AWS CLI, `aws codecommit list-repositories`), `CreateRepository` (`aws codecommit create-repository`) and `PutRepositoryTriggers` (`aws codecommit put-repository-triggers`) actions generate entries in the CloudTrail log files, as well as Git client calls to `GitPull` and `GitPush`. In addition, if you have a CodeCommit repository configured as a source for a pipeline in CodePipeline, you will see calls to CodeCommit access permission actions such as `UploadArchive` from CodePipeline. Since CodeCommit uses AWS Key Management Service to encrypt and decrypt repositories, you will also see calls from CodeCommit to `Encrypt` and `Decrypt` actions from AWS KMS in CloudTrail logs.

Every log entry contains information about who generated the request. The user identity information in the log entry helps you determine the following:

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user, or made by an assumed role
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element.

You can store your log files in your Amazon S3 bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

Understanding CodeCommit log file entries

CloudTrail log files can contain one or more log entries. Each entry lists multiple JSON-formatted events. A log event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. Log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

> **Note**
>
> This example has been formatted to improve readability. In a CloudTrail log file, all entries and events are concatenated into a single line. This example has also been limited to a single CodeCommit entry. In a real CloudTrail log file, you see entries and events from multiple AWS services.

Contents

- Example: A log entry for listing CodeCommit repositories
- Example: A log entry for creating a CodeCommit repository
- Examples: Log entries for Git pull calls to a CodeCommit repository
- Example: A log entry for a successful push to a CodeCommit repository

Example: A log entry for listing CodeCommit repositories

The following example shows a CloudTrail log entry that demonstrates the `ListRepositories` action.

> **Note**
>
> Although `ListRepositories` returns a list of repositories, non-mutable responses are not recorded in CloudTrail logs, so `responseElements` is shown as `null` in the log file.


```
{
  "eventVersion":"1.05",
  "userIdentity": {
    "type":"IAMUser",
    "principalId":"AIDACKCEVSQ6C2EXAMPLE",
    "arn":"arn:aws:iam::444455556666:user/Mary_Major",
    "accountId":"444455556666",
    "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
    "userName":"Mary_Major"
    },
  "eventTime":"2016-12-14T17:57:36Z",
  "eventSource":"codecommit.amazonaws.com",
  "eventName":"ListRepositories",
```

```
  "awsRegion":"us-east-1",
  "sourceIPAddress":"203.0.113.12",
  "userAgent":"aws-cli/1.10.53 Python/2.7.9 Windows/8 botocore/1.4.43",
  "requestParameters":null,
  "responseElements":null,
  "requestID":"cb8c167e-EXAMPLE",
  "eventID":"e3c6f4ce-EXAMPLE",
  "readOnly":true,
  "eventType":"AwsApiCall",
  "apiVersion":"2015-04-13",
  "recipientAccountId":"444455556666"
}
```

Example: A log entry for creating a CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates the `CreateRepository` action in the US East (Ohio) Region.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName":"Mary_Major"
  },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "CreateRepository",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "aws-cli/1.10.53 Python/2.7.9 Windows/8 botocore/1.4.43",
  "requestParameters": {
    "repositoryDescription": "Creating a demonstration repository.",
    "repositoryName": "MyDemoRepo"
  },
  "responseElements": {
    "repositoryMetadata": {
      "arn": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "creationDate": "Dec 14, 2016 6:19:14 PM",
      "repositoryId": "8afe792d-EXAMPLE",
```

```
      "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo",
      "repositoryName": "MyDemoRepo",
      "accountId": "111122223333",
      "cloneUrlHttp": "https://git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo",
      "repositoryDescription": "Creating a demonstration repository.",
      "lastModifiedDate": "Dec 14, 2016 6:19:14 PM"
    }
  },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": false,
  "resources": [
    {
      "ARN": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
    }
  ],
  "eventType": "AwsApiCall",
  "apiVersion": "2015-04-13",
  "recipientAccountId": "111122223333"
}
```

Examples: Log entries for Git pull calls to a CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates the `GitPull` action where the local repo is already up-to-date.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName":"Mary_Major"
    },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPull",
```

```
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "git/2.11.0.windows.1",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "dataTransferred": false,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
      }
    ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

The following example shows a CloudTrail log entry that demonstrates the `GitPull` action where the local repo is not up-to-date and so data is transferred from the CodeCommit repository to the local repo.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName":"Mary_Major"
    },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPull",
  "awsRegion": "us-east-2",
```

```
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "git/2.10.1",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "capabilities": [
      "multi_ack_detailed",
      "side-band-64k",
      "thin-pack"
      ],
    "dataTransferred": true,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    "shallow": false
    },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
      }
    ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Example: A log entry for a successful push to a CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates a successful `GitPush` action. The `GitPush` action appears twice in a log entry for a successful push.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
    "userName":"Mary_Major"
    },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPush",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "git/2.10.1",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "dataTransferred": false,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": false,
  "resources": [
    {
      "ARN": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
      }
    ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
},
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName":"Mary_Major"
    },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPush",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
```

```
  "userAgent": "git/2.10.1",
  "requestParameters": {
    "references": [
      {
        "commit": "100644EXAMPLE",
        "ref": "refs/heads/main"
        }
      ]
    },
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "capabilities": [
      "report-status",
      "side-band-64k"
      ],
    "dataTransferred": true,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": false,
  "resources": [
    {
      "ARN": "arn:aws:codecommit:us-east-2:111122223333:MyDemoRepo",
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
      }
    ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Creating CodeCommit resources with AWS CloudFormation

AWS CodeCommit is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as repositories), and CloudFormation provisions and configures those resources for you.

When you use CloudFormation, you can reuse your template to set up your CodeCommit resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

CodeCommit and CloudFormation templates

To provision and configure resources for CodeCommit and related services, you must understand CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use CloudFormation Designer to help you get started with CloudFormation templates. For more information, see What is CloudFormation Designer? in the *AWS CloudFormation User Guide*.

CodeCommit supports creating repositories in CloudFormation Unlike creating repositories from the console or command line, you can use CloudFormation to create repositories and automatically commit code to the newly created repository from a specified .zip file in an Amazon S3 bucket. For more information, including examples of JSON and YAML templates for repositories, see AWS::CodeCommit::Repository.

When you create a CodeCommit repository using CloudFormation, you have the option to commit code to that repository as part of the creation process as long as the archive is less than 20 MB by configuring properties in AWS:CodeCommit::Repository Code. You can specify the Amazon S3 bucket where the code is stored, and optionally use the BranchName property to specify the name of the default branch that will be created in the initial commit of that code. These properties are only used in initial repository creation, and are ignored on stack updates. You cannot use these properties to make additional commits to a repository, or to change the name of the default branch after the initial commit is made.

> **Note**
>
> On January 19, 2021, AWS changed the name of the default branch in CodeCommit from *master* to *main*. This name change affects the default behavior of CodeCommit when creating the initial commit for repositories using the CodeCommit console, the CodeCommit APIs, the AWS SDKs, and the AWS CLI. Repositories created with CloudFormation or the AWS CDK with an initial commit of code as part of creation align with this change beginning March 4, 2021. This change does not affect existing repositories or branches. Customers who use local Git clients to create their initial commits have a default branch name that follows the configuration of those Git clients. For more information, see Working with branches, Create a commit, and Change branch settings.

You can also create templates that create related resources, such as notification rules for repositories, AWS CodeBuild build projects, AWS CodeDeploy applications, and AWS CodePipeline pipelines.

Template examples

The following examples create a CodeCommit repository named *MyDemoRepo*. The newly created repository is populated with code stored in an Amazon S3 bucket named *MySourceCodeBucket* and placed in a branch named *development*, which is the default branch for the repository.

> **Note**
>
> The name of the Amazon S3 bucket that contains the ZIP file with the content that will be committed to the new repository can be specified using an ARN or the name of the bucket in the Amazon Web Services account. The Amazon S3 object key is as defined in the Amazon S3 Developer Guide.

JSON:

```
{
    "MyRepo": {
        "Type": "AWS::CodeCommit::Repository",
        "Properties": {
            "RepositoryName": "MyDemoRepo",
```

```
            "RepositoryDescription": "This is a repository for my project with code
 from MySourceCodeBucket.",
            "Code": {
                "BranchName": "development",
                "S3": {
                    "Bucket": "MySourceCodeBucket",
                    "Key": "MyKey",
                    "ObjectVersion": "1"
                }
            }
        }
    }
}
```

YAML:

```
MyRepo:
  Type: AWS::CodeCommit::Repository
  Properties:
    RepositoryName: MyDemoRepo
    RepositoryDescription: This is a repository for my project with code from
 MySourceCodeBucket.
    Code:
      BranchName: development
      S3:
        Bucket: MySourceCodeBucket,
        Key: MyKey,
        ObjectVersion: 1
```

For more examples, see AWS::CodeCommit::Repository.

CloudFormation, CodeCommit, and the AWS Cloud Development Kit (AWS CDK)

Repositories created using the AWS CDK use CloudFormation functionality in their creation. Understanding how CloudFormation templates work with CodeCommit resources can help you create and manage your AWS CDK code. For more information about the AWS CDK, see the AWS Cloud Development Kit (AWS CDK) Developer Guide and the AWS CDK API Reference.

The following AWS CDK Typescript example creates a CodeCommit repository named *MyDemoRepo*. The newly created repository is populated with code stored in an Amazon S3 bucket

named *MySourceCodeBucket* and placed in a branch named *development*, which is the default branch for the repository.

```
import * as cdk from '@aws-cdk/core';
import codecommit = require('@aws-cdk/aws-codecommit');
export class CdkCodecommitStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    // The code creates a CodeCommit repository with a default branch name development
      new codecommit.CfnRepository(this, 'MyRepoResource', {
            repositoryName: "MyDemoRepo",
            code: {
              "branchName": "development",
              "s3": {
                "bucket": "MySourceCodeBucket",
                "key": "MyKey"
              }
            },
        }
     );
  }
}
```

Learn more about CloudFormation

To learn more about CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation Command Line Interface User Guide

Troubleshooting AWS CodeCommit

The following information might help you troubleshoot common issues in AWS CodeCommit.

Topics

- Troubleshooting Git credentials and HTTPS connections to AWS CodeCommit
- Troubleshooting git-remote-codecommit and AWS CodeCommit
- Troubleshooting SSH connections to AWS CodeCommit
- Troubleshooting the credential helper and HTTPS connections to AWS CodeCommit
- Troubleshooting Git clients and AWS CodeCommit
- Troubleshooting access errors and AWS CodeCommit
- Troubleshooting configuration errors and AWS CodeCommit
- Troubleshooting console errors and AWS CodeCommit
- Troubleshooting triggers and AWS CodeCommit
- Turn on debugging

Troubleshooting Git credentials and HTTPS connections to AWS CodeCommit

The following information might help you troubleshoot common issues when using Git credentials and HTTPS to connect to AWS CodeCommit repositories.

Topics

- Git credentials for AWS CodeCommit: I keep seeing a prompt for credentials when I connect to my CodeCommit repository at the terminal or command line
- Git credentials for AWS CodeCommit: I set up Git credentials, but my system is not using them

Git credentials for AWS CodeCommit: I keep seeing a prompt for credentials when I connect to my CodeCommit repository at the terminal or command line

Problem: When you try to push, pull, or otherwise interact with a CodeCommit repository from the terminal or command line, you are prompted to provide a user name and password, and you must supply the Git credentials for your IAM user.

Possible fixes: The most common causes for this error are that your local computer is running an operating system that does not support credential management, or it does not have a credential management utility installed, or the Git credentials for your IAM user have not been saved to one of these credential management systems. Depending on your operating system and local environment, you might need to install a credential manager, configure the credential manager that is included in your operating system, or customize your local environment to use credential storage. For example, if your computer is running macOS, you can use the Keychain Access utility to store your credentials. If your computer is running Windows, you can use the Git Credential Manager that is installed with Git for Windows. For more information, see For HTTPS users using Git credentials and Credential Storage in the Git documentation.

Git credentials for AWS CodeCommit: I set up Git credentials, but my system is not using them

Problem: When you try to use CodeCommit with a Git client, the client does not appear to use the Git credentials for your IAM user.

Possible fixes: The most common cause for this error is that you previously set up your computer to use the credential helper that is included with the AWS CLI. Check your .gitconfig file for configuration sections similar to the following, and remove them:

```
[credential "https://git-codecommit.*.amazonaws.com"]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true
```

Save the file, and then open a new command line or terminal session before you attempt to connect again.

You may also have multiple credential helpers or managers set up on your computer, and your system might be defaulting to another configuration. To reset which credential helper is used as

the default, you can use the **--system** option instead of **--global** or **--local** when running the **git config** command.

For more information, see For HTTPS users using Git credentials and Credential Storage in the Git documentation.

Troubleshooting git-remote-codecommit and AWS CodeCommit

The following information might help you troubleshoot issues with **git-remote-codecommit** when connecting with AWS CodeCommit repositories.

Topics

- I see an error: git: 'remote-codecommit' is not a git command
- I see an error: fatal: Unable to find remote helper for 'codecommit'
- Cloning error: I cannot clone a CodeCommit repository from an IDE
- Push or pull error: I cannot push or pull commits from an IDE to a CodeCommit repository
- Installation error: I see an externally-managed-environment error when I try to install git-remote-codecommit

I see an error: git: 'remote-codecommit' is not a git command

Problem: When you try to use git-remote-codecommit, you see an error that git-remote-codecommit is not a git command. See 'git --help'".

Possible fixes: The most common reason for this error is that either you have not added the git-remote-codecommit executable to your PATH, or that the string contains a syntax error. This can happen where a hyphen is missing between git and remote-codecommit, or when an extra git is placed before git-remote-codecommit.

You might also encounter this error if you have updated your local installation to an unsupported version. For more information about supported versions of Python, see git-remote-codecommit.

For more information about setting up and using git-remote-codecommit, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

I see an error: fatal: Unable to find remote helper for 'codecommit'

Problem: When you try to use git-remote-codecommit, you see an error stating "fatal: Unable to find remote helper for 'codecommit'".

Possible fixes: The most common reasons for this error are:

- The setup is not complete for git-remote-codecommit
- You have installed git-remote-codecommit in a location that is not in your path or not configured as part of the `Path` environment variable
- Python is not in your path or not configured as part of the `Path` environment variable
- You are using a terminal or command line window that has not been restarted since the installation of git-remote-codecommit completed

For more information about setting up and using git-remote-codecommit, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

Cloning error: I cannot clone a CodeCommit repository from an IDE

Problem: When you try to clone a CodeCommit repository in an IDE, you see an error that says the endpoint or URL is not valid.

Possible fixes: Not all IDEs support the URL used by **git-remote-codecommit** during cloning. Clone the repository locally from the terminal or command line, and then add that local repo to your IDE. For more information, see Step 3: Connect to the CodeCommit console and clone the repository.

Push or pull error: I cannot push or pull commits from an IDE to a CodeCommit repository

Problem: When you try to pull or push code from an IDE, you see a connection error.

Possible fixes: The most common reason for this error is that the IDE is not compatible with Git remote helpers such as **git-remote-codecommit**. Instead of using the IDE functionality to commit, push, and pull code, update the local repo manually from the command line or terminal using Git commands.

For more information about remote helpers and Git, see the Git documentation.

Installation error: I see an externally-managed-environment error when I try to install git-remote-codecommit

Problem: When you try to run the `pip install git-remote-codecommit` command, you see an error indicating that the environment is externally managed.

Possible fixes: The most common reason for this error is that you're running a distribution (distro) of Python that defines an EXTERNALLY-MANAGED marker file. The best solution for this is to create and use a virtual environment.

For more information about externally managed environments and Python, see Externally Managed Environments and Install packages in a virtual environment using pip and venv in the Python documentation.

Troubleshooting SSH connections to AWS CodeCommit

The following information might help you troubleshoot common issues when using SSH to connect to CodeCommit repositories.

Topics

- Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems
- Access error: Public key is uploaded successfully to IAM and SSH tested successfully but connection fails on Windows systems
- Authentication challenge: Authenticity of host can't be established when connecting to a CodeCommit repository
- IAM error: 'Invalid format' when attempting to add a public key to IAM
- I need to access CodeCommit repositories in multiple Amazon Web Services accounts with SSH credentials
- Git on Windows: Bash emulator or command line freezes when attempting to connect using SSH
- Public key format requires specification in some distributions of Linux
- Access error: SSH public key denied when connecting to a CodeCommit repository

Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems

Problem: When you try to connect to an SSH endpoint to communicate with a CodeCommit repository, either when testing the connection or cloning a repository, the connection fails or is refused.

Possible fixes: The SSH key ID assigned to your public key in IAM might not be associated with your connection attempt. You might not have configured a config file, you might not have access to the configuration file, another setting might be preventing a successful read of the config file, you might have provided the wrong key ID, or you might have provided the ID of the IAM user instead of the key ID.

The SSH key ID can be found in the IAM console in the profile for your IAM user:



> **Note**
>
> If you have more than one SSH key IDs uploaded, the keys are listed alphabetically by key ID, not by upload date. Make sure that you have copied the key ID that is associated with the correct upload date.

Try testing the connection with the following command:

```
ssh Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com
```

If you see a success message after confirming the connection, your SSH key ID is valid. Edit your config file to associate your connection attempts with your public key in IAM. If you do not want to edit your config file, you can preface all connection attempts to your repository with your SSH key ID. For example, if you wanted to clone a repository named *MyDemoRepo* without modifying your config file to associate your connection attempts, you would run the following command:

```
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo my-demo-repo
```

For more information, see For SSH connections on Linux, macOS, or Unix.

Access error: Public key is uploaded successfully to IAM and SSH tested successfully but connection fails on Windows systems

Problem: When you try to use an SSH endpoint to clone or communicate with a CodeCommit repository, an error message appears containing the phrase `No supported authentication methods available`.

Possible fixes: The most common reason for this error is that you have a Windows system environment variable set that directs Windows to use another program when you attempt to use SSH. For example, you might have set a GIT_SSH variable to point to one of the PuTTY set of tools (plink.exe). This might be a legacy configuration, or it might be required for one or more other programs installed on your computer. If you are sure that this environment variable is not required, you can remove it by opening your system properties.

To work around this issue, open a Bash emulator and then try your SSH connection again, but include `GIT_SSH_COMMAND="SSH"` as a prefix. For example, to clone a repository using SSH:

```
GIT_SSH_COMMAND="ssh" git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo my-demo-repo
```

A similar problem might occur if your version of Windows requires that you include the SSH key ID as part of the connection string when connecting through SSH at the Windows command line. Try your connection again, this time including the SSH key ID copied from IAM as part of the command. For example:

```
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/
MyDemoRepo my-demo-repo
```

Authentication challenge: Authenticity of host can't be established when connecting to a CodeCommit repository

Problem: When you try to use an SSH endpoint to communicate with a CodeCommit repository, a warning message appears containing the phrase `The authenticity of host 'host-name' can't be established.`

Possible fixes: Your credentials might not be set up correctly. Follow the instructions in For SSH connections on Linux, macOS, or Unix or For SSH connections on Windows.

If you have followed those steps and the problem persists, someone might be attempting a man-in-the-middle attack. When you see the following message, type no, and press Enter.

```
Are you sure you want to continue connecting (yes/no)?
```

Make sure the fingerprint and public key for CodeCommit connections match those documented in the SSH setup topics before you continue with the connection.

Public fingerprints for CodeCommit

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-east-2.amazonaws.com	MD5	a9:6d:03:ed:08:42:21:be:06:e1:e0:2a:d1:75:31:5e
git-codecommit.us-east-2.amazonaws.com	SHA256	3lBlW2g5xn/NA2Ck6dyeJIrQOWvn7n8UEs56fG6ZIzQ
git-codecommit.us-east-1.amazonaws.com	MD5	a6:9c:7d:bc:35:f5:d4:5f:8b:ba:6f:c8:bc:d4:83:84
git-codecommit.us-east-1.amazonaws.com	SHA256	eLMY1j0DKA4uvDZcl/KgtIayZANwX6t8+8isPtotBoY

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-west-2.amazonaws.com	MD5	a8:68:53:e3:99:ac:6e:d7:04:7e:f7:92:95:77:a9:77
git-codecommit.us-west-2.amazonaws.com	SHA256	0pJx9SQpkbPUAHwy58UVIq0IHcyo1fwCpOOuVgcAWPo
git-codecommit.eu-west-1.amazonaws.com	MD5	93:42:36:ea:22:1f:f1:0f:20:02:4a:79:ff:ea:12:1d
git-codecommit.eu-west-1.amazonaws.com	SHA256	tKjRkOL8dmJyTmSbeSdN1S8F/f0iql3RlvqgTOP1UyQ
git-codecommit.ap-northeast-1.amazonaws.com	MD5	8e:a3:f0:80:98:48:1c:5c:6f:59:db:a7:8f:6e:c6:cb
git-codecommit.ap-northeast-1.amazonaws.com	SHA256	Xk/WeYD/K/bnBybzhiuu4dWpBJtXPf7E30jHU7se4Ow
git-codecommit.ap-southeast-1.amazonaws.com	MD5	65:e5:27:c3:09:68:0d:8e:b7:6d:94:25:80:3e:93:cf
git-codecommit.ap-southeast-1.amazonaws.com	SHA256	ZIsVa7OVzxrTIf+Rk4UbhPv6Es22mSB3uTBojfPXIno
git-codecommit.ap-southeast-2.amazonaws.com	MD5	7b:d2:c1:24:e6:91:a5:7b:fa:c1:0c:35:95:87:da:a0

Server	Cryptographic hash type	Fingerprint
git-codecommit.ap-southeast-2.amazonaws.com	SHA256	nYp+gHas80HY3DqbP4yanCDFhqDVjseefVbHEXqH2Ec
git-codecommit.ap-southeast-3.amazonaws.com	MD5	64:d9:e0:53:19:4f:a8:91:9a:c3:53:22:a6:a8:ed:a6
git-codecommit.ap-southeast-3.amazonaws.com	SHA256	ATdkGSFhpqIu7RqUVT/1RZo6MLxxxUW9NoDVMbAc/6g
git-codecommit.me-central-1.amazonaws.com	MD5	bd:fa:e2:f9:05:84:d6:39:6f:bc:d6:8d:fe:de:61:76
git-codecommit.me-central-1.amazonaws.com	SHA256	grceUDWubo4MzG1NoaKZKUfrgPvfN3ijliOnQr1lTZA
git-codecommit.eu-central-1.amazonaws.com	MD5	74:5a:e8:02:fc:b2:9c:06:10:b4:78:84:65:94:22:2d
git-codecommit.eu-central-1.amazonaws.com	SHA256	MwGrkiEki8QkkBtlAgXbYt0hoZYBnZF62VY5RzGJEUY
git-codecommit.ap-northeast-2.amazonaws.com	MD5	9f:68:48:9b:5f:fc:96:69:39:45:58:87:95:b3:69:ed
git-codecommit.ap-northeast-2.amazonaws.com	SHA256	eegAPQrWY9YsYo9ZHIKOmxetfXBHzAZd8Eya53Qcwko

Server	Cryptographic hash type	Fingerprint
git-codecommit.sa-east-1.amazonaws.com	MD5	74:99:9d:ff:2b:ef:63:c6:4b:b4:6a:7f:62:c5:4b:51
git-codecommit.sa-east-1.amazonaws.com	SHA256	kW+VKB0jpRaG/ZbXkgbtMQbKgEDK7JnISV3SVoyCmzU
git-codecommit.us-west-1.amazonaws.com	MD5	3b:76:18:83:13:2c:f8:eb:e9:a3:d0:51:10:32:e7:d1
git-codecommit.us-west-1.amazonaws.com	SHA256	gzauWTWXDK2u5KuMMi5vbKTmfyerdIwgSbzYBODLpzg
git-codecommit.eu-west-2.amazonaws.com	MD5	a5:65:a6:b1:84:02:b1:95:43:f9:0e:de:dd:ed:61:d3
git-codecommit.eu-west-2.amazonaws.com	SHA256	r0Rwz5k/IHp/QyrRnfiM9j02D5UEqMbtFNTuDG2hNbs
git-codecommit.ap-south-1.amazonaws.com	MD5	da:41:1e:07:3b:9e:76:a0:c5:1e:64:88:03:69:86:21
git-codecommit.ap-south-1.amazonaws.com	SHA256	hUKwnTj7+Xpx4Kddb6p45j4RazIJ4IhAMD8k29itOfE
git-codecommit.ap-south-2.amazonaws.com	MD5	bc:cc:9f:15:f8:f3:58:a2:68:65:21:e2:23:71:8d:ce

Server	Cryptographic hash type	Fingerprint
git-codecommit.ap-south-2.amazonaws.com	SHA256	Xe0CyZEOvgR5Xa2YUGqf+jn8/Ut7l7nX/CmslSFNEig
git-codecommit.ca-central-1.amazonaws.com	MD5	9f:7c:a2:2f:8c:b5:74:fd:ab:b7:e1:fd:af:46:ed:23
git-codecommit.ca-central-1.amazonaws.com	SHA256	Qz5puafQdANVprLlj6r0Qyh4lCNsF6ob61dGcPtFS7w
git-codecommit.eu-west-3.amazonaws.com	MD5	1b:7f:97:dd:d7:76:8a:32:2c:bd:2c:7b:33:74:6a:76
git-codecommit.eu-west-3.amazonaws.com	SHA256	uw7c2FL564jVoFgtc+ikzILnKBsZz7t9+CFdSJjKbLI
git-codecommit.us-gov-west-1.amazonaws.com	MD5	9f:6c:19:3b:88:cd:e8:88:1b:9c:98:6a:95:31:8a:69
git-codecommit.us-gov-west-1.amazonaws.com	SHA256	djXQoSIFcg8vHe0KVH1xW/gOF9X37tWTqu4Hkng75x4
git-codecommit.us-gov-east-1.amazonaws.com	MD5	00:8d:b5:55:6f:05:78:05:ed:ea:cb:3f:e6:f0:62:f2
git-codecommit.us-gov-east-1.amazonaws.com	SHA256	fVb+R0z7qW7minenW+rUpAABRCRBTCzmETAJEQrg98

Server	Cryptographic hash type	Fingerprint
git-codecommit.eu-north-1.amazonaws.com	MD5	8e:53:d8:59:35:88:82:fd:73:4b:60:8a:50:70:38:f4
git-codecommit.eu-north-1.amazonaws.com	SHA256	b6KSK7xKq+V8jl7iuAcjqXsG7zkqoUZZmmhYYFBq1wQ
git-codecommit.me-south-1.amazonaws.com	MD5	0e:39:28:56:d5:41:e6:8d:fa:81:45:37:fb:f3:cd:f7
git-codecommit.me-south-1.amazonaws.com	SHA256	O+NToCGgjrHekiBuOl0ad7ROGEsz+DBLXOd/c9wc0JU
git-codecommit.ap-east-1.amazonaws.com	MD5	a8:00:3d:24:52:9d:61:0e:f6:e3:88:c8:96:01:1c:fe
git-codecommit.ap-east-1.amazonaws.com	SHA256	LafadYwUYW8hONoTRpojbjNs9IRnbEwHtezD3aAIBX0
git-codecommit.cn-north-1.amazonaws.com.cn	MD5	11:7e:2d:74:9e:3b:94:a2:69:14:75:6f:5e:22:3b:b3
git-codecommit.cn-north-1.amazonaws.com.cn	SHA256	IYUXxH2OpTDsyYMLIp+JY8CTLS4UX+ZC5JVZXPRaxc8
git-codecommit.cn-northwest-1.amazonaws.com.cn	MD5	2e:a7:fb:4c:33:ac:6c:f9:aa:f2:bc:fb:0a:7b:1e:b6

Server	Cryptographic hash type	Fingerprint
git-codecommit.cn-northwest-1.amazonaws.com.cn	SHA256	wqjd6eHd0+mOBx+dCNuL0omUoCNjaDtZiEpWj5TmCfQ
git-codecommit.eu-south-1.amazonaws.com	MD5	b9:f6:5d:e2:48:92:3f:a9:37:1e:c4:d0:32:0e:fb:11
git-codecommit.eu-south-1.amazonaws.com	SHA256	lyXrWbCg3uQmJrl1XxB/ASR7ugW1Ysf5yzYOJbudHsI
git-codecommit.ap-northeast-3.amazonaws.com	MD5	25:17:40:da:b9:d4:18:c3:b6:b3:fb:ed:1c:20:fe:29
git-codecommit.ap-northeast-3.amazonaws.com	SHA256	2B815B9F0AvwLnRxSVxUz4kDYmtEQUGGdQYP8OQLXhA
git-codecommit.af-south-1.amazonaws.com	MD5	21:a0:ba:d7:c1:d1:b5:39:98:8d:4d:7c:96:f5:ca:29
git-codecommit.af-south-1.amazonaws.com	SHA256	C34ji3x/cnsDZjUpyNGXdE5pjHYimqJrQZ3leTgqJHM
git-codecommit.il-central-1.amazonaws.com	MD5	04:74:89:16:98:7a:61:b1:69:46:42:3c:d1:b4:ac:a9
git-codecommit.il-central-1.amazonaws.com	SHA256	uFxhp51kUWhleTLeYbxQVYm4RnNLNZ5Dbdm1cgdSl/8

IAM error: 'Invalid format' when attempting to add a public key to IAM

Problem: In IAM, when attempting to set up to use SSH with CodeCommit, an error message appears containing the phrase `Invalid format` when you attempt to add your public key.

Possible fixes: IAM requires that the public key must be encoded in ssh-rsa format or PEM format. It accepts public keys in the OpenSSH format only and has additional requirements as specified in Use SSH Keys with CodeCommit in the *IAM User Guide*. If you provide your public key in another format, or if the key does not contain the required number of bits, you see this error.

- When you copied the SSH public key, your operating system might have introduced line breaks. Make sure that there are no line breaks in the public key that you add to IAM.
- Some Windows operating systems do not use the OpenSSH format. To generate a key pair and copy the OpenSSH format required by IAM, see the section called "Step 3: Set up the public and private keys for Git and CodeCommit ".

For more information about the requirements for SSH keys in IAM, see Use SSH Keys with CodeCommit in the *IAM User Guide*.

I need to access CodeCommit repositories in multiple Amazon Web Services accounts with SSH credentials

Problem: I want to set up SSH access to CodeCommit repositories in more than one Amazon Web Services account.

Possible fixes: You can create unique SSH public/private key pairs for each Amazon Web Services account and configure IAM with each key. You can then configure your ~/.ssh/config file with information about each IAM User ID associated with the public key. For example:

```
Host codecommit-1
    Hostname git-codecommit.us-east-1.amazonaws.com
    User SSH-KEY-ID-1 # This is the SSH Key ID you copied from IAM in Amazon Web
 Services account 1 (for example, APKAEIBAERJR2EXAMPLE1).
    IdentityFile ~/.ssh/codecommit_rsa # This is the path to the associated public key
 file, such as id_rsa.  We advise creating CodeCommit specific _rsa files.

Host codecommit-2
    Hostname git-codecommit.us-east-1.amazonaws.com
```

```
    User SSH-KEY-ID-2 # This is the SSH Key ID you copied from IAM in Amazon Web
Services account 2 (for example, APKAEIBAERJR2EXAMPLE2).
    IdentityFile ~/.ssh/codecommit_2_rsa # This is the path to the other associated
public key file.  We advise creating CodeCommit specific _rsa files.
```

In this configuration, you will be able to replace 'git-codecommit.us-east-1.amazonaws.com' with 'codecommit-2'. For example, to clone a repository in your second Amazon Web Services account:

```
git clone ssh://codecommit-2/v1/repos/YourRepositoryName
```

To set up a remote for your repository, run **git remote add**. For example:

```
git remote add origin ssh://codecommit-2/v1/repos/YourRepositoryName
```

For more examples, see this forum post and this contribution on GitHub.

Git on Windows: Bash emulator or command line freezes when attempting to connect using SSH

Problem: After you configure SSH access for Windows and confirm connectivity at the command line or terminal, you see a message that the server's host key is not cached in the registry, and the prompt to store the key in the cache is frozen (does not accept y/n/return input) when you attempt to use commands such as **git pull**, **git push**, or **git clone** at the command prompt or Bash emulator.

Possible fixes: The most common cause for this error is that your Git environment is configured to use something other than OpenSSH for authentication (probably PuTTY). This is known to cause problems with the caching of keys in some configurations. To fix this problem, try one of the following:

- Open a Bash emulator and add the `GIT_SSH_COMMAND="ssh"` parameter before the Git command. For example, if you are attempting to push to a repository, instead of typing **git push**, type:

  ```
  GIT_SSH_COMMAND="ssh" git push
  ```

- If you have PuTTY installed, open PuTTY, and in **Host Name (or IP address)**, enter the CodeCommit endpoint you want to reach (for example, git-codecommit.us-east-2.amazonaws.com). Choose **Open**. When prompted by the PuTTY security alert, choose **Yes** to permanently cache the key.

- Rename or delete the `GIT_SSH` environment variable if you are no longer using it. Then open a new command prompt or Bash emulator session, and try your command again.

For other solutions, see Git clone/pull continually freezing at Store key in cache on Stack Overflow.

Public key format requires specification in some distributions of Linux

Problem: When you try to configure a public-private key pair, you receive an error.

Possible fixes: Some distributions of Linux require an additional line of configuration in the `~/.ssh/config` file that specifies the accepted types of public keys. For more information, see the documentation for your distribution about `PubkeyAcceptedKeyTypes`.

Access error: SSH public key denied when connecting to a CodeCommit repository

Problem: When you try to use an SSH endpoint to communicate with a CodeCommit repository, an error message appears containing the phrase `Error: public key denied`.

Possible fixes: The most common reason for this error is that you have not completed setup for SSH connections. Configure a public and private SSH key pair, and then associate the public key with your IAM user. For more information about configuring SSH, see For SSH connections on Linux, macOS, or Unix and For SSH connections on Windows.

Troubleshooting the credential helper and HTTPS connections to AWS CodeCommit

The following information might help you troubleshoot common issues when you use the credential helper included with the AWS CLI and HTTPS to connect to CodeCommit repositories.

> **Note**
>
> Although the credential helper is a supported method for connecting to CodeCommit using federated access, an identity provider, or temporary credentials, the recommended method is to install and use the **git-remote-codecommit** utility. For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit.

Topics

- I receive an error when running the git config command to configure the credential helper
- I see an error reading the user name when attempting to clone a repository
- I get a command not found error in Windows when using the credential helper
- I am prompted for a user name when I connect to a CodeCommit repository
- Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository (403)
- Git for Windows: I installed Git for Windows, but I am denied access to my repository (403)

I receive an error when running the `git config` command to configure the credential helper

Problem: When you try to run the git config command to configure the credential helper to communicate with a CodeCommit repository, you see an error that there are too few arguments, or a usage prompt suggesting Git config commands and syntax.

Possible fixes: The most common reason for this error is that either single quotes are used for the command on a Windows operating system, or double quotes are used for the command in a Linux, macOS, or Unix operating system. The correct syntax is as follows:

- Windows: `git config --global credential.helper "!aws codecommit credential-helper $@"`
- Linux, macOS, or Unix: `git config --global credential.helper '!aws codecommit credential-helper $@'`

I see an error reading the user name when attempting to clone a repository

Problem: When you try to clone a repository using the credential helper, you see an error stating that the system could not read the user name for the repository URL.

Possible fixes: The most common reason for this error is that the .gitconfig file is not created or configured correctly on your computer. Open your .gitconfig file and make sure that the credential helper is set correctly. If you are on a computer running Linux, macOS, or Unix, also make sure that the value of `$HOME` is set correctly for your system.

I get a command not found error in Windows when using the credential helper

Problem: After updating the AWS CLI, credential helper connections to CodeCommit repositories fail with `aws codecommit credential-helper $@ get: aws: command not found`.

Cause: The most common reason for this error is that your AWS CLI version has been updated to a version that uses Python 3. There is a known issue with the MSI package. To verify whether you have one of the affected versions, open a command line and run the following command: `aws --version`

If the output Python version begins with a 3, you have an affected version. For example:

```
aws-cli/1.16.62 Python/3.6.2 Darwin/16.7.0 botocore/1.12.52
```

Possible fixes: You can work around this issue by doing one of the following:

- Install and configure the AWS CLI on Windows using Python and pip instead of the MSI. For more information, see Install Python, pip, and the AWS CLI on Windows.
- Manually edit your `.gitconfig` file to change the `[credential]` section to explicitly point to `aws.cmd` on your local computer. For example:

  ```
  [credential]
      helper = !"\C:\\Program Files\\Amazon\\AWSCLI\\bin\\aws.cmd\" codecommit
   credential-helper $@
      UseHttpPath = true
  ```

- Run the **git config** command to update your `.gitconfig` file to explicitly reference `aws.cmd`, and manually update your PATH environment variable to include the path to the command as needed. For example:

  ```
  git config --global credential.helper "!aws.cmd codecommit credential-helper $@"
  git config --global credential.UseHttpPath true
  ```

I am prompted for a user name when I connect to a CodeCommit repository

Problem: When you try to use the credential helper to communicate with a CodeCommit repository, a message appears prompting you for your user name.

Possible fixes: Configure your AWS profile or make sure the profile you are using is the one you configured for working with CodeCommit. For more information about setting up, see Setup steps for HTTPS connections to AWS CodeCommit repositories on Linux, macOS, or Unix with the AWS CLI credential helper or Setup steps for HTTPS connections to AWS CodeCommit repositories on Windows with the AWS CLI credential helper. For more information about IAM, access keys, and secret keys, see Managing Access Keys for IAM Users and How Do I Get Credentials?

Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository (403)

Problem: On macOS, the credential helper does not seem to access or use your credentials as expected. This can be caused by two different problems:

- The AWS CLI is configured for an AWS Region different from the one where the repository exists.
- The Keychain Access utility has saved credentials that have since expired.

Possible fixes: To verify whether the AWS CLI is configured for the correct region, run the **aws configure** command, and review the displayed information. If the CodeCommit repository is in an AWS Region different from the one shown for the AWS CLI, you must run the **aws configure** command and change the values to ones appropriate for that Region. For more information, see Step 1: Initial configuration for CodeCommit.

The default version of Git released on OS X and macOS uses the Keychain Access utility to save generated credentials. For security reasons, the password generated for access to your CodeCommit repository is temporary, so the credentials stored in the keychain stop working after about 15 minutes. If you are only accessing Git with CodeCommit, try the following:

1. In Terminal, run the **git config** command to find the Git configuration file (`gitconfig`) where the Keychain Access utility is defined. Depending on your local system and preferences, you might have more than one `gitconfig` file.

```
git config -l --show-origin | grep credential
```

In the output from this command, search for results similar to:

```
file:/path/to/gitconfig  credential.helper=osxkeychain
```

The file listed at the beginning of this line is the Git configuration file you must edit.

2. To edit the Git configuration file, use a plain-text editor or run the following command:

```
nano /usr/local/git/etc/gitconfig
```

3. Modify the configuration using one of the following strategies:

   - Comment out or delete the credential section that contains `helper = osxkeychain`. For example:

     ```
     # helper = osxkeychain
     ```

   - Update both the `aws credential helper` and `osxkeychain` credential helper sections to have context. For example, if `osxkeychain` is used to authenticate to GitHub:

     ```
     [credential "https://git-codecommit.us-east-1.amazonaws\.com"]
       helper = !aws --profile CodeCommitProfile codecommit credential-helper $@
       UseHttpPath = true
     [credential "https://github.com"]
       helper = osxkeychain
     ```

     In this configuration, Git will use the `osxkeychain` helper when the remote host matches "`https://github.com`" and the credential helper when the remote host matches "`https://git-codecommit\.us-east-1\.amazonaws.com`".

   - Include an empty string helper before the credential helper. For example, to not use the `osxkeychain` helper when using the profile named *`CodeCommitProfile`* with the CLI:

     ```
     [credential]
       helper =
       helper = !aws --profile CodeCommitProfile codecommit credential-helper $@
       UseHttpPath = true
     ```

> **Tip**
>
> You could also configure the line following the empty string helper line to not match CodeCommit if you want it to exclude all profiles:
>
> ```
> helper = !aws codecommit credential-helper $@
> ```

Alternatively, if you want to continue to use the Keychain Access utility to cache credentials for other Git repositories, modify the header instead of commenting out the line. For example, to allow cached credentials for GitHub, you could modify the header as follows:

```
[credential "https://github.com"]
   helper = osxkeychain
```

If you are accessing other repositories with Git, you can configure the Keychain Access utility so that it does not supply credentials for your CodeCommit repositories. To configure the Keychain Access utility:

1. Open the Keychain Access utility. (You can use Finder to locate it.)
2. Search for `git-codecommit.`*`us-east-2`*`.amazonaws.com` and replace *`us-east-2`* with the AWS Region where the repository exists. Highlight the row, open the context (right-click) menu, and then choose **Get Info**.
3. Choose the **Access Control** tab.
4. In **Confirm before allowing access**, choose `git-credential-osxkeychain`, and then choose the minus sign to remove it from the list.

> **Note**
>
> After removing `git-credential-osxkeychain` from the list, you see a dialog box whenever you run a Git command. Choose **Deny** to continue. If you find the pop-ups too disruptive, here are some alternatives:

- Connect to CodeCommit using SSH or Git credentials instead of the credential helper with HTTPS. For more information, see For SSH connections on Linux, macOS, or Unix and Setup for HTTPS users using Git credentials.
- In the Keychain Access utility, on the **Access Control** tab for `git-codecommit.us-east-2.amazonaws.com`, choose the **Allow all applications to access this item (access to this item is not restricted)** option. This prevents the pop-ups, but the credentials eventually expire (on average, this takes about 15 minutes) and you then see a 403 error message. When this happens, you must delete the keychain item to restore functionality.
- Install a version of Git that does not use the keychain by default.
- Consider a scripting solution for deleting the keychain item. To view a community-generated sample of a scripted solution, see Mac OS X Script to Periodically Delete Cached Credentials in the OS X Certificate Store in Product and service integrations.

If you want to stop Git from using the Keychain Access utility entirely, you can configure Git to stop using osxkeychain as the credential helper. For example, if you open a terminal and run the command `git config --system credential.helper`, and it returns `osxkeychain`, Git is set to use the Keychain Access utility. You can change this by running the following command:

```
git config --system --unset credential.helper
```

Be aware that by running this command with the `--system` option changes the Git behavior system-wide for all users, and this might have unintended consequences for other users, or for other repositories if you're using other repository services in addition to CodeCommit. Also be aware that this approach might require the use of `sudo`, and that your account might not have sufficient system permissions to apply this change. Make sure to verify that the command applied successfully by running the `git config --system credential.helper` command again. For more information, see Customizing Git - Git Configuration and this article on Stack Overflow.

Git for Windows: I installed Git for Windows, but I am denied access to my repository (403)

Problem: On Windows, the credential helper does not seem to access or use your credentials as expected. This can be caused by different problems:

- The AWS CLI is configured for an AWS Region different from the one where the repository exists.
- By default, Git for Windows installs a Git Credential Manager utility that is not compatible with CodeCommit connections that use the AWS credential helper. When installed, it causes connections to the repository to fail even though the credential helper has been installed with the AWS CLI and configured for connections to CodeCommit.
- Some versions of Git for Windows might not be in full compliance with RFC 2617 and RFC 4559, which could potentially cause issues with both Git credentials and the credential helper included with the AWS CLI. For more information, see Version 2.11.0(3) does not ask for username/password.

Possible fixes:

- If you are attempting to use the credential helper included with the AWS CLI, consider connecting with Git credentials over HTTPS instead of using the credential helper. Git credentials configured for your IAM user are compatible with the Git Credential Manager for Windows, unlike the credential helper for AWS CodeCommit. For more information, see For HTTPS users using Git credentials.

  If you want to use the credential helper, to verify whether the AWS CLI is configured for the correct AWS Region, run the **aws configure** command, and review the displayed information. If the CodeCommit repository is in an AWS Region different from the one shown for the AWS CLI, you must run the **aws configure** command and change the values to ones appropriate for that Region. For more information, see Step 1: Initial configuration for CodeCommit.
- If possible, uninstall and reinstall Git for Windows. When you install Git for Windows, clear the check box for the option to install the Git Credential Manager utility. This credential manager is not compatible with the credential helper for AWS CodeCommit. If you installed the Git Credential Manager or another credential management utility and you do not want to uninstall it, you can modify your `.gitconfig` file and add credential management for CodeCommit:

  1. Open **Control Panel**, choose **Credential Manager**, and remove any stored credentials for CodeCommit.
  2. Open your `.gitconfig` file in any plain-text editor, such as Notepad.

> **Note**
>
> If you work with multiple Git profiles, you might have both local and global `.gitconfig` files. Be sure to edit the appropriate file.

3. Add the following section to your `.gitconfig` file:

```
[credential "https://git-codecommit.*.amazonaws.com"]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true
```

4. Save the file, and then open a new command line session before you attempt to connect again.

You can also use this approach if you want to use the credential helper for AWS CodeCommit when you connect to CodeCommit repositories and another credential management system when you connect to other hosted repositories, such as GitHub repositories.

To reset which credential helper is used as the default, you can use the **--system** option instead of **--global** or **--local** when you run the **git config** command.

- If you are using Git credentials on a Windows computer, you can try to work around any RFC noncompliance issues by including your Git credential user name as part of the connection string. For example, to work around the issue and clone a repository named *MyDemoRepo* in the US East (Ohio) Region:

```
git clone https://Your-Git-Credential-Username@git-codecommit.us-
east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

> **Note**
>
> This approach does not work if you have an `@` character in your Git credentials user name. You must URL-encode (also known as URL escaping or percent-encoding) the character.

Troubleshooting Git clients and AWS CodeCommit

The following information might help you troubleshoot common issues when using Git with AWS CodeCommit repositories. For troubleshooting problems related to Git clients when using HTTPS or SSH, also see Troubleshooting Git credentials (HTTPS), Troubleshooting SSH connections, and Troubleshooting the credential helper (HTTPS).

Topics

- Git error: Error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly
- Git error: Too many reference update commands
- Git error: Push via HTTPS is broken in some versions of Git
- Git error: 'gnutls_handshake() failed'
- Git error: Git cannot find the CodeCommit repository or does not have permission to access the repository
- Git on Windows: No supported authentication methods available (publickey)

Git error: Error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly

Problem: When pushing a large change, a large number of changes, or a large repository, long-running HTTPS connections are often terminated prematurely due to networking issues or firewall settings.

Possible fixes: Push with SSH instead, or when you are migrating a large repository, follow the steps in Migrate a repository in increments. Also, make sure you are not exceeding the size limits for individual files. For more information, see Quotas.

Git error: Too many reference update commands

Problem: The maximum number of reference updates per push is 4,000. This error appears when the push contains more than 4,000 reference updates.

Possible fixes: Try pushing branches and tags individually with `git push --all` and `git push --tags`. If you have too many tags, split the tags into multiple pushes. For more information, see Quotas.

Git error: Push via HTTPS is broken in some versions of Git

Problem: An issue with the curl update to 7.41.0 causes SSPI-based digest authentication to fail. Known affected versions of Git include 1.9.5.msysgit.1. Some versions of Git for Windows might not be in full compliance with RFC 2617 and RFC 4559, which could potentially cause issues with HTTPS connections using either Git credentials or the credential helper included with the AWS CLI.

Possible fixes: Check your version of Git for known issues or use an earlier or later version. For more information about mysysgit, see Push to HTTPS Is Broken in the GitHub forums. For more information about Git for Windows version issues, see Version 2.11.0(3) does not ask for username/password.

Git error: 'gnutls_handshake() failed'

Problem: In Linux, when you try to use Git to communicate with a CodeCommit repository, an error message appears containing the phrase `error: gnutls_handshake() failed`.

Possible fixes: Compile Git against OpenSSL. For one approach, see "Error: gnutls_handshake() failed" When Connecting to HTTPS Servers in the Ask Ubuntu forums.

Alternatively, use SSH instead of HTTPS to communicate with CodeCommit repositories.

Git error: Git cannot find the CodeCommit repository or does not have permission to access the repository

Problem: A trailing slash in the connection string can cause connection attempts to fail.

Possible fixes: Make sure that you have provided the correct name and connection string for the repository, and that there are no trailing slashes. For more information, see Connect to a repository.

Git on Windows: No supported authentication methods available (publickey)

Problem: After you configure SSH access for Windows, you see an access denied error when you attempt to use commands such as **git pull**, **git push**, or **git clone**.

Possible fixes: The most common cause for this error is that a GIT_SSH environment variable exists on your computer and is configured to support another connection utility, such as PuTTY. To fix this problem, try one of the following:

- Open a Bash emulator and add the `GIT_SSH_COMMAND="ssh"` parameter before the Git command. For example, if you are attempting to clone a repository, instead of running **git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo**, run:

```
GIT_SSH_COMMAND="ssh" git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/
repos/MyDemoRepo my-demo-repo
```

- Rename or delete the `GIT_SSH` environment variable if you are no longer using it. Then open a new command prompt or Bash emulator session, and try your command again.

For more information about troubleshooting Git issues on Windows when using SSH, see Troubleshooting SSH connections.

Troubleshooting access errors and AWS CodeCommit

The following information might help you troubleshoot access errors when connecting with AWS CodeCommit repositories.

Topics

- Access error: I am prompted for a user name and password when I connect to a CodeCommit repository from Windows
- Access error: Public key denied when connecting to a CodeCommit repository
- Access error: "Rate Exceeded" or "429" message when connecting to a CodeCommit repository

Access error: I am prompted for a user name and password when I connect to a CodeCommit repository from Windows

Problem: When you try to use Git to communicate with a CodeCommit repository, you see a dialog box that prompts you for your user name and password.

Possible fixes: This might be the built-in credential management system for Windows. Depending on your configuration, do one of the following:

- If you are using HTTPS with Git credentials, your Git credentials are not yet stored in the system. Provide the Git credentials and continue. You should not be prompted again. For more information, see For HTTPS users using Git credentials.

- If you are using HTTPS with the credential helper for AWS CodeCommit, it is not compatible with the Windows credential management system. Choose **Cancel**.

  This might also be an indication that you installed the Git Credential Manager when you installed Git for Windows. The Git Credential Manager is not compatible with the credential helper for CodeCommit included in the AWS CLI. Consider uninstalling the Git Credential Manager. You can also install and configure **git-remote-codecommit** as an alternative to using the credential helper for CodeCommit.

  For more information, see Setup steps for HTTPS connections to AWS CodeCommit with git-remote-codecommit, For HTTPS connections on Windows with the AWS CLI credential helper, and Git for Windows: I installed Git for Windows, but I am denied access to my repository (403).

Access error: Public key denied when connecting to a CodeCommit repository

Problem: When you try to use an SSH endpoint to communicate with a CodeCommit repository, an error message appears containing the phrase `Error: public key denied`.

Possible fixes: The most common reason for this error is that you have not completed setup for SSH connections. Configure a public and private SSH key pair, and then associate the public key with your IAM user. For more information about configuring SSH, see For SSH connections on Linux, macOS, or Unix and For SSH connections on Windows.

Access error: "Rate Exceeded" or "429" message when connecting to a CodeCommit repository

Problem: When you try to communicate with a CodeCommit repository, a message appears that says "Rate Exceeded" or with an error code of "429". Communication either slows significantly or fails.

Cause: All calls to CodeCommit, whether from an application, the AWS CLI, a Git client, or the AWS Management Console, are subject to a maximum number of requests per second and overall active requests. You cannot exceed the maximum allowed request rate for an Amazon Web Services account in any AWS Region. If requests exceed the maximum rate, you receive an error and further calls are temporarily throttled for your Amazon Web Services account. During the throttling period, your connections to CodeCommit are slowed and might fail.

Possible fixes: Take steps to reduce the number of connections or calls to CodeCommit or to spread out requests. Some approaches to consider:

- **Implement jitter in requests, particularly in periodic polling requests**

  If you have an application that is polling CodeCommit periodically and this application is running on multiple Amazon EC2 instances, introduce jitter (a random amount of delay) so that different Amazon EC2 instances do not poll at the same second. We recommend a random number from 0 to 59 seconds to evenly distribute polling mechanisms across a one-minute timeframe.

- **Use an event-based architecture rather than polling**

  Rather than polling, use an event-based architecture so that calls are only made when an event occurs. Consider using CloudWatch Events notifications for AWS CodeCommit events to trigger your workflow.

- **Implement error retries and exponential backoffs for APIs and automated Git actions**

  Error retries and exponential backoffs can help limit the rate of calls. Each AWS SDK implements automatic retry logic and exponential backoff algorithms. For automated Git push and Git pull, you might need to implement your own retry logic. For more information, see Error Retries and Exponential Backoff in AWS.

- **Request a CodeCommit service quota increase in the AWS Support Center**

  To receive a service limit increase, you must confirm that you have already followed the suggestions offered here, including implementation of error retries or exponential backoff methods. In your request, you must also provide the AWS Region, Amazon Web Services account, and timeframe affected by the throttling issues.

Troubleshooting configuration errors and AWS CodeCommit

The following information might help you troubleshoot configuration errors you might see when connecting with AWS CodeCommit repositories.

Topics

- Configuration error: Cannot configure AWS CLI credentials on macOS

Configuration error: Cannot configure AWS CLI credentials on macOS

Problem: When you run `aws configure` to configure the AWS CLI, you see a `ConfigParseError` message.

Possible fixes: The most common cause for this error is that a credentials file already exists. Browse to ~/.aws and look for a file named `credentials`. Rename or delete that file, and then run **aws configure** again.

Troubleshooting console errors and AWS CodeCommit

The following information might help you troubleshoot console errors when using AWS CodeCommit repositories.

Topics

- Access error: Encryption key access denied for a CodeCommit repository from the console or AWS CLI
- Encryption error: Repository can't be decrypted
- Console error: Cannot browse the code in a CodeCommit repository from the console
- Display error: Cannot view a file or a comparison between files

Access error: Encryption key access denied for a CodeCommit repository from the console or AWS CLI

Problem: When you try to access CodeCommit from the console or the AWS CLI, an error message appears containing the phrase `EncryptionKeyAccessDeniedException` or `User is not authorized for the KMS default key for CodeCommit 'aws/codecommit' in your account`.

Possible fixes: The most common cause for this error is that your Amazon Web Services account is not subscribed to AWS Key Management Service, which is required for CodeCommit. Open the AWS KMS console, choose **AWS managed keys**, and then choose **Get Started Now**. If you see a message that you are not currently subscribed to the AWS Key Management Service service, follow the instructions on that page to subscribe. For more information about CodeCommit and AWS Key Management Service, see AWS KMS and encryption.

Encryption error: Repository can't be decrypted

Problem: When you try to access a CodeCommit repository from the console or the AWS CLI, an error message appears containing the phrase `Repository can't be decrypted`.

Possible fixes: The most common cause for this error is that the AWS KMS key used to encrypt and decrypt data for this repository is not active or pending deletion. An active AWS managed key or customer managed key in AWS Key Management Serviceis required for CodeCommit. Open the AWS KMS console, choose **AWS managed keys** or **Customer managed keys**, and make sure that the key used for the repository is present in the AWS Region where the repository exists and that its state is **Active**. For more information about CodeCommit and AWS Key Management Service, see AWS KMS and encryption.

> **Important**
>
> If the key that was used to encrypt and decrypt the data for the repository has been permanently deleted or is otherwise inaccessible, data in the repositories encrypted with that key cannot be accessed.

Console error: Cannot browse the code in a CodeCommit repository from the console

Problem: When you try to browse the contents of a repository from the console, an error message appears denying access.

Possible fixes: The most common cause for this error is that an IAM policy applied to your Amazon Web Services account denies one or more of the permissions required for browsing code from the CodeCommit console. For more information about CodeCommit access permissions and browsing, see Authentication and access control for AWS CodeCommit.

Display error: Cannot view a file or a comparison between files

Problem: When you try to view a file or a comparison between two versions of a file in the CodeCommit console, an error appears stating that the file or difference is too large to display.

Possible fixes: The most common cause for this error is that either the file is too large to display, contains one or more lines that exceeds the character limit for a single line in a file, or that the

difference between the two versions of the file exceeds the line limit. For more information, see Quotas. To view the file or the differences between the version of the file, you can open the file locally in your preferred IDE, use a Git diff tool, or run the **git diff** command.

Troubleshooting triggers and AWS CodeCommit

The following information might help you troubleshoot issues with triggers in AWS CodeCommit.

Topics

- Trigger error: A repository trigger does not run when expected

Trigger error: A repository trigger does not run when expected

Problem: One or more triggers configured for a repository does not appear to run or does not run as expected.

Possible fixes: If the target of the trigger is an AWS Lambda function, make sure you have configured the function's resource policy for access by CodeCommit.

Alternatively, edit the trigger and make sure the events for which you want to trigger actions have been selected and that the branches for the trigger include the branch where you want to see responses to actions. Try changing the settings for the trigger to **All repository events** and **All branches** and then testing the trigger. For more information, see Edit triggers for a repository.

Turn on debugging

Problem: I want to turn on debugging to get more information about my repository and how Git is executing commands.

Possible fixes: Try the following:

1. At the terminal or command prompt, run the following commands on your local machine before running Git commands:

   On Linux, macOS, or Unix:

   ```
   export GIT_TRACE_PACKET=1
   export GIT_TRACE=1
   ```

```
export GIT_CURL_VERBOSE=1
```

On Windows:

```
set GIT_TRACE_PACKET=1
set GIT_TRACE=1
set GIT_CURL_VERBOSE=1
```

> **Note**
>
> Setting `GIT_CURL_VERBOSE` is useful for HTTPS connections only. SSH does not use the `libcurl` library.

2. To get more information about your Git repository, we recommend installing the latest version of git-sizer. Follow the instructions for intalling the utility appropriate to your operating system and environment. Once installed, at the command line or terminal, change directories to your local repository and then run the following command:

```
git-sizer --verbose
```

> **Tip**
>
> Consider saving the output of the command to a file so that you can easily share it with others when troubleshooting problems, particularly over time.

AWS CodeCommit reference

The following reference topics can help you better understand CodeCommit, Git, AWS Regions, service limits, and more.

Topics

- Regions and Git connection endpoints for AWS CodeCommit
- Using AWS CodeCommit with interface VPC endpoints
- Quotas in AWS CodeCommit
- AWS CodeCommit command line reference
- Basic Git commands

Regions and Git connection endpoints for AWS CodeCommit

Each CodeCommit repository is associated with an AWS Region. CodeCommit offers regional endpoints to make your requests to the service. In addition, CodeCommit provides Git connection endpoints for both SSH and HTTPS protocols in every Region where CodeCommit is available.

All of the examples in this guide use the same endpoint URL for Git in US East (Ohio): `git-codecommit.us-east-2.amazonaws.com`. However, when you use Git and configure your connections, make sure you choose the Git connection endpoint that matches the AWS Region that hosts your CodeCommit repository. For example, if you want to make a connection to a repository in US East (N. Virginia), use the endpoint URL of `git-codecommit.us-east-1.amazonaws.com`. This is also true for API calls. When you make connections to a CodeCommit repository with the AWS CLI or the SDKs, make sure you use the correct regional endpoint for the repository.

Topics

- Supported AWS Regions for CodeCommit
- Git connection endpoints
- Server fingerprints for CodeCommit

Supported AWS Regions for CodeCommit

You can create and use CodeCommit repositories in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Frankfurt)
- Europe (Stockholm)
- Europe (Milan)
- Africa (Cape Town)
- Israel (Tel Aviv)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Jakarta)
- Middle East (UAE)
- Asia Pacific (Seoul)
- Asia Pacific (Osaka)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Hong Kong)
- South America (São Paulo)
- Middle East (Bahrain)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- AWS GovCloud (US-West)
- AWS GovCloud (US-East)

CodeCommit has added support for the Federal Information Processing Standard (FIPS) Publication 140-2 government standard in some regions. For more information about FIPS and FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2 Overview. For Git connection endpoints that support FIPS, see Git connection endpoints.

For more information about regional endpoints for AWS CLI, service, and API calls to CodeCommit, see AWS CodeCommit Endpoints and Quotas.

Git connection endpoints

Use the following URLs when you configure Git connections to CodeCommit repositories:

Git connection endpoints for AWS CodeCommit

Region name	Region	Endpoint URL	Protocol
US East (Ohio)	us-east-2	https://git-codecommit.us-east-2.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	ssh://git-codecommit.us-east-2.amazonaws.com	SSH
US East (Ohio)	us-east-2	https://git-codecommit-fips.us-east-2.amazonaws.com	HTTPS
US East (N. Virginia)	us-east-1	https://git-codecommit.us-east-1.amazonaws.com	HTTPS
US East (N. Virginia)	us-east-1	ssh://git-codecommit.us-east-1.amazonaws.com	SSH
US East (N. Virginia)	us-east-1	https://git-codecommit-fips.us-east-1.amazonaws.com	HTTPS

Region name	Region	Endpoint URL	Protocol
US West (Oregon)	us-west-2	https://git-codecommit.us-west-2.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	ssh://git-codecommit.us-west-2.amazonaws.com	SSH
US West (Oregon)	us-west-2	https://git-codecommit-fips.us-west-2.amazonaws.com	HTTPS
US West (N. California)	us-west-1	https://git-codecommit.us-west-1.amazonaws.com	HTTPS
US West (N. California)	us-west-1	ssh://git-codecommit.us-west-1.amazonaws.com	SSH
US West (N. California)	us-west-1	https://git-codecommit-fips.us-west-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	https://git-codecommit.eu-west-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	ssh://git-codecommit.eu-west-1.amazonaws.com	SSH
Asia Pacific (Tokyo)	ap-northeast-1	https://git-codecommit.ap-northeast-1.amazonaws.com	HTTPS

Region name	Region	Endpoint URL	Protocol
Asia Pacific (Tokyo)	ap-northeast-1	ssh://git-codecommit.ap-northeast-1.amazonaws.com	SSH
Asia Pacific (Singapore)	ap-southeast-1	https://git-codecommit.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	ssh://git-codecommit.ap-southeast-1.amazonaws.com	SSH
Asia Pacific (Sydney)	ap-southeast-2	https://git-codecommit.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	ssh://git-codecommit.ap-southeast-2.amazonaws.com	SSH
Asia Pacific (Jakarta)	ap-southeast-3	https://git-codecommit.ap-southeast-3.amazonaws.com	HTTPS
Asia Pacific (Jakarta)	ap-southeast-3	ssh://git-codecommit.ap-southeast-3.amazonaws.com	SSH
Middle East (UAE)	me-central-1	https://git-codecommit.me-central-1.amazonaws.com	HTTPS
Middle East (UAE)	me-central-1	ssh://git-codecommit.me-central-1.amazonaws.com	SSH

Region name	Region	Endpoint URL	Protocol
Europe (Frankfurt)	eu-central-1	https://git-codecommit.eu-central-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	ssh://git-codecommit.eu-central-1.amazonaws.com	SSH
Asia Pacific (Seoul)	ap-northeast-2	https://git-codecommit.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	ssh://git-codecommit.ap-northeast-2.amazonaws.com	SSH
South America (São Paulo)	sa-east-1	https://git-codecommit.sa-east-1.amazonaws.com	HTTPS
South America (São Paulo)	sa-east-1	ssh://git-codecommit.sa-east-1.amazonaws.com	SSH
Europe (London)	eu-west-2	https://git-codecommit.eu-west-2.amazonaws.com	HTTPS
Europe (London)	eu-west-2	ssh://git-codecommit.eu-west-2.amazonaws.com	SSH
Asia Pacific (Mumbai)	ap-south-1	https://git-codecommit.ap-south-1.amazonaws.com	HTTPS

Region name	Region	Endpoint URL	Protocol
Asia Pacific (Mumbai)	ap-south-1	ssh://git-codecommit.ap-south-1.amazonaws.com	SSH
Asia Pacific (Hyderabad)	ap-south-2	https://git-codecommit.ap-south-2.amazonaws.com	HTTPS
Asia Pacific (Hyderabad)	ap-south-2	ssh://git-codecommit.ap-south-2.amazonaws.com	SSH
Canada (Central)	ca-central-1	https://git-codecommit.ca-central-1.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	ssh://git-codecommit.ca-central-1.amazonaws.com	SSH
Canada (Central)	ca-central-1	https://git-codecommit-fips.ca-central-1.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	https://git-codecommit.eu-west-3.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	ssh://git-codecommit.eu-west-3.amazonaws.com	SSH
AWS GovCloud (US-West)	us-gov-west-1	https://git-codecommit.us-gov-west-1.amazonaws.com	HTTPS

Region name	Region	Endpoint URL	Protocol
AWS GovCloud (US-West)	us-gov-west-1	ssh://git-codecommit.us-gov-west-1.amazonaws.com	SSH
AWS GovCloud (US-West)	us-gov-west-1	https://git-codecommit-fips.us-gov-west-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	https://git-codecommit.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	ssh://git-codecommit.us-gov-east-1.amazonaws.com	SSH
AWS GovCloud (US-East)	us-gov-east-1	https://git-codecommit-fips.us-gov-east-1.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	https://git-codecommit.eu-north-1.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	ssh://git-codecommit.eu-north-1.amazonaws.com	SSH
Middle East (Bahrain)	me-south-1	https://git-codecommit.me-south-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	ssh://git-codecommit.me-south-1.amazonaws.com	SSH

Region name	Region	Endpoint URL	Protocol
Asia Pacific (Hong Kong)	ap-east-1	https://git-codecommit.ap-east-1.amazonaws.com	HTTPS
Asia Pacific (Hong Kong)	ap-east-1	ssh://git-codecommit.ap-east-1.amazonaws.com	SSH
China (Beijing)	cn-north-1	https://git-codecommit.cn-north-1.amazonaws.com.cn	HTTPS
China (Beijing)	cn-north-1	ssh://git-codecommit.cn-north-1.amazonaws.com.cn	SSH
China (Ningxia)	cn-northwest-1	https://git-codecommit.cn-northwest-1.amazonaws.com.cn	HTTPS
China (Ningxia)	cn-northwest-1	ssh://git-codecommit.cn-northwest-1.amazonaws.com.cn	SSH
Europe (Milan)	eu-south-1	https://git-codecommit.eu-south-1.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	ssh://git-codecommit.eu-south-1.amazonaws.com	SSH
Asia Pacific (Osaka)	ap-northeast-3	https://git-codecommit.ap-northeast-3.amazonaws.com	HTTPS

Region name	Region	Endpoint URL	Protocol
Asia Pacific (Osaka)	ap-northeast-3	ssh://git-codecommit.ap-northeast-3.amazonaws.com	SSH
Africa (Cape Town)	af-south-1	https://git-codecommit.af-south-1.amazonaws.com	HTTPS
Africa (Cape Town)	af-south-1	ssh://git-codecommit.af-south-1.amazonaws.com	SSH
Israel (Tel Aviv)	il-central-1	https://git-codecommit.il-central-1.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	ssh://git-codecommit.il-central-1.amazonaws.com	SSH

Server fingerprints for CodeCommit

The following table lists the public fingerprints for Git connection endpoints in CodeCommit. These server fingerprints are displayed as part of the verification process for adding an endpoint to your known hosts file.

Public fingerprints for CodeCommit

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-east-2.amazonaws.com	MD5	`a9:6d:03:ed:08:42:21:be:06:e1:e0:2a:d1:75:31:5e`

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-east-2.amazonaws.com	SHA256	3lBlW2g5xn/NA2Ck6dyeJIrQOWvn7n8UEs56fG6ZIzQ
git-codecommit.us-east-1.amazonaws.com	MD5	a6:9c:7d:bc:35:f5:d4:5f:8b:ba:6f:c8:bc:d4:83:84
git-codecommit.us-east-1.amazonaws.com	SHA256	eLMY1j0DKA4uvDZcl/KgtIayZANwX6t8+8isPtotBoY
git-codecommit.us-west-2.amazonaws.com	MD5	a8:68:53:e3:99:ac:6e:d7:04:7e:f7:92:95:77:a9:77
git-codecommit.us-west-2.amazonaws.com	SHA256	0pJx9SQpkbPUAHwy58UVIq0IHcyo1fwCpOOuVgcAWPo
git-codecommit.eu-west-1.amazonaws.com	MD5	93:42:36:ea:22:1f:f1:0f:20:02:4a:79:ff:ea:12:1d
git-codecommit.eu-west-1.amazonaws.com	SHA256	tKjRkOL8dmJyTmSbeSdN1S8F/f0iql3RlvqgTOP1UyQ
git-codecommit.ap-northeast-1.amazonaws.com	MD5	8e:a3:f0:80:98:48:1c:5c:6f:59:db:a7:8f:6e:c6:cb
git-codecommit.ap-northeast-1.amazonaws.com	SHA256	Xk/WeYD/K/bnBybzhiuu4dWpBJtXPf7E30jHU7se4Ow

Server	Cryptographic hash type	Fingerprint
git-codecommit.ap-southeast-1.amazonaws.com	MD5	65:e5:27:c3:09:68:0d:8e:b7:6d:94:25:80:3e:93:cf
git-codecommit.ap-southeast-1.amazonaws.com	SHA256	ZIsVa7OVzxrTIf+Rk4UbhPv6Es22mSB3uTBojfPXIno
git-codecommit.ap-southeast-2.amazonaws.com	MD5	7b:d2:c1:24:e6:91:a5:7b:fa:c1:0c:35:95:87:da:a0
git-codecommit.ap-southeast-2.amazonaws.com	SHA256	nYp+gHas80HY3DqbP4yanCDFhqDVjseefVbHEXqH2Ec
git-codecommit.ap-southeast-3.amazonaws.com	MD5	64:d9:e0:53:19:4f:a8:91:9a:c3:53:22:a6:a8:ed:a6
git-codecommit.ap-southeast-3.amazonaws.com	SHA256	ATdkGSFhpqIu7RqUVT/1RZo6MLxxxUW9NoDVMbAc/6g
git-codecommit.me-central-1.amazonaws.com	MD5	bd:fa:e2:f9:05:84:d6:39:6f:bc:d6:8d:fe:de:61:76
git-codecommit.me-central-1.amazonaws.com	SHA256	grceUDWubo4MzG1NoaKZKUfrgPvfN3ijliOnQr1lTZA
git-codecommit.eu-central-1.amazonaws.com	MD5	74:5a:e8:02:fc:b2:9c:06:10:b4:78:84:65:94:22:2d

Server	Cryptographic hash type	Fingerprint
git-codecommit.eu-central-1.amazonaws.com	SHA256	MwGrkiEki8QkkBtlAgXbYt0hoZYBnZF62VY5RzGJEUY
git-codecommit.ap-northeast-2.amazonaws.com	MD5	9f:68:48:9b:5f:fc:96:69:39:45:58:87:95:b3:69:ed
git-codecommit.ap-northeast-2.amazonaws.com	SHA256	eegAPQrWY9YsYo9ZHIKOmxetfXBHzAZd8Eya53Qcwko
git-codecommit.sa-east-1.amazonaws.com	MD5	74:99:9d:ff:2b:ef:63:c6:4b:b4:6a:7f:62:c5:4b:51
git-codecommit.sa-east-1.amazonaws.com	SHA256	kW+VKB0jpRaG/ZbXkgbtMQbKgEDK7JnISV3SVoyCmzU
git-codecommit.us-west-1.amazonaws.com	MD5	3b:76:18:83:13:2c:f8:eb:e9:a3:d0:51:10:32:e7:d1
git-codecommit.us-west-1.amazonaws.com	SHA256	gzauWTWXDK2u5KuMMi5vbKTmfyerdIwgSbzYBODLpzg
git-codecommit.eu-west-2.amazonaws.com	MD5	a5:65:a6:b1:84:02:b1:95:43:f9:0e:de:dd:ed:61:d3
git-codecommit.eu-west-2.amazonaws.com	SHA256	r0Rwz5k/IHp/QyrRnfiM9j02D5UEqMbtFNTuDG2hNbs

Server	Cryptographic hash type	Fingerprint
git-codecommit.ap-south-1.amazonaws.com	MD5	da:41:1e:07:3b:9e:76:a0:c5:1e:64:88:03:69:86:21
git-codecommit.ap-south-1.amazonaws.com	SHA256	hUKwnTj7+Xpx4Kddb6p45j4RazIJ4IhAMD8k29itOfE
git-codecommit.ap-south-2.amazonaws.com	MD5	bc:cc:9f:15:f8:f3:58:a2:68:65:21:e2:23:71:8d:ce
git-codecommit.ap-south-2.amazonaws.com	SHA256	Xe0CyZEOvgR5Xa2YUGqf+jn8/Ut7l7nX/CmslSFNEig
git-codecommit.ca-central-1.amazonaws.com	MD5	9f:7c:a2:2f:8c:b5:74:fd:ab:b7:e1:fd:af:46:ed:23
git-codecommit.ca-central-1.amazonaws.com	SHA256	Qz5puafQdANVprLlj6r0Qyh4lCNsF6ob61dGcPtFS7w
git-codecommit.eu-west-3.amazonaws.com	MD5	1b:7f:97:dd:d7:76:8a:32:2c:bd:2c:7b:33:74:6a:76
git-codecommit.eu-west-3.amazonaws.com	SHA256	uw7c2FL564jVoFgtc+ikzILnKBsZz7t9+CFdSJjKbLI
git-codecommit.us-gov-west-1.amazonaws.com	MD5	9f:6c:19:3b:88:cd:e8:88:1b:9c:98:6a:95:31:8a:69

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-gov-west-1.amazonaws.com	SHA256	djXQoSIFcg8vHe0KVH1xW/gOF9X37tWTqu4Hkng75x4
git-codecommit.us-gov-east-1.amazonaws.com	MD5	00:8d:b5:55:6f:05:78:05:ed:ea:cb:3f:e6:f0:62:f2
git-codecommit.us-gov-east-1.amazonaws.com	SHA256	fVb+R0z7qW7minenW+rUpAABRCRBTCzmETAJEQrg98
git-codecommit.eu-north-1.amazonaws.com	MD5	8e:53:d8:59:35:88:82:fd:73:4b:60:8a:50:70:38:f4
git-codecommit.eu-north-1.amazonaws.com	SHA256	b6KSK7xKq+V8jl7iuAcjqXsG7zkqoUZZmmhYYFBq1wQ
git-codecommit.me-south-1.amazonaws.com	MD5	0e:39:28:56:d5:41:e6:8d:fa:81:45:37:fb:f3:cd:f7
git-codecommit.me-south-1.amazonaws.com	SHA256	O+NToCGgjrHekiBuOl0ad7ROGEsz+DBLXOd/c9wc0JU
git-codecommit.ap-east-1.amazonaws.com	MD5	a8:00:3d:24:52:9d:61:0e:f6:e3:88:c8:96:01:1c:fe
git-codecommit.ap-east-1.amazonaws.com	SHA256	LafadYwUYW8hONoTRpojbjNs9IRnbEwHtezD3aAIBX0

Server	Cryptographic hash type	Fingerprint
git-codecommit.cn-north-1.amazonaws.com.cn	MD5	11:7e:2d:74:9e:3b:94:a2:69:14:75:6f:5e:22:3b:b3
git-codecommit.cn-north-1.amazonaws.com.cn	SHA256	IYUXxH2OpTDsyYMLIp+JY8CTLS4UX+ZC5JVZXPRaxc8
git-codecommit.cn-northwest-1.amazonaws.com.cn	MD5	2e:a7:fb:4c:33:ac:6c:f9:aa:f2:bc:fb:0a:7b:1e:b6
git-codecommit.cn-northwest-1.amazonaws.com.cn	SHA256	wqjd6eHd0+mOBx+dCNuL0omUoCNjaDtZiEpWj5TmCfQ
git-codecommit.eu-south-1.amazonaws.com	MD5	b9:f6:5d:e2:48:92:3f:a9:37:1e:c4:d0:32:0e:fb:11
git-codecommit.eu-south-1.amazonaws.com	SHA256	lyXrWbCg3uQmJrl1XxB/ASR7ugW1Ysf5yzYOJbudHsI
git-codecommit.ap-northeast-3.amazonaws.com	MD5	25:17:40:da:b9:d4:18:c3:b6:b3:fb:ed:1c:20:fe:29
git-codecommit.ap-northeast-3.amazonaws.com	SHA256	2B815B9F0AvwLnRxSVxUz4kDYmtEQUGGdQYP8OQLXhA
git-codecommit.af-south-1.amazonaws.com	MD5	21:a0:ba:d7:c1:d1:b5:39:98:8d:4d:7c:96:f5:ca:29

Server	Cryptographic hash type	Fingerprint
git-codecommit.af-south-1.amazonaws.com	SHA256	C34ji3x/cnsDZjUpyNGXdE5pjHYimqJrQZ3leTgqJHM
git-codecommit.il-central-1.amazonaws.com	MD5	04:74:89:16:98:7a:61:b1:69:46:42:3c:d1:b4:ac:a9
git-codecommit.il-central-1.amazonaws.com	SHA256	uFxhp51kUWhleTLeYbxQVYm4RnNLNZ5Dbdm1cgdSl/8

Using AWS CodeCommit with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and CodeCommit. You can use this connection to enable CodeCommit to communicate with your resources on your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. With VPC endpoints, the routing between the VPC and AWS services is handled by the AWS network, and you can use IAM policies to control access to service resources.

To connect your VPC to CodeCommit, you define an *interface VPC endpoint* for CodeCommit. An interface endpoint is an elastic network interface with a private IP address that serves as an entry point for traffic destined to a supported AWS service. The endpoint provides reliable, scalable connectivity to CodeCommit without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see What Is Amazon VPC in the *Amazon VPC User Guide*.

> **Note**
>
> Other AWS services that provide VPC support and integrate with CodeCommit, such as AWS CodePipeline, might not support using Amazon VPC endpoints for that integration.

For example, traffic between CodePipeline and CodeCommit cannot be restricted to the VPC subnet range. Services that do support integration, such as AWS Cloud9, might require additional services such as AWS Systems Manager.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see AWS PrivateLink.

The following steps are for users of Amazon VPC. For more information, see Getting Started in the *Amazon VPC User Guide*.

Availability

CodeCommit currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Frankfurt)
- Europe (Stockholm)
- Europe (Milan)
- Africa (Cape Town)
- Israel (Tel Aviv)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Jakarta)
- Middle East (UAE)
- Asia Pacific (Seoul)

- Asia Pacific (Osaka)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Hong Kong)
- South America (São Paulo)
- Middle East (Bahrain)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- AWS GovCloud (US-West)
- AWS GovCloud (US-East)

Create VPC endpoints for CodeCommit

To start using CodeCommit with your VPC, create an interface VPC endpoint for CodeCommit. CodeCommit requires separate endpoints for Git operations and for CodeCommit API operations. Depending on your business needs, you might need to create more than one VPC endpoint. When you create a VPC endpoint for CodeCommit, choose **AWS Services**, and in **Service Name**, choose from the following options:

- **com.amazonaws.*region*.git-codecommit**: Choose this option if you want to create a VPC endpoint for Git operations with CodeCommit repositories. For example, choose this option if your users use a Git client and commands such as `git pull`, `git commit`, and `git push` when they interact with CodeCommit repositories.
- **com.amazonaws.*region*.git-codecommit-fips**: Choose this option if you want to create a VPC endpoint for Git operations with CodeCommit repositories that complies with the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard.

  > **Note**
  >
  > FIPS endpoints for Git are not available in all AWS Regions. For more information, see Git connection endpoints.

- **com.amazonaws.*region*.codecommit**: Choose this option if you want to create a VPC endpoint for CodeCommit API operations. For example, choose this option if your users use the AWS CLI,

the CodeCommit API, or the AWS SDKs to interact with CodeCommit for operations such as `CreateRepository`, `ListRepositories`, and `PutFile`.

- **com.amazonaws.*region*.codecommit-fips**: Choose this option if you want to create a VPC endpoint for CodeCommit API operations that complies with the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard.

> **Note**
>
> FIPS endpoints are not available in all AWS Regions. For more information, see the entry for AWS CodeCommit in Federal Information Processing Standard (FIPS) 140-2 Overview.

Create a VPC endpoint policy for CodeCommit

You can create a policy for Amazon VPC endpoints for CodeCommit in which you can specify:

- The principal that can perform actions.
- The actions that can be performed.
- The resources that can have actions performed on them.

For example, a company might want to restrict access to repositories to the network address range for a VPC. You can view an example of this kind of policy here: Example 3: Allow a user connecting from a specified IP address range access to a repository. The company configured two Git VPC endpoints for the US East (Ohio) region: `com.amazonaws.us-east-2.codecommit` and `com-amazonaws.us-east-2.git-codecommit-fips`. They want to allow code pushes to a CodeCommit repository named *MyDemoRepo* only on the FIPS-compliant endpoint only. To enforce this, they would configure a policy similar to the following on the `com.amazonaws.us-east-2.codecommit` endpoint that specifically denies Git push actions:

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
```

```
            "Action": "codecommit:GitPush",
            "Effect": "Deny",
            "Resource": "arn:aws:codecommit:us-east-2:123456789012:MyDemoRepo",
            "Principal": "*"
        }
    ]
}
```

Important

The global condition key `aws:VpcSourceIp` is not supported CodeCommit repositories in IAM policies for `git push` commands.

For more information, see Creating an Interface Endpoint in the *Amazon VPC User Guide*.

Quotas in AWS CodeCommit

The following table describes quotas in CodeCommit. For information about quotas that can be changed, see AWS CodeCommit Endpoints and Quotas. For information about requesting a service quota increase, see AWS Service Quotas. For information about required versions of Git and other software, see Compatibility for CodeCommit, Git, and other components.

Approval rule and approval rule template names	Any combination of letters, numbers, periods, spaces, underscores, and dashes between 1 and 100 characters in length. Names are case sensitive. Names cannot end in .git and cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :`
Approval rule content length	3000 characters
Approval rule template description length	1000 characters
Approval rule template destination references	100
Approval rule templates	1000 in an AWS Region

Approval rules on a pull request	Up to 30 maximum. Up to 25 of these can be from approval rule templates.
Approval rules on a pull request created from an approval rule template	25
Approvals on a pull request	200
Approvers in an approval pool	50
Branch names	Any combination of allowed characters between 1 and 256 characters in length, except that branch names of exactly 40 hexadecimal characters are not allowed. Branch names cannot: • begin or end with a slash (/) or period (.) • consist of the single character @ • contain two or more consecutive periods (..), forward slashes (//), or the following character combination: @{ • contain spaces or any of the following characters: ? ^ * [\ ~ : Branch names are references. Many of the limitations on branch names are based on the Git reference standard. For more information, see Git Internals and git-check-ref-format.
Comment length	Maximum of 10,240 characters.
Compressed archive size for initializing a CodeCommit repository in AWS CloudFormation	Maximum of 20 GB when compressed.

Custom data for triggers	This is a string field limited to 1,000 characters. It cannot be used to pass any dynamic parameters.
Display in the console	A file or a comparison between files might not be viewable in the console if: • The file is greater than 2 MB • The file contains more than 25,000 characters in a single line • A comparison contains more than 6,500 total lines of differences
Email addresses in commits made in the console	Any combination of allowed characters between 1 and 256 characters in length. Email addresses are not validated.

File paths	Any combination of allowed characters between 1 and 4,096 characters in length. File paths must be an unambiguous name that specifies the file and the exact location of the file. File paths cannot exceed 20 directories in depth. In addition, file paths cannot:
	• contain empty strings
	• be a relative file path
	• include any of the following character combinations:
	`/./`
	`/../`
	`//`
	• end with a trailing slash or backslash
	
	File names and paths must be fully qualified . The name and path to a file on your local computer must follow the standards for that operating system. When specifying the path to a file in a CodeCommit repository, use the standards for Amazon Linux.

File size	Maximum of 6 MB for any individual file when using the CodeCommit console, APIs, or the AWS CLI.

Git blob size	Maximum of 2 GB. **Note** There is no limit on the number or the total size of all files in a single commit, as long as the metadata does not exceed 6 MB and a single blob does not exceed 2 GB.
Graph display of branches in the Commit Visualizer	35 per page. If there are more than 35 branches on a single page, the graph is not displayed.
Metadata for a commit	Maximum of 20 MB for the combined metadata for a commit (for example, the combination of author information, date, parent commit list, and commit messages) when using the CodeCommit console, APIs, or the AWS CLI. **Note** There is no limit on the number or the total size of all files in a single commit, as long as the metadata does not exceed 6 MB, an individual file does not exceed 6 MB, and a single blob does not exceed 2 GB.
Number of files in a commit	Maximum of 100.
Number of open pull requests	Maximum of 1,000.
Number of references in a single push	Maximum of 4,000, including create, delete, and update. There is no limit on the overall number of references in the repository.

Number of repositories	Maximum of 5,000 per Amazon Web Services account. This limit can be changed. For more information, see AWS CodeCommit Endpoints and Quotas and AWS Service Quotas.
Number of triggers in a repository	Maximum of 10.

Regions	CodeCommit is available in the following AWS Regions: • US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Europe (Ireland) • Europe (London) • Europe (Paris) • Europe (Frankfurt) • Europe (Stockholm) • Europe (Milan) • Africa (Cape Town) • Israel (Tel Aviv) • Asia Pacific (Tokyo) • Asia Pacific (Singapore) • Asia Pacific (Sydney) • Asia Pacific (Jakarta) • Middle East (UAE) • Asia Pacific (Seoul) • Asia Pacific (Osaka) • Asia Pacific (Mumbai) • Asia Pacific (Hyderabad) • Asia Pacific (Hong Kong) • South America (São Paulo) • Middle East (Bahrain) • Canada (Central) • China (Beijing) • China (Ningxia) • AWS GovCloud (US-West)

	• AWS GovCloud (US-East) For more information, see Regions and Git connection endpoints.
Repository descriptions	Any combination of characters between 0 and 1,000 characters in length. Repository descriptions are optional.
Repository names	Any combination of letters, numbers, periods, underscores, and dashes between 1 and 100 characters in length. Names are case sensitive . Repository names cannot end in .git and cannot contain any of the following characters: `! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :`
Repository tag key names	Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 128 characters in length. Allowed characters are `+ - = . _ : / @` Tag key names must be unique, and each key can only have one value. A tag cannot: • begin with `aws:` • consist only of spaces • end with a space • contain emojis or any of the following characters: `? ^ * [ \ ~ ! # $ % & * ( ) > < | " ' ` [ ] { } ;`

Repository tag values	Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 256 characters in length. Allowed characters are `+ - = . _ : / @` A key can only have one value, but many keys can have the same value. A tag cannot: • consist only of spaces • end with a space • contain emojis or any of the following characters: `? ^ * [ \ ~ ! # $ % & * ( ) > < | " ' ` [ ] { } ;`
Repository tags	Tags are case sensitive. Maximum of 50 per resource. Tag names of exactly 40 hexadecim al characters are not allowed.
Trigger names	Any combination of letters, numbers, periods, underscores, and dashes between 1 and 100 characters in length. Trigger names cannot contain spaces or commas.
User names in commits made in the console	Any combination of allowed characters between 1 and 1,024 characters in length.

AWS CodeCommit command line reference

This reference helps you learn how to use the AWS CLI.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with CodeCommit from the command line. We recommend that you install AWS CLI version 2. It is the most recent major version of the AWS CLI and supports all of the latest features. It is the only version of the AWS CLI that supports using a root account, federated access, or temporary credentials with **git-remote-codecommit**.

For more information, see Getting Set Up with the AWS Command Line Interface.

> **Note**
>
> CodeCommit works only with AWS CLI versions 1.7.38 and later. As a best practice, install or upgrade the AWS CLI to the latest version available. To determine which version of the AWS CLI you have installed, run the **aws --version** command.
> To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify that the CodeCommit commands for the AWS CLI are installed.

```
aws codecommit help
```

This command returns a list of CodeCommit commands.

3. Configure the AWS CLI with a profile by using the **configure** command, as follows:.

```
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user to use with CodeCommit. Also, be sure to specify the AWS Region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example, if you are configuring a profile for an IAM user:

```
AWS Access Key ID [None]: Type your IAM user AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your IAM user AWS secret access key here, and
 then press Enter
Default region name [None]: Type a supported region for CodeCommit here, and then
 press Enter
Default output format [None]: Type json here, and then press Enter
```

For more information about creating and configuring profiles to use with the AWS CLI, see the following:

- Named Profiles
- Using an IAM Role in the AWS CLI

- Set command
- Connecting to AWS CodeCommit repositories with rotating credentials

To connect to a repository or a resource in another AWS Region, you must reconfigure the AWS CLI with the default Region name. Supported default Region names for CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- ap-southeast-3
- me-central-1
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1
- eu-north-1
- ap-east-1
- me-south-1
- cn-north-1
- cn-northwest-1
- eu-south-1

- ap-northeast-3
- af-south-1
- il-central-1

For more information about CodeCommit and AWS Region, see Regions and Git connection endpoints. For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users. For more information about the AWS CLI and profiles, see Named Profiles.

To view a list of all available CodeCommit commands, run the following command:

```
aws codecommit help
```

To view information about a CodeCommit command, run the following command, where *command-name* is the name of the command (for example, **create-repository**):

```
aws codecommit command-name help
```

See the following for descriptions and example usage of the commands in the AWS CLI:

- associate-approval-rule-template-with-repository
- batch-associate-approval-rule-template-with-repositories
- batch-disassociate-approval-rule-template-from-repositories
- batch-describe-merge-conflicts
- batch-get-commits
- batch-get-repositories
- create-approval-rule-template
- create-branch
- create-commit
- create-pull-request
- create-pull-request-approval-rule
- create-repository
- create-unreferenced-merge-commit

- delete-approval-rule-template
- delete-branch
- delete-comment-content
- delete-file
- delete-repository
- describe-merge-conflicts
- delete-pull-request-approval-rule
- describe-pull-request-events
- disassociate-pull-request-approval-rule-template-from-repository
- evaluate-pull-request-approval-rules
- get-approval-rule-template
- get-blob
- get-blob-differences
- get-branch
- get-comment
- get-comment-reactions
- get-comments-for-compared-commit
- get-comments-for-pull-request
- get-commit
- get-differences
- get-merge-commit
- get-merge-conflicts
- get-merge-options
- get-pull-request
- get-pull-request-approval-states
- get-pull-request-override-state
- get-repository
- get-repository-triggers
- list-approval-rule-templates

- list-associated-approval-rule-templates-for-repository
- list-branches
- list-pull-requests
- list-repositories
- list-repositories-for-approval-rule-template
- list-tags-for-resource
- merge-branches-by-fast-forward
- merge-branches-by-squash
- merge-branches-by-three-way
- merge-pull-request-by-fast-forward
- merge-pull-request-by-squash
- merge-pull-request-by-three-way
- override-pull-request-approval-rules
- post-comment-for-compared-commit
- post-comment-for-pull-request
- post-comment-reply
- put-comment-reaction
- put-file
- put-repository-triggers
- tag-resource
- test-repository-triggers
- untag-resource
- update-approval-rule-template-content
- update-approval-rule-template-description
- update-approval-rule-template-name
- update-comment
- update-default-branch
- update-pull-request-approval-rule-content
- update-pull-request-approval-state

- update-pull-request-description
- update-pull-request-status
- update-pull-request-title
- update-repository-description
- update-repository-name

Basic Git commands

You can use Git to work with a local repo and the CodeCommit repository to which you've connected the local repo.

The following are some basic examples of frequently used Git commands.

For more options, see your Git documentation.

Topics

- Configuration variables
- Remote repositories
- Commits
- Branches
- Tags

Configuration variables

Lists all configuration variables.	`git config --list`
Lists only local configuration variables.	`git config --local -l`
Lists only system configuration variables.	`git config --system -l`
Lists only global configuration variables.	`git config --global -l`
Sets a configuration variable in the specified configuration file.	`git config [--local | --global | --system] variable-name variable-value`

Sets the default branch name to *main* for all local repositories when an initial commit is made to a repository that does not yet have a default branch	**git config --global init.defaultBranch main**
Edits a configuration file directly. Can also be used to discover the location of a specific configuration file. To exit edit mode, typically you type `:q` (to exit without saving changes) or `:wq` (to save changes and then exit), and then press Enter.	`git config [--local | --global | --system] --edit`

Remote repositories

Initializes a local repo in preparation for connecting it to an CodeCommit repository.	`git init`
Can be used to set up a connection between a local repo and a remote repository (such as a CodeCommit repository) using the specified nickname the local repo has for the CodeCommit repository and the specified URL to the CodeCommit repository.	`git remote add` *`remote-name`* *`remote-url`*
Creates a local repo by making a copy of a CodeCommit repository at the specified URL, in the specified subfolder of the current folder on the local machine. This command also creates a remote tracking branch for each branch in the cloned CodeCommit repositor y and creates and checks out an initial branch that is forked from the current default branch in the cloned CodeCommit repository.	`git clone` *`remote-url`* *`local-sub folder-name`*
Shows the nickname the local repo uses for the CodeCommit repository.	`git remote`

Shows the nickname and the URL the local repo uses for fetches and pushes to the CodeCommit repository.	`git remote -v`
Pushes finalized commits from the local repo to the CodeCommit repository, using the specified nickname the local repo has for the CodeCommit repository and the specified branch. Also sets up upstream tracking information for the local repo during the push.	`git push -u` *`remote-name branch-name`*
Pushes finalized commits from the local repo to the CodeCommit repository after upstream tracking information is set.	`git push`
Pulls finalized commits to the local repo from the CodeCommit repository, using the specified nickname the local repo has for the CodeCommit repository and the specified branch	`git pull` *`remote-name branch-name`*
Pulls finalized commits to the local repo from the CodeCommit repository after upstream tracking information is set.	`git pull`
Disconnects the local repo from the CodeCommit repository, using the specified nickname the local repo has for the CodeCommit repository.	`git remote rm` *`remote-name`*

Commits

Shows what has or hasn't been added to the pending commit in the local repo.	`git status`

Shows what has or hasn't been added to the pending commit in the local repo in a concise format. (`M` = modified, `A` = added, `D` = deleted, and so on)	`git status -sb`
Shows changes between the pending commit and the latest commit in the local repo.	`git diff HEAD`
Adds specific files to the pending commit in the local repo.	`git add [`*`file-name-1 file-name-2 file-name-N`*` | `*`file-pattern`*` ]`
Adds all new, modified, and deleted files to the pending commit in the local repo.	`git add`
Begins finalizing the pending commit in the local repo, which displays an editor to provide a commit message. After the message is entered, the pending commit is finalized.	`git commit`
Finalizes the pending commit in the local repo, including specifying a commit message at the same time.	`git commit -m "`*`Some meaningful commit comment`*`"`
Lists recent commits in the local repo.	`git log`
Lists recent commits in the local repo in a graph format.	`git log --graph`
Lists recent commits in the local repo in a predefined condensed format.	`git log --pretty=oneline`
Lists recent commits in the local repo in a predefined condensed format, with a graph.	`git log --graph --pretty=oneline`

Lists recent commits in the local repo in a custom format, with a graph. (For more options, see Understanding History: Commits in the Git documentation.)	`git log --graph --pretty=format:"%H (%h) : %cn : %ar : %s"`

Branches

Lists all branches in the local repo with an asterisk (*) displayed next to your current branch.	`git branch`
Pulls information about all existing branches in the CodeCommit repository to the local repo.	`git fetch`
Lists all branches in the local repo and remote tracking branches in the local repo.	`git branch -a`
Lists only remote tracking branches in the local repo.	`git branch -r`
Creates a new branch in the local repo using the specified branch name.	`git branch` *`new-branch-name`*
Switches to another branch in the local repo using the specified branch name.	`git checkout` *`other-branch-name`*
Creates a new branch in the local repo using the specified branch name, and then switches to it.	`git checkout -b` *`new-branch-name`*
Pushes a new branch from the local repo to the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository and the specified branch name. Also sets up upstream tracking	`git push -u` *`remote-name new-branch-name`*

information for the branch in the local repo during the push.	
Creates a new branch in the local repo using the specified branch name. Then connects the new branch in the local repo to an existing branch in the CodeCommit repository, using the specified nickname the local repo has for the CodeCommit repository and the specified branch name.	`git branch --track` *`new-branch-name remote-name /remote-branch-name`*
Merges changes from another branch in the local repo to the current branch in the local repo.	`git merge` *`from-other-branch-name`*
Deletes a branch in the local repo unless it contains work that has not been merged.	`git branch -d` *`branch-name`*
Deletes a branch in the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository and the specified branch name. (Note the use of the colon (`:`).)	`git push` *`remote-name`* `:`*`branch-name`*

Tags

Lists all tags in the local repo.	`git tag`
Pulls all tags from the CodeCommit repository to the local repo.	`git fetch --tags`
Shows information about a specific tag in the local repo.	`git show` *`tag-name`*
Creates a "lightweight" tag in the local repo.	`git tag` *`tag-name commit-id-to-point-tag-at`*

Pushes a specific tag from the local repo to the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository and the specified tag name.	`git push` *`remote-name`* *`tag-name`*
Pushes all tags from the local repo to the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository.	`git push` *`remote-name`* `--tags`
Deletes a tag in the local repo.	`git tag -d` *`tag-name`*
Deletes a tag in the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository and the specified tag name. (Note the use of the colon (`:`).)	`git push` *`remote-name`* `:`*`tag-name`*

AWS CodeCommit User Guide document history

The following table describes important changes to the documentation for CodeCommit. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version:** 2015-04-13

Change	Description	Date
AWS CodeCommit is now available to new customers.	AWS CodeCommit is now available to new customers . Existing customers of AWS CodeCommit can continue to use the service as normal.	November 25, 2025
AWS CodeCommit is no longer available to new customers.	AWS CodeCommit is no longer available to new customers. Existing customers of AWS CodeCommit can continue to use the service as normal. Learn more"	July 25, 2024
Updated troubleshooting information for the credential helper	Added information about troubleshooting errors caused by incomplete or misconfig ured .gitconfig files.	July 9, 2024
Updated troubleshooting information for `git-remote-codecommit`	Added additional information about troubleshooting errors with externally managed environments.	July 9, 2024
Removed references to AWS Toolkit for Eclipse	The AWS Toolkit for Eclipse reached end of life and is no longer supported. References to this toolkit have been removed.	July 9, 2024

CodeCommit now supports using customer managed keys	You can now use a customer managed key or a AWS managed key for encryptin g and decrypting data in a repository. For more informati on, see AWS KMS and encryption, Create a repositor y, and Change repository settings.	December 21, 2023
CodeCommit is now available in Israel (Tel Aviv);	CodeCommit is now available in Israel (Tel Aviv). For more information, see Regions and Git Connection Endpoints.	August 28, 2023
Changes to managed policies for CodeCommit	The AWSCodeCommitPower User and AWSCodeCo mmitFullAccess policies have been updated with an additional permission. For more information, see CodeCommit updates to AWS managed policies.	May 16, 2023
CodeCommit is now available in three additional AWS Regions	CodeCommit is now available in three additional AWS Regions: Asia Pacific (Jakarta) , Middle East (UAE), and Asia Pacific (Hyderabad). For more information, see Regions and Git Connection Endpoints.	February 28, 2023
CodeCommit is now available in Africa (Cape Town)	CodeCommit is now available in an additional AWS Region: Africa (Cape Town). For more information, see Regions and Git Connection Endpoints.	September 15, 2021

Changes to managed policies for CodeCommit	Details about updates to AWS managed policies for CodeCommit are now available. For more informati on, see CodeCommit updates to AWS managed policies.	August 18, 2021
CodeCommit is now available in Asia Pacific (Osaka)	CodeCommit is now available in an additional AWS Region: Asia Pacific (Osaka). For more information, see Regions and Git Connection Endpoints.	April 14, 2021
AWS CloudFormation and AWS Cloud Development Kit (AWS CDK) changes naming behavior for default branches in CodeCommit	Repositories created using CloudFormation or the AWS CDK with an initial commit of code now use the default branch name of *main*. This change does not affect existing repositories or branches. Customers who use local Git clients to create their initial commits have a default branch name that follows the configuration of those Git clients. For more information, see Creating CodeCommit resources with AWS CloudFormation.	March 4, 2021

CodeCommit changes naming behavior for default branches	As of January 19, 2021, the default branch name created by an initial commit to a CodeCommit repository is *main*. This change does not affect existing repositories or branches. Customers who use local Git clients to create their initial commits have a default branch name that follows the configuration of those Git clients. For more information, see Working with branches, Create a commit, and Change branch settings.	January 19, 2021
CodeCommit is now available in Europe (Milan)	CodeCommit is now available in an additional AWS Region: Europe (Milan). For more information, see Regions and Git Connection Endpoints.	September 16, 2020
CodeCommit adds support for emoji reactions to comments	CodeCommit now supports reacting to comments from other users with emojis. For more information, see Comment on a Commit and Review a Pull Request.	June 24, 2020
CodeCommit now available in China (Beijing) and China (Ningxia)	CodeCommit is now available in two additional AWS Regions: China (Beijing) and China (Ningxia). For more information, see Regions and Git Connection Endpoints.	April 23, 2020

CodeCommit adds support for git-remote-codecommit	CodeCommit supports connections to CodeCommit repositories over HTTPS with **git-remote-codecommit**, a utility that modifies Git. This is the recommended approach for federated or temporary access connections to CodeCommit repositories. You can also use **git-remote-codecommit** with an IAM user. **git-remote-codecommit** does not require you to set up Git credentials for the user. For more information, see Setup Steps for HTTPS Connections to AWS CodeCommit with git-remote-codecommit.	March 4, 2020
CodeCommit supports session tags	CodeCommit supports the use of session tags, which are key-value pair attributes that you pass when you assume an IAM role, use temporary credentials, or federate a user in AWS Security Token Service (AWS STS). You can use the information provided in these tags to make it easier to identify who made a change or caused an event. For more information, see Monitoring CodeCommit and Using Tags to Provide Identity Information in CodeCommit.	December 19, 2019

CodeCommit is available in Asia Pacific (Hong Kong)	You can now use CodeCommit in Asia Pacific (Hong Kong). For more information, including Git connection endpoints, see Regions.	December 11, 2019
CodeCommit supports Amazon CodeGuru Reviewer	CodeCommit supports Amazon CodeGuru Reviewer, an automated code review service that uses program analysis and machine learning to detect common issues and recommend fixes in your Java or Python code. For more information, see Associate or Disassociate a Repository with Amazon CodeGuru Reviewer and Working with Pull Requests.	December 3, 2019
CodeCommit supports approval rules	You can now use approval rules to help you customize your development workflows across repositories so that different branches have appropriate levels of approvals and controls for pull requests. For more information, see Working with Approval Rule Templates and Working with Pull Requests.	November 20, 2019

CodeCommit supports notification rules	You can now use notification rules to inform users of important changes in repositories. The functionality in this feature replaces notifications created before November 5, 2019. For more information, see Create a Notification Rule.	November 5, 2019
CodeCommit is available in Middle East (Bahrain)	You can now use CodeCommit in Middle East (Bahrain). For more information, including Git connection endpoints, see Regions.	October 30, 2019
CodeCommit adds support for retrieving information about multiple commits	You can get information about multiple commits by using the batch-get-commits command in the AWS CLI. For more information, see View Commit Details.	August 15, 2019
CodeCommit is available in Europe (Stockholm)	You can now use CodeCommit in Europe (Stockholm). For more information, including Git connection endpoints, see Regions.	July 31, 2019
CodeCommit adds support for tagging repositories in the CodeCommit console	You can now add, manage, and remove tags for a repository to help you manage your AWS resources from the CodeCommit console. For more information, see Tagging a Repository.	July 2, 2019

CodeCommit adds support for additional Git merge strategies	You can now choose between Git merge strategies when merging pull requests in CodeCommit. You can also resolve merge conflicts in the CodeCommit console. For more information, see Working with Pull Requests.	June 10, 2019
CodeCommit is available in AWS GovCloud (US-East)	You can now use CodeCommit in AWS GovCloud (US-East). For more information, including Git connection endpoints, see Regions.	May 31, 2019
CodeCommit adds support for tagging repositories	You can now add, manage, and remove tags for a repository to help you manage your AWS resources. For more information, see Tagging a Repository.	May 30, 2019
Find resources in the console	You can now quickly search for your resources, such as repositories, build projects, deployment applications, and pipelines. Choose **Go to resource** or press the / key, and then type the name of the resource. For more information, see CodeCommit Tutorial.	May 14, 2019

CodeCommit is available in AWS GovCloud (US-West)	You can now use CodeCommit in AWS GovCloud (US-West). For more information, including Git connection endpoints, see Regions.	April 18, 2019
CodeCommit adds support for Amazon VPC endpoints	You can now establish a private connection between your VPC and CodeCommit. For more information, see Using CodeCommit with Interface VPC Endpoints.	March 7, 2019
CodeCommit adds a new API	CodeCommit has added an API for creating commits. For more information, see Create a Commit.	February 20, 2019
Content update	The content in this guide has been updated with minor fixes and additional troubleshooting guidance.	January 2, 2019
Content update	The content in this guide has been updated to support the new CodeCommit console experience.	October 30, 2018

CodeCommit and the federal information processing standard (FIPS)	CodeCommit has added support for the Federal Information Processing Standard (FIPS) Publication 140-2 government standard in some regions. For more information about FIPS and FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2 Overview. For more information about Git connection endpoints, see Regions.	October 25, 2018
CodeCommit adds three APIs	CodeCommit has added three APIs to support working with files. For more information about Git connection endpoints, see Permissions for Actions on Individual Files and AWS CodeCommit API Reference.	September 27, 2018
CodeCommit documentation history notification available through RSS feed	You can now receive notification about updates to the CodeCommit documentation by subscribing to an RSS feed.	June 29, 2018

Earlier updates

The following table describes important changes to the documentation prior to June 29, 2018.

Change	Description	Date changed
New topic	The Limit pushes and merges to branches topic has been added. The CodeCommit permissions reference topic has been updated.	May 16, 2018
New section	The Working with files in AWS CodeCommit repositories section has been added. The CodeCommit permissions reference and Getting started with AWS CodeCommit topics have been updated.	February 21, 2018
New topic	The Configure cross-account access to an AWS CodeCommit repository using roles topic has been added.	February 21, 2018
New topic	The Integrate AWS Cloud9 with AWS CodeCommit topic has been added. The Product and service integrations topic has been updated with information about AWS Cloud9.	December 1, 2017
New section	The Working with pull requests in AWS CodeCommit repositories section has been added. The Authentication and access control for AWS CodeCommit section has been updated with information about permissions for pull requests and commenting. It also includes updated managed policy statements.	November 20, 2017
Updated topics	The Product and service integrations topic has been updated to include links for customers who want to update their existing pipelines to use Amazon CloudWatch Events to start pipelines in response to changes in a CodeCommit repository.	October 11, 2017
New topics	The Authentication and access control for AWS CodeCommit section has been added. It replaces the Access Permissions Reference topic.	September 11, 2017

Change	Description	Date changed
Updated topics	The Manage triggers for a repository section has been updated to reflect changes in trigger configuration. Topics and images have been updated throughout the guide to reflect changes in the navigation bar.	August 29, 2017
New topic	The Working with user preferences topic has been added. The View tag details topic has been updated. The Product and service integrations topics has been updated with information about integrating with Amazon CloudWatch Events.	August 3, 2017
New topics	The Integrate Eclipse with AWS CodeCommit and Integrate Visual Studio with AWS CodeCommit topics have been added.	June 29, 2017
Updated topic	CodeCommit is now available in two additional regions: Asia Pacific (Mumbai), and Canada (Central). The Regions and Git connection endpoints topic has been updated.	June 29, 2017
Updated topic	CodeCommit is now available in four additional regions: Asia Pacific (Seoul), South America (São Paulo), US West (N. California), and Europe (London). The Regions and Git connection endpoints topic has been updated.	June 6, 2017
Updated topic	CodeCommit is now available in four additional regions: Asia Pacific (Tokyo), Asia Pacific (Singapore), Asia Pacific (Sydney), and Europe (Frankfurt). The Regions and Git connection endpoints topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	May 25, 2017

Change	Description	Date changed
New topic	The Compare and merge branches topic has been added. The contents of the Working with branches section have been updated with information about using the CodeCommit console to work with branches in a repository.	May 18, 2017
New topic	The Compare commits topic has been added with information about comparing commits. The structure of the user guide has been updated for working with repositories, commits,, and branches.	March 28, 2017
Updated topic	The View commit details topic has been updated with information about viewing the difference between a commit and its parent in the console, and using the **get-differences** command to view differences between commits using the AWS CLI.	January 24, 2017
New topic	The Logging AWS CodeCommit API calls with AWS CloudTrail topic has been added with information about logging connections to CodeCommit using AWS CloudFormation.	January 11, 2017
New topic	The For HTTPS users using Git credentials topic has been added with information about setting up connections to CodeCommit using Git credentials over HTTPS.	December 22, 2016
Updated topic	The Product and service integrations topic has been updated to include information about integration with AWS CodeBuild.	December 5, 2016
Updated topic	CodeCommit is now available in another region, Europe (Ireland). The Regions and Git connection endpoints topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	November 16, 2016

Change	Description	Date changed
Updated topic	CodeCommit is now available in another region, US West (Oregon). The Regions and Git connection endpoints topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	November 14, 2016
New topic	The Create a trigger for a Lambda function topic has been updated to reflect the ability to create CodeCommit triggers as part of creating the Lambda function. This simplified process streamlines trigger creation and automatically configures the trigger with the permissions required for CodeCommit to invoke the Lambda function. The Create a trigger for an existing Lambda function topic has been added to include information about creating triggers for existing Lambda functions in the CodeCommit console.	October 19, 2016
New topic	CodeCommit is now available in another region, US East (Ohio). The Regions and Git connection endpoints topic has been added to provide information about Git connection endpoints and supported regions for CodeCommit.	October 17, 2016
Topic update	The Product and service integrations topic has been updated to include information about integration with AWS Elastic Beanstalk.	October 13, 2016
Topic update	The Product and service integrations topic has been updated to include information about integration with CloudFormation.	October 6, 2016
Topic update	The For SSH connections on Windows topic has been revised to provide guidance for using a Bash emulator for SSH connections on Windows instead of the PuTTY suite of tools.	September 29, 2016

Change	Description	Date changed
Topic update	The View commit details and Getting started with CodeCommit topics have been updated to include information about the Commit Visualizer in the CodeCommit console. The Quotas topic has been updated with the increase to the number of references allowed in a single push.	September 14, 2016
Topic update	The View commit details and Getting started with CodeCommit topics have been updated to include information about viewing the history of commits in the CodeCommit console.	July 28, 2016
New topics	The Migrate a Git repository to AWS CodeCommit and Migrate local or unversioned content to AWS CodeCommit topics have been added.	June 29, 2016
Topic update	Minor updates have been made to the Troubleshooting and For HTTPS connections on Windows with the AWS CLI credential helper topics.	June 22, 2016
Topic update	The Product and service integrations and Access Permissions Reference topics have been updated to include information about integration with CodePipeline.	April 18, 2016
New topics	The Manage triggers for a repository section has been added. New topics include examples, including policy and code samples, of how to create, edit, and delete triggers.	March 7, 2016
New topic	The Product and service integrations topic has been added. Minor updates have been made to Troubleshooting.	March 7, 2016

Change	Description	Date changed
Topic update	In addition to the MD5 server fingerprint, the SHA256 server fingerprint for CodeCommit has been added to For SSH connections on Linux, macOS, or Unix and For SSH connections on Windows.	December 9, 2015
New topic	The Browse files in a repository topic has been added. New issues have been added to Troubleshooting. Minor improvements and fixes have been made throughout the user guide.	October 5, 2015
New topic	The For SSH users not using the AWS CLI topic has been added. The topics in the Setting up section have been streamlined. Guidance to help users determine which steps to follow for their operating systems and preferred protocols has been provided.	August 5, 2015
Topic update	Clarification and examples have been added to the SSH key ID steps in SSH and Linux, macOS, or Unix: Set up the public and private keys for Git and CodeCommit and Step 3: Set up the public and private keys for Git and CodeCommit .	July 24, 2015
Topic update	Steps in Step 3: Set up the public and private keys for Git and CodeCommit have been updated to address an issue with IAM and saving the public key file.	July 22, 2015
Topic update	Troubleshooting has been updated with navigation aids. More troubleshooting information for credential keychain issues has been added.	July 20, 2015
Topic update	More information about AWS Key Management Service permissions has been added to the AWS KMS and encryption and the Access Permissions Reference topics.	July 17, 2015

Change	Description	Date changed
Topic update	Another section has been added to Troubleshooting with information about troubleshooting issues with AWS Key Management Service.	July 10, 2015
Initial release	This is the initial release of the *CodeCommit User Guide*.	July 9, 2015

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.