
AWS CodeCommit

User Guide

API Version 2015-04-13



AWS CodeCommit: User Guide

Copyright © 2019 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.

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?	3
How Is CodeCommit Different from File Versioning in Amazon S3?	4
How Do I Get Started with CodeCommit?	5
Where Can I Learn More About Git?	5
Setting Up	6
View and Manage Your Credentials	6
Setting Up Using Git Credentials	7
Setting Up Using Other Methods	7
Compatibility for CodeCommit, Git, and Other Components	8
For HTTPS Users Using Git Credentials	8
Step 1: Initial Configuration for CodeCommit	9
Step 2: Install Git	9
Step 3: Create Git Credentials for HTTPS Connections to CodeCommit	10
Step 4: Connect to the CodeCommit Console and Clone the Repository	11
Next Steps	12
For Connections from Development Tools	12
Integrate AWS Cloud9 with AWS CodeCommit	15
Integrate Visual Studio with AWS CodeCommit	18
Integrate Eclipse with AWS CodeCommit	22
For SSH Users Not Using the AWS CLI	26
Step 1: Associate Your Public Key with Your IAM User	27
Step 2: Add CodeCommit to Your SSH Configuration	27
Next Steps	28
For SSH Connections on Linux, macOS, or Unix	28
Step 1: Initial Configuration for CodeCommit	28
Step 2: Install Git	29
Step 3: Configure Credentials on Linux, macOS, or Unix	29
Step 4: Connect to the CodeCommit Console and Clone the Repository	32
Next Steps	32
For SSH Connections on Windows	32
Step 1: Initial Configuration for CodeCommit	33
Step 2: Install Git	33
SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit	34
Step 4: Connect to the CodeCommit Console and Clone the Repository	36
Next Steps	37
For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper	37
Step 1: Initial Configuration for CodeCommit	37
Step 2: Install Git	39
Step 3: Set Up the Credential Helper	39
Step 4: Connect to the CodeCommit Console and Clone the Repository	40
Next Steps	41
For HTTPS Connections on Windows with the AWS CLI Credential Helper	41
Step 1: Initial Configuration for CodeCommit	42
Step 2: Install Git	43
Step 3: Set Up the Credential Helper	44
Step 4: Connect to the CodeCommit Console and Clone the Repository	45
Next Steps	46
Getting Started	47
CodeCommit Tutorial	47
Step 1: Create a CodeCommit Repository	49
Step 2: Add Files to Your Repository	50

Step 3: Browse the Contents of Your Repository	52
Step 4: Create and Collaborate on a Pull Request	58
Step 5: Next Steps	63
Step 6: Clean Up	63
Git with CodeCommit Tutorial	64
Step 1: Create a CodeCommit Repository	64
Step 2: Create a Local Repo	65
Step 3: Create Your First Commit	65
Step 4: Push Your First Commit	66
Step 5: Share the CodeCommit Repository and Push and Pull Another Commit	66
Step 6: Create and Share a Branch	68
Step 7: Create and Share a Tag	69
Step 8: Set Up Access Permissions	70
Step 9: Clean Up	72
Product and Service Integrations	73
Integration with Other AWS Services	73
Integration Examples from the Community	78
Blog Posts	78
Code Samples	80
Working with Repositories	81
Create a Repository	82
Create a Repository (Console)	82
Create a Repository (AWS CLI)	83
Connect to a Repository	84
Prerequisites for Connecting to a CodeCommit Repository	85
Connect to the CodeCommit Repository by Cloning the Repository	85
Connect a Local Repo to the CodeCommit Repository	86
Share a Repository	87
Choose the Connection Protocol to Share with Your Users	87
Create IAM Policies for Your Repository	88
Create an IAM Group for Repository Users	89
Share the Connection Information with Your Users	89
Configuring Notifications for Repository Events	90
Using Repository Notifications	91
Configure Repository Notifications	92
Change, Disable, or Enable Notifications	94
Delete Notification Settings for a Repository	95
Tagging a Repository	96
Add a Tag to a Repository	96
View Tags for a Repository	98
Edit Tags for a Repository	99
Remove a Tag from a Repository	100
Manage Triggers for a Repository	101
Create the Resource and Add Permissions for CodeCommit	102
Create a Trigger for an Amazon SNS Topic	102
Create a Trigger for a Lambda Function	107
Create a Trigger for an Existing Lambda Function	110
Edit Triggers for a Repository	115
Test Triggers for a Repository	116
Delete Triggers from a Repository	118
View Repository Details	119
View Repository Details (Console)	120
View CodeCommit Repository Details (Git)	120
View CodeCommit Repository Details (AWS CLI)	121
Change Repository Settings	123
Change Repository Settings (Console)	124
Change AWS CodeCommit Repository Settings (AWS CLI)	125

Sync Changes Between Repositories	126
Push Commits to Two Repositories	127
Configure Cross-Account Access to a Repository	129
Cross-Account Repository Access: Actions for the Administrator in AccountA	130
Cross-Account Repository Access: Actions for the Administrator in AccountB	133
Cross-Account Repository Access: Actions for the Repository User in AccountB	134
Delete a Repository	138
Delete a CodeCommit Repository (Console)	138
Delete a Local Repo	139
Delete a CodeCommit Repository (AWS CLI)	139
Working with Files	140
Browse Files in a Repository	141
Browse a CodeCommit Repository	141
Create or Add a File	142
Create or Upload a File (Console)	143
Add a File (AWS CLI)	144
Add a File (Git)	145
Edit the Contents of a File	145
Edit a File (Console)	146
Edit or Delete a File (AWS CLI)	147
Edit a File (Git)	148
Working with Pull Requests	149
Create a Pull Request	151
Create a Pull Request (Console)	151
Create a Pull Request (AWS CLI)	153
View Pull Requests	154
View Pull Requests (Console)	154
View Pull Requests (AWS CLI)	155
Review a Pull Request	157
Review a Pull Request (Console)	158
Review Pull Requests (AWS CLI)	161
Update a Pull Request	164
Update a Pull Request (Git)	164
Update a Pull Request (Console)	165
Update Pull Requests (AWS CLI)	166
Merge a Pull Request	167
Merge a Pull Request (Console)	168
Merge a Pull Request (AWS CLI)	170
Resolve Conflicts in a Pull Request	173
Resolve Conflicts in a Pull Request (Console)	173
Resolve Conflicts in a Pull Request (AWS CLI)	176
Close a Pull Request	181
Close a Pull Request (Console)	182
Close a Pull Request (AWS CLI)	182
Working with Commits	184
Create a Commit	185
Create a Commit Using a Git Client	185
Create a Commit Using the AWS CLI	187
View Commit Details	189
Browse Commits in a Repository	189
View Commit Details (AWS CLI)	192
View Commit Details (Git)	194
Compare Commits	196
Compare a Commit to Its Parent	196
Compare Any Two Commit Specifiers	199
Comment on a Commit	201
View Comments on a Commit in a Repository	202

Add and Reply to Comments on a Commit in a Repository	202
View, Add, Update, and Reply to Comments (AWS CLI)	206
Create a Git Tag	210
Use Git to Create a Tag	210
View Tag Details	211
View Tag Details (Console)	211
View Git Tag Details (Git)	212
Delete a Tag	213
Use Git to Delete a Git Tag	213
Working with Branches	215
Create a Branch	216
Create a Branch (Console)	216
Create a Branch (Git)	217
Create a Branch (AWS CLI)	218
Limit Pushes and Merges to Branches	219
Configure an IAM Policy to Limit Pushes and Merges to a Branch	219
Apply the IAM Policy to an IAM Group or Role	221
Test the Policy	221
View Branch Details	221
View Branch Details (Console)	221
View Branch Details (Git)	222
View Branch Details (AWS CLI)	223
Compare Branches	224
Compare a Branch to the Default Branch	224
Compare Two Specific Branches	224
Change Branch Settings	225
Change the Default Branch (Console)	225
Change the Default Branch (AWS CLI)	226
Delete a Branch	226
Delete a Branch (Console)	227
Delete a Branch (AWS CLI)	227
Delete a Branch (Git)	227
Working with User Preferences	229
Migrate to CodeCommit	230
Migrate a Git Repository to AWS CodeCommit	230
Step 0: Setup Required for Access to CodeCommit	231
Step 1: Create a CodeCommit Repository	233
Step 2: Clone the Repository and Push to the CodeCommit Repository	235
Step 3: View Files in CodeCommit	236
Step 4: Share the CodeCommit Repository	236
Migrate Content to CodeCommit	238
Step 0: Setup Required for Access to CodeCommit	238
Step 1: Create a CodeCommit Repository	241
Step 2: Migrate Local Content to the CodeCommit Repository	242
Step 3: View Files in CodeCommit	243
Step 4: Share the CodeCommit Repository	243
Migrate a Repository in Increments	245
Step 0: Determine Whether to Migrate Incrementally	245
Step 1: Install Prerequisites and Add the CodeCommit Repository as a Remote	245
Step 2: Create the Script to Use for Migrating Incrementally	247
Step 3: Run the Script and Migrate Incrementally to CodeCommit	247
Appendix: Sample Script <code>incremental-repo-migration.py</code>	248
Troubleshooting	253
Troubleshooting Git Credentials (HTTPS)	253
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	253
Git Credentials for AWS CodeCommit: I set up Git credentials, but my system is not using them ..	254

Troubleshooting SSH Connections	254
Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems	254
Access error: Public key is uploaded successfully to IAM and SSH tested successfully but connection fails on Windows systems	255
Authentication challenge: Authenticity of host can't be established when connecting to a CodeCommit repository	256
IAM error: 'Invalid format' when attempting to add a public key to IAM	258
Git on Windows: Bash emulator or command line freezes when attempting to connect using SSH	258
Troubleshooting the Credential Helper (HTTPS)	259
I get a command not found error in Windows when using the credential helper	259
I am prompted for a user name when I connect to a CodeCommit Repository	260
Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository (403)	260
Git for Windows: I installed Git for Windows, but I am denied access to my repository (403)	262
Troubleshooting Git Clients	263
Git error: error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly	263
Git error: Too many reference update commands	263
Git error: Push via HTTPS is broken in some versions of Git	264
Git error: 'gnutls_handshake() failed'	264
Git error: Git cannot find the CodeCommit repository or does not have permission to access the repository	264
Git on Windows: No supported authentication methods available (publickey)	264
Troubleshooting Access Errors	265
Access error: I am prompted for a user name and password when I connect to a CodeCommit repository from Windows	265
Access error: Public key denied when connecting to a CodeCommit repository	265
Access error: "Rate Exceeded" or "429" message when connecting to a CodeCommit repository	266
Troubleshooting Configuration Errors	266
Configuration error: Cannot configure AWS CLI credentials on macOS	267
Troubleshooting Console Errors	267
Access error: Encryption key access denied for a CodeCommit repository from the console or AWS CLI	266
Console error: Cannot browse the code in a CodeCommit repository from the console	267
Troubleshooting Triggers	268
Trigger error: A repository trigger does not run when expected	268
Turn on Debugging	268
Authentication and Access Control	270
Authentication	270
Access Control	271
Overview of Managing Access	271
Resources and Operations	272
Understanding Resource Ownership	273
Managing Access to Resources	273
Resource Scoping in CodeCommit	274
Specifying Policy Elements: Resources, Actions, Effects, and Principals	275
Specifying Conditions in a Policy	275
Using Identity-Based Policies (IAM Policies)	275
Permissions Required to Use the CodeCommit Console	276
Viewing Resources in the Console	278
AWS Managed (Predefined) Policies for CodeCommit	279
Customer Managed Policy Examples	284
CodeCommit Permissions Reference	291
Required Permissions for Git Client Commands	292
Permissions for Actions on Branches	292

Permissions for Actions on Merges	294
Permissions for Actions on Pull Requests	294
Permissions for Actions on Individual Files	297
Permissions for Actions on Comments	297
Permissions for Actions on Committed Code	298
Permissions for Actions on Repositories	300
Permissions for Actions on Tags	301
Permissions for Actions on Triggers	301
Permissions for Actions on CodePipeline Integration	302
CodeCommit Reference	304
Regions and Git Connection Endpoints	304
Supported Regions for CodeCommit	304
Git Connection Endpoints	305
Server Fingerprints for CodeCommit	308
Using AWS CodeCommit with Interface VPC Endpoints	310
Availability	310
Create VPC Endpoints for CodeCommit	311
Create a VPC Endpoint Policy for CodeCommit	311
Limits	312
Temporary Access	315
Step 1: Complete the Prerequisites	316
Step 2: Get Temporary Access Credentials	316
Step 3: Configure the AWS CLI with Your Temporary Access Credentials	317
Step 4: Access the CodeCommit Repositories	318
AWS KMS and Encryption	318
Encryption Context	319
Logging AWS CodeCommit API Calls with AWS CloudTrail	319
CodeCommit Information in CloudTrail	319
Understanding CodeCommit Log File Entries	320
Command Line Reference	325
Basic Git Commands	327
Configuration Variables	328
Remote Repositories	328
Commits	329
Branches	330
Tags	331
Document History	332
Earlier Updates	333
AWS Glossary	338

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 \(p. 305\)](#).

Topics

- [Introducing CodeCommit \(p. 1\)](#)
- [CodeCommit, Git, and Choosing the Right AWS Service for Your Needs \(p. 2\)](#)
- [How Does CodeCommit Work? \(p. 3\)](#)
- [How Is CodeCommit Different from File Versioning in Amazon S3? \(p. 4\)](#)
- [How Do I Get Started with CodeCommit? \(p. 5\)](#)
- [Where Can I Learn More About Git? \(p. 5\)](#)

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 \(p. 73\)](#).
- **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 reconstruct 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 Developer Guide .
Database	Git repositories grow larger over time. Because versioning tracks all changes, any change will increase your repository size. In other words, as you commit data, even if you delete data in a commit, data is added to a repository. As there is more data to process and transmit over time, Git will slow down. This is particularly detrimental to a database use case. Git was not designed as a database.	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 .

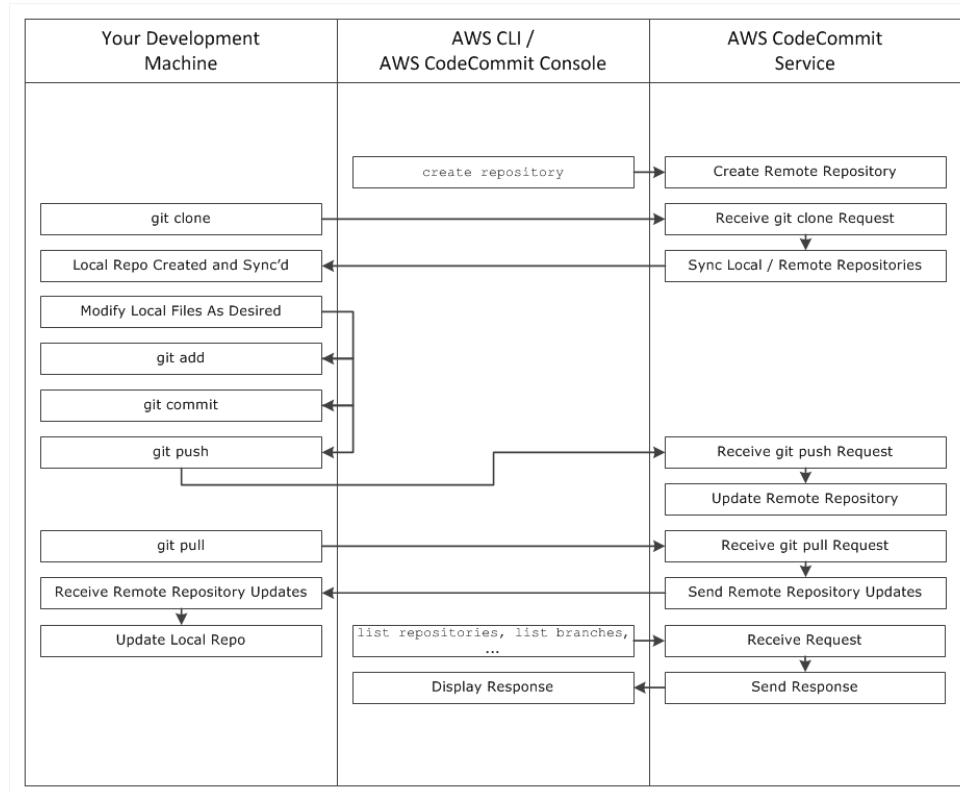
Use case	Description	Other services to consider
Audit trails	Typically, audit trails are kept for long periods of time and are continuously generated by system processes at a very frequent cadence. Git was designed to securely store source code generated by groups of developers on a development cycle. 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 (p. 127) , 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 .
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 negotiation) 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: <ul style="list-style-type: none"> • Linux, macOS, or Unix, or Bash emulator on Windows: <pre>git ls-remote wc -l</pre> • Powershell: <pre>git ls-remote Measure-Object -line</pre>

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.

The following figure shows how you use your development machine, the AWS CLI or CodeCommit console, and the CodeCommit service to create and manage repositories:



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 \(p. 6\)](#) to prepare your development machines.
2. Follow the steps in one or more of the tutorials in [Getting Started \(p. 47\)](#).
3. [Create \(p. 82\)](#) version control projects in CodeCommit or [migrate \(p. 230\)](#) 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 \(p. 327\)](#). 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 \(p. 140\)](#) 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. Read through these other options carefully, to decide which alternative method works best for you.

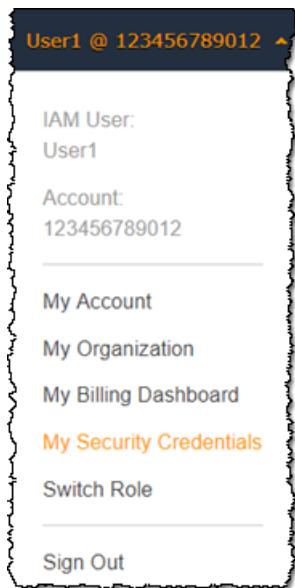
- [Setting Up Using Git Credentials \(p. 7\)](#)
- [Setting Up Using Other Methods \(p. 7\)](#)
- [Compatibility for CodeCommit, Git, and Other Components \(p. 8\)](#)

For information about using CodeCommit and Amazon Virtual Private Cloud, see [Using AWS CodeCommit with Interface VPC Endpoints \(p. 310\)](#).

View and Manage Your Credentials

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

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation bar on the upper right, choose your user name, and then choose **My 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 \(p. 8\)](#): Follow these instructions to set up connections between your local computer and CodeCommit repositories using Git credentials.
- [For Connections from Development Tools \(p. 12\)](#): 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, Eclipse, 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 \(p. 26\)](#): 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 \(p. 28\)](#): 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 \(p. 32\)](#): 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 or temporary credentials, or if you do not want to configure IAM users, you can set up connections to CodeCommit repositories using the credential helper included in the AWS CLI. The credential helper 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. **This is the only connection method for CodeCommit repositories that does not require an IAM user, so it is the only method that supports federated access and temporary credentials.** 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 ease of use, consider creating IAM users and configuring Git credentials with HTTPS connections instead of using the credential helper.

- [For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper \(p. 37\)](#): 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 \(p. 41\)](#): 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 AWS 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 \(p. 129\)](#): Follow these instructions for a step-by-step walkthrough of configuring cross-account access in one AWS account to users in an IAM group in another AWS 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.

Version Compatibility Information for AWS CodeCommit

Component	Version
Git	CodeCommit supports Git versions 1.7.9 and later.
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 (p. 253) .

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 individual development environment (IDE).

that supports HTTPS authentication using a static user name and password. For examples, see [For Connections from Development Tools \(p. 12\)](#).

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 AWS account, create an IAM user, and configure access to CodeCommit.

To create and configure an IAM user for accessing CodeCommit

1. Create an AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

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.

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 \(p. 253\)](#).

Step 3: Create Git Credentials for HTTPS Connections to CodeCommit

After you have installed Git, create Git credentials for your IAM user in IAM. For more information, see [Use Git Credentials and HTTPS with AWS CodeCommit](#) in the *IAM User Guide*.

To set up HTTPS Git credentials for CodeCommit

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 \(p. 6\)](#).

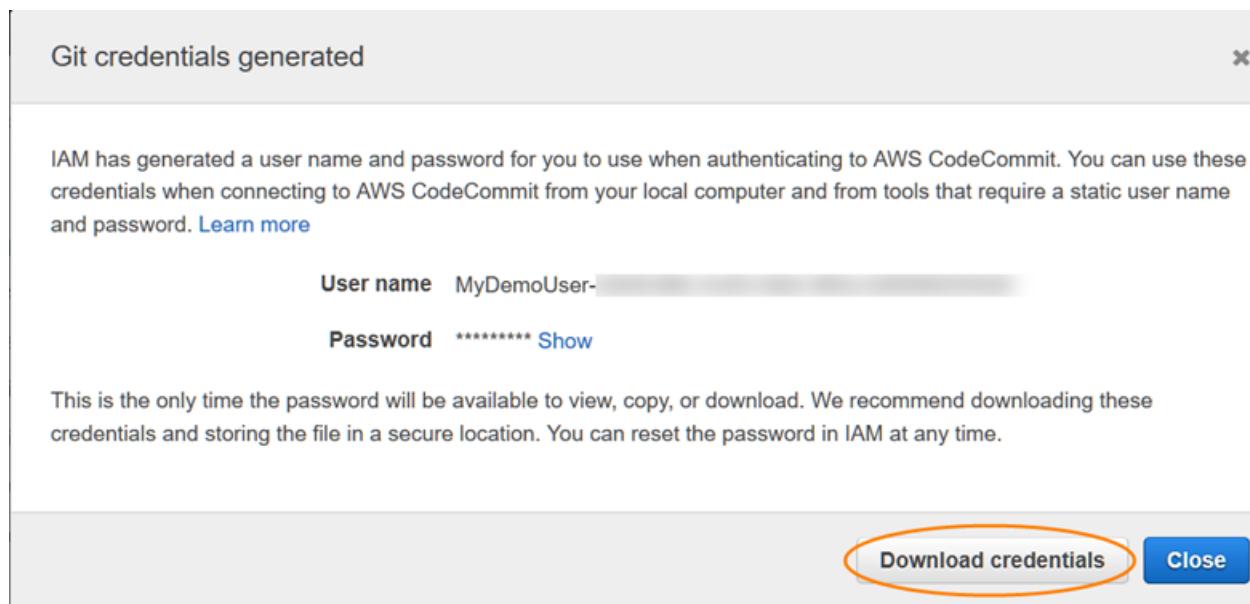
3. On the user details page, choose the **Security Credentials** tab, and in **HTTPS Git credentials for AWS CodeCommit**, choose **Generate**.

The screenshot shows the AWS IAM User Details page. The left sidebar has a 'Users' section selected. The main area shows the 'SSH keys for AWS CodeCommit' and 'HTTPS Git credentials for AWS CodeCommit' sections. The 'Generate' button in the 'HTTPS Git credentials' section is circled in orange.

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.



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 \(p. 304\)](#).
3. Choose the repository you want to connect to from the list. This opens the **Code** page for that repository.

If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see [the section called "Create a Repository" \(p. 82\)](#) or follow the steps in the [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.

4. Choose **Connect**. Review the instructions and copy the URL to use when connecting to the repository.
5. Open a terminal, command line, or Git shell. Using the HTTPS URL you copied, run the **git clone** command 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 (for example, Keychain Access for macOS), 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 \(p. 10\)](#)). 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, deactivate 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 \(p. 85\)](#) and [Create a Commit \(p. 185\)](#).

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) to start using CodeCommit.

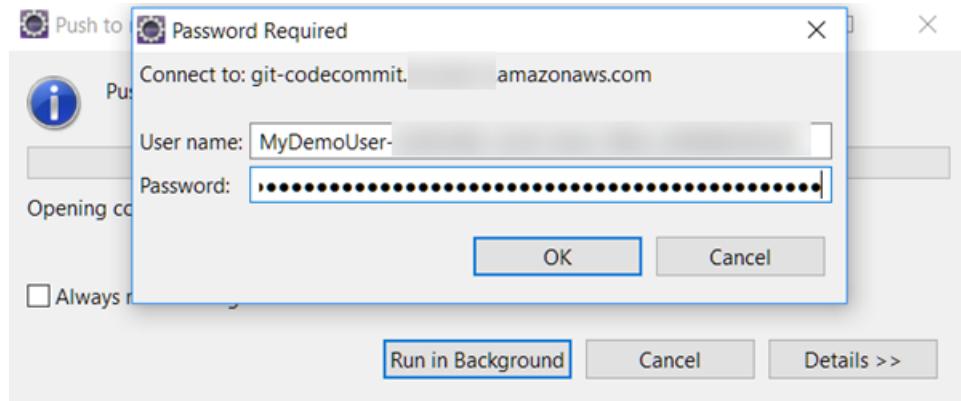
Set Up Connections from Development Tools Using Git Credentials

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, Eclipse, 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.

Topics

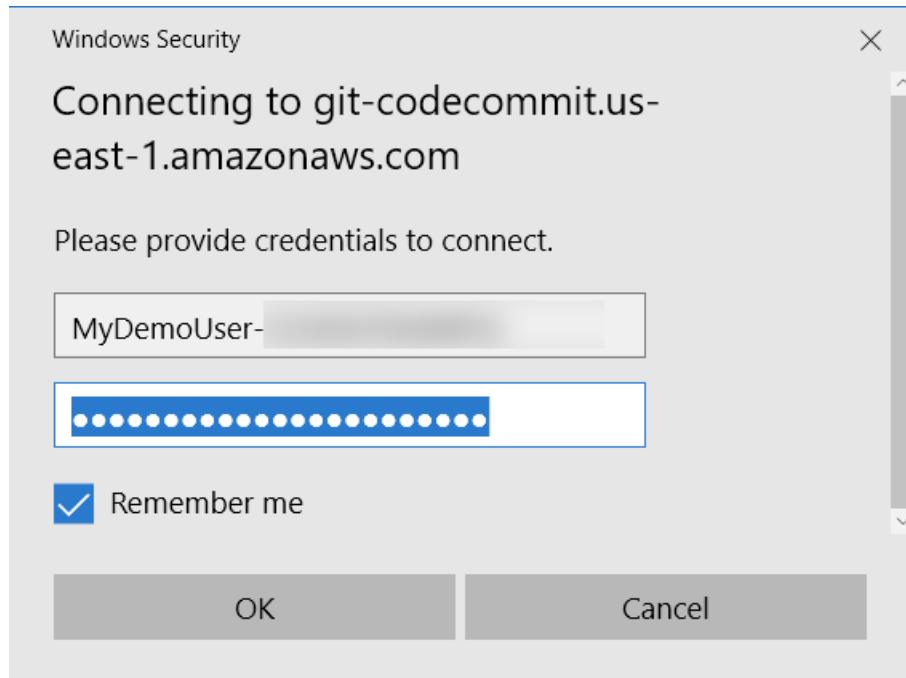
- [Integrate AWS Cloud9 with AWS CodeCommit \(p. 15\)](#)
- [Integrate Visual Studio with AWS CodeCommit \(p. 18\)](#)
- [Integrate Eclipse with AWS CodeCommit \(p. 22\)](#)

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 example, if you are prompted for a user name and password in Eclipse, you would provide your Git credentials as follows:



For more information about AWS Regions and endpoints for CodeCommit, see [Regions and Git Connection Endpoints \(p. 304\)](#).

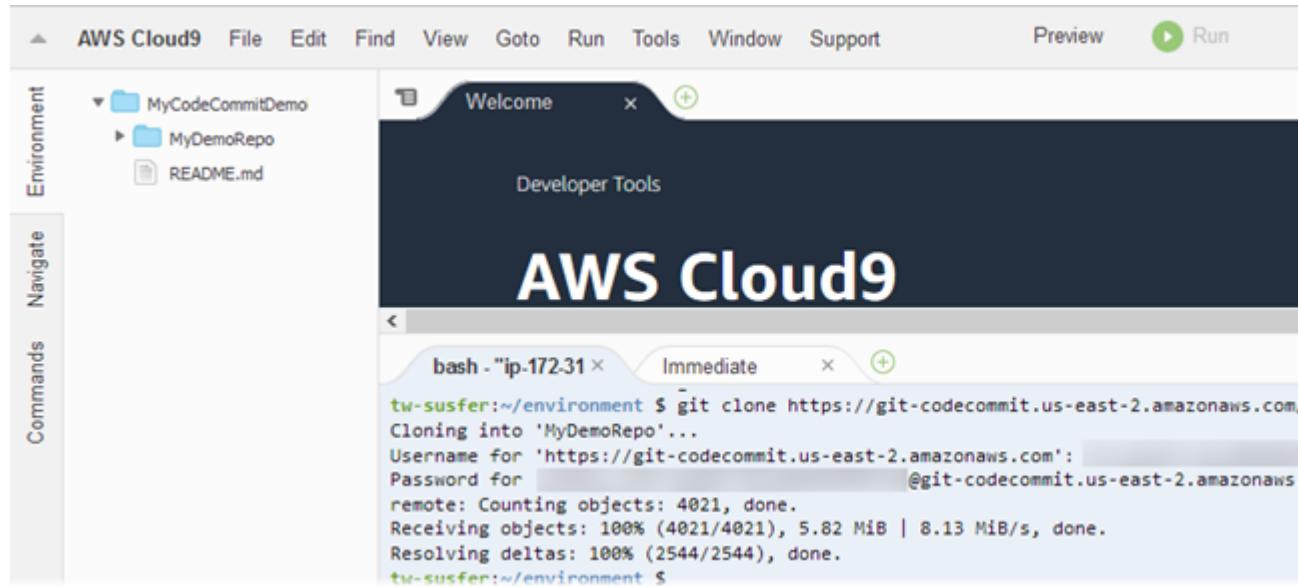
You might also see a prompt from your operating system to store your user name and password. For example, in Windows, you would provide your Git credentials as follows:



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

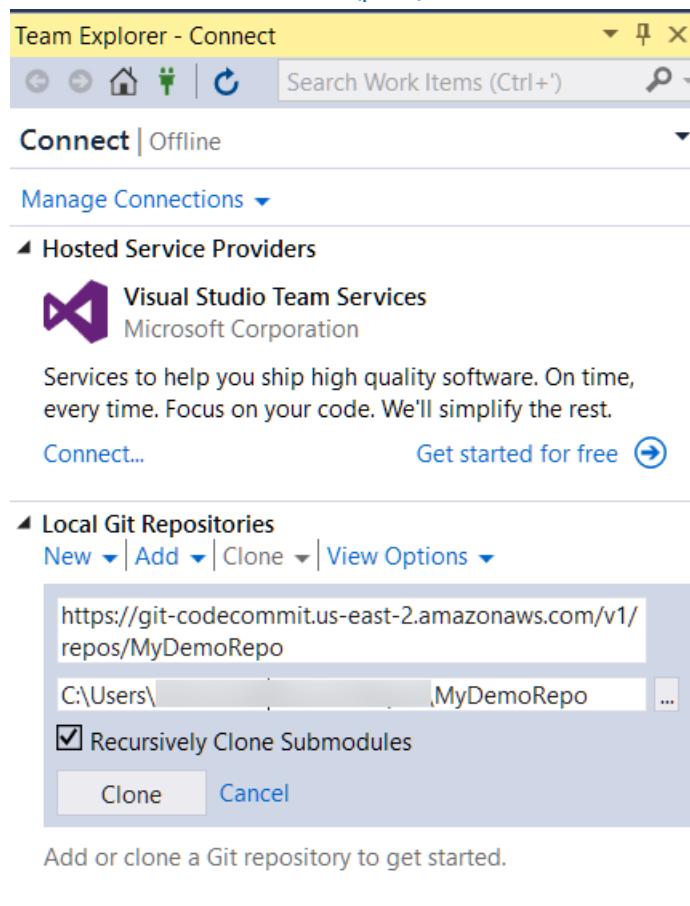
The following is not a comprehensive list of IDEs. The links are provided solely to help you learn more about these tools. AWS is not responsible for the content of any of these topics.

- [AWS Cloud9 \(p. 15\)](#)



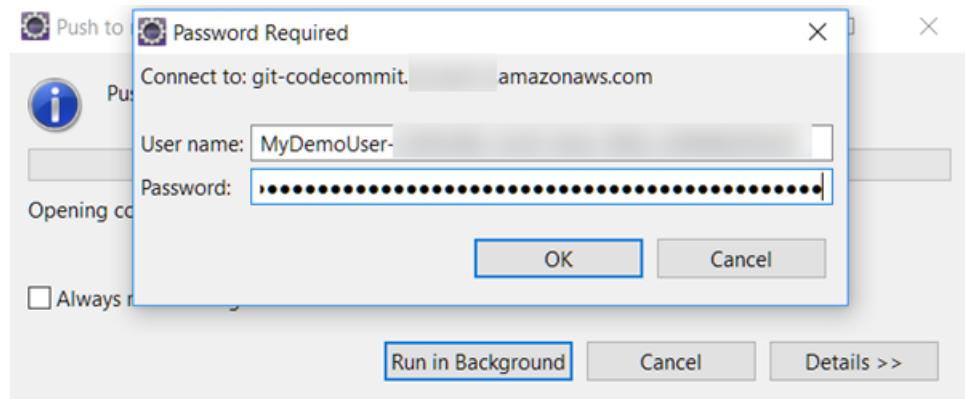
- [Visual Studio](#)

Alternatively, install the AWS Toolkit for Visual Studio. For more information, see [Integrate Visual Studio with AWS CodeCommit \(p. 18\)](#).

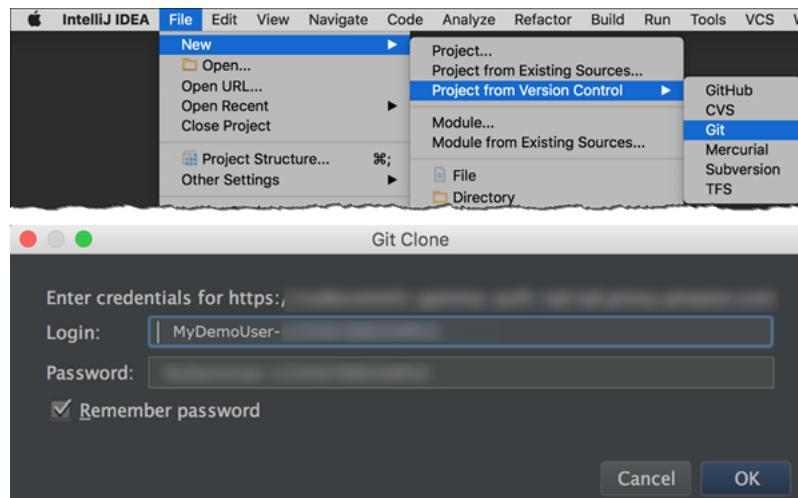


- [EGit with Eclipse](#)

Alternatively, install the AWS Toolkit for Eclipse. For more information, see [Integrate Eclipse with AWS CodeCommit \(p. 22\)](#).



- [IntelliJ](#)



- [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 \(Predefined\) Policies for CodeCommit \(p. 279\)](#) and [Understanding and Getting Your Security Credentials](#).

Topics

- [Step 1: Create an AWS Cloud9 Development Environment \(p. 16\)](#)
- [Step 2: Configure the AWS CLI Credential Helper on Your AWS Cloud9 EC2 Development Environment \(p. 17\)](#)
- [Step 3: Clone a CodeCommit Repository into Your AWS Cloud9 EC2 Development Environment \(p. 18\)](#)
- [Next Steps \(p. 18\)](#)

Step 1: Create an AWS Cloud9 Development Environment

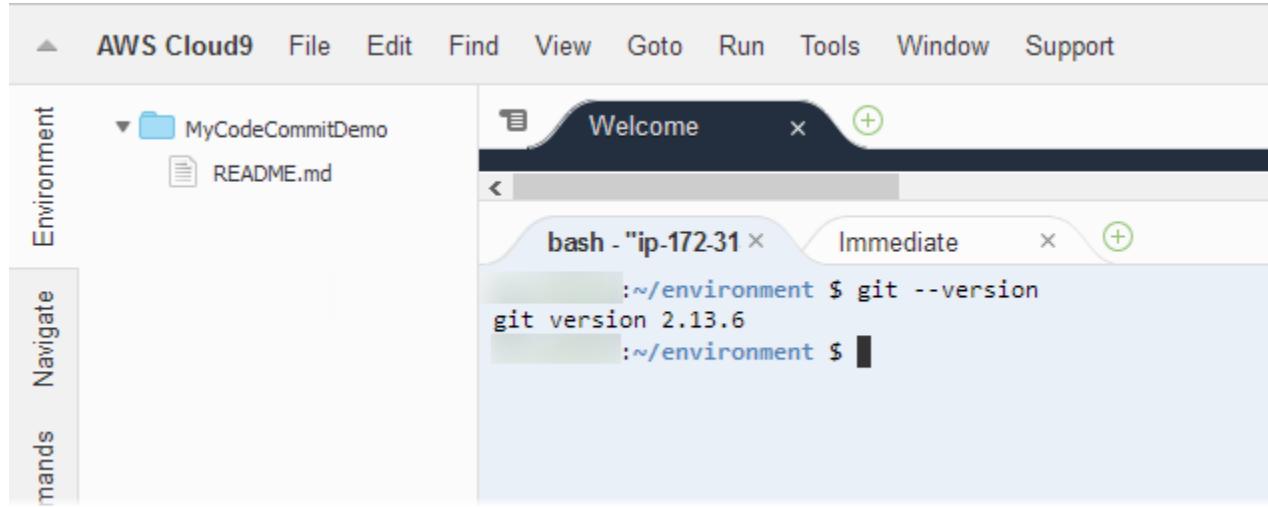
AWS Cloud9 hosts your development environment on an Amazon EC2 instance. This is the easiest way to integrate, because you can use the AWS managed temporary credentials for the instance to connect to your CodeCommit repository. If you want to use your own server instead, see the [AWS Cloud9 User Guide](#).

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 AWS 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. 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, both in **Repositories** and in the **Clone URL** information on the **Code** page of the repository.

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](#). For more information about using Git with CodeCommit, see [Git with AWS CodeCommit Tutorial \(p. 64\)](#).

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 Team Explorer. 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'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 \(Predefined\) Policies for CodeCommit \(p. 279\)](#) 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](#).

Topics

- [Step 1: Get an Access Key and Secret Key for Your IAM User \(p. 19\)](#)
- [Step 2: Install AWS Toolkit for Visual Studio and Connect to CodeCommit \(p. 20\)](#)
- [Clone a CodeCommit Repository from Visual Studio \(p. 21\)](#)
- [Create a CodeCommit Repository from Visual Studio \(p. 21\)](#)
- [Working with CodeCommit Repositories \(p. 22\)](#)

Step 1: Get an Access Key and Secret Key for Your IAM User

If you do not already have a credential profile set up on the computer where Visual Studio is installed, you can [configure one with the AWS CLI and the aws configure command](#). Alternatively, you can follow the steps in this procedure to create and download your credentials. Provide them to the Toolkit for Visual Studio when prompted.

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, [create a new administrator IAM user](#) with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see [Permissions Required to Access IAM Resources](#) in the *IAM User Guide*.

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.

3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
 - Access key ID: AKIAIOSFODNN7EXAMPLE
 - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

7. After you download the .csv file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics

- [What Is IAM?](#) in the *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

Step 2: Install AWS Toolkit for Visual Studio and Connect to CodeCommit

The Toolkit for Visual Studio is a software package you can add to Visual Studio. After you've installed it, you can connect to CodeCommit from Team Explorer in Visual Studio.

To install the Toolkit for Visual Studio with the AWS CodeCommit module and configure access to your project repository

1. Install Visual Studio on your local computer if you don't have a supported version already installed.
2. [Download and install the Toolkit for Visual Studio](#) and save the file to a local folder or directory. Launch the installation wizard by opening the file. When prompted on the **Getting Started with the AWS Toolkit for Visual Studio** page, enter or import your AWS credentials (your access key and secret key), and then choose **Save and Close**.
3. In Visual Studio, open **Team Explorer**. In **Hosted Service Providers**, find **AWS CodeCommit**, and then choose **Connect**.



AWS CodeCommit is a fully-managed source control service that makes it easy for companies to host secure and highly scalable private Git repositories.

[Connect...](#)

[Sign up](#)

4. Do one of the following:

- If you have a single credential profile already configured on your computer, the Toolkit for Visual Studio applies it automatically. No action is required. The AWS CodeCommit connection panel appears in Team Explorer.

- If you have more than one credential profile configured on your computer, you are prompted to choose the one you want to use. Choose the profile associated with the IAM user you'll use for connecting to CodeCommit repositories, and then choose **OK**.
- If you do not have a profile configured, a dialog box appears and asks for your AWS security credentials (your access key and secret key). Enter or import them, and then choose **OK**.

After you are signed in with a profile, the AWS CodeCommit connection panel appears in Team Explorer with options to clone, create, or sign out. Choosing **Clone** clones an existing CodeCommit repository to your local computer, so you can start working on code. This is the most frequently used option.

If you don't have repositories, or want to create a repository, choose **Create**. For more information, see [Create a CodeCommit Repository from Visual Studio \(p. 21\)](#).

Clone a CodeCommit Repository from Visual Studio

After you're connected to CodeCommit, you can clone a repository to a local repo on your computer. Then you can start working with the code.

1. In **Manage Connections**, choose **Clone**. In **Region**, choose the AWS Region where the repository was created in CodeCommit. Choose your project's repository and the folder on your local computer you want to clone the repository into, and then choose **OK**.
2. If you are prompted to create Git credentials, choose **Yes**. The toolkit attempts to create credentials on your behalf. You must have the IAMSelfManageServiceSpecificCredentials applied to your IAM user, or the equivalent permissions. When prompted, save the credentials file in a secure location. This is the only opportunity you have to save these Git credentials.

If the toolkit cannot create Git credentials on your behalf, or if you chose **No**, you must create and provide your own Git credentials. For more information, see [For HTTPS Users Using Git Credentials \(p. 8\)](#), or follow the online directions.

3. When you have finished cloning the project, you're ready to start editing your code in Visual Studio and committing and pushing your changes to your project's repository in CodeCommit.

Create a CodeCommit Repository from Visual Studio

You can create CodeCommit repositories from Visual Studio with the Toolkit for Visual Studio. As part of creating the repository, you also clone it to a local repo on your computer, so you can start working with it right away.

1. In **Manage Connections**, choose **Create**.
2. In **Region**, choose the AWS Region where you want to create the repository. CodeCommit repositories are organized by AWS Region.
3. In **Name**, enter a name for this repository. Repository names must be unique within an AWS account. There are character and length limits. For more information, see [Limits \(p. 312\)](#). In **Description**, enter an optional description for this repository. This helps others understand what the repository is for, and helps distinguish it from other repositories in the region.
4. In **Clone into**, enter or browse to the folder or directory where you want to clone this repository on your local computer. Visual Studio automatically clones the repository after it's created and creates the local repo in the location you choose.
5. When you are satisfied with your choices, choose **OK**.
6. If prompted to create Git credentials, choose **Yes**. The toolkit attempts to create credentials on your behalf. You must have the IAMSelfManageServiceSpecificCredentials applied to your IAM user, or the equivalent permissions. When prompted, save the credentials file in a secure location. This is the only opportunity you have to save these Git credentials.

If the toolkit cannot create Git credentials on your behalf, or if you chose **No**, you must create and provide your own Git credentials. For more information, see [For HTTPS Users Using Git Credentials \(p. 8\)](#), or follow the online directions.

Working with CodeCommit Repositories

After you have connected to CodeCommit, you can see a list of repositories associated with your AWS account. You can browse the contents of these repositories in Visual Studio. Open the context menu for the repository you're interested in, and choose **Browse in Console**.

Git operations in Visual Studio for CodeCommit repositories work exactly as they do for any other Git-based repository. You can make changes to code, add files, and create local commits. When you are ready to share, you use the **Sync** option in Team Explorer to push your commits to the CodeCommit repository. Because your Git credentials for your IAM user are already stored locally and associated with your connected AWS credential profile, you won't be prompted to supply them again when you push to CodeCommit.

For more information about working with Toolkit for Visual Studio, see the [AWS Toolkit for Visual Studio User Guide](#).

Integrate Eclipse with AWS CodeCommit

You can use Eclipse to make code changes in a CodeCommit repository. The Toolkit for Eclipse 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.

To use Toolkit for Eclipse with CodeCommit, you need the following:

- Eclipse installed on your local computer.
- 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 \(Predefined\) Policies for CodeCommit \(p. 279\)](#) and [Understanding and Getting Your Security Credentials](#).

- An active set of Git credentials configured for the user in IAM. For more information, see [Step 3: Create Git Credentials for HTTPS Connections to CodeCommit \(p. 10\)](#).

Topics

- [Step 1: Get an Access Key and Secret Key for Your IAM User \(p. 23\)](#)
- [Step 2: Install AWS Toolkit for Eclipse and Connect to CodeCommit \(p. 23\)](#)
- [Clone a CodeCommit Repository from Eclipse \(p. 25\)](#)
- [Create a CodeCommit Repository from Eclipse \(p. 26\)](#)
- [Working with CodeCommit Repositories \(p. 26\)](#)

Step 1: Get an Access Key and Secret Key for Your IAM User

If you do not already have a credential profile set up on the computer where Eclipse is installed, you can [configure one with the AWS CLI and the aws configure command](#). Alternatively, you can follow the steps in this procedure to create and download your credentials. Provide them to the Toolkit for Eclipse when prompted.

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, [create a new administrator IAM user](#) with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see [Permissions Required to Access IAM Resources](#) in the *IAM User Guide*.

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
 - Access key ID: AKIAIOSFODNN7EXAMPLE
 - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

7. After you download the .csv file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics

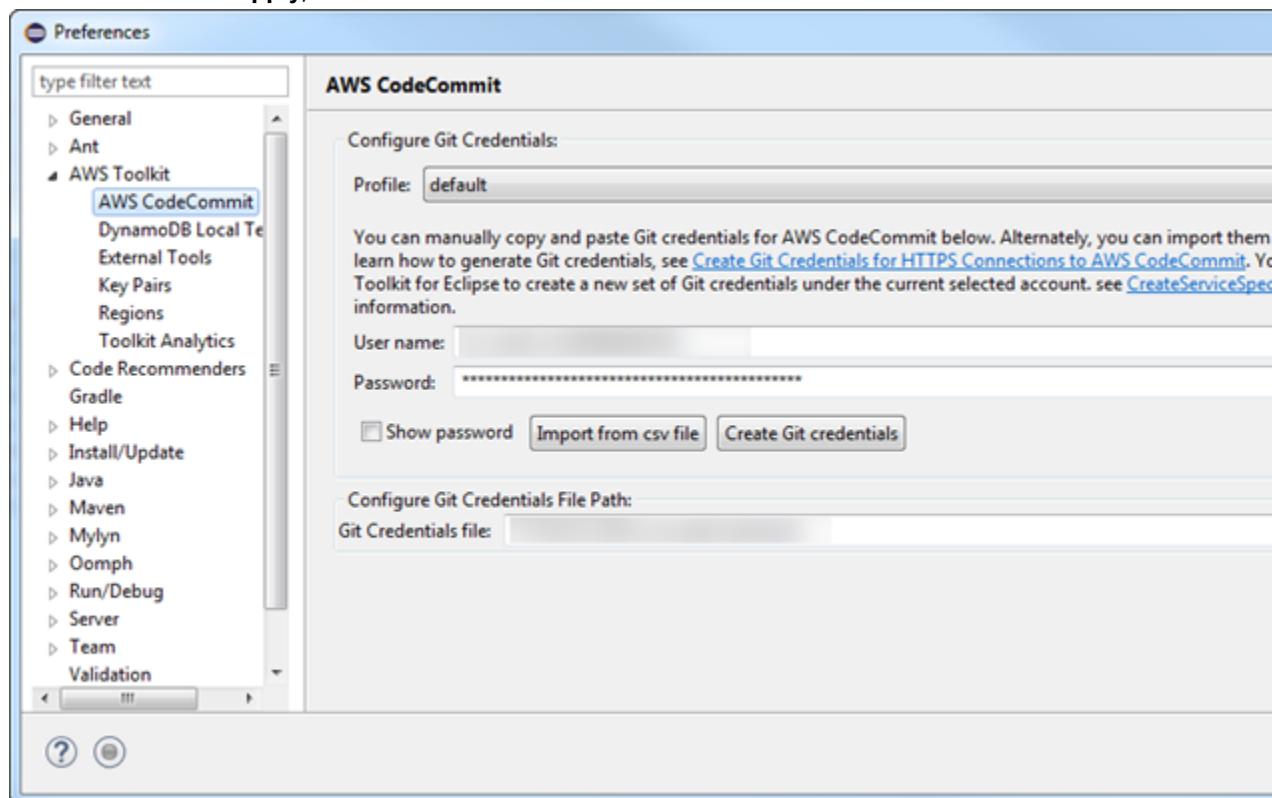
- [What Is IAM?](#) in the *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

Step 2: Install AWS Toolkit for Eclipse and Connect to CodeCommit

The Toolkit for Eclipse is a software package you can add to Eclipse. After you've installed it and configured it with your AWS credential profile, you can connect to CodeCommit from the AWS Explorer in Eclipse.

To install the Toolkit for Eclipse with the AWS CodeCommit module and configure access to your project repository

1. Install Toolkit for Eclipse on your local computer if you don't have a supported version already installed. If you need to update your version of Toolkit for Eclipse, follow the instructions in [Set Up the Toolkit](#).
2. In Eclipse, either follow the firstrun experience, or open **Preferences** from the Eclipse menu system (the location varies depending on your version and operating system) and choose **AWS Toolkit**.
3. Do one of the following:
 - If you are following the firstrun experience, provide your AWS security credentials when prompted to set up your credential profile.
 - If you are configuring in **Preferences** and have a credential profile already configured on your computer, choose it from **Default Profile**.
 - If you are configuring in **Preferences** and you do not see the profile you want to use, or if the list is empty, choose **Add profile**. In **Profile Details**, enter a name for the profile and the credentials for the IAM user (access key and secret key), or alternatively, enter the location of the credentials file.
 - If you are configuring in **Preferences** and you do not have a profile configured, use the links for signing up for an account or managing your existing AWS security credentials.
4. In Eclipse, expand the **AWS Toolkit** menu and choose **AWS CodeCommit**. Choose your credential profile, and then enter the user name and password for your Git credentials or import them from the .csv file. Choose **Apply**, and then choose **OK**.



After you are signed in with a profile, the AWS CodeCommit connection panel appears in Team Explorer with options to clone, create, or sign out. Choosing **Clone** clones an existing CodeCommit repository to your local computer, so you can start working on code. This is the most frequently used option.

If you don't have any repositories, or want to create a repository, choose **Create**.

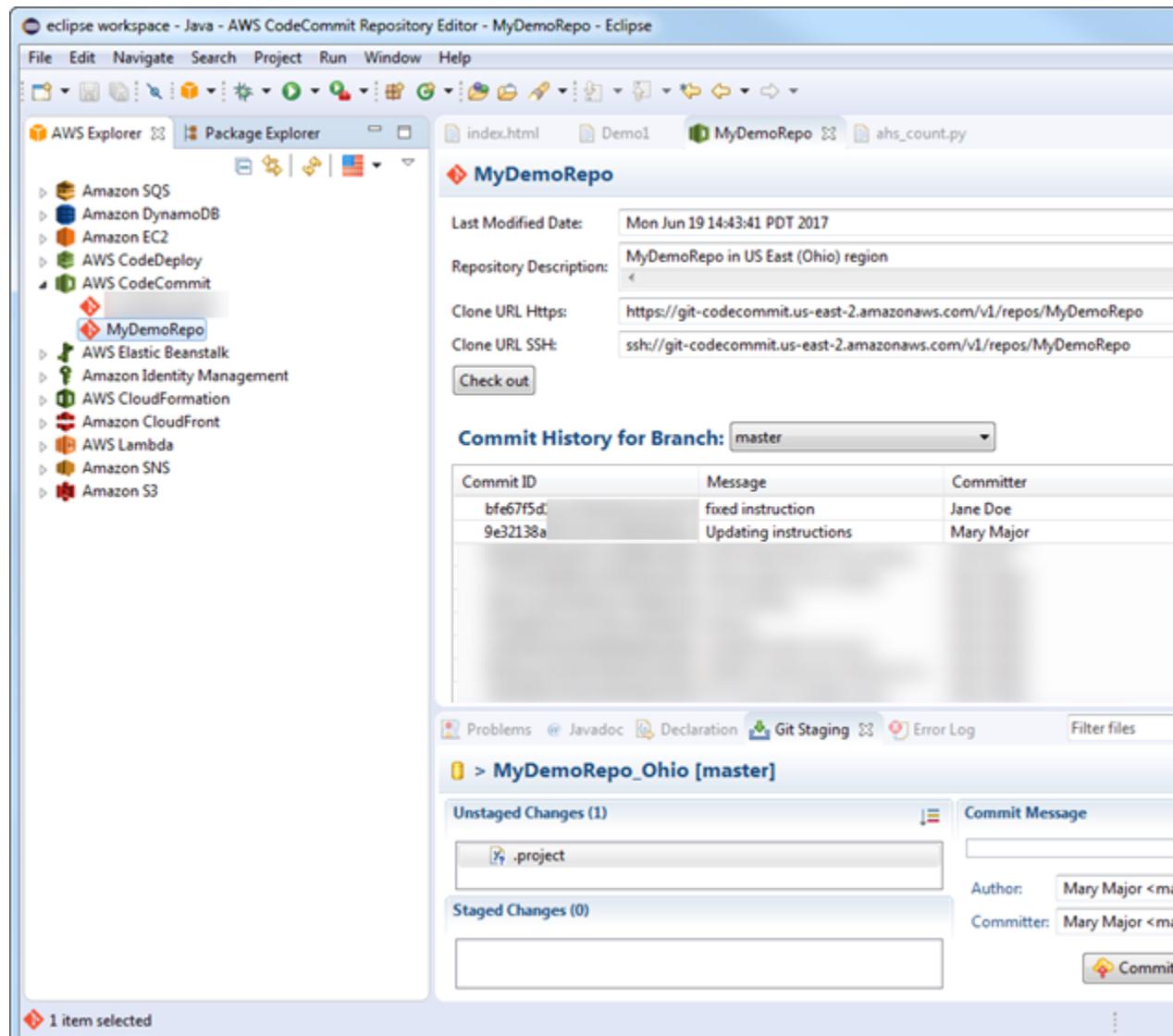
Clone a CodeCommit Repository from Eclipse

After you've configured your credentials, you can clone a repository to a local repo on your computer by checking it out in Eclipse. Then you can start working with the code.

1. In Eclipse, open **AWS Explorer**. For information about where to find it, see [How to Access AWS Explorer](#). Expand **AWS CodeCommit**, and choose the CodeCommit repository you want to work in. You can view the commit history and other details of the repository, which can help you determine if this is the repository and branch you want to clone.

Note

If you do not see your repository, choose the flag icon to open the AWS Regions menu, and choose the AWS Region where the repository was created.



2. Choose **Check out**, and follow the instructions to clone the repository to your local computer.
3. When you have finished cloning the project, you're ready to start editing your code in Eclipse and staging, committing, and pushing your changes to your project's repository in CodeCommit.

Create a CodeCommit Repository from Eclipse

You can create CodeCommit repositories from Eclipse with the Toolkit for Eclipse. As part of creating the repository, you also clone it to a local repo on your computer, so you can start working with it right away.

1. In AWS Explorer, right-click **AWS CodeCommit**, and then choose **Create repository**.

Note

Repositories are region-specific. Before you create the repository, make sure you have selected the correct AWS Region. You cannot choose the AWS Region after you have started the repository creation process.

2. In **Repository Name**, enter a name for this repository. Repository names must be unique within an AWS account. There are character and length limits. For more information, see [Limits \(p. 312\)](#). In **Repository Description**, enter an optional description for this repository. This helps others understand what this repository is for, and helps distinguish it from other repositories in the region. Choose **OK**.
3. In AWS Explorer, expand **AWS CodeCommit**, and then choose the CodeCommit repository you just created. You see that this repository has no commit history. Choose **Check out**, and follow the instructions to clone the repository to your local computer.

Working with CodeCommit Repositories

After you have connected to CodeCommit, you can see a list of repositories associated with your account, by AWS Region, in AWS Explorer. Choose the flag menu to change the region.

Note

CodeCommit might not be available in all AWS Regions supported by Toolkit for Eclipse.

In Toolkit for Eclipse, you can browse the contents of these repositories from the **Navigation** and **Package Explorer** views. To open a file, choose it from the list.

Git operations in Toolkit for Eclipse for CodeCommit repositories work exactly as they do for any other Git-based repository. You can make changes to code, add files, and create local commits. When you are ready to share, you use the **Git Staging** option to push your commits to the CodeCommit repository. If you haven't configured your author and committer information in a Git profile, you can do this before you commit and push. Because your Git credentials for your IAM user are already stored locally and associated with your connected AWS credential profile, you won't be prompted to supply them again when you push to CodeCommit.

For more information about working with Toolkit for Eclipse, see the [AWS Toolkit for Eclipse Getting Started 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 \(p. 275\)](#).
- 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 \(p. 28\)](#) or [For SSH Connections on Windows \(p. 32\)](#).

Topics

- [Step 1: Associate Your Public Key with Your IAM User \(p. 27\)](#)
- [Step 2: Add CodeCommit to Your SSH Configuration \(p. 27\)](#)
- [Next Steps \(p. 28\)](#)

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://console.aws.amazon.com/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-
AfICCD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMakGA1UEBhMCVVMxCzAJB
gNVBagTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBdb2
5zb2x1MRIwEAYDVQQDEwlUZXN0Q21sYWmxHzAdBgkqhkiG9w0BCQEWE5vb25lQGFtYXpvbi5jb20whc
NMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMakGA1UEBhMCVVMxCzAJBgnNVBAGTAldBMRAw
DgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDAS=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`:

```
Host git-codecommit.*.amazonaws.com
User Your-SSH-Key-ID, such as APKAEIBAERJR2EXAMPLE
IdentityFile Your-Private-Key-File, such as ~/.ssh/codecommit_rsa or ~/.ssh/id_rsa
```

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:

```
ssh git-codecommit.us-east-2.amazonaws.com
```

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.
Interactive shells are not supported. Connection to git-codecommit.us-
east-2.amazonaws.com closed by remote host.
```

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) to start using CodeCommit.

To connect to a repository, follow the steps in [Connect to a Repository \(p. 84\)](#). To create a repository, follow the steps in [Create a Repository \(p. 82\)](#).

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 the initial configuration steps. This topic walks you through the steps for setting up your computer and AWS profile, connecting to a CodeCommit repository, and cloning 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? \(p. 5\)](#).

Topics

- [Step 1: Initial Configuration for CodeCommit \(p. 28\)](#)
- [Step 2: Install Git \(p. 29\)](#)
- [Step 3: Configure Credentials on Linux, macOS, or Unix \(p. 29\)](#)
- [Step 4: Connect to the CodeCommit Console and Clone the Repository \(p. 32\)](#)
- [Next Steps \(p. 32\)](#)

Step 1: Initial Configuration for CodeCommit

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

To create and configure an IAM user for accessing CodeCommit

1. Create an AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.

2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

Note

If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

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.

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 \(p. 253\)](#).

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

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.
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
gNVBAgTA1dBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAstC01LBTSDb2
5zb2xlMRIwEAYDVQQDEw1UZXN0Q2lsYWMyHzAdBgkqhkiG9w0BCQEWEGB5vb25lQGFtYXpvbi5jb20wHhc
NMTEwNDI1MjaONTIxWhcNMTEwNDI0MjaONTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTA1dBMRAw
DgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDAS=EXAMPLE user-name@ip-192-0-2-137
```

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

Note

You can directly view and manage your CodeCommit credentials in **My Security Credentials**. For more information, see [View and Manage Your Credentials \(p. 6\)](#).

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*).

SSH keys for AWS CodeCommit			
Use SSH public keys to authenticate to AWS CodeCommit repositories. Learn more about SSH keys.			
SSH Key ID	Uploaded	Status	Actions
APKAEIBAERJR2EXAMPLE	2015-07-21 16:32 PDT	Active	Make Inactive Show SSH Key Delete

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:

```
Host git-codecommit.*.amazonaws.com
User APKAEIBAERJR2EXAMPLE
IdentityFile ~/.ssh/codecommit_rsa
```

Note

If you gave your private key file a name other than `codecommit_rsa`, be sure to use it here.

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:

```
ssh git-codecommit.us-east-2.amazonaws.com
```

You are asked to confirm the connection because `git-codecommit.us-east-2.amazonaws.com` is not yet included in your known hosts file. The CodeCommit server fingerprint is displayed as part of the verification (a9:6d:03:ed:08:42:21:be:06:e1:e0:2a:d1:75:31:5e for MD5 or 31BlW2g5xn/NA2Ck6dyeJrQOWvn7n8UEs56fG6ZIzQ for SHA256).

Note

CodeCommit server fingerprints are unique for every AWS Region. To view the server fingerprints for an AWS Region, see [Server Fingerprints for CodeCommit \(p. 308\)](#).

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:

```
ssh -v git-codecommit.us-east-2.amazonaws.com
```

For information to help you troubleshoot connection problems, see [Troubleshooting \(p. 253\)](#).

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 \(p. 304\)](#).
3. Choose the repository you want to connect to from the list. This opens the **Code** page for that repository.

If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see [the section called "Create a Repository" \(p. 82\)](#) or follow the steps in the [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.

4. Copy the SSH URL to use when connecting to the repository.
5. Open a terminal. From the /tmp directory, using the SSH URL you copied, run the **git clone** command 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 ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

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:

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

For more information, see [Access error: Public key is uploaded successfully to IAM but connection fails on Linux, macOS, or Unix systems \(p. 254\)](#).

For more information about how to connect to repositories, see [Connect to the CodeCommit Repository by Cloning the Repository \(p. 85\)](#).

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) 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 the initial configuration steps. This topic walks you through the steps for setting up your computer and AWS

profile, connecting to a CodeCommit repository, and cloning 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? \(p. 5\)](#).

Topics

- [Step 1: Initial Configuration for CodeCommit \(p. 33\)](#)
- [Step 2: Install Git \(p. 33\)](#)
- [SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit \(p. 34\)](#)
- [Step 4: Connect to the CodeCommit Console and Clone the Repository \(p. 36\)](#)
- [Next Steps \(p. 37\)](#)

Step 1: Initial Configuration for CodeCommit

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

To create and configure an IAM user for accessing CodeCommit

1. Create an AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

Note

If you want to use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

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.

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 \(p. 253\)](#).

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.

SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit

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.
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  
gNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC01BTSDb2  
5zb2x1MRIwEAYDVQQDEw1UZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWE5vb251QGFtYXpvbi5jb20wHhc  
NMTEwNDI1MjaONTIxWhcNMTIwNDI0MjaONTIxWjCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAw  
DgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDAS=EXAMPLE user-name@computer-name
```

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.

Note

You can directly view and manage your CodeCommit credentials in **My Security Credentials**. For more information, see [View and Manage Your Credentials \(p. 6\)](#).

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:

```
Host git-codecommit.*.amazonaws.com
User APKAEIBAERJR2EXAMPLE
IdentityFile ~/.ssh/codecommit_rsa
```

Note

If you gave your private key file a name other than `codecommit_rsa`, be sure to use it here.

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:

```
ssh git-codecommit.us-east-2.amazonaws.com
```

You are asked to confirm the connection because `git-codecommit.us-east-2.amazonaws.com` is not yet included in your known hosts file. The CodeCommit server fingerprint is displayed as part of the verification (`a9:6d:03:ed:08:42:21:be:06:e1:e0:2a:d1:75:31:5e` for MD5 or `31B1W2g5xn/NA2Ck6dyeJ1rQOWvn7n8UEs56fG6Z1zQ` for SHA256).

Note

CodeCommit server fingerprints are unique for every AWS Region. To view the server fingerprints for an AWS Region, see [Server Fingerprints for CodeCommit \(p. 308\)](#).

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`).

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

```
ssh -v git-codecommit.us-east-2.amazonaws.com
```

For information to help you troubleshoot connection problems, see [Troubleshooting \(p. 253\)](#).

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 \(p. 304\)](#).
3. Choose the repository you want to connect to from the list. This opens the **Code** page for that repository.

If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see [the section called "Create a Repository" \(p. 82\)](#) or follow the steps in the [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.

4. Choose **Clone URL**, and then copy the SSH URL.
5. In the Bash emulator, using the SSH URL you just copied, run the **git clone** command to clone the repository. This command creates the local repo 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` in the US East (Ohio) Region:

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

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`:

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

For more information, see [Connect to the CodeCommit Repository by Cloning the Repository \(p. 85\)](#) and [Create a Commit \(p. 185\)](#).

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) 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. This topic walks you through the steps to 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? \(p. 5\)](#).

Topics

- [Step 1: Initial Configuration for CodeCommit \(p. 37\)](#)
- [Step 2: Install Git \(p. 39\)](#)
- [Step 3: Set Up the Credential Helper \(p. 39\)](#)
- [Step 4: Connect to the CodeCommit Console and Clone the Repository \(p. 40\)](#)
- [Next Steps \(p. 41\)](#)

Step 1: Initial Configuration for CodeCommit

Follow these steps to set up an AWS 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 AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

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. 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. 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 the CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of CodeCommit commands.

3. Configure the AWS CLI with 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:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target 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
```

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

- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1

For more information about CodeCommit and AWS Regions, see [Regions and Git Connection Endpoints \(p. 304\)](#). For more information about IAM, access keys, and secret keys, see [How Do I Get Credentials?](#) and [Managing Access Keys for IAM Users](#).

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.

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 \(p. 253\)](#).

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 \(p. 253\)](#).

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 an CodeCommit repository \(p. 84\)](#). After you connect to a CodeCommit repository with HTTPS for the first time, subsequent access fails after about fifteen 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 \(p. 28\)](#).
- 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.
- Install a version of Git that does not use the keychain by default.

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 \(p. 304\)](#).
3. Choose the repository you want to connect to from the list. This opens the **Code** page for that repository.

If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see [the section called "Create a Repository" \(p. 82\)](#) or follow the steps in the [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.

4. Copy the HTTPS URL to use when connecting to the repository.
5. Open a terminal and from the /tmp directory, use the URL to clone the repository with the **git clone** command. 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
```

For more information, see [Connect to the CodeCommit Repository by Cloning the Repository \(p. 85\)](#) and [Create a Commit \(p. 185\)](#).

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) 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 \(p. 8\)](#). However, if you want to connect to CodeCommit using a root account, federated access, or temporary credentials, you must use the credential helper that is included in the AWS CLI.

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? \(p. 5\)](#).

Topics

- [Step 1: Initial Configuration for CodeCommit \(p. 42\)](#)
- [Step 2: Install Git \(p. 43\)](#)
- [Step 3: Set Up the Credential Helper \(p. 44\)](#)
- [Step 4: Connect to the CodeCommit Console and Clone the Repository \(p. 45\)](#)
- [Next Steps \(p. 46\)](#)

Step 1: Initial Configuration for CodeCommit

Follow these steps to set up an AWS 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 AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

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. 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. 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 the CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of CodeCommit commands.

3. Configure the AWS CLI with 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:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target 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
```

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
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1

For more information about CodeCommit and AWS Regions, see [Regions and Git Connection Endpoints \(p. 304\)](#). For more information about IAM, access keys, and secret keys, see [How Do I Get Credentials?](#) and [Managing Access Keys for IAM Users](#).

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.

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 \(p. 8\)](#) and [Git for Windows: I installed Git for Windows, but I am denied access to my repository \(403\) \(p. 262\)](#).

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 \(p. 253\)](#).

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 \(p. 304\)](#).
3. Choose the repository you want to connect to from the list. This opens the **Code** page for that repository.

If you see a **Welcome** page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see [the section called "Create a Repository" \(p. 82\)](#) or follow the steps in the [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.

4. Copy the HTTPS URL to use when connecting to the repository.
5. Open a command prompt and use the URL to clone the repository with 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** in the US East (Ohio) Region:

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

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**.

For more information about how to connect to repositories, see [Connect to the CodeCommit Repository by Cloning the Repository \(p. 85\)](#).

Next Steps

You have completed the prerequisites. Follow the steps in [CodeCommit Tutorial \(p. 47\)](#) to start using CodeCommit.

Getting Started with AWS CodeCommit

The easiest way to get started with CodeCommit is to follow the steps in [CodeCommit Tutorial \(p. 47\)](#). If you are new to Git as well as CodeCommit, you should also consider following the steps in [Git with CodeCommit Tutorial \(p. 64\)](#). This will help you familiarize yourself with CodeCommit as well as 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 assume you have completed the [prerequisites and setup \(p. 6\)](#), 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 Tutorial \(p. 47\)](#)
- [Git with AWS CodeCommit Tutorial \(p. 64\)](#)

Getting Started with AWS CodeCommit Tutorial

If you're new to CodeCommit, this tutorial helps you learn how to use its features. In this tutorial, you create a repository in CodeCommit. After you commit some changes to the CodeCommit repository, you browse the files and view the changes. You can also create a pull request for others to review and comment on changes to your code.

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 screenshot shows the AWS CodeCommit console interface. At the top, there's a navigation bar with 'Developer Tools > CodeCommit > Repositories > MyDemoRepo'. Below the navigation is the repository name 'MyDemoRepo' in large bold letters. There are three buttons: 'master' (with a dropdown arrow), 'Create pull request', and 'Clone URL'. A note below says 'View contents of your repository. You can view your repository files and subdirectories by choosing them. Change the branch to view files as they exist in that branch.' The main area displays a file named 'cl_sample.js' with the following code:

```
1 var aws = require('aws-sdk');
2 var codecommit = new aws.CodeCommit({ apiVersion: '2015-04-13'});
3
4 exports.handler = function(event, context) {
5     //Log the updated references from the event
6     var references = event.Records[0].codecommit.references.map(function(reference)
7         console.log('References:', references);
8
9     //Get the repository from the event and show its git clone URL
10    var repository = event.Records[0].eventSourceARN.split(":")[5];
11    var params = {
12        repositoryName: repository
13    };
14    codecommitgetRepository(params, function(err, data) {
15        if (err) {
16            console.log(err);
17            var message = "Error getting repository metadata for repository " + repository;
18            console.log(message);
19            context.fail(message);
20        }
21    });
22}
23
```

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 \(p. 278\)](#).

If you are not familiar with Git, you might want to complete the [Git with CodeCommit Tutorial \(p. 64\)](#) in addition to this tutorial. After you finish this tutorial, you should have enough practice to start using CodeCommit for your own projects and in team environments.

Important

Before you begin, you must complete the [prerequisites and setup \(p. 6\)](#), 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 \(p. 49\)](#)
- [Step 2: Add Files to Your Repository \(p. 50\)](#)

- [Step 3: Browse the Contents of Your Repository \(p. 52\)](#)
- [Step 4: Create and Collaborate on a Pull Request \(p. 58\)](#)
- [Step 5: Next Steps \(p. 63\)](#)
- [Step 6: Clean Up \(p. 63\)](#)

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. In the region selector, choose the AWS Region where you want to create the repository. For more information, see [Regions and Git Connection Endpoints \(p. 304\)](#).
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 \(p. 313\)](#).

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 \(p. 96\)](#).
7. Choose **Create**.

Developer Tools > CodeCommit > Repositories > Create repository

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description. Repository names are included in the URLs for that repository.

Repository settings

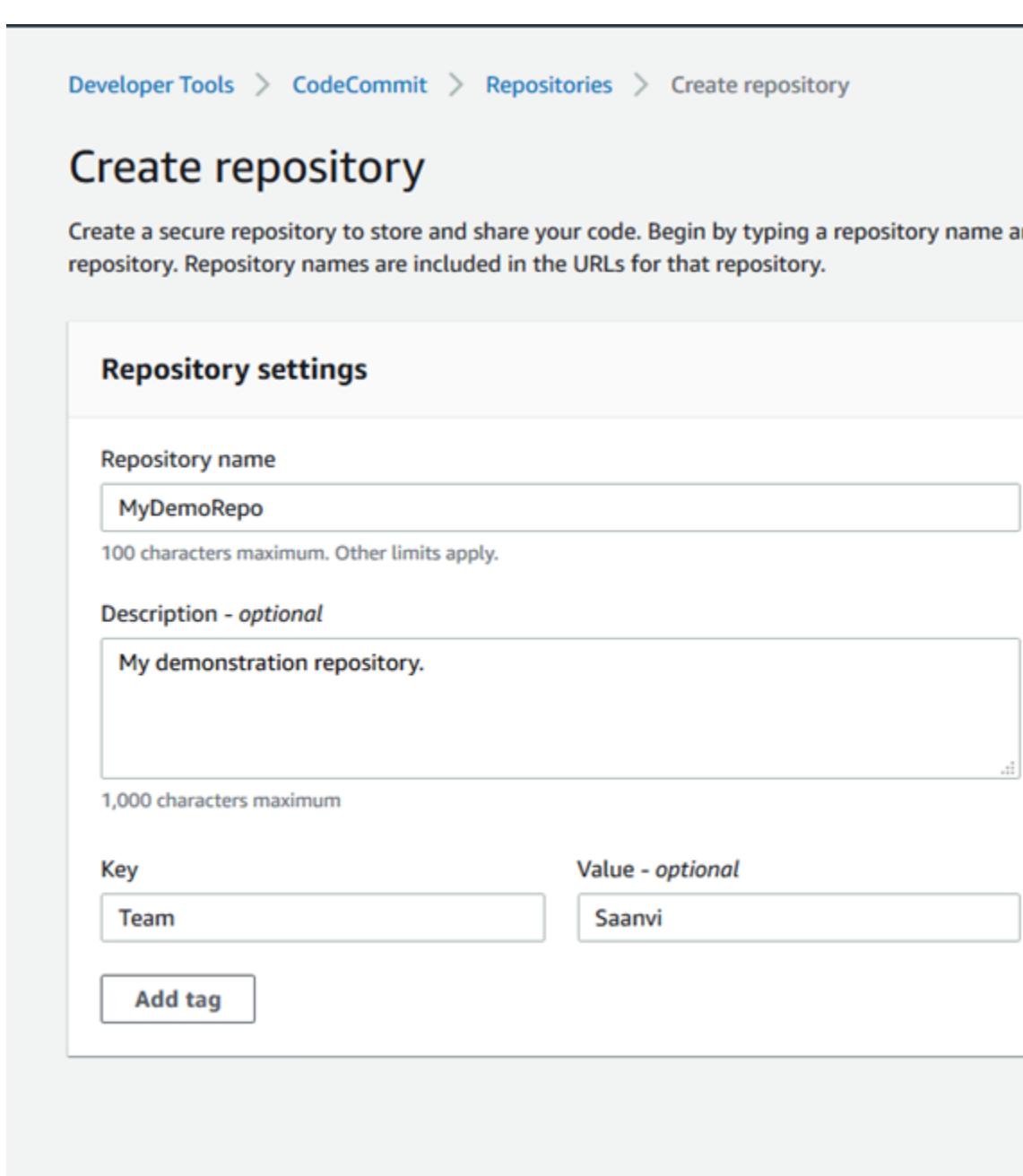
Repository name
MyDemoRepo
100 characters maximum. Other limits apply.

Description - optional
My demonstration repository.
1,000 characters maximum

Key	Value - optional	Remove
Team	Saanvi	Remove

[Add tag](#)

[Cancel](#)



Note

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

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.

- Uploading a file from your local computer using the CodeCommit console.
- Using a Git client to clone the repository to your local computer, and then adding, committing, and pushing files to the CodeCommit repository.

The simplest way to get started is to add a file from the CodeCommit console.

1. In the navigation bar for the repository, choose **Code**.
2. Choose **Add file**, and then choose to create or upload a file from your computer.
3. Do the following:
 - If you want to add the file to a different branch, from 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 *master*.
 - 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.
 - If you're uploading a file, choose the file you want to upload.
 - If you're creating a file, in **File name**, enter a name for the file, and then in the code editor, enter the code you want to add.

Upload a file

Choose the branch where you will upload the file, and then choose the file to upload. Add your user information and a comment about why you added this file.

MyDemoRepo		
Name	Size	Actions
		Upload file Choose a file to upload. <input type="button" value="Choose file"/>

Commit changes to master	
Author name	<input type="text" value="Maria Garcia"/>
Email address	<input type="text" value="maria_garcia@example.com"/>
Commit message - optional	<small>A default commit message will be used if you do not provide one.</small>
<input type="text" value="Adding my first file to the repository."/>	

4. Choose **Commit changes**.

For more information, see [Working with Files in AWS CodeCommit Repositories \(p. 140\)](#).

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 [Git with CodeCommit Tutorial \(p. 64\)](#). 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 \(p. 185\)](#), [Step 2: Create a Local Repo \(p. 65\)](#), or [Connect to a Repository \(p. 84\)](#).

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.

1. From **Repositories**, choose MyDemoRepo.
2. The contents of the repository are displayed in the default branch for your repository. To change the view to another branch, or to view the code at a specific tag, choose the view selector button, and then choose the branch or tag you want to view from the list. Here the view is set to the **master** branch.

The screenshot shows the AWS CodeCommit console interface. On the left, there is a sidebar with a tree view of services: 'Source > CodeCommit', 'Getting started', 'Repositories', 'Code' (which is expanded), 'Pull requests', 'Commits', 'Branches', 'Tags', and 'Settings'. Below these are sections for 'Build > CodeBuild', 'Deploy > CodeDeploy', and 'Pipeline > CodePipeline'. The main area is titled 'MyDemoRepo' and shows the 'Info' tab selected. At the top right, there is a dropdown menu set to 'master' and a 'Create pull request' button. The main content area displays a list of files in the 'master' branch:

Name
anothernew
batch files for https
batch files for ssh
new
ahs_count.py
apis_meliponini.txt
bees.txt
bird.txt
bumblebee.txt
cat.txt

3. To view the contents of a file in your repository, choose the file from the list. Choose the gear icon to change the color display option of the displayed code.

The screenshot shows the AWS CodeCommit interface. On the left, a sidebar titled 'Developer Tools' has 'CodeCommit' selected. Under 'Source', 'Code' is also selected. The main content area is titled 'MyDemoRepo'. At the top right, there's a dropdown set to 'master' and a button for 'Create pull request'. Below the title, it says 'MyDemoRepo / cl_sample.js' with an 'Info' link. The code in 'cl_sample.js' is:

```
1 var aws = require('aws-sdk');
2 var codecommit = new aws.CodeCommit({ apiVersion: '2015-04-13' });
3
4 exports.handler = function(event, context) {
5     //Log the updated references from the event
6     var references = event.Records[0].codecommit.references.map(function(reference)
7         console.log('References:', reference);
8
9     //Get the repository from the event and show its git clone URL
10    var repository = event.Records[0].eventSourceARN.split(":")[-1];
11    var params = {
12        repositoryName: repository
13    };
14    codecommitgetRepository(params, function(err, data) {
15        if (err) {
16            console.log(err);
17            var message = "Error getting repository metadata for repository " + repository;
18            console.log(message);
19            context.fail(message);
20        } else {
21            console.log("Clone URL", data.repositoryMetadata.cloneUrlHttp);
22            context.succeed(data.repositoryMetadata.cloneUrlHttp);
23        }
24    });
25};
```

For more information, see [Browse Files in a Repository \(p. 141\)](#).

You can also browse the commit history of a repository. This helps you identify changes made in a repository, including when and by whom those changes were made.

1. In the navigation pane for a repository, choose **Commits**. A history of commits for the repository in the default branch is displayed, in reverse chronological order.

The screenshot shows the AWS CodeCommit interface. On the left, there's a sidebar with 'Developer Tools' at the top, followed by 'CodeCommit'. Under 'Source', there are links for 'Getting started', 'Repositories', 'Code', 'Pull requests', and 'Commits' (which is highlighted in orange). Below 'Source' are sections for 'Build', 'Deploy', and 'Pipeline'. The main content area is titled 'MyDemoRepo' and has tabs for 'Commits' (which is selected), 'Commit visualizer', and 'Compare commits'. The 'Commits' tab has a sub-tab 'Info'. Below this is a table with the following data:

Commit ID	Commit message	Commit date	Author
6b65eb76	Added a ruby sample that does the same thing as Mary's and Saanvi's samples.	Just now	Maria Garcia
831b2d9e	Added a quick C++ sample.	2 minutes ago	Mary Major
1003b80b	Added a quick sample in python and created the samples folder for the team to us...	4 minutes ago	Saanvi Sarkar
a8ac2537			Saanvi Sarkar
e52347fb			Saanvi Sarkar
e9d894d5			Saanvi Sarkar
53faa89d			Maria Garcia
92470856			Maria Garcia

2. Review the commit history by [branch \(p. 215\)](#) or by [tag \(p. 211\)](#), and get details about commits by author, date, and more.
3. To view the differences between a commit and its parent, 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 \(p. 229\)](#).

The screenshot shows the AWS CodeCommit interface. At the top, a breadcrumb navigation path is visible: Developer Tools > CodeCommit > Repositories > MyDemoRepo > Commits > 7d09e44c. Below this, the title "Commit 7d09e44c" is displayed, along with two buttons: "Copy commit ID" and "Browse".

The main content area is titled "Details" and contains the following information:

Author	Commit date	Parent commit
Mary Major mary_major@example.com	48 minutes ago	e6aca768

Below the author information, there is a "Commit message" section with the text: "Adding a readme file to the repository."

At the bottom of the commit details, there are navigation controls: "< Page 1 of 1 >" and "Go to file" with a dropdown arrow. To the right are two toggle buttons: "Hide whitespace changes" (which is selected) and "Untracked files".

The bottom half of the screenshot shows the content of the "readme.md" file. The file content is:

```
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
    \ No newline at end of file
```

Next to the file content are two buttons: "Browse file contents" and "Comments".

4. 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.

ahs_count.py

[Browse file contents](#)

```
*** @ -4,7 +4,7 @@
4   z = z.count('z')
5
6   total = (ess + z)
7 - alv = "Number of alveolar hissing sibilants: {}"
7 + ahs = "Number of alveolar hissing sibilants: {}"
```

 Li Juan commented Just now

You've reverted to the old value he should remain alv.

[Reply](#) [Edit](#)

[New comment](#)

```
8 print(alv.format(total))
9
10 #When using this script, make sure that you ask
```

```
8 print(ahs.format(total))
9
10 #When using this script,
```

For more information, see [Comment on a Commit \(p. 201\)](#).

5. 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**.

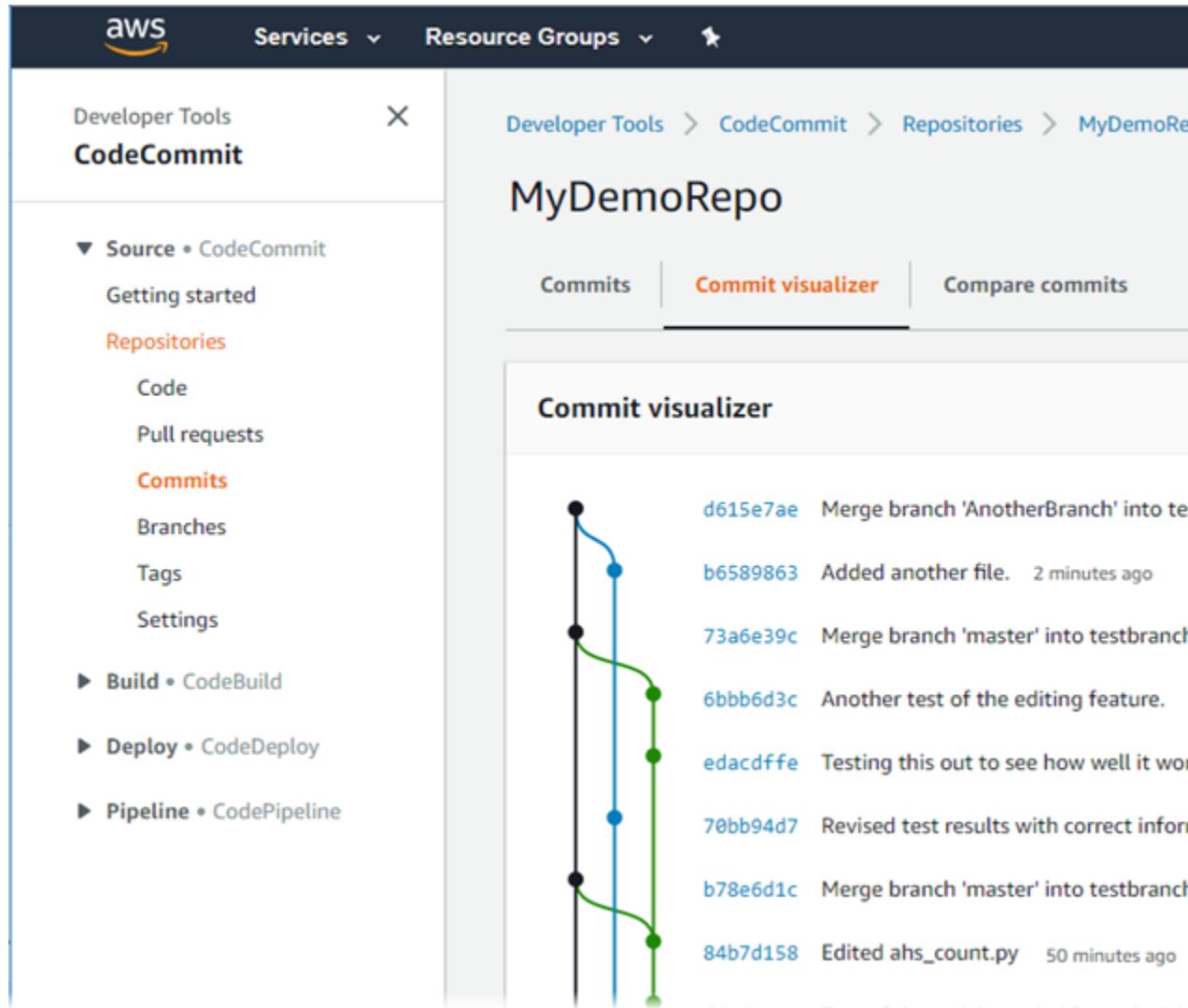
The screenshot shows the AWS CodeCommit interface for comparing repository contents. At the top, a breadcrumb navigation path is visible: Developer Tools > CodeCommit > Repositories > MyDemoRepo > Compare. Below this, the repository name "MyDemoRepo" is displayed. A navigation bar includes tabs for "Commits", "Commit visualizer", and "Compare commits", with "Compare commits" being the active tab. Underneath, there are dropdown menus for "Destination" set to "AnotherBranch" and "Source" set to the commit hash "6b65eb76". A prominent orange "Compare" button is located next to the source dropdown. To the right of the compare button are "Cancel" and "Hide whitespace changes" (with a checked checkbox). Below the compare controls, there are navigation links for "Page 1 of 1" and "Go to file", along with a dropdown menu. The main content area displays a diff for the file "ahs_count.py". The diff highlights changes in red (deletions) and green (additions). The code snippet is as follows:

```
*** *** @@ -5,6 +5,6 @@
 5   5
 6   6     total = (ess + z)
 7   7     ahs = "Number of alveolar hissing sibilants: {}"
 8 - print(ahs.format(total))
 8 + print(alv.format(total))
 9   9
10  10    #When using this script, make sure that you ask the subject to use one of the provided texts, suc
```

Below the "ahs_count.py" diff, another file "anothernew/dir2/anotherfile.txt" is listed with the status "Added".

For more information, see [Browse the Commit History of a Repository \(p. 189\)](#) and [Compare Commits \(p. 196\)](#).

6. 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.

7. 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 \(p. 190\)](#).

Step 4: Create and Collaborate on a Pull Request

When you work with other repository users, you might want to 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 before you merge those changes into another 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 \(p. 90\)](#).

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 \(p. 216\)](#).

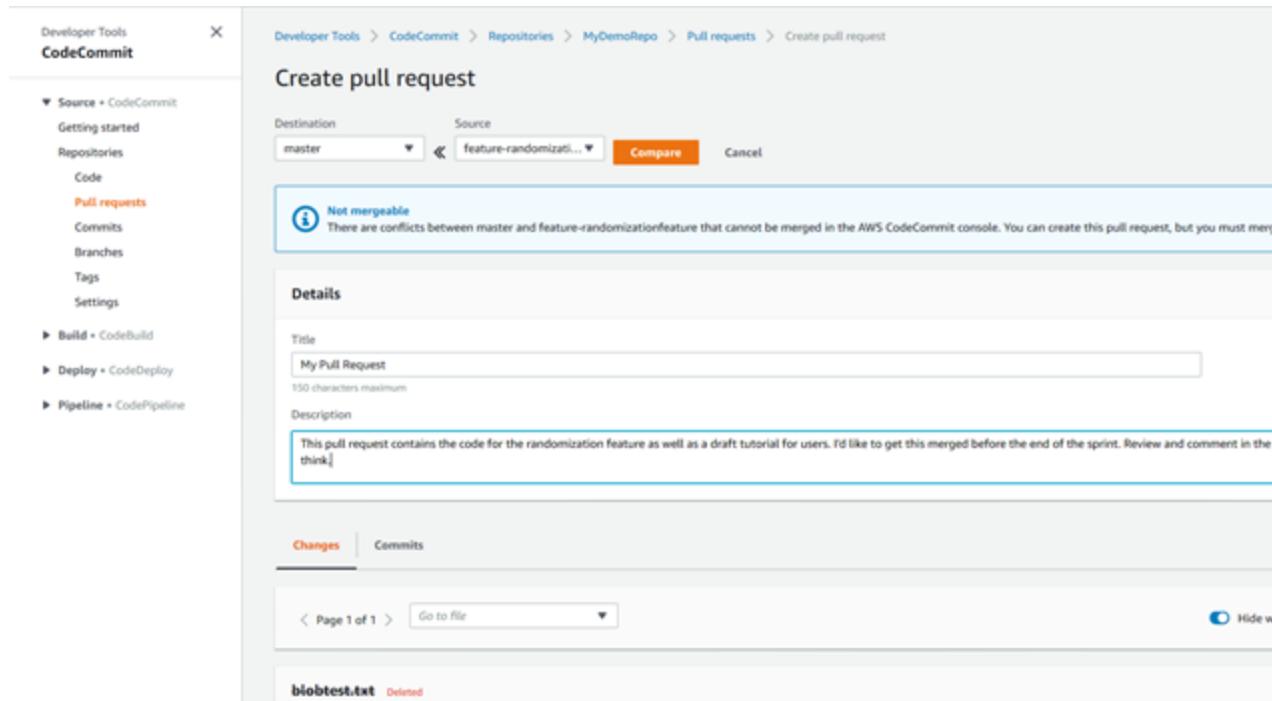
1. In the navigation pane, choose **Pull requests**.
2. In **Pull request**, choose **Create pull request**.

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.

The screenshot shows the AWS CodeCommit interface. On the left, a sidebar menu for 'CodeCommit' includes 'Source' (selected), 'Getting started', 'Repositories', 'Code', 'Pull requests' (selected), 'Commits', 'Branches', 'Tags', and 'Settings'. Below these are sections for 'Build' (CodeBuild), 'Deploy' (CodeDeploy), and 'Pipeline' (CodePipeline). The main content area is titled 'MyDemoRepo' and shows a 'Pull requests' list. The list header includes 'Pull requests' and 'Info' buttons, and a 'All pull requests' link. The table columns are 'Pull request', 'Author', 'Destination', and 'Last activity'. Six pull requests are listed: 6: My Pull Request (Author: [redacted], Destination: master, Last activity: Just now); 4:, 1:, 5:, 3:, and 2: (all blurred out).

5. 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. If necessary, you can push changes to the pull request branch, which updates the pull request.

The screenshot shows the AWS CodeCommit Activity tab for a pull request. It displays three comments:

- Comment on line 12 of second-randomizer.py**

reroll = raw_input("Reroll?")

 commented 4 minutes ago

Are we sure this is the approved string?

[Reply](#) [Edit](#)
- Comment on [REDACTED]**

 commented 5 minutes ago

I'm not sure why you removed this test.

[Reply](#) [Edit](#)
- Comment on changes**

 commented 5 minutes ago

I think our users will find this a fun feature, but I'm not convinced it's fully formed. Did you double-check against the user stories and

[Reply](#) [Edit](#)

6. 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 merge the branches as part of closing 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.

Merge pull request 9: Bug fix for unhandled exception

Merge request details

Pull request: #9 Bug fix for unhandled exception

Destination master << Source bugfix-bug1234

Merge strategy [Info](#)

Determines the way in which the current pull request will be merged into the destination branch

Fast forward merge

`git merge --ff-only`

Merges the branches and moves the destination branch pointer to the tip of the source branch. This is the default merge strategy in Git.



Squash and merge

`git merge --squash`

Combine all commits from the source branch into a single merge commit in the destination branch.



Commit message - *optional*

Squashed commit of the following

commit d49940ad

Author: Li Juan <li_juan@example.com>

Date: Tue May 07 2019 15:12:48 GMT-0700 (Pacific Daylight Time)

Fixing the bug reported in 1234.

Author name

Maria Garcia

Email address

maria_garcia@example.com

Delete source branch bugfix-bug1234 after merging?

[Cancel](#)

- If you want to close the pull request without merging branches, choose **Close pull request**.
- If there are merge conflicts in the branches that cannot be resolved automatically, you can try to 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 \(p. 173\)](#).

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 \(p. 149\)](#).

Step 5: 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 [Git with CodeCommit Tutorial \(p. 64\)](#) tutorial.
- If you want to work with others in a CodeCommit repository, see [Share a Repository \(p. 87\)](#). (If you want to share your repository with users in another AWS account, see [Configure Cross-Account Access to an AWS CodeCommit Repository \(p. 129\)](#).)
- If you want to migrate a repository to CodeCommit, follow the steps in [Migrate to CodeCommit \(p. 230\)](#).
- 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 \(p. 73\)](#).

Step 6: Clean Up

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

Important

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. This action cannot be undone.

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 \(p. 118\)](#).

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.

Git with AWS CodeCommit Tutorial

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 [CodeCommit Tutorial \(p. 47\)](#).

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 \(p. 6\)](#), including:

- Assign permissions to the IAM user.
- Set up credential management for HTTPS or SSH connections on the local machine you use for this tutorial.
- 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 \(p. 64\)](#)
- [Step 2: Create a Local Repo \(p. 65\)](#)
- [Step 3: Create Your First Commit \(p. 65\)](#)
- [Step 4: Push Your First Commit \(p. 66\)](#)
- [Step 5: Share the CodeCommit Repository and Push and Pull Another Commit \(p. 66\)](#)
- [Step 6: Create and Share a Branch \(p. 68\)](#)
- [Step 7: Create and Share a Tag \(p. 69\)](#)
- [Step 8: Set Up Access Permissions \(p. 70\)](#)
- [Step 9: Clean Up \(p. 72\)](#)

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. In the region selector, choose the AWS Region where you want to create the repository. For more information, see [Regions and Git Connection Endpoints \(p. 304\)](#).
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 \(p. 313\)](#).

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 \(p. 96\)](#).
7. Choose **Create**.

Note

The remaining steps in this tutorial use **MyDemoRepo** for the name of your 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 \(p. 82\)](#).

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 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 \(p. 304\)](#).
3. On the **Dashboard** page, choose the name of the repository you want to share.
4. On the **Code** page, choose **Clone URL**, and then choose the protocol you want your users to use.
5. Copy the displayed URL for the connection protocol your users will use when connecting to your CodeCommit repository.
6. Send your users the connection information 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, for HTTPS, configuring the credential helper for Git).

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 add** to stage the change:

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

3. 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 (**master**):

```
git push -u origin master
```

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 \(p. 141\)](#).

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 \(p. 65\)](#). (Typically, this directory would be 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 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 must 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 \(p. 32\)](#).

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* (for example, John Doe and johndoe@example.com). This makes it easier to identify the commits the other user made:

```
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 (`master`):

```
git push -u origin master
```

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 `master` 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 `master` branch and view its list of commits. The Git commands should look like this:

```
git checkout master  
git log
```

5. Switch to the `master` 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 `master`. The Git commands should look like this:

```
git checkout master  
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.

To do this, you use the IAM console to create an IAM 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 assumes you have permissions to create IAM users in your AWS account. If you don't have these permissions, then you can't perform the procedures in this step. Skip ahead to [Step 9: Clean Up \(p. 72\)](#) 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://console.aws.amazon.com/iam/>.

Be sure to sign in with the same user name and password you used in [Setting Up \(p. 6\)](#).

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 \(p. 8\)](#) 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 \(p. 29\)](#) or [SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit \(p. 34\)](#) 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:

```
{  
    "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:

```
{  
    "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 \(p. 270\)](#).

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 credentials. For more information, see the instructions in [Step 3: Set Up the Credential Helper \(p. 39\)](#) or [Step 3: Set Up the Credential Helper \(p. 44\)](#).

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 \(p. 29\)](#) or [SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit \(p. 34\)](#).

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**, type **MyDemoRepo**, and then choose **Delete**.

To delete the CodeCommit repository (AWS CLI)

Run the [delete-repository \(p. 139\)](#) 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.

Note

You can automatically build and deploy commits to a CodeCommit repository by integrating with CodePipeline. To learn more, follow the steps in the [AWS for DevOps Getting Started Guide](#).

Topics

- [Integration with Other AWS Services \(p. 73\)](#)
- [Integration Examples from the Community \(p. 78\)](#)

Integration with Other AWS Services

CodeCommit is integrated with the following AWS services:

AWS Amplify	<p>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.</p> <p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• AWS Amplify User Guide• Getting Started
AWS Cloud9	<p>AWS Cloud9 contains a collection of tools that you use to code, build, run, test, debug, and release software in the cloud. This collection of tools is referred to as the AWS Cloud9 integrated development environment, or IDE.</p>

	<p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• AWS Cloud9 User Guide• AWS CodeCommit Sample for AWS Cloud9• Integrate AWS Cloud9 with AWS CodeCommit (p. 15)
AWS CloudFormation	<p>AWS 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 AWS CloudFormation takes care of provisioning and configuring those resources for you.</p> <p>Learn more:</p> <ul style="list-style-type: none">• AWS CodeCommit Repository resource page
AWS CloudTrail	<p>CloudTrail captures AWS API calls and related events made by or on behalf of an AWS 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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Logging AWS CodeCommit API Calls with AWS CloudTrail (p. 319)

Amazon CloudWatch Events	<p>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.</p> <p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• CloudWatch Events User Guide• AWS CodeCommit Events• Blog post: Build Serverless AWS CodeCommit Workflows using Amazon CloudWatch Events and JGit
AWS CodeBuild	<p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Plan a Build• Create a Build Project• Use CodePipeline with AWS CodeBuild to Run Builds

AWS CodePipeline	<p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• 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 CodeStar	<p>AWS CodeStar is a cloud-based service for creating, managing, and working with software development projects on AWS. You can quickly develop, build, and deploy applications on AWS with an AWS CodeStar project. An AWS CodeStar project creates and integrates AWS services for your project development toolchain, including a CodeCommit repository for the project. AWS CodeStar also assigns permissions to team members for that project. These permissions are applied automatically, including permissions for accessing CodeCommit, creating and managing Git credentials, and more.</p> <p>You can configure repositories created for AWS CodeStar projects just as you would any other CodeCommit repository by using the AWS CodeCommit console, CodeCommit commands from the AWS CLI, the local Git client, and from the CodeCommit API.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Working with Repositories (p. 81)• Working with AWS CodeStar Projects• Working with AWS CodeStar Teams

AWS Elastic Beanstalk	<p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Using the EB CLI with AWS CodeCommit• Using an Existing AWS CodeCommit Repository• eb codesource (EB CLI command)
AWS Key Management Service	<p>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.</p> <p>Learn more:</p> <ul style="list-style-type: none">• AWS KMS and Encryption (p. 318)
AWS Lambda	<p>Lambda lets you run code without provisioning or managing servers. You can configure triggers for CodeCommit repositories that invoke Lambda functions in response to repository events.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Create a Trigger for a Lambda Function (p. 107)• AWS Lambda Developer Guide
Amazon Simple Notification Service	<p>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 notifications to integrate with other AWS services. For example, you can use an Amazon SNS notification to send messages to an Amazon Simple Queue Service queue.</p> <p>Learn more:</p> <ul style="list-style-type: none">• Create a Trigger for an Amazon SNS Topic (p. 102)• 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 \(p. 78\)](#)
- [Code Samples \(p. 80\)](#)

Blog Posts

- [**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**](#)

API Version 2015-04-13

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

- [Replicating and Automating Sync-Ups for a Repository with AWS CodeCommit](#)

Learn how to back up or replicate a CodeCommit repository to another AWS region, and how to back up repositories hosted on other services to CodeCommit.

Published March 17, 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 AWS 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 AWS OpsWorks and AWS CodeCommit](#)

Learn how AWS 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 Coetze

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.

The screenshot shows the AWS CodeCommit interface. On the left, the navigation pane includes sections for Developer Tools, CodeCommit, Source (CodeCommit), Getting started, Repositories, Code (selected), Pull requests, Commits, Branches, Tags, and Settings. Under Build, Deploy, and Pipeline, there are links to CodeBuild, CodeDeploy, and CodePipeline respectively. The main content area shows the 'MyDemoRepo' repository details, including a dropdown for the branch (set to master) and a 'Create pull request' button. Below this, a table lists the repository's contents:

Name
anothernew
batch files for https
batch files for ssh
new
ahs_count.py
apis_meliponini.txt
bees.txt
bird.txt
bumblebee.txt
cat.txt

Before you can push changes to a CodeCommit repository, you must configure your IAM user in your AWS account. For more information, see [Step 1: Initial Configuration for CodeCommit \(p. 9\)](#).

For information about working with other aspects of your repository in CodeCommit, see [Working with Files \(p. 140\)](#), [Working with Pull Requests \(p. 149\)](#), [Working with Commits \(p. 184\)](#), [Working with Branches \(p. 215\)](#), and [Working with User Preferences \(p. 229\)](#). For information about migrating to CodeCommit, see [Migrate to CodeCommit \(p. 230\)](#).

Topics

- [Create an AWS CodeCommit Repository \(p. 82\)](#)
- [Connect to an AWS CodeCommit Repository \(p. 84\)](#)
- [Share a AWS CodeCommit Repository \(p. 87\)](#)
- [Configuring Notifications for Events in an AWS CodeCommit Repository \(p. 90\)](#)
- [Tagging Repositories in AWS CodeCommit \(p. 96\)](#)
- [Manage Triggers for an AWS CodeCommit Repository \(p. 101\)](#)
- [View CodeCommit Repository Details \(p. 119\)](#)
- [Change AWS CodeCommit Repository Settings \(p. 123\)](#)
- [Synchronize Changes Between a Local Repo and an AWS CodeCommit Repository \(p. 126\)](#)
- [Push Commits to an Additional Git Repository \(p. 127\)](#)
- [Configure Cross-Account Access to an AWS CodeCommit Repository \(p. 129\)](#)
- [Delete an AWS CodeCommit Repository \(p. 138\)](#)

Create an AWS CodeCommit Repository

Use AWS CLI or the CodeCommit console to create an empty CodeCommit repository. If you use the AWS CLI to create a CodeCommit repository, you can add tags to the repository as part of creating it. To add tags to a repository after you create it, see [Add a Tag to a Repository \(p. 96\)](#).

These instructions are written with the assumption that you have already completed the steps in [Setting Up \(p. 6\)](#).

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 \(Console\) \(p. 82\)](#)
- [Create a Repository \(AWS CLI\) \(p. 83\)](#)

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 \(p. 304\)](#).
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 AWS 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 \(p. 96\)](#).
7. 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 \(p. 6\)](#). 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

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 AWS 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 \(p. 304\)](#).

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 AWS 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).

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`:

```
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 AWS account associated with the CodeCommit repository (`accountId`).

Here is some 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",  
        "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 \(p. 98\)](#) 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 \(AWS CLI\) \(p. 121\)](#).

After you create a repository, you can connect to it and start adding code. For more information, see [Connect to a Repository \(p. 84\)](#). 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 \(p. 143\)](#) to and [edit files \(p. 146\)](#) 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 \(p. 230\)](#).

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 \(p. 85\)](#)
- [Connect to the CodeCommit Repository by Cloning the Repository \(p. 85\)](#)
- [Connect a Local Repo to the CodeCommit Repository \(p. 86\)](#)

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. For more information, see [Setting Up \(p. 6\)](#).
- You must have the clone URL of the CodeCommit repository to which you want to connect. This URL includes the name of the repository and its AWS Region. For more information, see [View Repository Details \(p. 119\)](#).

If you have not yet created a CodeCommit repository, follow the instructions in [Create a Repository \(p. 82\)](#), 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 \(p. 119\)](#).

- 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 \(p. 6\)](#).

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 example shows how to clone a repository named `MyDemoRepo` in the US East (Ohio) Region:

For HTTPS:

```
git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/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 \(p. 304\)](#).

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 \(p. 32\)](#) and [Troubleshooting \(p. 253\)](#).

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 \(p. 6\)](#), 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 \(p. 127\)](#) instead.

1. Complete the [prerequisites \(p. 85\)](#).
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, 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.

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 \(p. 142\)](#) and [Edit the Contents of a File in an AWS CodeCommit Repository \(p. 145\)](#).

These instructions are written with the assumption that you have already completed the steps in [Setting Up \(p. 6\)](#) and [Create a Repository \(p. 82\)](#).

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 \(p. 87\)](#)
- [Create IAM Policies for Your Repository \(p. 88\)](#)
- [Create an IAM Group for Repository Users \(p. 89\)](#)
- [Share the Connection Information with Your Users \(p. 89\)](#)

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, which your repository users must configure in the credential helper included in the AWS CLI. (This is the only method 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, and remote repositories, see [Setting Up \(p. 6\)](#) or consult your Git documentation. For a general overview of communication protocols and how each communicates with remote repositories, see [Git on the Server - The Protocols](#).

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 AWS 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 \(p. 270\)](#).

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page, next to **Copy an AWS Managed Policy**, choose **Select**.
4. On the **Copy an AWS Managed Policy** page, in **Search Policies**, enter **AWSCodeCommitPowerUser**. Choose **Select** next to the policy name.
5. On the **Review Policy** page, in **Policy Name**, enter a new name for the policy (for example, **AWSCodeCommitPowerUser-MyDemoRepo**).

In **Policy Document**, replace the "*" portion of the **Resource** line 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 and choose the repository name from the list. For more information, see [View Repository Details \(p. 119\)](#).

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"  
]
```

6. Choose **Validate Policy**. After the policy is validated, 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.

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://console.aws.amazon.com/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 AWS 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 AWS 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 AWS 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 \(p. 304\)](#).
3. On the **Repositories** page, find the name of the repository you want to share.
4. In **Clone URL**, choose the protocol (HTTPS or SSH) 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, for HTTPS, configuring the credential helper for Git).

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://console.aws.amazon.com/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 \(p. 65\).
```

You can find complete setup instructions in [Setting Up \(p. 6\)](#).

Configuring Notifications for Events in an AWS CodeCommit Repository

You can set up notifications for a repository so that repository users receive emails about the repository event types you specify. When you configure notifications, CodeCommit creates an Amazon CloudWatch Events rule for your repository. This rule responds to the event types you select from the preconfigured options in the CodeCommit console. Notifications are sent when events match the rule settings. You can create an Amazon SNS topic to use for notifications or use an existing one in your AWS account.

You use the CodeCommit console to configure notifications.

SNS topic	CloudWatch event rule	Event status
Enabled	Not enabled	<input checked="" type="checkbox"/> Enabled
		Commit comment events Notify subscribers when commits to commits.

Topics

- [Using Repository Notifications \(p. 91\)](#)
- [Configure Repository Notifications \(p. 92\)](#)
- [Change, Disable, or Enable Notifications \(p. 94\)](#)
- [Delete Notification Settings for a Repository \(p. 95\)](#)

Using Repository Notifications

Configuring notifications helps your repository users by sending emails when someone takes an action that affects another user. For example, you can configure a repository to send notifications when comments are made on commits. In this configuration, when a repository user comments on a line of code in a commit, other repository users receive an email. They can sign in and view the comment. Responses to comments also generate emails, so repository users stay informed.

Notification event types are grouped into the following categories:

- **Pull request update events:** If you select this option, users receive emails when:
 - A pull request is created or closed.
 - A pull request is updated with code changes.
 - The title or description of the pull request changes.
- **Pull request comment events:** If you select this option, users receive emails when someone comments or replies to a comment in a pull request.
- **Commit comment events:** If you select this option, users receive emails when someone comments on a commit outside of a pull request. This includes comments on:
 - Lines of code in a commit.
 - Files in a commit.

- The commit itself.

For more information, see [Comment on a Commit \(p. 201\)](#).

Repository notifications are different from repository triggers. Although you can configure a trigger to use Amazon SNS to send emails about some repository events, those events are limited to operational events, such as creating branches and pushing code to a branch. Triggers do not use CloudWatch Events rules to evaluate repository events. They are more limited in scope. For more information about using triggers, see [Manage Triggers for a Repository \(p. 101\)](#).

Configure Repository Notifications

You can keep repository users informed of repository events by configuring notifications. When you configure notifications, subscribed users receive emails about the events that you specify, such as when someone comments on a commit. For more information, see [Using Repository Notifications \(p. 91\)](#).

To use the AWS CodeCommit console to configure notifications for a repository in CodeCommit, you must have the following managed policies or the equivalent permissions attached to your IAM user:

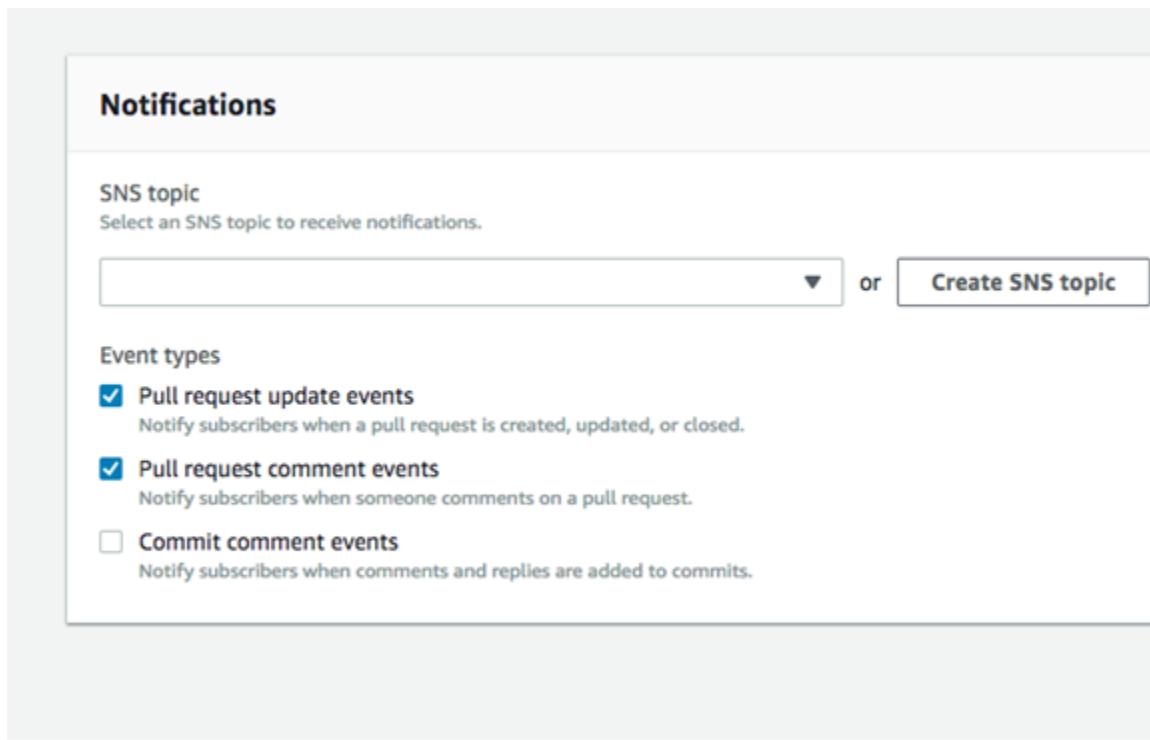
- **CloudWatchEventsFullAccess**
- **AmazonSNSFullAccess**

Note

Equivalent permissions are included in the **AWSCodeCommitFullAccess** policy, which is required to configure repository notifications. If you have this policy applied, you do not need the other two policies. If you have a customized policy applied, you might need to modify it to include the permissions required for CloudWatch Events and Amazon SNS.

To configure notifications for a 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 configure notifications.
3. In the navigation pane, choose **Settings**. Choose **Notifications**.
4. Choose **Set up**.
5. Select the event types you want included in the CloudWatch Events rule for the repository.



6. In **SNS topic**, either choose a topic from the list of Amazon SNS topics in your AWS account, or create one to use for this repository.

Note

If you create a topic, you can manage subscriptions for that policy from the CodeCommit console. If you use an existing topic, you cannot manage subscriptions for that topic unless you have permissions to manage subscriptions for all topics in Amazon SNS. For more information, see [Amazon Simple Notification Service Developer Guide](#).

If you create a topic, in **Topic name**, enter a name for the topic after the underscore. (The first part of the topic name is populated for you. Keep this first part of the name.) In **Display name**, enter an optional short name. Choose **Create**.

7. To add email addresses of the repository users, in **Subscribers**, choose **Add**. In **Add email subscriber**, enter the email address of a repository user, and then choose **Save**. You can add only one email address at a time.

Note

A confirmation email is sent to the address as soon as you choose **Save**. However, the status of the subscription is not updated while you remain in **Manage subscriptions**.

After you have added all the email addresses to the list of subscribers, choose **Close**.

Tip

Amazon SNS coordinates and manages the delivery and sending of messages to subscribing endpoints and email addresses. Endpoints include web servers, email addresses, Amazon Simple Queue Service queues, and AWS Lambda functions. For more information, see [What Is Amazon Simple Notification Service?](#) and [Sending Amazon SNS Messages to HTTP/HTTPS Endpoints](#) in the [Amazon SNS Developer Guide](#).

8. To finish configuring notifications, choose **Save**.

After you have configured notifications for a repository, you can view the CloudWatch Events rule automatically created for the repository.

Important

Do not edit or delete this rule. Changing or deleting the rule might cause operational issues. For example, emails might not be sent to subscribers or you might not be able to change notification settings for a repository in CodeCommit.

To view the CloudWatch Events rule for a repository

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation bar, under **Events**, choose **Rules**.
3. Choose the rule for your repository. The rule name is displayed on the **Notifications** tab in your repository settings.
4. View the rule summary information.

Important

Do not edit, delete, or disable this rule.

Change, Disable, or Enable Notifications

You can use the AWS CodeCommit console to change how notifications are configured, including the event types that send emails to users and the Amazon SNS topic used to send emails about the repository. You can also use the CodeCommit console to manage the list of email addresses and endpoints subscribed to the topic or to temporarily disable notifications.

To change notification settings

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 configure notifications.
3. In the navigation pane, choose **Settings**, and then choose **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. For example, you might want to disable notifications while you are performing repository maintenance. Because the configuration is preserved, you can quickly enable notifications when you are ready.

To permanently delete the notification settings, follow the steps in [Delete Notification Settings for a Repository \(p. 95\)](#).

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**.
4. Choose **Disable**.
5. The notification state 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.

To enable 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 notifications are disabled.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**.
4. Choose **Enable**.
5. The notification state changes to **Enabled**. Emails about events are sent. The CloudWatch Events rule for the repository is enabled automatically.

Delete Notification Settings for a Repository

If you no longer want notifications for your repository, you can delete the settings. Deleting the settings also deletes the CloudWatch Events rule created for repository notifications. It does not delete any subscriptions or the Amazon SNS topic used for notifications.

Note

If you change the name of a repository from the console, notifications continue to work without modification. However, if you change the name of your repository from the command line or by using the API, notifications no longer work. The easiest way to restore notifications is to delete the notification settings and then configure them again.

To delete notification settings

1. Open the CodeCommit console at <https://console.aws.amazon.com/codesuite/codecommit/home>.
2. In **Repositories**, choose the name of the repository where notifications are disabled.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**.

The screenshot shows the AWS CodeCommit console interface. On the left, there's a sidebar with 'Developer Tools' and 'CodeCommit' selected. Under 'Source + CodeCommit', 'Settings' is highlighted. The main content area is titled 'MyDemoRepo'. At the top right, there are tabs for 'Settings', 'Notifications' (which is active), and 'Triggers'. Below this, the 'Notifications' section has an 'Edit' button. It shows an 'SNS topic' and a 'CloudWatch event rule', both of which have a green 'Enabled' status indicator. Under 'Event status', it says 'Enabled'. The 'Subscribers' section below says 'Choose the users or services you want to receive notifications of repository events.' At the bottom, there are fields for 'Protocol' and 'Address or endpoint'.

4. (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://console.aws.amazon.com/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 \(p. 313\)](#).

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 tips on using tags, see the [AWS Tagging Strategies](#) post on the [AWS Answers blog](#).

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 \(p. 288\)](#).

Topics

- [Add a Tag to a Repository \(p. 96\)](#)
- [View Tags for a Repository \(p. 98\)](#)
- [Edit Tags for a Repository \(p. 99\)](#)
- [Remove a Tag from a Repository \(p. 100\)](#)

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 \(p. 313\)](#). After you have tags, you can create IAM policies to manage access to the repository based on these tags. You can use the 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 \(p. 288\)](#).

For more information about adding tags to a repository when you create it, see [Create a Repository \(Console\) \(p. 82\)](#).

Topics

- [Add a Tag to a Repository \(Console\) \(p. 97\)](#)
- [Add a Tag to a Repository \(AWS CLI\) \(p. 97\)](#)

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**.

Key	Value - optional
Team	Saanvi

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 \(AWS CLI\) \(p. 83\)](#).

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 tips on using tags, see the [AWS Tagging Strategies](#) post on the AWS Answers blog. For examples of tag-based access policies, see [Example 5: Deny or Allow Actions on Repositories with Tags \(p. 288\)](#).

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**.

The screenshot shows the AWS CodeCommit console interface. On the left, there is a sidebar titled 'Developer Tools' with 'CodeCommit' selected. Under 'Source + CodeCommit', the 'Tags' option is highlighted. The main content area shows the 'MyDemoRepo' repository details. At the top right, there are tabs for 'Settings', 'Notifications', 'Triggers', and 'Repository tags', with 'Repository tags' being the active tab. Below this, there is a table titled 'Repository Tags' with an 'Info' link. The table has two columns: 'Key' and 'Value'. There are two entries: 'Status' with value 'Secret' and 'Team' with value 'Saanvi'. A scroll bar is visible on the right side of the table.

Key	Value
Status	Secret
Team	Saanvi

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 \(p. 313\)](#).

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 \(p. 288\)](#).

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.

Developer Tools > CodeCommit > Repositories > MyDemoRepo > Settings > Repository tags

Edit Repository Tags

Repository Tags Info

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources. Each resource can have up to 50 tags. Keys cannot begin with "AWS:".

Key	Value - optional	Remove
Status	Secret	Remove
Team	Saanvi	Remove

Add tag

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 \(p. 288\)](#).

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 \(p. 102\)](#)
- [Example: Create an AWS CodeCommit Trigger for an Amazon SNS Topic \(p. 102\)](#)
- [Example: Create an AWS CodeCommit Trigger for an AWS Lambda Function \(p. 107\)](#)
- [Example: Create a Trigger in AWS CodeCommit for an Existing AWS Lambda Function \(p. 110\)](#)
- [Edit Triggers for a AWS CodeCommit Repository \(p. 115\)](#)
- [Test Triggers for an AWS CodeCommit Repository \(p. 116\)](#)
- [Delete Triggers from an AWS CodeCommit Repository \(p. 118\)](#)

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](#).
- If you want to configure your trigger to use an Amazon SNS topic in another AWS account, you must first configure that topic with a policy that allows CodeCommit to publish to that topic. For more information, see [Example 1: Create a Policy That Enables Cross-Account Access to an Amazon SNS Topic \(p. 289\)](#).
- 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 \(p. 110\)](#) and [Example 3: Create a Policy for AWS Lambda Integration with a CodeCommit Trigger \(p. 291\)](#).

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](#).

Topics

- [Create a Trigger to an Amazon SNS Topic for a CodeCommit Repository \(Console\) \(p. 103\)](#)
- [Create a Trigger to an Amazon SNS Topic for a CodeCommit Repository \(AWS CLI\) \(p. 103\)](#)

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.
 - 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.
- 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, *master* and *preprod*:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": [
        {
            "name": "MyFirstTrigger",
            "destinationArn": "arn:aws:sns:us-east-2:80398EXAMPLE:MySNSTopic",
            "customData": "",
            "branches": [
                "master", "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-2:80398EXAMPLE:MySNSTopic",
            "customData": "",
            "branches": [
                "master", "preprod"
            ],
            "events": [
                "all"
            ]
        },
        {
            "name": "MySecondTrigger",
            "destinationArn": "arn:aws:sns:us-east-2:80398EXAMPLE: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-2:80398EXAMPLE:MySNSTopic",
        "branches": [
            "master",
            "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-2:80398EXAMPLE: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::80398EXAMPLE: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

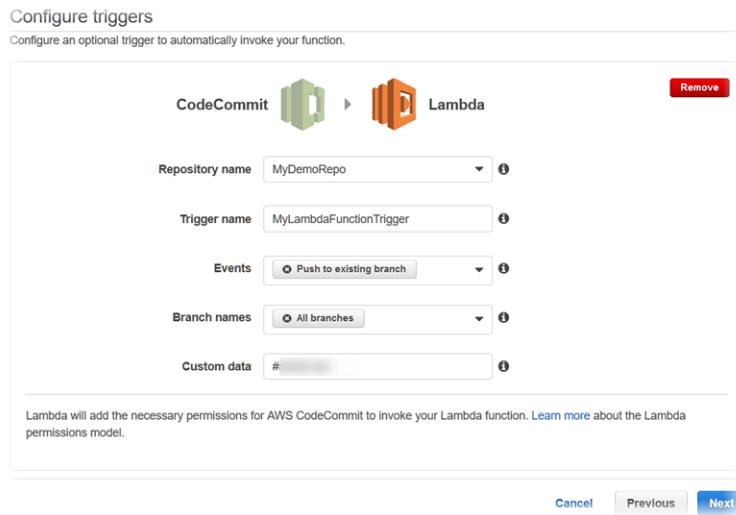
- [Create the Lambda Function \(p. 107\)](#)
- [View the Trigger for the Lambda Function in the AWS CodeCommit Repository \(p. 109\)](#)

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://console.aws.amazon.com/lambda/>.
2. On the **Lambda Functions** page, choose **Create a Lambda function**. (If you have not used Lambda before, choose **Get Started Now**.)
3. On the **Select blueprint** page, choose **Blank function**.
4. On the **Configure triggers** page, choose AWS 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 **Next**.

5. On the **Configure function** page, in **Name**, enter a name for the function (for example, *MyCodeCommitFunction*). In **Description**, enter an optional description for the function. If you want to create a sample JavaScript function, in **Runtime**, choose **Node.js**. If you want to create a sample Python function, choose **Python 2.7**.
6. 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:

```
var aws = require('aws-sdk');
var codecommit = new aws.CodeCommit({ apiVersion: '2015-04-13' });

exports.handler = function(event, context) {

    //Log the updated references from the event
    var references = event.Records[0].codecommit.references.map(function(reference)
{return reference.ref;});
    console.log('References:', references);

    //Get the repository from the event and show its git clone URL
    var repository = event.Records[0].eventSourceARN.split(":")[5];
    var params = {
        repositoryName: repository
    };
    codecommitgetRepository(params, function(err, data) {
        if (err) {
            console.log(err);
            var message = "Error getting repository metadata for repository " +
repository;
            console.log(message);
            context.fail(message);
        } else {
            console.log('Clone URL:', data.repositoryMetadata.cloneUrlHttp);
            context.succeed(data.repositoryMetadata.cloneUrlHttp);
        }
    });
};
```

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
```

7. In **Lambda function handler and role**, do the following:

- In **Handler**, leave the default value as derived from the function (`index.handler` for the Node.js sample or `lambda_function.lambda_handler` for the Python sample).
- In **Role**, choose **Create a custom role**. In the IAM console, do the following:
 - In **IAM Role**, choose `lambda_basic_execution`.
 - In **Policy Name**, choose **Create a new role policy**.
 - Choose **Allow** to create the role and then return to the Lambda console. A value of `lambda_basic_execution` should now be displayed for **Role**.

Note

If you choose a different role or a different name for the role, be sure to use it in the steps in this topic.

Choose **Next**.

8. On the **Review** page, review the settings for the function, and then choose **Create function**.

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/master' ]
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 \(p. 107\)](#).

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 \(p. 110\)](#)
- [Create a Trigger for the Lambda Function in a CodeCommit Repository \(Console\) \(p. 112\)](#)
- [Create a Trigger to a Lambda Function for a CodeCommit Repository \(AWS CLI\) \(p. 112\)](#)

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-1:80398EXAMPLE:MyDemoRepo",  
    "SourceAccount": "80398EXAMPLE"  
}
```

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\":\"80398EXAMPLE\",\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo\"},\"Action\":[\"lambda:InvokeFunction\"],\"Resource\":\"arn:aws:lambda:us-east-1:80398EXAMPLE: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://console.aws.amazon.com/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-1:80398EXAMPLE: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:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Stmt1111111111",  
            "Effect": "Allow",  
            "Action": [  
                "codecommit:GetRepository"  
            ],  
            "Resource": [  
                "arn:aws:codecommit:us-east-1:80398EXAMPLE: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 \(p. 110\)](#).

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, *master* and *preprod*:

```
{  
    "repositoryName": "MyDemoRepo",  
    "triggers": [  
        {  
            "name": "MyLambdaFunctionTrigger",  
            "destinationArn": "arn:aws:lambda:us-  
east-1:80398EXAMPLE:function:MyCodeCommitFunction",  
            "customData": "",  
            "branches": [  
                "master", "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-1:80398EXAMPLE:function:MyCodeCommitFunction",  
            "customData": "",  
            "branches": [  
                "master", "preprod"  
            ],  
            "events": [  
                "all"  
            ]  
        },  
        {  
            "name": "MyOtherLambdaFunctionTrigger",  
            "destinationArn": "arn:aws:lambda:us-  
east-1:80398EXAMPLE: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-1:80398EXAMPLE:MyCodeCommitFunction",  
            "branches": [  
                "master",  
                "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 a 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 \(Console\) \(p. 115\)](#)
- [Edit a Trigger for a Repository \(AWS CLI\) \(p. 115\)](#)

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 master 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-2:80398EXAMPLE:MyCodeCommitTopic",  
            "branches": [  
                "master",  
                "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 \(Console\) \(p. 117\)](#)
- [Test a Trigger for a Repository \(AWS CLI\) \(p. 117\)](#)

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 *TestTrigger.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-2:80398EXAMPLE:MyCodeCommitTopic",
      "branches": [
        "master",
        "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 \(Console\) \(p. 118\)](#)
- [Delete a Trigger from a Repository \(AWS CLI\) \(p. 118\)](#)

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-2:80398EXAMPLE:MyCodeCommitTopic",
      "branches": [
        "master",
        "develop"
      ]
    }
  ]
}
```

```
        "preprod"
    ],
    "name": "MyFirstTrigger",
    "customData": "",
    "events": [
        "all"
    ]
},
{
    "destinationArn": "arn:aws:lambda:us-
east-2:80398EXAMPLE:function:MyCodeCommitJSFunction",
    "branches": [],
    "name": "MyLambdaTrigger",
    "events": [
        "all"
    ]
}
}
```

- 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": []
}
```

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 \(p. 6\)](#).

Topics

- [View Repository Details \(Console\) \(p. 120\)](#)
- [View CodeCommit Repository Details \(Git\) \(p. 120\)](#)
- [View CodeCommit Repository Details \(AWS CLI\) \(p. 121\)](#)

View Repository Details (Console)

Use the AWS CodeCommit console to quickly view all repositories created with your AWS account.

1. Open the CodeCommit console at <https://console.aws.amazon.com/codesuite/codecommit/home>.
2. In **Repositories**, choose the name of the repository.
3. 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 details for the repository, 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 \(p. 229\)](#).

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 \(p. 84\)](#).

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
    master 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)
    master pushes to master (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
    master 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)
  master pushes to master (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 \(p. 325\)](#).

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 \(p. 121\)](#).
- To view information about a single CodeCommit repository, run [get-repository \(p. 122\)](#).
- To view information about multiple repositories in CodeCommit, run [batch-get-repositories \(p. 122\)](#).

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 AWS account.

Here is some example output based on the preceding command:

```
{
  "repositories": [
    {
      "repositoryName": "MyDemoRepo",
      "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
    },
    {
      "repositoryName": "MyOtherDemoRepo",
      "repositoryId": "fcfc29ac4-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 \(p. 121\)](#) 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 AWS account associated with the repository (`accountId`).

Here is some example output, based on the preceding example command:

```
{  
    "repositoryMetadata": {  
        "creationDate": 1429203623.625,  
        "defaultBranch": "master",  
        "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 \(p. 121\)](#) 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 AWS account associated with the repository (`accountId`).

Here is some example output, based on the preceding example command:

```
{  
    "repositoriesNotFound": [],  
    "repositories": [  
        {  
            "creationDate": 1429203623.625,  
            "defaultBranch": "master",  
            "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": "master",  
            "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 \(Console\) \(p. 124\)](#)
- [Change AWS CodeCommit Repository Settings \(AWS CLI\) \(p. 125\)](#)

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 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 \(p. 325\)](#).

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 \(p. 125\)](#) to change the description of an CodeCommit repository.
- [update-repository-name \(p. 125\)](#) 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 \(p. 121\)](#) 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 \(p. 121\)](#) 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 **master** branch in the CodeCommit repository with the nickname **origin**:

```
git push origin master
```

Similarly, to pull changes to the local repo from the **master** branch in the CodeCommit repository with the nickname **origin**:

```
git pull origin master
```

Tip

If you add the **-u** option to **git push**, you set upstream tracking information. For example, if you run **git push -u origin master**, 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 \(p. 230\)](#).

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 master** 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  master -> master  
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  master -> master
```

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  master -> master  
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  master -> master
```

For more options, see your Git documentation.

Configure Cross-Account Access to an AWS CodeCommit Repository

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 installing and configuring the AWS CLI credential helper on a local computer or Amazon EC2 instance, as described in [For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper \(p. 37\)](#) or [For HTTPS Connections on Windows with the AWS CLI Credential Helper \(p. 41\)](#).

For more information, see [IAM users](#).

Topics

- [Cross-Account Repository Access: Actions for the Administrator in AccountA \(p. 130\)](#)
- [Cross-Account Repository Access: Actions for the Administrator in AccountB \(p. 133\)](#)
- [Cross-Account Repository Access: Actions for the Repository User in AccountB \(p. 134\)](#)

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 \(p. 131\)](#)
- [Step 2: Create a Role for Repository Access in AccountA \(p. 132\)](#)

Step 1: Create a Policy for Repository Access in AccountA

You can create a policy in AccountA that grants access to the repository 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 role for repository access

1. Sign in to the AWS Management Console as an IAM user with permissions to create roles in AccountA.
2. Open the IAM console at <https://console.aws.amazon.com/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:

```
{  
    "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: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:

```
{
```

```
"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:codecommit:us-east-2:111122223333:MySharedDemoRepo"
        ]
    },
    {
        "Effect": "Allow",
        "Action": "codecommit>ListRepositories",
        "Resource": "*"
    }
]
```

This access makes it easier for users who assume this role 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 AWS account ID and repository name with your AWS 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 policy for repository access

1. In the IAM console, choose **Roles**.
2. Choose **Create role**.
3. Choose **Another AWS account**.
4. In **Account ID**, enter the AWS 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 \(p. 133\)](#)
- [Step 2: Create a Policy and Add Users to the IAM Group \(p. 133\)](#)

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://console.aws.amazon.com/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 \(p. 131\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Resource":  
                "arn:aws: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 \(p. 134\)](#)
- [Step 2: Clone and Access the CodeCommit Repository in AccountA \(p. 137\)](#)

Step 1: Configure the AWS CLI and Git for an AccountB User to Access the Repository in AccountA

You must use the credential helper, not SSH keys or Git credentials, to access repositories in another AWS account. AccountB users must configure their computers to use the credential helper to access the shared CodeCommit repository in AccountA. You cannot use SSH keys or Git credentials to access repositories in another AWS account. However, you can continue to use SSH keys or Git credentials when accessing repositories in AccountB.

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](#).

When you install Git, do not install the Git Credential Manager. The Git Credential Manager is not compatible with the credential helper included with the AWS CLI.

Note

CodeCommit supports Git versions 1.7.9 and later. 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 \(p. 253\)](#).

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 AccountA 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 `MyCrossAccountAccessProfile` profile you just created. It should look similar to the following:

```
[profile MyCrossAccountAccessProfile]
region = US East (Ohio)
output = json
```

Add two lines to the profile configuration, `role_arn` and `source_profile`. Provide 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 (Ohio)
role_arn = arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole
source_profile = default
output = json
```

Save your changes, and close the plain-text editor.

8. Run the `git config` command twice: once to configure Git to use the AWS CLI credential helper with the AWS CLI profile you just created, and again to use HTTP. For example:

```
git config --global credential.helper '!aws --profile MyCrossAccountAccessProfile
codecommit credential-helper $@'
```

```
git config --global credential.UseHttpPath true
```

Note

The syntax for the first `git config` command will vary slightly depending on your operating system. Linux, macOS, or Unix users should use single quotes (') where Windows users should use double quotes (").

9. Run the `git config -l` command to verify your configuration. A successful configuration contains lines similar to the following:

```
user.name=Saanvi Sarkar
user.email=saanvi_sarkar@example.com
credential.helper=!aws --profile MyCrossAccountAccessProfile codecommit credential-
helper $@
credential.usehttppath=true
```

Note

If you see the following line in your configuration file, another credential manager is configured for your system, which can prevent the AWS CLI credential helper from working:

```
credential.helper=manager
```

To fix this problem, run the following command:

```
git config --system --unset credential.helper
```

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 that 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

- At the command line or terminal, in the directory where you want to clone the repository, run the **git clone** command with the HTTPS clone URL. For example:

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

Unless you specify otherwise, the repository is cloned into a subdirectory with the same name as the repository.

- 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*.
- 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 [Git with AWS CodeCommit Tutorial \(p. 64\)](#).

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

- 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.
- 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.

- 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 Tutorial \(p. 47\)](#).

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 \(Console\) \(p. 138\)](#)
- [Delete a Local Repo \(p. 139\)](#)
- [Delete a CodeCommit Repository \(AWS CLI\) \(p. 139\)](#)

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 **Repository settings** page, 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 \(p. 325\)](#).

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 \(p. 121\)](#) 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.

The screenshot shows the AWS CodeCommit 'Create a file' interface. At the top, a navigation bar shows 'Developer Tools > CodeCommit > Repositories > MyDemoRepo > File'. A 'master' branch indicator is in the top right. Below the navigation, the title 'Create a file' is displayed. The main area shows a code editor with a Python script named 'samples/hello.py'. The script content is:

```
1 import sys
2
3 print('Hello, World!')
4
5 print('The sum of 2 and 3 is 5.')
6
7 sum = int(sys.argv[1]) + int(sys.argv[2])
8
9 print('The sum of {0} and {1} is {2}.'.format(sys.argv[1], sys.argv[2], sum))
```

Below the code editor, a 'Commit changes to master' section is shown. It includes fields for 'File name' (set to 'samples/hello.py'), 'Author name' (set to 'Saanvi Sarkar'), and a note indicating the file path 'MyDemoRepo/samples/hello.py'.

For information about working with other aspects of your repository in CodeCommit, see [Working with Repositories \(p. 81\)](#), [Working with Pull Requests \(p. 149\)](#), [Working with Branches \(p. 215\)](#), [Working with Commits \(p. 184\)](#), and [Working with User Preferences \(p. 229\)](#).

Topics

- [Browse Files in an AWS CodeCommit Repository \(p. 141\)](#)
- [Create or Add a File to an AWS CodeCommit Repository \(p. 142\)](#)
- [Edit the Contents of a File in an AWS CodeCommit Repository \(p. 145\)](#)

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](#).

The screenshot shows the AWS CodeCommit console interface. On the left, a sidebar titled "Developer Tools" has "CodeCommit" selected. Under "Source", "CodeCommit" is also selected. The main content area shows the "MyDemoRepo" repository. At the top, the path is "Developer Tools > CodeCommit > Repositories > MyDemoRepo". Below the path, there's a dropdown for "master" and a "Create pull request" button. The main content area displays the file "MyDemoRepo / cl_sample.js" with the following code:

```
1 var aws = require('aws-sdk');
2 var codecommit = new aws.CodeCommit({ apiVersion: '2015-04-13' });
3
4 exports.handler = function(event, context) {
5     //Log the updated references from the event
6     var references = event.Records[0].codecommit.references.map(function(reference) {
7         console.log('References:', reference);
8
9     });
10    //Get the repository from the event and show its git clone URL
11    var repository = event.Records[0].eventSourceARN.split(":")[5];
12    var params = {
13        repositoryName: repository
14    };
15    codecommitgetRepository(params, function(err, data) {
16        if (err) {
17            console.log(err);
18            var message = "Error getting repository metadata for repository " + repository;
19            console.log(message);
20            context.fail(message);
21        } else {
22            console.log('Clone URL', data.repositoryMetadata.cloneUrlHttp);
23            context.succeed(data.repositoryMetadata.cloneUrlHttp);
24        }
25    });
26}
```

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 [Limits \(p. 312\)](#). To exit the file view, from the code navigation bar, choose the directory you want to view.

Note

If you choose a binary file, 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.

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.

Upload a file

Choose the branch where you will upload the file, and then choose the file to upload. Add your user information and a comment about why you added this file.

MyDemoRepo

Name	Size	Actions
		Upload file Choose a file to upload. <input type="button" value="Choose file"/>

Commit changes to master

Author name

Email address

Commit message - optional
A default commit message will be used if you do not provide one.

Topics

- [Create or Upload a File \(Console\) \(p. 143\)](#)
- [Add a File \(AWS CLI\) \(p. 144\)](#)
- [Add a File \(Git\) \(p. 145\)](#)

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 \(p. 325\)](#).

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 "Maria Garcia" --email "maria_garcia@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 [Git with AWS CodeCommit Tutorial \(p. 64\)](#).

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.

Developer Tools > CodeCommit > Repositories > MyDemoRepo > File

Edit a file

MyDemoRepo / cl_sample.js [Info](#)

```
1 var aws = require('aws-sdk');
2 var codecommit = new aws.CodeCommit({ apiVersion: '2015-04-13' });
3
4 exports.handler = function(event, context) {
5     //Log the updated references from the event
6     var references = event.Records[0].codecommit.references.map(function(reference) {return reference.ref;});
7     console.log('References:', references);
8
9     //Get the repository from the event and show its git clone URL
10    var repository = event.Records[0].eventSourceARN.split(":")[5];
11    var params = {
12        repositoryName: repository
13    };
14    codecommitgetRepository(params, function(err, data) {
15        if (err) {
16            console.log(err);
17            var message = "Error getting repository metadata for repository " + repository;
18            console.log(message);
19            context.fail(message);
20        } else {
21            console.log('Clone URL', data.repositoryMetadata.cloneUrlHttp);
22            context.succeed(data.repositoryMetadata.cloneUrlHttp);
23        }
24    });
25};
```

Commit changes to AnotherBranch

File: MyDemoRepo/cl_sample.js

Author name

Topics

- [Edit a File \(Console\) \(p. 146\)](#)
- [Edit or Delete a File \(AWS CLI\) \(p. 147\)](#)
- [Edit a File \(Git\) \(p. 148\)](#)

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 \(p. 325\)](#).

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 "Maria Garcia" --email "maria_garcia@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 `master` with a most recent commit ID of `c5709475EXAMPLE` in a repository named `MyDemoRepo`:

```
aws codecommit delete-file --repository-name MyDemoRepo --branch-name master --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 [Git with AWS CodeCommit Tutorial \(p. 64\)](#).

Working with Pull Requests in AWS CodeCommit Repositories

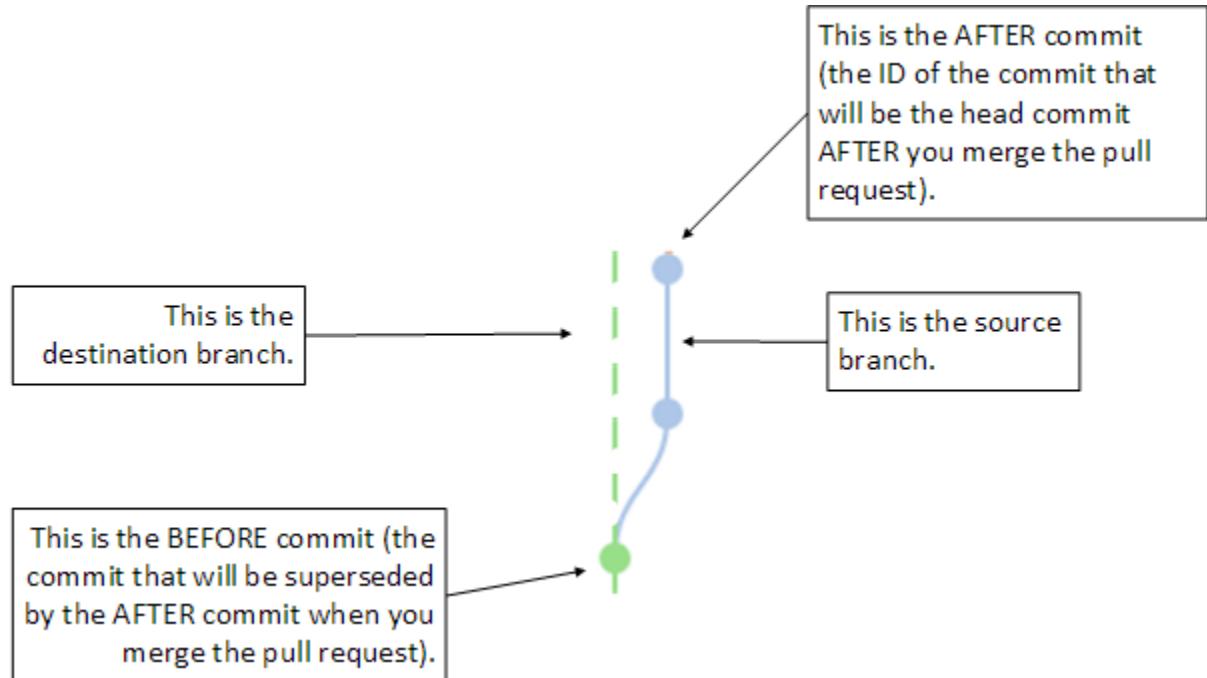
A pull request is the primary way you and other repository users can review, comment on, and merge code changes from one branch to another. You can use pull requests to collaboratively review code changes for minor changes or fixes, major feature additions, or new versions of your released software. 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). 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, she or one of her reviewers merges her code and closes the pull request.

The screenshot shows the AWS CodeCommit 'Create pull request' interface. On the left, there's a sidebar with 'Developer Tools' and 'CodeCommit'. Under 'Source + CodeCommit', 'Pull requests' is selected. The main area shows a 'Create pull request' dialog. In the 'Destination' dropdown, 'master' is selected. In the 'Source' dropdown, 'feature-randomizationfeature' is selected. A message box says 'Not mergeable' with the note: 'There are conflicts between master and feature-randomizationfeature that cannot be merged in the AWS CodeCommit console. You can create this pull request, but you must merge the branch manually.' Below this, the 'Details' section has a 'Title' field containing 'My Pull Request' and a 'Description' field with the text: 'This pull request contains the code for the randomization feature as well as a draft tutorial for users. I'd like to get this merged before the end of the sprint. Review and comment in the next 72 hours.' At the bottom, there are tabs for 'Changes' (which is selected) and 'Commits'. The 'Changes' tab shows a file named 'blobtest.txt' with a status of 'Deleted'.

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 "before" 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 \(Console\) \(p. 168\)](#).



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, you can close the pull request in one of several ways:

- Merge the branches locally and push your changes. This closes the request automatically.
- Use the AWS CodeCommit console to either close the pull request without merging, resolve conflicts in a merge, or, if there are no conflicts, to 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.)

Pull requests are more effective when you've set up IAM users for your repository users in your AWS account. It's easier to identify which user made which comment. IAM users also have the advantage of being able to use Git credentials for repository access. For more information, see [Step 1: Initial Configuration for CodeCommit \(p. 9\)](#). 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 \(p. 81\)](#), [Working with Files \(p. 140\)](#), [Working with Commits \(p. 184\)](#), [Working with Branches \(p. 215\)](#), and [Working with User Preferences \(p. 229\)](#).

Topics

- [Create a Pull Request \(p. 151\)](#)
- [View Pull Requests in an AWS CodeCommit Repository \(p. 154\)](#)
- [Review a Pull Request \(p. 157\)](#)
- [Update a Pull Request \(p. 164\)](#)
- [Merge a Pull Request in an AWS CodeCommit Repository \(p. 167\)](#)
- [Resolve Conflicts in a Pull Request in an AWS CodeCommit Repository \(p. 173\)](#)
- [Close a Pull Request in an AWS CodeCommit Repository \(p. 181\)](#)

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 \(Console\) \(p. 151\)](#)
- [Create a Pull Request \(AWS CLI\) \(p. 153\)](#)

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 \(p. 90\)](#), 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**.

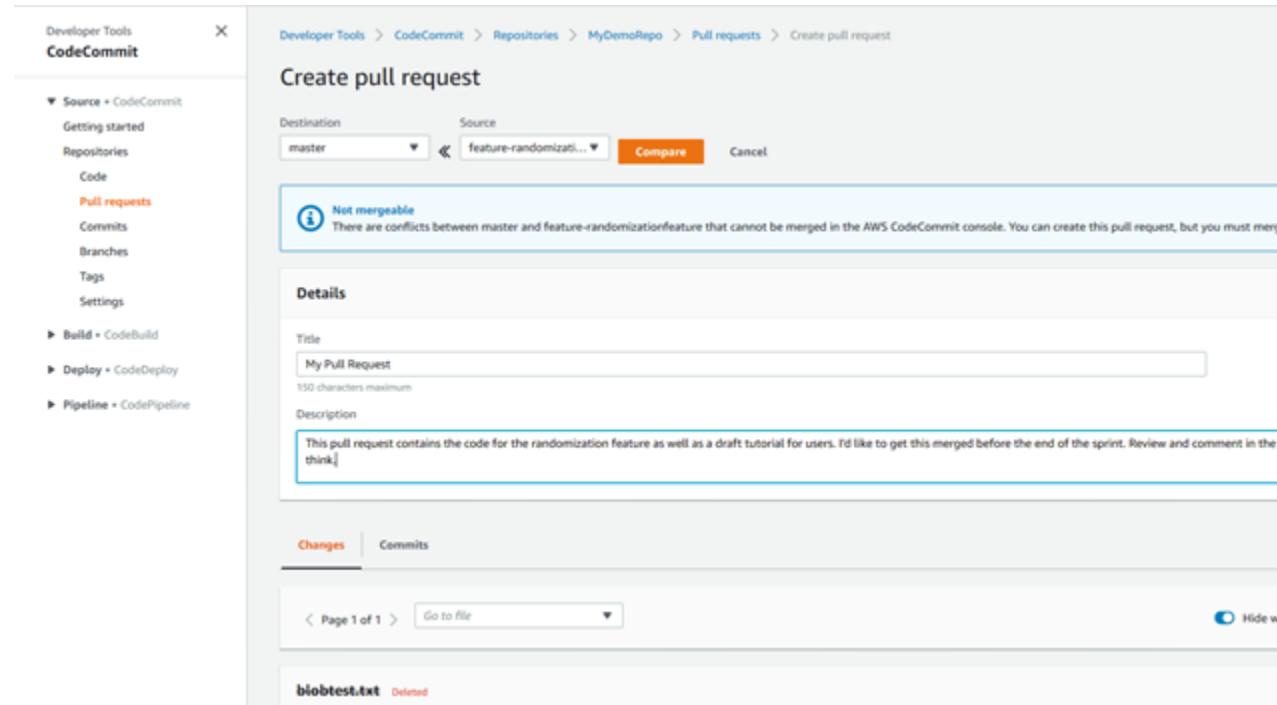
Tip

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

4. Choose **Create pull request**.

The screenshot shows the AWS CodeCommit console interface. On the left, there is a sidebar titled 'Developer Tools' with a 'CodeCommit' section. Under 'Source', 'Pull requests' is highlighted. Other options include 'Commits', 'Branches', 'Tags', and 'Settings'. Below 'Source' are sections for 'Build' (CodeBuild), 'Deploy' (CodeDeploy), and 'Pipeline' (CodePipeline). The main content area is titled 'MyDemoRepo' and shows a list of 'Pull requests'. The list includes one item: '6: My Pull Request' (Author: [redacted], Destination: master, Last activity: Just now). A button labeled 'All pull requests' is visible on the right side of the list.

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 \(p. 90\)](#), 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 \(p. 325\)](#).

To use the AWS CLI to create a pull request in a CodeCommit repository

1. 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 **sourceBranch** (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).

For example, to create a pull request named **My Pull Request** with a description of **Please review these changes by Tuesday** that targets the **MyNewBranch** source branch and is to be merged to the default branch **master** in a CodeCommit repository named **MyDemoRepo**:

```
aws codecommit create-pull-request --title "My Pull Request" --description "Please review these changes by Tuesday" --client-request-token 123Example --targets repositoryName=MyDemoRepo,sourceReference=MyNewBranch
```

2. If successful, this command produces output similar to the following:

```
{  
    "pullRequest": {  
        "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",  
        "clientRequestToken": "123Example",  
        "creationDate": 1508962823.285,  
        "description": "Please review these changes by Tuesday",  
        "lastActivityDate": 1508962823.285,  
        "pullRequestId": "42",  
        "pullRequestStatus": "OPEN",  
        "pullRequestTargets": [  
            {  
                "destinationCommit": "5d036259EXAMPLE",  
                "destinationReference": "refs/heads/master",  
                "mergeMetadata": {  
                    "isMerged": false,  
                },  
                "repositoryName": "MyDemoRepo",  
                "sourceCommit": "317f8570EXAMPLE",  
                "sourceReference": "refs/heads/MyNewBranch"  
            }  
        ],  
        "title": "My Pull Request"  
    }  
}
```

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 \(Console\) \(p. 154\)](#)
- [View Pull Requests \(AWS CLI\) \(p. 155\)](#)

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.

The screenshot shows the AWS CodeCommit console interface. On the left, there's a sidebar with navigation links for Source (Getting started, Repositories, Code, Pull requests), Build (CodeBuild), Deploy (CodeDeploy), and Pipeline (CodePipeline). The 'Pull requests' link under 'Source' is highlighted. The main area is titled 'MyDemoRepo' and shows a table of pull requests. The table has columns for 'Pull request' (listing numbers 6, 4, 1, 5, 3, 2), 'Author' (all blurred), 'Destination' (master), and 'Last activity' (Just now). A button labeled 'All pull requests' is visible at the top right of the table area.

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 \(p. 325\)](#).

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 `42`:

```
aws codecommit get-pull-request --pull-request-id 42
```

If successful, this command produces output similar to the following:

```
{  
    "pullRequest": {  
        "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",  
        "title": "Pronunciation difficulty analyzer"  
        "pullRequestTargets": [  
            {  
                "destinationReference": "refs/heads/master",  
                "destinationCommit": "5d036259EXAMPLE",  
                "sourceReference": "refs/heads/jane-branch"  
                "sourceCommit": "317f8570EXAMPLE",  
                "repositoryName": "MyDemoRepo",  
                "mergeMetadata": {  
                    "isMerged": false,  
                },  
            },  
        ],  
        "lastActivityDate": 1508442444,  
        "pullRequestId": "42",  
        "clientRequestToken": "123Example",  
        "pullRequestStatus": "OPEN",  
        "creationDate": 1508962823,  
        "description": "A code review of the new feature I just added to the service."  
    }  
}
```

3. 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"  
        }  
    ]  
}
```

4. 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 **master** in a repository named **MyDemoRepo**:

```
aws codecommit get-merge-conflicts --repository-name MyDemoRepo --source-commit-specifier my-feature-branch --destination-commit-specifier master --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 \(p. 90\)](#), 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 **git diff** command or a diff tool.

Topics

- [Review a Pull Request \(Console\) \(p. 158\)](#)
- [Review Pull Requests \(AWS CLI\) \(p. 161\)](#)

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. You can also comment on a closed or merged pull request.

The screenshot shows the AWS CodeCommit console interface. The left sidebar has a 'Developer Tools' header and a 'CodeCommit' section. Under 'Source', 'Pull requests' is highlighted in orange. Other options include 'Commits', 'Branches', 'Tags', and 'Settings'. Under 'Build', there's a 'CodeBuild' option. Under 'Deploy', there's a 'CodeDeploy' option. Under 'Pipeline', there's a 'CodePipeline' option. The main content area is titled 'MyDemoRepo' and shows a 'Pull requests' table. The table has columns for 'Pull request', 'Author', 'Destination', and 'Last activity'. There are six pull requests listed, each with a blue blurred preview. A button labeled 'All pull requests' is visible in the top right corner of the table area.

5. In the pull request, choose **Changes**.
6. Do one of the following:
 - To add a general comment, 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.

Comments on changes

New comment

P

Did we also change the variable name in blf.py and concat.py?

Save

- To add a comment to a file in the commit, in **Changes**, find the name of the file. Choose the comment bubble that appears next to the file name , enter a comment, and then choose **Save**.

ahs_count.py

Browse file contents

New comment

P

|

Cancel

- 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 bubble , enter a comment, and then choose **Save**.

The screenshot shows the AWS CodeCommit interface for reviewing a pull request. On the left, a sidebar menu includes 'Source + CodeCommit' (Getting started, Repositories, Code, Pull requests, Commits, Branches, Tags, Settings), 'Build + CodeBuild', 'Deploy + CodeDeploy', and 'Pipeline + CodePipeline'. The main area is titled 'Mergeable' with a 'Learn more' link. It features tabs for 'Details', 'Activity', 'Changes' (which is selected), and 'Commits'. Below these tabs is a navigation bar with '< Page 1 of 1 >' and a 'Go to file' dropdown. A 'Hide' button is also present. The central part of the screen displays a diff view for a file named 'ahs_count.py'. The diff shows code changes between two versions of the file. A red highlight covers line 8 of the old version, which contains '- print(ahs.format(total))'. A green highlight covers line 8 of the new version, which contains '+ print(alv.format(total))'. The code itself includes lines like 'total = (ess + z)' and 'ahs = "Number of alveolar hissing sibilants: {}"'. To the right of the diff, there is a 'Browse' button. Below the diff, a 'New comment' input field contains the placeholder 'You've switched back to the old varia...'. At the bottom right are 'Save' and 'Cancel' buttons.

7. To reply to comments on a commit, in **Changes** or **Activity**, choose **Reply**.

The screenshot shows three separate comment sections within a pull request interface. Each section has a header, a comment body, a timestamp, and two buttons: 'Reply' and 'Edit'. The first section is titled 'Comment on line 12 of second-randomizer.py' and contains a code snippet. The second section is titled 'Comment on [REDACTED]'. The third section is titled 'Comment on changes'.

Comment on line 12 of second-randomizer.py

```
reroll = raw_input("Reroll?")
```

commented 4 minutes ago

Are we sure this is the approved string?

Reply **Edit**

Comment on [REDACTED]

commented 5 minutes ago

I'm not sure why you removed this test.

Reply **Edit**

Comment on changes

commented 5 minutes ago

I think our users will find this a fun feature, but I'm not convinced it's fully formed. Did you double-check against the user stories and

Reply **Edit**

If notifications (p. 90) are configured, the user who created the pull request receives email about your comments. You receive email if a user replies to your comments or if the pull request is updated.

Review Pull Requests (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

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, 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,
    "CommentId": "",
    "lastModifiedDate": 1508369622.123
  },
  "location": {
    "filePath": "ahs_count.py",
    "filePosition": 367,
    "relativeFileVersion": "AFTER"
  },
  "repositoryName": "MyDemoRepo",
  "pullRequestId": "47"
}
```

- 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 full commit ID of the commit in the source branch that was the tip of the branch at the time the comment was made (with the **--after-commit-id** option).
 - The full commit ID of the commit in the destination branch that was the tip of the branch at the time the pull request was created (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 for a pull request in a repository named `MyDemoRepo`:

```
aws codecommit get-comments-for-pull-request --repository-name MyDemoRepo --before-commit-ID 317f8570EXAMPLE --after-commit-id 5d036259EXAMPLE
```

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
        },
        {
          "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
          "clientRequestToken": "",
          "commentId": "442b498bEXAMPLE5756813",
          "content": "Good catch. I'll remove them.",
          "creationDate": 1508369829.104,
          "deleted": false,
          "commentId": "abcd1234EXAMPLEb5678efgh",
          "lastModifiedDate": 150836912.273
        }
      ],
      "location": {
        "filePath": "ahs_count.py",
        "filePosition": 367,
        "relativeFileVersion": "AFTER"
      },
      "repositoryName": "MyDemoRepo",
      "pullRequestId": "42"
    }
  ],
  "nextToken": "exampleToken"
}
```

3. 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
```

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  
    }  
}
```

Update a Pull Request

You can use your local Git client to push commits to the source branch, which updates the pull request with code changes. You might update the pull request with more commits because:

- You want users to review code changes you made to the source branch code in response to comments in the pull request.
- One or more commits have been made to the destination branch since the pull request was created. You want to incorporate those changes into the source branch (forward integration). This changes the state of the pull request to **Mergeable** and enables the merging and closing of the pull request from the console.

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 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.

Topics

- [Update a Pull Request \(Git\) \(p. 164\)](#)
- [Update a Pull Request \(Console\) \(p. 165\)](#)
- [Update Pull Requests \(AWS CLI\) \(p. 166\)](#)

Update a Pull Request (Git)

You can use Git to update the source branch of a pull request with changes to the code to:

- Add more code to the review.

- Incorporate changes suggested in review comments.
- Forward-integrate changes made in the destination branch into the source branch.
- Make sure that all the changes to be merged into the destination branch have been reviewed in the pull request.

You make the changes on your local computer, and then commit and push them to the source branch. If [notifications are configured for the repository \(p. 90\)](#), users subscribed to the topic receive emails when you push changes to the source branch of an open pull request.

To update the source branch with code changes

1. From the local repo on your computer, at the terminal or command line, make sure you have pulled the latest changes to the repository, and then run the **git checkout** command, specifying the source branch of the pull request. For example, to check out a source branch of a pull request named *pullrequestbranch*:

```
git checkout pullrequestbranch
```

2. Make any changes you want reviewed. For example, if you want to change the code in the source branch in responses to user comments, edit the files with those changes. If you want to integrate changes that have been made to the destination branch into the source branch (forward integration), run the **git merge** command, specifying the destination branch, to merge those changes into the source branch.

Tip

You might want to use diff tool or merge tool software to help view and choose the changes you want integrated into a source branch.

3. After you have made your changes, run the **git add** and **git commit** commands to stage and commit them.

Tip

You can run these commands separately, or you can use the **-a** option to add changed files to a commit automatically. For example, you could run a command similar to the following:

```
git commit -am "This is an example commit message."
```

For more information, see [Basic Git Commands \(p. 327\)](#) or consult your Git documentation.

4. Run the **git push** command to push your changes to CodeCommit. Your pull request is updated with the changes you made to the source branch.

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.

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 \(p. 325\)](#).

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"
```

If successful, this command produces output similar to the following:

```
{  
  "pullRequest": {  
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",  
    "clientRequestToken": "",  
    "creationDate": 1508530823.12,  
    "description": "Review the latest changes and updates to the global variables.  
I have updated this request with some changes, including removing some unused  
variables.",  
    "lastActivityDate": 1508372657.188,  
    "pullRequestId": "47",  
    "pullRequestStatus": "OPEN",  
    "pullRequestTargets": [  
      {  
        "destinationCommit": "9f31c968EXAMPLE",  
        "destinationReference": "refs/heads/master",  
        "mergeMetadata": {  
          "isMerged": false,  
        },  
        "repositoryName": "MyDemoRepo",  
        "sourceCommit": "99132ab0EXAMPLE",  
        "sourceReference": "refs/heads/variables-branch"  
      }  
    ],  
    "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."
```

If successful, this command produces output similar to the following:

```
{  
  "pullRequest": {  
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",  
    "clientRequestToken": "",  
    "creationDate": 1508530823.155,  
    "description": "Updated the pull request to remove unused global variable.",  
    "lastActivityDate": 1508372423.204,  
    "pullRequestId": "47",  
    "pullRequestStatus": "OPEN",  
    "pullRequestTargets": [  
      {  
        "destinationCommit": "9f31c968EXAMPLE",  
        "destinationReference": "refs/heads/master",  
        "mergeMetadata": {  
          "isMerged": false,  
        },  
        "repositoryName": "MyDemoRepo",  
        "sourceCommit": "99132ab0EXAMPLE",  
        "sourceReference": "refs/heads/variables-branch"  
      }  
    ],  
    "title": "Consolidation of global variables"  
  }  
}
```

Merge a Pull Request in an AWS CodeCommit Repository

When you're satisfied that your code has been reviewed, you can merge a pull request in one of several ways:

- 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 closes the pull request automatically if the pull request is merged using the fast-forward merge strategy. The **git merge** command also 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.
- In the console, you might be able to merge your source branch to the destination branch using one of the available merge strategies, which also closes the pull request. You see an advisory message that the pull request is mergeable. When you choose **Merge**, the merge is performed using the available merge strategy that you choose. The default merge strategy is to use fast-forward if it is applicable, 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.
- 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 use the AWS CLI to attempt to merge and close the pull request using the fast-forward, squash, or 3-way merge strategy.

Topics

- [Merge a Pull Request \(Console\) \(p. 168\)](#)

- Merge a Pull Request (AWS CLI) (p. 170)

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 will no longer appear 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.

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 between the available merge strategies. Merge strategies that cannot be applied will 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 \(p. 173\)](#).

Merge pull request 9: Bug fix for unhandled exception

Merge request details

Pull request: #9 Bug fix for unhandled exception

Destination master << Source bugfix-bug1234

Merge strategy [Info](#)

Determines the way in which the current pull request will be merged into the destination branch

Fast forward merge

`git merge --ff-only`

Merges the branches and moves the destination branch pointer to the tip of the source branch. This is the default merge strategy in Git.



Squash and merge

`git merge --squash`

Combine all commits from the source branch into a single merge commit in the destination branch.



3-way merge

`git merge --no-ff`

Create a merge commit that contains all commits to the destination branch.



Commit message - *optional*

Squashed commit of the following

commit d49940ad

Author: Li Juan <li_juan@example.com>

Date: Tue May 07 2019 15:12:48 GMT-0700 (Pacific Daylight Time)

Fixing the bug reported in 1234.

Author name

Maria Garcia

Email address

maria_garcia@example.com

Delete source branch bugfix-bug1234 after merging?

[Cancel](#)

- A fast-forward merge will move 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 will be 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, as no merge commit is created, and the tip of the source branch is fast-forwarded to the tip of the destination branch.

- A squash merge will create one commit that contains the changes in the source branch and apply 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 will be 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 will create a merge commit for the merge in the destination branch, but also retain 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.
6. 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 for the commit history.
 7. If desired, deselect the option to delete the source branch as part of the merge. The default option is to delete the source branch when merging a pull request.
 8. 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 \(p. 325\)](#).

To use the AWS CLI to merge pull requests in a CodeCommit repository

1. 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": {  
    "authorArn": "arn:aws:iam::111111111111: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/master",  
        "mergeMetadata": {  
          "isMerged": true,
```

```

        "mergedBy": "arn:aws:iam::111111111111:user/Mary_Major"
    },
    "repositoryName": "MyDemoRepo",
    "sourceCommit": "99132ab0EXAMPLE",
    "sourceReference": "refs/heads/variables-branch"
}
],
"title": "Consolidation of global variables"
}
}

```

- 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 **--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).

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** using 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 --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": {
    "authorArn": "arn:aws:iam::111111111111: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/master",
        "mergeMetadata": {
          "isMerged": true,
          "mergedBy": "arn:aws:iam::111111111111:user/Jorge_Souza"
        },
        "repositoryName": "MyDemoRepo",
        "sourceCommit": "99132ab0EXAMPLE",
        "sourceReference": "refs/heads/variables-branch"
      }
    ]
  }
}

```

```

        "sourceReference": "refs/heads/variables-branch"
    }
],
"title": "Consolidation of global variables"
}
}
```

3. 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 **--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).

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** using the default options for conflict detail and conflict resolution strategy:

```
aws codecommit merge-pull-request-by-fast-forward --pull-request-id 47 --source-commit-id 99132ab0EXAMPLE --repository-name MyDemoRepo --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": {
    "authorArn": "arn:aws:iam::111111111111: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/master",
        "mergeMetadata": {
          "isMerged": true,
          "mergedBy": "arn:aws:iam::111111111111:user/Maria_Garcia"
        },
        "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 will update 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 \(Console\) \(p. 173\)](#)
- [Resolve Conflicts in a Pull Request \(AWS CLI\) \(p. 176\)](#)

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.

The screenshot shows the AWS CodeCommit console interface. On the left is a navigation sidebar with sections for Source (Getting started, Repositories, Code, Pull requests), Build (CodeBuild), Deploy (CodeDeploy), Pipeline (CodePipeline), and links for Go to resource and Feedback. The main content area is titled "10: Merge bugfix from different branches". It shows a pull request created by "Open" (the user) from branch "bugfix-bug1234" to "master". A "Resolve conflicts" button with a "Learn more" link is present. Below this, there are tabs for Details, Activity, Changes, and Commits, with "Details" being the active tab. The "Details" section contains a message: "We've got two versions of this bugfix in different branches. Let's resolve on the best fix and merge to master." A "Resolve conflicts" button is located in the top right corner of the main content area.

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.

Developer Tools > CodeCommit > Repositories > MyDemoRepo > Pull requests > 10 > Resolve conflicts

Resolve conflicts

Resolve conflicts in each of the files in the list. When you have resolved all conflicts, update the pull request and merge strategies available. [Info](#)

Destination master << Source bugfix-bug1234

Editing: helloworld.py [Reset file](#) [Delete file](#) [Use source content](#) [Use destination content](#)

helloworld.py 1

```
1 import sys
2
3 print('Hello, World!')
4
5 <<<<< HEAD:helloworld.py
6 print('The sum of 2 and 3 is 5.')
7 =====
8 print('The sum of 3 and 2 is 5.')
9 >>>>> bugfix-bug1234:helloworld.py
10
11 sum = int(sys.argv[1]) + int(sys.argv[2])
12
13 print('The sum of {0} and {1} is {2}.format(sys.argv[1], sys.argv[2]))
```

Allow the merge to proceed with Git conflict markers still present.

[Cancel](#)

- 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 will 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 will 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 \(p. 325\)](#).

No single AWS CLI command will enable 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 **master** in a repository named **MyDemoRepo**:

```
aws codecommit get-merge-options --source-commit-specifier bugfix-1234 --destination-commit-specifier master --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 master using the three-way merge strategy in a repository named MyDemoRepo:

```
aws codecommit get-merge-conflicts --source-commit-specifier feature-
randomizationfeature --destination-commit-specifier master --merge-option
THREE_WAY_MERGE --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
    "mergeable": false,
    "destinationCommitId": "86958e0aEXAMPLE",
    "sourceCommitId": "6cccd57fdEXAMPLE",
    "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"
            }
        }
    ]
}
```

```

        "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 *master* 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 master --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 `master` using the `THREE_WAY_MERGE` strategy in a repository named `MyDemoRepo`:

```
aws codecommit describe-merge-conflicts --source-commit-specifier feature-randomizationfeature --destination-commit-specifier master --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 **master** using the **ACCEPT_SOURCE** strategy in a repository named **MyDemoRepo**:

```
aws codecommit create-unreferenced-merge-commit --source-commit-specifier bugfix-1234  
--destination-commit-specifier master --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 \(Console\) \(p. 182\)](#)
- [Close a Pull Request \(AWS CLI\) \(p. 182\)](#)

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.

The screenshot shows the AWS CodeCommit console interface. On the left, there is a sidebar with 'Developer Tools' and 'CodeCommit' selected. Under 'Source > CodeCommit', the 'Pull requests' option is highlighted in orange. The main content area is titled 'MyDemoRepo' and shows a table titled 'Pull requests'. The table has columns for 'Pull request', 'Author', 'Destination', and 'Last activity'. There are six rows in the table, each representing a pull request. The first row is labeled '6: My Pull Request' and has 'master' in the 'Destination' column and 'Just now' in the 'Last activity' column. The other five rows are partially visible and have blurred content.

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 \(p. 325\)](#).

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:

```
{  
    "pullRequest": {  
        "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",  
        "clientRequestToken": "123Example",  
        "creationDate": 1508962823.165,  
        "description": "A code review of the new feature I just added to the service.",  
        "lastActivityDate": 1508442444.12,  
        "pullRequestId": "42",  
        "pullRequestStatus": "CLOSED",  
        "pullRequestTargets": [  
            {  
                "destinationCommit": "5d036259EXAMPLE",  
                "destinationReference": "refs/heads/master",  
                "mergeMetadata": {  
                    "isMerged": false,  
                },  
                "repositoryName": "MyDemoRepo",  
                "sourceCommit": "317f8570EXAMPLE",  
                "sourceReference": "refs/heads/jane-branch"  
            }  
        ],  
        "title": "Pronunciation difficulty analyzer"  
    }  
}
```

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.

The screenshot shows a code editor interface for a file named `ahs_count.py`. The code contains a diff between two versions of the file. The diff highlights a change in line 7, where the variable `alv` is being reverted from a string to a variable. A comment box is open over this line, stating: "You've reverted to the old value here, which should remain `alv`". Below the code editor are "Save" and "Cancel" buttons.

```

ahs_count.py
Browse file contents
*** @ -4,7 +4,7 @@
4 z = z.count('z')
5
6 total = (ess + z)
7 - alv = "Number of alveolar hissing sibilants: {}"
    + ahs = "Number of alveolar his
*** @ -4,7 +4,7 @@
4 z = z.count('z')
5
6 total = (ess + z)
7 + ahs = "Number of alveolar his

```

New comment Preview

You've reverted to the old value here, which should remain `alv`.

Save Cancel

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 \(p. 8\)](#).

For information about working with other aspects of your repository in CodeCommit, see [Working with Repositories \(p. 81\)](#), [Working with Files \(p. 140\)](#), [Working with Pull Requests \(p. 149\)](#), [Working with Branches \(p. 215\)](#), and [Working with User Preferences \(p. 229\)](#).

Topics

- [Create a Commit in AWS CodeCommit \(p. 185\)](#)
- [View Commit Details in AWS CodeCommit \(p. 189\)](#)
- [Compare Commits in AWS CodeCommit \(p. 196\)](#)
- [Comment on a Commit in AWS CodeCommit \(p. 201\)](#)

- [Create a Git Tag in AWS CodeCommit \(p. 210\)](#)
- [View Git Tag Details in AWS CodeCommit \(p. 211\)](#)
- [Delete a Git Tag in AWS CodeCommit \(p. 213\)](#)

Create a Commit in AWS CodeCommit

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 \(p. 142\)](#) and [Edit the Contents of a File in an AWS CodeCommit Repository \(p. 145\)](#).

Note

If using 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 a Commit Using a Git Client \(p. 185\)](#)
- [Create a Commit Using the AWS CLI \(p. 187\)](#)

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 \(p. 6\)](#).

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`.
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/master** would show output similar to the following:

```
bird.txt | 1 +
1 file changed, 1 insertion(+)
```

Of course, the output assumes you have already connected the local repo to the CodeCommit repository. (For instructions, see [Connect to a Repository \(p. 84\)](#).)

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 master** 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 master -> master
```

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 master -> master
```

Tip

If you add the `-u` option to `git push` (for example, `git push -u origin master`), 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.

Note

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

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 an initial commit for a repository that adds a `readme.md` file to a repository named `MyDemoRepo` in the `master` branch:

```
aws codecommit create-commit --repository-name MyDemoRepo --branch-name master --put-files "filePath=readme.md,fileContent='Welcome to our team repository.'"
```

If successful, this command returns output similar to the following:

```
{  
  "commitId": "4df8b524-EXAMPLE",  
  "treeId": "55b57003-EXAMPLE",  
  "author": {  
    "name": "John Doe",  
    "email": "john.doe@example.com",  
    "date": "2015-04-13T12:00:00Z"  
  },  
  "committer": {  
    "name": "John Doe",  
    "email": "john.doe@example.com",  
    "date": "2015-04-13T12:00:00Z"  
  },  
  "files": [  
    {"path": "readme.md", "content": "Welcome to our team repository."}  
  ]  
}
```

```

"filesAdded": [
    {
        "blobId": "5e1c309d-EXAMPLE",
        "absolutePath": "readme.md",
        "fileMode": "NORMAL"
    }
],
"filesDeleted": [],
"filesUpdated": []
}

```

To create a commit that makes changes to files named `file1.py` and `file2.py`, 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 --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 will vary slightly 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": "NORMAL"
        }
    ]
}

```

```
        "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 \(p. 189\)](#)
- [View a Graph of the Commit History of a Repository \(p. 190\)](#)

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.

Commit ID	Commit message	Commit date	Author
d615e7ae	Merge branch 'AnotherBranch' into testbranch	5 days ago	Maria Garcia
b6589863	Added another file.	5 days ago	Li Juan
73a6e39c	Merge branch 'master' into testbranch	5 days ago	Maria Garcia
6bbb6d3c	Another test of the editing feature.	5 days ago	Li Juan
edacdfbe	Testing this out to see how well it works.	5 days ago	Li Juan
70bb94d7	Revised test results with correct information.	5 days ago	Li Juan
b78e6d1c	Merge branch 'master' into testbranch	5 days ago	Maria Garcia
84b7d158	Edited ahs_count.py	22 days ago	Maria Garcia

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 \(p. 196\)](#) and [Comment on a Commit \(p. 201\)](#). 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 \(p. 199\)](#).
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

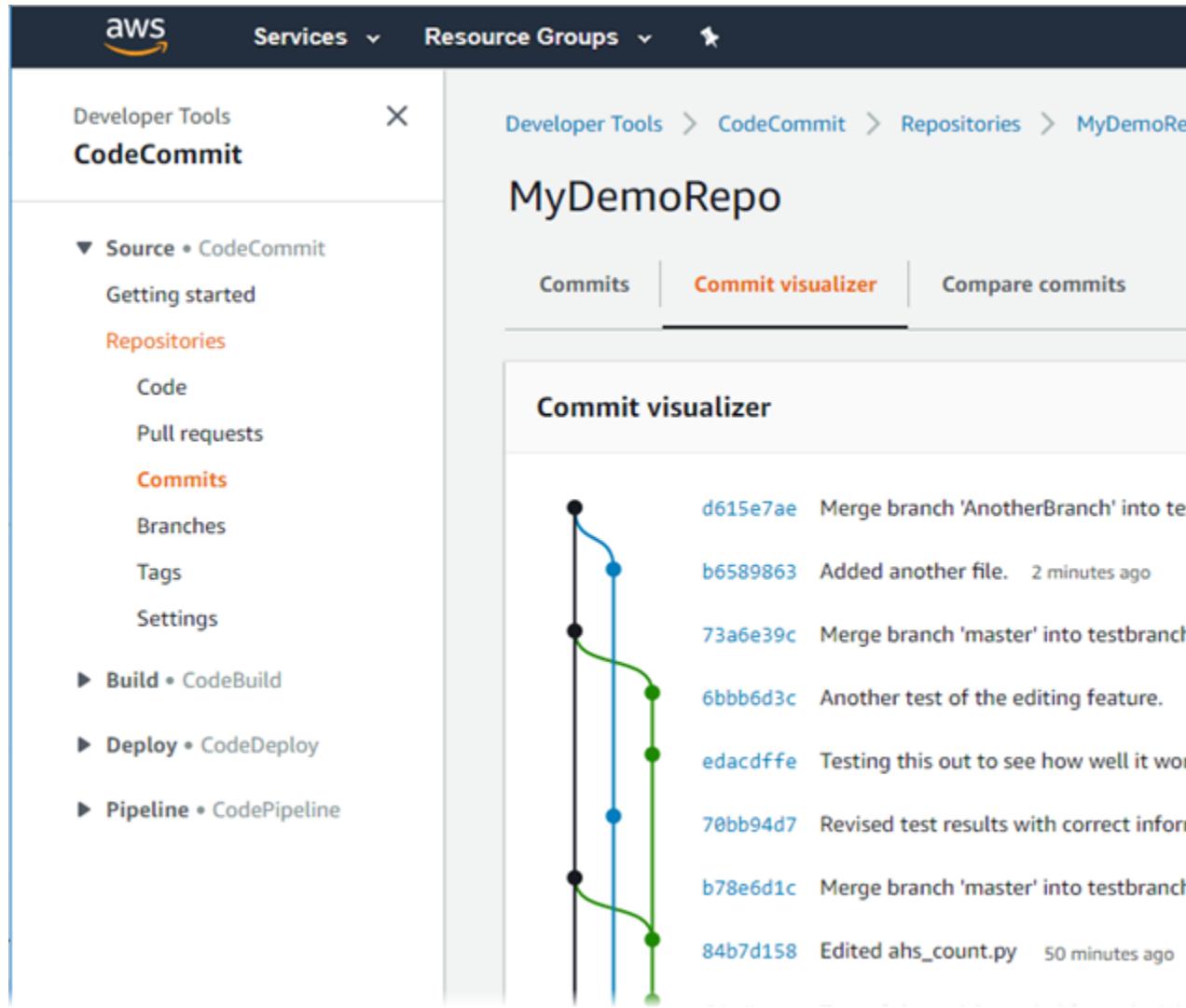
You can view a graph of the commits made to a repository. The **Commit Visualizer** view is a directed acyclic graph (DAG) representation of all the commits made to a branch of the repository. This graphical representation can help you understand when commits and associated features were added or merged. 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.

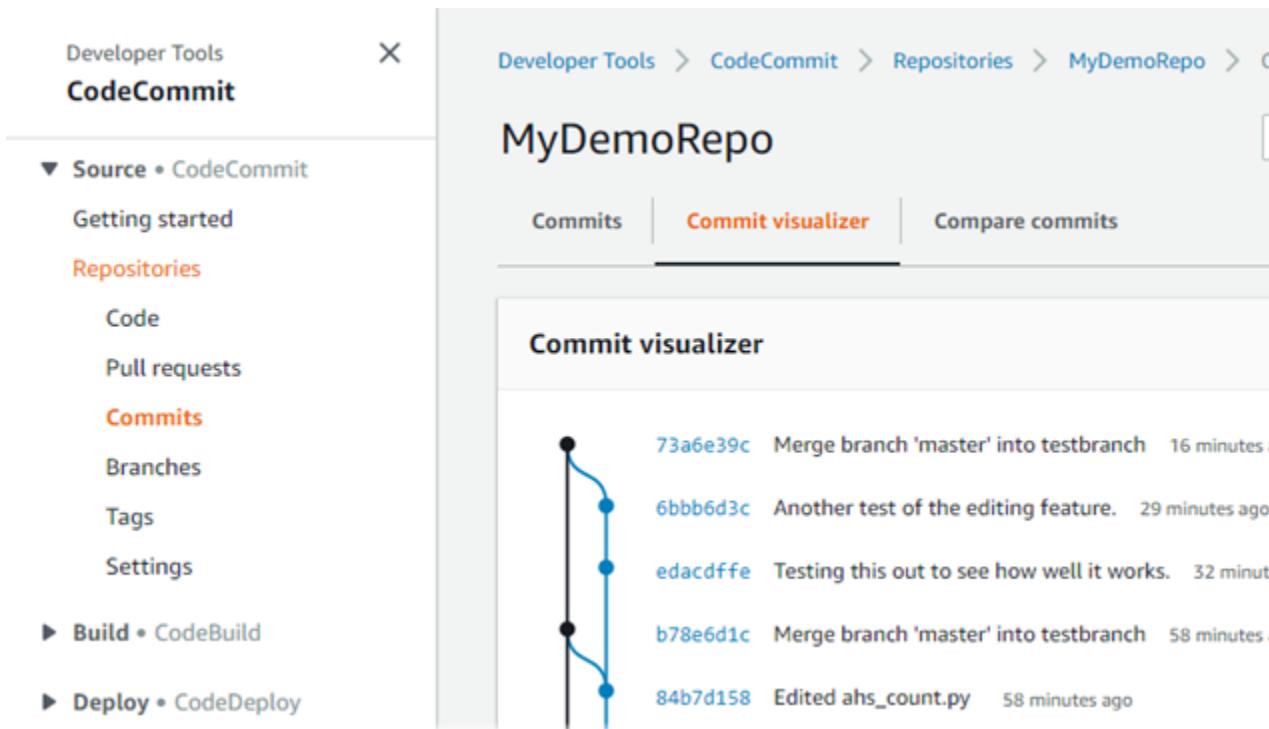


In the commit graph, the abbreviated commit ID and the subject for each commit message appears next to that point in the graph.

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 \(p. 192\)](#).
- 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 \(p. 193\)](#).
- To view the base64-encoded content of a Git blob object in a repository, run [aws codecommit get-blob \(p. 194\)](#).

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 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="  
}
```

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 \(p. 84\)](#).

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 94bale60

commit 94bale60
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 94bale60
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:

```
94bale60 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 \(p. 196\)](#)
- [Compare Any Two Commit Specifiers \(p. 199\)](#)

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.

The screenshot shows the AWS CodeCommit interface. At the top, a navigation bar lists: Developer Tools > CodeCommit > Repositories > MyDemoRepo > Commits > 6bbb6d3c. Below the navigation is the title "Commit 6bbb6d3c". A "Details" section contains author information (Li Juan, li_juan@example.com), commit date (5 days ago), and parent commit (edacdf fe). The commit message is "Another test of the editing feature.". Below the details is a file list with "test3results.txt" selected. A "Browse file content" button is visible next to the file name. At the bottom of the file list are navigation controls: "Page 1 of 1", "Go to file", and a dropdown menu.

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 \(p. 201\)](#).

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 \(p. 229\)](#).

The screenshot shows the AWS CodeCommit interface. At the top, a navigation bar indicates the path: Developer Tools > CodeCommit > Repositories > MyDemoRepo > Commits > 7d09e44c. Below this, the title "Commit 7d09e44c" is displayed, along with "Copy commit ID" and "Browse" buttons.

The main content area is titled "Details". It shows the following information:

Author	Commit date	Parent commit
Mary Major mary_major@example.com	48 minutes ago	e6aca768

The "Commit message" is described as "Adding a readme file to the repository."

Below the details, there is a navigation bar with "Page 1 of 1", "Go to file", and dropdown menus for "Hide whitespace changes" and "Un".

The bottom section displays the file "readme.md" with its content:

```
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
    \ No newline at end of file
```

Buttons for "Browse file contents" and "Comments" are visible next to the file name.

ahs_count.py

[Browse file contents](#)

```
*** @ -4,7 +4,7 @@
4   z = z.count('z')
5
6   total = (ess + z)
7 - alv = "Number of alveolar hissing sibilants: {}"
7 + ahs = "Number of alveolar hissing sibilants: {}"
```

 Li Juan commented Just now

You've reverted to the old value he should remain alv.

[Reply](#) [Edit](#)

[New comment](#)

```
8 print(alv.format(total))
9
10 #When using this script, make sure that you ask
```

```
8 print(ahs.format(total))
9
10 #When using this script,
```

Note

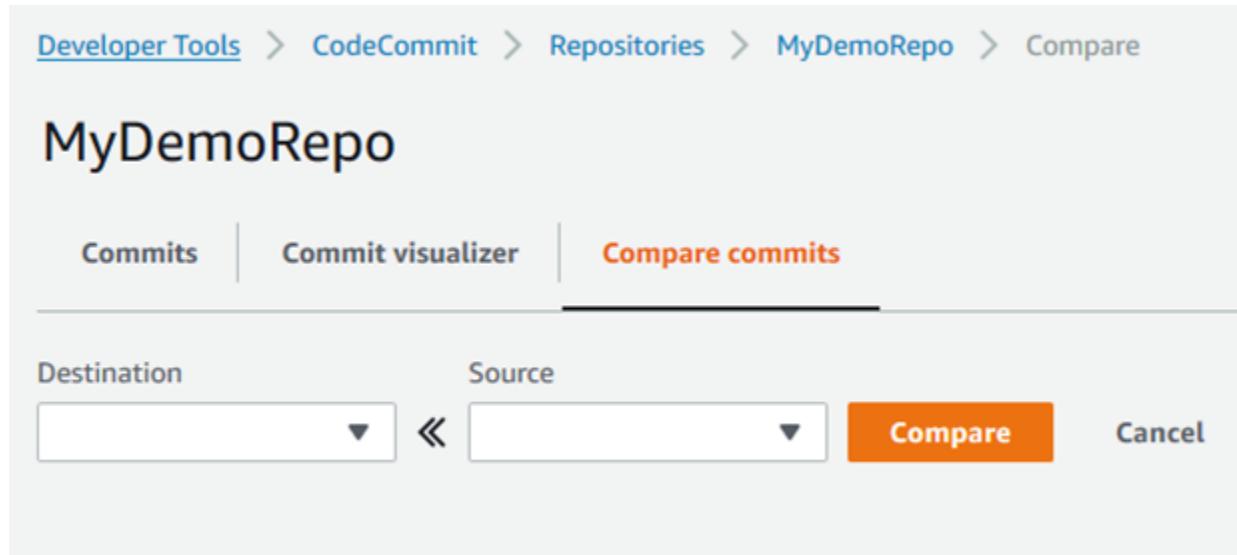
Depending on line ending style, your code editor, and other factors, you might see entire lines added or deleted instead of specific changes in a line. The level of detail matches what's returned in the **git show** or **git diff** commands.

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.

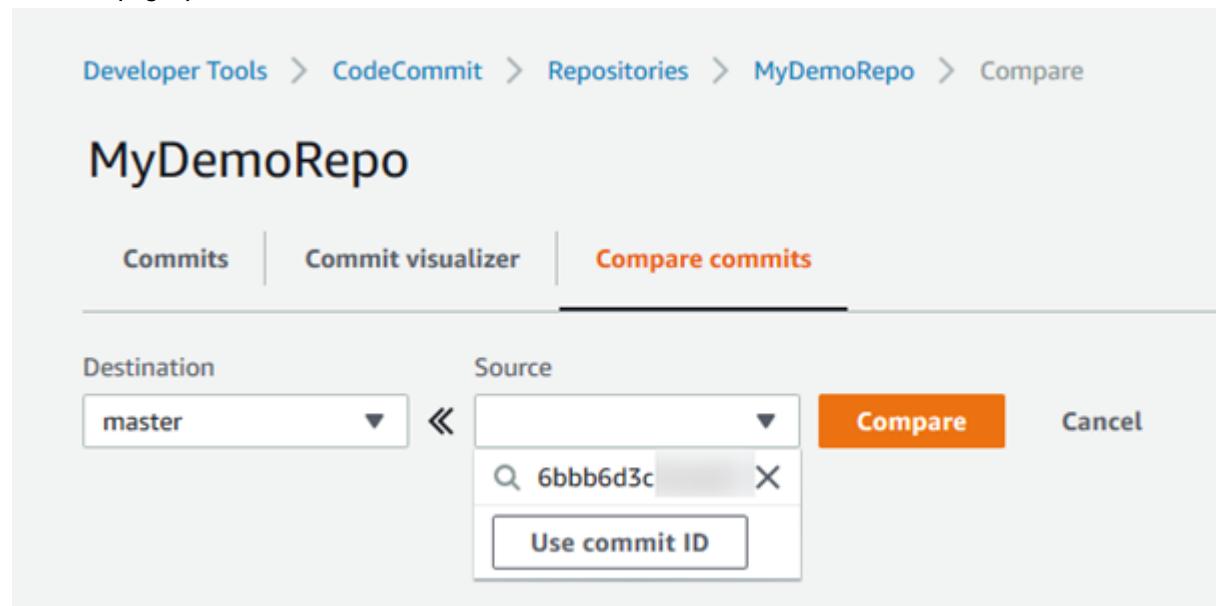
Compare Any Two Commit Specifiers

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**.

The screenshot shows the AWS CodeCommit interface for comparing commits. At the top, a navigation bar includes links to Developer Tools, CodeCommit, Repositories, MyDemoRepo, and Compare. Below this, the repository name 'MyDemoRepo' is displayed. A tab bar at the top of the main content area includes 'Commits', 'Commit visualizer', and 'Compare commits', with 'Compare commits' being the active tab. Underneath, there are dropdown menus for 'Destination' (set to 'master') and 'Source' (set to 'AnotherBranch'), along with a 'Compare' button which is highlighted with a blue border. Below these controls is a search bar with the placeholder 'Go to file'. On the right side of the search bar is a 'Hide whitespace changes' toggle switch. The main content area displays a 'ahs_count.py' file with a unified diff view. The diff shows code changes between two branches. Lines 8 and 9 are highlighted in red, indicating they have been deleted in the source branch. Line 8 contains the code '- print(alv.format(total))'. Line 9 contains the comment '#When using this script, make sure that you ask the subject to use one of the provided texts, such as bumblebee.txt.'. Lines 6 and 7 are highlighted in green, indicating they have been added in the source branch. Line 6 contains the code 'total = (ess + z)'. Line 7 contains the comment 'ahs = "Number of alveolar hissing sibilants: {}"'. Lines 5 and 10 are shown in grey, representing common code. At the bottom of the file view, there is a link to 'text-mecano.txt'.

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.

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 \(p. 202\)](#)
- [Add and Reply to Comments on a Commit in a Repository \(p. 202\)](#)
- [View, Add, Update, and Reply to Comments \(AWS CLI\) \(p. 206\)](#)

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.

Add and Reply to Comments on a Commit (Console)

You can add and reply to comments to a commit. Your comments are marked as those 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.

The screenshot shows a modal window titled "Comments on changes". At the top, there is a "New comment" input field containing the text "Did we also change the variable name in blf.py and concat.py?". Below the input field is a large empty area for the comment body. At the bottom of the modal is an orange "Save" button.

- 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**.

ahs_count.py

Browse file contents

New comment

|

Save

Cancel

- 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**.

ahs_count.py

Browse file contents

```
*** @ -4,7 +4,7 @@
4 z = z.count('z')
5
6 total = (ess + z)
```

 7 - alv = "Number of alveolar hissing sibilants: {}"

```
*** @ -4,7 +4,7 @@
4 z = z.count('z')
5
6 total = (ess + z)
```

 7 + ahs = "Number of alveo

New comment

P

You've reverted to the old value here,
should remain alv.|

Save

Cancel

8 print(alv.format(total))

8 print(alv.format(total))

Note

You can edit your comment after you have saved it, but you cannot delete it from the CodeCommit console. Consider using the **Preview markdown** mode for your comment before you save it.

- To reply to comments on a commit, choose **Reply**.

ahs_count.py

[Browse file contents](#)

```
@@ -4,7 +4,7 @@
4     z = z.count('z')
5
6     total = (ess + z)
7 - alv = "Number of alveolar hissing sibilants: {}"
+ ahs = "Number of alveolar hissing sibilants: {}"
```

Li Juan commented Just now

You've reverted to the old value he should remain alv.

[Reply](#) [Edit](#)

[New comment](#)

```
8     print(alv.format(total))
9
10    #When using this script, make sure that you ask
```

```
8     print(ahs.format(total))
9
10    #When using this script,
```

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.

The screenshot shows the AWS CodeCommit 'Compare' interface. At the top, there's a breadcrumb navigation: Developer Tools > CodeCommit > Repositories > MyDemoRepo > Compare. Below the breadcrumb, the repository name 'MyDemoRepo' is displayed. A navigation bar at the top has three tabs: 'Commits', 'Commit visualizer', and 'Compare commits', with 'Compare commits' being the active tab. Underneath the tabs, there are two dropdown menus labeled 'Destination' and 'Source'. To the right of these dropdowns are two buttons: 'Compare' (orange) and 'Cancel'. The main content area is currently empty, indicating no comparison results have been generated.

4. Use the **Destination** and **Source** fields to compare two commit specifiers. Use the drop-down lists or paste in commit IDs. Choose **Compare**.

The screenshot shows the AWS CodeCommit 'Compare' interface with a comparison between 'AnotherBranch' and commit '6b65eb76'. The 'Compare' button is highlighted in orange. Below the comparison form, the 'ahs_count.py' file is displayed with a diff view. The changes are highlighted in red and green. A 'Browse file contents' button is visible next to the file name. There is also a 'Hide whitespace changes' toggle switch.

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 Comments (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 \(p. 206\)](#).
- To view details on a comment, run [get-comment \(p. 207\)](#).
- To delete the contents of a comment that you created, run [delete-comment-content \(p. 207\)](#).
- To create a comment on the comparison between two commits, run [post-comment-for-compared-commit \(p. 208\)](#).
- To update a comment, run [update-comment \(p. 209\)](#).
- To reply to a comment, [post-comment-reply \(p. 209\)](#).

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": "This is a test comment.",  
          "creationDate": "2018-01-12T12:00:00Z",  
          "lastModifiedDate": "2018-01-12T12:00:00Z",  
          "threadId": "ff30b348EXAMPLEb9aa670f",  
          "threadOrder": 1  
        }  
      ]  
    }  
  ]  
}
```

```
        "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
    },
    {
        "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
    }
],
"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
    }
}
```

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 that you created.

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"  
    }  
}
```

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
    },
    "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.

Note

You can only update the content of a comment that you created.

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
    }
}
```

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  
    }  
}
```

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 \(p. 211\)](#).

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 \(p. 84\)](#).

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 \(p. 210\)](#). Git tags are different from repository tags. For more information about how to use repository tags, see [Add a Tag to a Repository \(p. 96\)](#).

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 \(Console\) \(p. 211\)](#)
- [View Git Tag Details \(Git\) \(p. 212\)](#)

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**.

The screenshot shows the AWS CodeCommit console interface. On the left, there is a navigation sidebar with 'Developer Tools' at the top, followed by 'CodeCommit'. Under 'Source * CodeCommit', there are links for 'Getting started', 'Repositories', 'Code', 'Pull requests', 'Commits', and 'Branches'. Under 'Tags', there is a link for 'Settings'. Below these, under 'Build * CodeBuild', 'Deploy * CodeDeploy', and 'Pipeline * CodePipeline', there are respective links. On the right, the main content area has a breadcrumb trail: 'Developer Tools > CodeCommit > Repositories > MyDemoRepo > Tags'. The title 'MyDemoRepo' is displayed above a section that says 'View a list of tags in your repository, including the date and message of the most recent commit. Content referenced by a tag cannot be edited or changed.' Below this is a 'Tags' section with a search bar. A table lists four tags: 'amended' (Commit ID: dd111962), 'prerelease-2.0' (Commit ID: 98aa867b), 'release' (Commit ID: 94ba1e60), and 'beta' (Commit ID: bdd75ed0). The 'Commit message' column for each tag provides a brief description of the commit.

Tag name	Commit ID	Commit message
amended	dd111962	Removed unneeded files and amended one file. \n The amended file also contains a...
prerelease-2.0	98aa867b	add image files for new feature
release	94ba1e60	Added horse.txt
beta	bdd75ed0	initial commit

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 \(p. 212\)](#) to view a list of Git tag names.
- [git show \(p. 212\)](#) to view information about a specific Git tag.
- [git ls-remote \(p. 213\)](#) 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 \(p. 84\)](#).

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 \(p. 84\)](#).

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 simply 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 impacting 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.

In CodeCommit, you can change the default branch for your repository. This default branch is the one used as the base or default branch in local repos when users clone the 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](#) (p. 190).

Branch name	Last commit date	Commit message
feature-randomizationfeature	7 months ago	Changed methodology to better match introductory class...
jane-branch	12 minutes ago	Adding an example blocking policy that would prevent a...
master	2 hours ago	Adding a readme file to the repository.
new-branch	1 year ago	Added a basic VI tutorial
preprod	2 years ago	...
working-branch	2 years ago	Merge branch 'jane-branch' into working-branch

For information about working with other aspects of your repository in CodeCommit, see [Working with Repositories](#) (p. 81), [Working with Files](#) (p. 140), [Working with Pull Requests](#) (p. 149), [Working with Commits](#) (p. 184), and [Working with User Preferences](#) (p. 229).

Topics

- [Create a Branch in AWS CodeCommit](#) (p. 216)
- [Limit Pushes and Merges to Branches in AWS CodeCommit](#) (p. 219)
- [View Branch Details in AWS CodeCommit](#) (p. 221)

- [Compare Branches in AWS CodeCommit \(p. 224\)](#)
- [Change Branch Settings in AWS CodeCommit \(p. 225\)](#)
- [Delete a Branch in AWS CodeCommit \(p. 226\)](#)

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.

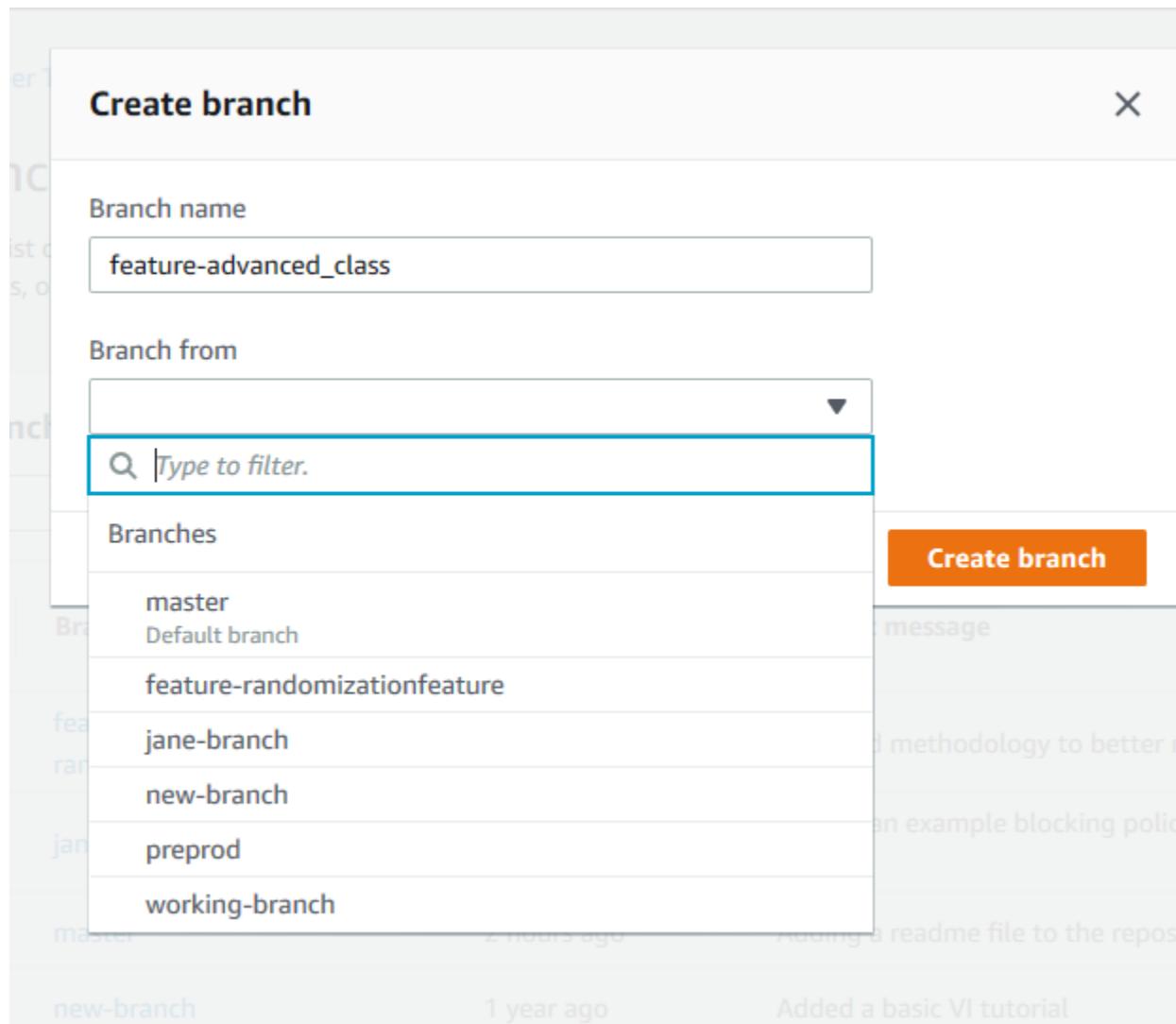
Topics

- [Create a Branch \(Console\) \(p. 216\)](#)
- [Create a Branch \(Git\) \(p. 217\)](#)
- [Create a Branch \(AWS CLI\) \(p. 218\)](#)

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 \(p. 84\)](#).

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 master`), 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 \(p. 325\)](#).

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.

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 \(p. 121\)](#) 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 administrative 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 \(p. 219\)](#)
- [Apply the IAM Policy to an IAM Group or Role \(p. 221\)](#)
- [Test the Policy \(p. 221\)](#)

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 AWS account.

To create a conditional policy for branches

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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, merging pull requests, and adding files to a branch named `master` and a branch named `prod` in a repository named `MyDemoRepo`:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": [  
                "codecommit:GitPush",  
                "codecommit>DeleteBranch",  
                "codecommit:PutFile",  
                "codecommit:MergeBranchesByFastForward",  
                "codecommit:MergeBranchesBySquash",  
                "codecommit:MergeBranchesByThreeWay",  
                "codecommit:MergePullRequestByFastForward",  
                "codecommit:MergePullRequestBySquash",  
                "codecommit:MergePullRequestByThreeWay"  
            ],  
            "Resource": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",  
            "Condition": {  
                "StringEqualsIfExists": {  
                    "codecommit:References": [  
                        "refs/heads/master",  
                        "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 `master` in all repositories in an AWS 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 `DenyChangesToMaster`.
 - 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://console.aws.amazon.com/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 Detaching 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 \(Console\) \(p. 221\)](#)
- [View Branch Details \(Git\) \(p. 222\)](#)
- [View Branch Details \(AWS CLI\) \(p. 223\)](#)

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**.

Branch name	Last commit date	Commit message
feature-randomizationfeature	7 months ago	Changed methodology to better match introductory...
jane-branch	12 minutes ago	Adding an example blocking policy that would pre...
master	2 hours ago	Adding a readme file to the repository.
new-branch	1 year ago	Added a basic VI tutorial
preprod	2 years ago	..
working-branch	2 years ago	Merge branch 'jane-branch' into working-branch

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 \(p. 84\)](#).

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
* master
  remotes/origin/MyNewBranch
  remotes/origin/master
```

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 \(p. 325\)](#).

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 \(p. 223\)](#).
- To view information about a specific branch, run [get-branch \(p. 223\)](#).

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 \(p. 121\)](#) 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",  
        "master"  
    ]  
}
```

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 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 \(p. 224\)](#)
- [Compare Two Specific Branches \(p. 224\)](#)

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 \(p. 229\)](#).

The screenshot shows the AWS CodeCommit interface for comparing branches. The top navigation bar includes 'Developer Tools > CodeCommit > Repositories > MyDemoRepo > Compare'. Below this, the repository name 'MyDemoRepo' is displayed. A navigation bar at the top of the comparison view includes 'Commits', 'Commit visualizer', and 'Compare commits' (which is highlighted). Underneath, there are dropdown menus for 'Destination' set to 'master' and 'Source' set to 'AnotherBranch'. A prominent orange 'Compare' button is centered between them. Below the buttons are page navigation controls ('Page 1 of 1', 'Go to file') and a 'Hide whitespace' checkbox. The main content area displays a diff for the file 'ahs_count.py'. The diff highlights changes in lines 8 and 10. Line 8 changes '- print(alv.format(total))' to '+ print(ahs.format(total))'. Line 10 changes '#When using this script, make sure that you ask the subject to use one of the provided texts, such as bumblebee.txt.' to '#When using this script, make sure use one of the provided texts, such'. A 'Browse file content' link is visible on the right side of the diff view.

Change Branch Settings in AWS CodeCommit

- You can change the default branch to use in the AWS CodeCommit console. You can use the AWS CLI to change the default branch for a repository. To change other branch settings, you can use Git from a local repo connected to the CodeCommit repository.

Topics

- [Change the Default Branch \(Console\) \(p. 225\)](#)
- [Change the Default Branch \(AWS CLI\) \(p. 226\)](#)

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**.

Change the Default Branch (AWS CLI)

To use AWS CLI commands with CodeCommit, install the AWS CLI. For more information, see [Command Line Reference \(p. 325\)](#).

To use the AWS CLI to change a repository's branch settings in a CodeCommit repository, run the following command:

- [update-default-branch \(p. 226\)](#) 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 \(p. 121\)](#) 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 \(p. 223\)](#) 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 \(p. 216\)](#) and [Change Branch Settings \(p. 225\)](#).

Topics

- [Delete a Branch \(Console\) \(p. 227\)](#)
- [Delete a Branch \(AWS CLI\) \(p. 227\)](#)
- [Delete a Branch \(Git\) \(p. 227\)](#)

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 \(p. 325\)](#).

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 \(p. 121\)](#) command.
1. The name of the branch to delete (with the **branch-name** option).

Tip

To get the name of the branch, run the [list-branches \(p. 223\)](#) 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 \(p. 84\)](#).

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 \(p. 245\)](#).

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.

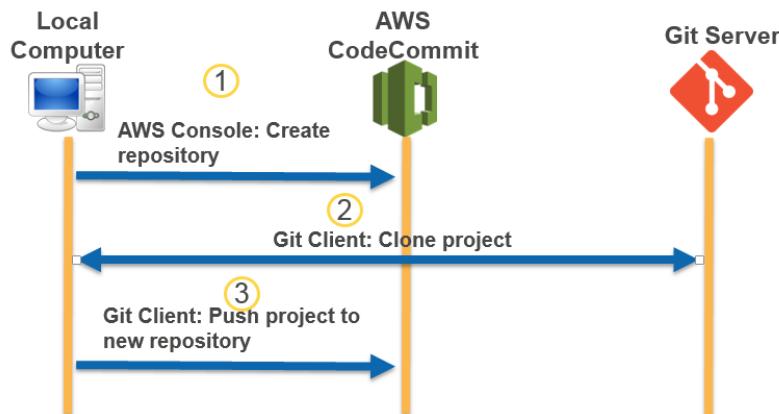
Topics

- [Migrate a Git Repository to AWS CodeCommit \(p. 230\)](#)
- [Migrate Local or Unversioned Content to AWS CodeCommit \(p. 238\)](#)
- [Migrate a Repository Incrementally \(p. 245\)](#)

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 \(p. 231\)](#)
- [Step 1: Create a CodeCommit Repository \(p. 233\)](#)
- [Step 2: Clone the Repository and Push to the CodeCommit Repository \(p. 235\)](#)
- [Step 3: View Files in CodeCommit \(p. 236\)](#)
- [Step 4: Share the CodeCommit Repository \(p. 236\)](#)

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 \(p. 233\)](#).

To create and configure an IAM user for accessing CodeCommit

1. Create an AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

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. 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. 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 the CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of CodeCommit commands.

3. Configure the AWS CLI with 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:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target 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
```

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`
- `eu-central-1`
- `ap-northeast-2`
- `sa-east-1`
- `us-west-1`
- `eu-west-2`
- `ap-south-1`
- `ca-central-1`
- `us-gov-west-1`
- `us-gov-east-1`

For more information about CodeCommit and AWS Regions, see [Regions and Git Connection Endpoints \(p. 304\)](#). For more information about IAM, access keys, and secret keys, see [How Do I Get Credentials?](#) and [Managing Access Keys for IAM Users](#).

- **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.

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 \(p. 253\)](#).

- **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.

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 \(p. 8\)](#) and [Git for Windows: I installed Git for Windows, but I am denied access to my repository \(403\) \(p. 262\)](#).

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 \(p. 253\)](#).

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), 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 \(p. 10\)](#).
- 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 \(p. 29\)](#).
- For SSH on Windows, see [SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit \(p. 34\)](#).
- For the credential helper on Linux, macOS, or Unix, see [Set Up the Credential Helper \(Linux, macOS, or Unix\) \(p. 39\)](#).
- For the credential helper on Windows, see [Set Up the Credential Helper \(Windows\) \(p. 44\)](#).

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 \(AWS CLI\) \(p. 83\)](#).

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 \(p. 304\)](#).

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 AWS 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 \(p. 96\)](#).
7. Choose **Create**.

[Developer Tools](#) > [CodeCommit](#) > [Repositories](#) > [Create repository](#)

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for the repository. Repository names are included in the URLs for that repository.

Repository settings

Repository name

100 characters maximum. Other limits apply.

createRepository.form.field.repositoryDescription.label - optional

This repository was initially cloned from https://

1,000 characters maximum

Can

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 West (Oregon) Region, the URL looks like the following:

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

You need this URL later in [Step 2: Clone the Repository and Push to the CodeCommit Repository \(p. 235\)](#).

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 \(p. 233\)](#)).

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 \(p. 245\)](#).

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.

The screenshot shows the AWS CodeCommit console interface. At the top, there is a navigation bar with 'Developer Tools > CodeCommit > Repositories > MyClonedRepository'. Below the navigation, the repository name 'MyClonedRepository' is displayed. To the right of the repository name is a dropdown menu set to 'master' and a 'Create pull' button. Underneath the repository name, there is a section titled 'MyClonedRepository' with a 'Info' link. This section contains a table with three rows, each representing a file: 'appspec.yml', 'duplicate.py', and 'get-differences.normal.json'. Each row has a small icon and a blue link to the file's details.

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://console.aws.amazon.com/iam/>.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page, next to **Copy an AWS Managed Policy**, choose **Select**.
4. On the **Copy an AWS Managed Policy** page, in **Search Policies**, enter **AWSCodeCommitPowerUser**. Choose **Select** next to the policy name.
5. On the **Review Policy** page, in **Policy Name**, enter a new name for the policy (for example, *AWSCodeCommitPowerUser-MyDemoRepo*).

In **Policy Document**, replace the "*" portion of the Resource line 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 and choose the repository name from the list. For more information, see [View Repository Details \(p. 119\)](#).

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"  
]
```

6. Choose **Validate Policy**. After the policy is validated, 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://console.aws.amazon.com/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 AWS 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 AWS 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 AWS 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.

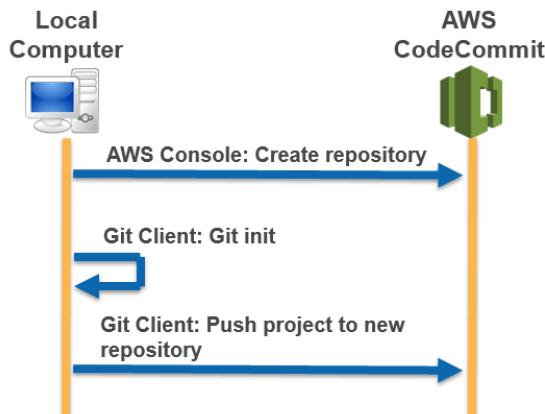
1. Open the CodeCommit console at <https://console.aws.amazon.com/codesuite/codecommit/home>.
2. In the region selector, choose the AWS Region where the repository was created. Repositories are specific to an AWS Region. For more information, see [Regions and Git Connection Endpoints \(p. 304\)](#).
3. On the **Repositories** page, find the name of the repository you want to share.

4. In **Clone URL**, choose the protocol (HTTPS or SSH) 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, for HTTPS, configuring the credential helper for Git).

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 \(p. 238\)](#)
- [Step 1: Create a CodeCommit Repository \(p. 241\)](#)
- [Step 2: Migrate Local Content to the CodeCommit Repository \(p. 242\)](#)
- [Step 3: View Files in CodeCommit \(p. 243\)](#)
- [Step 4: Share the CodeCommit Repository \(p. 243\)](#)

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 \(p. 241\)](#).

To create and configure an IAM user for accessing CodeCommit

1. Create an AWS account by going to <http://aws.amazon.com> and choosing **Sign Up**.
2. Create an IAM user, or use an existing one, in your AWS 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 AWS 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 \(p. 318\)](#).

3. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/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 **AWSCodeCommitFullAccess** or another managed policy for CodeCommit access. For more information, see [AWS Managed \(Predefined\) Policies for CodeCommit \(p. 279\)](#).

After you have selected the policy you want to attach, choose **Next: Review** to review the list of policies that will be attached 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 \(p. 87\)](#) and [Authentication and Access Control for AWS CodeCommit \(p. 270\)](#).

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. 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. 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 the CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of CodeCommit commands.

3. Configure the AWS CLI with 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:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target 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
```

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
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1

For more information about CodeCommit and AWS Regions, see [Regions and Git Connection Endpoints \(p. 304\)](#). For more information about IAM, access keys, and secret keys, see [How Do I Get Credentials?](#) and [Managing Access Keys for IAM Users](#).

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.

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 \(p. 253\)](#).

- **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.

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 \(p. 8\)](#) and [Git for Windows: I installed Git for Windows, but I am denied access to my repository \(403\) \(p. 262\)](#).

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 \(p. 253\)](#).

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, 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 \(p. 10\)](#).
- 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 \(p. 29\)](#).
- For SSH on Windows, see [SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit \(p. 34\)](#).
- For the credential helper on Linux, macOS, or Unix, see [Set Up the Credential Helper \(Linux, macOS, or Unix\) \(p. 39\)](#).
- For the credential helper on Windows, see [Set Up the Credential Helper \(Windows\) \(p. 44\)](#).

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 \(AWS CLI\) \(p. 83\)](#).

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 \(p. 304\)](#).
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 AWS 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 \(p. 96\)](#).

7. Choose **Create**.

Developer Tools > CodeCommit > Repositories > Create repository

Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.

Repository settings

Repository name

MyFirstRepo

100 characters maximum. Other limits apply.

createRepository.form.field.repositoryDescription.label - optional

My first AWS CodeCommit repository.

1,000 characters maximum

Cancel **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 \(p. 242\)](#).

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 **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
```

3. 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 .
```

4. Create a commit for the added files with a commit message.

```
git commit -m "Initial commit"
```

5. 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 \(p. 241\)](#).)

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://console.aws.amazon.com/iam/>.
2. In the **Dashboard** navigation area, choose **Policies**, and then choose **Create Policy**.
3. On the **Create Policy** page, next to **Copy an AWS Managed Policy**, choose **Select**.
4. On the **Copy an AWS Managed Policy** page, in **Search Policies**, enter **AWSCodeCommitPowerUser**. Choose **Select** next to the policy name.
5. On the **Review Policy** page, in **Policy Name**, enter a new name for the policy (for example, *AWSCodeCommitPowerUser-MyDemoRepo*).

In **Policy Document**, replace the "*" portion of the **Resource** line 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 and choose the repository name from the list. For more information, see [View Repository Details \(p. 119\)](#).

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"  
]
```

6. Choose **Validate Policy**. After the policy is validated, 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://console.aws.amazon.com/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 AWS 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 AWS 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 AWS 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 \(p. 304\)](#).
3. On the **Repositories** page, find the name of the repository you want to share.
4. In **Clone URL**, choose the protocol (HTTPS or SSH) 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, for HTTPS, configuring the credential helper for Git).

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 \(p. 6\)](#) and [Create a Repository \(p. 82\)](#).

Topics

- [Step 0: Determine Whether to Migrate Incrementally \(p. 245\)](#)
- [Step 1: Install Prerequisites and Add the CodeCommit Repository as a Remote \(p. 245\)](#)
- [Step 2: Create the Script to Use for Migrating Incrementally \(p. 247\)](#)
- [Step 3: Run the Script and Migrate Incrementally to CodeCommit \(p. 247\)](#)
- [Appendix: Sample Script incremental-repo-migration.py \(p. 248\)](#)

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 \(p. 28\)](#) or [For SSH Connections on Windows \(p. 32\)](#).

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 \(p. 248\)](#) 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" % cur_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)
```

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 \(p. 253\)](#)
- [Troubleshooting SSH Connections to AWS CodeCommit \(p. 254\)](#)
- [Troubleshooting the Credential Helper and HTTPS Connections to AWS CodeCommit \(p. 259\)](#)
- [Troubleshooting Git Clients and AWS CodeCommit \(p. 263\)](#)
- [Troubleshooting Access Errors and AWS CodeCommit \(p. 265\)](#)
- [Troubleshooting Configuration Errors and AWS CodeCommit \(p. 266\)](#)
- [Troubleshooting Console Errors and AWS CodeCommit \(p. 267\)](#)
- [Troubleshooting Triggers and AWS CodeCommit \(p. 268\)](#)
- [Turn on Debugging \(p. 268\)](#)

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 \(p. 253\)](#)
- [Git Credentials for AWS CodeCommit: I set up Git credentials, but my system is not using them \(p. 254\)](#)

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 \(p. 8\)](#) 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 \(p. 8\)](#) and [Credential Storage](#) in the Git 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 \(p. 254\)](#)
- [Access error: Public key is uploaded successfully to IAM and SSH tested successfully but connection fails on Windows systems \(p. 255\)](#)
- [Authentication challenge: Authenticity of host can't be established when connecting to a CodeCommit repository \(p. 256\)](#)
- [IAM error: 'Invalid format' when attempting to add a public key to IAM \(p. 258\)](#)
- [Git on Windows: Bash emulator or command line freezes when attempting to connect using SSH \(p. 258\)](#)

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 \(p. 31\)](#), 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:

The screenshot shows the 'SSH keys for AWS CodeCommit' section of the IAM console. It displays a table with one row. The columns are 'SSH Key ID', 'Uploaded', 'Status', and 'Actions'. The 'SSH Key ID' column contains 'APKAEIBAERJR2EXAMPLE', which is circled in orange. The 'Uploaded' column shows '2015-07-21 16:32 PDT'. The 'Status' column shows 'Active'. The 'Actions' column contains 'Make Inactive | Show SSH Key | Delete'.

SSH Key ID	Uploaded	Status	Actions
APKAEIBAERJR2EXAMPLE	2015-07-21 16:32 PDT	Active	Make Inactive Show SSH Key Delete

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 \(p. 28\)](#).

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 \(p. 28\)](#) or [For SSH Connections on Windows \(p. 32\)](#).

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:0f:0e:0d:00
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:c8:00:00:00:00
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:f7:92:00:00:00:00
git-codecommit.us-west-2.amazonaws.com	SHA256	0pJx9SQpkbPUAHwy58UVIq0IHcyo1fwCp0Ou
git-codecommit.eu-west-1.amazonaws.com	MD5	93:42:36:ea:22:1f:f1:0f:20:02:4a:79:00:00:00:00
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:a7:00:00:00
git-codecommit.ap-northeast-1.amazonaws.com	SHA256	Xk/WeYD/K/bnBybzhiuu4dWpBJtXPf7E30jHU7se40w

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:
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:
git-codecommit.ap-southeast-2.amazonaws.com	SHA256	nYp +gHas80HY3DqbP4yanCDFhqDVjseefVbHEXc
git-codecommit.eu-central-1.amazonaws.com	MD5	74:5a:e8:02:fc:b2:9c:06:10:b4:78:84:
git-codecommit.eu-central-1.amazonaws.com	SHA256	MwGrkiEki8QkkBt1AgXbYt0hoZYBnZF62VY5
git-codecommit.ap-northeast-2.amazonaws.com	MD5	9f:68:48:9b:5f:fc:96:69:39:45:58:87:
git-codecommit.ap-northeast-2.amazonaws.com	SHA256	eegAPOrWY9YsYo9ZHIK0mxetfXBHzAzd8Eya
git-codecommit.sa-east-1.amazonaws.com	MD5	74:99:9d:ff:2b:ef:63:c6:4b:b4:6a:7f:
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:
git-codecommit.us-west-1.amazonaws.com	SHA256	gzauWTWXDK2u5KuMMi5vbKTmfyerdIwgSbzY
git-codecommit.eu-west-2.amazonaws.com	MD5	a5:65:a6:b1:84:02:b1:95:43:f9:0e:de:
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:
git-codecommit.ap-south-1.amazonaws.com	SHA256	hUKwnTj7+Xpx4Kddb6p45j4RazIJ4IhAMD8k
git-codecommit.ca-central-1.amazonaws.com	MD5	9f:7c:a2:2f:8c:b5:74:fd:ab:b7:e1:fd:
git-codecommit.ca-central-1.amazonaws.com	SHA256	Qz5puafQdANVprLljj6r0Qyh41CNsF6ob61d0
git-codecommit.eu-west-3.amazonaws.com	MD5	1b:7f:97:dd:d7:76:8a:32:2c:bd:2c:7b:
git-codecommit.eu-west-3.amazonaws.com	SHA256	uw7c2FL564jVoFgtc +ikzILnKBsZz7t9+CFdSJjkBLI

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-gov-west-1.amazonaws.com	MD5	9f:6c:19:3b:88:cd:e8:88:1b:9c:98:6a:
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:
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:
git-codecommit.eu-north-1.amazonaws.com	SHA256	b6KSK7xKq+V8jl7iuAcjqXsG7zkqoUZZmmhYYFBq1wQ

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 accepts public keys in the OpenSSH format only. 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. This problem most commonly occurs when the public-private key pairs are generated on Windows computers. To generate a key pair and copy the OpenSSH format required by IAM, see [the section called "SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit" \(p. 34\)](#).

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.

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.

Topics

- [I get a command not found error in Windows when using the credential helper \(p. 259\)](#)
- [I am prompted for a user name when I connect to a CodeCommit Repository \(p. 260\)](#)
- [Git for macOS: I configured the credential helper successfully, but now I am denied access to my repository \(403\) \(p. 260\)](#)
- [Git for Windows: I installed Git for Windows, but I am denied access to my repository \(403\) \(p. 262\)](#)

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 \(p. 37\)](#) or [Setup Steps for HTTPS Connections to AWS CodeCommit Repositories on Windows with the AWS CLI Credential Helper \(p. 41\)](#). 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 \(p. 37\)](#).

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
```

In the output from this command, find a line that contains the following option:

```
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. Comment out the following line of text:

```
# helper = osxkeychain
```

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`. 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 instead of HTTPS. For more information, see [For SSH Connections on Linux, macOS, or Unix \(p. 28\)](#).
- 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 \(p. 80\)](#) in [Product and Service Integrations \(p. 73\)](#).

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 \(p. 8\)](#).

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 \(p. 42\)](#).

- 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 \(HTTPS\) \(p. 253\)](#), [Troubleshooting SSH Connections \(p. 254\)](#), and [Troubleshooting the Credential Helper \(HTTPS\) \(p. 259\)](#).

Topics

- [Git error: error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly \(p. 263\)](#)
- [Git error: Too many reference update commands \(p. 263\)](#)
- [Git error: Push via HTTPS is broken in some versions of Git \(p. 264\)](#)
- [Git error: 'gnutls_handshake\(\) failed' \(p. 264\)](#)
- [Git error: Git cannot find the CodeCommit repository or does not have permission to access the repository \(p. 264\)](#)
- [Git on Windows: No supported authentication methods available \(publickey\) \(p. 264\)](#)

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 \(p. 245\)](#). Also, make sure you are not exceeding the size limits for individual files. For more information, see [Limits \(p. 312\)](#).

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 [Limits \(p. 312\)](#).

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 msysgit, 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 \(p. 84\)](#).

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 \(p. 254\)](#).

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 \(p. 265\)](#)
- [Access error: Public key denied when connecting to a CodeCommit repository \(p. 265\)](#)
- [Access error: "Rate Exceeded" or "429" message when connecting to a CodeCommit repository \(p. 266\)](#)

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 \(p. 8\)](#).
- 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 CodeCommit. Consider uninstalling it.

For more information, see [For HTTPS Connections on Windows with the AWS CLI Credential Helper \(p. 41\)](#) and [Git for Windows: I installed Git for Windows, but I am denied access to my repository \(403\) \(p. 262\)](#).

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 \(p. 28\)](#) and [For SSH Connections on Windows \(p. 32\)](#).

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 AWS account in any AWS Region. If requests exceed the maximum rate, you receive an error and further calls are temporarily throttled for your AWS 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 limit 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, AWS 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 \(p. 267\)](#)

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 \(p. 266\)](#)
- [Console error: Cannot browse the code in a CodeCommit repository from the console \(p. 267\)](#)

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 master key for CodeCommit 'aws/codecommit' in your account.`

Possible fixes: The most common cause for this error is that your AWS account is not subscribed to AWS Key Management Service, which is required for CodeCommit. Open the IAM console, choose **Encryption 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 \(p. 318\)](#).

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 AWS 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 \(p. 270\)](#).

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 \(p. 268\)](#)

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. For more information, see [Example 3: Create a Policy for AWS Lambda Integration with a CodeCommit Trigger \(p. 291\)](#).

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 \(p. 115\)](#).

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, create a shell script similar to the following, and then run the script:

```
#!/bin/sh
```

```
gc_output=`script -q -c 'git gc' | grep Total`  
object_count=$(echo $gc_output | awk -F ' \|(\|)' '{print $2}')  
delta_count=$(echo $gc_output | awk -F ' \|(\|)' '{print $5}')  
  
verify_pack_output=`git verify-pack -v objects/pack/pack-* .git/objects/pack/pack-* .pack 2>/dev/null`  
largest_object=$(echo "$verify_pack_output" | grep blob | sort -k3nr | head -n 1 | awk  
'{print $3/1024" KiB"}')  
largest_commit=$(echo "$verify_pack_output" | grep 'tree\|commit\|tag' | sort -k3nr |  
head -n 1 | awk '{print $3/1024" KiB"}')  
longest_delta_chain=$(echo "$verify_pack_output" | grep chain | tail -n 1 | awk -F '  
' '{print $4}')  
  
branch_count=`git branch -a | grep remotes/origin | grep -v HEAD | wc -l`  
if [ $branch_count -eq 0 ]; then  
    branch_count=`git branch -l | wc -l`  
fi  
  
echo "Size: `git count-objects -v | grep size-pack | awk '{print $2}'` KiB"  
echo "Branches: $branch_count"  
echo "Tags: `git show-ref --tags | wc -l`"  
echo "Commits: `git rev-list --all | wc -l`"  
echo "Objects: $object_count"  
echo "Delta objects: $delta_count"  
echo "Largest blob: $largest_object"  
echo "Largest commit/tag/tree: $largest_commit"  
echo "Longest delta chain: $longest_delta_chain"
```

3. If these steps do not provide enough information for you to resolve the issue on your own, ask for help on the [AWS CodeCommit forum](#). Be sure to include relevant output from these steps in your post.

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 \(p. 270\)](#)
- [Access Control \(p. 271\)](#)

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 simply an identity within your AWS 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. 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.

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](#).

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials*, and they provide complete access to all of your AWS resources. Certain CodeCommit features are not available for root account users. In addition, the only way to use Git with your root account is 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 role** – Like an IAM user, an [IAM role](#) is an IAM identity that you can create in your account to grant specific permissions. It is similar to an IAM user, but it is not associated with a specific person. Unlike an IAM user identity, you cannot use Git credentials or SSH keys with this identity type. However, an [IAM role](#) enables you to obtain temporary access keys that you can use to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
 - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as [federated users](#). AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) 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.

- **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the [IAM User Guide](#).
- **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows AWS Lambda to access a CodeCommit repository on your behalf. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the [IAM User Guide](#).
- **Applications running on Amazon EC2** – Instead of storing access keys within an EC2 instance for use by applications running on the instance and for making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An [instance profile](#) contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see [Using Roles for Applications on Amazon EC2](#) in the [IAM User Guide](#).

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 \(p. 271\)](#)
- [Using Identity-Based Policies \(IAM Policies\) for CodeCommit \(p. 275\)](#)
- [CodeCommit Permissions Reference \(p. 291\)](#)

Overview of Managing Access Permissions to Your CodeCommit Resources

Every AWS resource is owned by an AWS 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 \(p. 272\)](#)
- [Understanding Resource Ownership \(p. 273\)](#)
- [Managing Access to Resources \(p. 273\)](#)
- [Resource Scoping in CodeCommit \(p. 274\)](#)
- [Specifying Policy Elements: Resources, Actions, Effects, and Principals \(p. 275\)](#)
- [Specifying Conditions in a Policy \(p. 275\)](#)

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: <i>region</i> : <i>account-id</i> : <i>repository-name</i>
All CodeCommit repositories	arn:aws:codecommit:*
All CodeCommit repositories owned by the specified account in the specified region	arn:aws:codecommit: <i>region</i> : <i>account-id</i> :*

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 AWS 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 \(p. 291\)](#).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created them. Specifically, the resource owner is the AWS 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 AWS account and grant permissions to create CodeCommit resources to that user, the user can create CodeCommit resources. However, your AWS account, to which the user belongs, owns the CodeCommit resources.
- If you use the root account credentials of your AWS account to create a rule, your AWS account is the owner of the CodeCommit resource.
- If you create an IAM role in your AWS account with permissions to create CodeCommit resources, anyone who can assume the role can create CodeCommit resources. Your AWS 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 [AWS 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 \(IAM Policies\) \(p. 273\)](#)
- [Resource-Based Policies \(p. 274\)](#)

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 AWS 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*:

```
{  
    "Version": "2012-10-17",  
    "Statement" : [  
        {  
            "Effect" : "Allow",  
            "Action" : [  
                "codecommit>CreateBranch"  
            ],  
            "Resource" : "arn:aws:codecommit:us-east-2:111111111111: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 [???](#) (p. 284).

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 \(IAM Policies\)](#) (p. 273).

Resource Scoping in CodeCommit

In CodeCommit, you can scope identity-based policies and permissions to resources, as described in [CodeCommit Resources and Operations](#) (p. 272). 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 \(p. 272\)](#) and [CodeCommit Permissions Reference \(p. 291\)](#).

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 \(p. 272\)](#).
- **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 [AWS 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 \(p. 291\)](#).

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 \(p. 271\)](#).

Topics

- Permissions Required to Use the CodeCommit Console (p. 276)
 - Viewing Resources in the Console (p. 278)
 - AWS Managed (Predefined) Policies for CodeCommit (p. 279)
 - Customer Managed Policy Examples (p. 284)

The following is an example of an identity-based permissions policy:

```
{  
    "Version": "2012-10-17",  
    "Statement" : [  
        {  
            "Effect" : "Allow",  
            "Action" : [  
                "codecommit:BatchGetRepositories"  
            ],  
            "Resource" : [  
                "arn:aws:codecommit:us-east-2:111111111111:MyDestinationRepo",  
                "arn:aws:codecommit:us-east-2:111111111111: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 \(p. 291\)](#).

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, as shown in the following permissions policy:

```
    "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:events:*::*:rule/codecommit*"
},
{
  "Sid": "SNSTopicAndSubscriptionAccess",
  "Effect": "Allow",
  "Action": [
    "sns:Subscribe",
    "sns:Unsubscribe"
],
"Resource": "arn:aws: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",
    "iam>ListAWSKMSKeys"
]
}
```

```
    "iam>ListAccessKeys",
    "iam:GetSSHPublicKey"
],
"Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
  "Sid": "IAMUserSSHKeys",
  "Effect": "Allow",
  "Action": [
    "iam>DeleteSSHPublicKey",
    "iam:GetSSHPublicKey",
    "iam>ListSSHPublicKeys",
    "iam:UpdateSSHPublicKey",
    "iam:UploadSSHPublicKey"
  ],
  "Resource": "arn:aws:iam::*:user/${aws:username}"
},
{
  "Sid": "IAMSelfManageServiceSpecificCredentials",
  "Effect": "Allow",
  "Action": [
    "iam>CreateServiceSpecificCredential",
    "iam:UpdateServiceSpecificCredential",
    "iam>DeleteServiceSpecificCredential",
    "iam:ResetServiceSpecificCredential"
  ],
  "Resource": "arn:aws:iam::*:user/${aws:username}"
}
]
```

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 \(p. 318\)](#).

Viewing Resources in the Console

The CodeCommit console requires the `ListRepositories` permission to display a list of repositories for your AWS 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 AWS 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 (Predefined) Policies for CodeCommit

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:

- **AWSCodeCommitFullAccess** – 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 AWS account, including the ability to delete repositories.

The AWSCodeCommitFullAccess policy contains the following policy statement:

```
{
  "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:events:::*:rule/codecommit*"
    },
    {
      "Sid": "SNSTopicAndSubscriptionAccess",
      "Effect": "Allow",
      "Action": [
        "sns:CreateTopic",
        "sns:DeleteTopic",
        "sns:ListTopics",
        "sns:Subscribe",
        "sns:Unsubscribe",
        "sns:Publish"
      ],
      "Resource": "*"
    }
  ]
}
```

```
        "sns:CreateTopic",
        "sns:DeleteTopic",
        "sns:Subscribe",
        "sns:Unsubscribe",
        "sns:SetTopicAttributes"
    ],
    "Resource": "arn:aws: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",
        "iam>ListAccessKeys",
        "iam:GetSSHPublicKey"
    ],
    "Resource": "arn:aws:iam::user/${aws:username}"
},
{
    "Sid": "IAMUserSSHKeys",
    "Effect": "Allow",
    "Action": [
        "iam>DeleteSSHPublicKey",
        "iam:GetSSHPublicKey",
        "iam>ListSSHPublicKeys",
        "iam:UpdateSSHPublicKey",
        "iam:UploadSSHPublicKey"
    ],
    "Resource": "arn:aws:iam::user/${aws:username}"
},
{
    "Sid": "IAMSelfManageServiceSpecificCredentials",
    "Effect": "Allow",
    "Action": [
        "iam>CreateServiceSpecificCredential",
        "iam:UpdateServiceSpecificCredential",
        "iam>DeleteServiceSpecificCredential",
        "iam:ResetServiceSpecificCredential"
    ]
}
```

```

        ],
        "Resource": "arn:aws:iam::*:user/${aws:username}"
    }
]
}

```

- **AWSCodeCommitPowerUser** – 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:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGet*",
                "codecommit:Get*",
                "codecommit>List*",
                "codecommit>Create*",
                "codecommit>DeleteBranch",
                "codecommit:Describe*",
                "codecommit:Put*",
                "codecommit:Post*",
                "codecommit:Merge*",
                "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:events:*::rule/codecommit*"
        },
        {
            "Sid": "SNSTopicAndSubscriptionAccess",
            "Effect": "Allow",
            "Action": [
                "sns>Subscribe",
                "sns>Unsubscribe"
            ],
            "Resource": "arn:aws: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",
        "iam>ListAccessKeys",
        "iam>GetSSHPublicKey"
    ],
    "Resource": "arn:aws:iam:::user/${aws:username}"
},
{
    "Sid": "IAMUserSSHKeys",
    "Effect": "Allow",
    "Action": [
        "iam>DeleteSSHPublicKey",
        "iam>GetSSHPublicKey",
        "iam>ListSSHPublicKeys",
        "iam>UpdateSSHPublicKey",
        "iam>UploadSSHPublicKey"
    ],
    "Resource": "arn:aws:iam:::user/${aws:username}"
},
{
    "Sid": "IAMSelfManageServiceSpecificCredentials",
    "Effect": "Allow",
    "Action": [
        "iam>CreateServiceSpecificCredential",
        "iam>UpdateServiceSpecificCredential",
        "iam>DeleteServiceSpecificCredential",
        "iam>ResetServiceSpecificCredential"
    ],
    "Resource": "arn:aws:iam:::user/${aws:username}"
}
]
}

```

- **AWSCodeCommitReadOnly** – 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:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "codecommit:BatchGet*",  
                "codecommit:Get*",  
                "codecommit:Describe*",  
                "codecommit>List*",  
                "codecommit:GitPull"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "CloudWatchEventsCodeCommitRulesReadOnlyAccess",  
            "Effect": "Allow",  
            "Action": [  
                "events:DescribeRule",  
                "events>ListTargetsByRule"  
            ],  
            "Resource": "arn:aws: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>ListAccessKeys",  
                "iam:GetSSHPublicKey"  
            ],  
            "Resource": "arn:aws:iam::user/${aws:username}"  
        }  
    ]  
}
```

```
}
```

For more information, see [AWS Managed Policies](#) in the *IAM User Guide*.

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 \(p. 284\)](#)
- [Customer Managed Integration Policy Examples \(p. 289\)](#)

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 Region \(p. 284\)](#)
- [Example 2: Allow a User to Use Git for a Single Repository \(p. 285\)](#)
- [Example 3: Allow a User Connecting from a Specified IP Address Range Access to a Repository \(p. 285\)](#)
- [Example 4: Deny or Allow Actions on Branches \(p. 286\)](#)
- [Example 5: Deny or Allow Actions on Repositories with Tags \(p. 288\)](#)

Example 1: Allow a User to Perform CodeCommit Operations in a Single 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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codecommit:*",
            "Resource": "arn:aws:codecommit:us-east-2:111111111111:/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": "us-east-2"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "codecommit:*",
            "Resource": "arn:aws:codecommit:us-east-2:111111111111:/*",
            "Condition": {
                "NotStringEquals": {
                    "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`:

```
{
    "Version": "2012-10-17",
    "Statement" : [
        {
            "Effect" : "Allow",
            "Action" : [
                "codecommit:GitPull",
                "codecommit:GitPush"
            ],
            "Resource" : "arn:aws:codecommit:us-east-2:111111111111: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`:

```
{
    "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`:

```
{
    "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:codecommit:us-east-2:111111111111: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 \(p. 219\)](#).

For example, you can create a `Deny` policy that denies users the ability to make changes to a branch named `master`, 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 *master*, add or edit a file in the *master* branch in the CodeCommit console, or merge branches or a pull request into the *master* 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 *master* in all repositories in your AWS account, for **Resource**, specify an asterisk (*) instead of a repository ARN.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": [  
                "codecommit:GitPush",  
                "codecommit>DeleteBranch",  
                "codecommit:PutFile",  
                "codecommit:Merge*"  
            ],  
            "Resource": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",  
            "Condition": {  
                "StringEqualsIfExists": {  
                    "codecommit:References": [  
                        "refs/heads/master"  
                    ]  
                },  
                "Null": {  
                    "codecommit:References": false  
                }  
            }  
        }  
    ]  
}
```

The following example policy allows a user to make changes to a branch named *master* in all repositories in an AWS account. You might use this policy with the **AWSCodeCommitReadOnly** managed policy to allow automated pushes to the repository. Because the Effect is **Allow**, this example policy would not work with managed policies such as **AWSCodeCommitPowerUser**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "codecommit:GitPush",  
                "codecommit:Merge*"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringNotEqualsIfExists": {  
                    "codecommit:References": [  
                        "refs/heads/master"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

```
}
```

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*:

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "codecommit:*"
      "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 all repositories except those tagged with the specified tags:

```
{
  "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:TagResource",
        "codecommit:Test*",
        "codecommit:UntagResource",
        "codecommit:Update*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/Status: Secret",
          "aws:ResourceTag/Team: Saanvi"
        }
      }
    }
  ]
}
```

```
    ]  
}
```

Customer Managed Integration Policy Examples

This section provides example customer-managed user policies that grant permissions for integrations between CodeCommit and other AWS services. For specific examples of policies that allow cross-account access to a CodeCommit repository, see [Configure Cross-Account Access to an AWS CodeCommit Repository \(p. 129\)](#).

Note

All examples use the US West (Oregon) Region (us-west-2) when a region is required, and contain fictitious account IDs.

Examples

- [Example 1: Create a Policy That Enables Cross-Account Access to an Amazon SNS Topic \(p. 289\)](#)
- [Example 2: Create an Amazon Simple Notification Service \(Amazon SNS\) Topic Policy to Allow Amazon CloudWatch Events to Publish CodeCommit Events to the Topic \(p. 290\)](#)
- [Example 3: Create a Policy for AWS Lambda Integration with a CodeCommit Trigger \(p. 291\)](#)

Example 1: Create a Policy That Enables Cross-Account Access to an Amazon SNS Topic

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). If you create the Amazon SNS topic with the same account used to create the CodeCommit repository, you do not need to configure additional IAM policies or permissions. You can create the topic, and then create the trigger for the repository. For more information, see [Create a Trigger for an Amazon SNS Topic \(p. 102\)](#).

However, if you want to configure your trigger to use an Amazon SNS topic in another AWS account, you must first configure that topic with a policy that allows CodeCommit to publish to that topic. From that other account, open the Amazon SNS console, choose the topic from the list, and for **Other topic actions**, choose **Edit topic policy**. On the **Advanced** tab, modify the policy for the topic to allow CodeCommit to publish to that topic. For example, if the policy is the default policy, you would modify the policy as follows, changing the items in *red italic text* to match the values for your repository, Amazon SNS topic, and account:

```
{  
  "Version": "2008-10-17",  
  "Id": "__default_policy_ID",  
  "Statement": [  
    {  
      "Sid": "__default_statement_ID",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "*"  
      },  
      "Action": [  
        "SNS:Subscribe",  
        "SNS>ListSubscriptionsByTopic",  
        "SNS>DeleteTopic",  
        "SNS>GetTopicAttributes",  
        "SNS>Publish",  
        "SNS>RemovePermission",  
        "SNS>AddPermission",  
        "SNS>Receive",  
        "SNS>SetTopicAttributes"  
      ],  
    }  
  ]  
},
```

```

    "Resource": "arn:aws:sns:us-east-2:111111111111:NotMySNSTopic",
    "Condition": {
        "StringEquals": {
            "AWS:SourceOwner": "111111111111"
        }
    },
    {
        "Sid": "CodeCommit-Policy_ID",
        "Effect": "Allow",
        "Principal": {
            "Service": "codecommit.amazonaws.com"
        },
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:111111111111:NotMySNSTopic",
        "Condition": {
            "StringEquals": {
                "AWS:SourceArn": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",
                "AWS:SourceAccount": "111111111111"
            }
        }
    }
]
}

```

Example 2: Create an Amazon Simple Notification Service (Amazon SNS) Topic Policy to Allow Amazon CloudWatch Events to Publish CodeCommit Events to the Topic

You can configure CloudWatch Events to publish to an Amazon SNS topic when events occur, including CodeCommit events. To do so, you must make sure that CloudWatch Events has permission to publish events to your Amazon SNS topic by creating a policy for the topic or modifying an existing policy for the topic similar to the following:

```
{
    "Version": "2012-10-17",
    "Id": "__default_policy_ID",
    "Statement": [
        {
            "Sid": "__default_statement_ID",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": {
                "SNS:Publish"
            },
            "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",
            "Condition": {
                "StringEquals": {"AWS:SourceOwner": "123456789012"}
            }
        },
        {
            "Sid": "Allow_Publish_Events",
            "Effect": "Allow",
            "Principal": {"Service": "events.amazonaws.com"},
            "Action": "sns:Publish",
            "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
        }
    ]
}
```

For more information about CodeCommit and CloudWatch Events, see [CloudWatch Events Event Examples From Supported Services](#).

Example 3: Create a Policy for AWS Lambda Integration with a CodeCommit Trigger

You can configure a CodeCommit repository so that code pushes or other events trigger actions, such as invoking a function in AWS Lambda. For more information, see [Create a Trigger for a Lambda Function \(p. 107\)](#). This information is specific to triggers, and not CloudWatch Events.

If you want your trigger to run a Lambda function directly (instead of using an Amazon SNS topic to invoke the Lambda function), and you do not configure the trigger in the Lambda console, you must include a policy similar to the following in the function's resource policy:

```
{  
    "Statement": {  
        "StatementId": "Id-1",  
        "Action": "lambda:InvokeFunction",  
        "Principal": "codecommit.amazonaws.com",  
        "SourceArn": "arn:aws:codecommit:us-east-2:111111111111:MyDemoRepo",  
        "SourceAccount": "111111111111"  
    }  
}
```

When manually configuring a CodeCommit trigger that invokes a Lambda function, you must also use the Lambda [AddPermission](#) command to grant permission for CodeCommit to invoke the function. For an example, see the [To allow CodeCommit to run a Lambda function \(p. 110\)](#) section of [Create a Trigger for an Existing Lambda Function \(p. 110\)](#).

For more information about resource policies for Lambda functions, see [AddPermission](#) and [The Pull/Push Event Models](#) in the *AWS Lambda Developer Guide*.

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 \(p. 271\)](#) 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*.

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 \(p. 292\)](#)
- [Permissions for Actions on Branches \(p. 292\)](#)
- [Permissions for Actions on Merges \(p. 294\)](#)
- [Permissions for Actions on Pull Requests \(p. 294\)](#)
- [Permissions for Actions on Individual Files \(p. 297\)](#)
- [Permissions for Actions on Comments \(p. 297\)](#)
- [Permissions for Actions on Committed Code \(p. 298\)](#)
- [Permissions for Actions on Repositories \(p. 300\)](#)
- [Permissions for Actions on Tags \(p. 301\)](#)
- [Permissions for Actions on Triggers \(p. 301\)](#)
- [Permissions for Actions on CodePipeline Integration \(p. 302\)](#)

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.

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 \(p. 292\)](#). For related permissions on pull requests, see [Permissions for Actions on Pull Requests \(p. 294\)](#).

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 \(p. 297\)](#).

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*

DescribePullRequestEvents

Action(s): codecommit:DescribePullRequestEvents

Required to return information about one or more pull request events.

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 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.

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): codecommitMergePullRequestByFastForward

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): codecommitMergePullRequestBySquash

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): codecommitMergePullRequestByThreeWay

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*

[PostCommentForPullRequest](#)

Action(s): codecommitPostCommentForPullRequest

Required to post a comment on a pull request in a CodeCommit repository.

Resource: arn:aws:codecommit:*region*:*account-id*:*repository-name*

[UpdatePullRequestDescription](#)

Action(s): codecommitUpdatePullRequestDescription

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): codecommitUpdatePullRequestStatus

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): codecommitUpdatePullRequestTitle

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 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`

[GetFile](#)

Action(s): `codecommit:GetFile`

Required to view the encoded content of an 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 \(p. 294\)](#).

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*

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*

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 AWS 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 AWS account. The only allowed value for Resource for this action is all repositories (*).

Resource: *

[UpdateRepositoryDescription](#)

Action(s): codecommitUpdateRepositoryDescription

Required to change the description of a CodeCommit repository.

Resource: arn:aws:codecommit:*region*:*account-id*:*repository-name*

[UpdateRepositoryName](#)

Action(s): codecommitUpdateRepositoryName

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): codecommitListTagsForResource

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): codecommitTagResource

Required to add or edit AWS tags for a repository.

Resource: arn:aws:codecommit:*region*:*account-id*:*repository-name*

[UntagResource](#)

Action(s): codecommitUntagResource

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`

AWS CodeCommit Reference

The following reference topics can help you better understand CodeCommit, Git, AWS Regions, product limitations, and more.

Topics

- [Regions and Git Connection Endpoints for AWS CodeCommit \(p. 304\)](#)
- [Using AWS CodeCommit with Interface VPC Endpoints \(p. 310\)](#)
- [Limits in AWS CodeCommit \(p. 312\)](#)
- [Temporary Access to AWS CodeCommit Repositories \(p. 315\)](#)
- [AWS Key Management Service and Encryption for AWS CodeCommit Repositories \(p. 318\)](#)
- [Logging AWS CodeCommit API Calls with AWS CloudTrail \(p. 319\)](#)
- [AWS CodeCommit Command Line Reference \(p. 325\)](#)
- [Basic Git Commands \(p. 327\)](#)

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 Regions for CodeCommit \(p. 304\)](#)
- [Git Connection Endpoints \(p. 305\)](#)
- [Server Fingerprints for CodeCommit \(p. 308\)](#)

Supported 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)
- EU (Ireland)
- EU (London)
- EU (Paris)
- EU (Frankfurt)

- EU (Stockholm)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Seoul)
- Asia Pacific (Mumbai)
- South America (São Paulo)
- Canada (Central)
- 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 \(p. 305\)](#).

For more information about regional endpoints for AWS CLI, service, and API calls to CodeCommit, see [AWS Regions and Endpoints](#).

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
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

Region Name	Region	Endpoint URL	Protocol
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
EU (Ireland)	eu-west-1	https://git-codecommit.eu-west-1.amazonaws.com	HTTPS
EU (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
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
EU (Frankfurt)	eu-central-1	https://git-codecommit.eu-central-1.amazonaws.com	HTTPS
EU (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

Region Name	Region	Endpoint URL	Protocol
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
EU (London)	eu-west-2	https://git-codecommit.eu-west-2.amazonaws.com	HTTPS
EU (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
Asia Pacific (Mumbai)	ap-south-1	ssh://git-codecommit.ap-south-1.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
EU (Paris)	eu-west-3	https://git-codecommit.eu-west-3.amazonaws.com	HTTPS
EU (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
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

Region Name	Region	Endpoint URL	Protocol
AWS GovCloud (US-East)	us-gov-east-1	https://git-codecommit-fips.us-gov-east-1.amazonaws.com	HTTPS
EU (Stockholm)	eu-north-1	https://git-codecommit.eu-north-1.amazonaws.com	HTTPS
EU (Stockholm)	eu-north-1	ssh://git-codecommit.eu-north-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:
git-codecommit.us-east-2.amazonaws.com	SHA256	31B1W2g5xn/NA2Ck6dyeJIrQOWvn7n8UEs56fG6ZIzQ
git-codecommit.us-east-1.amazonaws.com	MD5	a6:9c:7d:bc:35:f5:d4:5f:8b:ba:6f:c8:
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:
git-codecommit.us-west-2.amazonaws.com	SHA256	0pJx9SQpkbPUAHwy58UVIq0IHcyo1fwCp0Ou
git-codecommit.eu-west-1.amazonaws.com	MD5	93:42:36:ea:22:1f:f1:0f:20:02:4a:79:
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:
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:
git-codecommit.ap-southeast-1.amazonaws.com	SHA256	ZIsVa7OVzxrTIf+Rk4UbhPv6Es22mSB3uTBojfPXIno

Server	Cryptographic hash type	Fingerprint
git-codecommit.ap-southeast-2.amazonaws.com	MD5	7b:d2:c1:24:e6:91:a5:7b:fa:c1:0c:35:
git-codecommit.ap-southeast-2.amazonaws.com	SHA256	nYp +gHas80HY3DqbP4yanCDFhqDVjseefVbHEXo
git-codecommit.eu-central-1.amazonaws.com	MD5	74:5a:e8:02:fc:b2:9c:06:10:b4:78:84:
git-codecommit.eu-central-1.amazonaws.com	SHA256	MwGrkiEki8QkkBtlAgXbYt0hoZYBnZF62VY5
git-codecommit.ap-northeast-2.amazonaws.com	MD5	9f:68:48:9b:5f:fc:96:69:39:45:58:87:
git-codecommit.ap-northeast-2.amazonaws.com	SHA256	eegAPOrWY9YsYo9ZHIK0mxetfXBHzAZd8Eya
git-codecommit.sa-east-1.amazonaws.com	MD5	74:99:9d:ff:2b:ef:63:c6:4b:b4:6a:7f:
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:
git-codecommit.us-west-1.amazonaws.com	SHA256	gzauWTWXDK2u5KuMMi5vbKTmfyerdIwgSbzY
git-codecommit.eu-west-2.amazonaws.com	MD5	a5:65:a6:b1:84:02:b1:95:43:f9:0e:de:
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:
git-codecommit.ap-south-1.amazonaws.com	SHA256	hUKwnTj7+Xpx4Kddb6p45j4RaziJ4IhAMD8l
git-codecommit.ca-central-1.amazonaws.com	MD5	9f:7c:a2:2f:8c:b5:74:fd:ab:b7:e1:fd:
git-codecommit.ca-central-1.amazonaws.com	SHA256	Qz5puafQdANVprLlj6r0Qyh4lCNsF6ob61d0
git-codecommit.eu-west-3.amazonaws.com	MD5	1b:7f:97:dd:d7:76:8a:32:2c:bd:2c:7b:
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:
git-codecommit.us-gov-west-1.amazonaws.com	SHA256	djXQoSIFcg8vHe0Kvh1xW/ gOF9X37tWTqu4Hkng75x4

Server	Cryptographic hash type	Fingerprint
git-codecommit.us-gov-east-1.amazonaws.com	MD5	00:8d:b5:55:6f:05:78:05:ed:ea:cb:3f:
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:
git-codecommit.eu-north-1.amazonaws.com	SHA256	b6KSK7xKq +v8jl7iuAcjqXSG7zkqoUZZmmhYYFBq1wQ

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.

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)
- EU (Ireland)

- EU (London)
- EU (Paris)
- EU (Frankfurt)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Seoul)
- Asia Pacific (Mumbai)
- South America (São Paulo)
- Canada (Central)
- AWS GovCloud (US-West)
- EU (Stockholm)

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.
- **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.

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 \(p. 285\)](#). 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-west-2:123456789012:MyDemoRepo",
            "Principal": "*"
        }
    ]
}
```

For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

Limits in AWS CodeCommit

The following table describes limits in CodeCommit. For information about limits that can be changed, see [AWS Service Limits](#).

Number of repositories	Maximum of 1,000 per AWS account. This limit can be changed. For more information, see AWS Service Limits .
Regions	CodeCommit is available in the following regions: <ul style="list-style-type: none">• US East (Ohio)• US East (N. Virginia)• US West (N. California)• US West (Oregon)• EU (Ireland)• EU (London)• EU (Paris)• EU (Frankfurt)• EU (Stockholm)• Asia Pacific (Tokyo)• Asia Pacific (Singapore)• Asia Pacific (Sydney)• Asia Pacific (Seoul)• Asia Pacific (Mumbai)• South America (São Paulo)• Canada (Central)• AWS GovCloud (US-West)• AWS GovCloud (US-East)

	For more information, see Regions and Git Connection Endpoints (p. 304) .
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 triggers in a repository	Maximum of 10.
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: ! ? @ # \$ % ^ & * () + = { } [] \ / > < ~ ` ' " ; :
Branch names	<p>Any combination of allowed characters between 1 and 256 characters in length. Branch names cannot:</p> <ul style="list-style-type: none"> 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: ? ^ * [\ ~ : <p>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.</p>
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.
Repository tags	Tags are case sensitive. Maximum of 50 per resource.
Repository tag key names	<p>Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 128 characters in length. Allowed characters are + - = . _ : / @</p> <p>Tag key names must be unique, and each key can only have one value. A tag cannot:</p> <ul style="list-style-type: none"> begin with aws : consist only of spaces end with a space contain emojis or any of the following characters: ? ^ * [\ ~ ! # \$ % & * () > < " ' ` [] { } ;

Repository tag values	<p>Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 256 characters in length. Allowed characters are + - = . _ : / @</p> <p>A key can only have one value, but many keys can have the same value. A tag cannot:</p> <ul style="list-style-type: none"> • begin with aws : • consist only of spaces • end with a space • contain emojis or any of the following characters: ? ^ * [\ ~ ! # \$ % & * () > < " ' ` [] { } ;
User names in commits made in the console	Any combination of allowed characters between 1 and 1,024 characters in length.
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.
Repository descriptions	Any combination of characters between 0 and 1,000 characters in length. Repository descriptions are optional.
Metadata for a commit	<p>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.</p> <p>Note There is no limit on the number or the total size of all files in a single commit, as long as the data does not exceed 20 MB, an individual file does not exceed 6 MB, and a single blob does not exceed 2 GB.</p>
File size	Maximum of 6 MB for any individual file when using the CodeCommit console, APIs, or the AWS CLI.

File paths	<p>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:</p> <ul style="list-style-type: none"> • contain empty strings • be a relative file path • include any of the following character combinations: <p style="text-align: center;">/ . / / . . / / / • end with a trailing slash or backslash</p> <p>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.</p>
Git blob size	<p>Maximum of 2 GB.</p> <p>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.</p>
Custom data for triggers	This is a string field limited to 1,000 characters. It cannot be used to pass any dynamic parameters.
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.

Temporary Access to AWS CodeCommit Repositories

You can give users temporary access to your AWS CodeCommit repositories. You might do this to allow IAM users to access CodeCommit repositories in separate AWS 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 \(p. 129\)](#).

You can also 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 the AWS CLI credential helper to connect to CodeCommit repositories. You cannot use SSH or Git credentials and HTTPS to connect to CodeCommit repositories with temporary access credentials.

To give users temporary access to your CodeCommit repositories, complete the following steps.

Do not 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 \(p. 37\)](#) or [For HTTPS Connections on Windows with the AWS CLI Credential Helper \(p. 41\)](#).
- 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 \(p. 37\)](#) or [For HTTPS Connections on Windows with the AWS CLI Credential Helper \(p. 41\)](#).

Amazon EC2 instances that meet the preceding requirements are already set up to communicate temporary access credentials to CodeCommit on your behalf.

Step 1: Complete the Prerequisites

Complete the setup steps to provide a user with temporary access to your CodeCommit repositories:

- For cross-account access, see [Walkthrough: Delegating Access Across AWS Accounts Using IAM 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 \(p. 270\)](#) to specify the CodeCommit permissions you want to temporarily grant the user.

Step 2: Get Temporary Access Credentials

Depending on the way you set up temporary access, your user can get temporary access 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.

Your user should receive a set of temporary access credentials, which include an AWS access key ID, a secret access key, and a session token. Your user should make a note of these three values because they are used in the next step.

Step 3: Configure the AWS CLI with Your Temporary Access Credentials

Your user must configure the development machine to use those temporary access credentials.

1. Follow the instructions in [Setting Up \(p. 6\)](#) to set up the AWS CLI. Use the **aws configure** command to configure a profile.

Note

Before you continue, make sure the `gitconfig` file is configured to use the AWS profile you configured in the AWS CLI.

2. You can associate the temporary access credentials with the user's AWS CLI named profile in one of the following ways. Do not use the **aws configure** command.
 - In the `~/.aws/credentials` file (for Linux) or the `%UserProfile%\.aws\credentials` file (for Windows), add to the user's AWS CLI named profile the `aws_access_key_id`, `aws_secret_access_key`, and `aws_session_token` setting values:

```
[CodeCommitProfileName]
aws_access_key_id=TheAccessKeyID
aws_secret_access_key=TheSecretAccessKey
aws_session_token=TheSessionToken
```

-OR-

- Set the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables:

For Linux, macOS, or Unix:

```
export AWS_ACCESS_KEY_ID=TheAccessKey
export AWS_SECRET_ACCESS_KEY=TheSecretAccessKey
export AWS_SESSION_TOKEN=TheSessionToken
```

For Windows:

```
set AWS_ACCESS_KEY_ID=TheAccessKey
set AWS_SECRET_ACCESS_KEY=TheSecretAccessKey
set AWS_SESSION_TOKEN=TheSessionToken
```

For more information, see [Configuring the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

3. Set up the Git credential helper with the AWS CLI named profile associated with the temporary access credentials.
 - [Linux, macOS, or Unix \(p. 37\)](#)
 - [Windows \(p. 41\)](#)

As you follow these steps, do not call the `aws configure` command. You already specified temporary access credentials through the credentials file or the environment variables. If you use environment variables instead of the credentials file, in the Git credential helper, specify `default` as the profile name.

Step 4: Access the CodeCommit Repositories

Assuming your user has followed the instructions in [Connect to a Repository \(p. 84\)](#) to connect to the CodeCommit repositories, the user then uses Git to call `git clone`, `git push`, and `git pull` to clone, push to, and pull from, the CodeCommit repositories to which he or she has temporary access.

When the user uses the AWS CLI and specifies the AWS CLI named profile associated with the temporary access credentials, results scoped to that profile are returned.

If the user receives the `403: Forbidden` error in response to calling a Git command or a command in the AWS CLI, it's likely the temporary access credentials have expired. The user must go back to [step 2 \(p. 316\)](#) and get a new set of temporary access 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.

The first time you create a CodeCommit repository in a new AWS Region in your AWS account, CodeCommit creates an AWS-managed key (the `aws/codecommit` key) in that same AWS Region in AWS Key Management Service (AWS KMS). This key is used only by CodeCommit (the `aws/codecommit` key). It is stored in your AWS account. CodeCommit uses this AWS-managed key to encrypt and decrypt the data in this and all other CodeCommit repositories within that region in your AWS account.

Important

CodeCommit performs the following AWS KMS actions against the default `aws/codecommit` key. An IAM 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. When you create your first repository, your AWS account must not have any of the following permissions set to deny:

- `"kms:Encrypt"`
- `"kms:Decrypt"`
- `"kms:ReEncrypt"`
- `"kms:GenerateDataKey"`
- `"kms:GenerateDataKeyWithoutPlaintext"`
- `"kms:DescribeKey"`

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 IAM console at <https://console.aws.amazon.com/iam/>.
2. In the service navigation pane, choose **Encryption Keys**. (If a welcome page appears, choose **Get Started Now**.)
3. In **Filter**, choose the AWS Region for your repository. For example, if the repository was created in us-east-2, make sure the filter is set to US East (Ohio).
4. In the list of encryption keys, choose the AWS-managed key with the alias **aws/codecommit**. Basic information about the AWS-managed key is displayed.

You cannot change or delete this AWS-managed key. You cannot use a customer-managed key in AWS KMS to encrypt or decrypt data in CodeCommit repositories.

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](#).

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 AWS 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 AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS 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 AWS 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 \(p. 291\)](#). 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 \(p. 321\)](#)
- [Example: A log entry for creating a CodeCommit repository \(p. 321\)](#)

- Examples: Log entries for Git pull calls to a CodeCommit repository (p. 322)
- Example: A log entry for a successful push to a CodeCommit repository (p. 323)

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.


```
        "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"
}
```

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. 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. 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 the CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of CodeCommit commands.

3. Configure the AWS CLI with 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:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target 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
```

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
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ca-central-1
- us-gov-west-1
- us-gov-east-1

For more information about CodeCommit and AWS Regions, see [Regions and Git Connection Endpoints \(p. 304\)](#). For more information about IAM, access keys, and secret keys, see [How Do I Get Credentials?](#) and [Managing Access Keys for IAM Users](#).

To view a list of all available CodeCommit commands, run the following command:

```
aws codecommit help
```

To view information about a specific 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 sections to view descriptions and example usage of the commands in the AWS CLI:

- [batch-describe-merge-conflicts \(p. 178\)](#)
- [batch-get-repositories \(p. 122\)](#)
- [create-branch \(p. 218\)](#)
- [create-commit \(p. 187\)](#)
- [create-pull-request \(p. 153\)](#)
- [create-repository \(p. 83\)](#)
- [create-unreferenced-merge-commit \(p. 181\)](#)
- [delete-branch \(p. 227\)](#)
- [delete-comment-content \(p. 207\)](#)
- [delete-file \(p. 147\)](#)
- [delete-repository \(p. 139\)](#)
- [describe-merge-conflicts \(p. 179\)](#)

- [describe-pull-request-events \(p. 156\)](#)
- [get-blob \(p. 194\)](#)
- [get-branch \(p. 223\)](#)
- [get-comment \(p. 207\)](#)
- [get-comments-for-compared-commit \(p. 206\)](#)
- [get-comments-for-pull-request \(p. 162\)](#)
- [get-commit \(p. 192\)](#)
- [get-differences \(p. 193\)](#)
- [get-merge-conflicts \(p. 157\)](#)
- [get-pull-request \(p. 156\)](#)
- [get-repository \(p. 122\)](#)
- [get-repository-triggers \(p. 115\)](#)
- [list-branches \(p. 223\)](#)
- [list-pull-requests \(p. 155\)](#)
- [list-repositories \(p. 121\)](#)
- [list-tags-for-resource \(p. 98\)](#)
- [merge-pull-request-by-fast-forward \(p. 170\)](#)
- [post-comment-for-compared-commit \(p. 208\)](#)
- [post-comment-for-pull-request \(p. 161\)](#)
- [post-comment-reply \(p. 209\)](#)
- [put-file \(p. 144\)](#)
- [put-repository-triggers \(p. 115\)](#)
- [tag-resource \(p. 96\)](#)
- [test-repository-triggers \(p. 117\)](#)
- [untag-resource \(p. 100\)](#)
- [update-comment \(p. 209\)](#)
- [update-default-branch \(p. 226\)](#)
- [update-pull-request-description \(p. 166\)](#)
- [update-pull-request-status \(p. 182\)](#)
- [update-pull-request-title \(p. 166\)](#)
- [update-repository-description \(p. 125\)](#)
- [update-repository-name \(p. 125\)](#)

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 \(p. 328\)](#)
- [Remote Repositories \(p. 328\)](#)
- [Commits \(p. 329\)](#)
- [Branches \(p. 330\)](#)
- [Tags \(p. 331\)](#)

Configuration Variables

Lists all configuration variables.	<code>git config --list</code>
Lists only local configuration variables.	<code>git config --local -l</code>
Lists only system configuration variables.	<code>git config --system -l</code>
Lists only global configuration variables.	<code>git config --global -l</code>
Sets a configuration variable in the specified configuration file.	<code>git config [--local --global --system] <i>variable-name</i> <i>variable-value</i></code>
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.	<code>git config [--local --global --system] --edit</code>

Remote Repositories

Initializes a local repo in preparation for connecting it to an CodeCommit repository.	<code>git init</code>
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.	<code>git remote add <i>remote-name</i> <i>remote-url</i></code>
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 repository and creates and checks out an initial branch that is forked from the current default branch in the cloned CodeCommit repository.	<code>git clone <i>remote-url</i> <i>local-subfolder-name</i></code>
Shows the nickname the local repo uses for the CodeCommit repository.	<code>git remote</code>
Shows the nickname and the URL the local repo uses for fetches and pushes to the CodeCommit repository.	<code>git remote -v</code>

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.	<code>git push -u <i>remote-name</i> <i>branch-name</i></code>
Pushes finalized commits from the local repo to the CodeCommit repository after upstream tracking information is set.	<code>git push</code>
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	<code>git pull <i>remote-name</i> <i>branch-name</i></code>
Pulls finalized commits to the local repo from the CodeCommit repository after upstream tracking information is set.	<code>git pull</code>
Disconnects the local repo from the CodeCommit repository, using the specified nickname the local repo has for the CodeCommit repository.	<code>git remote rm <i>remote-name</i></code>

Commits

Shows what has or hasn't been added to the pending commit in the local repo.	<code>git status</code>
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)	<code>git status -sb</code>
Shows changes between the pending commit and the latest commit in the local repo.	<code>git diff HEAD</code>
Adds specific files to the pending commit in the local repo.	<code>git add [<i>file-name-1</i> <i>file-name-2</i> <i>file-name-N</i> <i>file-pattern</i>]</code>
Adds all new, modified, and deleted files to the pending commit in the local repo.	<code>git add</code>
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.	<code>git commit</code>
Finalizes the pending commit in the local repo, including specifying a commit message at the same time.	<code>git commit -m "<i>Some meaningful commit comment</i>"</code>
Lists recent commits in the local repo.	<code>git log</code>
Lists recent commits in the local repo in a graph format.	<code>git log --graph</code>

Lists recent commits in the local repo in a predefined condensed format.	<code>git log --pretty=oneline</code>
Lists recent commits in the local repo in a predefined condensed format, with a graph.	<code>git log --graph --pretty=oneline</code>
Lists recent commits in the local repo in a custom format, with a graph. (For more options, see Git Basics - Viewing the Commit History)	<code>git log --graph --pretty=format:"%H (%h) : %cn : %ar : %s"</code>

Branches

Lists all branches in the local repo with an asterisk (*) displayed next to your current branch.	<code>git branch</code>
Pulls information about all existing branches in the CodeCommit repository to the local repo.	<code>git fetch</code>
Lists all branches in the local repo and remote tracking branches in the local repo.	<code>git branch -a</code>
Lists only remote tracking branches in the local repo.	<code>git branch -r</code>
Creates a new branch in the local repo using the specified branch name.	<code>git branch <i>new-branch-name</i></code>
Switches to another branch in the local repo using the specified branch name.	<code>git checkout <i>other-branch-name</i></code>
Creates a new branch in the local repo using the specified branch name, and then switches to it.	<code>git checkout -b <i>new-branch-name</i></code>
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 information for the branch in the local repo during the push.	<code>git push -u <i>remote-name</i> <i>new-branch-name</i></code>
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.	<code>git branch --track <i>new-branch-name</i> <i>remote-name/remote-branch-name</i></code>
Merges changes from another branch in the local repo to the current branch in the local repo.	<code>git merge <i>from-other-branch-name</i></code>
Deletes a branch in the local repo unless it contains work that has not been merged.	<code>git branch -d <i>branch-name</i></code>
Deletes a branch in the CodeCommit repository using the specified nickname the local repo has	<code>git push <i>remote-name</i> :<i>branch-name</i></code>

for the CodeCommit repository and the specified branch name. (Note the use of the colon (:).)

Tags

Lists all tags in the local repo.	<code>git tag</code>
Pulls all tags from the CodeCommit repository to the local repo.	<code>git fetch --tags</code>
Shows information about a specific tag in the local repo.	<code>git show <i>tag-name</i></code>
Creates a "lightweight" tag in the local repo.	<code>git tag <i>tag-name commit-id-to-point-tag-at</i></code>
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.	<code>git push <i>remote-name tag-name</i></code>
Pushes all tags from the local repo to the CodeCommit repository using the specified nickname the local repo has for the CodeCommit repository.	<code>git push <i>remote-name --tags</i></code>
Deletes a tag in the local repo.	<code>git tag -d <i>tag-name</i></code>
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 (:).)	<code>git push <i>remote-name :tag-name</i></code>

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
- **Latest documentation update:** July 31, 2019

update-history-change	update-history-description	update-history-date
CodeCommit is available in EU (Stockholm) (p. 332)	You can now use CodeCommit in EU (Stockholm). For more information, including Git connection endpoints, see Regions .	July 31, 2019
CodeCommit adds support for tagging repositories in the CodeCommit console (p. 332)	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 (p. 332)	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) (p. 332)	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 (p. 332)	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 (p. 332)	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) (p. 332)	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 (p. 332)	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 (p. 332)	CodeCommit has added an API for creating commits. For more information, see Create a Commit .	February 20, 2019
Content update (p. 332)	The content in this guide has been updated with minor fixes and additional troubleshooting guidance.	January 2, 2019
Content update (p. 332)	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) (p. 332)	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 (p. 332)	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 (p. 332)	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 (p. 219) topic has been added. The CodeCommit Permissions Reference (p. 291) topic has been updated.	May 16, 2018
New section	The Working with Files in AWS CodeCommit Repositories (p. 140) section has been added. The CodeCommit Permissions Reference (p. 291) and Getting Started with AWS CodeCommit Tutorial (p. 47) topics have been updated.	February 21, 2018
New topic	The Configure Cross-Account Access to an AWS CodeCommit Repository (p. 129) topic has been added.	February 21, 2018
New topic	The Integrate AWS Cloud9 with AWS CodeCommit (p. 15) topic has been added. The Product and Service Integrations (p. 73) topic has been updated with information about AWS Cloud9.	December 1, 2017
New section	The Working with Pull Requests in AWS CodeCommit Repositories (p. 149) section has been added. The Authentication and Access Control for AWS CodeCommit (p. 270) 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 (p. 73) 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 (p. 270) section has been added. It replaces the Access Permissions Reference topic.	September 11, 2017
Updated topics	The Manage Triggers for a Repository (p. 101) 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 (p. 229) topic has been added. The View Tag Details (p. 211) topic has been updated. The Product and Service Integrations (p. 73) topics has been updated with information about integrating with Amazon CloudWatch Events.	August 3, 2017
New topics	The Integrate Eclipse with AWS CodeCommit (p. 22) and Integrate Visual Studio with AWS CodeCommit (p. 18) 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 (p. 304) topic has been updated.	June 29, 2017

Change	Description	Date Changed
Updated topic	CodeCommit is now available in four additional regions: Asia Pacific (Seoul), South America (São Paulo), US West (N. California), and EU (London). The Regions and Git Connection Endpoints (p. 304) 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 EU (Frankfurt). The Regions and Git Connection Endpoints (p. 304) topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	May 25, 2017
New topic	The Compare Branches (p. 224) topic has been added. The contents of the Working with Branches (p. 215) 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 (p. 196) topic has been added with information about comparing commits. The structure of the user guide has been updated for working with repositories (p. 81) , commits, (p. 184) , and branches (p. 215) .	March 28, 2017
Updated topic	The View Commit Details (p. 189) 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 (p. 319) 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 (p. 8) 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 (p. 73) topic has been updated to include information about integration with AWS CodeBuild.	December 5, 2016
Updated topic	CodeCommit is now available in another region, EU (Ireland). The Regions and Git Connection Endpoints (p. 304) topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	November 16, 2016
Updated topic	CodeCommit is now available in another region, US West (Oregon). The Regions and Git Connection Endpoints (p. 304) topic has been updated to provide information about Git connection endpoints and supported regions for CodeCommit.	November 14, 2016

Change	Description	Date Changed
New topic	The Create a Trigger for a Lambda Function (p. 107) 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 (p. 110) 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 (p. 304) 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 (p. 73) topic has been updated to include information about integration with AWS Elastic Beanstalk.	October 13, 2016
Topic update	The Product and Service Integrations (p. 73) topic has been updated to include information about integration with AWS CloudFormation.	October 6, 2016
Topic update	The For SSH Connections on Windows (p. 32) 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
Topic update	The View Commit Details (p. 189) and CodeCommit Tutorial (p. 47) topics have been updated to include information about the Commit Visualizer in the CodeCommit console. The Limits (p. 312) 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 (p. 189) and CodeCommit Tutorial (p. 47) 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 (p. 230) and Migrate Local or Unversioned Content to AWS CodeCommit (p. 238) topics have been added.	June 29, 2016
Topic update	Minor updates have been made to the Troubleshooting (p. 253) and For HTTPS Connections on Windows with the AWS CLI Credential Helper (p. 41) topics.	June 22, 2016
Topic update	The Product and Service Integrations (p. 73) 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 (p. 101) section has been added. New topics include examples, including policy and code samples, of how to create, edit, and delete triggers.	March 7, 2016

Change	Description	Date Changed
New topic	The Product and Service Integrations (p. 73) topic has been added. Minor updates have been made to Troubleshooting (p. 253) .	March 7, 2016
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 (p. 28) and For SSH Connections on Windows (p. 32) .	December 9, 2015
New topic	The Browse Files in a Repository (p. 141) topic has been added. New issues have been added to Troubleshooting (p. 253) . 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 (p. 26) topic has been added. The topics in the Setting Up (p. 6) 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 (p. 29) and SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit (p. 34) .	July 24, 2015
Topic update	Steps in SSH and Windows: Set Up the Public and Private Keys for Git and CodeCommit (p. 34) have been updated to address an issue with IAM and saving the public key file.	July 22, 2015
Topic update	Troubleshooting (p. 253) 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 (p. 318) and the Access Permissions Reference topics.	July 17, 2015
Topic update	Another section has been added to Troubleshooting (p. 253) with information about troubleshooting issues with AWS Key Management Service.	July 10, 2015
Initial release	This is the initial release of the <i>CodeCommit User Guide</i> .	July 9, 2015

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.