

CodeArts Repo

User Guide

Issue 01
Date 2023-09-05



Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: <https://www.huawei.com>

Email: support@huawei.com

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to "Vul. Response Process". For details about the policy, see the following website:<https://www.huawei.com/en/psirt/vul-response-process>
For enterprise customers who need to obtain vulnerability information, visit:<https://securitybulletin.huawei.com/enterprise/en/security-advisory>

Contents

1 Overview.....	1
2 Git Installation and Configuration.....	5
2.1 Installing and Configuring Git.....	5
2.2 Installing Git Bash for Windows.....	5
2.3 Installing TortoiseGit for Windows.....	6
2.4 Installing Git for Linux.....	8
2.5 Installing Git for macOS.....	8
3 Setting SSH Key or HTTPS Password for CodeArts Repo Repository.....	9
3.1 Overview.....	9
3.2 SSH Keys.....	10
3.3 HTTPS Password.....	13
4 Migrating Data to CodeArts Repo.....	16
4.1 Overview.....	16
4.2 Migrating an SVN Repository to CodeArts Repo.....	16
4.3 Importing a Remote Git Repository to CodeArts Repo.....	20
4.4 Uploading Local Code to CodeArts Repo.....	23
5 Creating a CodeArts Repo Repository.....	25
5.1 Overview.....	25
5.2 Creating an Empty Repository.....	26
5.3 Creating a Repository Using a Template.....	29
5.4 Importing an External Repository.....	31
5.5 Forking a Repository.....	33
6 Associating the CodeArts Repo Repository.....	37
7 Cloning or Downloading Code from CodeArts Repo to a Local PC.....	40
7.1 Overview.....	40
7.2 Using SSH to Clone Code from CodeArts Repo to a Local PC.....	40
7.3 Using HTTPS to Clone Code from CodeArts Repo to a Local Computer.....	44
7.4 Downloading a Code Package on a Browser.....	48
8 Using CodeArts Repo.....	49
8.1 Viewing the Repository List.....	49

8.2 Viewing Repository Details.....	50
8.3 Viewing Repository Homepage.....	52
8.4 Managing Code Files.....	54
8.4.1 Managing Files.....	54
8.4.2 Managing Commits.....	59
8.4.3 Managing Branches.....	59
8.4.4 Managing Tags.....	70
8.4.5 Managing Comparison.....	76
8.5 Managing MRs.....	77
8.5.1 Managing MRs.....	77
8.5.2 Resolving Code Conflicts in an MR.....	84
8.5.3 Detailed Description of Review Comments Gate.....	91
8.5.4 Detailed Description of Pipeline Gate.....	92
8.5.5 Detailed Description of E2E Ticket Number Association Gate.....	93
8.5.6 Detailed Description of Review Gate.....	94
8.5.7 Detailed Description of Approval Gate.....	96
8.6 Viewing Review Records of a Repository.....	97
8.7 Viewing Associated Work Items.....	99
8.7.1 Introduction.....	99
8.7.2 Commit Association.....	102
8.8 Viewing Repository Statistics.....	106
8.9 Viewing Activities.....	107
8.10 Managing Repository Members.....	107
8.10.1 IAM Users, Project Members, and Repository Members.....	108
8.10.2 Configuring Member Management.....	108
8.10.3 Repository Member Permissions.....	111
9 Configuring CodeArts Repo.....	121
9.1 General Settings.....	121
9.1.1 Repository Information.....	121
9.1.2 Notifications.....	122
9.2 Repository Management.....	124
9.2.1 Repositories.....	124
9.2.2 Space Freeing.....	127
9.2.3 Synchronization.....	127
9.2.4 Submodules.....	128
9.2.5 Repository Backup.....	131
9.3 Policy Settings.....	132
9.3.1 Protected Branches.....	132
9.3.2 Protected Tags.....	133
9.3.3 Commit Rules.....	134
9.3.4 Merge Requests.....	139
9.4 Service Integration.....	144

9.4.1 E2E Settings.....	144
9.4.2 Webhooks.....	149
9.5 Security Management.....	151
9.5.1 Deploy Keys.....	151
9.5.2 IP Address Whitelists.....	151
9.5.3 Risky Operations.....	154
9.5.4 Watermarks.....	154
9.5.5 Repository Locking.....	155
9.5.6 Audit Logs.....	156
10 Submitting Code to the CodeArts Repo.....	157
10.1 Creating a Commit.....	157
10.2 Transmitting and Storing a File in Encryption Mode.....	160
10.3 Viewing Commit History.....	170
10.4 Pushing Code to CodeArts Repo Using Eclipse.....	171
11 More About Git.....	183
11.1 Using the Git Client.....	183
11.2 Setting Password-Free Access via HTTPS.....	186
11.3 Using the TortoiseGit Client.....	188
11.4 Use Cases on the Git Client.....	193
11.4.1 Uploading and Downloading Code.....	193
11.4.2 Committing Letter Case Changes in File Names to the Server.....	194
11.4.3 Setting the Line Ending Conversion.....	194
11.4.4 Committing Hidden Files.....	195
11.4.5 Pushing a File That Has Been Changed on the Server.....	195
11.5 Common Git Commands.....	196
11.6 Using Git LFS.....	202
11.7 Git Workflows.....	204
11.7.1 Overview.....	204
11.7.2 Centralized Workflow.....	204
11.7.3 Branch Development Workflow.....	205
11.7.4 GitFlow.....	206
11.7.5 Forking Workflow.....	208

1 Overview

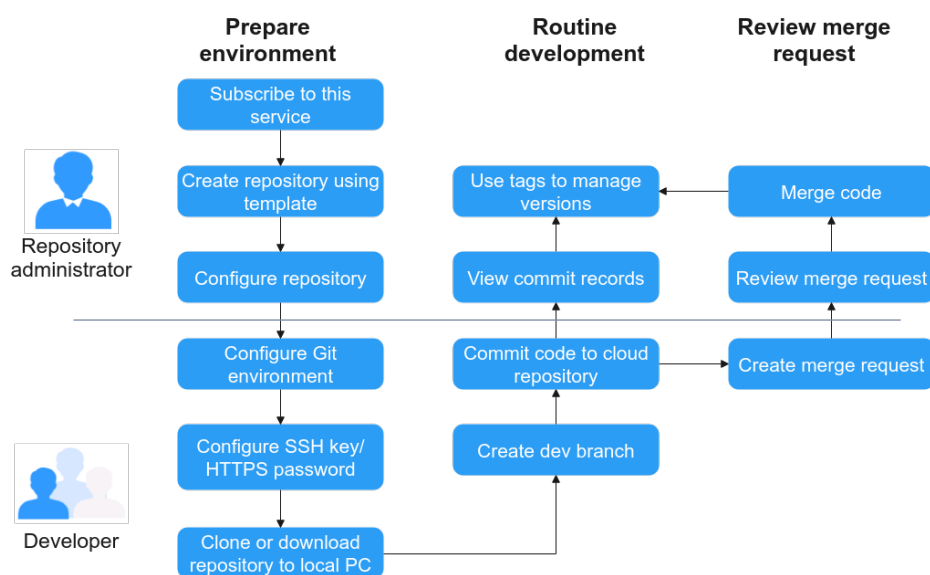
CodeArts Repo is a distributed version management platform that uses the Git workflow. It provides functions such as security management, member and permission management, branch protection and merge, online editing, and statistical analysis. The service aims to address issues such as cross-distance collaboration, multi-branch concurrent development, code version management, and security.

To start a new project, you can use CodeArts Repo built-in repository templates to create a repository for development. For details, see [Starting R&D Projects in CodeArts Repo](#).

If you are developing a project locally and want to use CodeArts Repo to manage versions, you can migrate the project to CodeArts Repo. For details, see [Migrating a Local Project to CodeArts Repo](#).

Starting R&D Projects in CodeArts Repo

You can use repository templates provided by CodeArts Repo to create a project and start development. The following figure shows the workflow.

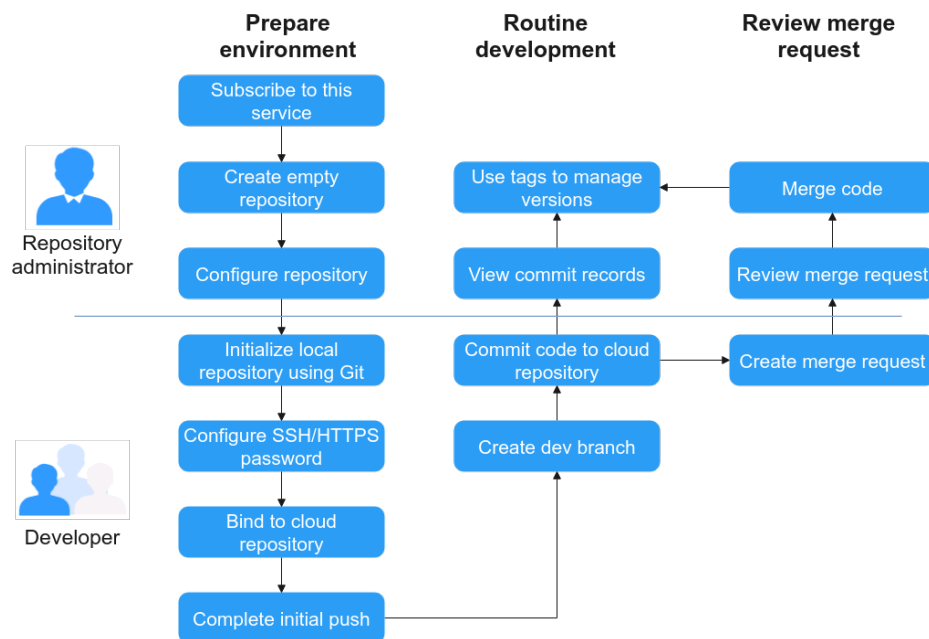


The operations involved are as follows:

- [5.3 Creating a Repository Using a Template](#)
- [8.10.2 Configuring Member Management](#)
- [9 Configuring CodeArts Repo](#)
- [2 Git Installation and Configuration](#)
- [7 Cloning or Downloading Code from CodeArts Repo to a Local PC](#)
- [8.4.3 Managing Branches](#)
- [8.4.4 Managing Tags](#)
- [10 Submitting Code to the CodeArts Repo](#)
- [8.5.1 Managing MRs](#)
- [5.5 Forking a Repository](#)

Migrating a Local Project to CodeArts Repo

To manage code versions of a locally developed project using CodeArts Repo, you can bind the local repository to CodeArts Repo and complete initial push. Then, you can continue developing your project in the distributed version management mode. The following figure shows the workflow.



The operations involved are as follows:

- [5.2 Creating an Empty Repository](#)
- [8.10.2 Configuring Member Management](#)
- [9 Configuring CodeArts Repo](#)
- [2 Git Installation and Configuration](#)
- [6 Associating the CodeArts Repo Repository](#)
- [7 Cloning or Downloading Code from CodeArts Repo to a Local PC](#)
- [8.4.3 Managing Branches](#)
- [8.4.4 Managing Tags](#)

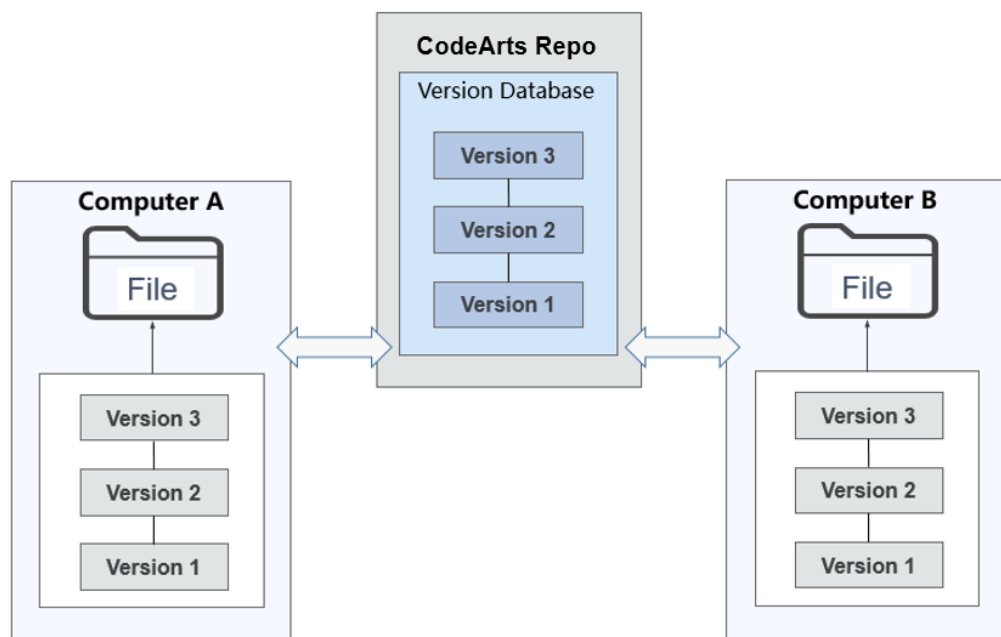
- [10 Submitting Code to the CodeArts Repo](#)
- [8.5.1 Managing MRs](#)
- [5.5 Forking a Repository](#)

Distributed Version Management

There is a complete code repository on your local computer and in CodeArts Repo respectively.

All version information can be synchronized to the local computer for viewing.

You can commit code offline on the local computer and push the code to the CodeArts Repo repository when the network is connected.

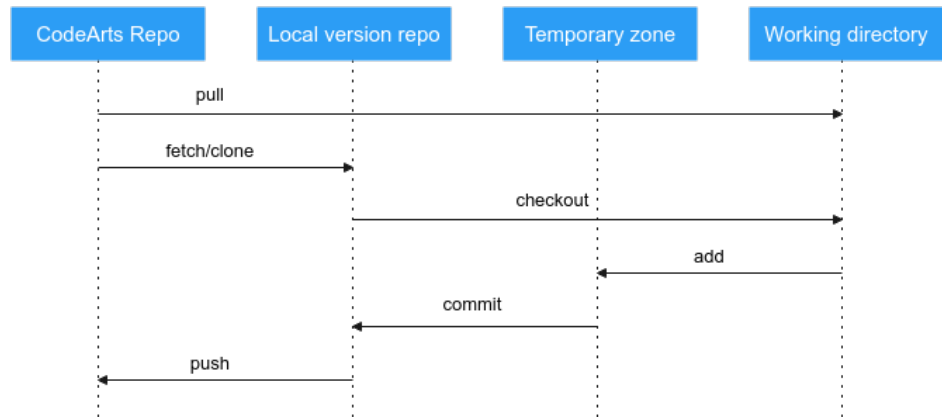


Basic Workflow

CodeArts Repo is a cloud repository service that uses the Git workflow.

- Data in a Git local repository can be in one of the three statuses: modified, staged, and committed. The file you modified in the repository is in the modified state. You can run the **add** command to add the changes to the local staging area. Then, the file is in the staged state. Run the **commit** command to commit the changes to the local repository for management. The corresponding version and version number are generated upon each commit. You can switch and roll back a version based on the version number. A version can have multiple branches and tags. Each branch, tag, or commit is an independent version that can be checked out using the **checkout** command.
- As a cloud repository service, CodeArts Repo not only has the basic features of local Git repositories, but also serves as the remote repository of each local repository and provides configurable security policies and authentication.
- A CodeArts Repo cloud repository interacts with a Git repository in the following scenarios:

- **clone**: clones the branch in CodeArts Repo to the local computer as a local repository.
- **push**: pushes changes in the local repository to CodeArts Repo.
- **fetch**: fetches a version from CodeArts Repo to the working directory.
- **pull**: fetches a version from CodeArts Repo to the working directory and tries to merge it into the current branch. If the operation fails, you need to manually resolve the file conflict.



2 Git Installation and Configuration

- [2.1 Installing and Configuring Git](#)
- [2.2 Installing Git Bash for Windows](#)
- [2.3 Installing TortoiseGit for Windows](#)
- [2.4 Installing Git for Linux](#)
- [2.5 Installing Git for macOS](#)

2.1 Installing and Configuring Git

CodeArts Repo is a Git-based service. Git clients such as Git Bash or TortoiseGit must be installed on local computers to connect to CodeArts Repo. The following sections describe how to install and configure Git Bash and TortoiseGit on Windows, Linux, and macOS.

If you have installed Git and configured the signature and email address, skip the following sections:

- [2.2 Installing Git Bash for Windows](#)
- [2.3 Installing TortoiseGit for Windows](#)
- [2.4 Installing Git for Linux](#)
- [2.5 Installing Git for macOS](#)

NOTE

GitHub Desktop is not supported in CodeArts Repo.

2.2 Installing Git Bash for Windows

Git Bash is a simple and efficient client on Windows for users who are familiar with Git commands. If you are unfamiliar with Git commands, you can use TortoiseGit by referring to [2.3 Installing TortoiseGit for Windows](#).

1. Install the Git Bash client.
 - a. Go to the [Git Bash website](#) and download the installation package for 32-bit or 64-bit Windows.

- b. Double-click the installation package. In the installation window displayed, click **Next** for several times and then click **Install**.
2. Open the Git Bash client.
3. Configure the Git Bash client.

Click the Windows start icon, enter **Git Bash** in the search box, and press **Enter** to open Git Bash. You are advised to pin Git Bash to the Windows taskbar.

Enter the following commands in Git Bash to configure your username and email address:

```
git config --global user.name your_username  
git config --global user.email your_email_address
```

Run the following command to view the configurations:

```
git config -l
```

NOTE

- A username can contain letters, digits, and special characters. You are advised to set the same username as that in CodeArts Repo.
- The email address should be written in the standard format.
- The **--global** parameter in the commands indicates that the configurations apply to all Git repositories on your computer. However, you can set a different username and email address for a specific repository.

2.3 Installing TortoiseGit for Windows

TortoiseGit is a better choice if you are not familiar with Git commands or you hope to migrate code from an SVN client such as TortoiseSVN. TortoiseGit is a Windows shell interface to Git as TortoiseSVN to SVN.

Prerequisites

1. Go to the [TortoiseGit website](#) and download the installation package for 32-bit or 64-bit Windows.
2. Double-click the installation package. In the window displayed, click **Next** for several times and then click **Install** to complete the installation. Click **Finish** to run the tool.
3. In the first start wizard displayed, select a language, enter a Git.exe path (the field is automatically filled with an available path if there is any), and configure a username and email address. Keep the default values and click **Next** till the settings are finished.

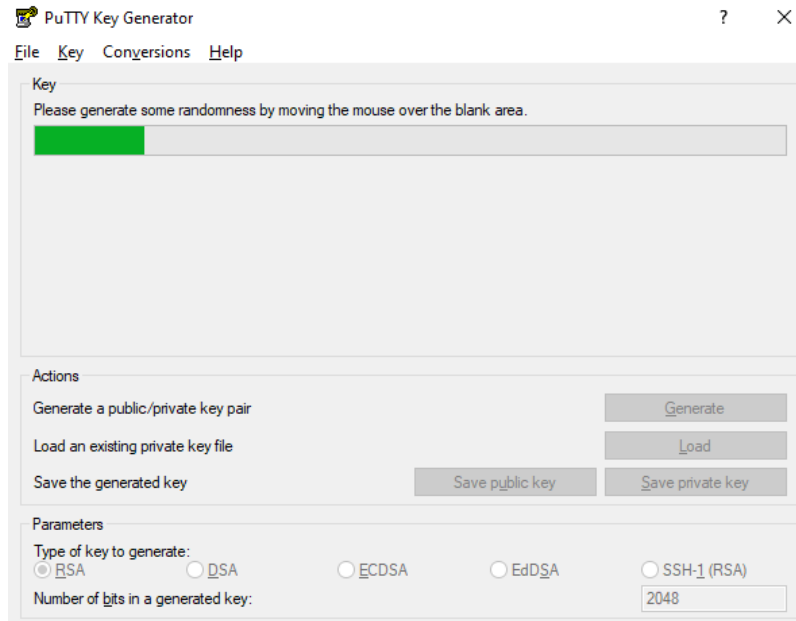
(Optional) Localization

TortoiseGit is installed in English by default. If you want to use a translated version of TortoiseGit, go to the [TortoiseGit website](#) to download your desired language pack.

Configurations

TortoiseGit also requires a key pair for authentication with the CodeArts Repo server. To generate a key pair, perform the following steps:

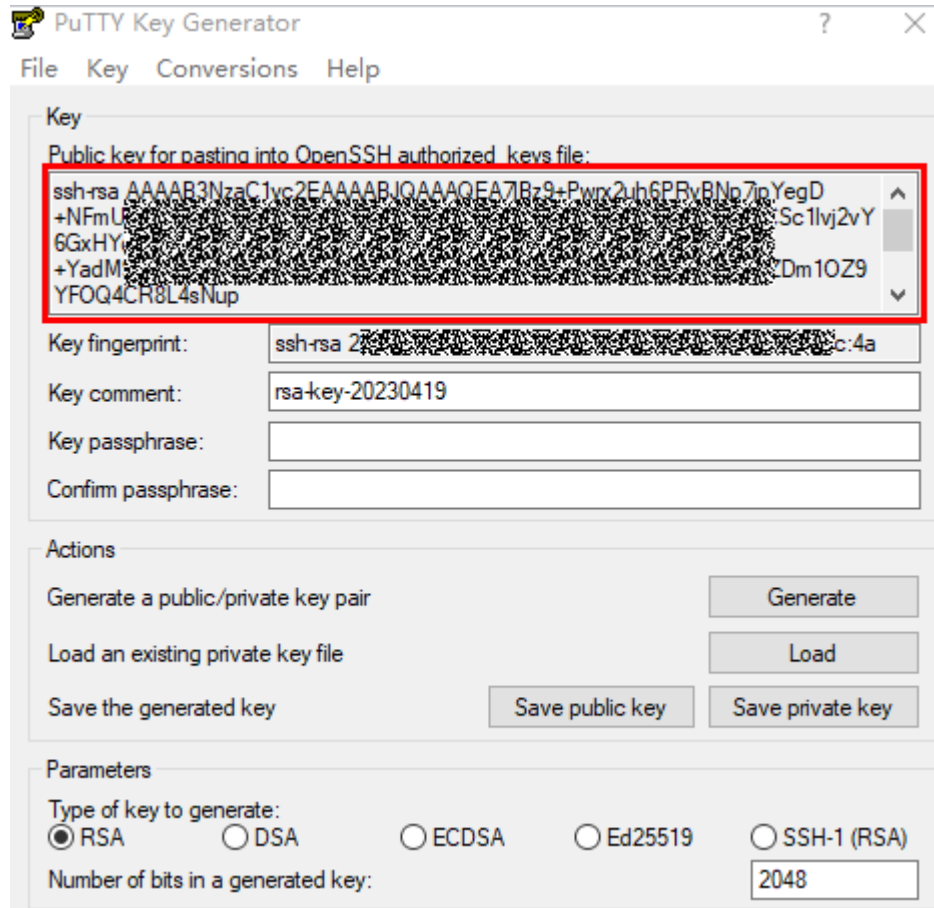
1. Search for PuTTYgen and open it. In the displayed window, click **Generate** to generate a key pair.



 **NOTE**

PuTTYgen is a powerful, compact, and easy-to-use tool for generating pairs of public and private keys. It is installed along with the TortoiseGit installation and does not conflict with the one built in PuTTY.

2. After the key pair is generated, store the public and private keys.
 - Click **Save private key**. In the dialog box that is displayed, enter a file name and save the private key file.
 - Click **Save public key**. In the dialog box that is displayed, enter a file name and save the public key file.
3. Copy the public key in the red box in the following figure and **bind it to CodeArts Repo**.



4. Bind the private key to the local client.

Search for Pageant and open it. In the displayed window, click **Add Key**, and select the generated private key file.

2.4 Installing Git for Linux

- Debian or Ubuntu
Run the following command in the terminal:

```
apt-get install git
```
- Fedora, CentOS, or Red Hat
Run the following command in the terminal:

```
yum install git
```
- For more OSs, see the Git official website.

2.5 Installing Git for macOS

- You can quickly install Git on macOS by installing Xcode command line tools.
- On Mavericks 10.9 or a later version, run the **git** command on the Terminal. The system will prompt you to install the command line tools if you have not.
- If you want to install Git of a later version, go to the Git website and download the latest version for macOS.

3 Setting SSH Key or HTTPS Password for CodeArts Repo Repository

[3.1 Overview](#)

[3.2 SSH Keys](#)

[3.3 HTTPS Password](#)

3.1 Overview

What Is an SSH Key and HTTPS Password?

When you push code to or pull code from CodeArts Repo repository, the repository needs to verify your identity and permissions. SSH and HTTPS are two authentication modes for remote access to CodeArts Repo.

- **3.2 SSH Keys:** An SSH key is used to establish a secure connection between your local computer and CodeArts Repo under your account.

Before connecting to CodeArts Repo in SSH mode, generate an SSH key on your computer and configure it in CodeArts Repo.

After you configure an SSH key on a local computer and add the public key to CodeArts Repo, you can use the SSH key to access all code repositories under your account from your computer.

- **3.3 HTTPS Password:** An HTTPS password is a user credential used for pulling and pushing code using the HTTPS protocol.

The maximum size of a package that can be pushed at a time using HTTPS is 200 MB. If the size is greater than 200 MB, use the SSH mode.

Federated users cannot be bound to email addresses and do not support the HTTPS protocol.

NOTE

Either SSH or HTTPS can be used to push or pull code. Set SSH keys or HTTPS passwords as required.

3.2 SSH Keys

Introduction

When you push code to or pull code from CodeArts Repo, the repository needs to verify your identity and permissions. SSH is an authentication mode for remote access to CodeArts Repo.

- An SSH key is an encrypted network transmission protocol that establishes a secure connection between your computer and CodeArts Repo under your account.
- After you configure an SSH key on a local computer and add the public key to CodeArts Repo, you can use the SSH key to access all code repositories under your account from your computer.
- Before connecting to CodeArts Repo in SSH mode, generate an SSH key on your computer and configure it in CodeArts Repo.

Generating and Configuring an SSH Key

The following procedure describes how to generate a public key and bind it.

Step 1 Install the Git Bash client by referring to [2.2 Installing Git Bash for Windows](#).

Step 2 Check whether your computer has generated a key.

Run the following command on the local Git client:

```
cat ~/.ssh/id_rsa.pub
```

- If **No such file or directory** is displayed, no SSH key has been generated on the computer. Go to [Step 3](#) to generate and configure an SSH key.

```
DL0373 MINGW64 /d/gitTest
$ cat ~/.ssh/id_rsa.pub
cat: /c/Users/lwx/.ssh/id_rsa.pub: No such file or directory
```

- If at least one group of keys is returned, an SSH key has been generated on your computer. To use the generated key, go to [Step 4](#) directly. To generate a new key, go to [Step 3](#).

```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/...
(master)
$ cat ~/.ssh/id_rsa.pub
ssh-rsa A... EAAAADAQABAAQ... HI5f//Xxe/ESu8j6DoyE... EJ
j4w509eCP... OuSSRmJz/+rpp... 6rdvqD+aEXImVMeQGUIL... g3
d4TJkJBRI... JQF3hJ2kn5OMQ... 7JKPuBSpJrbz0vpX4wab... hP
51liJifyh... yRpRX+YL5DzqU... 4BaJyX+5E0Jd8yL6MFFc... ln
LlXspkHYw... 07/z/k7055nDq... JuEdgHKnz9xGUQ3tc66z... a+
mz0ym1CZw... mi z9GNI BrLN2C... yhNqvzSt1LgmYTYwSGbV... JZ
+yL4nzVFC... rsPFC96nNaqBx... g/nimvjobaDHCj8ijL67... @y
ibaijin.com
```

Step 3 Generate an SSH private key.

Run the following command on the local Git client to generate a new SSH key:

```
ssh-keygen -t rsa -C "Your SSH key comment"
```



```
Administrator@ecstest-paas-1wx MINGW64 ~/.ssh
$ ssh-keygen -t rsa -C "XXXXXXXXXX"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): ①
/c/Users/Administrator/.ssh/id_rsa already exists.
Overwrite (y/n)? y ②
Enter passphrase (empty for no passphrase): ③
Enter same passphrase again:
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:NoSGrzQ6mHGUUUaNGwTd4a97GkC2gH+PJoUTudJHosM XXXXXXXXXXXX
The key's randomart image is:
+----[RSA 3072]-----+
  .+B* . . .
   =o.o. .
  + * + .
  o + % . .
   . E S % .
  = = @ * .
  o o o + +
   . o o .
   . . .
+----[SHA256]-----+
Administrator@ecstest-paas-1wx MINGW64 ~/.ssh
```

Perform the following operations. If information similar to the preceding figure is displayed, the key is generated.

1. The system prompts you to enter the storage path of the key. You can press **Enter** to use the default path.
2. If a key already exists in the local path, the system asks you whether to overwrite it. Enter **n** to cancel key generation, or enter **y** and press **Enter** to overwrite the existing key. In this example, the existing key is overwritten.
3. The system prompts you to set a password for the key and confirm the password. If you do not want to set a password, press **Enter**.

NOTICE

- If a password is set (recommended), the generated private key file is stored after being encrypted by AES-128-CBC.
- If you press **Enter** without entering the password, the generated private key file **id_rsa** is stored locally in plaintext. Keep it secure.


Step 4 Copy the SSH public key to the clipboard.

Run the following command based on your operating system to copy the SSH public key to your clipboard. Take Windows as an example. If no command output is displayed, the public key is copied.

- **Windows**
clip < ~/.ssh/id_rsa.pub
- **macOS**
pbcopy < ~/.ssh/id_rsa.pub
- **Linux (xclip required)**
xclip -sel clip < ~/.ssh/id_rsa.pub

Step 5 Log in to the CodeArts Repo service repository list page, click the alias in the upper right corner, and choose **This Account Settings > SSH Keys**.



Alternatively, log in to the repository list page of CodeArts Repo and click the  **Set SSH Key** icon to go to the **SSH Key** page.

Step 6 On the **SSH Keys** page, click **Add SSH Key**. The **Add SSH Key** page is displayed.

Add SSH Key
For details about how to generate an SSH key, see the guidance below.

- Key Name

- Key

You can add 5000 more characters.

I have read and agree to the [Privacy Statement](#) and [CodeArts Service Statement](#).

Step 7 Enter a key name, paste the SSH public key copied in [Step 4](#) to the **Key** text box, select **I have read and agree to the Privacy Statement and CodeArts Service Statement**, and click **OK**. A message is displayed, indicating that the operation is successful.

NOTE

- An SSH key cannot be added repeatedly. If an SSH key fails to be added, check whether it has already been added or whether there are redundant spaces in the key.
- After the key is added, you can view it on the **SSH Keys** page. If it is no longer used, you can delete it.
- The difference between an SSH key and repository deploy key is that the former is associated with a user/computer and the latter is associated with a repository. The SSH key has the read and write permissions on the repository, and the deploy key has the read-only permission on the repository.

----End

Verifying Whether an SSH Key Is Bound

When an SSH key is bound, you can perform **SSH-clone** on the repository that you have the access permission on the client. If the clone is successful, the key is bound.

NOTE

If you use SSH to clone a repository to the local computer for the first time, the message "The authenticity of host *.*.com can't be established. RSA key... (yes/no)?" is displayed. Enter **yes** to continue.

3.3 HTTPS Password

Introduction

When you push code to or pull code from CodeArts Repo, the repository needs to verify your identity and permissions. HTTPS is an authentication mode for remote access to CodeArts Repo.

- **HTTPS username**

The value can be the tenant name or IAM username. Enter the complete username. If you want to add the username to the URL, escape '/' to '%2F'.

NOTE

When setting the HTTPS password for the account (the account name is the same as the username), you can enter only the account name.

- **HTTPS password**

- Enter a password containing 8 to 32 characters. The password must contain at least three types of digits, uppercase letters, lowercase letters, and special characters. It cannot be the same as the username or the username spelled backwards.
- An HTTPS password is a user credential used for pulling or pushing code using the HTTPS protocol. Each developer needs to set a password only once and can use it for all repositories.
- Keep your HTTPS password secure and change it periodically to avoid security risks. If you forget the password, set a new HTTPS password.

NOTE


By default, the HTTPS password is the Huawei Cloud login password. The password can be synchronized in real time. You can also select **Set new password** to change the password.

Changing the HTTPS Password

You need to set the initial password upon the first login. You can also change the HTTPS password at any time. The procedure is as follows:

- Step 1** Log in to the CodeArts Repo service repository list page, click the alias in the upper right corner, and choose **This Account Settings > HTTPS Password**. The page is displayed.



Alternatively, log in to the repository list page of CodeArts Repo and click the  Set HTTPS Password icon to go to the **SSH Key** page.

- Step 2** Choose **Set new password** to reset the password. (If you have set an HTTPS password and are using it, click **Change**.)

HTTPS Password

Use Huawei Cloud login password Set new password

Username



Password

- Step 3** Enter the new password and email verification code, select **I have read and agree to the Privacy Statement and CodeArtsService Statement**, and click **OK**. A message is displayed, indicating that the operation is successful.
- Step 4** After the password is reset, you need to regenerate the repository credential locally and check the **IP address whitelist**. Otherwise, you cannot interact with the CodeArts Repo repository.

Delete the local credential (for example, on Windows, choose **Control Panel > User Accounts > Manage Windows Credentials > Generic Credentials**), use HTTPS to clone the cloud repository again, and enter the correct account and password in the dialog box that is displayed.

 **NOTE**

If **SSL certificate problem** is displayed, run the following command on Git client:

```
git config --global http.sslVerify false
```

----End

 **NOTE**

- You can click **Use Huawei Cloud Account Password** to reset the password and customize a password at any time.
- The maximum size of a package that can be pushed at a time using HTTPS is 200 MB. If the size is greater than 200 MB, use the SSH mode.

Verifying Whether an HTTPS Password Takes Effect

After setting an HTTPS password, you can perform **HTTPS-clone** on the repository that you have the access permission on the client. A dialog box is displayed, asking you to enter the account and password. If the clone is successful, the password is configured.

 **NOTE**

You can also use the HTTPS protocol to set password-free code submission. For details, see [Setting Password-Free Access via HTTPS](#)

4 Migrating Data to CodeArts Repo

[4.1 Overview](#)

[4.2 Migrating an SVN Repository to CodeArts Repo](#)

[4.3 Importing a Remote Git Repository to CodeArts Repo](#)

[4.4 Uploading Local Code to CodeArts Repo](#)

4.1 Overview

This section describes how to migrate your repository to CodeArts Repo. Select one of the following migration solutions based on your repository storage mode:

- [Migrating an SVN Repository to CodeArts Repo](#)
- [4.3 Importing a Remote Git Repository to CodeArts Repo](#)
- [4.4 Uploading Local Code to CodeArts Repo](#)

4.2 Migrating an SVN Repository to CodeArts Repo

This section uses a code repository with the standard SVN layout as an example to describe how to migrate an existing SVN repository to CodeArts Repo. The following figure shows the directory structure of the repository.

```
|-- .svn
|-- KotlinGallery
    |-- trunk          Main Development Directory
        |-- app
        |-- gradle
        |-- ..
    |-- branches      Branch Development Directory
        |-- rl_hotfix
            |-- app
            |-- gradle
            |-- ...
    |-- tags          Tag Archive Directory(The modification is not allowed)
        |-- rl.0
            |-- app
            |-- gradle
            |-- ...
        |-- rl.1
```

There are two methods of migrating the existing SVN code. Both methods effectively migrate the SVN code and operation records. The differences of the two methods are as follows. In the first method, the branches and tags folders of the SVN repository are mapped to Git branches and tags during the migration. This facilitates subsequent development on CodeArts Repo, but the migration process is complex. The second method is simple because the branch and tag folders of the SVN repository are migrated without mapping, but it is inconvenient for subsequent development. You can select a method as required.

- **Migration Method 1: Import on the Git Bash Client:** applicable to the scenarios where only part of code is stored in the SVN during project development
- **Migration Method 2: Online Import Using HTTP:** applicable to the scenarios where the complete project code is stored in the SVN when the project is complete

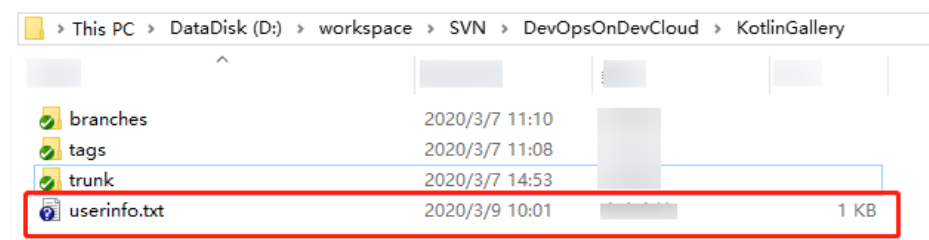
Migration Method 1: Import on the Git Bash Client

Step 1 Obtain committer information of the SVN repository.

1. Use TortoiseSVN to download the repository to be migrated to the local computer.
2. Go to the local SVN repository (**KotlinGallery** in this example) and run the following command on the Git Bash client:

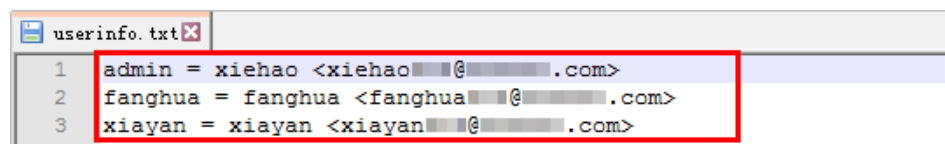
```
svn log --xml | grep "^<author" | sort -u | \awk -F '<author>' '{print $2}' | awk -F '</author>' '{print $1}' > userinfo.txt
```

The **userinfo.txt** file is generated in the directory.



3. Open the **userinfo.txt** file. You can view the information about all committers who have committed code to the repository in the file.
4. Git uses an email address to identify a committer. To better map the SVN repository information to a Git repository, create a mapping between the SVN and Git usernames.

Modify the **userinfo.txt** file. Each line should be in the format of `svn_committer = git_committer_nickname <email_address>`.



Step 2 Create a local Git repository.

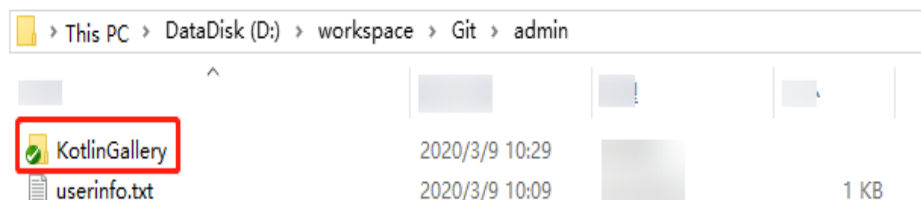
1. **Create an empty Git repository directory on the local computer**, and copy the **userinfo.txt** file obtained in **Step 1** to the directory.
2. Start the Git Bash client in the directory and run the following command to clone a Git repository:

```
git svn clone <svn_repository_address> --no-metadata --authors-file=userinfo.txt --trunk=trunk --tags=tags --branches=branches
```

The following table lists parameters in the command. Set the parameters as required.

Parameter	Description
--no-metadata	Prevents the Git from exporting useless information contained in the SVN.
--authors-file	File that maps all SVN accounts to Git accounts
--trunk	Main development project
--branches	Branch projects
--tags	Tags

After the command is executed, a Git repository is generated locally.



3. Run the following commands to go to the **KotlinGallery** folder and verify the current Git repository branch structure:

```
cd KotlinGallery  
git branch -a
```

```
MINGW64 /d/workspace/Git/admin  
$ cd KotlinGallery/  
  
MINGW64 /d/workspace/Git/admin/KotlinGallery (master)  
$ git branch -a  
* master  
remotes/origin/r1.1_hotfix  
remotes/origin/tags/r1.0  
remotes/origin/tags/r1.1  
remotes/origin/trunk
```

As shown in the preceding figure, all SVN directory structures are successfully migrated in the form of Git branches.

Step 3 Correct local branches.

In **Step 2**, the **git svn clone** command is used to save the **tags** folder in the SVN repository as a branch, which does not comply with the Git usage specifications. Therefore, before uploading tags to CodeArts Repo, adjust the local branches to comply with the Git usage specifications.

1. Go to the local Git repository and run the following commands on the Git Bash client to change the tags branch to appropriate Git tags:

```
cp -Rf .git/refs/remotes/origin/tags/* .git/refs/tags/  
rm -Rf .git/refs/remotes/origin/tags  
git branch -a  
git tag
```



```
MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ cp -Rf .git/refs/remotes/origin/tags/* .git/refs/tags/

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ rm -Rf .git/refs/remotes/origin/tags

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git branch -a
* master
  remotes/origin/r1.1_hotfix
  remotes/origin/trunk

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git tag
r1.0
r1.1
```

2. Run the following commands to change the remaining indexes under **refs/remotes** to local branches:

```
cp -Rf .git/refs/remotes/origin/* .git/refs/heads/
rm -Rf .git/refs/remotes/origin
git branch -a
git tag
```

```
MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ cp -Rf .git/refs/remotes/origin/* .git/refs/heads/

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ rm -Rf .git/refs/remotes/origin

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git branch -a
* master
  r1.1_hotfix
  trunk

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git tag
r1.0
r1.1
```

3. Run the following commands to merge the trunk branch into the master branch and delete the trunk branch:

```
git merge trunk
git branch -d trunk
git branch -a
git tag
```

```
MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git merge trunk
Already up to date.

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git branch -d trunk
Deleted branch trunk (was bccf0d8).

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git branch -a
* master
  r1.1_hotfix

MINGW64 /d/workspace/Git/admin/KotlinGallery (master)
$ git tag
r1.0
r1.1
```

Step 4 Upload the local code.

1. Set the SSH key of the repository by referring to [3.1 Overview](#).

2. Run the following commands to associate the local repository with the CodeArts Repo repository and push the master branch to CodeArts Repo:

```
git remote add origin <CodeArts Repo_repository_address>  
git push --set-upstream origin master
```

After the push is successful, log in to CodeArts Repo and view the master branch of the repository after clicking the **Code** and **Branches** tabs.
3. Run the following command to push other branches from the local computer to CodeArts Repo:

```
git push origin --all
```

After the push is successful, the r1.1_hotfix branch is added to the repository after clicking the **Code** and **Branches** tabs.
4. Run the following command to push tags from the local computer to CodeArts Repo:

```
git push origin --tags
```

After the push is successful, click the **Code** and **Branches** tabs and view tags **r1.0** and **r1.1** added to CodeArts Repo.

----End

Migration Method 2: Online Import Using HTTP

Ensure that your SVN server supports HTTP or HTTPS access. You can enter **http(s)://SVN server address/Name of the repository to be accessed** in any browser for verification.

Step 1 On the CodeArts Repo list page, click  next to **New Repository** and choose **Import Repository** from the drop-down list.

Step 2 Enter the source repository URL, enter the SVN username and password, select **I have read and agree to the *Privacy Statement* and *CodeArts Service Statement***, and click **Next**.

Enter the name of the repository to be created, configure permissions, and click **OK**.

Step 3 After the repository is created, click the repository name to view details.

----End

4.3 Importing a Remote Git Repository to CodeArts Repo


Background

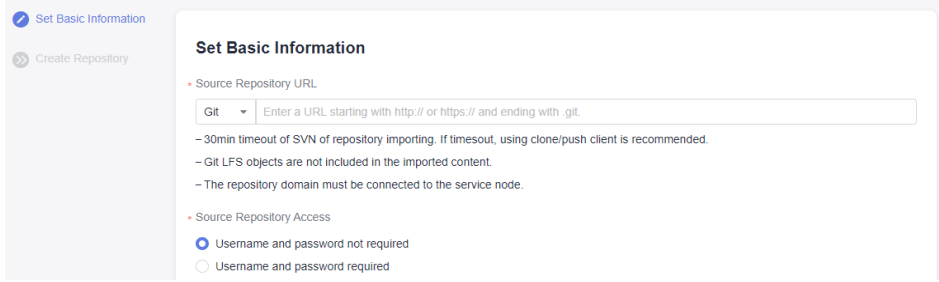
CodeArts Repo allows you to import Git-based remote repositories.

Git-based remote repositories are cloud repositories hosted in storage services such as GitHub.

Method 1: Online Import

You can directly import your remote repository to CodeArts Repo online. The import speed will be affected by network conditions of the source repository.

1. On the CodeArts Repo homepage, click  next to **New Repository** and select **Import Repository** from the drop-down list. The **Import Repository** page is displayed.
2. Enter information in the **Source Repository URL** field. If the source repository is open-source (public repository), select **Username and password not required**. If the source repository is private, select **Username and password required**.
3. Click **Next**. On the **Create Repository** page, enter the basic information about the repository.
4. Click **OK** to import the repository. The repository list page is displayed.



For details, see [5.4 Importing an External Repository](#).


Method 2: Cloning the Git Repository to the Local Computer and Associating and Pushing It to CodeArts Repo

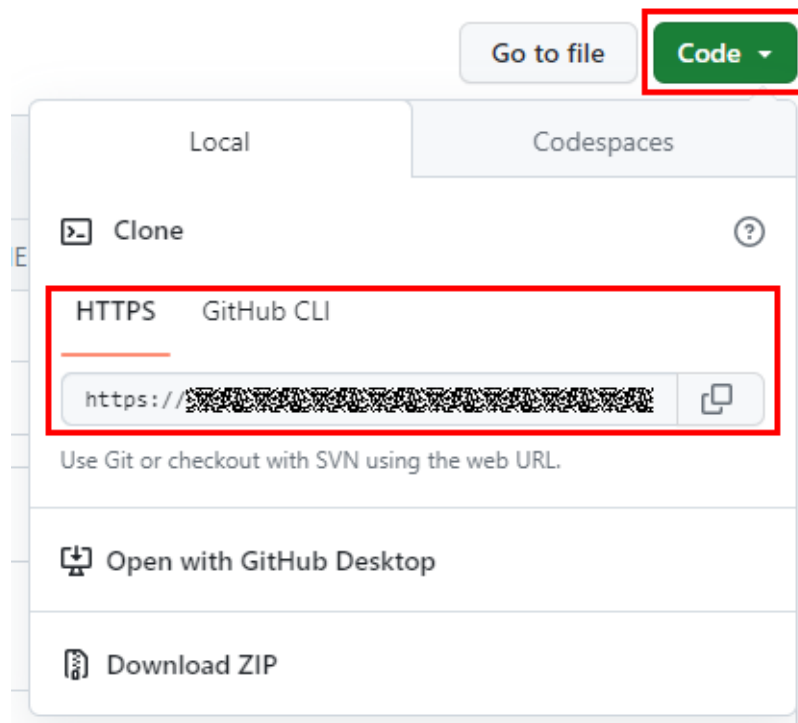
If you cannot [import a repository online](#) due to network issues, use this method. Using this method, you can clone a remote repository to the local computer, and then associate and push it to CodeArts Repo.

Step 1 [Install and configure the Git client](#).

Step 2 Download a bare repository using the source repository address.

The following uses GitHub as an example:


1. Open a browser and enter the address of the GitHub code repository.
2. Click **Code** on the right, click the **HTTPS** tab, and click  on the right.



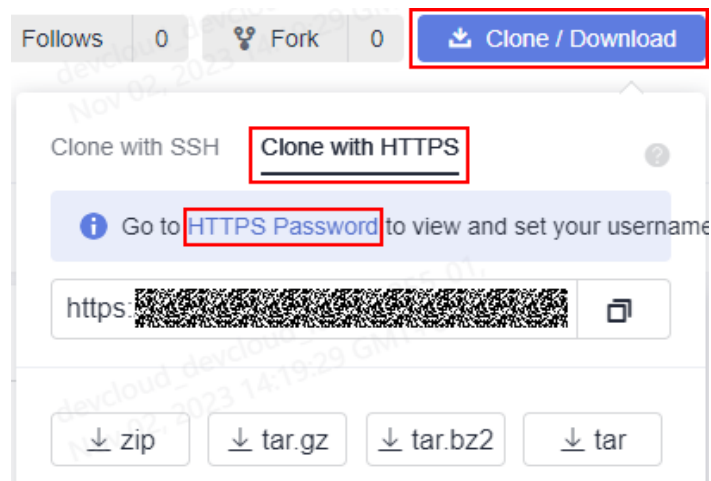
3. Open the Git Bash client on the local PC, run the following command to clone the repository to the local PC, and run the `cd` command to go to the repository directory:

```
git clone --bare <source_repository_address>
```

Step 3 Associate the local repository with CodeArts Repo and push it to CodeArts Repo.

1. On the CodeArts Repo homepage, click **New Repository**. In the **Permissions** area, do not select **Allow generation of a README file**.
2. Go to the repository details page created in 1, click **Clone/Download**, click the **Clone with SSH** or **Clone with HTTPS** tab as required, and click  to obtain the repository address.

In this example, the HTTPS address is used.



3. In the root directory of local source code, open the Git Bash client and run the following command to push the local repository to the new repository:

```
git push --mirror <new_repository_address>
```

When the command is executed, the system prompts you to enter the HTTPS account and password of the CodeArts Repo repository. Enter the correct account and password. (For details about how to obtain an HTTP account and password, see [Changing the HTTPS Password.](#))

```
Administrator@...-test MINGW64 ~/Desktop/GitFile/... .git (BARE:master)
$ git push --mirror https://...
Enumerating objects: 1466, done.
Counting objects: 100% (1466/1466), done.
Delta compression using up to 2 threads
Compressing objects: 100% (1043/1043), done.
Writing objects: 100% (1466/1466), 38.73 MiB | 1.28 MiB/s, done.
Total 1466 (delta 402), reused 1466 (delta 402), pack-reused 0
remote: Resolving deltas: 100% (402/402), done.
To https://...
 * [new branch]      master -> master
Administrator@...-test MINGW64 ~/Desktop/GitFile/... .git (BARE:master)
$ |
```

If your source repository has branches and tags, they will also be pushed to CodeArts Repo.

----End

After the push is successful, check whether the migration is complete in CodeArts Repo. (For details about how to view a CodeArts Repo repository, see [8.1 Viewing the Repository List.](#))

4.4 Uploading Local Code to CodeArts Repo

Background

CodeArts Repo allows you to perform Git initialization on local code and upload the code to a CodeArts Repo repository.

Procedure

Step 1 [Create an empty repository](#) in CodeArts Repo.

- Do not configure **Programming Language of .gitignore**.
- Deselect **Allow generation of a README file**.

Step 2 Prepare the source code to be uploaded on the local computer.

- If the source code is from the SVN server, refer to [Migrating an SVN Repository](#).
- If the source code is not managed by any version control systems, run the following Git command in the root directory of the source code (Git Bash is used as an example):

- a. Initialize a Git repository on the local computer:

```
git init
```

```
Administrator@... MINGW64 ~/Desktop/GIT/task (master)
$ git init
Initialized empty Git repository in C:/Users/.../Desktop/GIT/task/.git/
```

- b. Add the code files to the local repository:

```
git add *
```

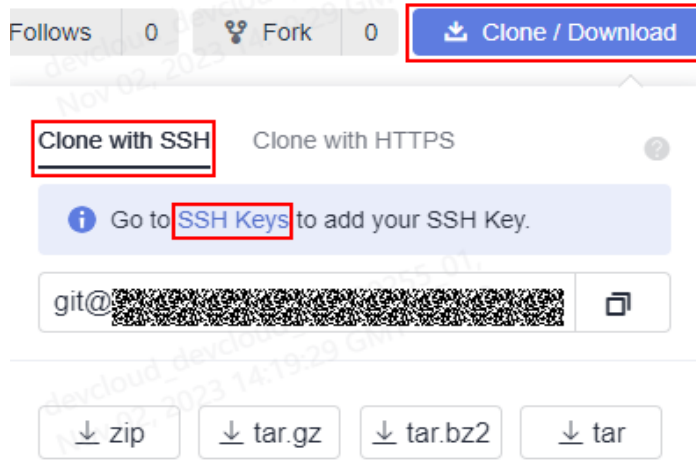
- c. Create an initial commit:
`git commit -m "init commit"`

Step 3 Set a remote server address for the local repository.

- If the Git repository is cloned from other systems, run the following command to add a new remote repository:

```
git remote add new git@***.***.com:testtransfer/Repo1.git # (replace the part after new with the repository address)
```

The repository address is displayed on the repository details page. The following figure shows how to obtain the repository address.



- If the Git repository is just initialized, run the following command to add a remote repository named **origin**.

```
git remote add origin git@***.***.com:testtransfer/Repo1.git # (replace the part after origin with the repository address)
```

Step 4 Push all code to CodeArts Repo.

```
git push new master # (when the Git repository is cloned from other systems)  
git push origin master # (when the Git repository is just initialized)
```

----End

NOTE

Basic Git knowledge is required for the preceding operations. If you have any questions during the operation, see [the Git website](#) or contact technical support.

5 Creating a CodeArts Repo Repository

[5.1 Overview](#)

[5.2 Creating an Empty Repository](#)

[5.3 Creating a Repository Using a Template](#)

[5.4 Importing an External Repository](#)

[5.5 Forking a Repository](#)

5.1 Overview

Currently, CodeArts Repo provides the following ways to create a repository.

- **5.2 Creating an Empty Repository:** You can create a local repository and synchronize it to CodeArts Repo.
- **5.3 Creating a Repository Using a Template:** You can create a repository using a CodeArts Repo template when there is no local repository.
- **5.4 Importing an External Repository:** You can import a cloud repository to CodeArts Repo or import a CodeArts Repo repository from a region to another region (see [9.2.5 Repository Backup](#)). The imported repository is independent of the source repository.
 - Scenario 1: Migrate Gitee and GitHub repositories and projects to CodeArts Repo.
 - Scenario 2: Migrate CodeArts projects from a region to other regions.
- **Forking a Repository:** You can fork a CodeArts Repo repository, make changes to the fork, and merge the changes to the source repository.
 - Scenario 1: Carry out new projects based on historical projects without damaging the repository structure of the historical projects.
 - Scenario 2: Share projects of your organization with others.

NOTICE

- The capacity of a single repository cannot exceed 2 GB (including LFS usage). If the capacity exceeds 2 GB, the repository cannot be used properly and cannot be expanded.
- When the capacity of a repository exceeds the upper limit, the repository is frozen. In this case, you are advised to delete the repository, control the capacity locally, and push the repository again.

Common Repository Settings

- [9.2.1 Repositories](#)
- [9.3.3 Commit Rules](#)
- [9.3.4 Merge Requests](#)
- [9.3.1 Protected Branches](#)
- [9.5.2 IP Address Whitelists](#)
- [More settings](#)

5.2 Creating an Empty Repository

You can create an empty repository and synchronize a local repository to CodeArts Repo. To create an empty repository on the CodeArts Repo console, perform the following steps:

- Step 1** Access the repository list page.
- Step 2** Click **New Repository**. On the page that is displayed, enter basic repository information.

Table 5-1 Parameters for creating an empty repository

Parameter	Mandatory	Remarks
Repository Name	Yes	The name must start with a letter, digit, or underscore (_) and can contain periods (.) and hyphens (-), but cannot end with .git, .atom, or period (.). The name can contain a maximum of 200 characters.
Project	Yes	<ul style="list-style-type: none">• A repository must be associated with a project.• If the account does not have a project, click Create Project in the drop-down list box to create a basic, a Scrum or an IPD-Self-Operated Software/Cloud Service project. <p>NOTE If you create a repository in a project, the project is selected for Project by default, and the Project parameter is hidden on the repository creation page.</p>

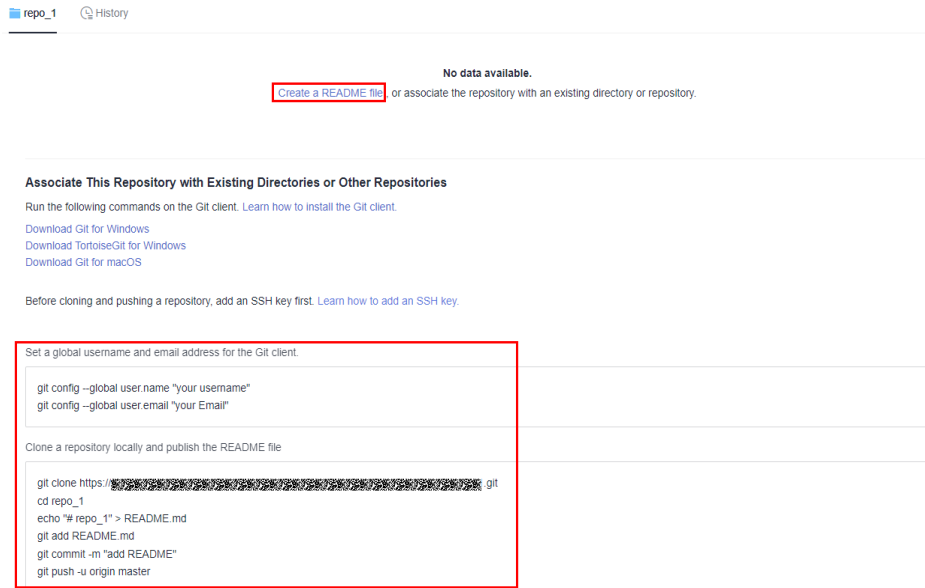
Parameter	Mandatory	Remarks
Description	No	Enter a description for your repository. The description can contain a maximum of 2000 characters.
Programming Language of .gitignore	No	The .gitignore file is generated based on your selection. (For details about gitignore, see Documentation .)
Permissions	No	The options are as follows: <ul style="list-style-type: none">• Make all project developers automatic repository members If you select this option, the project developer is automatically added as a repository member. By default, the project manager is a repository member.• Allow generation of a README file You can edit the README file to record information such as the project architecture and compilation purpose, which is similar to a comment on the entire repository.• Create a code check task automatically (for free). After the repository is created, you can view the code check task of the repository in the CodeArts Check task list after switching to the region where the repository is located.
Visibility	Yes	The options are as follows: <ul style="list-style-type: none">• Private The repository is visible only to repository members. Repository members can access the repository or commit code.• Public read-only The repository is open and read-only to all guests, but is not displayed in their repository list or search results. You can select an open-source license as the remarks.

Step 3 Click **OK** to create the repository. The repository list page is displayed.

----End

Associating with an Existing Directory or Repository

If you do not generate a README file when creating a common repository, you can click the **Code** tab, click **Create a README file** or associate the repository with an existing directory or repository. The procedure is as follows:



repo_1 History

No data available.
[Create a README file](#) or associate the repository with an existing directory or repository.

Associate This Repository with Existing Directories or Other Repositories

Run the following commands on the Git client. [Learn how to install the Git client.](#)

[Download Git for Windows](#)
[Download TortoiseGit for Windows](#)
[Download Git for macOS](#)

Before cloning and pushing a repository, add an SSH key first. [Learn how to add an SSH key.](#)

Set a global username and email address for the Git client.

```
git config --global user.name "your username"
git config --global user.email "your Email"
```

Clone a repository locally and publish the README file.

```
git clone https://... git
cd repo_1
echo "# repo_1" > README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Prerequisites

- You need to run following commands on the Git client. Install the Git client and configure the Git global username and user email address. For details, see [2 Git Installation and Configuration](#).
- Set the SSH key. For details, see [3.2 SSH Keys](#).

Procedure

NOTE

The following commands have been automatically generated in the new repository. You can copy them on the **Code** tab page of the repository.

Step 1 Clone the repository on the local host and push the new README file.

```
git clone HTTP_download_address
cd taskecho "# Repository_name" > README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Step 2 Associate an existing code directory with the repository.

```
cd <Your directory path>
mv README.md README-backup.md
git init
git remote add origin HTTP_download_address
git pull origin master
git add --all
git commit -m "Initial commit"
git push -u origin master
```

Step 3 Associate with an existing Git repository.

```
cd <Your Git repository path>
git remote remove origin > /dev/null 2>&1
git remote add origin HTTP_download_address
git push -u origin --all -f
git push -u origin --tags -f
```

----End

5.3 Creating a Repository Using a Template

You can create a repository using a CodeArts Repo template on the CodeArts Repo console.

Procedure


- Step 1** Access the repository list page.
- Step 2** Click  next to **New Repository** and select **Template Repository** from the drop-down list. The **Select Template** page is displayed.
- Step 3** On the **Select Template** page, enter a keyword for fuzzy search and select a template as required.
- Step 4** Click **Next**. On the **Basic Information** page, enter basic repository information.

Table 5-2 Parameters for creating a repository using a template

Parameter	Man dato ry	Remarks
Repository Name	Yes	The name must start with a letter, digit, or underscore (_) and can contain periods (.) and hyphens (-), but cannot end with .git, .atom, or period (.). Min. 2 characters; Max. 200 characters.
Project	Yes	<ul style="list-style-type: none">A repository must be associated with a project.If the account does not have a project, click Create Project in the drop-down list box to create a basic, a Scrum or an IPD-Self-Operated Software/Cloud Service project. <p>NOTE If you create a repository in a project, the project is selected for Project by default, and the Project parameter is hidden on the repository creation page.</p>
Description	No	Enter a description for your repository. The description can contain a maximum of 2000 characters.
Permissions	No	<ul style="list-style-type: none">Make all project developers automatic repository members If you select this option, the project developer is automatically added as a repository member. By default, the project manager is a repository member.Create a code check task automatically (for free). After the repository is created, you can view the code check task of the repository in the code check task list

Parameter	Mandatory	Remarks
Visibility	Yes	The options are as follows: <ul style="list-style-type: none">• Private The repository is visible only to repository members. Repository members can access the repository or commit code.• Public The repository is open and read-only to all guests, but is not displayed in their repository list or search results. You can select an open-source license as the remarks.

Step 5 Click **OK** to create the repository.

----End


 **NOTE**

When you create a repository by template, the repository type of the selected template will be automatically configured for the repository.

The repository created using the template contains the repository file structure preset in the template.

Automatically Creating a Pipeline

A pipeline can be automatically created when a repository is created using a template. Note that the host used in CodeArts Deploy must be changed to the actual environment so that the pipeline can be successfully executed.

Step 1 On CodeArts Repo, click  next to **New Repository** and select **Template Repository**.

Step 2 On the **Select Template** page, set **Automated Pipeline Creation** to **Yes** in the navigation pane to display templates that can be used to automatically create a pipeline.

Automated Pipeline Creation 

- All
 Yes
 No

Step 3 Select a template as required, click **Next**, enter basic repository information, and click **OK**.

Step 4 After the repository is created, you can view the pipeline that is automatically created on the pipeline list page displayed.


----End

5.4 Importing an External Repository

You can import a cloud repository to CodeArts Repo or import a CodeArts Repo repository from a region to another region (see [9.2.5 Repository Backup](#)). The imported repository is independent of the source repository.

To import an external repository on the CodeArts Repo console, perform the following steps:

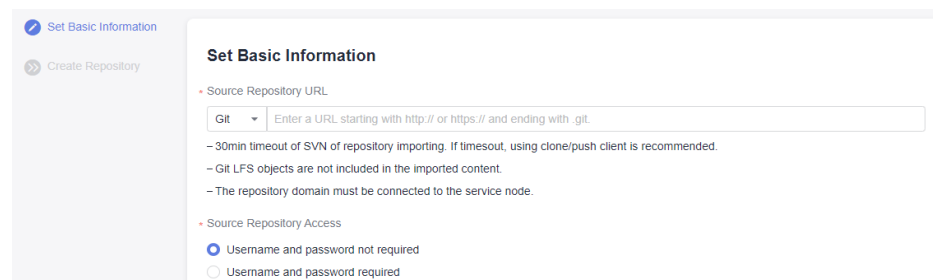
Step 1 Access the repository list page.

Step 2 Click  next to **New Repository** and select **Import Repository** from the drop-down list.

NOTICE

- An external repository can be a Git remote repository (HTTPS) or SVN repository.
- The source repository port can be 80, 443, or greater than 1024.
- Currently, GitHub, Gitee, GitLab, and SVN source repositories are supported. If the import using other types of source repositories fails, contact technical support to check the source server whitelist.

Step 3 Enter the source repository path, and enter the username and password for accessing the source repository. (This parameter is not required for open-source repositories.)



Step 4 Click **Next**. On the **Create Repository** page, enter the basic information about the repository.

Table 5-3 Parameter description

Parameter	Mandatory	Remarks
Repository Name	Yes	The name must start with a letter, digit, or underscore (_) and can contain periods (.) and hyphens (-), but cannot end with .git, .atom. The name can contain a maximum of 200 characters.

Parameter	Mandatory	Remarks
Description	No	Enter a description for your repository. The description can contain a maximum of 2,000 characters.
Permissions	No	<ul style="list-style-type: none">• Make all project developers automatic repository members If you select this option, the project developer is automatically added as a repository member. By default, the project manager is a repository member.• Create a code check task automatically (for free). After the repository is created, you can view the code check task of the repository in the check task list
Visibility	Yes	The options are as follows: <ul style="list-style-type: none">• Private The repository is visible only to repository members. Repository members can access the repository or commit code.• Public read-only The repository is open and read-only to all visitors. You can select an open-source license as the remarks.
Branch	Yes	You can choose to synchronize the default branch or all branches of the source repository.
Schedule	No	Select Schedule sync into repo. <ul style="list-style-type: none">• The default branch of the source repository is automatically imported to the default branch of the new repository every day.• The repository becomes a read-only image repository and cannot be written. In addition, only the branches of the third-party repository corresponding to the default branch of the current repository are synchronized.

Step 5 Click **OK** to import the repository. The repository list page is displayed.

----End

 **NOTE**

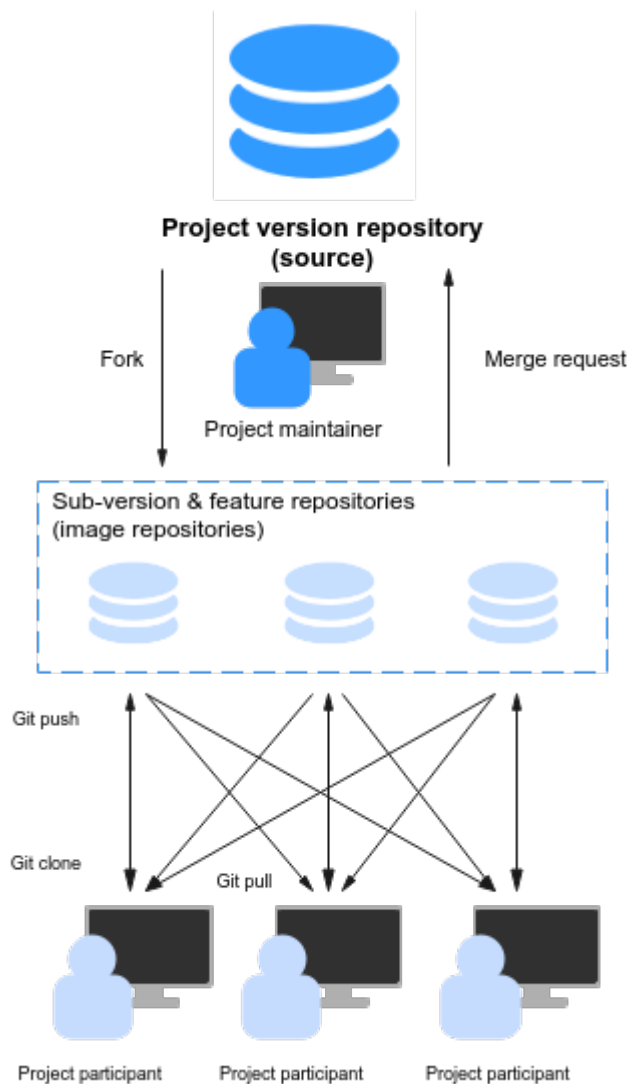
- The timeout interval for importing a repository is 30 minutes. If the import times out, use the clone/push function on the client.
- The [Git LFS](#) object is not imported.
- The repository domain must be connected to the service node.

5.5 Forking a Repository

Application Scenarios

You can fork a CodeArts Repo repository based on an image repository, make changes to the fork, and merge the changes to the source repository. Before changes are merged, the changes of the fork or the source repository will not affect each other.

As shown in the following figure, fork is applicable to the development scenario where a large-scale project contains multiple sub-projects. The complex development process occurs only in image repositories and the project repository (source repository) is not affected. Only new features that are completed can be merged to the project repository. Fork can be considered as a team collaboration mode.



Differences Between Forking a Repository and Importing an External Repository

The two modes are both repository replication. The main difference lies in the association between the source repository and the copied repository. The details are as follows:

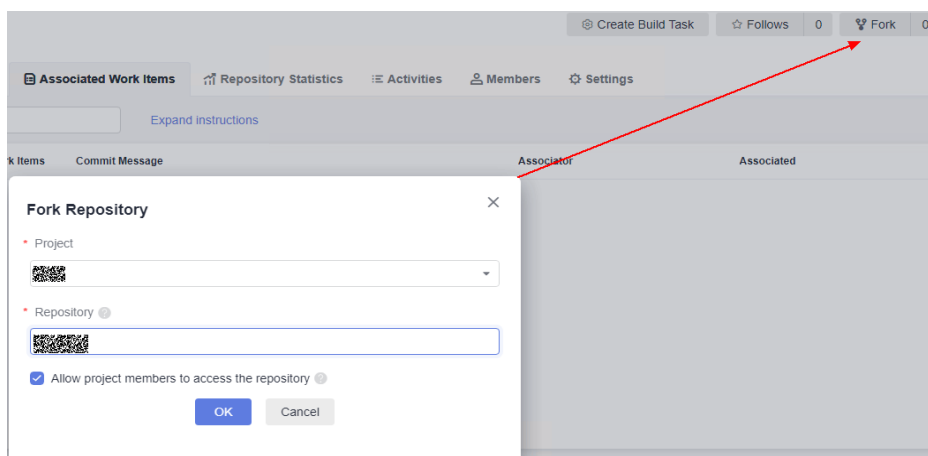
- **Fork**
 - Forks are used to copy repositories on CodeArts Repo.
 - A fork generates a repository copy based on the current version of the source repository. You can apply for merging changes made on the fork to the source repository (cross-repository branch merge), but you cannot pull updates from the resource repository to the fork.
- **Import**
 - You can import repositories of other version management platforms (mainly Git- and SVN-based hosting platforms) or your own repository to CodeArts Repo.
 - An import also generates a repository copy based on the current version of the source repository. The difference is that you can pull the default branch of the source repository to the repository copy at any time to obtain the latest version, but you cannot apply for merging changes made on the repository copy to the source repository.

Forking a Repository

Step 1 Access the repository list page.

Step 2 Click a repository name to go to the target repository.

Step 3 Click **Fork** in the upper right corner of the page. In the **Fork Repository** dialog box that is displayed, select the target project, enter the repository name, and select **Allow project members to access the repository** .



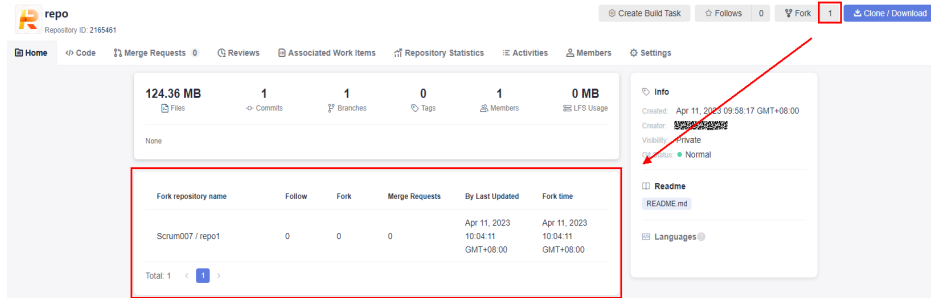
Step 4 Click **OK** to fork the repository.

----End

Viewing the List of Forked Repositories

- Step 1** Access the repository list page.
- Step 2** Click the source repository name.
- Step 3** Click **Fork** in the upper right corner of the page to view the list of forked repositories, as shown in the following figure.

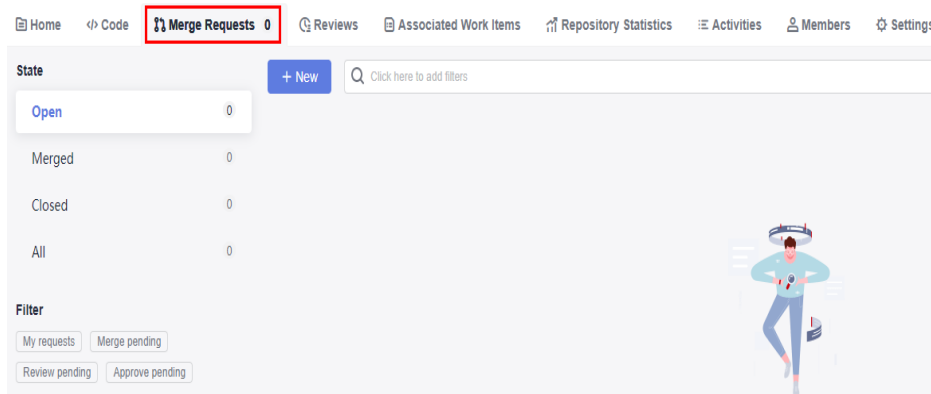
You can click the name of a forked repository to access the repository.



----End

Merging Changes of a Fork to the Source Repository

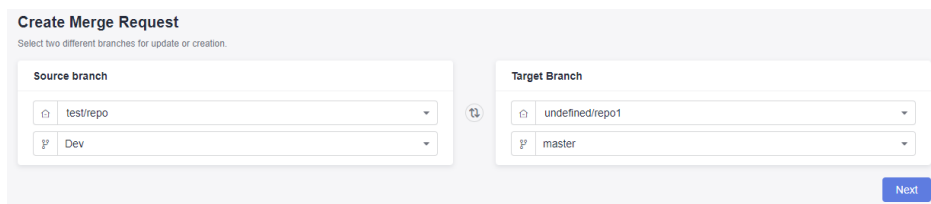
- Step 1** Access the repository list page.
- Step 2** Click the name of the forked repository.
- Step 3** Click the **Merge Requests** tab.



- Step 4** Click **New**. The **Create Merge Request** page is displayed.

Source Branch is the one that requests merging.

Target Branch is the one that merges content.



Step 5 Click **Next**. The page for creating a merge request is displayed. The subsequent operation process is the same as that of creating a merge request in the repository. For details, see [Creating a Merge Request](#).

----End

 **NOTE**

A cross-repository MR belongs to the source repository and can be viewed only on the **Merge Requests** tab of the source repository. Therefore, reviewers, approvers, and mergers must be members of the source repository.

6 Associating the CodeArts Repo Repository

Before using CodeArts Repo, initialize the local project files to a Git repository and associate it with a CodeArts Repo repository.

Prerequisites

You have installed the [Git client](#) and [bound the SSH key of the Git client to CodeArts Repo](#).

Procedure

Step 1 Create a CodeArts Repo repository.

If you select gitignore based on your local code library, some non-development files will be ignored and will not be managed in Git.

Step 2 Initialize the local repository to a Git repository.

Open the Git Bash client in your repository and run the following command:

```
git init
```

The following figure shows that the initialization is successful. The current folder is the local Git repository.



```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/liu'Code/java
$ git init
Initialized empty Git repository in C:/Users/Administrator/Desktop/liu'Code/java/.git/
```

Step 3 Bind the CodeArts Repo repository.

1. Go to the CodeArts Repo repository and obtain the repository address.
2. Run the remote command to bind the local repository to the cloud repository.
`git remote add <repository_alias> <repository_address>`

Example:

```
git remote add origin git@*****/java-remote.git # Change the address to that of your repository.
```

By default, **origin** is used as the repository alias when you clone a remote repository to the local computer. You can change the alias.

If the system displays a message indicating that the repository alias already exists, use another one.

If no command output is displayed, the binding is successful.

Step 4 Pull the master branch of the CodeArts Repo repository to the local repository.

This step is performed to avoid conflicts.

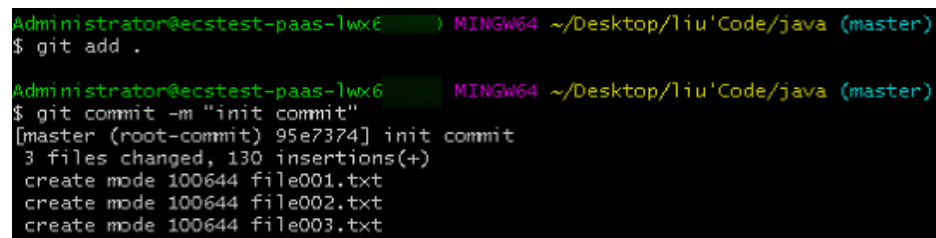
```
git fetch origin master # Change origin to your repository alias.
```

Step 5 Commit local code files to the master branch.

Run the following commands:

```
git add .  
git commit -m "<your_commit_message>"
```

The following figure shows a successful execution.



```
Administrator@ecstest-paas-lwx6 ( ) MINGW64 ~/Desktop/liu'Code/java (master)  
$ git add .  
Administrator@ecstest-paas-lwx6 ( ) MINGW64 ~/Desktop/liu'Code/java (master)  
$ git commit -m "init commit"  
[master (root-commit) 95e7374] init commit  
3 files changed, 130 insertions(+)  
create mode 100644 file001.txt  
create mode 100644 file002.txt  
create mode 100644 file003.txt
```

Step 6 Bind the local master branch to the master branch of CodeArts Repo repository.

```
git branch --set-upstream-to=origin/master master # Change origin to your repository alias.
```

If the following information is displayed, the binding is successful.



```
Administrator@ecstest-paas-l ( ) MINGW64 ~/Desktop/liu'Code/java (master)  
$ git branch --set-upstream-to=origin/master master  
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Step 7 Merge the files in the CodeArts Repo repository and local repository and store them locally.

```
git pull --rebase origin master # Change origin to your repository alias.
```

The following figure is displayed, indicating that the merged repository has been placed in the working directory and repository.



```
Administrator@ecstest-paas-lwx ( ) MINGW64 ~/Desktop/liu'Code/java (master)  
$ git pull --rebase origin master  
From https://gitee.com:ecstest0001/java-remote  
* branch      master      -> FETCH_HEAD  
Successfully rebased and updated refs/heads/master.
```

Step 8 Push the local repository to overwrite the CodeArts Repo repository.

Run the **push** command because the repositories have been bound:

```
git push
```

After the operation is successful, pull the repository to verify that the version of the CodeArts Repo repository is the same as that of the local repository.

```
Administrator@ecstest-paas-... MINGW64 ~/Desktop/liu'Code/java (master)
$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 427 bytes | 427.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
To ... .git
   Oca3cd3..bafb729 master -> master

Administrator@ecstest-paas-1wx... MINGW64 ~/Desktop/liu'Code/java (master)
$ git pull
Already up to date.
```

----End

7 Cloning or Downloading Code from CodeArts Repo to a Local PC

[7.1 Overview](#)

[7.2 Using SSH to Clone Code from CodeArts Repo to a Local PC](#)

[7.3 Using HTTPS to Clone Code from CodeArts Repo to a Local Computer](#)

[7.4 Downloading a Code Package on a Browser](#)

7.1 Overview

In addition to [8.4.1 Managing Files](#), the Git-based CodeArts Repo also allows you to download repository files to a local PC.

There are three methods of cloning or downloading a repository to a local PC for the first time:

- [7.2 Using SSH to Clone Code from CodeArts Repo to a Local PC](#)
- [7.3 Using HTTPS to Clone Code from CodeArts Repo to a Local Computer](#)
- [7.4 Downloading a Code Package on a Browser](#)

7.2 Using SSH to Clone Code from CodeArts Repo to a Local PC

Prerequisites

Your network can access CodeArts Repo. For details, see [Network Connectivity Verification](#).

Cloning Code on the Git Bash Client Using SSH

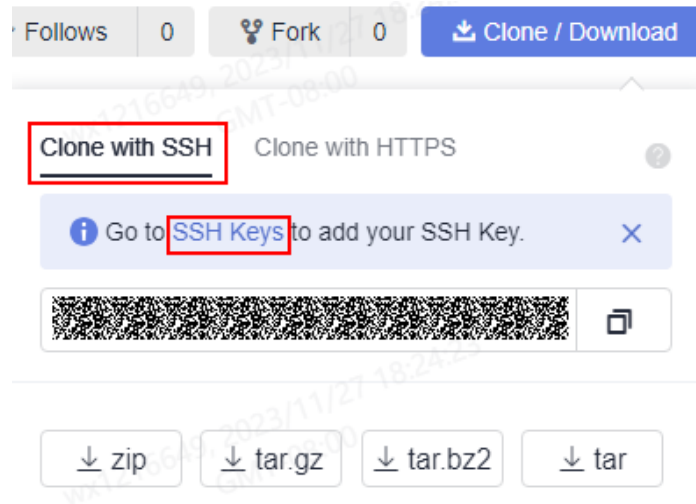
This section describes how to use the Git Bash client to clone a repository of CodeArts Repo to a local PC.

Step 1 Download and install the Git Bash client.

Step 2 Configure an SSH key.

Step 3 Obtain the repository address. (If there is no repository, [create one](#).)

On the repository details page, click **Clone/Download** to obtain the SSH address. You can use this address to connect to CodeArts Repo from the local PC.



NOTE

If no SSH key is available, click **SSH Keys** to configure one. For details, see [SSH key](#). You can obtain the SSH address from **URL** in the repository list of CodeArts Repo.

Step 4 Open the Git Bash client.

Create a folder on the local PC to store the code repository. Right-click the blank area in the folder and open the Git Bash client.

NOTE

The repository is automatically initialized during clone. You do not need to run the **init** command.

Step 5 Run the following command to clone code from CodeArts Repo:

```
git clone <repository_address>
```

repository_address in the command is the SSH address obtained in [Step 3](#).

If you clone the repository for the first time, the system asks you whether to trust the remote repository. Enter **yes**.

After the command is executed, a folder with the same name as CodeArts Repo is displayed, and a hidden **.git** folder exists in the folder, indicating that the repository is cloned.

Step 6 Run the following command to go to the repository directory:

```
cd <repository_name>
```

You will be taken to the **master** branch by default.

```
Administrator@gittestcce MINGW64 /c/git-test
$ cd test_War_Java_Demo

Administrator@gittestcce MINGW64 /c/git-test/test_War_Java_Demo (master)
$
```

----End

NOTE

If the **git clone** command fails to be executed, locate the fault as follows:

- Check whether your network can access CodeArts Repo.

Run the following command on the Git client to test the network connectivity:

```
ssh -vT git@*****.com
```

If the returned information contains **Could not resolve hostname code*****.com: Name or service not known** as shown in the following figure, your network is restricted and you cannot access CodeArts Repo. In this case, contact your local network administrator.

- Check the SSH key. If necessary, [regenerate a key and configure it on the CodeArts Repo console](#).
- Only PCs that [enabled the IP address whitelist](#) can be cloned on the Git client.

Cloning Code on the TortoiseGit Client Using SSH

This section describes how to use the TortoiseGit client to clone a repository of CodeArts Repo to a local PC.

Step 1 [Download and install the TortoiseGit client](#).

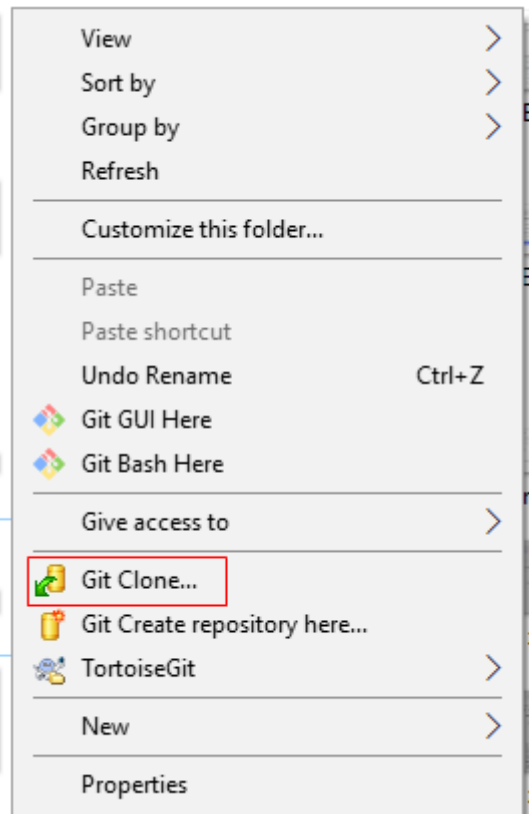
Step 2 Obtain the repository address. (If there is no repository, [create one](#).)

On the repository details page, click **Clone/Download** to obtain the SSH address. You can use this address to connect to CodeArts Repo from the local PC.

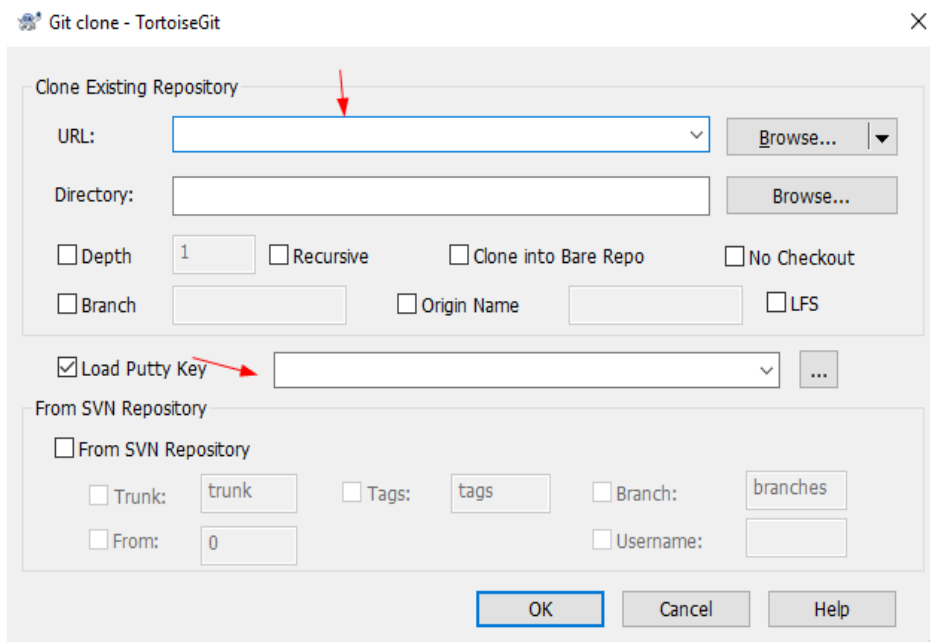
NOTE

You can obtain the SSH address from **URL** in the repository list of CodeArts Repo.

Step 3 Go to the local directory where you want to clone the repository, and choose **Git Clone...** from the right-click menu.

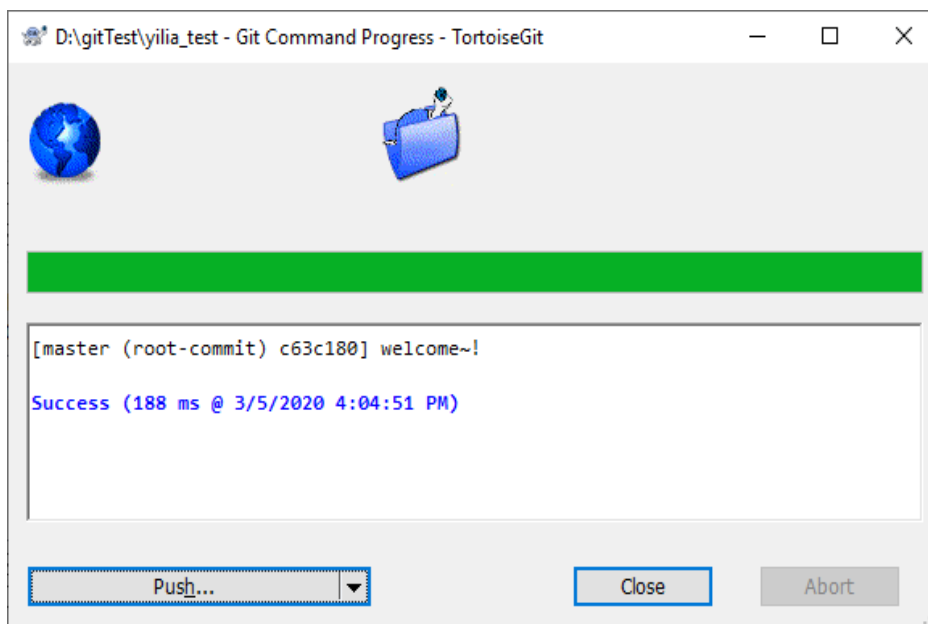


Step 4 In the dialog box displayed, paste the copied repository address to the **URL** field, select **Load Putty Key**, choose the **private key** file, and click **OK**.



Step 5 Click **OK** to start cloning the repository. If you clone the repository for the first time, the TortoiseGit client asks you whether to trust the remote repository. Click **Yes**.

Step 6 The cloning duration is affected by the repository size. The following figure shows the cloning process.



----End

Cloning a Repository on Linux or macOS Using SSH

After the environment is configured (see [2.4 Installing Git for Linux](#) or [2.5 Installing Git for macOS](#)), the clone operations of the Git client on Linux or macOS are the same as those in [Cloning Code on the Git Bash Client Using SSH](#).

7.3 Using HTTPS to Clone Code from CodeArts Repo to a Local Computer

Cloning Code on the Git Bash Client Using HTTPS

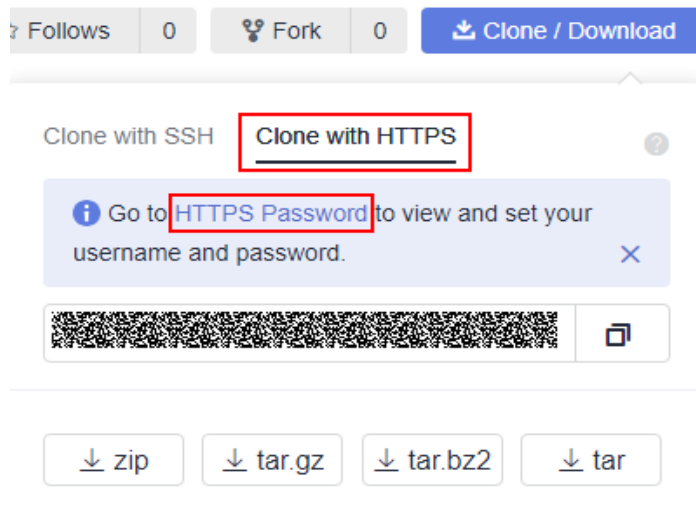
This section describes how to use the Git Bash client to clone a repository of CodeArts Repo to a local PC.

NOTICE

The maximum size of a package that can be pushed at a time using HTTPS is 200 MB. If the size is greater than 200 MB, use the SSH mode.

Federated users cannot be bound to email addresses and do not support the HTTPS protocol.

- Step 1** [Download and install the Git Bash client.](#)
- Step 2** [Configure an HTTPS password.](#)
- Step 3** On the CodeArts Repo homepage, click the name of a repository. On the repository details page displayed, click **Clone/Download**, click **Clone with HTTPS**, and copy the repository address.



NOTE

If no HTTPS password is available, click **HTTPS Password** to configure one. For details, see [HTTPS Password](#).

You can obtain the HTTPS address from **URL** in the repository list of CodeArts Repo.

Step 4 Open Git Bash, navigate to the directory where you want to clone the repository, and run the following command. For the first clone, enter the username (account name) and HTTPS password.

```
git clone HTTP_download_address
```

Step 5 After the username (account name) and HTTPS password are entered, the repository is cloned.

Step 6 Run the following command to go to the repository directory:

```
cd <repository_name>
```

You will be taken to the **master** branch by default.

----End

NOTE

If the git clone command fails to be executed, locate the fault as follows:

- Check whether your network can access CodeArts Repo.

Run the following command on the Git client to test the network connectivity:

```
ssh -vT git@*****.com
```

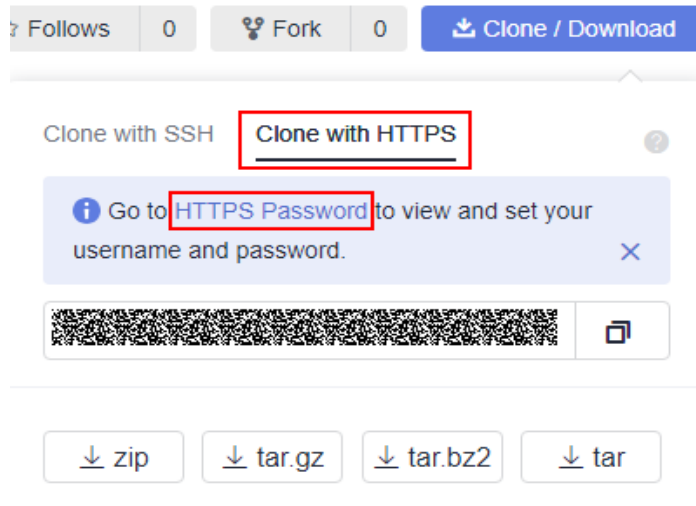
If the returned information contains **Could not resolve hostname code*****.com: Name or service not known** as shown in the following figure, your network is restricted and you cannot access CodeArts Repo. In this case, contact your local network administrator.

- Check the HTTPS password and [reset the password](#) if necessary.
- Only PCs that [enabled the IP address whitelist](#) can be cloned on the Git client.

Cloning Code on the TortoiseGit Client Using HTTPS

This section describes how to use the TortoiseGit client to clone a repository of CodeArts Repo to a local PC.

- Step 1** [Download and install the TortoiseGit client.](#)
- Step 2** [Configure an HTTPS password.](#)
- Step 3** On the CodeArts Repo homepage, click the name of a repository. On the repository details page displayed, click **Clone/Download**, click **Clone with HTTPS**, and copy the repository address.

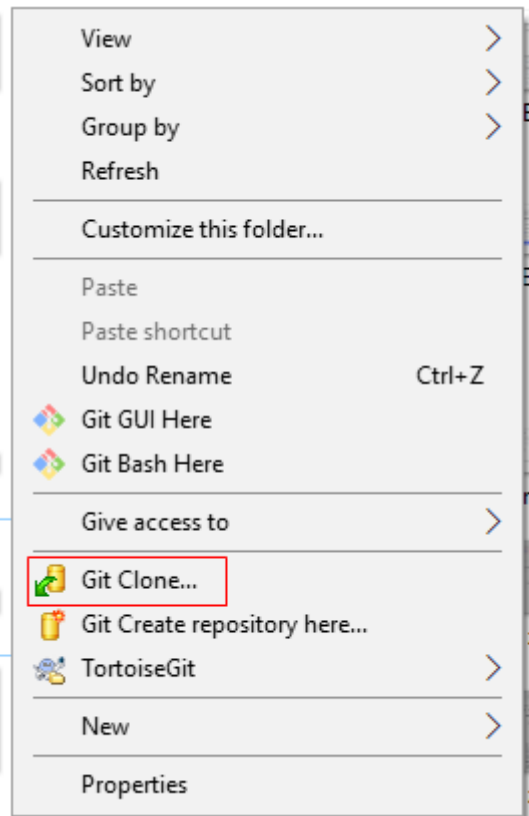


NOTE

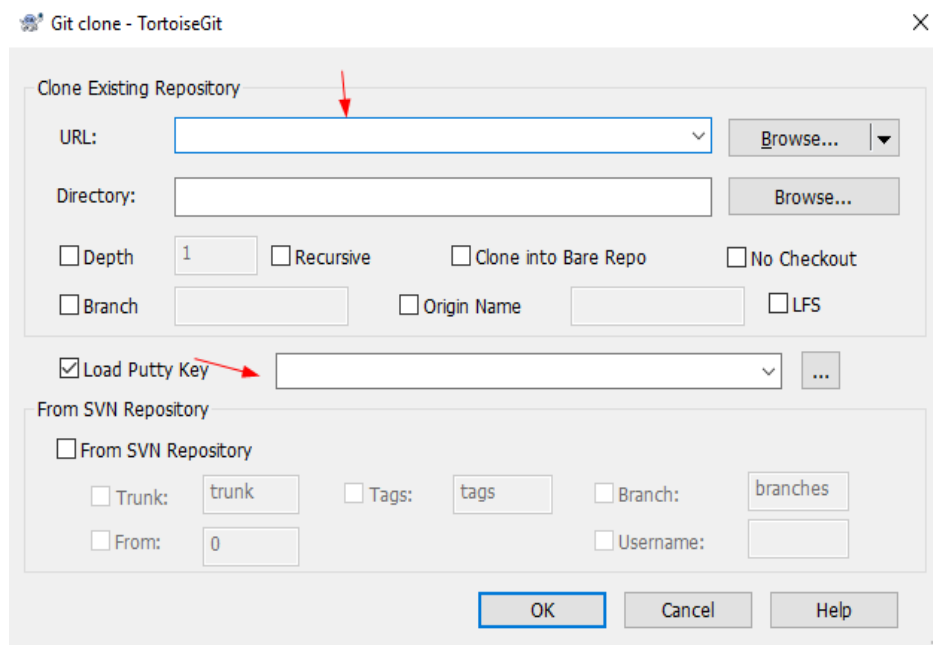
If no HTTPS password is available, click **HTTPS Password** to configure one. For details, see [HTTPS Password](#).

You can obtain the HTTPS address from **URL** in the repository list of CodeArts Repo.

- Step 4** Go to the directory where you want to clone the repository, and choose **Git Clone...** from the right-click menu.



Step 5 In the dialog box displayed, paste the copied repository address to the **URL** field and click **OK**.



Step 6 If you clone a repository on TortoiseGit for the first time, enter the username and HTTPS password as prompted.

Step 7 Wait until the clone is complete.

----End

Cloning a Repository on Linux or macOS Using HTTPS

After the environment is configured (see [2.4 Installing Git for Linux](#) or [2.5 Installing Git for macOS](#)), the clone operations of the Git client on Linux or macOS are the same as those in [Cloning Code on the Git Bash Client Using HTTPS](#).

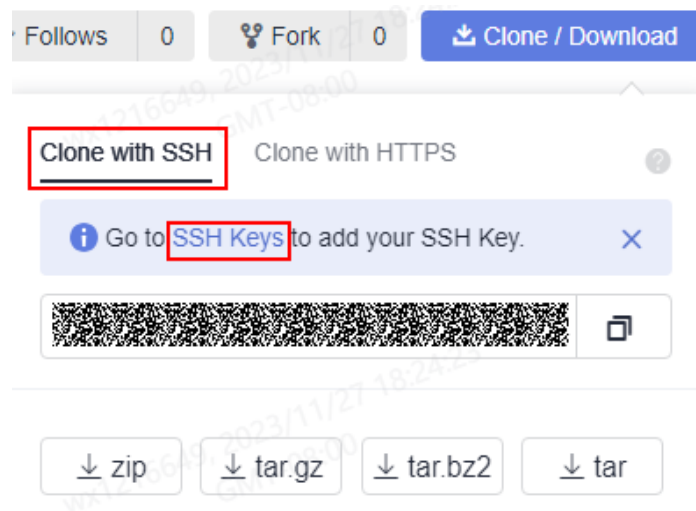
7.4 Downloading a Code Package on a Browser

In addition to clone, CodeArts Repo also allows you to package and download the code of a cloud repository to the local PC.

The downloaded code repository file is not associated with CodeArts Repo and cannot be pushed back to CodeArts Repo.

The procedure is as follows:

- Step 1** Access the repository list page.
- Step 2** Go to your repository. (If there is no repository, [create one](#).)
- Step 3** Click **Clone/Download**. In the dialog box that is displayed, click the required code package format.



----End

NOTE

- If an [IP address whitelist](#) is set for the repository, only hosts with whitelisted IP addresses can download the repository source code on the page. If no IP address whitelist is set for the repository, all hosts can download the repository source code.
- Currently, the zip, tar.gz, tar.bz2, and tar package formats are supported.
- The master branch of CodeArts Repo will be downloaded.

8 Using CodeArts Repo

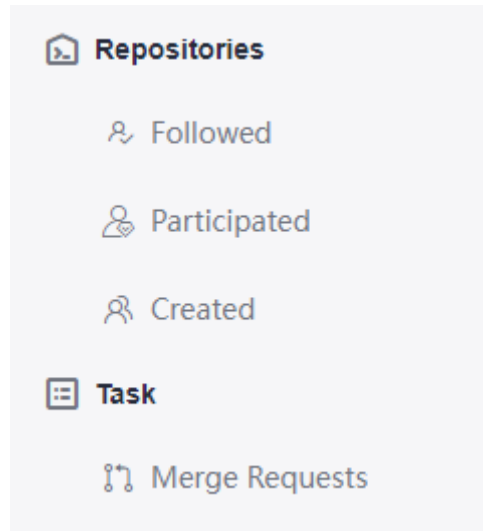
- [8.1 Viewing the Repository List](#)
- [8.2 Viewing Repository Details](#)
- [8.3 Viewing Repository Homepage](#)
- [8.4 Managing Code Files](#)
- [8.5 Managing MRs](#)
- [8.6 Viewing Review Records of a Repository](#)
- [8.7 Viewing Associated Work Items](#)
- [8.8 Viewing Repository Statistics](#)
- [8.9 Viewing Activities](#)
- [8.10 Managing Repository Members](#)

8.1 Viewing the Repository List

The repository list is the entry to CodeArts Repo. You can access the repository list in the following ways:

You can create a repository, configure a repository, and obtain the repository address.

- On your homepage, you can view repositories by category, such as **Followed**, **Participated**, and **Created**. You can click the name of a target repository to access the repository. You can view the combination requests of **Created by me**, **Merge pending**, **Review pending**, and **Approve pending**. You can click the name of a target merge request to access the combination request.

**NOTE**

If you access a project of CodeArts Repo, this function is hidden.

- You can [create a repository](#) by **New Repository**, **Template Repository** or **Import Repository**.
- Filter a Repository: You can select **All repositories**, **Unlocked repositories**, or **Locked repositories**. For details about how to lock a repository, see [Repository Locking](#).
- You can click the ☆ button to switch the following status of a repository.
- [Associated work Items](#) with CodeArts Req to improve efficiency.
- [Manage members](#) by synchronizing members from a project with one click or adjust the permission of a member separately.
- Delete a repository by entering a repository name.

NOTE

This operation cannot be canceled and deleted repositories cannot be restored. Please double-check.

8.2 Viewing Repository Details

In the repository list, click a repository name to go to the repository details page. CodeArts Repo provides abundant console operations.

Table 8-1 Description

Page	Function Description
Repository Homepage	Displays the repository capacity, commits number , branches number , tags number , members number, LFS usage, creation time, creator, visible scope, repository status, README file, language, and percentage of each language.

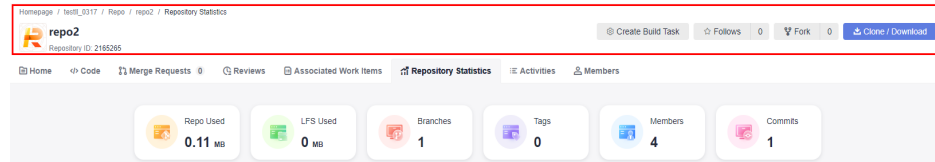
Page	Function Description
Code	<ul style="list-style-type: none">• File list: You can create files, directories, and submodules, upload files, modify files, modify blame, and view commit history.• Submit: You can view commit records and repository network diagrams.• Branch: Branches can be managed on the console.• Tag: Tags can be managed on the console.• Comparison: You can view code changes between branches or between tag versions by comparison.
Merge Requests	Merge requests of branches can be managed on the console.
Reviews	You can view the review records of MRs and commits.
Associated Work Items	List of associated work items. You can associate CodeArts Req work items with the repository code to improve efficiency.
Repository Statistics	Visualized charts of repository commits, such as code contribution.
Activity	You can view the dynamic information about the repository.
Members	You can manage repository members, for example, synchronizing members from the project by one click or changing the permissions of a member.
Settings	Repository settings. Only the repository administrator and the repository creator can view this tab page and configure settings.

In addition, the repository details page provides quick entries to the following functions:

- **Configure builds:** Create a build task.
- **Follow:** Click to follow the repository. The followed repositories are pinned on top.
- **Fork:** displays the number of forks of a repository. You can click this button to create a fork.
- **Clone/Download:** You can obtain the SSH address and HTTPS address of a repository or directly download the code package.

NOTE

The following figures show the **adaptation** function of CodeArts Repo. When the length of the repository page is greater than the window length, the repository tab page is moved to the top after you scroll down. The position in the red box in the following figure is collapsed so you can view repository information easily. After you scroll up, the page layout is restored.



8.3 Viewing Repository Homepage

The **Home** tab page displays the basic information about a repository.

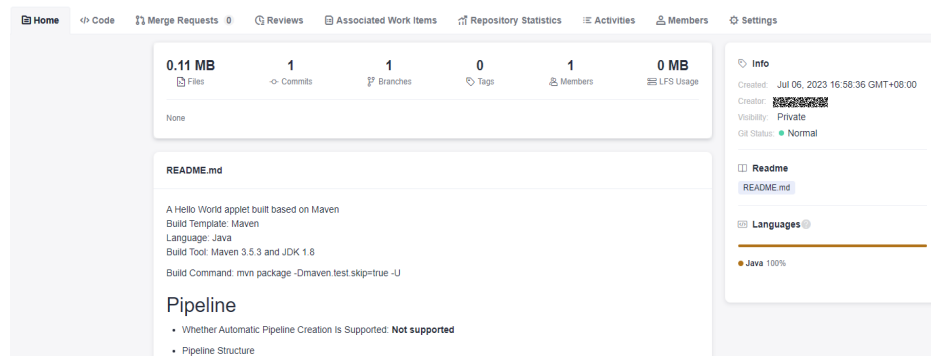
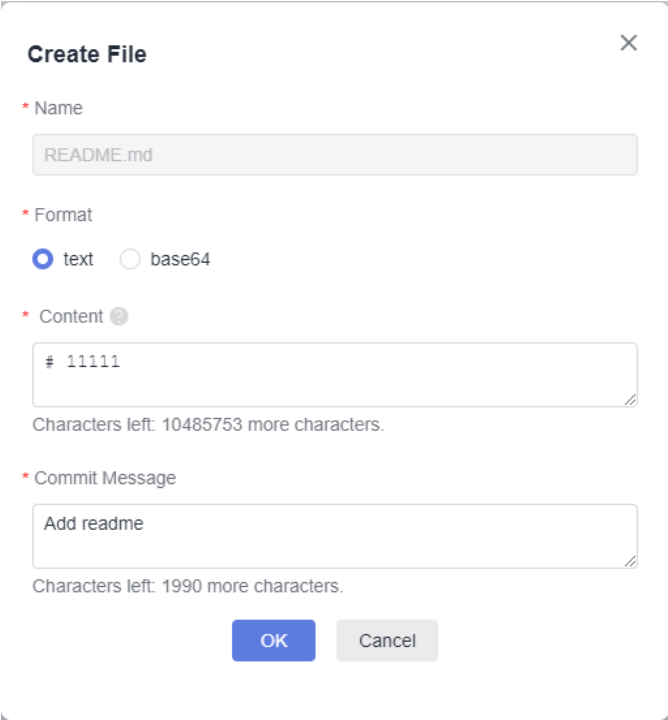


Table 8-2 Parameter description

Parameter	Description
Files	Capacity of the current repository NOTE <ul style="list-style-type: none"> The capacity of a single repository cannot exceed 2 GB (including LFS usage). If the capacity exceeds 2 GB, the repository cannot be used properly and cannot be expanded. When the capacity of a repository exceeds the upper limit, the repository is frozen. In this case, you are advised to delete the repository, control the capacity locally, and push the repository again.
Commits	Displays the number of commits in the current repository. You can click the icon to go to the Code tab page and view commit details.
Branches	Displays the number of branches in the current repository. You can click the icon to go to the Code tab page and manage branches.
Tags	Displays the number of tags in the current repository. You can click the icon to go to the Code tab page and manage tags.

Parameter	Description
Members	Displays the number of members in the current repository. You can click the icon to go to the Members tab page and manage members.
LFS Usage	Collect statistics on the LFS usage of the current repository.
Repository description	The description entered during repository creation.
README.md	<p>You can preview README files. If no Readme file exists in the repository, click Create Readme to create one.</p> <p>Name: The default file name is README.md.</p> <p>Format: The options are as follows:</p> <ul style="list-style-type: none">• text: indicates text data or a text string.• base64: Base64 is a method of representing binary data based on 64 printable characters. <p>Content: The value can be customized.</p> <ul style="list-style-type: none">• If the format is text, enter common text.• If the format is base64, enter Base64-encoded content that can pass the encoding verification. <p>Commit Message: Enter the commit information about the file or folder, which can be customized.</p> 
Info	Displays the creation time, creator, visible scope, and status of a repository.

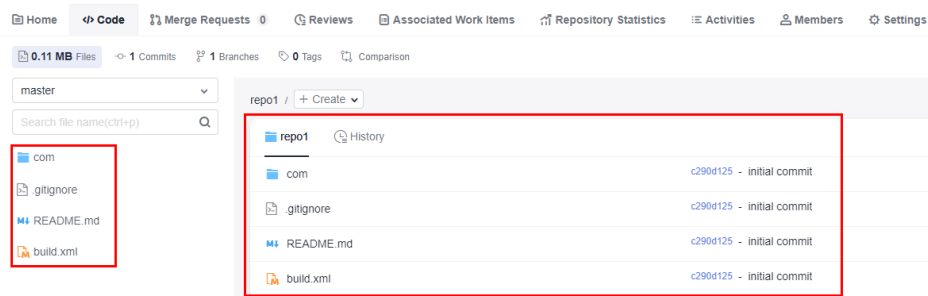
Parameter	Description
Readme	Displays the README file of the current repository. You can click the file name to go to the Code tab page and view the file content.
Languages	Displays the percentage of each language by file size in the current repository.

8.4 Managing Code Files

8.4.1 Managing Files

CodeArts Repo allows you to edit and compare files, and trace file changes.

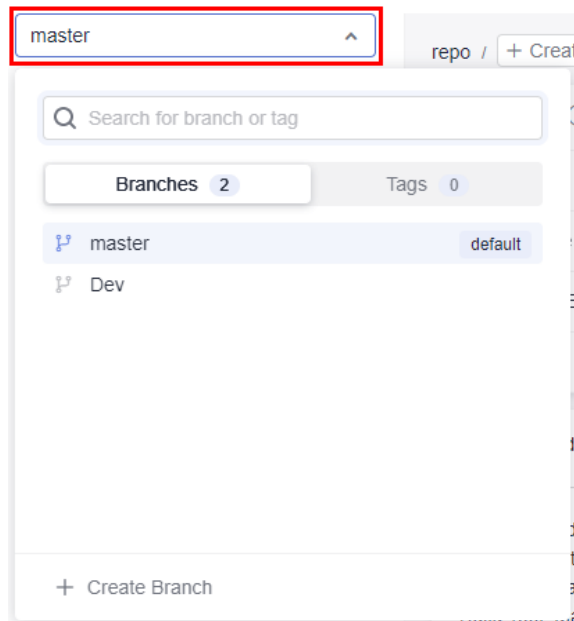
When you access [repository details console](#), the system locates the **Files** subtab on the **Code** tab page. You can switch to different branches and tags to view the files in the corresponding version. As shown in the following figure, the file list under the main branch is displayed on the left, the [Repository name \(file details of a branch or tag version\)](#) and [History \(branch or tag version\)](#) tab pages are displayed on the right.




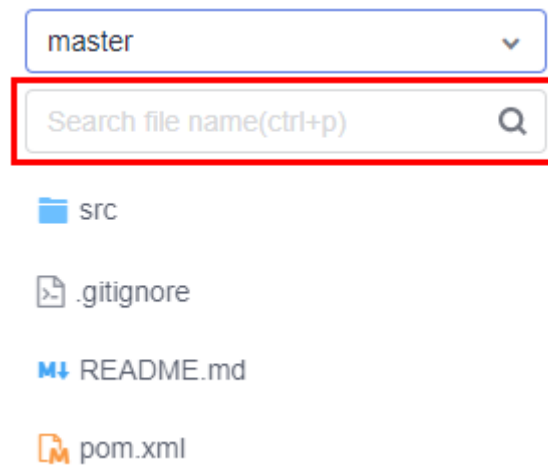
File List

The file list is on the left of the **Files** tab page of the repository. The file list provides the following functions:

1. Click a branch name to switch the branch and tag. After the branch and tag are switched, the file directory of the corresponding version is displayed.



2. Click  to display the search box. You can search for files in the file list.



3. Click . The following functions can be extended:

NOTICE

Multi-level directories are supported when you create a file, rename a file, create a directory, or create a submodule. Separate multi-level directories with slashes (/), for example, **java/com**.

– Creating a file

Creating a file on the CodeArts Repo console is to create a file and run the **add**, **commit**, and **push** commands. A commit record is generated.

On the **Create File** page, enter the file name, select the target template type, select the encoding type, enter the file content and commit information, and click **OK**.

 NOTE

The **Commit Message** field is equivalent to the **-m** message in git commit and can be used for [8.7 Viewing Associated Work Items](#).

– **Creating a directory**

Creating a directory on the CodeArts Repo console is to create a folder structure, and run the **add**, **commit**, and **push** commands. A commit record is generated.

A **.gitkeep** file is created at the bottom of the directory by default because Git does not allow a commit of an empty folder.

On the **Create Directory** page, enter the catalog name and commit information, and click **OK**.

– **Creating a submodule**


– **Uploading a file**

Uploading a file on the CodeArts Repo console is to create a file and run the **add**, **commit**, and **push** commands. A commit record is generated.

On the **Upload File** page, select the target file to be uploaded, enter the commit information, and click **OK**.


 NOTE

Move the cursor to the folder name and click  to perform the preceding operations in the folder.

4. Move the cursor to the file name and click  to change the file name.
Renaming a file on the CodeArts Repo console is to change a file name, and run the **add**, **commit**, and **push** commands. A commit record is generated.
5. You can click a file name to display the file content on the right of the page. You can modify the file content, trace file modification records, view historical records, and compare the file content.

Repository Name Tab Page: Viewing File Details of a Branch or Tag Version

By default, the **repository name** tab page displays file details of the master branch.



File Name	Commit Message	Update Time
com	c290d125 - initial commit	Repo Updated Mar 24, 2023 11:07:45 GMT+08:00
gitignore	c290d125 - initial commit	Repo Updated Mar 24, 2023 11:07:45 GMT+08:00
README.md	c290d125 - initial commit	Repo Updated Mar 24, 2023 11:07:45 GMT+08:00
build.xml	c290d125 - initial commit	Repo Updated Mar 24, 2023 11:07:45 GMT+08:00

It displays the following information:

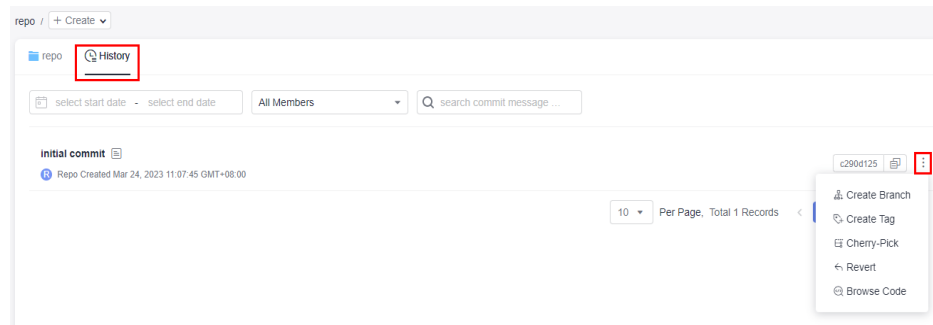
- **File:** name of a file or folder.
- **Commit message:** message of the last commit to the file or folder (**-m** in the **commit** command). You can click the message to display the commit record.
- **Creator:** creator of the last commit to the file or folder.
- **Update time:** last update time of the file or folder.

 NOTE


Commit messages are required for the edit and delete operations. They are similar to **-m** in the **git commit** command and can be used for associating work items. For details, see [8.7 Viewing Associated Work Items](#).

History Tab: Viewing the Commit History of a Branch or Tag Version

The **History** tab page displays the commit history of a branch or tag version.



On this page, you can perform the following operations on the commit history:

- Click a **commit name** to go to the commit details page.
- Click  to extend the following functions:
 - **Create Branch**.
 - **Create Tag**: You can create a tag for this commit. For details, see [What is a tag](#).
 - **Cherry-Pick**: Use the commit as the latest commit to overwrite a branch. It is used to retrieve a version.
 - **Revert**: undoing this commit
 - **Browse Code**.

Managing Repository Files


You can click a file name to manage the file. The functions are as follows:


 NOTE

When you maximize the browser window, the functions in the drop-down menu shown in the preceding figure are displayed in tile mode.

- **File name**: View the detailed content of the file.

Table 8-3 Screen description

Screen Function	Function Description
<i>File Capacity</i>	Indicates the capacity of the file.
Full Screen	Full screen to view the file content
Copy Code	Copy the file content to the clipboard.
Open Raw	You can view the original data of the file.
Edit	Edit the file online.
Download	Download the file to the local PC.
Delete	Delete a file
File content	The email content is displayed.
	Click this icon to add review comments.

- **Blame:** View the change history of a file and trace operations.
On this tab page, a modifier corresponds to their modified content. You can a record to view the commit details.
- **History:** View the commit history of the file.
On this page, you can perform the following operations on the commit history:
 - Click a **commit name** to go to the commit details page.
 -  provides the following functions:
 - Create Branch.
 - **Create Tag:** You can create a tag for this commit. ([Introduction](#))
 - **Cherry-Pick:** Use the commit as the latest commit to overwrite a branch. It is used to retrieve a version.
 - **Revert:** undoing this commit
 - **Browse Code.**
- **Comparison:** compares the committed differences.
The differences compared on the CodeArts Repo console are displayed in a better way than those on the Git Bash client. You can select different commit batches on the GUI for difference comparison.

 **NOTE**

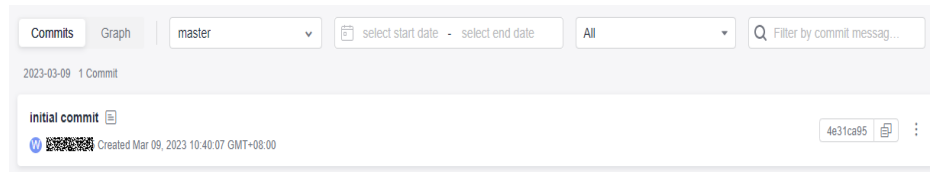
The comparison result shows the impact of merging from the left repository version to the right repository version on the files in the right repository. If you want to know the differences between the two file versions, you can adjust the left and right positions, compare them again, and learn all the differences based on the two results.

8.4.2 Managing Commits

On the **Code** and **Commits** tab pages, view the commit records and graph of the repository.

Commits

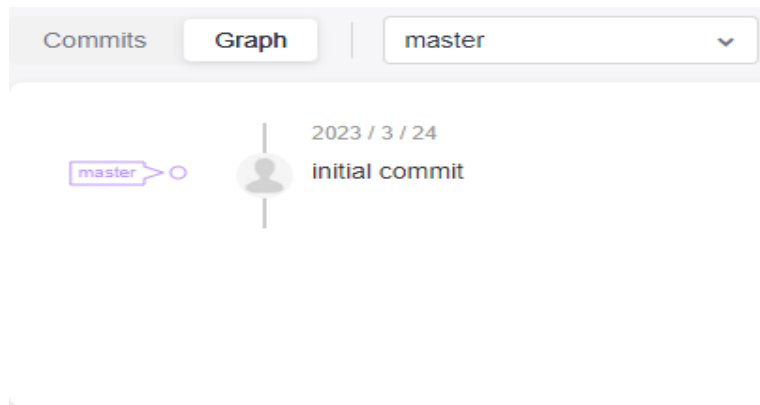
This tab displays the entire commit records of a branch or tag in the current repository. You can filter records by time segment, committer, commit message, or commit ID.



Graph

The commit graph of a repository displays the entire commit history (including the action, time, committer, commit message generated by the system or specified by the committer) of a branch or tag and the relationship between commits in flow chart.

You can switch between branches or tags. You can click a commit node or commit message to go to the corresponding commit record.



NOTE

Compared with the **History** tab page under the **Files** tab page, the commit graph can display the relationship between commits.

8.4.3 Managing Branches

Branching is the most commonly used method in version management. Branches isolate tasks in a project to prevent them from affecting each other, and can be **merged** for version release.

When you create a CodeArts Repo or Git repository, a master branch is generated by default and used as the branch of the latest version. You can create custom branches at any time for personalized scenarios.

GitFlow

As a branch-based code management workflow, **GitFlow** is highly recognized and widely used in the industry. It is recommended for you to start team-based development.

GitFlow provides a group of branch usage suggestions to help your team improve efficiency and reduce conflicts. It has the following features:

- **Concurrent development:** Multiple features and patches can be concurrently developed on different branches to prevent intervention during code writing.
- **Team collaboration:** In team-based development, the development content of each branch (or each sub-team) can be recorded separately and merged into the project version. An issue can be accurately detected and rectified separately without affecting other code in the main version.
- **Flexible adjustment:** Emergency fixes are developed on the hotfix branch without interrupting the main version and sub-projects of each team.

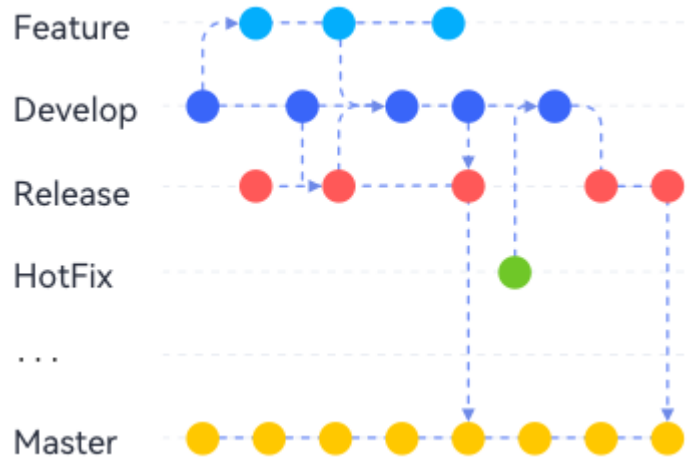


Table 8-4 Suggestions on using GitFlow branches

Branch	Master	Develop	Feature_1\ 2...	Release	HotFix_1\ 2..
Descri ption	Core branch, which is used together with tags to archive historical versions. Ensure that all versions are available.	Main development branch, which is used for routine development and must always be the branch with the latest and most complete functions.	Feature development branch, which is used to develop new features. Multiple branches can exist concurrently. Each branch corresponds to a new feature or a group of new features.	Release branch, which is used to check out a version to be released.	Emergency fix branch, which is used to fix bugs in the current version.
Validit y	Long-term	Long-term	Temporary	Long-term	Temporary

Branch	Master	Develop	Feature_1\ 2...	Release	HotFix_1\ 2.. ..
When to Create	Created when the project repository is created	Created after the master branch is created.	<ul style="list-style-type: none"> Created based on the develop branch when a new feature development task is received. Created based on the parent feature branch when the current feature development task is split into sub-tasks. 	Created based on the develop branch before the first release.	Created based on the corresponding version (usually the master branch) when issues are found in the master or bug version.
When to Develop This Branch	Never	Not recommended	Developed when being created.	Never	Developed when being created.

Branch	Master	Develop	Feature_1\ 2...	Release	HotFix_1\ 2.. ..
<p>When to Merge Other Branches into This Branch</p>	<ul style="list-style-type: none"> When the project version is frozen, the develop or release branch are merged into this branch. After bugs found in the released version are fixed, hotfix branches are merged into this branch. 	<ul style="list-style-type: none"> After new features are developed, feature branches are merged into this branch. When a new version starts to be developed, the last version (release or master branch) is merged into this branch. 	<p>After a child feature branch is developed and tested, it is merged into the parent feature branch.</p>	<p>When a version is to be released, the develop branch is merged into this branch.</p>	<p>-</p>

Branch	Master	Develop	Feature_1\ 2...	Release	HotFix_1\2.. ..
When to Merge This Branch to Other Branches	-	<ul style="list-style-type: none"> When a version is to be released, this branch is merged into the release branch. When a version is to be archived, this branch is merged into the master branch. 	After new features are developed and tested on this branch, it is merged into the develop branch.	<ul style="list-style-type: none"> When a version is released and archived, this branch is merged into the master branch. When a new version is developed based on a released version, this branch is merged into the develop branch to initialize the version. 	When the corresponding bug fixing task is complete, this branch is merged into the master and develop branches as a patch.
When to End	-	-	After the corresponding features are accepted (released and stable)	-	After the corresponding bugs are fixed and the version is accepted (released and stable)

 **NOTE**

GitFlow has the following rules:

- All feature branches are pulled from the develop branch.
- All hotfix branches are pulled from the master branch.
- All commits to the master branch must have tags to facilitate rollback.
- Any changes that are merged into the master branch must be merged into the develop branch for synchronization.
- The master and develop branches are the main branches and they are unique. Other types of branches can have multiple derived branches.

Creating a Branch on the Console


Step 1 Access the repository list.

Step 2 Click a repository to go to the details page.

Step 3 Click the **Code** and **Branches** tabs. The branch list page is displayed.

Step 4 Click **Create**. In the displayed dialog box, select a version (branch or tag) based on which you want to create a branch and enter the branch name. You can associate the branch with an existing work item.

Create Branch ✕

* Based On 

test ▼

* Branch Name

Max. 200 bytes.

Description

Description

Characters left: 2000 more characters.

Work Items to Associate

--Select-- ▼

OK Cancel

 **NOTE**

The branch name must meet the following requirements:

- The name cannot start with a **hyphen (-)**, **period (.)**, **refs/heads/**, **refs/remotes/**, or **slash (/)**.
- Spaces and special characters such as [`<~^:?!()"$&`; are not supported.
- The name cannot end with a **period (.)**, **slash (/)**, or **.lock**.
- Two consecutive periods (..) are not allowed.
- The name cannot contain this sequence `@{`.







The name cannot be the same as another branch or tag name.

Step 5 Click **OK**. The branch is created.

----End

Managing Branches on the Console

You can perform the following operations in the branch list:

- Filtering branches
 - **My**: displays all branches created by you. The branches are sorted by the latest commit time in descending order.
 - **Active**: displays the branches that have been developing in the past three months. Branches are sorted by the last commit time in descending order.
 - **Inactive**: displays the branches that have not been developed in the past three months. Branches are sorted by the last commit time in descending order.
 - **All**: displays all branches. The default branch is displayed on the top. Other branches are sorted by the last commit time in descending order.
- You can click a **branch name** to go to the **Files** tab page of the branch and view its content and history.
- You can click a commit ID to view the content latest committed on the details page.
- Select branches and click **Batch Delete** to delete branches in batches.
- You can click  to associate work items with the branch.
- You can click  to go to the **Comparison** tab page and compare the current branch with another branch.
- Click  to download its compressed package.
- You can click  to the **Merge Requests** tab page and create a **merge request**.
- Click  to go to the repository settings page and set the branch as protected.
- You can click  to delete a branch as prompted.

NOTICE

You can download the compressed package of source code on the page only for hosts that have [configured IP address whitelists](#).

If you delete a branch by mistake, submit a service ticket to contact technical support.

In addition, you can configure branches on the console.

- [Merge Requests](#)
- [Default Branches](#)
- [Protected Branches](#)

Common Git Commands for Branches

- **Creating a branch**

```
git branch <branch_name> # Create a branch based on the current working directory in the local repository.
```

Example:

```
git branch branch001 # Create a branch named branch001 based on the current working directory in the local repository.
```

If no command output is displayed, the creation is successful. If the branch name already exists, as shown in the following figure, create a branch with another name.

```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git branch branch001
fatal: A branch named 'branch001' already exists.
```

- **Switching a branch**

Switching a branch is to check out the branch file content to the current working directory.

```
git checkout <branch_name> # Switch to a specified branch.
```

Example:

```
git checkout branch002 # Switch to branch002.
```

The following information shows that the switch is successful.

```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git checkout branch001
Switched to branch 'branch001'
```

- **Switching to a new branch**

You can run the following command to create a branch and switch to the new branch directly.

```
git checkout -b <branch_name> # Create a branch based on the current working directory in the local repository and directly switch to the branch.
```

Example:

```
git checkout -b branch002 # Create a branch named branch002 based on the current working directory in the local repository and directly switch to the branch.
```

The following information shows that the command is successfully executed.

```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch001)
$ git checkout -b branch002
Switched to a new branch 'branch002'

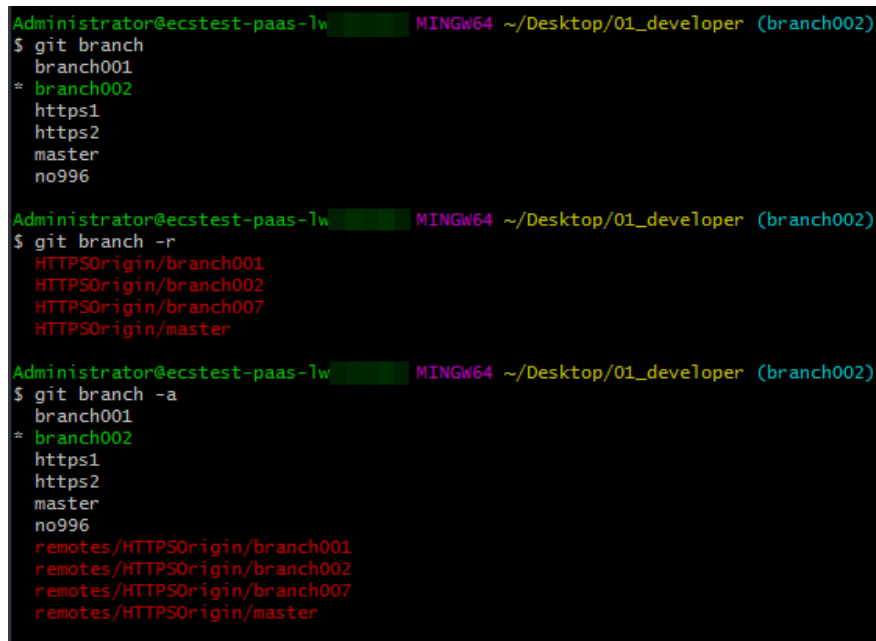
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch002)
$
```

- **Viewing a branch**

You can run the corresponding command to view the local repository branch, the remote repository branch, or all branches. These commands only list branch names. You can [switch to a branch](#) to view specific files in a branch.

```
git branch          # View the local repository branch.
git branch -r       # View the remote repository branch.
git branch -a       # View the branches of the local and remote repositories.
```

The following figure shows the execution result of the three commands in sequence. Git displays the branches of the local and remote repositories in different formats. (Remote repository branches are displayed in the format of `remote/<remote_repository_alias>/<branch_name>`.)



```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch002)
$ git branch
branch001
* branch002
https1
https2
master
no996

Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch002)
$ git branch -r
HTTPSorigin/branch001
HTTPSorigin/branch002
HTTPSorigin/branch007
HTTPSorigin/master

Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch002)
$ git branch -a
branch001
* branch002
https1
https2
master
no996
remotes/HTTPSorigin/branch001
remotes/HTTPSorigin/branch002
remotes/HTTPSorigin/branch007
remotes/HTTPSorigin/master
```

- **Merging a branch**

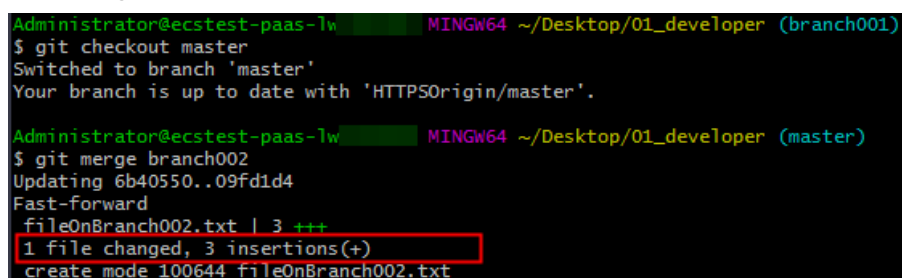
When a development task on a branch is complete, the branch needs to be merged into another branch to synchronize the latest changes.

```
git merge <name_of_the_branch_merged_to_the_current_branch> # Merge a branch into the current branch.
```

Before merging a branch, you need to switch to the target branch. The following describes how to merge **branch002** into the master branch.

```
git checkout master # Switch to the master branch.
git merge branch002 # Merge branch002 into the master branch.
```

The following figure shows the execution result of the preceding command. The merge is successful, and three lines are added to a file.



```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch001)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'HTTPSorigin/master'.

Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git merge branch002
Updating 6b40550..09Fd1d4
Fast-forward
 fileOnBranch002.txt | 3 +++
1 file changed, 3 insertions(+)
create mode 100644 fileOnBranch002.txt
```

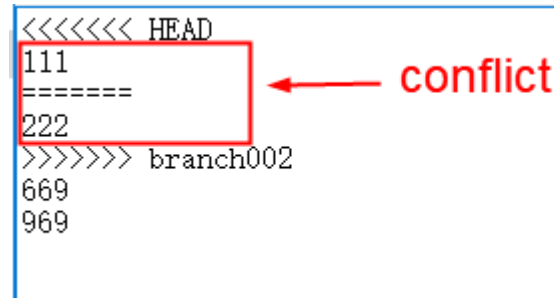
 NOTE

The system may prompt that a merge conflict occurs. The following shows that a conflict occurs in the `fileOnBranch002.txt` file.

```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (master)
$ git merge branch002
Auto-merging fileOnBranch002.txt
CONFLICT (content): Merge conflict in fileOnBranch002.txt
Automatic merge failed; fix conflicts and then commit the result.
```

To resolve the conflict, open the conflicting file, manually edit the conflicting code (as shown in the following figure), and save the file. Then run the `add` and `commit` commands again to save the result to the local repository.

```
<<<<<<< HEAD
111
-----
222
>>>>>> branch002
669
969
```



This is similar to resolving a conflict that occurs when you commit a file from the local repository to the remote repository. For details about the working principle, see [8.5.2 Resolving Code Conflicts in an MR](#).

[A proper collaboration mode](#) can prevent conflicts.

- **Deleting a local branch**

```
git branch -d <branch_name>
```

Example:

```
git branch -d branch002 # Delete branch002 from the local repository. The following
information shows that the operation is successful.
```

```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (master)
$ git branch -d branch002
Deleted branch branch002 (was 8ab93e7).
```

- **Deleting a branch from the remote repository**

```
git push <remote_repository_address_or_alias> -d <branch_name>
```

Example:

```
git push HTTPSOrigin -d branch002 # Delete branch002 from the remote repository whose alias
is HTTPSOrigin. The following information shows that the deletion is successful.
```

```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (master)
$ git push HTTPSOrigin -d branch002
To https://[redacted].git
- [deleted]          branch002
```

- **Pushing a new local branch to the remote repository**

```
git push <remote_repository_address_or_alias> <branch_name>
```

Example:

```
git push HTTPSOrigin branch002 # Push the local branch branch002 to the remote repository
whose alias is HTTPSOrigin. The following information shows that the push is successful.
```

```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (master)
$ git push HTTPSorigin branch002
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 2 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 861 bytes | 430.00 KiB/s, done.
Total 12 (delta 5), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for branch002, visit:
remote: https://[redacted].com/[redacted]639472/newmerge
remote:
To https://[redacted].git
 * [new branch]      branch002 -> branch002
```

NOTE

If the push fails, check the connectivity.

- Check whether your network can access CodeArts Repo.

Run the following command on the Git client to test the network connectivity:
ssh -vT git@[redacted].com

If the returned information contains **connect to host [redacted].com port 22: Connection timed out**, your network is restricted and you cannot access CodeArts Repo. In this case, contact your local network administrator.

- Check the SSH key. If necessary, regenerate a key and configure it on the CodeArts Repo console. For details, see [3.2 SSH Keys](#). Alternatively, check [whether the HTTPS password](#) is correctly configured.

8.4.4 Managing Tags

Git provides **tags** to help your team manage versions. You can use Git tags to mark commits to manage important versions in a project and search for historical versions.

A tag points to a commit like a reference. No matter how later versions change, the tag always points to the commit. It can be regarded as a version snapshot that is permanently saved (the version is removed from the repository only when being manually deleted).

When using Git to manage code, you can search for and trace historical versions based on commit IDs. A commit ID is a long string (as shown in the following figure) that is difficult to remember and not identifiable, compared with version numbers such as **V 1.0.0**. Therefore, you can tag and name important versions to easily remember and trace them. For example, tag a version as **myTag_V1.0.0** or **FirstCommercialVersion**.

```
commit 53538093c56de4df204b12ca4841926eef630bbd (tag: myTag_V1.0.0)
Author: 02_dev <[redacted]@[redacted].com>
Date: Sun Jun 28 17:40:09 [redacted]

fix #7369022 fix a bug
```

Creating a Tag for the Latest Commit on the Console

Step 1 Access the repository list.

Step 2 Click a repository to go to the details page.

Step 3 Click the **Code** and **Tags** tabs. The tag list is displayed.

Step 4 Click **Create**. In the following dialog box that is displayed, select a branch or tag.

Create Tag ✕

* Based On ?

test ▼

* Tag Name

Max. 200 bytes.

Description

Description

You can add 2000 more characters.

OK Cancel

 **NOTE**

The tag name must meet the following requirements:

- The name cannot start with a **hyphen (-)**, **period (.)**, **refs/heads/**, **refs/remotes/**, or **slash (/)**.
- Spaces and special characters such as [`<~^:?!()'"!$&`] are not supported.
- The name cannot end with a **period (.)**, **slash (/)**, or **.lock**.
- Two consecutive periods (..) are not allowed.
- The name cannot contain this sequence `@{`.

An annotated tag is generated if you enter a message (the content after `-m`). A lightweight tag is generated if you do not enter a message. For details about annotated tags, see [Tag Classification](#).

The name cannot be the same as another branch or tag name.


Step 5 Click **OK**. A tag is generated based on the latest version of the branch. The tag list is displayed.

----End

Creating a Tag for a Historical Version on the Console

Step 1 Access the repository list.

Step 2 Click a repository to go to the details page. On the **Code** tab page, click the **Files** and **History** tabs.

Step 3 In the historical commit list, click  next to a commit record and select **Create Tag**. The dialog box for creating a tag for the historical version is displayed.

 **NOTE**

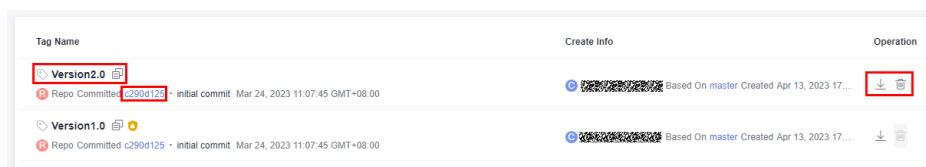
An annotated tag is generated if you enter a message (the content after **-m**). A lightweight tag is generated if you do not enter a message. For details about annotated tags, see [Tag Classification](#).





Step 4 Click **OK**. A tag is generated based on the specified historical version of the branch. The tag list is displayed.



----End

Managing Tags on the Console

- All tags in the remote repository are displayed in the tag list. You can perform the following operations:



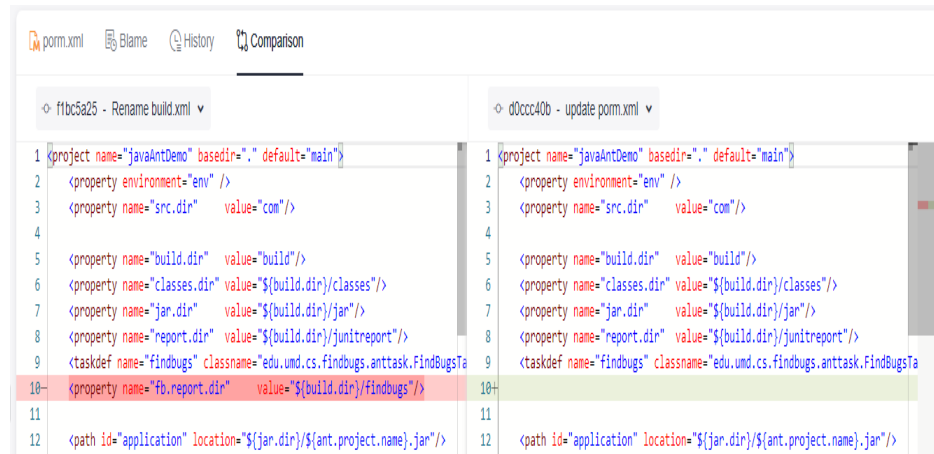
Tag Name	Create Info	Operation
Version2.0 Repo Committed: c290d125 · initial commit · Mar 24, 2023 11:07:45 GMT+08:00	Based On master Created Apr 13, 2023 17:...	 
Version1.0 Repo Committed: c290d125 · initial commit · Mar 24, 2023 11:07:45 GMT+08:00	Based On master Created Apr 13, 2023 17:...	 

- Click a tag in the **Tag Name** column to go to the file list of the tagged version.
- Click a **commit ID** to go to the commit details page.
- Click  to download the file package of the labeled version in tar.gz or zip format.
- Click  to delete a tag from CodeArts Repo. (To delete the tag from the local repository, perform the **clone**, **pull**, or **-d** operation.)

NOTICE

If an **IP address whitelist** is set for the repository, only hosts with whitelisted IP addresses can download the repository source code on the page. If no IP address whitelist is set for the repository, all hosts can download the repository source code on the page.

- You can **create a branch** based on a tag.
- On the console, click the **Files** tab and click the file name of the target file. Click the **Comparison** tab to compare commit records of the file.



Tag Classification

Git provides two types of tags:

- **Lightweight tag:** is only a reference pointing to a specific commit. It can be considered as an alias for the commit.
git tag <tag_name>

The following figure shows the information of a lightweight tag. You can find that it is an alias of a commit.

```
Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/01_developer (https)
$ git tag esay

Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/01_developer (https)
$ git show esay
commit d7dcaff34c62f0da4a2528bd1a725044b2c885f2 (HEAD -> https1, tag: esay, HTTPSOigin/master, master)
Author: <3eaf391356a7407aadbd89862@ecstest-paas-1wx6.com>
Date: Tue Jun 30 11:41:42 2023

    fix #7370149 fixtask

diff --git a/7370149fix b/7370149fix
new file mode 100644
index 0000000..76d9127
--- /dev/null
+++ b/7370149fix
@@ -0,0 +1 @@
+7370149fix
\ No newline at end of file
```

- **Annotated tag:** points to a specific commit, but is stored as a complete object in Git. Compared with lightweight tags, annotated tags contain messages (similar to code comments). In addition to the tag name and message, the tag information includes the name and email address of the person who creates the tag, and tag creation time/date.
git tag -a <tag_name> -m "<message>"

The following figure shows the information of an annotated tag, which points to a commit and contains more information than that of a lightweight tag.

```
Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/01_developer (https)
$ git tag -a name1 -m "This is my Tag for Test1"

Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/01_developer (https)
$ git show name1
tag name1
Tagger: 01_dev <74105@ecstest-paas-1wx6.com>
Date: Tue Jun 30 20:03:54 2023

This is my Tag for Test1

commit d7dcaff34c62f0da4a2528bd1a725044b2c885f2 (HEAD -> https1, tag: name1, tag: esay, HTTPSOigin/master, master)
Author: <3eaf391356a7407aadbd89862@ecstest-paas-1wx6.com>
Date: Tue Jun 30 11:41:42 2023

    fix #7370149 fixtask

diff --git a/7370149fix b/7370149fix
new file mode 100644
index 0000000..76d9127
--- /dev/null
+++ b/7370149fix
@@ -0,0 +1 @@
+7370149fix
\ No newline at end of file
```

 NOTE

Both types of tags can identify versions. **Annotated tags** contain more information and are stored in a more stable and secure structure in Git. They are more widely used in large enterprises and projects.

Common Git Commands for Tags

- **Creating a lightweight tag**

```
git tag <tag_name> # Add a lightweight tag to the latest commit.
```

Example:

```
git tag myTag1 # Add a lightweight tag myTag1 to the latest commit.
```

- **Creating an annotated tag**

```
git tag -a <tag_name> -m "<message>" # Add an annotated tag to the latest commit.
```

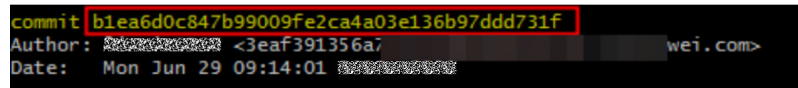
Example:

```
git tag -a myTag2 -m "This is a tag." # Add an annotated tag myTag2 to the latest commit, and the message is "This is a tag."
```

- **Tagging a historical version**

You can also tag a historical version by running the **git log** command to obtain the commit ID of the historical version. The following uses an annotated tag as an example:

```
git log # The historical commit information is displayed. Obtain the commit ID (only the first several digits are required), as shown in the following figure. Press q to return.
```



```
commit: b1ea6d0c847b99009fe2ca4a03e136b97ddd731f
Author: <3eaf391356a7> <3eaf391356a7@wei.com>
Date: Mon Jun 29 09:14:01 2023
```

```
git tag -a historyTag -m "Tag a historical version." 6a5b7c8db # Add tag historyTag to the historical version whose commit ID starts with 6a5b7c8db, and the message is "Tag a historical version."
```

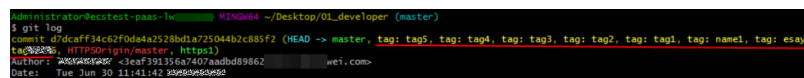
 NOTE

- If no command output is displayed, the tag is successfully created. If the command output is displayed, indicating that the tag name already exists (as shown in the following figure), change the tag name and perform the operation again.



```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/01_developer (master)
$ git tag tag1
fatal: tag 'tag1' already exists
```

- One commit can have multiple tags with unique names, as shown in the following figure.



```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/01_developer (master)
$ git log
commit d9dcaff34c62f0d44a2528bd1a725044b2c85f2 (HEAD -> master, tag: tag5, tag: tag4, tag: tag3, tag: tag2, tag: tag1, tag: name1, tag: esay, tag: tag2)
Author: <3eaf391356a7407aadb89862> <3eaf391356a7407aadb89862@wei.com>
Date: Tue Jun 30 11:41:43 2023
```

- **Viewing tags in the local repository**

You can list all tag names in the current repository and add parameters to filter tags when using them.

```
git tag
```

- **Viewing details about a specified tag**

```
git show <name_of_the_desired_tag>
```

Example:

Display the details about **myTag1** and the commit information. The following shows an example command output:

```
git show myTag1
```



```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git show myTag1
tag myTag1
Tagger: 01_dev <74105@ecstest-paas-lw.com>
Date: 2023-09-05 10:10:10 +0800

This is a tag for show you~!

commit 53538093c56de4df204b12ca4841926eef630bbd (tag: myTag1)
Author: 02_dev <yuhu@ecstest-paas-lw.com>
Date: 2023-09-05 10:10:10 +0800

    fix #7369022 fix a bug

diff --git a/file01 b/file01
index e0af0bd..b3b2032 100644
--- a/file01
+++ b/file01
```

- **Pushing a local tag to the remote repository**

- By default, tags are not pushed when you push files from the local repository to the remote one. Tags are automatically synchronized when you synchronize (clone or pull) content from the remote repository to the local one. Therefore, if you want to share local tags with others in the project, you need to run the following Git command separately.

```
git push <remote_repository_address_or_alias> <name_of_the_tag_to_be_pushed> # Push the specified tag to the remote repository.
```

Example:

Push the local tag **myTag1** to the remote repository whose alias is **origin**.

```
git push origin myTag1
```

- Run the following command to push all new local tags to the remote repository:

```
git push <remote_repository_address_or_alias> --tags
```

📖 NOTE

If you create a tag in the remote repository and a tag with the same name in the local repository, the tag will fail to be pushed due to the conflict. In this case, you need to delete one of the tags and push another tag again.

You can view all tags in the remote repository by referring to [Managing Tags on the Console](#).

- **Deleting a local tag**

```
git tag -d <name_of_the_tag_to_be_deleted>
```

The following shows an example of deleting the local tag **tag1**.

```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git tag -d tag1
Deleted tag 'tag1' (was d7dcaff)
```

- **Deleting a tag from the remote repository**

Similar to tag creation, tag deletion also needs to be manually pushed.

```
git push <remote_repository_address_or_alias> :refs/tags/<name_of_the_tag_to_be_deleted>
```

The following shows an example of deleting a tag.

```
git push HTTPSorigin :refs/tags/666 # Delete the tag 666 from the remote repository whose alias is HTTPSorigin.
```

```
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (master)
$ git push HTTPSorigin :refs/tags/666
To https://ecstest-paas-lw.com:8080/01_developer.git
- [deleted] 666
```

Obtaining a Historical Version Using Tags

If you want to view the code in a tagged version, you can check it out to the working directory. The code can be edited but cannot be added or committed because the checked-out version belongs only to a tag instead of a branch. You can create a branch based on the working directory, modify the code on the branch, and merge the branch into the master branch. The detailed steps are as follows:

1. Check out a historical version using a tag.

```
git checkout V2.0.0 # Check out the version tagged with V2.0.0 to the working directory.
```

```
MINGW64 /d/403 (master)
$ git checkout V2.0.0
Note: switching to 'V2.0.0'.
```

2. Create a branch based on the current working directory and switch to it.

```
git switch -c forFixV2.0.0 # Create a branch named forFixV2.0.0 and switch to it.
```

```
MINGW64 /d/403 ((V2.0.0))
$ git switch -c forFixV2.0.0
Switched to a new branch 'forFixV2.0.0'
```

3. (Optional) If the new branch is modified, commit the changes to the repository of the branch.

```
git add . # Add the changes to the staging area of the new branch.
git commit -m "fix bug for V2.0.0" # Save the changes to the repository of the branch.
```

```
$ git add .
MINGW64 /d/403 (forFixV2.0.0)
$ git commit -m "fix bug for V2.0.0"
remote: [forFixV2.0.0 72cce88] fix bug for V2.0.0
Committer: 
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:
```

4. Switch to the master branch and merge the new branch (**forFixV2.0.0** in this example) to the master branch.

```
git checkout master # Switch to the master branch.
git merge forFixV2.0.0 # Merge the changes based on the historical version into the master branch.
```

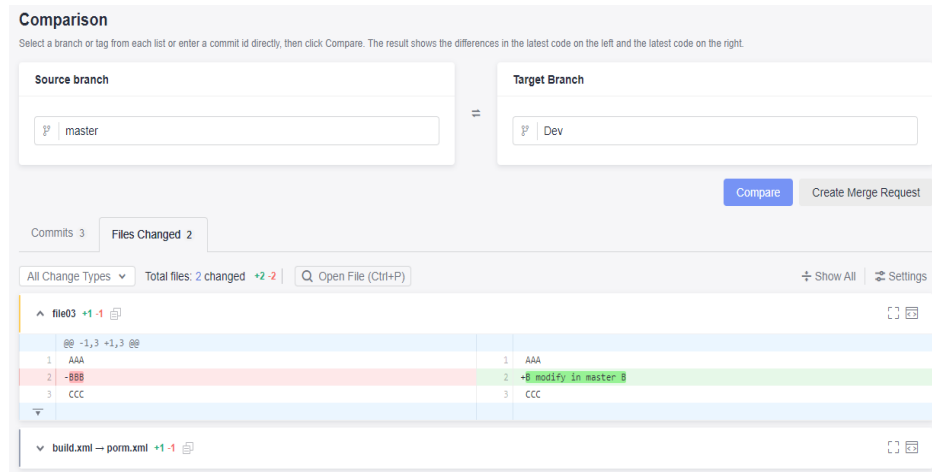
```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
MINGW64 /d/403 (master)
$ git merge forFixV2.0.0
remote: 
Merge made by the 'recursive' strategy.
 images.PNG | Bin 0 -> 109319 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 images.PNG
```

NOTE

The preceding commands are used to help you understand how to obtain a historical version using a tag. Omit or add Git commands as required.

8.4.5 Managing Comparison

Click the **Code** and **Comparison** tabs of the repository details page, you can view the code changes between branches or between tag versions through comparison.



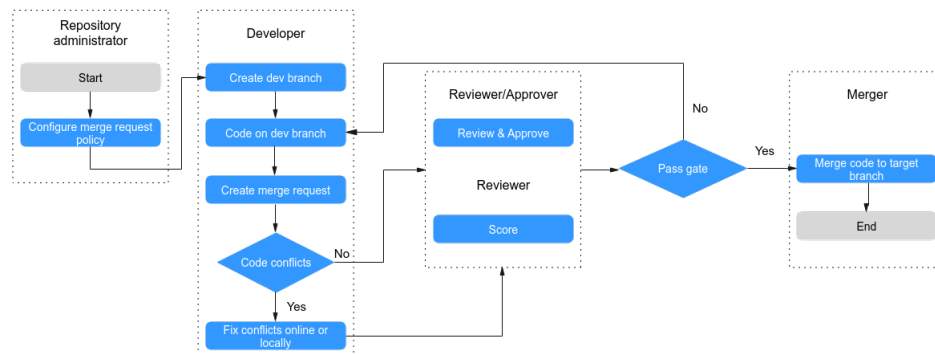
NOTE

After comparing branches, you can [create a merge request](#) as required.

8.5 Managing MRs

8.5.1 Managing MRs

CodeArts Repo supports development of multiple [branches](#) and establishes configurable review rules for branch merging. When a developer initiates an MR, some repository members can be selected to participate in code review to ensure the correctness of the merged code.



NOTE

When a merge request is created, reviewers, approvers, and mergers will be notified by emails and .

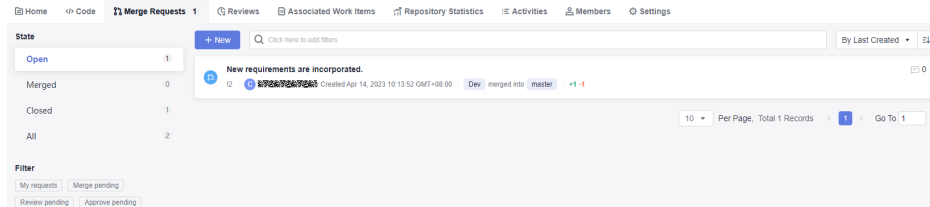
Based on the security of the code repository, you are advised to understand and configure the following functions before using merge requests:

- [9.3.4 Merge Requests](#): You can set rules for merging branches.
- [9.3.1 Protected Branches](#) describes how to configure the merge permission on a protected branch.

Merge Request List

On the **Merge Requests** tab page, you can view merge requests list page.

- You can switch between tabs to view MRs in different states.
- You can click a request to go to the details page.
- You can view the brief information about the request, including the involved branch, creation time, and creator.
- You can search for a request based on different conditions.
- You can click **New** in the upper left corner to create a request.



NOTE

Open: The request has entered the review or merge phase, and branches have not been merged.

Merged: indicates that the request is approved and the branch is merged.

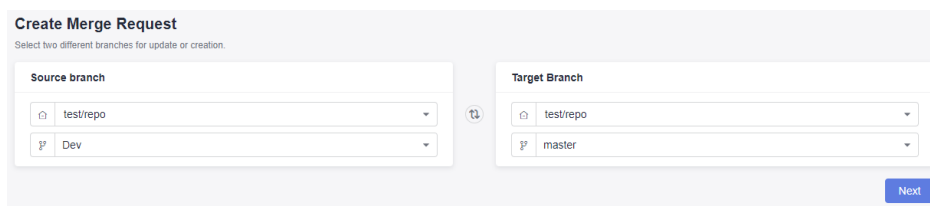
Closed: indicates that the request is canceled and the branch is not merged.

All: displays MRs in all states.

Creating a Merge Request

Assume that the administrator has set **branch merge rules**. To create an MR for a develop branch, perform the following steps:

- Step 1** Go to the details page of a target repository.
- Step 2** Switch to the **Merge Requests** tab page.
- Step 3** Click **New** and select the branch to be merged.



In the preceding figure, **Dev** (where the development task is completed) is merged into the **master** branch.

NOTE

The branch of a forked repository can be selected as the source branch.

- Step 4** Click **Next**. The system checks whether the two branches are different.
 - If there is no difference between the two, the system displays a message and the MR cannot be created.
 - If the branches are different, the following **Create Merge Request** page is displayed.

Create Merge Request

From `Scrum007/repo | Dev` into `Scrum007/repo | master` [Change Branch](#)

Title
Enter a title
Add [WIP] to the title, to prevent a Work In Progress (WIP) merge request from being merged before it is ready.

Description
merge "Dev" into "master"
Create File Function_1,
Create File Function_2
72/5000

Tip
Directly edit a work item in the associated work item. You can also use keywords fix, fixed, resolve, resolved, and close plus a number sign (#) in the description to associate with a work item. For example, fix #IR20230202018492 fix a bug.
To set work item status and transition, go to "automatic transition", To set E2E tracing for integration, go to "E2E Settings".

Associated Work Items
Enter a work item ID or keyword.

[Create Merge Request](#) [Cancel](#)

Commits 2 **Files Changed 2**

Create File Function_1
Created Apr 19, 2023 10:09:43 GMT+08:00

Create File Function_2
Created Apr 19, 2023 10:09:55 GMT+08:00

The lower part of the **Create Merge Request** page displays the file differences of the two branches and the commit records of the source branch.

Step 5 Set the parameters according to the following table.

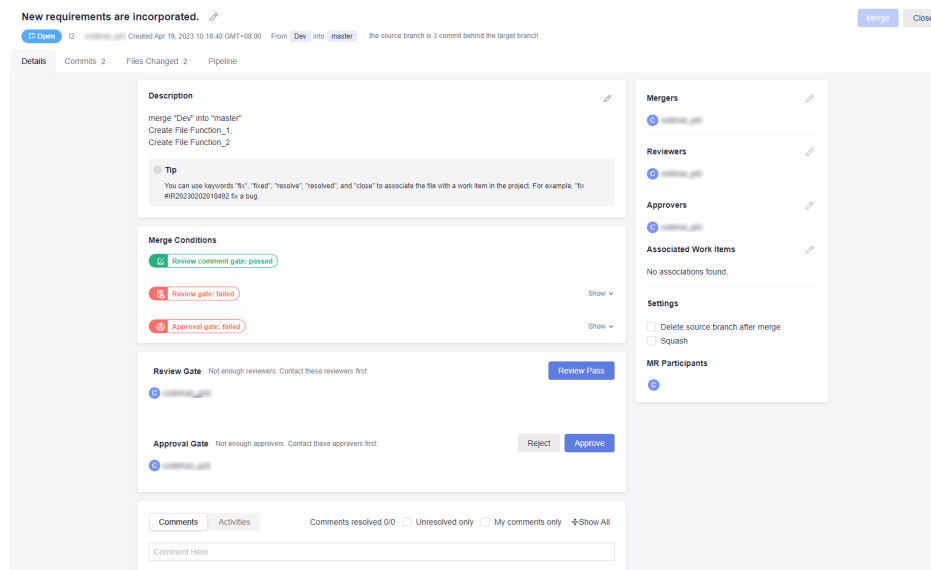
Table 8-5 Parameter description

Parameter	Description
Change Branch	Click to return to the previous step and change the branch to be merged.
Title	Enter the MR title.
Description	A default description is generated based on the merge and commit messages of the source branch. You can modify the description as required.
Associate Work Items	You can choose to associate a merge action with a work item to automatically change the status of the work item.
Mergers	Mergers have permissions to merge branches (by clicking the merge button) when all approvers approve MRs and all discussed issues are solved (or you can set the rule to allow merge with issues unsolved). They can also close the MR.
Reviewers	Specified to participate in the merge branch review and can raise questions to the initiator.

Parameter	Description
Approvers	Appointed to participate in the merge branch review. You can provide review comments (approved or rejected) or raise questions to the initiator.
Delete source branch after merge	You can choose whether to delete the source branch after merge. The preset status in the MR settings is initially used.
Squash	Enabling Squash merge keeps the history of the basic branch clean, with meaningful commit messages, and can be easily restored if necessary. For details, see Squash .

Step 6 Click **Create Merge Request** to submit the MR. The details page is displayed.

On the details page, merge rule statuses, mergers, reviewers, approvers, and associated work items are displayed. You can view review comments, mark a review comment as **Unsolved**, and view all activities related to the merge request.



- **Commits:** You can view commit records of the source branch.
- **Files Changed:** You can view the changed content in an MR and filter the change types such as addition, modification, deletion, and renaming.
- **Pipeline:** You can view the information about the pipeline.

----End

NOTE

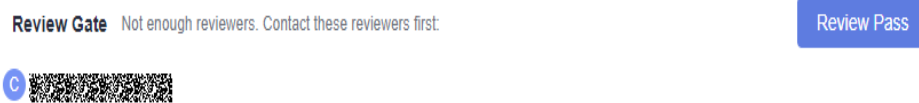
- When an MR is created, related members (reviewers and mergers) will be notified by emails. The reviewer cannot be the creator of the merge request.
- If a single file contains 5000 different lines and there are over 100 different files, you are advised to merge the branch using the client and then push it to CodeArts Repo.

Reviewing, Approving, and Merging MRs

If you are notified of an MR as a reviewer, approver, or merger, perform the following steps:

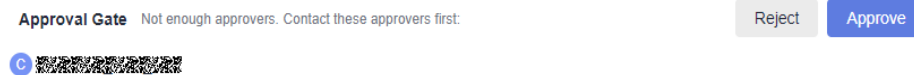
- Step 1** Go to the details page of a target repository.
- Step 2** Switch to the **Merge Requests** tab and click the name of the target merge request to view details.
- Step 3** Review the target merge request.

Both the reviewer and approver can review the merge request and provide review comments. If there is no comment, the reviewer can click **Review Pass** to complete the review.



- Step 4** Approve the target merge request.

The reviewer can click **Reject** or **Approve**.



- Step 5** Pass the gate.

Table 8-6 Merge conditions

Merge Condition	Description
Code merge conflicts	When the source branch code conflicts with the target branch code, you need to resolve the conflict before proceeding to the next step. For details about how to resolve the code conflict, see Resolving Code Conflicts in an MR .
Review comment gate	After the initiator resolves the reviews of all reviewers or approvers, the gate is passed. For details see Detailed Description of Review Comments Gate .
Pipeline gates	When the latest commit or pre-merged commit starts and successfully executes the pipeline, the gate is passed. For details see Detailed Description of Pipeline Gate .
E2E ticket number not associated	After the combination request is associated with a work item, the gate is passed. For details see Detailed Description of E2E Ticket Number Association Gate .
Review gate	When the number of reviewers reaches the minimum number, the gate is passed. For details see Detailed Description of Review Gate .

Merge Condition	Description
Approval gate	When the number of approvers reaches the minimum number, the gate is passed. For details see Detailed Description of Approval Gate .

Step 6 Merge the request.

After an initiator meets the preceding conditions, click **Merger** to merge the request. Otherwise, click **Close** to close the request.

----End

Squash

Squash is to merge all change commit information of an MR into one and keep a clean history. When you focus only on the current commit progress rather than the commit information, you can use squash.

 **NOTE**

If **Squash** is selected, multiple consecutive change records of the source branch can be merged into one commit record (information of **Configure Squash**), and this new commit record can be committed to the target branch.

- If the change history of the merge request contains only one commit, the commit record in the target branch is for the source branch after **Squash** is selected.
- If the change history of the merge request contains multiple commits, the commit record in the target branch contains the information of **Configure Squash** after **Squash** is selected.

To better understand this function, perform the following operations:

Step 1 [Create a repository](#).

Name it **repo**.

Step 2 [Create a branch](#).

Name it **Dev**.

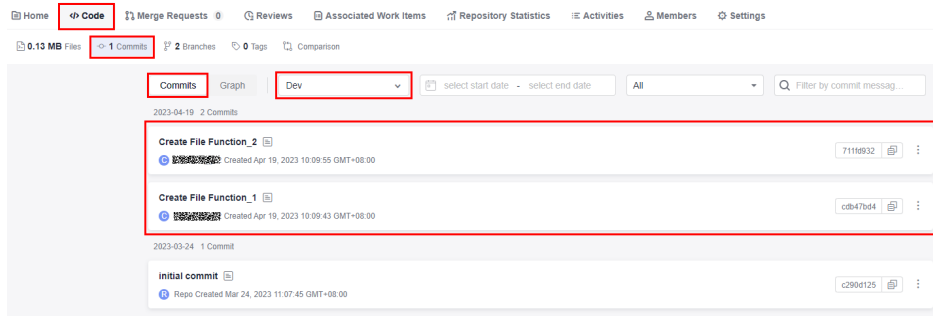
Step 3 Submit the creation.

Take [creating a file](#) as a commit.

Dev branch: Create two files and name them **Function_1** and **Function_2**.

Step 4 Check the effect before Squash is enabled.

Find the **Dev** branch. Click the **Code**, **Commits**, and **Commits** tabs to view the commit information.



Step 5 Create and merge a request.

1. Set the source branch to **Dev** and target branch to **master**. **Create a merge request**.

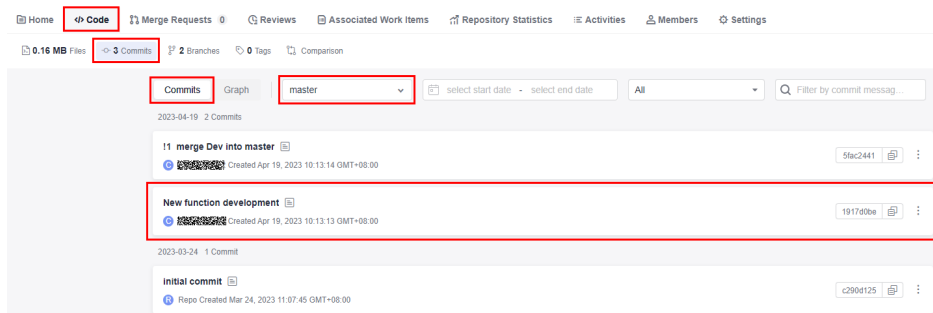
Dev branch: Name the merge request as **Code Merge**, select **Squash**, and enter **Configure Squash**.



2. Complete **the review and approval**.

Step 6 Check the effect after **Squash** is enabled.

After the request is successfully merged, click the **Code**, **Commits**, and **Commits** tabs, select the **master** branch. Compared with **Step 4**, the committed content has been merged.



----End

8.5.2 Resolving Code Conflicts in an MR

When using CodeArts Repo, you may encounter the situation where two members in the same team modify a file at the same time. Code fails to be pushed to a CodeArts Repo repository due to the code commit conflict. The following figure shows a push failure caused by the file change conflict in the local and remote repositories.

```
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/02_developer/ (master)
$ git push
To [redacted]:.git
 ! [rejected] master -> master (fetch first)
error: failed to push some refs to '[redacted]':
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/02_developer/ (master)
$
```

NOTE

- The returned messages vary depending on Git versions and compilers but have the same meaning.
- The information similar to "push failure" and "another repository member" in the returned message indicates that there is a commit conflict.
- Git automatically merges changes in different lines of the same file. A conflict occurs only when the same line of the same file is modified (the current version of the local repository is different from that of the remote repository).
- Conflicts may occur during branch merge. The locating method and solution are basically the same as those for the conflict during the commit to the remote repository. The following figure shows that a conflict occurs when the local **branch1** is merged into the master branch (due to the changes in the **file01** file).

```
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/02_developer/ (master)
$ git merge branch1
Auto-merging file01
CONFLICT (content): Merge conflict in file01
Automatic merge failed; fix conflicts and then commit the result.
```

Resolving a Code Commit Conflict

To resolve a code commit conflict, pull the remote repository to the working directory in the local repository. Git will merge the changes and display the conflicting file content that cannot be merged. Then, modify the conflicting content and push it to the remote repository again (by running the **add**, **commit**, and **push** commands in sequence).

The following figure shows that there is a file merge conflict when you run the **pull** command.

```
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/02_developer/ (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), 321 bytes | 14.00 KiB/s, done.
From [redacted]:com:
 9c5d50b..54848ef master -> origin/master
Auto-merging file01
CONFLICT (content): Merge conflict in file01
Automatic merge failed; fix conflicts and then commit the result.
```

Modify the conflicting file carefully. If necessary, negotiate with the other member to resolve the conflict and avoid overwriting the code of other members by mistake.

NOTE

git pull combines **git fetch** and **git merge**. The following describes the operations in detail.

```
git fetch origin master # Pull the latest content from the master branch of the remote host.  
git merge FETCH_HEAD # Merge the latest content into the current branch.
```

During merge, a message indicating that the merge fails due to a conflict is displayed.

Example: Conflict Generation and Resolution

The following shows an example to help you understand how a conflict is generated and resolved.

A company uses CodeArts Repo and Git to manage a project. A function (the **file01** file is modified) of the project is jointly developed by developer 1 (01_dev) and developer 2 (02_dev). The two developers encounter the following situation.

1. **file01** is stored in the remote repository. The following shows the file content.

```
file01  
1  ##file01AAAAAAAAAAAAA  
2  ##file02BBBBBBBBBBBBB  
3  ##file03CCCCCCCCCCCCC  
4  ##file04DDDDDDDDDDDD  
5  |
```

2. 01_dev modifies the second line of **file01** in the local repository and successfully pushes the file to the remote repository. The following shows the file content in the local and remote repositories of 01_dev.

```
file01  
1  ##file01AAAAAAAAAAAAA  
2  ##modify by 01_dev  
3  ##file03CCCCCCCCCCCCC  
4  ##file04DDDDDDDDDDDD  
5  ## add one line by 01_dev |
```

3. 02_dev also modifies the second line of **file01** in the local repository. When 02_dev pushes the file to the remote repository, a conflict message is displayed. The following shows the file content in the local repository of 02_dev, which is conflicting with that in the remote repository.

```
##file01AAAAAAAAAAAAA  
## modify by 02_dev  
##file03CCCCCCCCCCCCC  
##file04DDDDDDDDDDDD  
## add by 02_dev
```

- 02_dev pulls the code in the remote repository to the local repository, detects the conflict starting from the second line of the file, and immediately contacts 01_dev to **resolve the conflict**.
- We find that they both modified the second line and added content to the last line, as shown in the following figure. Git identifies the content starting from the second line as a conflict.

```
##file01AAAAAAAAAAAAA
<<<<<< HEAD
## modify by 02_dev      modify by 02_dev
##file03CCCCCCCCCCCCC
##file04DDDDDDDDDDDDD
## add by 02_dev

=====
##modify by 01_dev      modify by 01_dev
##file03CCCCCCCCCCCCC
##file04DDDDDDDDDDDDD
## add one line by 01 dev      commit ID
>>>>>> af5daac097230b2f8f
```

NOTE

Git displays the changes made by the two developers and separates them using =====.

- The content between <<<<<<HEAD and ===== indicates the changes of the local repository in the conflicting lines.
 - The content between ===== and >>>>>> indicates the changes of the remote repository in the conflicting lines, that is, the pulled content.
 - The content after >>>>>> is the commit ID.
 - Delete <<<<<<HEAD, =====, >>>>>>, and commit ID when resolving the conflict.
- The two developers agree to retain all changes after discussion. After 02_dev modifies the content, the modified and added lines are saved in the local repository of 02_dev, as shown in the following figure.

```
##file01AAAAAAAAAAAAA
## modify by 02_dev
##modify by 01_dev
##file03CCCCCCCCCCCCC
##file04DDDDDDDDDDDDD
## add by 02_dev
## add one line by 01_dev
```

- 02_dev pushes the merged changes to the remote repository (by running **add**, **commit**, and **push** commands in sequence). The following shows the file content in the remote repository after a successful push. The conflict is resolved.

```
file01
```

```
1  ##file01AAAAAAAAAAAAA
2  ## modify by 02_dev
3  ##modify by 01_dev
4  ##file03CCCCCCCCCCCCC
5  ##file04DDDDDDDDDDDDDD
6  ## add by 02_dev
7  ## add one line by 01_dev
```

NOTE

In the preceding example, TXT files are used for demonstration. In the actual situation, the conflict display varies in different text editors and Git plug-ins of programming tools.

Preventing a Conflict

Repository preprocessing before code development can prevent commit and merge conflicts.

In [Example: Conflict Generation and Resolution](#), 02_dev successfully resolves the conflict in the commit to the remote repository. For 02_dev, the latest code version of the local repository is the same as that of the remote repository. For 01_dev, version differences still exist between the local and remote repository. A conflict will occur when 01_dev pushes code to the local repository. The following describes methods to resolve the conflict.

Method 1 (recommended for beginners):

If your local repository is not frequently updated, clone the remote repository to the local repository to modify code locally, and commit the changes. This directly resolves the version differences. However, if the repository is large and there are a large number of update records, the clone process will be time-consuming.

Method 2:

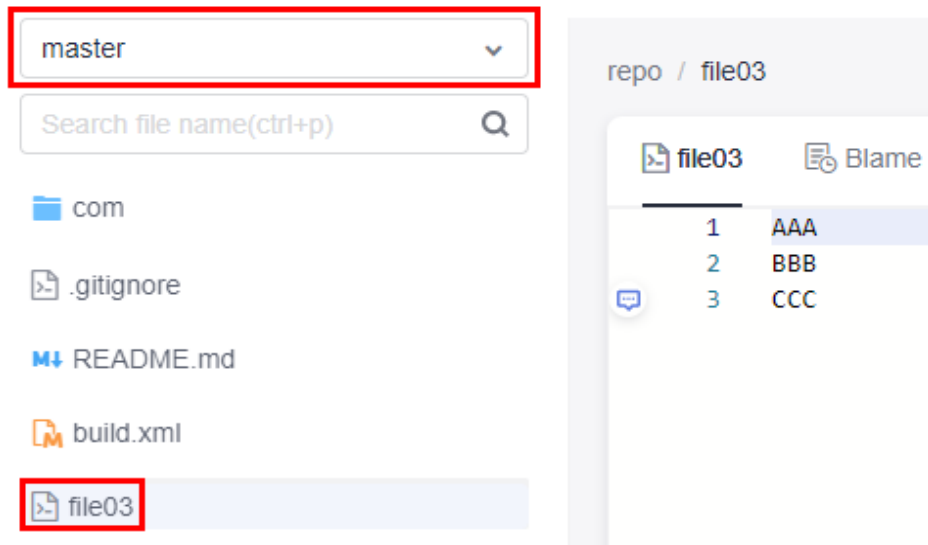
If you modify the local repository every day, create a develop branch in the local repository for code modification. When committing code to the remote repository, switch to the master branch, pull the latest content of the master branch in the remote repository to the local repository, merge the branches in the local repository, and resolve the conflict. After the content is successfully merged into the master branch, commit it to the remote repository.

Resolving a Merge Conflict on the Console

CodeArts Repo allows you to [manage branches](#). The following simulates a conflicting MR and describes how to resolve it.

Step 1 [Create a repository](#).

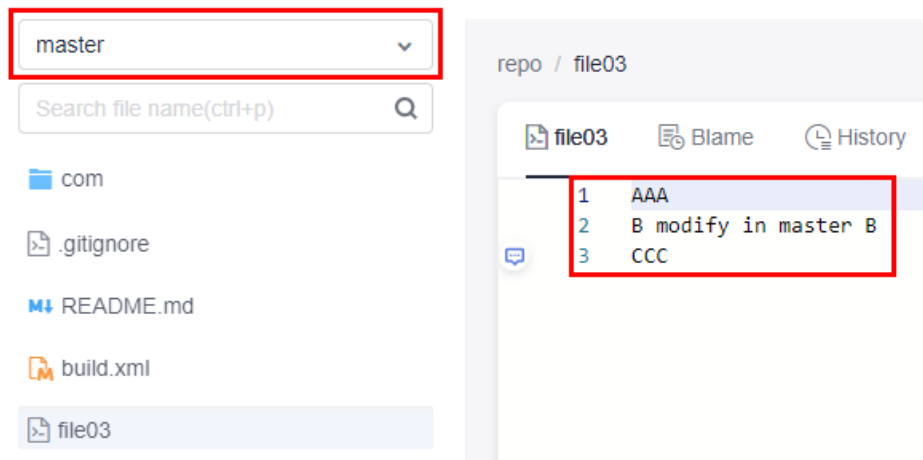
Step 2 [Create a file](#) named **file03** on the master branch in the repository. The initial content is as follows:



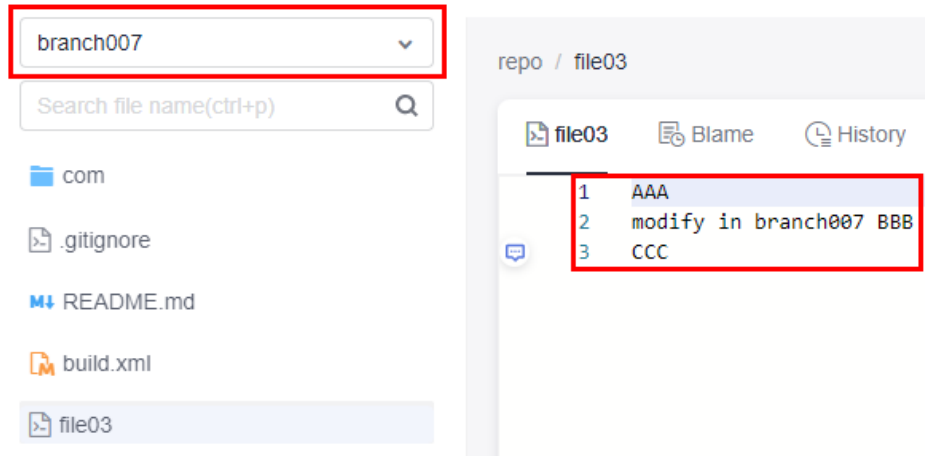
Step 3 Create a branch named **branch007** based on the master branch.

The content in the master branch is the same as that in **branch007**. The following describes how to make them different.

Step 4 In the master branch, modify **file03** as shown in the following figure, and enter the commit message **modify in master**.

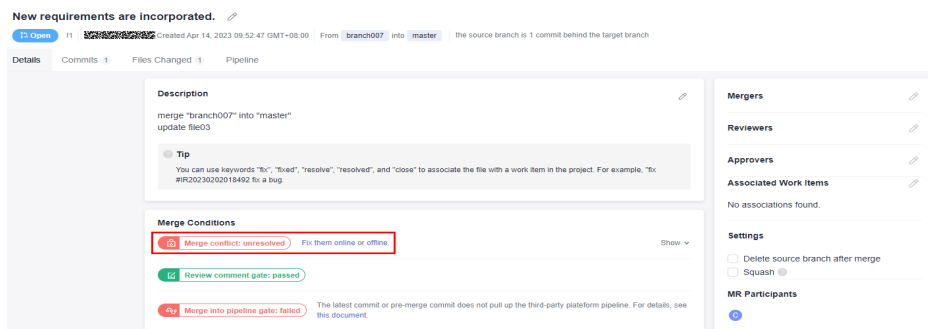


Step 5 Switch to **branch007**, modify **file03** as shown in the following figure, and enter the commit message **modify in branch007**. Then the two branches are different, that is, a conflict occurs.



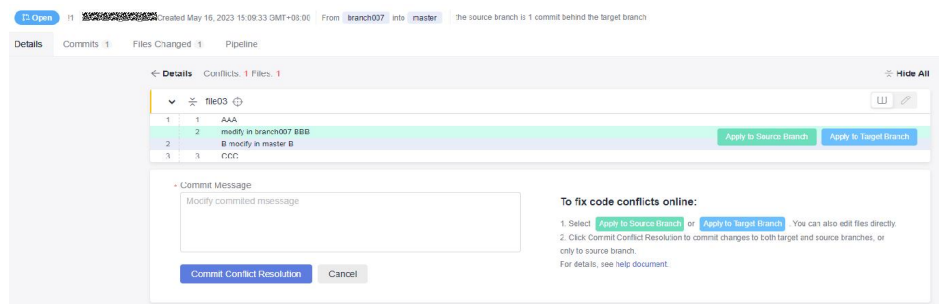
Step 6 Create an **MR** to merge **branch007** to the master branch. Click **Create Merge Request** to submit the MR.


Merge request details page is displayed. You can also click the name of the merge request in the merged requests list to access this page. **Merge conflict: unsolved** displays on the details page. You are advised to **Fix them online or offline**.

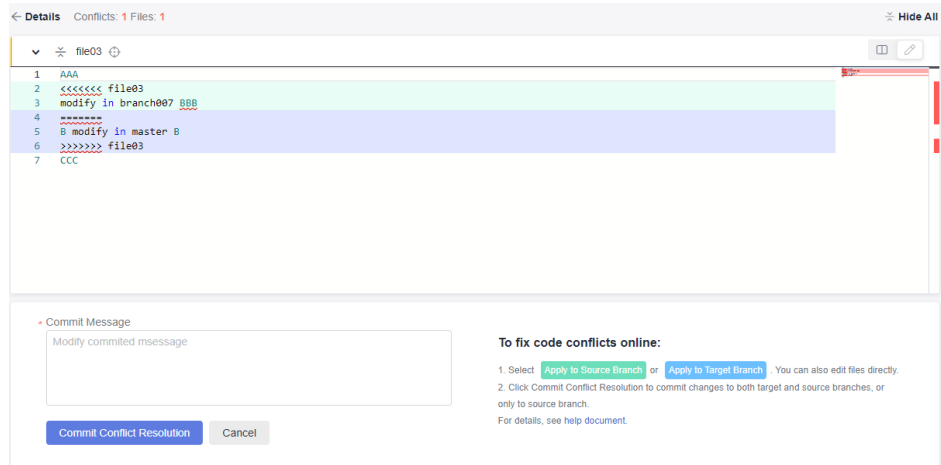


Step 7 Perform the following operation to resolve the conflict:

- **Fix them online** (recommended for small code volume)
 - a. Click **Fix them online**. The following page is displayed, showing the code conflict.



- b. If the conflict cannot be resolved by overwriting the file, click  to go to the **Manual Editing** page, as shown in the following figure. The conflict display format is similar to that in [Example: Conflict Generation and Resolution](#).



- c. Manually modify the code to resolve the conflict and commit the changes.

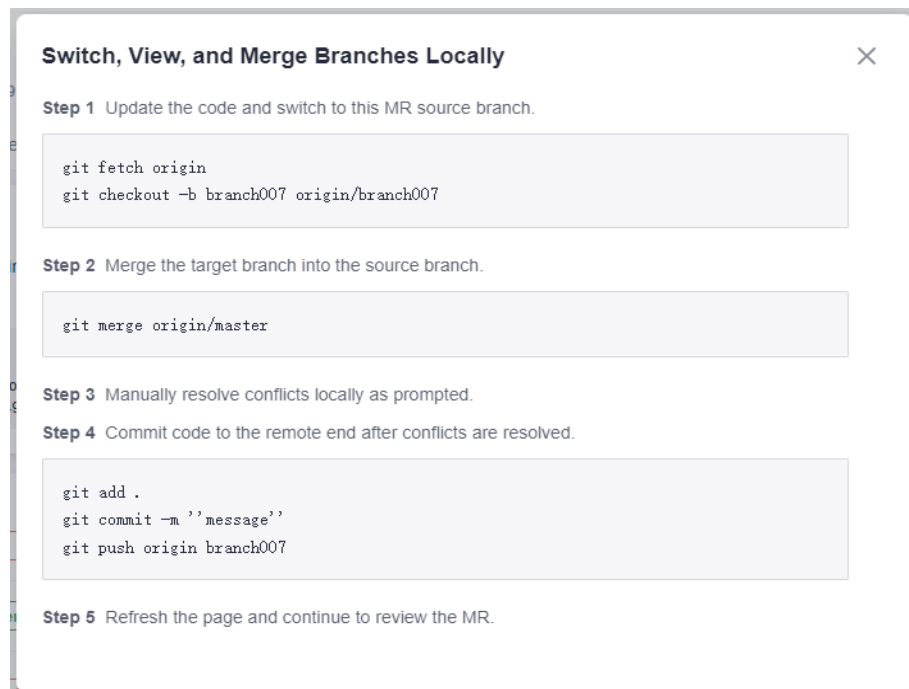
NOTE

Enter a commit message.

In the preceding figure, the following signs are used for conflict display and separation: <<<<, >>>>, and =====. Delete the lines where the signs are located when modifying code.

- **offline** (recommended for large-scale projects)

Click **offline**. The following page is displayed. Perform the operations as prompted.



NOTE

CodeArts Repo automatically generates Git commands based on your branch name. You only need to copy the commands and run them in the local repository.

Step 8 After the conflict is resolved by using either of the preceding methods, you can click **Merge** to merge branches. The system displays a message indicating that the merge is successful.

You can also follow the instructions in [8.5.1 Managing MRs](#).

Now, the content of the master and **branch007** branches is the same. You can switch between branches to check the content.

----End

8.5.3 Detailed Description of Review Comments Gate

Opening/Closing the Gate


Step 1 Go to the target repository and choose **Settings > Policy Settings > Merge Requests**.

Step 2 Configure the gate.

- Select **Merge after all reviews are resolved** and click **Submit** to save the settings. The access control is enabled.
- Deselect **Merge after all reviews are resolved** and click **Submit** to save the settings. The access control is closed.

----End

Effect of Gate Triggering

The reviewers or approvers can move the cursor to the code line in **Files Changed** of the **Merge Request** and click the  icon to add review comments. Alternatively, the reviewers or approvers can directly add review comments in **Details > Comments** of the **Merge Request**.

- **Review comment gate: passed:** It is displayed when there is no review comments in the merge request, or all review comments do not need to be resolved or have been resolved.

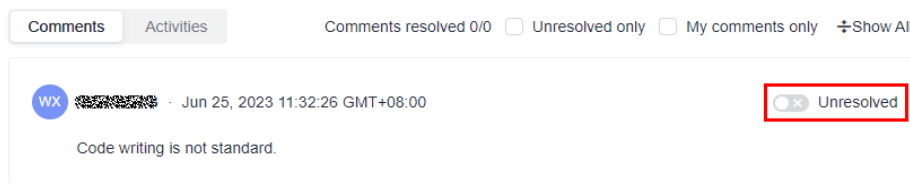


- **Review comment gate: failed:** It is displayed when the review comments in the Merge request are not resolved.



Passing of the Gate

After you have resolved the issue raised in the review comments, you can switch the status of the review comments from **Unresolved** to **Resolved** in **Details > Review Comments** of the **Merge Request**. In this case, the status of the review comments is displayed as **Review comment gate: passed**.



8.5.4 Detailed Description of Pipeline Gate

NOTE

Pipeline gate supports only merge requests whose merge mechanism is **Approval**.

Opening/Closing the Gate

Step 1 Go to the target repository and choose **Settings > Policy Settings > Merge Requests**.

Step 2 Click **Create** to set a branch policy for the target branch.

Step 3 Configure the gate.

- Select **Enable pipeline gate** under the policy and click **OK** to save the settings. The gate is enabled.
- Deselect **Enable pipeline gate** under the policy and click **OK** to save the settings. The gate is closed.

----End

Effect of Gate Triggering

- **Merge into pipeline gate: passed:** It is displayed when the pipeline is successfully started after the latest commit or pre-merge commit operation is performed.



- **Merge into pipeline gate: failed:** It is displayed when the repository has no associated pipeline task or the latest commit or pre-merge commit fails to start the pipeline.



Passing of the Gate

Step 1 Choose **CICD > Pipeline**.

Step 2 Click **Create Pipeline** and enter the following information:

- **Name:** Enter a custom name.
- **Pipeline Source:** Select **Repo**.
- **Repository:** Select the target code repository for which you want to create a merge request.

- **Default Branch:** Select the target branch of the merge request.
- Step 3** Click **Next**, select the target template as required, and click **OK**.
- Step 4** After the task is created, the system automatically switches to the **Task Orchestration** tab page in the task details and switches to the **Execution Plan** tab page.
- Step 5** Enable **Merge Request Event Triggering** and select the following trigger events based on the site requirements:
- **Create:** triggered when an MR is created.
 - **Update:** triggered when the content or setting of an MR is updated.
 - **Merge:** triggered when an MR is merged. The code submission event will also be triggered.
 - **Reopen:** triggered when an MR is reopened.
- Step 6** Configure other information about the pipeline task and click **Save**.
- Step 7** Return to the CodeArts Repo and trigger the event selected in **Execution Plan** to enable the repository to start the pipeline task.

----End

8.5.5 Detailed Description of E2E Ticket Number Association Gate

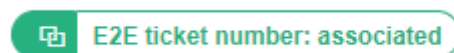
Opening/Closing the Gate

- Step 1** Go to the target repository and choose **Settings > Policy Settings > Merge Requests**.
- Step 2** Configure the Gate.
- Select **Must be associated with CodeArts Req** and click **Submit** to save the settings. The gate is enabled.
 - Deselect **Must be associated with CodeArts Req** and click **Submit** to save the settings. The gate is closed.

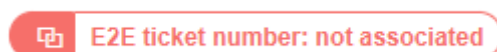
----End

Effect of Gate Triggering

- **E2E ticket number: associated:** It is displayed when the merge request is successfully associated with the work item.



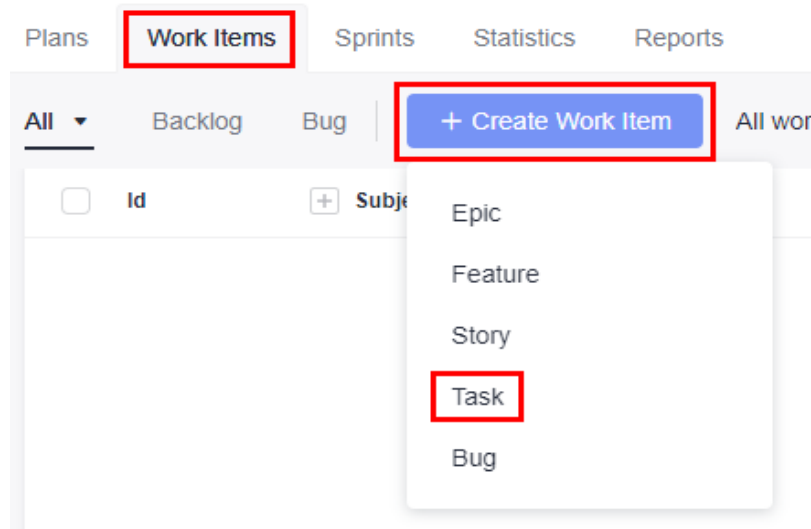
- **E2E ticket number: not associated:** It is displayed when the merge request has no associated work item.



Passing of the Gate

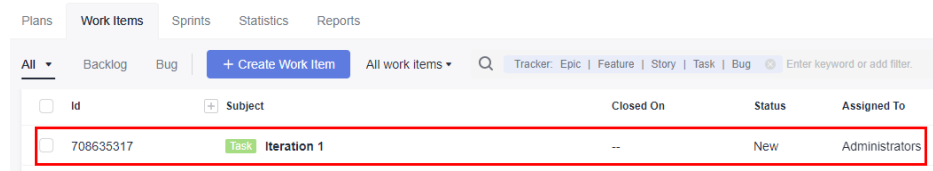
Step 1 Click the target project name to access the project.

Step 2 On the **Work Items** tab, click **Create Work Item** and choose **Task** from the drop-down list. The page for creating a work item is displayed.



Step 3 Enter a title, for example, Sprint 1.


Retain the default values for other parameters. Click **Save**.



Step 4 Choose **Code > CodeArts Repo**.

Step 5 Click a repository name to go to the target repository.

Step 6 Switch to the **Merge Requests** tab page and click the name of the target merge request to access the target merge request.

Step 7 On the **Details** page, click the  icon next to **Associated Work Items** to search for and select the target work item.

Step 8 Click **OK**. The E2E ticket number is associated.

----End

8.5.6 Detailed Description of Review Gate

NOTE

The review gate supports only the merge requests whose merge mechanism is **Approval**.

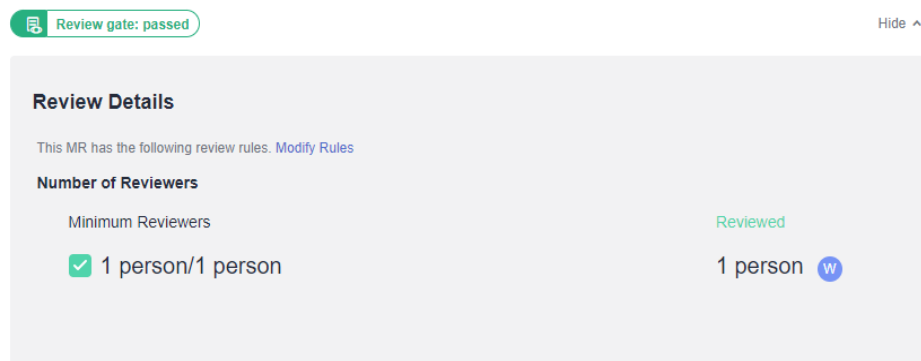
Opening/Closing the Gate

- Step 1** Go to the target repository and choose **Settings > Policy Settings > Merge Requests**.
- Step 2** Click **Create** to configure a branch policy for the target branch.
- Step 3** Configure the Gate.
 - Set **Reviewers Required** to a number except 0 and click **OK** to save the settings. The gate is enabled.
 - Set **Reviewers Required** to 0 and click **OK** to save the settings. The gate is closed.

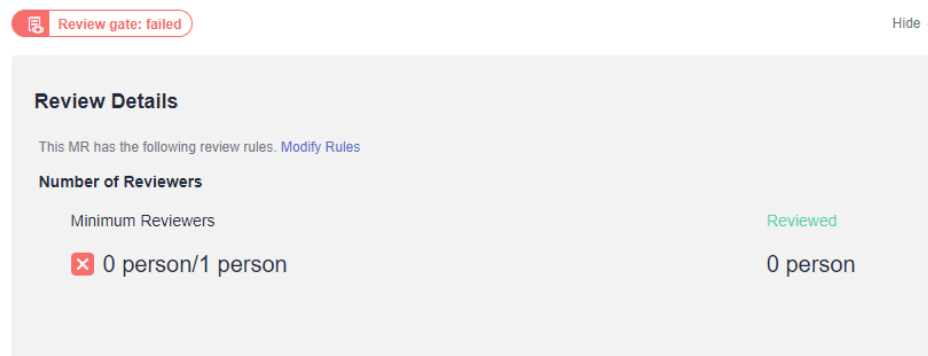
----End

Effect of Gate Triggering

- Review gate: passed:** It is displayed when the number of reviewers who give pass reaches the **Reviewers Required**.



- Review gate: failed:** It is displayed when the number of reviewers who give pass does not reach the **Reviewers Required**.



Passing of the Gate

After completing the review, the reviewer needs to choose **Details > Review Gate** and click **Pass**. The review is passed. For details, see [Setting Branch Policies](#).

8.5.7 Detailed Description of Approval Gate

NOTE

The approve gate supports only the merge requests whose merge mechanism is **Approval**.

Opening/Closing the Gate

Step 1 Go to the target repository and choose **Settings > Policy Settings > Merge Requests**.

Step 2 Click **Create** to configure a branch policy for the target branch.

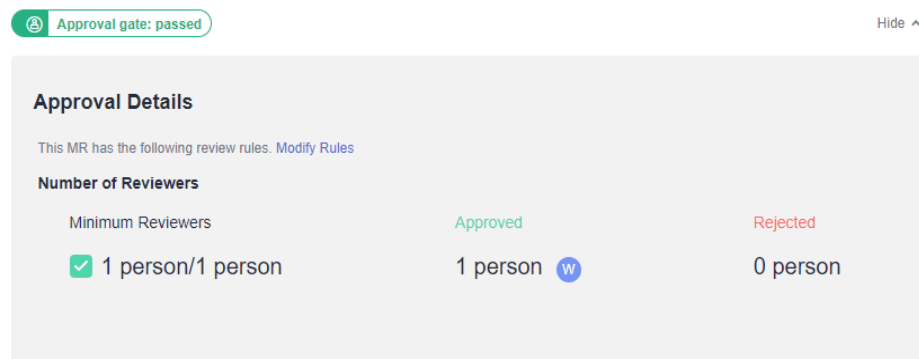
Step 3 Configure the Gate.

- Set **Approvals Required** to a number except 0 and click **OK** to save the settings. The gate is enabled.
- Set **Approvals Required** to 0 and click **OK** to save the settings. The gate is closed.

----End

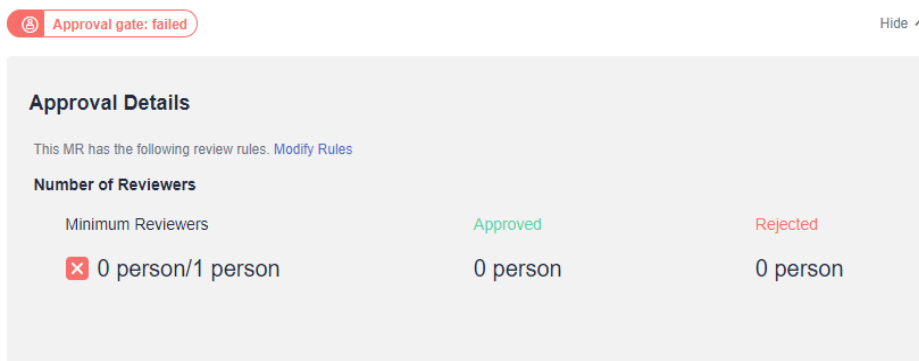
Effect of Gate Triggering

- **Approval gate: passed:** It is displayed when the number of approvers who give pass reaches the **Approvals Required**.



The screenshot shows a notification bar at the top with a green checkmark icon and the text "Approval gate: passed". To the right of the notification is a "Hide" link with a downward arrow. Below the notification is a light gray box titled "Approval Details". Inside this box, it says "This MR has the following review rules. [Modify Rules](#)". Underneath, there is a section titled "Number of Reviewers" with three columns: "Minimum Reviewers", "Approved", and "Rejected". The "Minimum Reviewers" column shows a green checkmark icon followed by "1 person/1 person". The "Approved" column shows "1 person" with a blue circle icon containing a white 'W'. The "Rejected" column shows "0 person".

- **Approval gate: failed:** It is displayed when the number of approvers who give pass does not reach the **Approvals Required**.



The screenshot shows a notification bar at the top with a red 'X' icon and the text "Approval gate: failed". To the right of the notification is a "Hide" link with a downward arrow. Below the notification is a light gray box titled "Approval Details". Inside this box, it says "This MR has the following review rules. [Modify Rules](#)". Underneath, there is a section titled "Number of Reviewers" with three columns: "Minimum Reviewers", "Approved", and "Rejected". The "Minimum Reviewers" column shows a red 'X' icon followed by "0 person/1 person". The "Approved" column shows "0 person". The "Rejected" column shows "0 person".

Passing of the Gate

After completing the approval, the approvers need to choose **Details > Approval Gate** and click **Pass**. The approval is passed. For details, see [Setting Branch Policies](#).

8.6 Viewing Review Records of a Repository

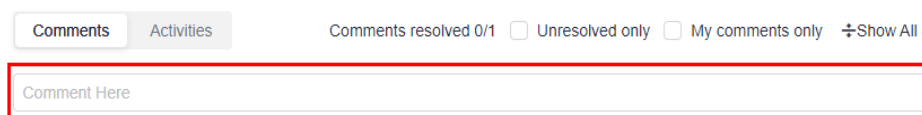
On the **Reviews** tab page of the repository details page, you can view the review information of the repository from MRs and commits. You can filter records based on the filter criteria.


Table 8-7 Review record parameters

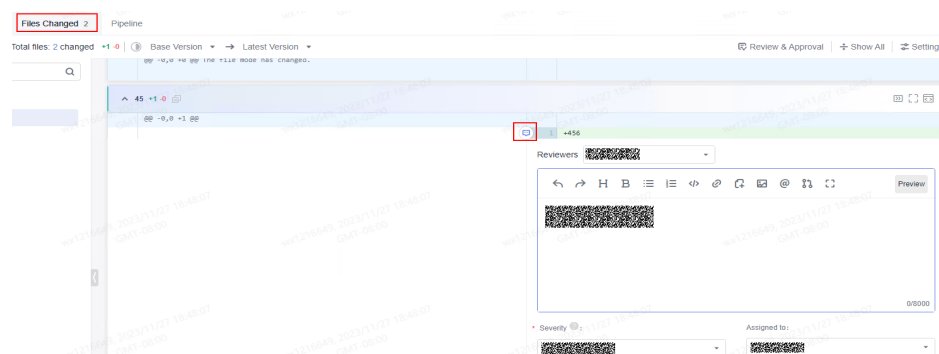
Parameter	Description
Status	Review records are classified into three statuses: Unresolved , Resolved , and Resolve Not Needed .
Review comment	Comment provided by the reviewer
Approver	Reviewer who provides the review comment
Review date	Date when the reviewer submits the review comments
Assign to	Assign the task to the default or specified personnel.

Adding Comments on the Reviews for MR Tab


Method 1: Go to the details page of the target merge request and add a comment at the bottom of the page.

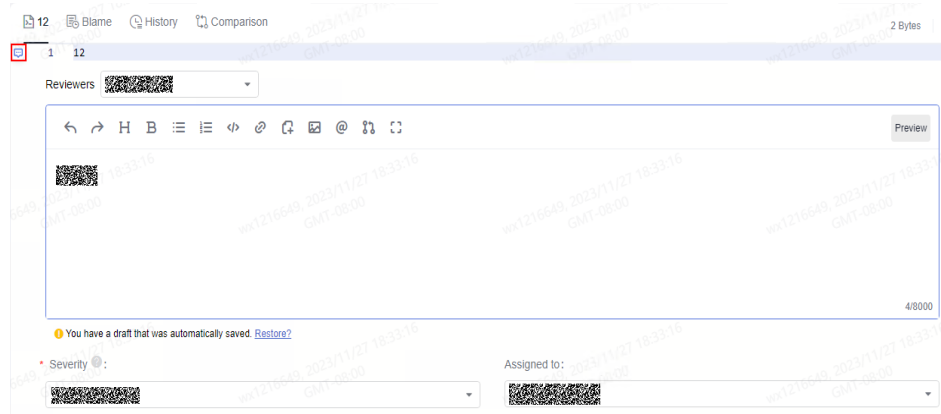


Method 2: Go to the details page of the target merge request, click **Files Changed**, and click the  icon next to a code line in the code file to add a review.

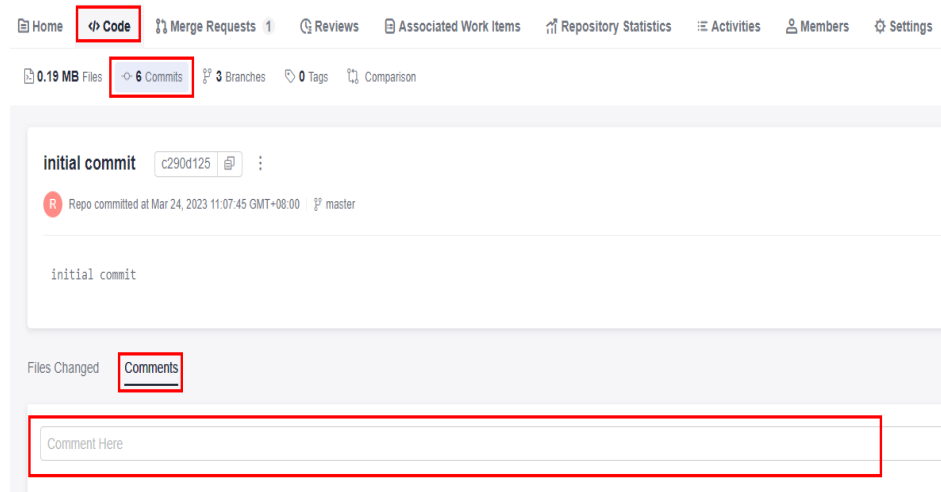



Adding Comments on the Reviews for Commit Tab

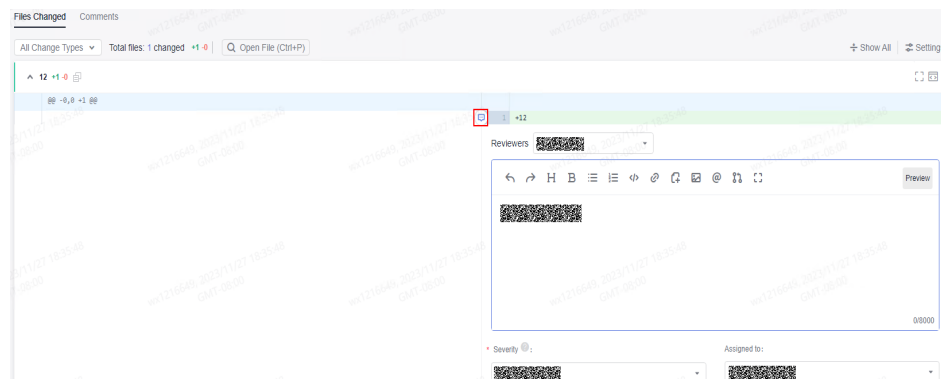
Method 1: In the code file, click  next to a line of code to add review comments.



Method 2: On the **Commits** tab, click a commit to switch to the comment page and add review comments.



Method 3: On the **Commits** page, click the **Files Changed** submenu and click the  icon next to a code line to add a review.



8.7 Viewing Associated Work Items

8.7.1 Introduction

Work item is used to track work content in CodeArts Req. A work item usually has a unique ID and a description. It can be a requirement, bug, or task. In CodeArts Req, work item is a work content list that supports GUI-based management.

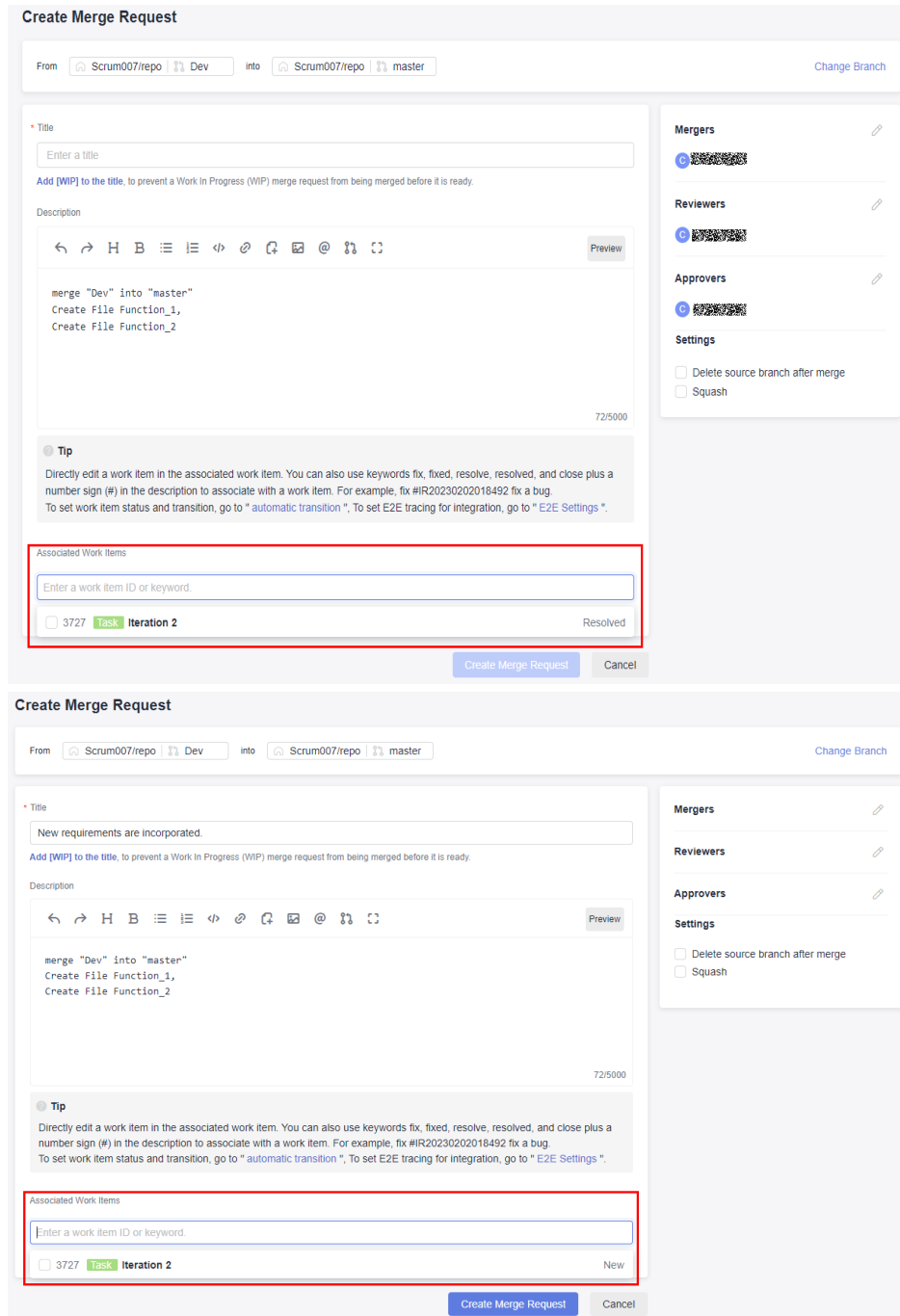
You can use the following associations and configure [E2E Tracing](#).

- **Commit association**
- Create a branch association.

You can select the target work item under **Associated Work Items** on the page for creating a branch.

The screenshot shows a 'Create Branch' dialog box. At the top, there's a title 'Create Branch' and a close button. Below that, there are three main sections: 'Based On' with a dropdown menu currently set to 'master'; 'Branch Name' with a text input field and a placeholder 'Enter a branch name. Max. 200 characters.'; and 'Description' with a text area and a placeholder 'Description'. Below these is a character count: 'Characters left: 2000 more characters.' At the bottom, there is a section titled 'Associated Work Items' which is highlighted with a red border. This section contains a dropdown menu with '--Select--', a search input field with a magnifying glass icon, and a list item '3727 Iteration 2' with an unchecked checkbox.

- Merge request association
You can select the target work item under **Associated Work Items** on the page for creating a merge request.



NOTE

CodeArts Req: a CodeArts service that provides R&D teams with efficient collaboration services. You can create multiple Agile Scrum and Lean Kanban projects to manage requirements, track bugs, create project Wiki, host documents in the cloud, analyze statistics, and manage person-hours.

Preparations

Step 1 (Optional) Configure the commit transition status.

 NOTE

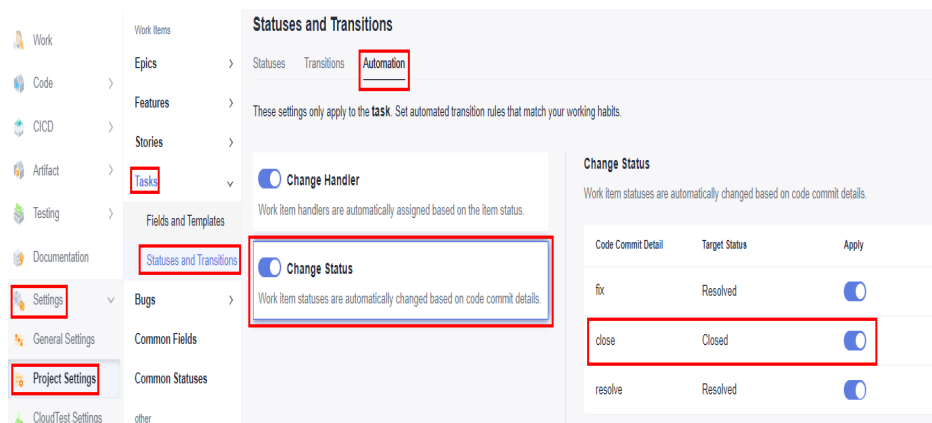
By default, the code commit status is configured as follows:


- The **fix** keyword is associated with the **Resolved** target state (enabled by default).
- The **close** keyword is associated with the **Closed** target state (disabled by default).
- The **resolve** keyword is associated with the **Resolved** target state (enabled by default).

In project settings, a project manager or another role with project setting permission can set three commit message keywords (such as **fix**, **close**, and **resolve**) for different work item types (**Epic**, **Feature**, **Story**, **Task**, and **Bug**). You can associate each keyword with a target status (for example, **Resolved** or **Closed**). The work item status can also be customized.

The following describes how to associate the **close** keyword to **Rejected** in a **Task** work item.

1. Click the target project name to access the project.
2. Find the code commit status corresponding to a task, as shown in the following figure.



3. Click the **Target Status** of **close**, set it to **Rejected**, and set **Apply** to  . The settings are automatically saved.

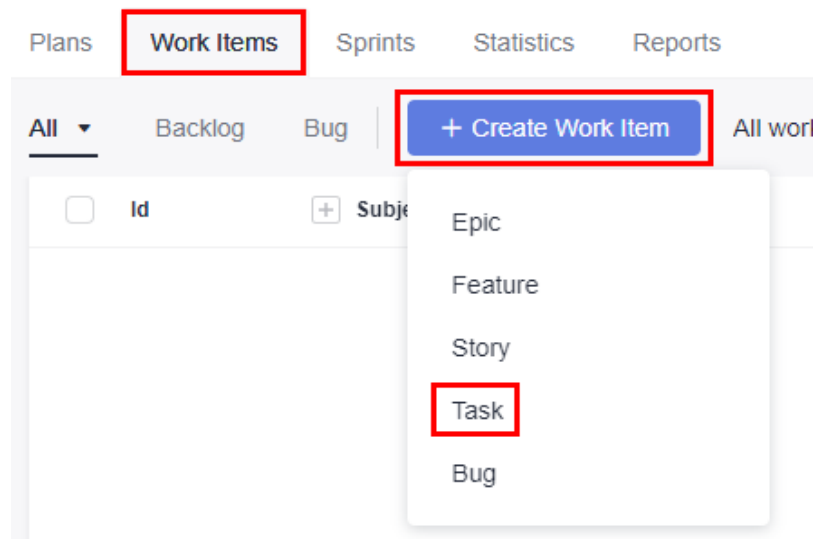
Then, you can use the **close** keyword in the commit message to change the status of a **Task** work item to **Rejected** when committing local code.

Example:

```
git commit -m "close #<task_work_item_id> <commit_message>"
```

Step 2 Create a work item.

1. Click the target project name to access the project.
2. On the **Work Items** tab, click **Create Work Item** and choose **Task** from the drop-down list. The page for creating a work item is displayed.



3. Enter a title, for example, Sprint 1.
Retain the default values for other parameters. Click **Save**.

NOTE

The work item management page is displayed. You can view the work item ID and the status is **New**.

In this example:

- The ID of **task01** is **708206208**.
- The ID of **task02** is **708206209**.

On the project homepage, choose **Work > Work Items** to obtain a work item ID.

----End

8.7.2 Commit Association

With CodeArts Repo, you can associate each code commit with a work item of CodeArts Req.

- Associated work items help developers accurately record tasks for fixing bugs and developing new features.
- Associated work items allow project managers to view information such as change committer and committed content involved in each requirement and bug fixing task.

NOTE

Commit: You can commit and save operations on files in the working directory, including creating, editing, and deleting files. The following shows the **commit** command, in which the **-m** parameter is mandatory and followed by the commit message.

```
git commit -m <commit_message>
```

On the CodeArts Repo console, a changed file can be saved only after you enter a commit message. Each saving operation on the console is a commit, and the mandatory message corresponds to the content after **-m** in the **commit** command.

CodeArts Repo automatically associates work items with code by capturing keywords from the commit message after **-m**. The most commonly used keyword

is **fix**, which is the recommended keyword in the prompt. The keyword must meet the following format:

```
git commit -m "fix #<work_item_id> <commit_message>"
```

If a work item is successfully associated, the system automatically changes the work item status based on the **configured code commit status transition**. By default, the **fix** keyword sets the work item to the resolved state.

Example:

```
git commit -m "fix #123456 fixed this bug"
```

The work item **123456** is set to the resolved state after being pushed to CodeArts Repo.

CodeArts Repo allows you to associate work items with code on the local PC or on the console. The following describes the two methods.

NOTE

- Only members of the same project and repository can associate work items with code.
- For the work item creator, specified modifier, or account (such as the project manager) that has the permission to modify all work items in the project, their association operations can change the work item status (new or resolved) and generate comment records. In the association records, **Transition successful** is displayed in the **Result** column. When you use an unauthorized account to perform operations, only association records are generated. The work item status is not changed, no comment record is generated, and **Association successful** is displayed in the **Result** column.

Associating a Work Item with Locally Committed Code

Prepare the Git environment on the local PC. For details, see [2.1 Installing and Configuring Git](#). If you can access the repository (**the corresponding remote repository has been associated**), perform the following operations:

Create a file on the local master branch and push the file to the remote repository. During the push, use the **fix** keyword in **-m** to associate the work item **task01** with code.

NOTE

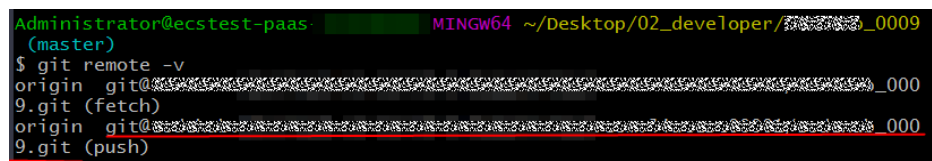
- In this example, the master branch is modified to simplify the process so that you can quickly understand how to associate a work item with code committed on the local PC.
- Do not modify the master branch in the actual situation. It is recommended that you create a branch for file operations, merge the changed file into the master branch, and push the master branch to the remote repository. (This is a default rule and good habit.)

Step 1 Right-click in the local repository folder to open the Git Bash client.

Step 2 Check whether the remote repository address is successfully associated.

```
git remote -v # View the remote repository address associated with the local repository.
```

In the following figure, the underlined part indicates the remote repository address associated with the local repository, and the information before the address is the alias of the remote repository on the local PC.



```
Administrator@ecstest-paas: ~$ git remote -v
origin git@github.com:ecstest-paas/02_developer/000_000_000 (fetch)
origin git@github.com:ecstest-paas/000_000_000 (push)
```

If the associated repository is not the one you want or the repository is not associated, **clone the desired repository to the local PC**.

After the clone is successful, run the **git remote -v** command again to verify the association.

Step 3 Check the repository status and switch to the master branch. (Skip this step for a repository cloned in the previous step.)

```
git status      # Check the repository status. You can view the current branch and whether there are
                # unsaved, uncommitted, and unpushed changes on the branch.
git checkout master # Switch to the master branch. Run the command when the current branch is not the
                # master branch.
```

Step 4 Create a file in the local repository folder and name the file **fileFor708206208**.

Step 5 Add the new file to the staging area using Git Bash.

```
git add fileFor708206208
```

Step 6 Commit the operation using Git Bash.

```
git commit -m "fix #708206208 Task01" #/ Use the fix keyword to associate task 01 whose ID is
708206208.
```

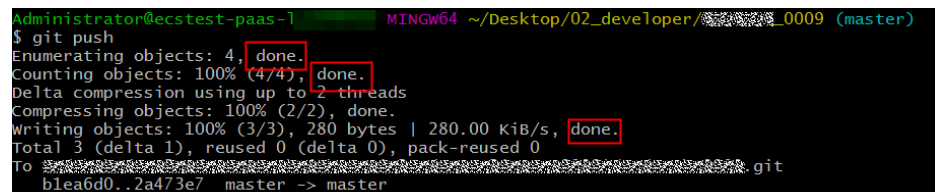
NOTE

708206208 is the ID of **task01**.

Step 7 Push the committed content to the associated CodeArts Repo repository using Git Bash.

```
git push
```

The command output varies depending on the repository structure. If **100%** or **done** is displayed for all steps, the push is successful. Push failures are usually caused by invalid **keys**.



```
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/02_developer/..._0009 (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 280 bytes | 280.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
To ... .git
b1ea6d0..2a473e7 master -> master
```

Step 8 Verify the association result.

Go to the work item list and locate the work item whose ID is 708206208 to view its details.

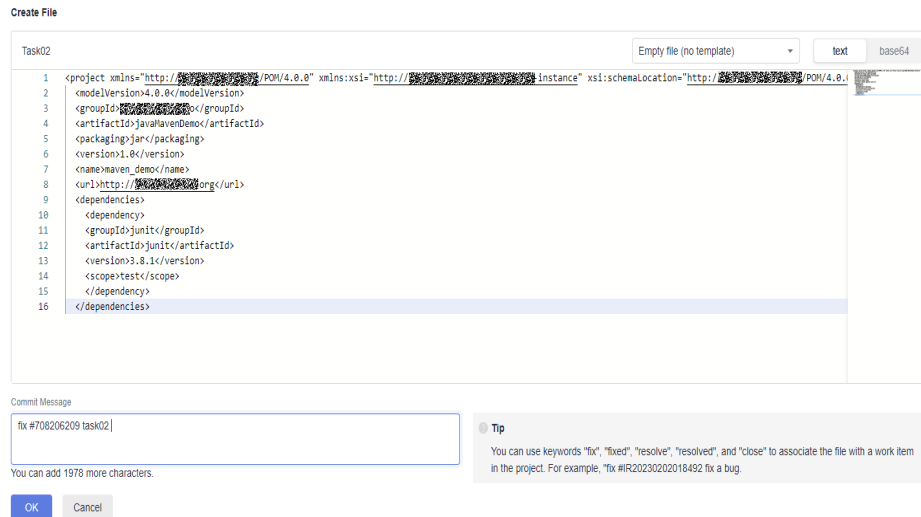
- The status is **Resolved**.
- An associated code commit record is added. You can click the commit ID to view the details.
- A comment is automatically generated to describe the work item association.

----End

Associating Work Items with Code Committed on the Console

Step 1 Go to the repository details page.

Step 2 **Create a file**, enter a commit message starting with **fix #708206209**, and set other parameters as required. The following figure shows an example.



NOTE

708206209 is the ID of **task02**.

Step 3 Click **OK**. The system performs the following operations on the CodeArts Repo repository:

```
Writes content to the new file.  
git add .  
git commit -m "fix #708206209 Task02"
```

That is, the system commits the new file and associates it with the **task02** work item using the **fix** keyword in the **-m** parameter.

Step 4 Verify the association.

View the **task02** work item.

- The status is **Resolved**.
- An associated code commit record is added. You can click the commit ID to view the details.
- A comment is automatically generated to describe the work item association.

Task Created by test at Apr 24, 2023 15:27:38 GMT+08:00 | Scrum009

#708206209 task01

Description **Associated (1)** Person-Hour Details Operation History

Associate with Work Item(0)

Code Commit Records(1)

Want to know how to get started? [Click here](#)

Branch	Commit Message	Committed By	Committed At
master	c3cfdbf9 - fix #708206209 task01	test	Apr 24, 2023 15:28:55 ...

Associated Code Branches(0)

Tag

Attachment

Comment

test Apr 24, 2023 15:28:55 GMT+08:00

Message From CodeArts Repo:
test use command 'fix' to commit code then the work item status has automatically changed to 'Resolved'

Status: Resolved

Assigned To: test

Module: --Select--

Sprint: --Select--

Start Date: Select a date.

Due Date: Select a date.

Order: 1

Priority: Middle

Severity: Minor

Notify: --Select--

ParentId: --Select--

Domain: --Select--

Show More

----End

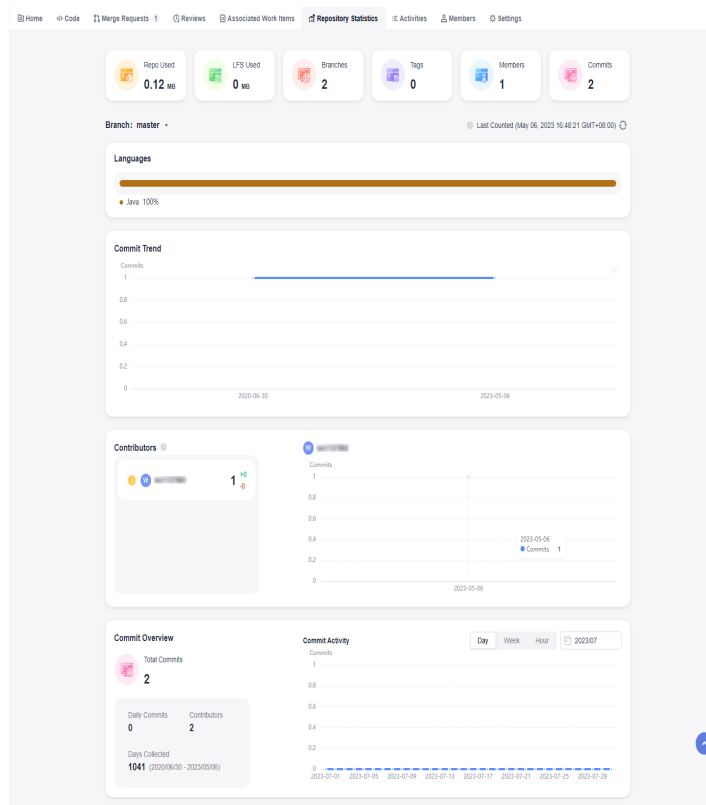
8.8 Viewing Repository Statistics

On the **Repository Statistics** tab page in the repository details, you can view the following repository statistics:

- **Repository summary:** Displays the Git repository capacity, **LFS** capacity, and the number of branches, tags, repository members, and commits. You can select a branch, and the statistical scope of commit trend, contributors, and commit overview will be changed, but the repository summary will not be affected.
- **Languages:** displays the distribution of each language in the current branch of the repository.
- **Commit trend:** displays the commit distribution of a branch in the repository.
- **Contributors:** collects statistics on the contribution of code committers in a branch (number of commits and number of code lines).
- **Commit overview:** collects statistics on code commits by different dimensions (weekly, daily, and hourly).

NOTE

- Developers and roles with higher permissions can trigger the collection of contributor and language statistics.
- Due to resource restrictions, statistics can be collected for each repository ten times a day.
- Each user can collect statistics for 1000 times a day.
- After the statistics are complete, the number of added and deleted code lines of each user is displayed before the deadline.
- Commits (an operation that combines two or more historical development records) of the merge node are not counted.



8.9 Viewing Activities

Access a repository and click the **Activities** tab page to view all activities of the current repository.

- **All:** This tab displays all operation records of the repository.
- **Push:** displays all push operation records of the repository, such as code push and branch creation and deletion.
- **Merge Request:** displays the operation records of all merge requests in the repository. You can click the sequence number of a merge request to view details, such as creating, closing, re-opening, and merging a merge request.
- **Review:** This tab displays all review comments of the repository. You can click the commit nID to view details such as adding or deleting comments.
- **Member:** displays the management records of all members in the repository, for example, adding or removing members and editing member permissions.

NOTE

- The displayed information includes the operator, operation content, and operation time.
- You can specify search criteria, such as the time range and operator, to filter and query data.

8.10 Managing Repository Members

8.10.1 IAM Users, Project Members, and Repository Members

Repository members come from project members of the project to which the repository belongs. Project members mainly come from IAM users of tenants. In addition to the tenant to which the project creator belongs, IAM accounts of other tenants can be invited to join the project. The following figure shows the relationships between IAM users, project members, and repository members.

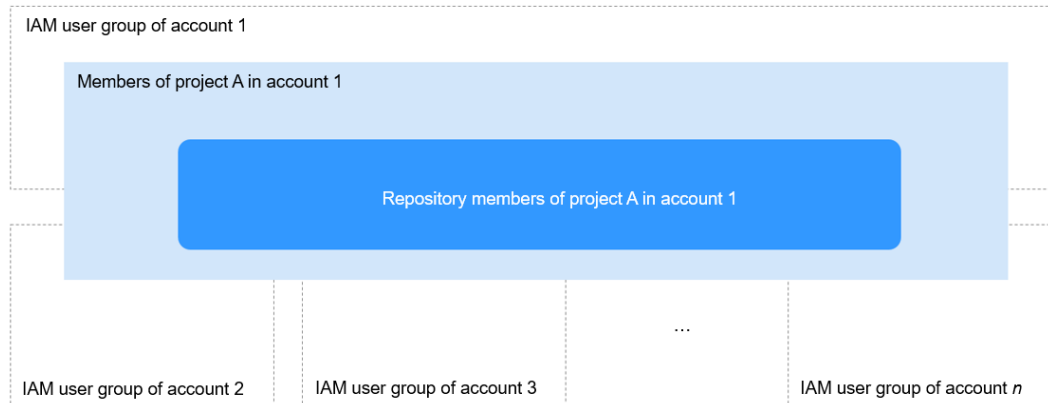


Table 8-8 Mapping between project roles and repository roles

Project Role	Repository Role
Project manager	Administrator
Developer	Developer
Test manager	Viewer
Tester	
Participant	
Viewer	
O&M manager	
Custom role	The repository role can be set as a committer, developer, or viewer by a project creator.

8.10.2 Configuring Member Management

You can manage repository members on the **Members** tab page. Only the repository creator (owner) and administrator can manage repository members. Other members can only view the repository member list. The following procedure shows how to configure member management.

NOTE

Currently, CodeArts Repo only allows you to import project members as repository members. For details about how to add project members or modify project member roles, see [Member Management](#).

Automatically Synchronizing Project Members to the Repository

Configure **Member Role Synchronization** to synchronize project roles to the repository. For details about the synchronization policies, see [Table 8-9](#).

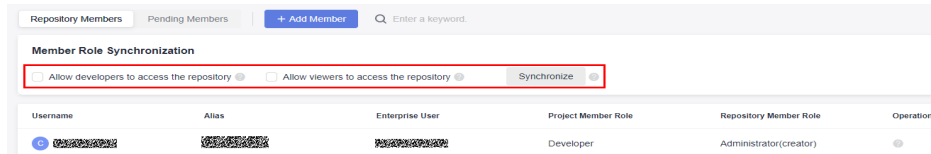
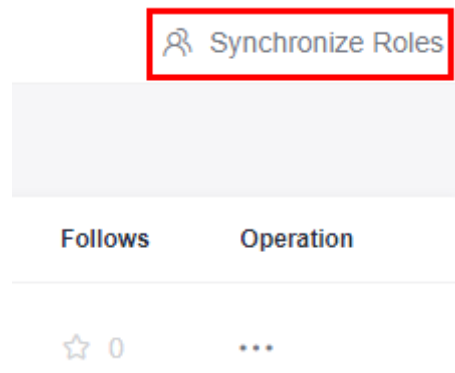


Table 8-9 Member role synchronization

Item	Project Role	Repository Role	Allowed Operation
---	Project manager	Administrator	---
Allow developers to access the repository	User-defined project role (Committer permission)	Committer	<ul style="list-style-type: none"> Set the role as a committer. Set the role as a developer Set the role as a viewer. Remove the member.
	Developer	Developer	<ul style="list-style-type: none"> Set the role as an administrator. Set the role as a committer. Set the role as a viewer. Remove the member.
	Custom role (developer permission)		<ul style="list-style-type: none"> Set the role as a developer Set the role as a viewer. Remove the member.
Allow viewers to access the repository	Test manager	Viewer	Remove the member.
	Tester		
	Participant		
	Viewer		
	Custom role (viewer permission)		

 NOTE

- By default, a project manager is the repository administrator. If you want to move the project manager out of the repository, you need to adjust the role of the project manager in the project settings.
- If you select a policy in **Member Role Synchronization**, related users added to the project are automatically synchronized to the repository.
- If you deselect policies in **Member Role Synchronization** and click **Synchronize**, related members will be removed immediately.
- On the repository list page, you can select **Synchronize Roles** to modify the repository role mapped from a custom project role as a project creator.



Manually Adding Project Members to the Repository

NOTICE

Manually configured repository members will be overwritten by **Automatically Synchronizing Project Members to the Repository**. You are advised to use either of the two functions.

Click **Add Member**. On the displayed dialog box, select a member from the member list of the corresponding project and add the member to the repository. A **default repository role** is assigned to the member based on the project role. For details about the role mapping, see the following table.

Table 8-10 Mapping between project roles and repository roles

Project Role	Repository Role	Allowed Operation
Project manager	Administrator (default)	<ul style="list-style-type: none">• Set the role as a committer.• Set the role as a developer.
	Developer	<ul style="list-style-type: none">• Set the role as an administrator.• Set the role as a committer.• Remove the member.
Developer	Administrator	<ul style="list-style-type: none">• Set the role as a committer.• Set the role as a developer.

Project Role	Repository Role	Allowed Operation
	Developer (default)	<ul style="list-style-type: none"> Set the role as an administrator. Set the role as a committer. Set the role as a viewer. Remove the member.
	Viewer	<ul style="list-style-type: none"> Set the role as a committer. Set the role as a developer. Remove the member.
Test manager	Viewer (default)	Remove the member.
Tester		
Participant		
Viewer		
O&M manager		
Custom role	Committer	<ul style="list-style-type: none"> Set the role as a committer. Set the role as a developer. Set the role as a viewer. Remove the member.
	Developer	<ul style="list-style-type: none"> Set the role as a developer. Set the role as a viewer. Remove the member.
	Viewer (default)	Remove the member.

 **NOTE**

If the project-level member list is empty, the project does not have members other than the repository creator. Add project members.

8.10.3 Repository Member Permissions

Repository Creation Permission

Table 8-11 Repository creation permission of project roles

Operation	Project Manager	Developer	Others
Create repositories	√	√	-

Repository Operation and Viewing Permission

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
Code	Access code online	√	√	√	√	√	-
	Edit code online	√	√	√	√	×	If a protected branch is set, permissions of this protected branch are used instead.
	Download code online	√	√	√	√	√	-
	Local code clone	√	√	√	√	√	-
	Local code push	√	√	√	√	×	If a protected branch is set, permissions of this protected branch are used instead.
Fork	Fork a project	√	√	√	√	√	When you select a project for the Fork repository, only the projects for which you have the project-level developer permission or higher are displayed.
Members	Add a member	√	√	×	×	×	-
	Edit a member	√	√	×	×	×	-
	Remove a member	√	√	×	×	×	-
	Approve a member	√	√	×	×	×	-

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	View a member	√	√	√	√	√	-
MR	Create an MR	√	√	√	√	×	-
	View an MR	√	√	√	√	√	-
	Merge an MR	√	√	√	×	×	<ol style="list-style-type: none"> If a protected branch is set, permissions of this protected branch are used instead. Developers cannot merge MRs by default. MRs can be merged by developers only when the target branch is set as a protected branch and developers have MR permissions.
	Edit an MR (Open)	√	√	√	×	×	<ol style="list-style-type: none"> The MR creator can perform this operation, but the MR creator must be a developer or role with higher permissions. The √ role can operate all MRs, including MRs created by others and MRs created by yourself.
	Close an MR	√	√	√	×	×	
	Re-open an MR	√	√	√	×	×	
	Edit a merged MR (Merged)	×	×	×	×	×	-

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Cherry-pick an MR (generate an MR)	√	√	√	√	×	A temporary branch containing cherry-pick is automatically generated. The cherry pick operation fails in the following scenarios: <ol style="list-style-type: none">1. If all branches are protected branches and the operator does not have the permission to create a branch (push), the operation fails.2. If the branch policy is configured and the temporary branch does not meet the policy, the operation fails.
	Revert an MR (generate an MR)	√	√	√	√	×	A temporary branch containing revert is automatically generated. The revert operation fails in the following scenarios: <ol style="list-style-type: none">1. If all branches are protected branches and the operator does not have the permission to create a branch (push), the operation fails.2. If the branch policy is configured and the temporary branch does not meet the policy, the operation fails.

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Cherry-pick an MR (No MR is generated, and new code is directly merged into the related branch.)	√	√	√	√	×	If a protected branch is set, permissions of this protected branch are used instead.
	Revert MR (No MR is generated, and new code is directly merged into the related branch.)	√	√	√	√	×	

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Delete the source branch	√	√	√	√	×	<ol style="list-style-type: none"> 1. The source branch can be deleted only when MR is performed between repository branches and the source branch is not protected. 2. If the Fork repository has committed an MR to the source repository, the source branch of the source repository cannot be deleted. 3. A protected source branch cannot be deleted.
	Vote scoring in the scoring mechanism	√	√	√	√	√	<ol style="list-style-type: none"> 1. All repository members can score the MR even if they are not configured as scorers of this MR. 2. By default, developers and roles with lower permissions can score from -1 to 1, and committers and roles with higher permissions can score from -2 to 2.
	Review in the approval mechanism	√	√	√	√	√	Only MR reviewers can review the MR.
	Approve in the approval mechanism	√	√	√	×	×	Only MR approvers and √ roles can review MRs.

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Delete an MR	×	×	×	×	×	No one can delete an MR.
Score	Score	√	√	√	√	×	The repository configuration prevails: <ol style="list-style-type: none"> 1. If Developers and above is selected, developers or users with higher permissions can give a score. 2. If Committers and above is selected, committer or or users with higher permissions can give a score.
Reviews	Add a review	√	√	√	√	√	You can add a review for which you have permission to view MR.
	Edit a review	×	×	×	×	×	Only reviewers can edit their reviews.
	Delete a review	×	×	×	×	×	
	Reply a review	√	√	√	√	√	You can reply a review for which you have permission to view.
	View a review	√	√	√	√	√	You can view all reviews for which you have permission to view MR.

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Resolve a review	√	√	√	×	×	<ol style="list-style-type: none"> When the severity of review is suggestion: MR creator, reviewer, committers and roles with higher permission can operate. When the severity of review is minor, major or fatal: Reviewer, committers, and roles with higher permission can operate, but the MR creator (Even if with supported roles) cannot operate.
Pipeline	Trigger an MR pipeline	√	√	√	√	×	The pipeline execution plan is enabled.
Branches	Create a branch	√	√	√	√	×	<ol style="list-style-type: none"> If Developers cannot create branches is selected, this operation cannot be performed. If Committers cannot create branches is selected, this operation cannot be performed.
	Edit a branch	√	√	√	√	×	
	Delete a branch	√	√	√	√	×	A protected branch cannot be deleted by any user.
	View a branch	√	√	√	√	√	-
Tag	Create a tag	√	√	√	√	×	If Developers cannot create tags is selected, this operation cannot be performed.

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
	Delete a tag	√	√	×	×	×	A protected tag cannot be deleted by any user.
	View a tag	√	√	√	√	√	-
Settings	View settings	√	√	×	×	×	-
	Edit settings	√	√	×	×	×	-
	Rename a repository	√	×	×	×	×	-
	Transfer repository ownership	√	×	×	×	×	-
Repository	Create a repository	√	√	√	√	×	-
	Delete a repository	√	√	×	×	×	-
	Display a repository	√	√	√	√	√	The repository is displayed for all repository members.
Activities	View updates	√	√	√	√	√	-

Type	Operation	Creator	Administrator	Committer	Developer	Viewer (Repository Member)	Remarks
Associated work items	View associated work items	√	√	√	√	√	-
Home	View home	√	√	√	√	√	-
Repository statistics	View the statistics	√	√	√	√	√	-
	Update the statistics	√	√	√	√	×	-
SSH and HTTP settings	View and edit	√	√	√	√	√	-
IP address white list	View and edit	×	×	×	×	×	The administrator can view and edit the information.

 **NOTE**

For details about how to set a protected branch policy, see [9.3.1 Protected Branches](#).

9 Configuring CodeArts Repo

- [9.1 General Settings](#)
- [9.2 Repository Management](#)
- [9.3 Policy Settings](#)
- [9.4 Service Integration](#)
- [9.5 Security Management](#)

9.1 General Settings

9.1.1 Repository Information

To view and modify the repository information, choose **Settings > General Settings > Repository Information** on the repository details page.

The settings take effect only for the repository configured.

Only the repository administrator and owner can view the page and have the setting permission.

Repository Description: remarks field when the template is open-source (public example template). It is used to facilitate search.

Visibility

- **Private:** Only repository members can access and commit code.
- **Public:** Read-only for visitors and hidden from repo lists and search results.
- **Public template:** The repository will be shared as a template in the whole site. **Template Title** and **Author** are mandatory

Repository Information

Repository Name

Repository Description

Visibility

Private Public Public template

Only Repository Members can access and commit code.

9.1.2 Notifications

CodeArts Repo Notifications

To set notifications, choose **Settings > General Settings > Notifications** on the repository details page.

The settings take effect only for the repository configured.

Only the repository administrator and owner can view the page and have the setting permission.

Email Notification

NOTE

If all notification types in the notification settings are disabled, the system sends an email notification to the creator or administrator by default when you perform the following operations:

- When a repository is created, an email notification is sent to the creator or administrator by default.
- When a non-repository member applies to join a repository, an email notification is sent to the creator by default.
- When a repository is frozen or closed, an email notification is sent to the creator or administrator by default.
- **Freeze a repository:** Send email notifications to the repository owner and administrators by default. This cannot be manually configured.
When a service is disabled or a repository is in arrears, the repository is frozen. No operation can be performed on the frozen repository.
Within 30 days after a repository is frozen, you can renew the repository or enable services to unfreeze the repository.
- **Close a repository:** Send email notifications to the repository owner and administrators by default. This cannot be manually configured.

Closing a repository is equivalent to permanently deleting the repository. When the repository is frozen for more than 30 days, the repository will be closed.

- **Delete a repository:** Send email notifications to the repository owner, administrator, committer, developer, and viewer. This can be manually configured.
- **Capacity warning:** This parameter is not enabled by default. You can manually set the capacity warning threshold as required. When the capacity of a single repository exceeds the threshold, the system emails the repository owner, administrators, committers, and developers. The warning email is sent only once unless you update the warning settings.
- **Open:** Push states of the merge request (including create and re-open) to specified roles by email. By default, the email notification is disabled. You can enable it to send email notifications to scorers, approvers, reviewers, and mergers.
- **Update:** Push code updates of the branch associated with the merge request to specified roles by email. By default, the email notification is disabled. You can enable it to send email notifications to scorers, approvers, or reviewers.
- **Merge:** Send email notifications to the MR creator by default. You can determine whether to also send an email notification to the merger.
- **Review:** Send email notifications to the MR creator by default. You can also disable the notification.
- **Approve:** Send email notifications to the MR creator by default. You can manually set not to send the notification.
- **Comment:** Send email notifications to the MR creator by default. You can also disable the notification.
- **Resolve Comment:** Send email notifications to the MR creator by default. You can manually set not to send the notification.

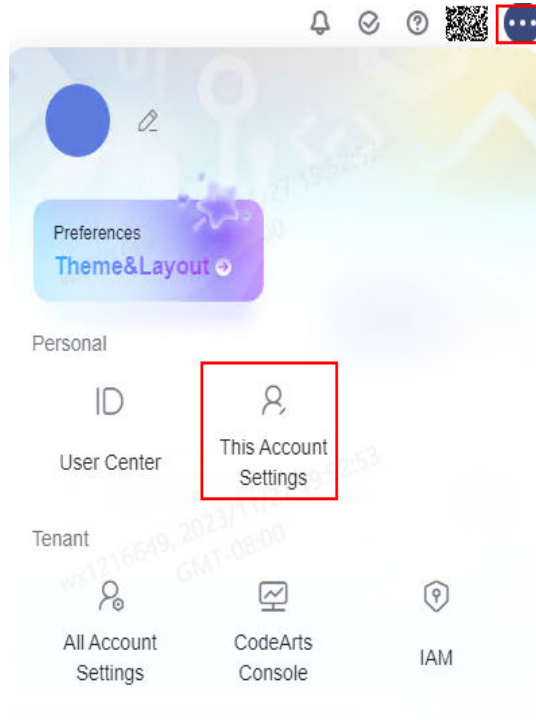
NOTE

If no email notification is received, go to [Notifications](#) to check whether the email and email notifications are enabled.

If you want to know repository changes in other ways than emails, you can choose [Service Integration](#) > [Webhooks](#) and customize notifications in your own system (third-party system).

CodeArts Notifications

CodeArts provides configurable notifications. On the CodeArts homepage, click your username in the upper right corner. In the dialog box that is displayed, click **This Account Settings** to configure notifications.



Choose **General Settings** > **Notifications**. Enable or disable and email notifications, and change the email address for receiving notifications.

You can also set a Do-Not-Disturb (DND) period so that you will not receive email notifications within the specified period.

Notifications

Do-Not-Disturb

After do-not-disturb (DND) is enabled, you will not receive email notifications within the specified period.

Email Notifications

Email Address for Receiving Notifications: [Edit Settings](#)

- Enable
 Disable

9.2 Repository Management

9.2.1 Repositories

To configure repository settings, you can choose **Settings** > **Repository Management** > **Repository Settings** on the repository details page.

The default branch is the branch selected by default when you enter the current repository and is also the default target branch when you create a merge request (MR). When a repository is created, the master branch is used as the default branch and can be manually adjusted at any time.

The settings take effect only for the repository configured.

Only the repository administrator and repository owner can view this page and have the setting permission. After the setting is complete, you can click **Commits** for the setting to take effect.

Table 9-1 Parameter description

Parameter	Description
Do not fork a repo.	This parameter is not selected by default. If this parameter is selected, all users cannot fork the repository.
Developers cannot create branches.	This parameter is not selected by default. If this parameter is selected, the developer role cannot create branches. NOTE A whitelist can be set to prevent developers who are not in the whitelist from creating branches.
Developers cannot create tags.	This parameter is not selected by default. If this parameter is selected, the developer role cannot create tags.
Committers cannot create branches	This parameter is not selected by default. If this parameter is selected, the committer role cannot create branches.
Pre-merge	By default, this option is not selected. After this option is selected, the server automatically generates MR pre-merging code. Compared with running commands on the client, this operation is more efficient and simple, and the build result is more accurate. This option applies to scenarios that have strict requirements on real-time build.
Branch name rule	<ul style="list-style-type: none">• The value cannot exceed 200 bytes.• The name cannot start with -, refs/heads/, or refs/remotes/, and cannot contain spaces or special characters such as brackets ([]), backward slashes (\), angle brackets (< >), tildes (~), circumflexes (^), colons (:), question marks (?), asterisks (*), exclamation marks (!), parentheses (()), single quotation marks ('), quotation marks ("), and vertical bars (). It cannot end with ./ or .lock.• The name of a new branch cannot be the same as that of an existing branch or tag.

Parameter	Description
Tag name rule	<ul style="list-style-type: none">• The value cannot exceed 200 bytes.• The name cannot start with -, refs/heads/, or refs/remotes/, and cannot contain spaces or special characters such as brackets ([]), backward slashes (\), angle brackets (<), tildes (~), circumflexes (^), colons (:), question marks (?), asterisks (*), exclamation marks (!), parentheses () , single quotation marks ('), quotation marks ("), and vertical bars (). It cannot end with ./ or .lock.• The name of a new tag cannot be the same as that of an existing branch or tag.

NOTE

- Byte: a group of adjacent binary digits. It is an important data unit of computers and is usually represented by B. 1 B = 8 bits.
- Character: a letter, digit, or another symbol that represents data and information.

Configuring MR Pre-combination

After an MR is created, you can customize the scripts for downloading plug-ins such as WebHook and CodeArts Pipeline. That is, you can control the downloaded code content.

- If you select **MR Pre-merge**, the server will generate a hidden branch, indicating that the MR code has been merged. You can directly download the code that already exists in the hidden branch.
- If **MR Pre-merge** is not selected, you need to perform pre-merge on the client. That is, download the code of the MR source branch and MR target branch and perform pre-merge on the build executor.

Command

The pre-merge command on the server is as follows:

```
git init
git remote add origin ${repo_url clone/download address}
git fetch origin +refs/merge-requests/${repo_MR_iid}/merge:refs/${repo_MR_iid}merge
```

If this option is not selected, you can perform the pre-merge operation on the client and create a clean working directory on the local host. The command is as follows:

```
git init
git remote add origin ${repo_url clone/download address}
git fetch origin +refs/heads/${repoTargetBranch}:refs/remotes/origin/${repoTargetBranch}
git checkout ${repoTargetBranch}
git fetch origin +refs/merge-requests/${repo_MR_iid}/head:refs/remotes/origin/${repo_MR_iid}/head
git merge refs/remotes/origin/${repo_MR_iid}/head --no-edit
```

Advantages

In scenarios that have high requirements on real-time build, for example, one MR may start the build of dozens or hundreds of servers, and the pre-merging result

generated by the local or client may be inconsistent with that generated by the server. As a result, the build code cannot be obtained accurately and the build result is inaccurate. Pre-merging on the server can solve this problem in real time. In addition, the script building command is simpler, and developers or CIEs can better use it.

9.2.2 Space Freeing

To enable space freeing, you can choose **Settings > Repository Management > Space Freeing** on the repository details page.

With space freeing, you can free up storage space to increase the read and write speed for the current repository by running background clean-up tasks, including compressing files and removing unused objects. Space freeing is similar to the garbage collect (gc) function in Git.

Only the repository administrator and owner can view the page and have the setting permission.

NOTE

It is recommended that you perform this operation once every month.

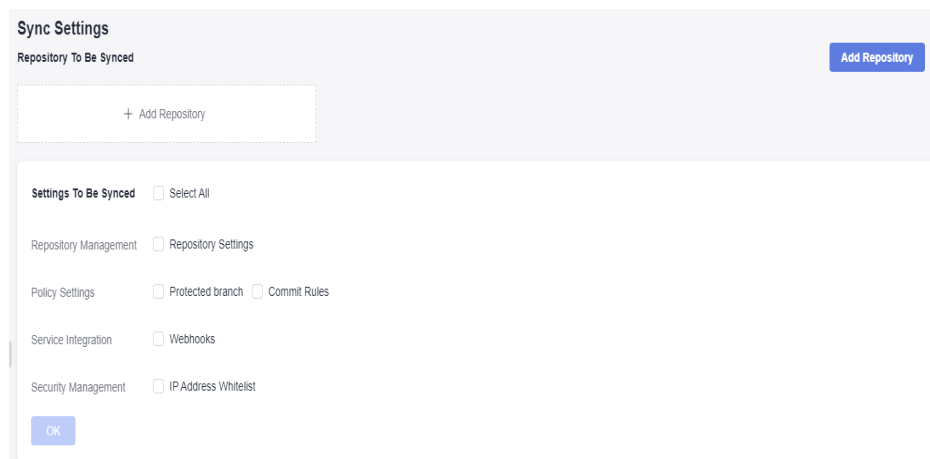
9.2.3 Synchronization

To configure repository settings, you can choose **Settings > Repository Management > Sync Settings** on the repository details page.

This function is used to synchronize the customized settings of the current repository to other repositories. This function supports cross-project synchronization but does not support cross-region synchronization.

This function is used for a repository forked based on the repository because the settings are not automatically copied during forking. For details, see [Forking a Repository](#)

Developers or roles with higher permissions can view this page. However, only the repository administrator and owner have the operation permission.



Sync Settings

Repository To Be Synced Add Repository

+ Add Repository

Settings To Be Synced Select All

Repository Management Repository Settings

Policy Settings Protected branch Commit Rules

Service Integration Webhooks

Security Management IP Address Whitelist

OK

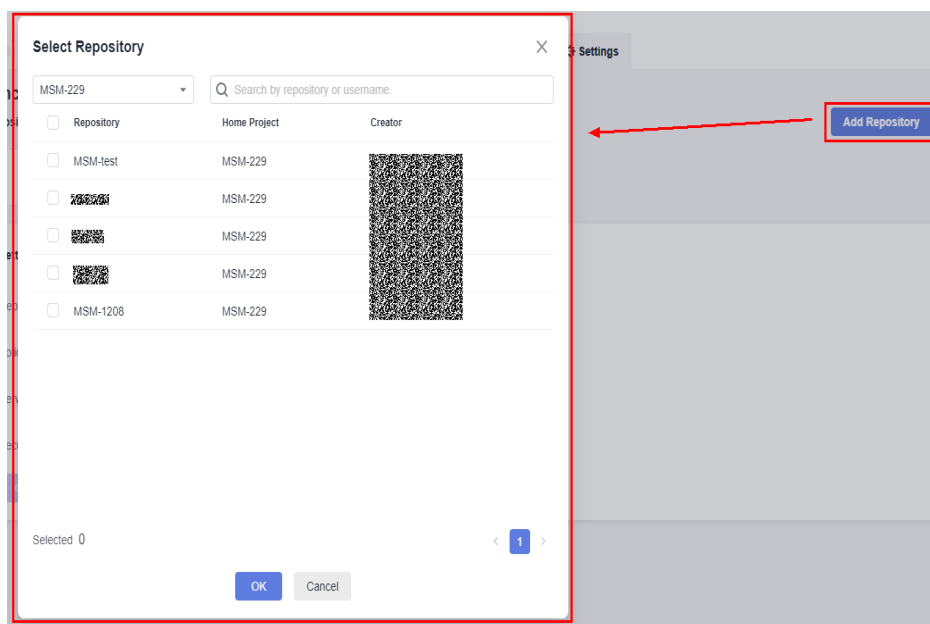
Adding a Synchronization Repository

NOTICE

Ensure that the network connection is normal before synchronizing a repository.

- For public platforms, CodeArts Repo supports access to code repositories.
- For private repository platforms on the intranet, ensure that the network connection between CodeArts Repo and your repository is normal.

Step 1 Click **Add Repository**. In the dialog box that is displayed, select the target repository.



Step 2 Click **OK**. The repository synchronization is complete.

----End

NOTE

Common Failure Causes

- Failed to synchronize **Commit Rules**: No commit rules are set for the source repository.
- Failed to synchronize protected branches: The branch names of the source repository and target repository are different.

9.2.4 Submodules

Background

A submodule is a Git tool used to manage shared repositories. It allows you to embed a shared repository as a subdirectory in a repository. You can isolate and reuse repositories, and pull latest changes from or push commits to shared repositories.

You may want to use project B (a third party repository, or a repository developed by yourself for multiple parent projects) in project A, and use them as two separate projects. Submodules allow you to clone a Git repository as a subdirectory into another Git repository while keeping commits separate.

The submodules are recorded in a file named **.gitmodules**, which records the information about the submodules.

```
[submodule "module_name"] # Submodule name  
path = file_path # File path of the submodule in the current repository (parent repository).  
url = repo_url # Remote repository IP address of the submodule (sub-repository).
```

In this case, the source code in the **file_path** directory is obtained from **repo_url**.

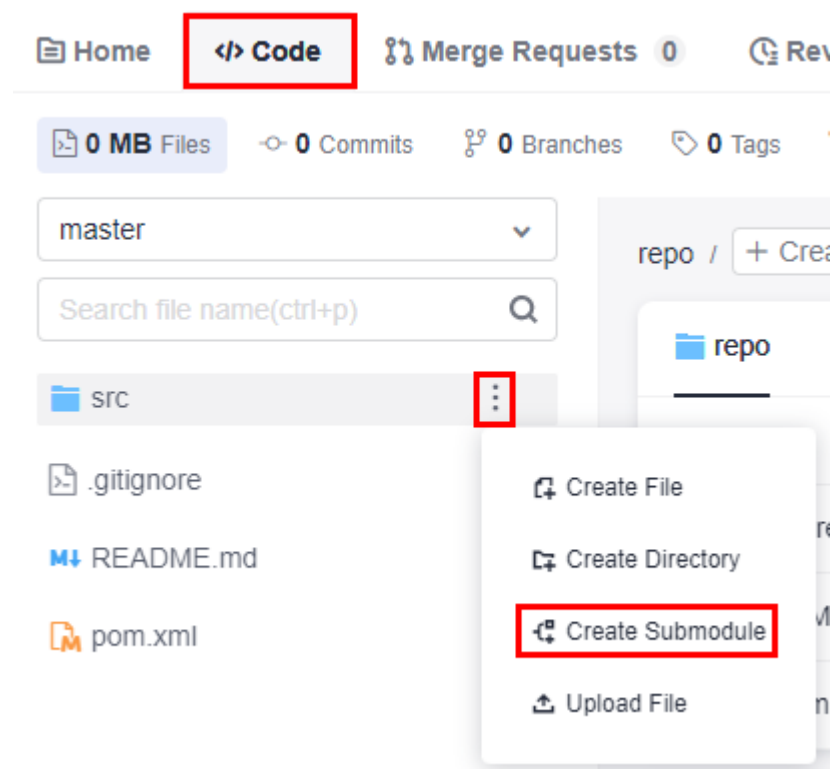
Using the Console

- **Creating a submodule**

- **Entry 1:**

You can add a submodule to a folder in the repository file list.

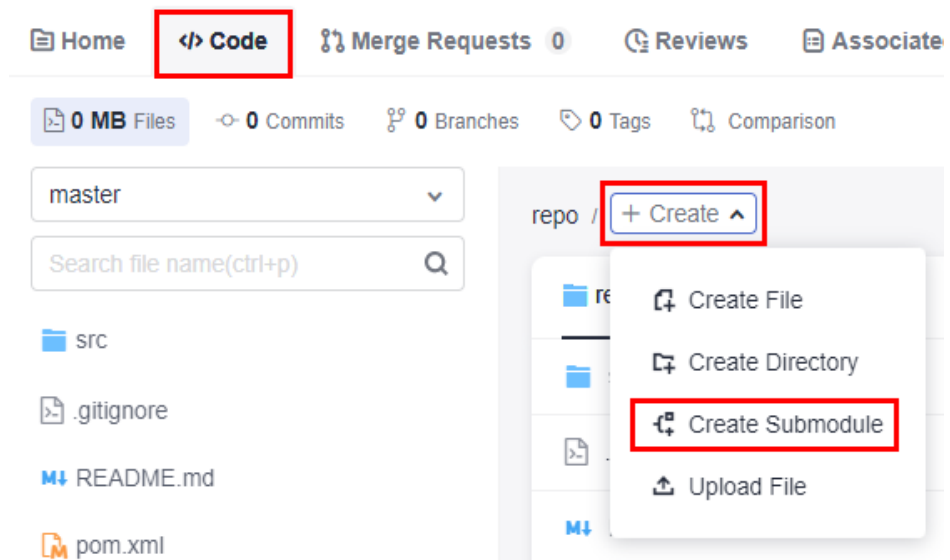
Click  and select **Create Submodule**, as shown in the following figure.



- **Entry 2**

You can create a submodule on the **Code** tab page

Click  and select **Create Submodule**, as shown in the following figure.




- **Entry 3:**
You can create a submodule in the repository settings.
Choose **Settings > Repository Management > Submodules > Create Submodule**.
- **Remarks:**
You can use one of the preceding methods to **create a submodule**.
Configure the following parameters and click **OK**.

Table 9-2 Parameters of creating a sub-repository

Parameter	Description
Submodule	Select a repository as the submodule.
Submodule Branch	Select the target branch of the submodule to be synchronized to the parent repository.
Submodule Path	The storage path of the sub-repository in the parent repository. Use slashes (/) to separate levels.
Details	Remarks for creating a submodule. You can find the operation in the file history. The value contains a maximum of 2000 characters.

NOTE

After the creation is complete, you can find the submodule (sub-repository) in the corresponding directory of the repository file list. The icon on the left of the corresponding file is .

- **Viewing, synchronizing, and deleting a submodule**
Choose **Settings > Repository Management > Submodules**. On the displayed page, repository administrators can view, synchronize, and delete submodules.
- **Synchronizing deploy keys**
If a submodule is added on the Git client, the repository administrator needs to synchronize the deploy key of the parent repository to the submodule on the **Settings > Repository Management > Submodules** page. In this way, the submodule can also be pulled during the build of the parent repository.

Using the Git Client

Step 1 Add a submodule.

```
git submodule add <repo> [<dir>] [-b <branch>] [<path>]
```

Example:

```
git submodule add git@***.***.com:****/WEB-INF.git
```

Step 2 Pulling a repository that contains a submodule

```
git clone <repo> [<dir>] --recursive
```

Example:

```
git clone git@***.***.com:****/WEB-INF.git --recursive
```

Step 3 Update a submodule based on the latest remote commit

```
git submodule update --remote
```

Step 4 Push updates to a submodule.

```
git push --recurse-submodules=check
```

Step 5 Delete a submodule.

1. Delete the entry of a submodule from the **.git submodule** file.
2. Delete the entry of a submodule from the **.git/config** file.
3. Run the following command to delete the folder of the submodule.

```
git rm --cached {submodule_path} # Replace {submodule_path} with your submodule path.
```

NOTE

Omit the slash (/) at the end of the path.

For example, if your submodule is stored in the **src/main/webapp/WEB-INF/** directory, run the following command:

```
git rm --cached src/main/webapp/WEB-INF
```

----End

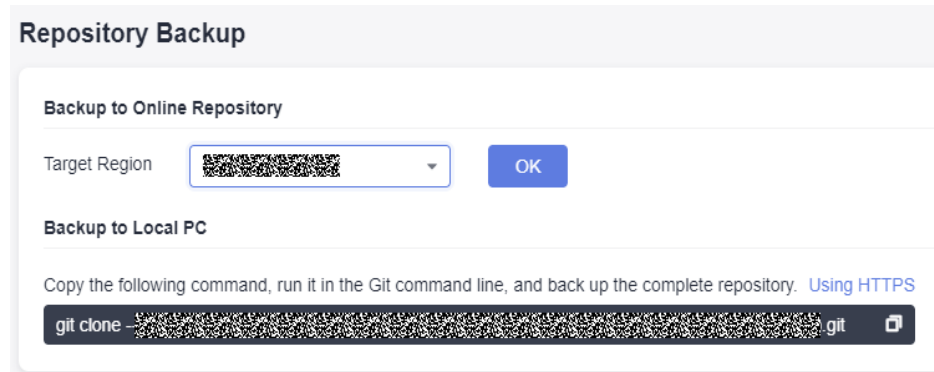
9.2.5 Repository Backup

To configure remote backup, choose **Settings > Repository Management > Repository Backup** on the repository details page.

The repository can be backed up in either of the following modes:

- **Backup to Online Repository:** Back up the repository to another region.
This mode imports a repository from a region to another region. For details, see [Importing an External Repository](#).

- **Backup to Local PC:** Back up the repository to your local PC.
You can use the HTTPS or SSH clone mode. The clone command is generated as shown in the following figure. You only need to paste the command to the local Git client and run it. (Ensure the repository connectivity.)
Only the repository administrators and owners can view this tab page and have permissions.



9.3 Policy Settings

9.3.1 Protected Branches

To configure protected branches, you can choose **Settings > Policy Settings > Protected Branches** on the repository details page.

The settings take effect only for the repository configured.

Only the repository administrator and owner can view the page and have the setting permission.

Functions of Protected Branches

- Ensure branch security and allow developers to use MRs to merge code.
- Prevent non-administrators from pushing codes.
- Prevent all forcibly push to this branch.
- Prevent anyone from deleting this branch.

NOTE



When you create a repository, the repository automatically sets the default branch (generally master) as the protection branch to ensure repository security.

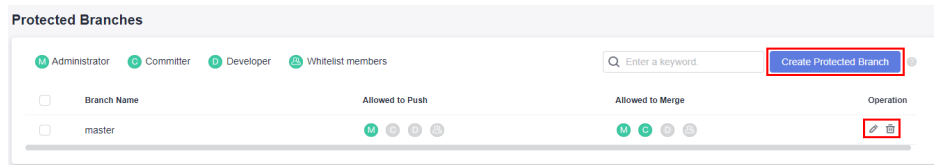
After you set a protected branch, the protected branch cannot be used as the target branch for code merging.

Editing Protected Branches

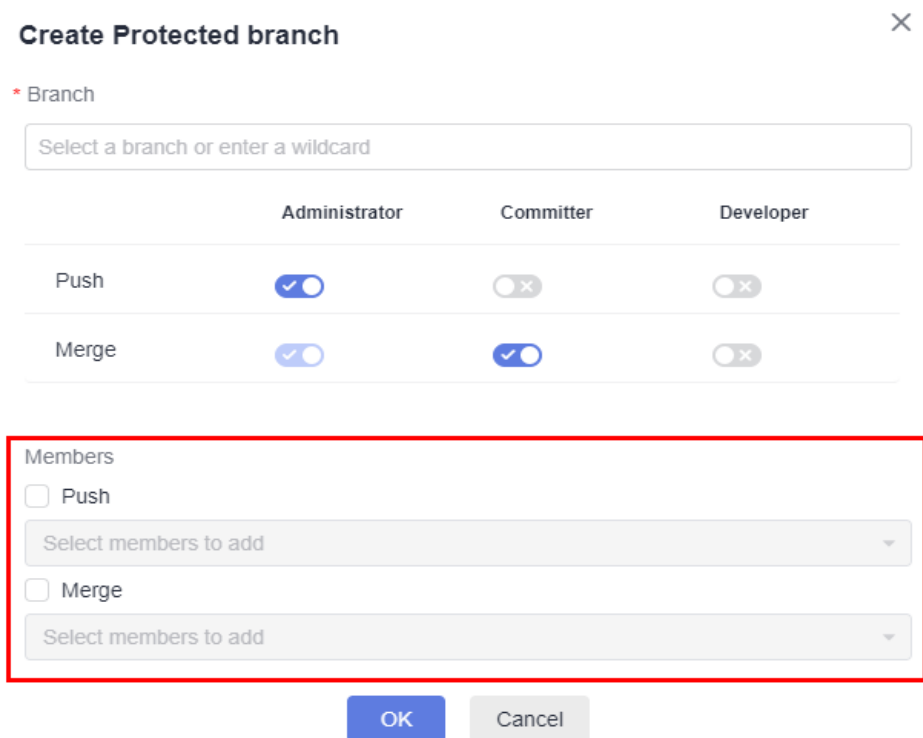
You can set a protected branch. The procedure is as follows:

- Click **Create Protected Branch**. In the **Added Protected Branch** dialog box, select a branch from the drop-down list or manually enter a branch name or wildcard character, select the corresponding permissions or assign permissions to users, and click **OK**.

- Click  to modify the configuration of the protected branch.
- Click  to delete the protected branch.



- Configure the whitelist: To assign permissions to one or more members of an unauthorized role, you can select **Push** and **Merge** under **Members** and click the drop-down list to add the members to the trustlist.



 **NOTE**

- Only developers and users with permissions higher than developers have the **Can push** and **Can merge** permissions.
- If **Administrator**, **Committer**, and **Developer** are selected for **Can push**, all these roles have the permissions. In this case, you do not need to select **Can push** or **Can merge** under **Members**.
- You can create, edit, and delete protected branches in batches.

9.3.2 Protected Tags

To configure protected tags, you can choose **Settings > Policy Settings > Protected Tags** on the repository details page.

Only the repository administrator and owner can view the page and have the setting permission.

You can set protected tags to prevent production tags or important tags from being deleted. The procedure is as follows:

Click **Create Protected Tag**. In the **Added Protected Tags** dialog box, select a tag from the **To be protect tag** drop-down list or create a wildcard, select **No one, Developers + Committer + Maintainers**, or **Maintainers** from the **Allowed to create** drop-down list, and click **OK**.

NOTE

- When a developer, committer and administrator, or administrator is allowed to create protected tags, or other members cannot create or delete the tags. If no one can create protected tags, all members cannot create or delete the tags.
- Click to delete protected tags.

9.3.3 Commit Rules

To configure commit rules, you can choose **Settings > Policy Settings > Commit Rules** on the repository details page.

On the **Commit Rules** page, you can establish a series of code commit verification and restriction rules to ensure code quality. The settings take effect only for the configured repository.

Only the repository administrator and owner can view the page and have the setting permission.

Table 9-3 Parameters on the Commit Rules page

Parameter	Description
Reject unsigned commits	<p>Only signed commits can be pushed to the repository.</p> <p>CodeArts Repo signature mode:</p> <p>When performing online commit in CodeArts Repo, use the following format to compile and submit information:</p> <pre>commit message # Enter the customized submission information. # This is a blank line. Signed-off-by: User-defined signature # Enter the user-defined signature after Signed-off-by:</pre> <p>Git client signature mode:</p> <p>When running the commit command on the Git client, you need to add the <code>-s</code> parameter.</p> <pre>git commit -s -m "<your commit message>"</pre> <p>You need to configure the signature and email address on the client in advance.</p>
Tags cannot be deleted	<p>After this option is selected, tags cannot be deleted on the page or by running commands on the client.</p>

Parameter	Description
Prevent committing secrets	Confidential files include ssh_server_rsa , id_rsa , and id_dsa . For details, see Description of Confidential Files .
Prevent git push -f	Indicates whether users can run the git push -f command on the client to push code. git push -f indicates that the current local code repository is pushed to and overwrites the code in CodeArts Repo. In general cases, you are not advised using this command.

Creating a Commit Rule

The repository administrator and repository owner can create a commit rule for a branch of the repository. Only one commit rule can be set for each branch.

NOTE

Priority matching mechanism of commit rules:

1. The target branch preferentially matches the configured commit rules.
2. If no rule is exactly matched, the first rule fuzzily match is used.
3. If no rule is fuzzily matched, the default rule is used.

Table 9-4 Parameters

Parameter	Description
Rule Name	This parameter is mandatory. The value contains a maximum of 200 characters.
Branch	This parameter is mandatory. Select a branch from the drop-down list or create a regular expression. This field supports a maximum of 500 characters.

Parameter	Description
Commit rules	<p>Parameters in this area are optional.</p> <ul style="list-style-type: none">Commit Message: This parameter is left empty by default, indicating that the commit message is not verified, and any parameter can be committed. This field supports a maximum of 500 characters. For example, you can set the format rule of the commit message as follows: <code>TraceNo:(REQ[0-9]{1,9})(.\n .n)Author:*(.\n .n)Description:*</code> The following is a commit message that complies with the rule: <code>TraceNo:REQ1234567 Author:**** Description:testpushfile</code> The following is a commit message that does not comply with the rule: <code>new files</code>Negative Match: This parameter is left empty by default, indicating that the commit information is not verified, and any parameter can be committed. This field supports a maximum of 500 characters. For example, you can set the format rule of the commit message as follows: <code>TraceNo:(REQ[0-9]{1,9})(.\n .n)Author:*(.\n .n)Description:*</code>Commit author: This parameter is left empty by default, indicating that the commit author is not verified, and any parameter can be committed. This field supports a maximum of 200 characters. The commit author can run the git config -l command to view the value of user.name and run the git config --global user.name command to set the value of user.name. Example: Rules for setting the commit author: <code>([a-z][A-Z]{3})([0-9]{1,9})</code>Commit author's email: This parameter is left empty by default, indicating that the commit author email is not verified, and any parameter can be committed. This field supports a maximum of 200 characters. The commit author can run the git config -l command to view the value of user.email and run the git config --global user.email command to set the email address. Example: Rules for setting the email of the commit author: <code>@huawei.com\$</code>

Parameter	Description
Basic Attributes	<p>Parameters in this area are optional.</p> <ul style="list-style-type: none">● File Name That Cannot Be Committed: This parameter is left empty by default, indicating that the file name is not verified, and any file can be committed. You are advised to use standard regular expressions to match the file name. By default, the file path is verified based on the file name rule. This field supports a maximum of 2000 characters. Example: Set File Name That Cannot Be Committed: <code>(\jar \.exe)\$</code>● Each File Size (MB): The default value is 50, indicating that the push is rejected if the size of the added or updated file exceeds 50 MB. The administrator can change the value from 0 to 200. <p>NOTE When a repository is created, the maximum size of a single file in the default submission rule (default) is 200 MB. When a repository is created, the recommended maximum size of a single file in the default submission rule is 50 MB.</p>
Binary Rules	<p>Parameters in this area are optional.</p> <p>These parameters are not set by default, indicating that binary files can be uploaded. The size of a single file cannot exceed the upper limit. Allow changes to binary files, Repo File Whitelist, and Privileged User take effect only when Do not allow new binary files is selected. If you select Allow changes to binary files, binary files in modifiable state are not intercepted and can be directly uploaded. Binary files can be deleted without binary check.</p> <ul style="list-style-type: none">● Do not allow new binary files (privileged users excepted)● Allow changes to binary files (privileged users excepted)● Binary file trustlist (files that can be directly imported to the database. This field supports a maximum of 2000 characters.)● Privileged User (Max. 50 privileged users.)
Effective Date	<p>Parameters in this area are optional.</p> <p>Before being pushed, all commitments created after the date specified by this parameter must match the hook settings. If this parameter is left empty, all commitments are checked regardless of the committing date.</p>

 **NOTE**

You are not advised storing binary files in CodeArts Repo. Otherwise, the performance and stability of the code repository will be affected.

Table 9-5 Examples of common regular expressions

Rule	Examples
Single a, b, or c	[abc]
Characters other than a, b, or c	[^abc]
Lowercase letters ranging from a to z	[a-z]
Characters other than the range of a to z	[^a-z]
Uppercase and lowercase letters in the range of a to z or A to Z	[a-zA-Z]
Any single character	.
Either a or b	a b
Any blank character	\s
Non-blank character	\S
Arabic numeral character	\d
Non-Arabic numeral characters	\D
Letters, digits, or underscores (_)	\w
Characters other than letters, digits, or underscores (_)	\W
Match the content in parentheses (not capture)	(?:...)
Match and capture the content in parentheses	(...)
No or one a	a?
No or more a's	a*
One or more a's	a+
Three a's	a{3}
More than three a's	a{3,}
3 to 6 a's	a{3,6}
Beginning of text	^
End of text	\$
Word boundary	\b
Non-word boundary	\B
Line breaker	\n
Carriage return character	\r
Tab key	\t

Rule	Examples
Null string	\0

9.3.4 Merge Requests

To configure MRs, you can choose **Settings > Policy Settings > Merge Requests** on the repository details page.

Merge Requests applies to merge MRs. MRs can be merged only when all configured MR conditions are met. You can select **Score** or **Approval** for **Merge Mechanism**.

The settings take effect only for the repository configured. Only the repository administrator and owner can view the page and have the setting permission.

Merge Mechanism

- **Score:** Code review is included. Based on scoring, the minimum merging score can be set and the score ranges from 0 to 5. The code can be merged only when the score and mandatory review meet pass conditions. When selecting the scoring mechanism, you need to set the minimum score.
- **Approval:** Code review and merge approval are included. Code can be merged only after the number of reviewers reaches gate requirements. You are advised to [configure branch policies](#) when you select the approval mechanism.

NOTE

By default, **Approval** is used. You can manually switch to **Score**.

After the merge mechanism is switched, the workflows of the MRs are changed. However, the early created MRs retain the previous merge mechanism.

Merge Conditions

Table 9-6 Parameters


Parameter	Description
Merge after all reviews are resolved.	After this parameter is selected, if Must resolve is selected as the review comment, a message Review comment gate: failed is displayed and the Merge button is unavailable. If it is a common review comment, the Resolved button does not exist, the MR is not intercepted by the merge condition.

Parameter	Description
Must be associated with CodeArts Req	<ul style="list-style-type: none">• Associate only one ticket number: If this parameter is selected, one MR can be associated with only one ticket number.• All E2E ticket numbers pass verification: If this parameter is selected, all associated E2E ticket numbers must pass the verification.• Branches to configure the MR policy: Multiple branches can be added. You can manually enter wildcard characters and press. Press Enter, for example, *-stable or production/*.

MR Settings

Table 9-7 Parameters

Parameter	Description
Do not merge your own requests	After this parameter is selected, the Merge button is unavailable when you view the MRs created by yourself. You need to ask the person who has the permission to merge the MRs.
A repo administrator can forcibly merge code	The project creator and administrator roles have the permission to forcibly merge MRs. If the merging conditions are not met, these roles can click Force Merge to merge MRs.
Continue with code review and comment after requests are merged	After this parameter is selected, you can continue to review and comment on the code that has been merged the MR.
Mark the automatically merged MRs as Closed (If all commits in the B MR are included in the A MR, the B MR is automatically merged after the A MR is merged. By default, the B MR is marked as merged . You can use this parameter to mark the B MR as closed .)	<ul style="list-style-type: none">• If this parameter is not selected, MRs that are automatically merged are marked as merged.• If this parameter is selected, MRs that are automatically merged are marked as closed.

Parameter	Description
Cannot re-open a Closed MR.	<p>If this option is selected, the branch merge request cannot be set back to Open after it is closed. Re-open in the upper right corner is hidden.</p>  <p>This parameter is used for process control to prevent review history from being tampered with.</p>
Delete source branch by default after the MR is merged	<p>After the merging, the source branch is deleted.</p> <ul style="list-style-type: none">• A protected source branch cannot be deleted.• This setting does not take effect for historical MRs. Therefore, you do not need to worry about branch loss.
Do not Squash	<p>After this parameter is selected, the Squash button is unavailable, and the entry for using this button is unavailable in the MR.</p>
Enable Squash merge for new MRs	<p>Squash merge means that when merging two branches, Git squashes all changes on the merged branch into one and appends them to the end of the current branch as merge commit, which simplifies the branch. The only difference between squash merge and common merge lies in the commitment history. For common merge, the merge commitment on the current branch usually has two commitment records, while squash merge has only one commitment record.</p>

Merge Method

Table 9-8 Parameters

Parameter	Description
Merge commit	<p>If this parameter is selected, a merge commit is created for every merge, and merging is allowed as long as there are no conflicts. That is, no matter whether the baseline node is the latest node, the baseline node can be merged if there is no conflict.</p> <ul style="list-style-type: none">• Do not generate Merge nodes during Squash merge: If this parameter is selected, no merge node is generated during the squash merging.• Use MR merger to generate Merge Commit: If this parameter is selected, the commit information is recorded.• Use MR creator to generate Merge Commit: If this parameter is selected, the commit information is recorded.
Merge commit with semi-linear history	<p>If this parameter is selected, a merge commit is recorded for each merge operation. However, different from Merge commit, the commitment must be performed based on the latest commit node of the target branch. Otherwise, the system prompts the developer to perform the rebase operation. In this merging mode, if the MR can be correctly constructed, the target branch can be correctly constructed after the merge is complete.</p>
Fast-forward	<p>If this parameter is selected, no merge commits are created and all merges are fast-forwarded, which means that merging is only allowed if the branch could be fast-forwarded. When fast-forward merge is not possible, the user is given the option to rebase.</p>

Configure Branch Policy

Click **Create** to set a merge policy for a specified branch or all branches in the repository.

 NOTE

Currently, branch policies can be set only for the Approval mechanism.

The following is an example of the branch policy priority:

- Assume that there are policies A and B in the repository and their branches are the same. The system uses the latest branch policy by default.
- Assume that there are policies A and B in the repository. Branch a and branch b are configured for policy A, and branch a is also configured for policy B. When a merge request whose target branch is branch a is committed, the system uses policy B by default.

If no branch policy is set in the approval mechanism, the default branch policy is used when a merge request is committed. The branch policy can be edited and viewed but cannot be deleted. The policy configuration is as follows:

- **Branches:** *. By default, all branches are used and cannot be modified.
- **Reviewers Required:** The default value is **0**.
- **Approvals Required:** The default value is **0**.
- **Reset approval gate:** This option is selected by default.
- **Reset review gate:** This option is selected by default.
- **Add approvers/reviewers only from the following ones:** This option is not selected by default.
- **Enable pipeline gate:** This option is not selected by default.
- **Mergers:** This parameter is left blank by default.
- **Approvers:** This parameter is left blank by default.
- **Reviewer:** This parameter is left blank by default.

Table 9-9 Parameters

Parameter	Description
Branches	Set policies for all branches or a branch.
Reviewers Required	Set Reviewers Required . When the number of reviewer who give pass meets the Reviewers Required , the gate is passed. 0 indicates that the review gate is optional. However, if an MR is rejected by a reviewer, it fails the gate.
Approvals Required	Set Approvals Required . When the number of approvals who give pass meets the Approvals Required , the gate is passed. 0 indicates that the approval gate is optional. However, if an MR is rejected by an approver, it fails the gate.
Reset approval gate	When code is re-pushed to the source branch of an MR.
Reset review gate	When code is re-pushed to the source branch of an MR.
Add approvers/reviewers only from the following ones	If this option is selected, you can configure the list of New Approvers and New Reviewers . If you want to add additional members, you can only add members from the lists.

Parameter	Description
Enable pipeline gate	If this option is selected, before the merge, you need to pass all pipeline gates. This rule integrates the CI into the code development process.
Mergers	The list of mandatory mergers can be configured. When a merger request is created, the list is automatically synchronized to the merger request.
Approvers	The list of mandatory reviewers can be configured. When a merge request is created, the list is automatically synchronized to the merge request.
Reviewer	The list of mandatory reviewers can be configured. When a merge request is created, the list is automatically synchronized to the merge request.

 NOTE

Example of a mandatory reviewer list:

- The **Reviewers Required** is 2. If the list of **mandatory reviewers** is empty, the 2 approvers in the list of **New Reviewers** give pass and the gate is passed.
- The **Reviewers Required** is 2. If the list of **mandatory reviewers** is not empty, the gate can be approved only after at least one reviewer in the list give pass.

9.4 Service Integration

9.4.1 E2E Settings

Repo uses this E2E tracing setting to log code merge reasons, such as implementing a requirement, fixing a bug, or completing a work item. Association is enabled by default.

Integrated Systems

It integrates with CodeArts Req and uses work items in CodeArts Req to associate with code commits.

 NOTE

The repositories of Kanban projects do not support E2E settings.

Integration Policies

(Optional) Specify available selection conditions when you associate with a work item.

Excluded States: States of work items that CANNOT be associated with.

Associable Types: Types of work items that can be associated with.

Applicable Branches: Branches to comply with preceding restrictions.

Automatic ID Rules Extraction

Automatic ID Rules Extraction (automatically extracting ticket numbers based on code commitment information) are as follows:

- **ID Prefix:** (Optional) A maximum of 10 prefixes are supported, for example, *[Trouble ticket number or Requirement ticket number]*.

NOTE

If **ID Prefix**, **Separator**, and **ID Suffix** are not empty, the automatic ticket number extraction function is enabled by default.

- **Separator:** (Optional) The default value is a semicolon (;).
- **ID Suffix:** (Optional) The default value is a newline character.

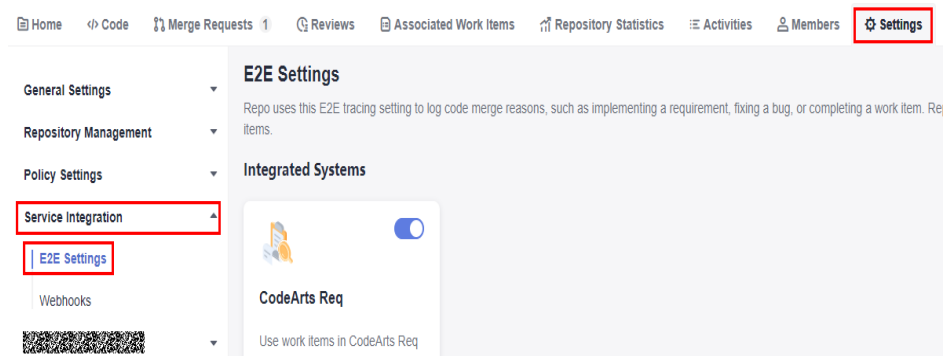
NOTE

- The values of **ID Prefix**, **Separator**, and **ID Suffix** cannot be the same.
- If **Separator** is left empty, the values of **ID Prefix** and **ID Suffix** cannot be two semicolons (;).
- If **ID Suffix** is left empty, the values of **ID Prefix** and **Separator** cannot be `\n`.
- The values of **ID Prefix**, **Separator**, and **ID Suffix** are matched in full character mode. Regular expressions are not supported.

Examples

Step 1 Configure E2E settings.

1. Go to the target repository.
2. Choose **Settings** > **Service Integration** > **E2E Settings**. The **E2E Settings** page is displayed.



3. Configure the following integration policies and click **Submit**.
Applicable Branches: Select the target branch, for example, **branch**.
ID Prefix: user-defined prefix, for example, **Incorporated requirements**.

Integration Policies

Excluded States States of work items that CANNOT be associated with

Associable Types Types of work items that can be associated with, e.g. Story/Task/Bug

Applicable Branches Branches to comply with preceding restrictions

Automatic ID Rules Extraction

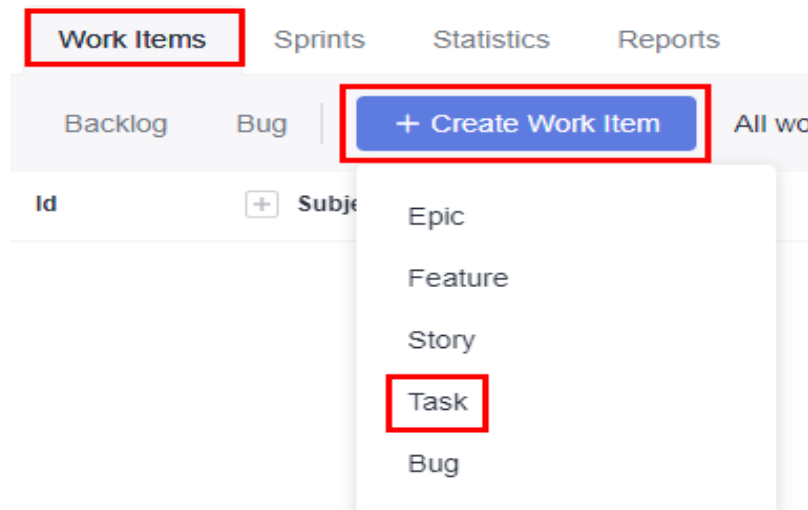
ID Prefix Enter a prefix of

Separator Specify the separator (; by default).

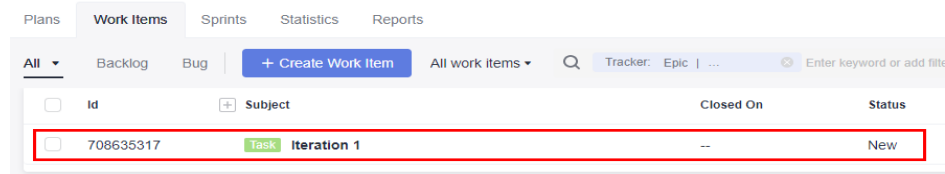
ID Suffix Specify the suffix (line break by default).

Step 2 Create a work item.

1. Click the target project name to access the project.
2. On the current **Work Items** tab, click **Create Work Item** and choose **Task** from the drop-down list box. The page for creating a work item is displayed.

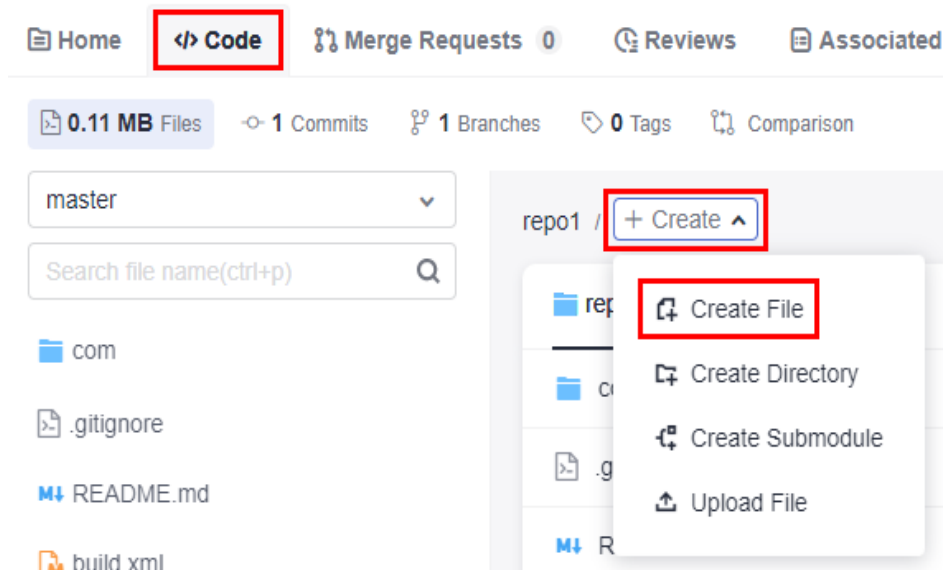


3. Enter a title, for example, Sprint 1.
Retain the default values for other parameters. Click **Save**.



Step 3 Create a File.

1. Go to the repository list page and click the name of the target repository.
2. On the **Code** tab, click **Create** and choose **Create File** from the drop-down list box. The page for creating a file is displayed.



3. Enter the following information, retain the default values for other parameters, and click **OK**.

File name: user-defined file name, for example, **Sample_Code**.

File content: user-defined file content.

Commit message: Enter the prefix and work item number in the E2E settings, for example, 708635317.

Create File

Sample_Code Empty file (no template) text base64

```
1 <project name="javaAntDemo" basedir="." default="main">
2   <property environment="env" />
3   <property name="src.dir" value="com"/>
4
5   <property name="build.dir" value="build"/>
6   <property name="classes.dir" value="${build.dir}/classes"/>
7   <property name="jar.dir" value="${build.dir}/jar"/>
8   <property name="report.dir" value="${build.dir}/junitreport"/>
9   <taskdef name="findbugs" classname="edu.umd.cs.findbugs.anttask.FindBugsTask"/>
10
11
12   <path id="application" location="${jar.dir}/${ant.project.name}.jar"/>
13
14   <property name="main-class" value="com.g42.HelloWorld"/>
15
16
17
18   <target name="clean">
19     <delete dir="${build.dir}"/>
20   </target>
21
```

Commit Message

Incorporated requirements:708635317

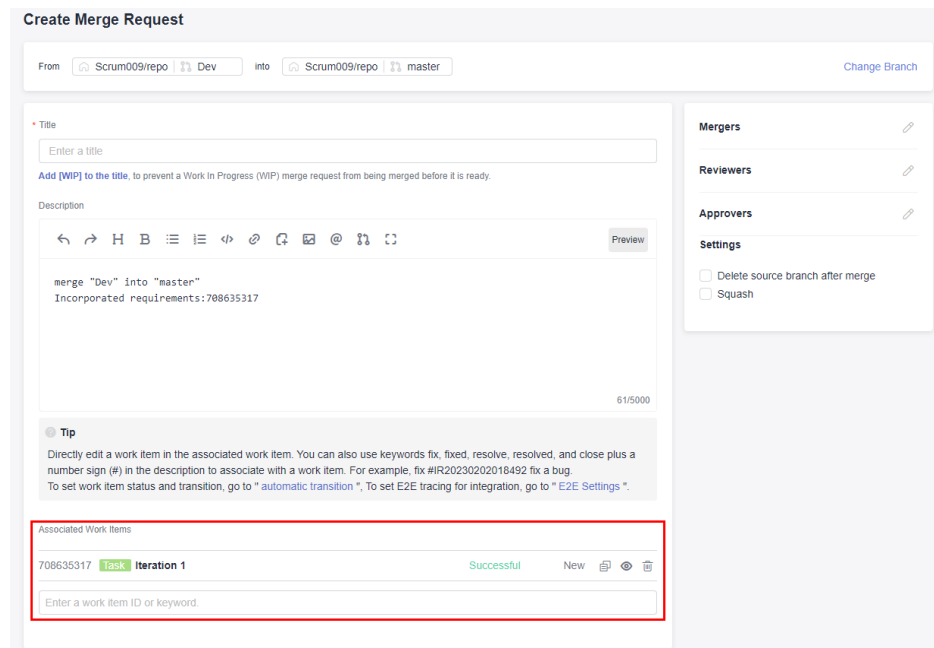
You can add 1965 more characters.

Tip
You can use keywords "fix", "fixed", "resolve", "resolved", and "close" to associate the file with a work item in the project. For example, "fix #IR20230202018492 fix a bug."

Step 4 Extract the ticket number when creating a merge request.

1. Switch to the **Merge Requests** tab and click **New**.
2. Select **Dev** as the source branch and **master** as the target branch, and click **Next**. The page for creating a merge request is displayed.

At this point, the work item is automatically extracted to the merge request.



----End

9.4.2 Webhooks

Introduction to Webhook

Developers can configure URLs of third-party systems on the Webhook page and subscribe to events such as branch push and tag push of CodeArts Repo based on project requirements. When a subscription event occurs, you can use a webhook to send a POST request to the URL of a third-party system to trigger operations related to your system (third-party system), such as popping up a notification window, building or updating images, or performing deployment.

If you want to email repository change notifications, you can configure [Notifications](#) in **General Settings**.

Configuring Webhooks

To configure webhooks, you can choose **Settings > Service Integration > Webhooks** on the repository details page.

The settings take effect only for the repository configured.

Only the repository administrator and owner can view the page and have the setting permission.

Table 9-10 Parameters for creating a webhook

Parameter	Description
Name	Custom name.

Parameter	Description
Description	Description of the webhook.
URL	(Mandatory) Provided by the third-party CI/CD system.
Token type	Used for webhook interface authentication of third-party services. The options are as follows: <ul style="list-style-type: none">● X-Repo-Token● X-Gitlab-Token● X-Auth-Token
Token	Used for third-party CI/CD system authentication. The authentication information is stored in the HTTP request header.
Event type	<p>The system can subscribe to the following events:</p> <ul style="list-style-type: none">● Push events<ul style="list-style-type: none">– If Push events is selected, Regular Expression for Branch Filtering is displayed.<p>NOTE Regular Expression for Branch Filtering: The default value is <code>.*</code>, indicating that all branches are matched. Max. 500 characters. The regular expression for branch filtering must comply with regular expressions.</p>– This event is triggered when code is updated in CodeArts Repo, such as code update in LFS files or submodules, and code pushed online or on a local Git client. ● Tag push events<p>This event is triggered when a tag is created or deleted.</p> ● Merge request events<ul style="list-style-type: none">– This event is triggered when a merge request is created.– This event is triggered when a merge request is updated. For example, when someone updates the code content, merge request status (closed or re-opened), merge request title or description, merger, and work items, deletes the source branch, and updates the squash.– This event is triggered when a request is merged. ● Comments<ul style="list-style-type: none">– This event is triggered when a review is added. For example, add a review for a file on the Files and Commits submenus of the Code tab page, or on the Files Changed submenu of the Merge Requests tab page.– This event is triggered when a comment is added on the Commits details page or on the Details page of Merge Requests tab page.

 NOTE

- A maximum of 20 webhooks can be created for a repository.
- You can configure a token when setting up a webhook. The token will be associated with the webhook URL and sent to you in the **X-Repo-Token** header.

9.5 Security Management

9.5.1 Deploy Keys

The deploy key is the public key of the SSH key generated locally. However, the deploy keys and SSH keys of a repository cannot be the same. Deploy keys allow you to clone repositories with read only access over SSH. They are mainly used in scenarios such as repository deployment and continuous integration.

 NOTE

- Multiple repositories can use the same deploy key, and a maximum of 10 deploy keys can be added to a repository.
- The difference between an SSH key and repository deploy key is that the former is associated with a user and PC and the latter is associated with a repository. The SSH key has the read and write permissions on the repository, and the deploy key has the read-only permission on the repository.
- The settings take effect only for the repository configured.
- Only the repository administrators and owners can view this tab page and configure deploy keys.

To configure the deploy keys, choose **Settings > Security Management > Deploy Keys** on the repository details page. The deploy key is a key that has only the read-only permission on the repositories.

Click **Add Deploy Key** to create a deploy key. For details about how to generate a local key, see [Generating and Configuring an SSH Key](#).

9.5.2 IP Address Whitelists

About IP Address Whitelists

- An IP address whitelist includes an IP address segment and several access control settings. The whitelist restricts users' access, upload, and download permissions to enhance repository security.
- The IP address whitelist can be configured only for repositories whose visibility is **Private**. Repositories whose visibility is **Public** or **Public template** are not supported.

IP Address Whitelist Formats

IPv4 and IPv6 are supported. The following table lists the three formats of IP address whitelists.

Table 9-11 IP address whitelist formats

Format	Description
Specified IP Address	This is the simplest IP address whitelist format. You can add the IP address of your PC to the whitelist, for example, 100.*.*.123.
IP address segment	If you have multiple servers and their IP addresses are consecutive or the IP address of your server dynamically changes in a network segment, you can add the IP address segment, for example, 100.*.*.0 to 100.*.*.255.
CIDR block	<ul style="list-style-type: none">When your server on a LAN uses the CIDR, you can specify a 32-bit egress IP address of the LAN and the number of bits for a specified network prefix.Requests from the same IP address are accepted if the network prefix is the same as the specified one.

Configuring IP Address Whitelists

IP address whitelists can be created in the following levels:

NOTE

If the **Private** repository for which the IP address whitelist has been configured is switched to a **Public** or **Public template** repository, you can also upload and download code on the CodeArts Repo page or Git client.

IP Address whitelists. The whitelists are set for all cloud services. IP addresses that are not in the whitelist are blocked upon login. For details, see [Access Control](#).

- **IP address whitelist for repository.** It allows access only from IP addresses in the whitelist to a specific repository. To set the whitelist, choose **Settings > Security Management > IP Address Whitelist** (IPv4 and IPv6 addresses are supported. For details, see [IP Address Whitelist Formats](#)).

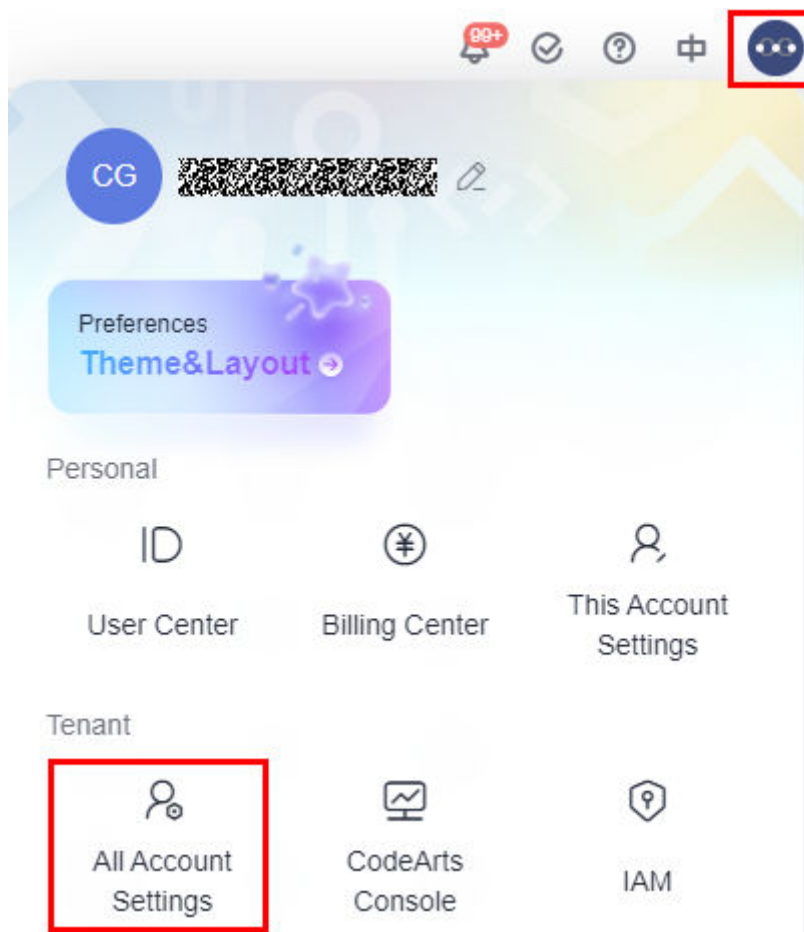
Allowed to access the repository: Only whitelisted IP addresses and the repository creator can access the repository.

Allowed to download code : Only whitelisted IP addresses can download code online and clone code locally.


Allowed to commit code: Only whitelisted IP addresses can modify and upload code online, or commit code locally. Code-based build project orchestration and YAML file synchronization are not affected.

NOTE

- **Commit code:** Create, edit, delete, upload and rename files, create and delete directories, submodules, branches, and tags, resolve code conflicts, create and merge MRs, cherry-pick, revert, use LFS storage, and rebase.
 - **Download code:** Download a single file and branches, tags, repositories and backup repositories.
 - **Local download:** Download code through SSH and HTTPS, and clone repository through deploying keys.
 - **Local commit:** Commit code through SSH and HTTPS.
 - Repository synchronization is not affected by the IP address whitelist.
- **Tenant-level IP address whitelist:** To set IP address whitelists for repositories of all accounts from a tenant, log in to the CodeArts Repo repository list page, click the alias in the upper right corner, and choose **All Account Settings > Repo > Whitelists for All Accounts**, as shown in the following figure. The configuration rules are the same as those of [repository-level IP address whitelists](#).



Only tenant accounts have permissions to configure **Whitelist for All**

Accounts. Click  next to **Add Address** and select **Prioritize this List**. For details about the logic of cloning the Git client or downloading the repository source code on the UI, see the following table.

Account-level Whitelist Prioritized (Prioritize This List)	Configure Tenant-level Whitelist	Configure Repository-level Whitelist	Priority
Enabled	×	×	All IP addresses are allowed.
	×	√	The repository-level whitelist prevails.
	√	×	The tenant-level whitelist prevails.
	√	√	The intersection of the tenant-level whitelist and repository-level whitelist prevails.
Disabled	×	×	All IP addresses are allowed.
	×	√	The repository-level whitelist prevails.
	√	×	The tenant-level whitelist prevails.
	√	√	The repository-level whitelist prevails.

9.5.3 Risky Operations

To configure risky operations, choose **Settings > Security Management > Risky Operations** on the repository details page.

Only the repository administrators and owners can view this tab page and configure risky operations.

Risky operations are as follows:

- **Transfer repository ownership:** The ownership of a repository can be transferred to another person in the repository but cannot be transferred to a viewer or custom role.
- **Delete repository:** The repository cannot be recovered after being deleted.
- **Rename repository:** After renaming a repository, check the configuration related to the repository name in a timely manner.

9.5.4 Watermarks

On the repository details page, choose **Settings > Security Management > Watermark**. The watermark content consists of your account name and current time.

Only repository administrators and owners can view this tab page and configure the watermark function.

Watermarks will be displayed on code repository pages to reduce the risk of code asset leakage.

Watermark

Watermarks protect your company's core assets. Use them to deter and track dissemination by photos, screenshots, and other unauthorized means.



9.5.5 Repository Locking

When a new software version is ready for release, administrators can lock the repository to protect it from being compromised. After the repository is locked, no one (including the administrators) can commit code to any of its branches.

To lock a repository, choose **Settings > Security Management > Repository Locking** on the repository details page.

Only the repository administrator and owner can view the page and have the setting permission.

After the administrator locks the repository, no one can use the repository functions in [Table 9-12](#).

Table 9-12 List of functions that cannot be executed

Tab Page	Function
Code	If the repository is locked, the following functions cannot be performed on the Code tab page: <ul style="list-style-type: none">• Create, edit, delete, rename, and upload a file• Create and delete a directory• Create and delete a submodule• Cherry-Pick and revert a file• Add, delete, edit, reply, and resolve a review and comment
Branch & Tag	If the repository is locked, the following functions cannot be performed on the Branch or Tag subtab of the Code tab page: <ul style="list-style-type: none">• Create, edit, and delete a branch, merge branches, and sett protected branches.• Create and delete a Tag
Merge Requests	If the repository is locked, the following functions cannot be performed on the Merge Requests details page: <ul style="list-style-type: none">• Create, edit, close, re-open, and merge a merge request• Cherry-Pick and revert a merge request• Resolve a code conflict• Add, delete, edit, reply, and resolve a review comment

Tab Page	Function
Repository & Members	If the repository is locked, the following functions cannot be performed: <ul style="list-style-type: none">• Fork a repository• Add, delete, edit, and approve a member
Settings	If the repository is locked, the following functions cannot be performed on the Settings tab page: <ul style="list-style-type: none">• Repository settings• Submodules• Deploy key synchronization• Space freeing• Policy settings (All)• Service integration (All)

 **NOTE**

After the repository is locked, changes to project members will be synchronized to the repository, affecting repository members.

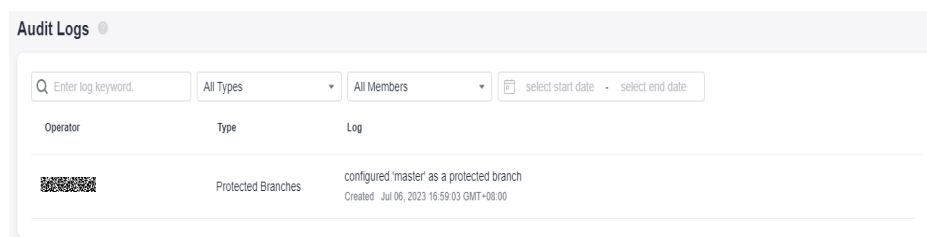
9.5.6 Audit Logs

To view audit logs, choose **Settings > Security Management > Audit Logs** on the repository details page.

Only the repository administrators and owners can view this tab page.

Audit logs record only changes to repository attributes. Check daily development activities such as MRs, reviews, and member changes from [repository dynamics](#).

You can filter logs by time segment, operator, operation type, or log information. The operation types include repository information, submission rule, merge request, and merge request policy.



10 Submitting Code to the CodeArts Repo

[10.1 Creating a Commit](#)

[10.2 Transmitting and Storing a File in Encryption Mode](#)

[10.3 Viewing Commit History](#)

[10.4 Pushing Code to CodeArts Repo Using Eclipse](#)

10.1 Creating a Commit

Background

In code development, developers usually clone code from CodeArts Repo to the local PC to develop code locally, and then commit the code to CodeArts Repo after completing the phased development task. This section describes how to use the Git client to commit the modified code.

Prerequisites

1. Git Installation and Configuration.
2. You have created a repository in CodeArts Repo. For details, see [Overview](#).
3. You have set the SSH keys or HTTPS password. For details, see [Setting SSH Key or HTTPS Password for CodeArts Repo Repository](#)
4. You have Cloned the CodeArts Repo Repository to the Local Host. For details, see [Overview](#).

Procedure

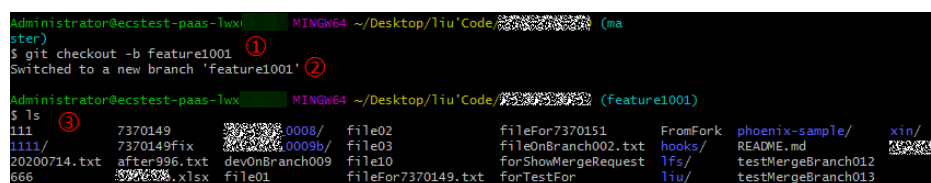
Generally, developers do not directly develop code in the master branch. Instead, they create a feature branch based on the master or develop branch, and develop code in it. Then they commit the **feature** branch to CodeArts Repo, and merge it into the **master** or **develop** branch. The preceding operations are simulated as follows:

- Step 1** Go to the local repository directory and open the Git client. Take Git Bash as an example. The principles and commands of other Git management tools are the same.
- Step 2** Create a **feature1001** branch based on the **master** branch, switch to the created branch, and run the following command in the **master** branch:

```
git checkout -b feature1001 #Shown in 1 in the following figure.
```

This command creates a branch and then switches to the branch.

If the command is successfully executed, 2 in the following figure is shown. You can run the **ls** command to view the files of the branch (as shown in 3 in the following figure), which are the same as those of the **master** branch currently.



```
Administrator@ecstest-paas-lwx: MINGW64 ~/Desktop/liu'Code (ma
ster)
$ git checkout -b feature1001 ①
Switched to a new branch 'feature1001' ②

Administrator@ecstest-paas-lwx: MINGW64 ~/Desktop/liu'Code (feature1001)
$ ls ③
111/ 7370149 0008/ file02 fileFor7370151 FromFork phoenix-sample/ xiu/
1111/ 7370148fix 0009b/ file03 fileOnBranch002.txt hooks/ README.md
20200714.txt after996.txt devOnBranch009 file10 ForShowMergeRequest lfs/ testMergeBranch012
666 20200714.txt xlsx file01 fileFor7370149.txt ForTestFor liu/ testMergeBranch013
```

- Step 3** Modify code in the **feature** branch (code development).

Git supports Linux commands. In this case, the **touch** command is used to create a file named **newFeature1001.html**, indicating that the developer has developed new features locally and a new file is added into the local code repository.

```
touch newFeature1001.html
```

Run the **ls** command again to view the created file.

- Step 4** Run the **add** and **commit** commands to add the file from the working directory to the staging area, and then commit the file to the local repository. (For details, see [1 Overview.](#))

You can also run the **status** command to check the file status.

1. Run the **status** command. The command output shows that a file in the working directory is not included in version management, as shown in 1 in the following figure.
2. Run the **add** command to add the file to the staging area, as shown in 2 in the following figure.

```
git add . # Period (.) means all files, including hidden files. You can also specify a file.
```
3. Run the **status** command. The command output shows that the file has been added to the staging area and is waiting to be committed, as shown in 3 in the following figure.
4. Run the **commit** command to commit the file to the local repository, as shown in 4 in the following figure.

```
git commit -m "<your_commit_message>"
```
5. Check the file status again. If no file to be committed exists, the commit is successful, as shown in 5 in the following figure.

```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git status ①
On branch feature1001
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        newFeature1001.html

nothing added to commit but untracked files present (use "git add" to track)

Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git add . ②

Administrator@ecstest-paas-lwx69 MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git status ③
On branch feature1001
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newFeature1001.html

Administrator@ecstest-paas-lwx60 MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git commit -m "This is a commit for feature1001~!" ④
[feature1001 4c8db12] This is a commit for feature1001~!
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFeature1001.html

Administrator@ecstest-paas-lw MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git status ⑤
On branch feature1001
nothing to commit, working tree clean
```

Step 5 Push a local branch to CodeArts Repo.

```
git push --set-upstream origin feature1001
```

Run the preceding command to create a branch that is the same as your local **feature1001** branch in CodeArts Repo, and associate them and synchronize the branch.

origin indicates the alias of your CodeArts Repo. The default alias of a directly controllable repository is **origin**. You can also use the repository address.

```
Administrator@ecstest-paas-lwx60 MINGW64 ~/Desktop/liu'Code/ (Feature1001)
$ git push --set-upstream origin feature1001
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 287.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for feature1001, visit:
remote: https://
remote:
To
 * [new branch]      feature1001 -> feature1001
Branch 'feature1001' set up to track remote branch 'feature1001' from 'origin'.
```

NOTE

If the push fails, check the connectivity.

- Check whether your network can access CodeArts Repo.

Run the following command on the Git client to test the network connectivity:

```
ssh -vT git@*****.com
```

If the command output contains **connect to host *****.com port 22: Connection timed out**, your network is restricted from accessing CodeArts Repo. Contact your network administrator.

- Check the SSH key. If necessary, regenerate a key and configure it on the CodeArts Repo console. For details, see [3.2 SSH Keys](#). Alternatively, check [whether the HTTPS password](#) is correctly configured.
- Check the [IP address whitelist](#). If no whitelist is configured, all IP addresses are allowed to access the repository. If a whitelist is configured, only IP addresses in the whitelist are allowed to access the repository.

Step 6 View the CodeArts Repo repository branch.

Log in to CodeArts Repo and go to your repository. In the **Files** tab page, you can switch to your branch in CodeArts Repo.

 **NOTE**

If the branch you just committed is not displayed, your **origin** may be bound to another repository. Use the repository address to commit the branch again.

Step 7 Create a merge request. For details, see [8.5.1 Managing MRs](#). Notify the approver to review the request and merge the new feature into the master or develop branch.

----End

10.2 Transmitting and Storing a File in Encryption Mode

CodeArts Repo uses git-crypt for encrypted storage and transmission of confidential and sensitive files.

About git-crypt

git-crypt is a third-party open-source software that can transparently encrypt and decrypt files in the Git repository. It can encrypt and store specified files and file types. Developers can store encrypted files (such as confidential information or sensitive data) and shared code in the same repository and pull and push them like in a common repository. Only the person who has the corresponding file key can view the content of the encrypted files, but others are not restricted to read and write unencrypted files.

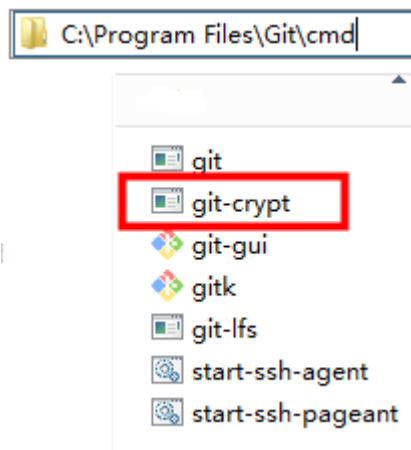
git-crypt allows you to encrypt only specific files without locking the entire repository, facilitating team cooperation and ensuring information security.

Using Key Pairs for Encryption and Decryption on Windows

Step 1 Install and initialize Git.

Step 2 Download the latest [Windows-based git-crypt](#) and save the downloaded .exe file to the **cmd** folder in the Git installation directory. The following figure uses the default Git Bash installation path of **Windows Server 2012 R2 Standard (64-bit)** as an example.

Put the .exe file in the folder. You do not need to run it.



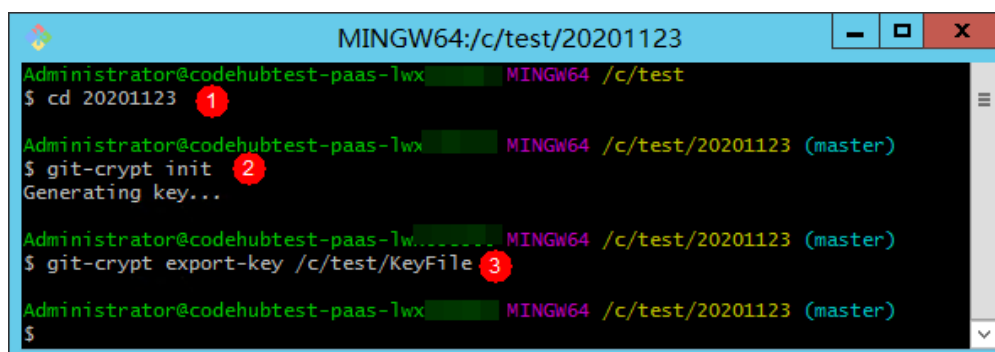
Step 3 Generate a key pair.

1. Open **Git Bash** and go to the local repository, as shown in 1 in the following figure.
2. Run the following command to generate a key pair, as shown in 2 in the following figure.

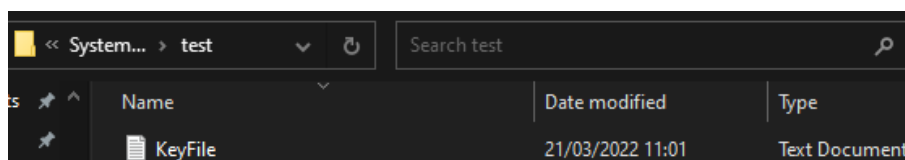
```
git-crypt init
```

3. Export the key file. In this example, the key file is exported to the **C:\test** directory and named **KeyFile**. Run the following command, as shown in 3 in the following figure.

```
git-crypt export-key /c/test/keyfile
```



4. Check whether the key is generated in the file path where the key is exported. In this example, check whether the **KeyFile** file exists in the **C:\test** directory, as shown in the following figure.



The computer containing the key file can decrypt the corresponding encrypted file.

Step 4 Configure the encryption scope for the repository.

1. Create a file named **.gitattributes** in the root directory of the repository.
2. Open the **.gitattributes** file and run the following command to set the encryption range.

```
<file_name_or_file_range>: filter=git-crypt diff=git-crypt
```

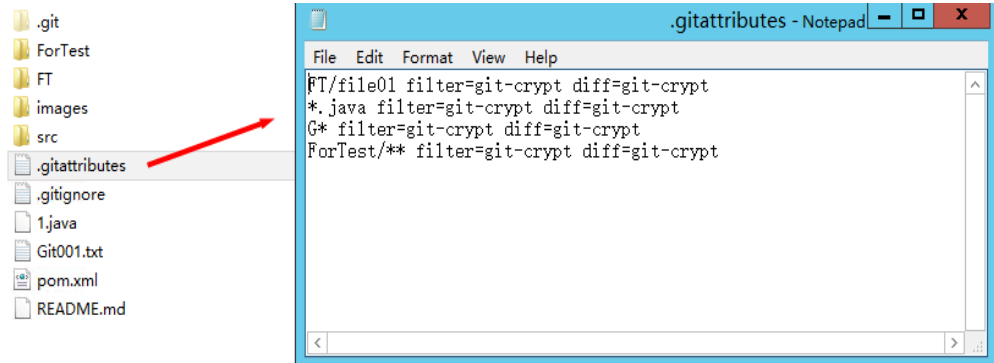
Four examples are as follows:

```
FT/file01.txt filter=git-crypt diff=git-crypt # Encrypt a specified file. In this example, the file01.txt file in the FT folder is encrypted.
```

```
*.java filter=git-crypt diff=git-crypt # The .java file is encrypted.
```

```
G* filter=git-crypt diff=git-crypt # Files which names start with G are encrypted.
```

```
ForTest/** filter=git-crypt diff=git-crypt # Files in the ForTest folder are encrypted.
```



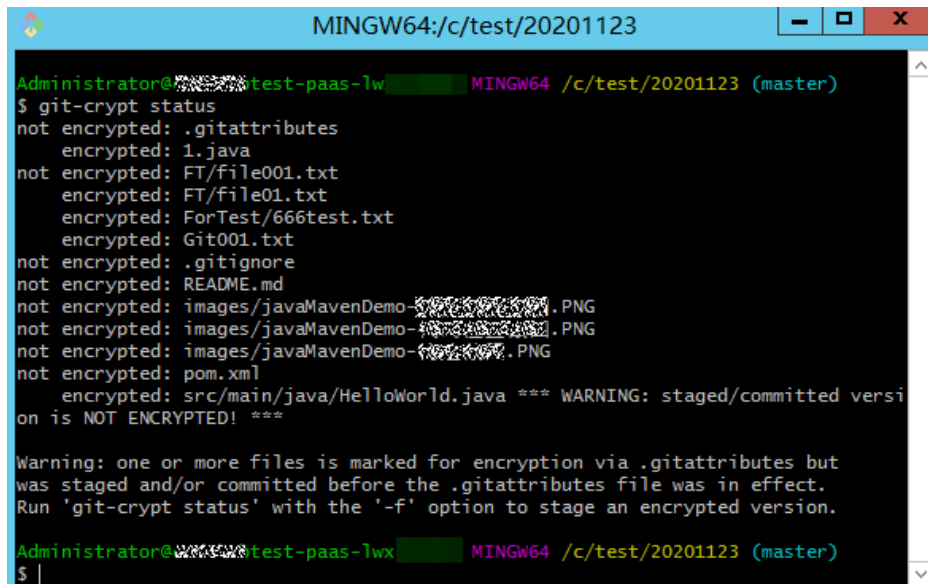
NOTE

- If the system prompts you to **enter the file name** when you create the **.gitattributes** file, you can enter **.gitattributes**. to create the file. If you run the Linux command to create the file, this problem does not occur.
- Do not save the **.gitattributes** file as a **.txt** file. Otherwise, the configuration does not take effect.

Step 5 Encrypt the file.

Open Git Bash in the root directory of the repository and run the following command to encrypt the file. The encryption status of the file is displayed.

git-crypt status



```
Administrator@...test-paas-lw MINGW64 /c/test/20201123 (master)
$ git-crypt status
not encrypted: .gitattributes
encrypted: 1.java
not encrypted: FT/file001.txt
encrypted: FT/file01.txt
encrypted: ForTest/666test.txt
encrypted: Git001.txt
not encrypted: .gitignore
not encrypted: README.md
not encrypted: images/javaMavenDemo-...PNG
not encrypted: images/javaMavenDemo-...PNG
not encrypted: images/javaMavenDemo-...PNG
not encrypted: pom.xml
encrypted: src/main/java/HelloWorld.java *** WARNING: staged/committed version is NOT ENCRYPTED! ***

Warning: one or more files is marked for encryption via .gitattributes but
was staged and/or committed before the .gitattributes file was in effect.
Run 'git-crypt status' with the '-f' option to stage an encrypted version.

Administrator@...test-paas-lw MINGW64 /c/test/20201123 (master)
$
```

After the encryption, you can still open and edit the encrypted files in plaintext in your local repository because your local repository has a key.

You can run the **add**, **commit**, and **push** commands to push the repository to CodeArts Repo. In this case, the encrypted files are pushed together.

Encrypted files are stored in CodeArts Repo as encrypted binary files and cannot be viewed directly. If you do not have a key, you cannot decrypt it even if you download it to the local computer.

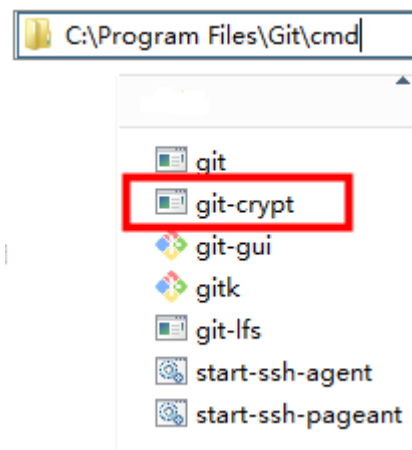
NOTE

git-crypt status encrypts only the files to be committed this time. It does not encrypt the historical files that are not modified this time. Git displays a message for the unencrypted files involved in this setting (see **Warning** in the preceding figure). If you want to encrypt all files of a specified type in the repository, run the **git-crypt status -f** command.

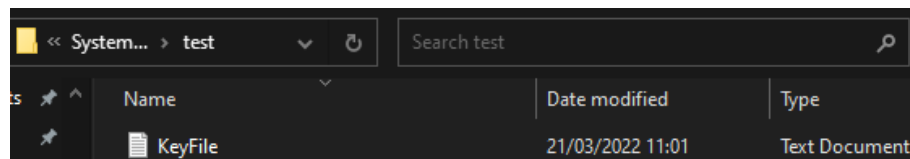
In team cooperation, **-f** (forcible execution) has certain risks and may cause the members' work output to remain unchanged. Exercise caution when using **-f**.

Step 6 Decrypt the file.

1. Ensure that the **git-crypt** file exists in the Git installation path on the local computer.



2. Clone the repository from CodeArts Repo to the local host.
3. Obtain the key file for encrypting the repository and store it on the local computer.



4. Go to the repository directory and right-click Git Bash.
5. Run the decryption command. If no command output is displayed, the command is successfully executed.

```
git-crypt unlock /C/test/KeyFile # Replace /C/test/KeyFile with the actual key storage path.
```

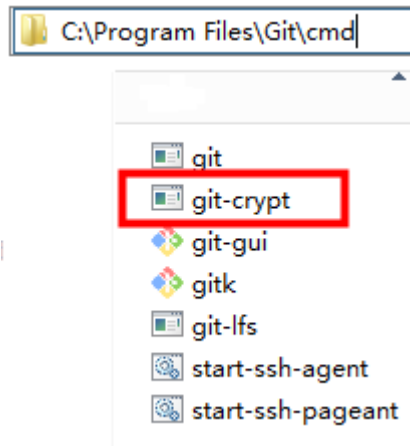
----End

Encrypting and Decrypting a File in GPG Mode on Windows

Step 1 Install and initialize Git.

Step 2 Download the latest [Windows-based git-crypt](#) and save the downloaded .exe file to the **cmd** folder in the Git installation directory. The following figure uses the default Git Bash installation path of **Windows Server 2012 R2 Standard (64-bit)** as an example.

Put the .exe file in the folder. You do not need to run it.



Step 3 **Download the GPG** of the latest version. When you are prompted to donate the open-source software, select **0** to skip the donation process.

OS	Where	Description
Windows	Gpg4win	Full featured Windows version of <i>GnuPG</i>
	download sig	Simple installer for the current <i>GnuPG</i>
	download sig	Simple installer for <i>GnuPG 1.4</i>
OS X	Mac GPG	Installer from the gpgtools project
	GnuPG for OS X	Installer for <i>GnuPG</i>
Debian	Debian site	<i>GnuPG</i> is part of Debian
RPM	rpmfind	RPM packages for different OS
Android	Guardian project	Provides a <i>GnuPG</i> framework
VMS	antinode.info	A port of <i>GnuPG 1.4</i> to OpenVMS
RISC OS	home page	A port of <i>GnuPG</i> to RISC OS

Double-click to start the installation. Click **Next** to complete the installation.

Step 4 **Generate a key pair in GPG mode.**

1. Open Git Bash and run the following command:
`gpg --gen-key`
2. Enter the name and email address as prompted.

```
Administrator@codehubtest-paas- [REDACTED] MINGW64 /c/dev/test
$ gpg --gen-key
gpg (GnuPG) 2.2.23-unknown; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

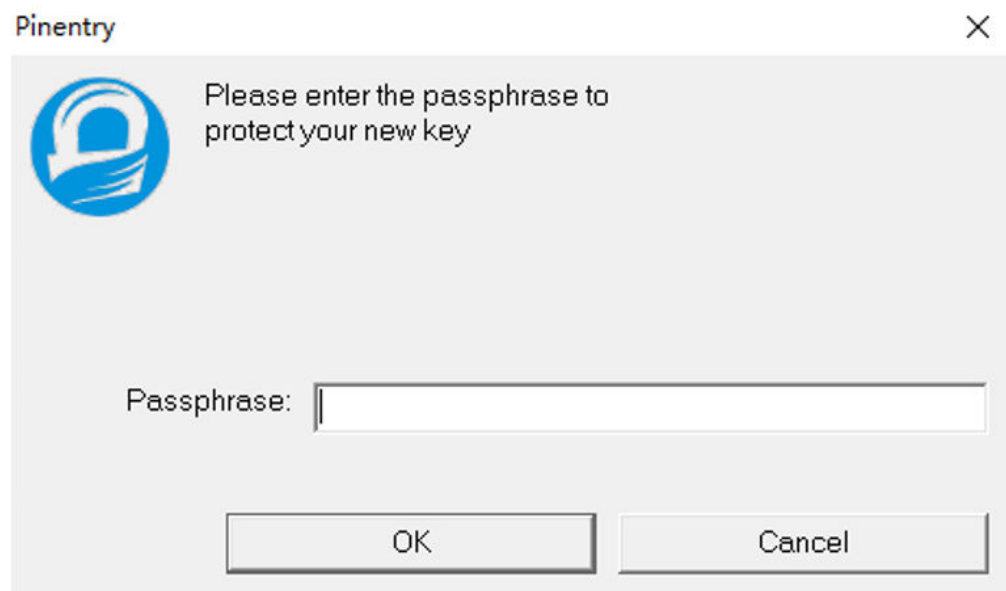
gpg: directory '/c/Users/Administrator/.gnupg' created
gpg: keybox '/c/Users/Administrator/.gnupg/pubring.kbx' created
Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: gpgTest
Email address: gpgTest@huahua.com
You selected this USER-ID:
    "gpgTest <gpgTest@huahua.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? |
```

3. Enter **o** as prompted and press **Enter**. The dialog boxes for entering and confirming the password are displayed.



The password can be empty. To ensure information security, you are advised to enter a password that complies with the standard (this password is required for decryption).

4. If the following information is displayed, the GPG key pair is generated successfully.

```
public and secret key created and signed.

pub  rsa3072 2020-11-24 [SC] [expires: 2022-11-24]
     OD[REDACTED] 71E0AD
uid  gpgTest <gpgTest@huahua.com>
sub  rsa3072 2020-11-24 [E] [expires: 2022-11-24]
```

Step 5 Initialize the repository encryption.

1. Open Git bash in the root directory of the repository and run the following command to initialize the repository:

```
git-crypt init
```

```
Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test
$ cd 20201124

Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ git-crypt init
Generating key...

Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ |
```

2. Run the following command to add a copy of the key to your repository. The copy has been encrypted using your public GPG key.

```
git-crypt add-gpg-user USER_ID
```

USER_ID can be the name, email address, or fingerprint that uniquely identifies the key, as shown in 1, 2, and 3 in the following figure in sequence.

```
public and secret key created and signed.

pub  rsa3072 2020-11-24 [SC] [expires: 2022-11-24]
     ③ ODI 71E0AD
uid  ① gpgTest <gpgTest@huahua.com> ②
sub  rsa3072 2020-11-24 [E] [expires: 2022-11-24]
```

After the command is executed, a message is displayed, indicating that the **.git-crypt** folder and two files in it are created.

```
MINGW64:/c/dev/test/20201124
Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ git-crypt add-gpg-user gpgTest
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2022-11-24
[master 2e4aa2b] Add 1 git-crypt collaborator
2 files changed, 4 insertions(+)
create mode 100644 .git-crypt/.gitattributes
create mode 100644 .git-crypt/keys/default/0/ODDF227
71E0AD.gpg

Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ |
```

Step 6 Configure the encryption scope for the repository.

1. Go to the **.git-crypt** folder in the repository.
2. Open the **.gitattributes** file and run the following command to set the encryption range.

```
<file_name_or_file_range>: filter=git-crypt diff=git-crypt
```

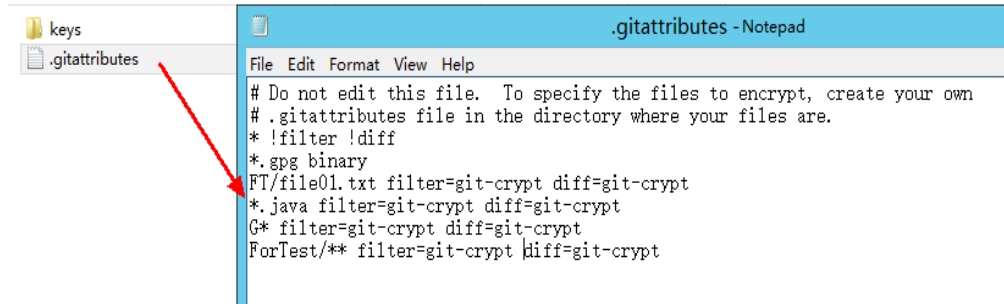
Four examples are as follows:

```
FT/file01.txt filter=git-crypt diff=git-crypt # Encrypt a specified file. In this example, the file01.txt file in the FT folder is encrypted.
```

```
*.java filter=git-crypt diff=git-crypt # The .java file is encrypted.
```

```
G* filter=git-crypt diff=git-crypt # Files which names start with G are encrypted.
```

```
ForTest/** filter=git-crypt diff=git-crypt # Files in the ForTest folder are encrypted.
```

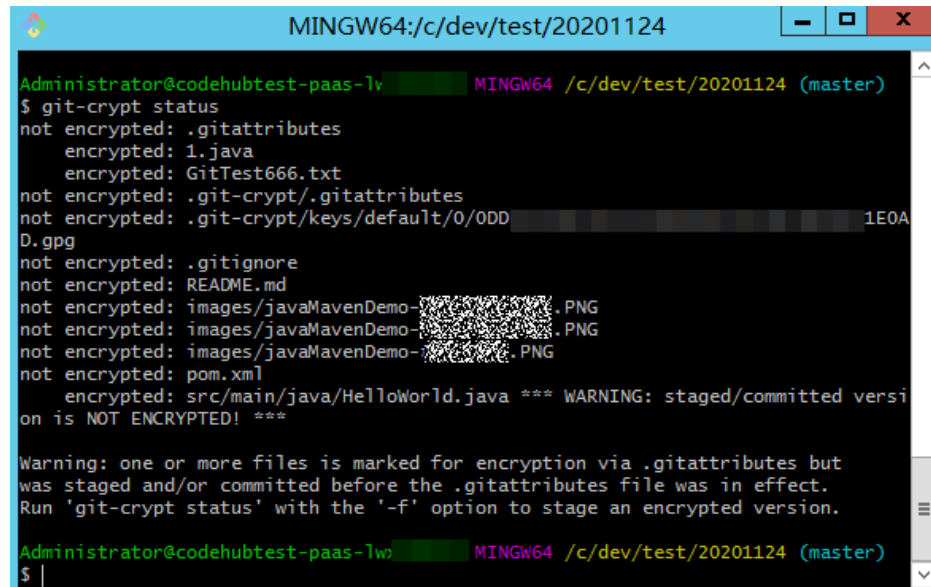


3. Copy the `.gitattributes` file to the root directory of the repository.

Step 7 Encrypt the file.

Open Git Bash in the root directory of the repository and run the following command to encrypt the file. The encryption status of the file is displayed.

```
git-crypt status
```



After the encryption, you can still open and edit the encrypted files in plaintext in your local repository because your local repository has a key.

You can run the **add**, **commit**, and **push** commands to push the repository to CodeArts Repo. In this case, the encrypted files are pushed together.

Encrypted files are stored in CodeArts Repo as encrypted binary files and cannot be viewed directly. If you do not have a key, you cannot decrypt it even if you download it to the local computer.

NOTE

git-crypt status encrypts only the files to be committed this time. It does not encrypt the historical files that are not modified this time. Git displays a message for the unencrypted files involved in this setting (see **Warning** in the preceding figure). If you want to encrypt all files of a specified type in the repository, run the **git-crypt status -f** command.

In team cooperation, **-f** (forcible execution) has certain risks and may cause the members' work output to remain unchanged. Exercise caution when using **-f**.

Step 8 Export the key.

1. Lists the currently visible keys. You can view the name, email address, and fingerprint of each key.

```
gpg --list-keys
```

```
Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ gpg --list-keys
/c/Users/Administrator/.gnupg/pubring.kbx
-----
pub   rsa3072 2020-11-24 [SC] [expires: 2022-11-24]
      ODD
uid   [ultimate] gpgTest <gpgTest@huahua.com>
sub   rsa3072 2020-11-24 [E] [expires: 2022-11-24]

Administrator@codehubtest-paas-1wx MINGW64 /c/dev/test/20201124 (master)
$ |
```

2. Run the **gpg --export-secret-key** command to export the keys. In this example, the **gpgTest** key is exported to **drive C** and named **Key**.
`gpg --export-secret-key -a gpgTest > /c/key # -a indicates that the key is displayed in text format.`

During the execution, the system prompts you to enter the key password. Enter the correct password.

No command output is displayed. You can view the key file in the corresponding directory (**drive C** in this example).

3. Send the generated key to the team members to share the encrypted file.

Step 9 Import the key and decrypt the file.

1. To decrypt files on another computer, you need to download and install git-crypt and GPG based on Git. For details, see the previous steps in this section.
2. Clone the corresponding repository to the local host.
3. Obtain the key of the corresponding encrypted file. For details about how to export the key, see the previous step. In this example, the obtained key is stored in **drive C**.
4. Go to the repository, open Git Bash, and run the **import** command to import the key.

```
gpg --import /c/key
# /c/Key is the key path and user-defined key name in this example. Replace them with the actual ones.
```

During the import, the system prompts you to enter the password of the key. If the import is successful, the following figure is displayed.

5. Run the **unlock** command to decrypt the file.

```
git-crypt unlock
```

During the decryption, a dialog box is displayed, prompting you to enter the password of the key. If no command output is displayed after you enter the correct password, the decryption is successful.

```
Administrator@codehubtest-paas-1wx MINGW64 /c/dev001/20201124 (master)
$ gpg --import /c/Key
gpg: /c/Users/Administrator/.gnupg/trustdb.gpg: trustdb created
gpg: key 3E3E EOAD: public key "gpgTest <gpgTest@huahua.com>" imported
gpg: key 3E3E EOAD: secret key imported
gpg: Total number processed: 1
gpg:         imported: 1
gpg:         secret keys read: 1
gpg:         secret keys imported: 1

Administrator@codehubtest-paas-1wx MINGW64 /c/dev001/20201124 (master)
$ git-crypt unlock
```

Step 10 View the file before and after decryption.

----End

Application of git-crypt Encryption in Teamwork

In most cases, a team needs to store files that have **restricted disclosure** in the code repository. It can use **CodeArts Repo**, **Git**, and **git-crypt** to encrypt some files in the distributed open-source repository.

Generally, **Key pair encryption** can meet the requirements of restricting the access to some files.

When a team needs to set different confidential levels for encrypted files, the **GPG encryption** can be used. This encryption mode allows you to use different keys to encrypt different files in the same repository and share the keys of different confidential levels with team members, restricting file access by level.

Installing git-crypt and gpg on Linux and macOS

Installing git-crypt and gpg on **Linux**

- Linux installation environment

Software	Debian/Ubuntu Package	RHEL/CentOS Package
Make	make	make
A C++11 compiler (e.g. gcc 4.9+)	g++	gcc-c++
OpenSSL development files	libssl-dev	openssl-devel

- In Linux, install git-crypt by compiling the source code.

```
make  
make install
```

Install git-crypt to a specified directory

```
make install PREFIX=/usr/local
```

- In Linux, install GPG by compiling the source code.

```
./configure  
make  
make install
```

- Install git-crypt using the Debian package.

The Debian package can be found in the **debian** branch of the project Git repository.

The software package is built using **git-buildpackage**, as shown in the following figure.

```
git checkout debian  
git-buildpackage -uc -us
```

- Install GPG using the build package in Debian.

```
sudo apt-get install gnupg
```

Install git-crypt and GPG on macOS.

- Install git-crypt on macOS.
Run the following command to install git-crypt using the brew package manager.

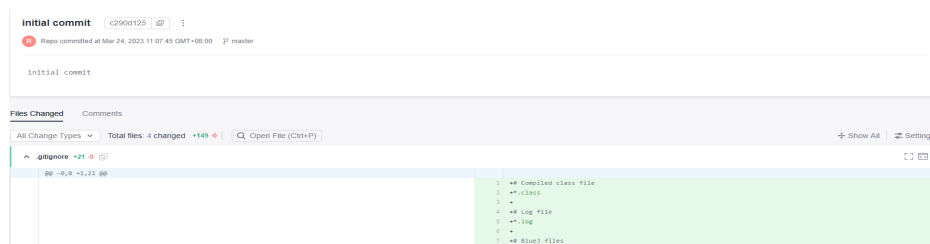
```
brew install git-crypt
```
- Install GPG on macOS.
Run the following command to install git-crypt using the brew package manager.

```
brew install gpg
```

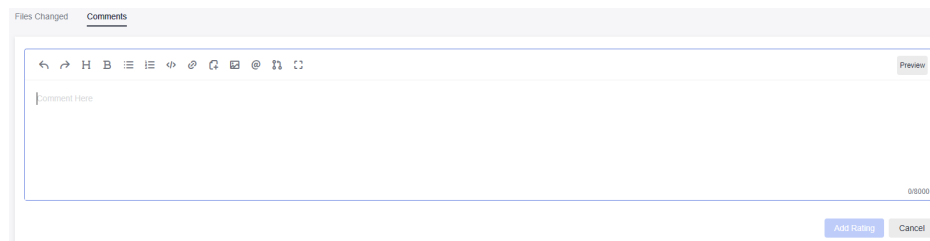
10.3 Viewing Commit History

CodeArts Repo allows you to view details about the commit history and related file changes.

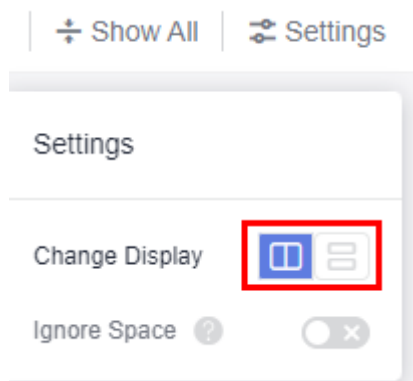
You can view the commit history on the **History** tab page of the **Files** or **repository dynamics**. You can click a commit record to view the committer, commit number, parent node, number of comments, and code change comparisons.



You can comment on a commit or reply a comment.



You can click the icon in the following figure to switch the horizontal or vertical display of code change comparison. You can click **Show All** to view the full text of the files involved in the commit.



10.4 Pushing Code to CodeArts Repo Using Eclipse

Background

You can install EGit on Eclipse so that Eclipse can be connected with CodeArts Repo and be used for operations such as committing code from a local Git repository to a remote one.

NOTE

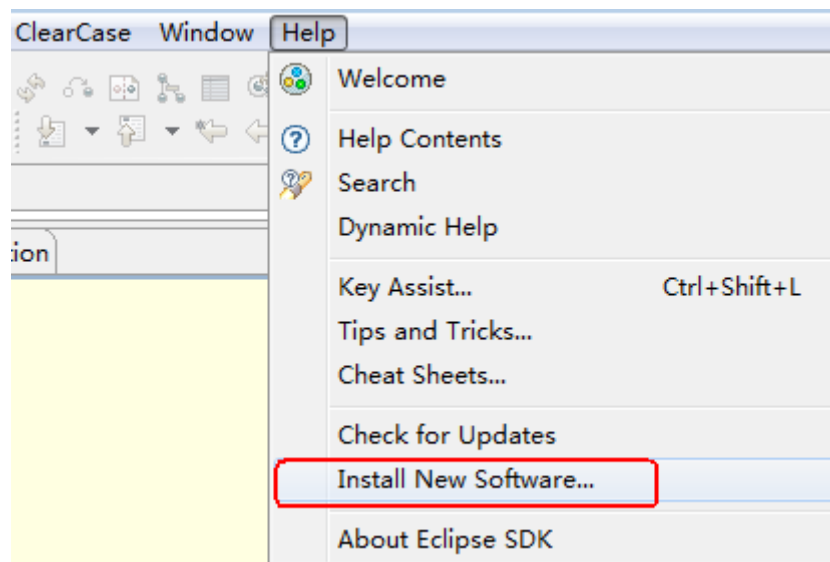
Only Eclipse 4.4 or later versions are supported.

- For the first push:
 1. Create a repository on the local computer, that is, the local repository.
 2. **Commit** the update to the local repository.
 3. Pull the code from the server to the local repository, merge the code, and push the repository to the server.
- If it is not the first push:
 1. Commit the modified code to the local repository.
 2. Pull the code from the server to the local repository, merge the code, and push the repository to the server.

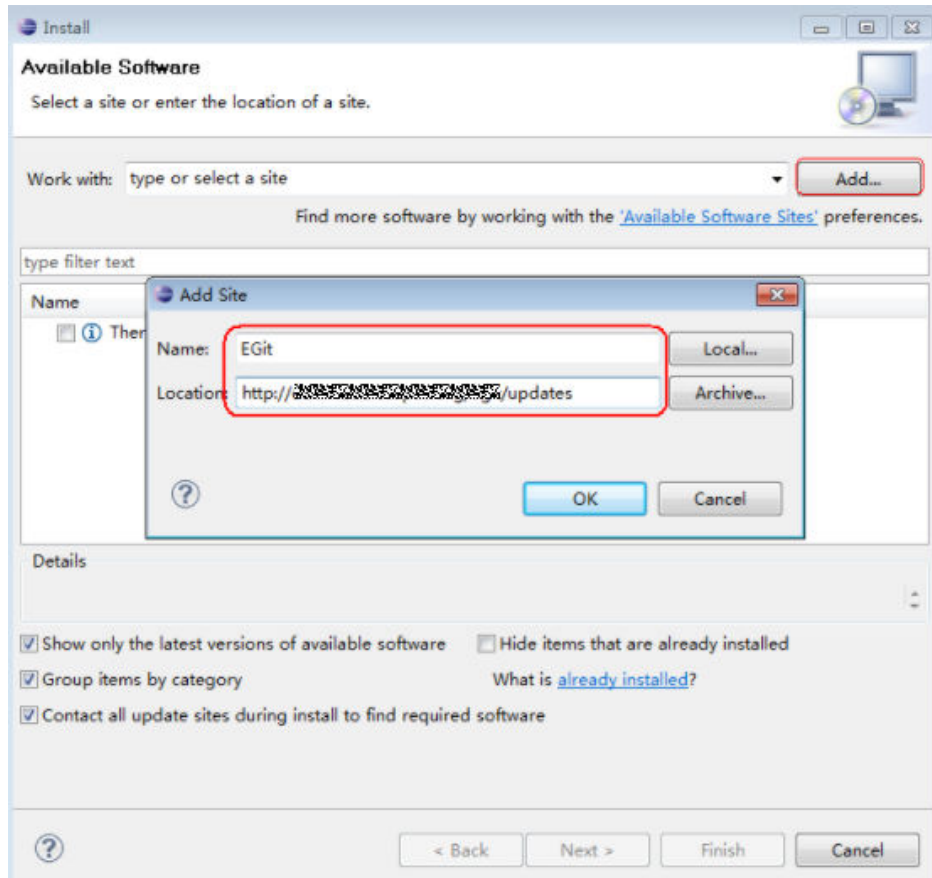
Step 1: Installing EGit on Eclipse

Eclipse 4.4 is used in the following procedure.

1. On the Eclipse toolbar, choose **Help > Install New Software...**



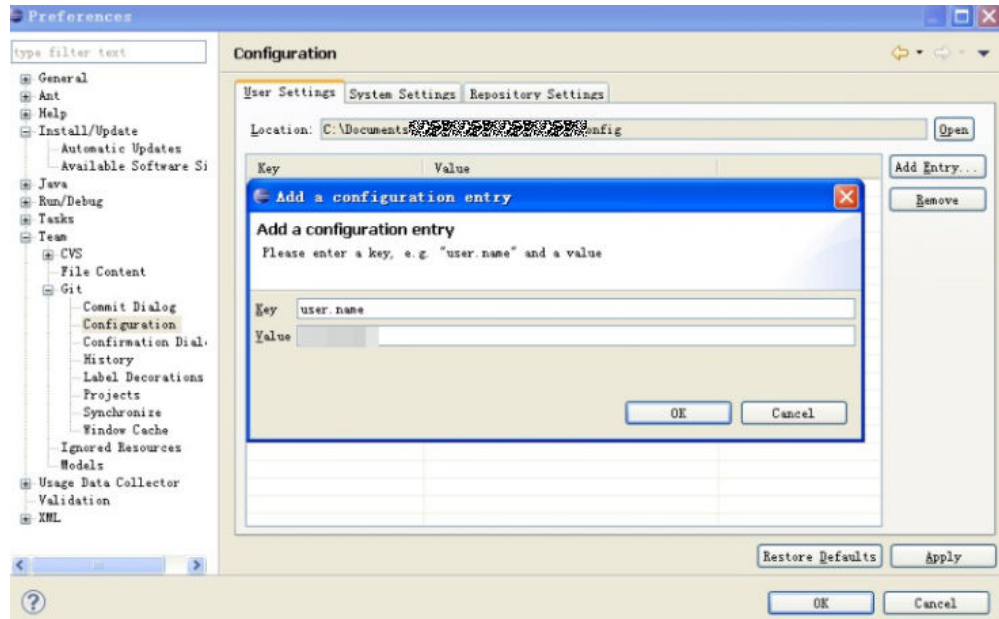
2. In the **Install** window displayed, click **Add...**
Set **Location** to <https://download.eclipse.org/egit/updates>.



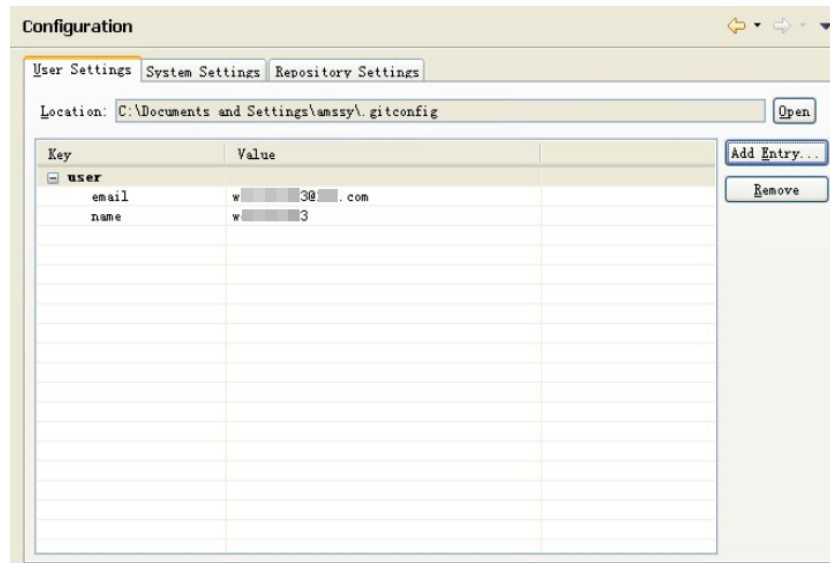
3. Click **OK**. Then, click **Next** until the installation is finished.
Restart Eclipse after the installation.

Step 2: Configuring EGit on Eclipse

1. On the Eclipse toolbar, choose **Window > Preferences > Team > Git > Configuration**.
Set **Key** to a registered username.

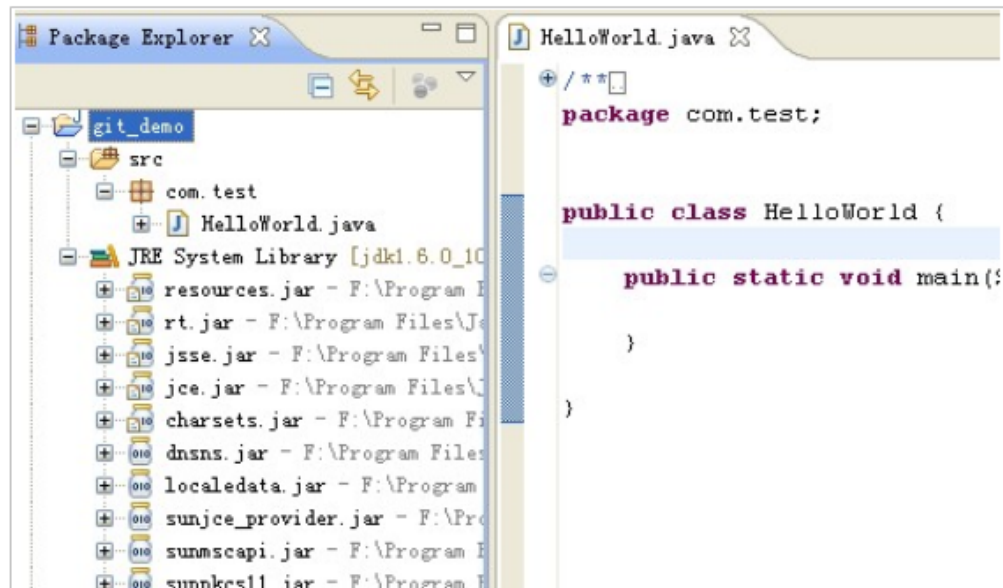


2. Click **OK**.
email indicates the bound email address. If the username is not set previously, set it in this step.

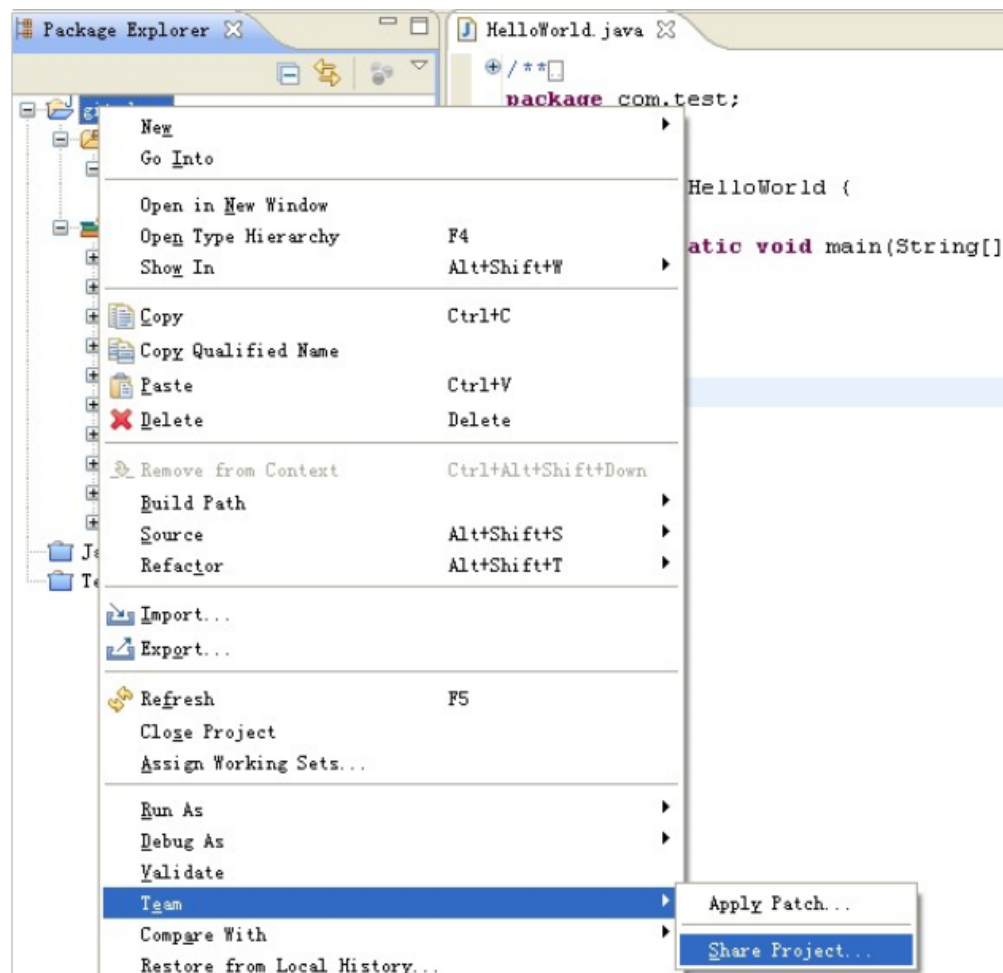


Step 3: Creating a Project and Committing Code to the Local Git Repository

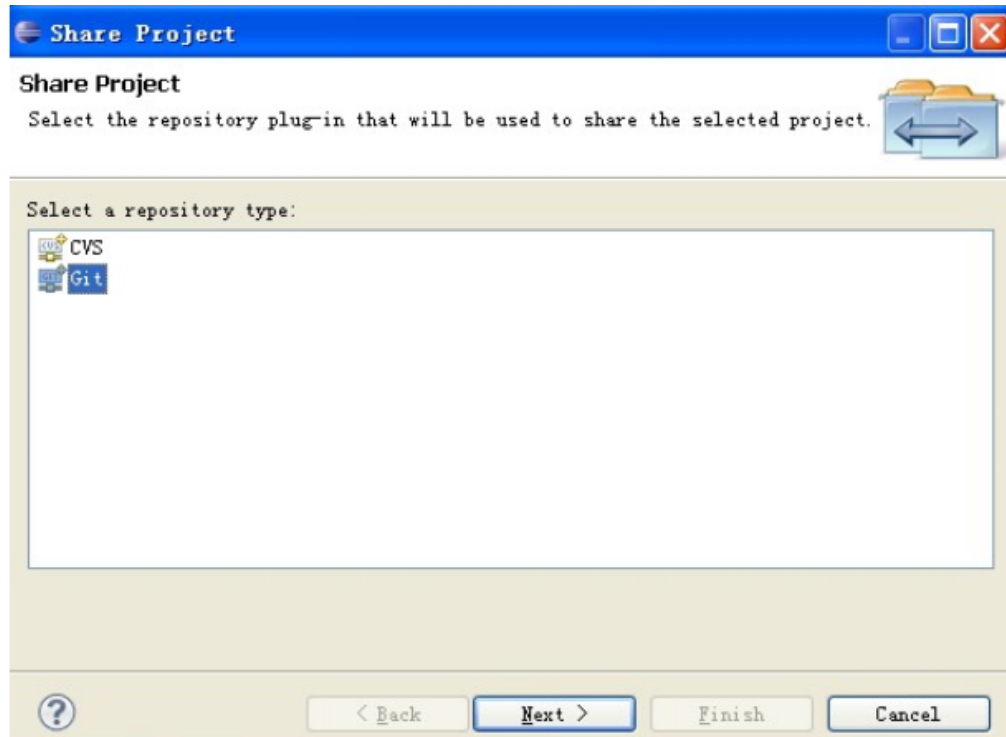
1. Create the **git_demo** project and the **HelloWorld.java** class.



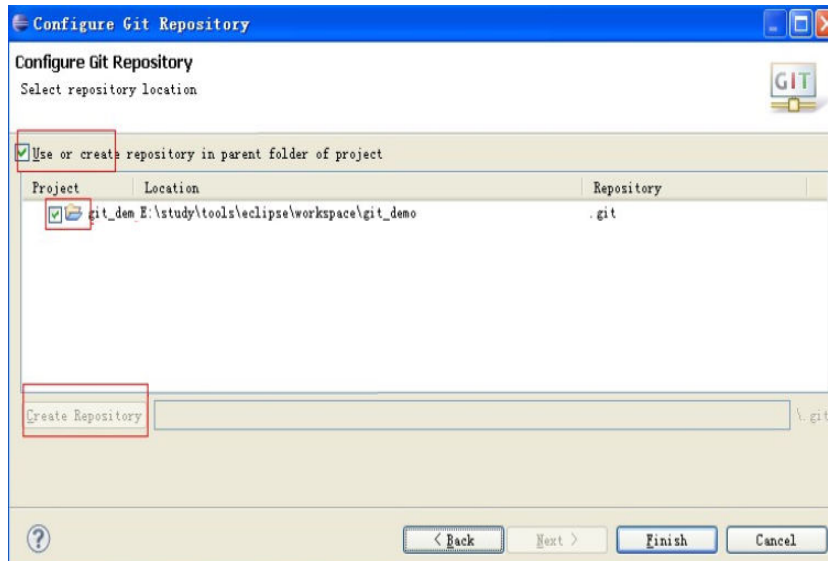
2. Share the **git_demo** project with the local repository.



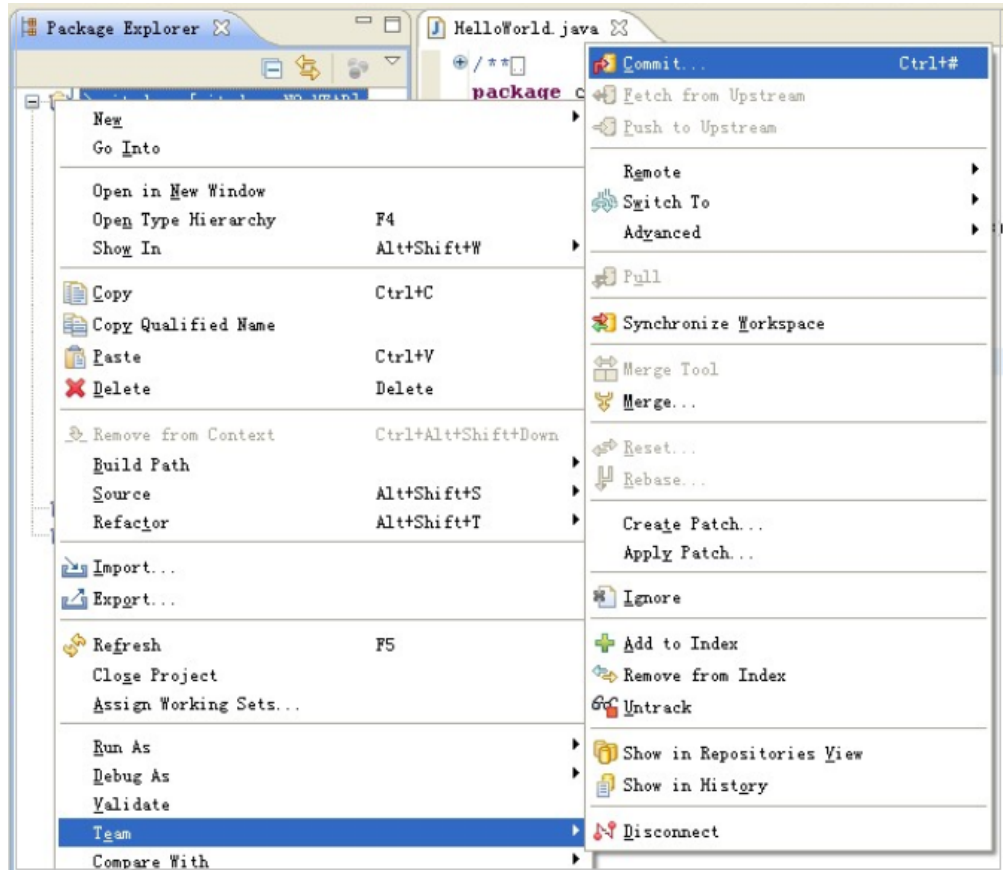
3. In the **Share Project** window displayed, select **Git**.



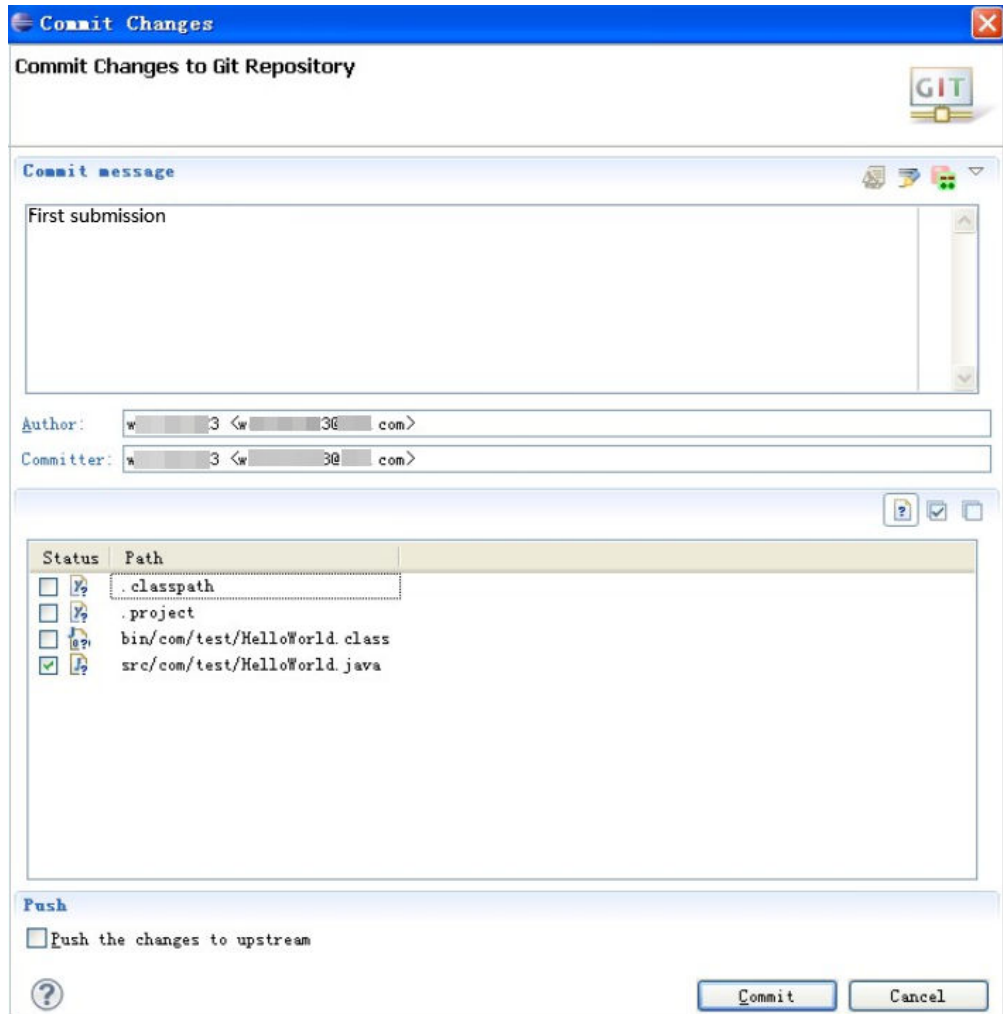
4. Click **Next**. The **Configure Git Repository** dialog box is displayed.



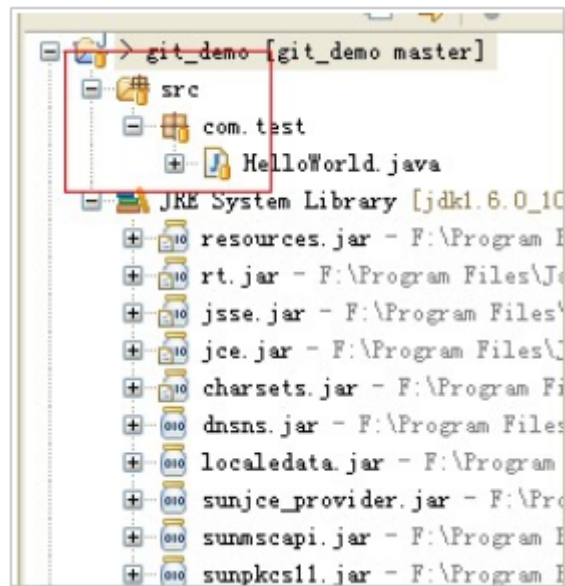
5. Click **Create Repository** to create a Git repository.
The directory is in the **untracked** status, indicated by a question mark (?).
Choose **Team > Commit...** to commit code to the local repository.



6. In the **Commit Changes** dialog box displayed, set the commit message.

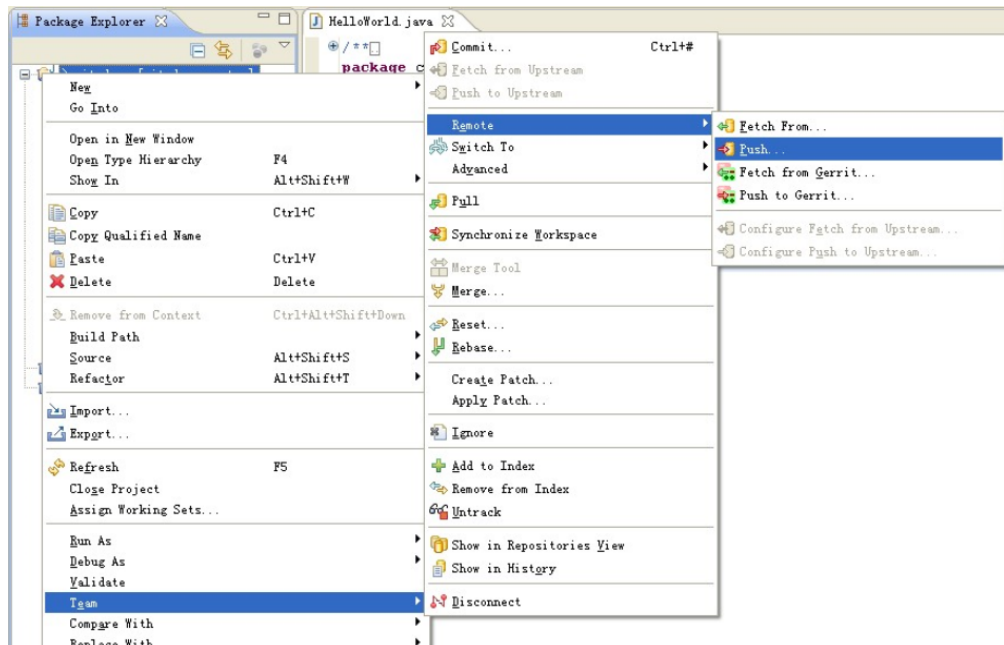


7. Click **Commit** to commit the code to the local repository.

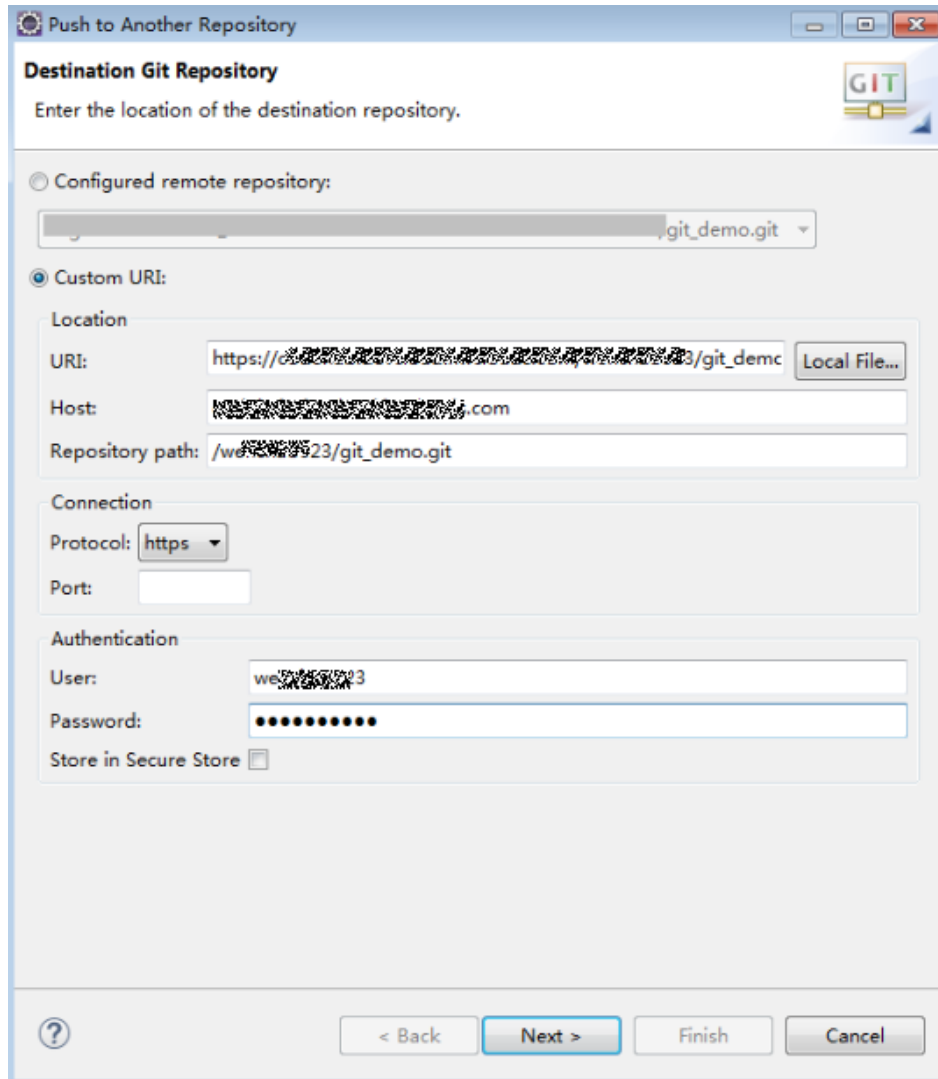


Step 4: Committing Code in the Local Repository to the Remote Git Repository

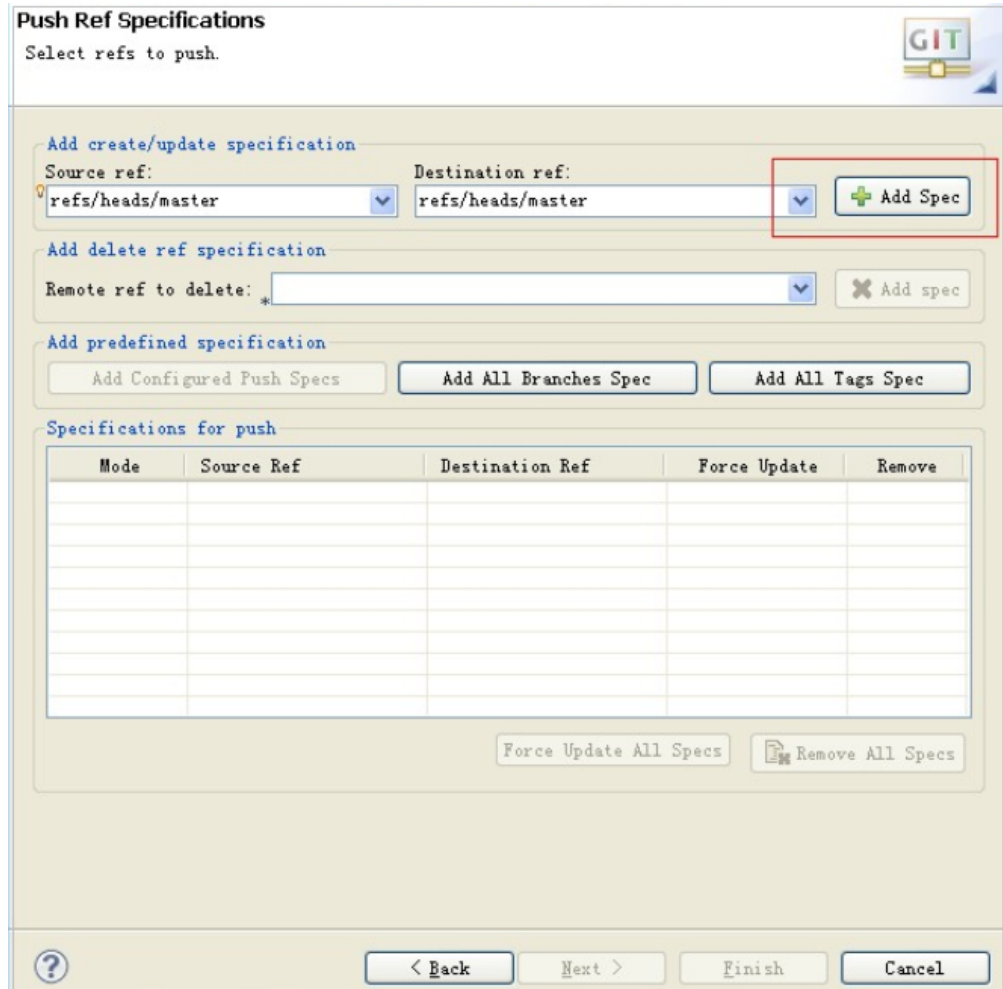
1. Create a repository in CodeArts Repo. For details, see [Overview](#).
Go to the repository details page and copy the repository URL.
2. Choose **Team > Remote > Push...** to push the code to the remote repository.



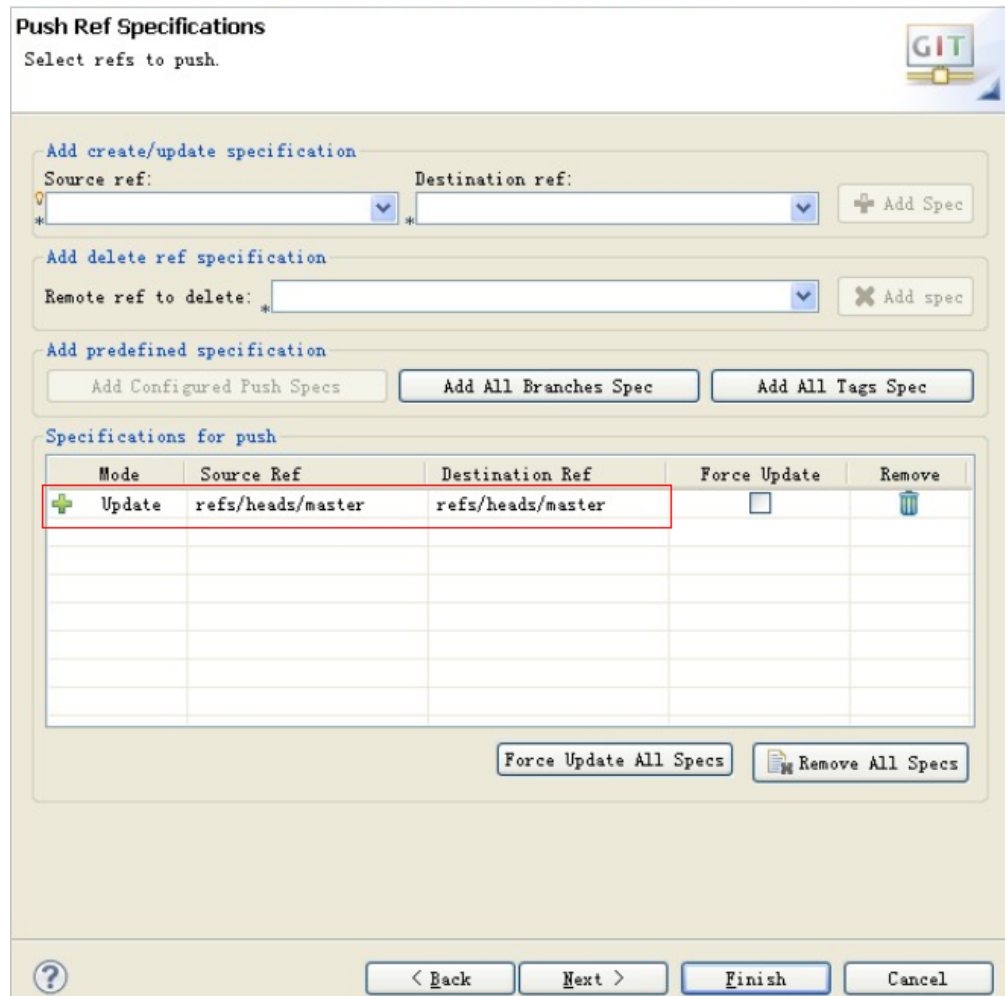
3. In the **Push to Another Repository** dialog box, set the parameters.



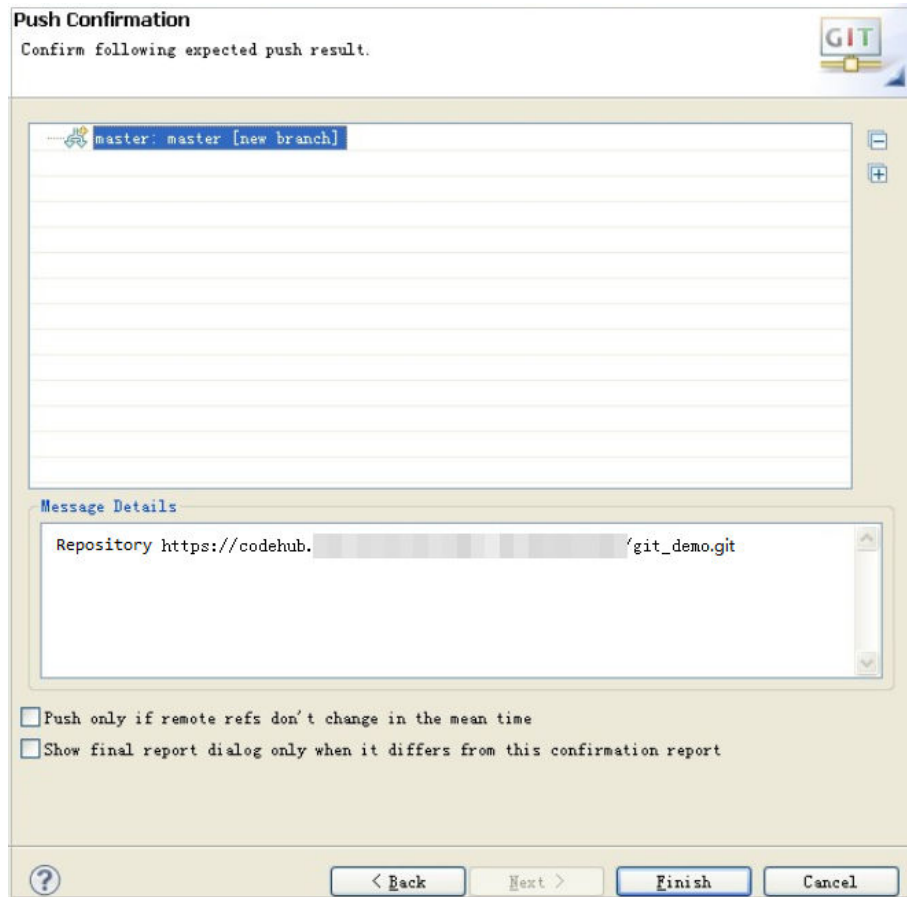
4. Click **Next**. The **Push Ref Specifications** dialog box is displayed.



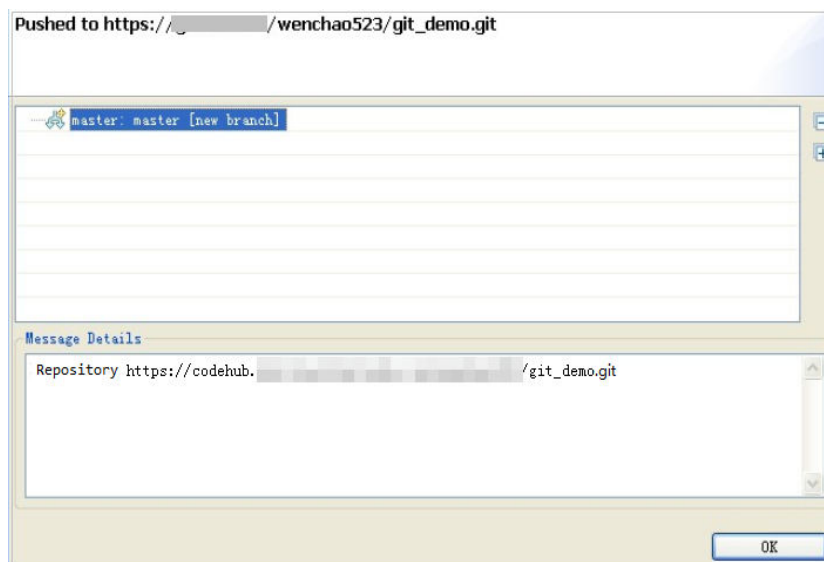
5. Click **Add Spec**.



6. Click **Next**. The **Push Confirmation** dialog box is displayed.



- 7. Click **Finish**.



- 8. Click **OK**.
Log in to the remote repository and check the submitted code.

11 More About Git

- [11.1 Using the Git Client](#)
- [11.2 Setting Password-Free Access via HTTPS](#)
- [11.3 Using the TortoiseGit Client](#)
- [11.4 Use Cases on the Git Client](#)
- [11.5 Common Git Commands](#)
- [11.6 Using Git LFS](#)
- [11.7 Git Workflows](#)

11.1 Using the Git Client

Background

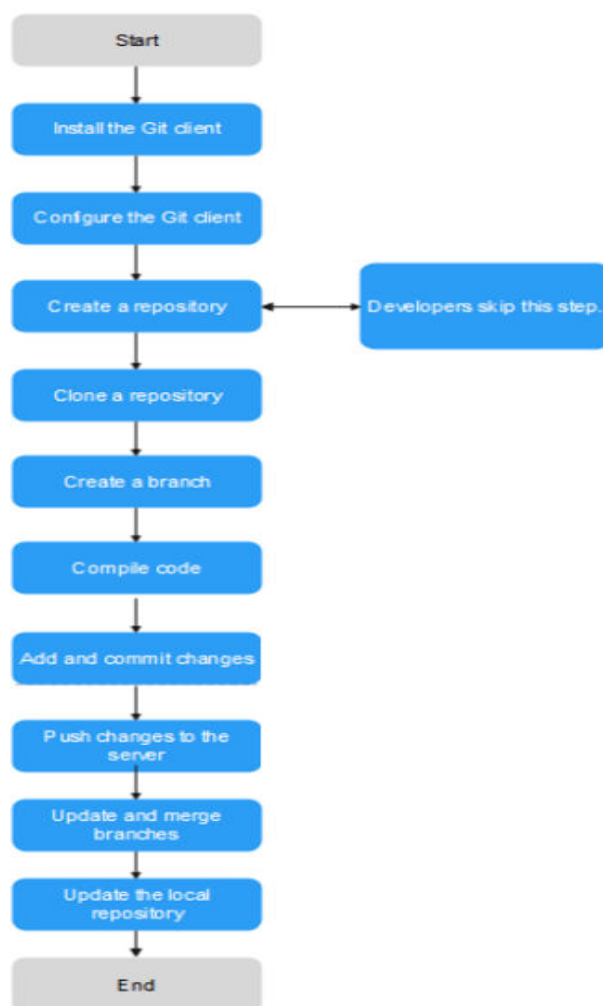
Before using the Git client, you need to understand the workflow and master basic operations, such as installing Git, creating and cloning repositories, adding, committing, and pushing changes, creating, updating, and merging branches, creating tags, and replacing local changes.

Prerequisites

The Git client has been installed.

Usage Process

The following figure shows the basic process of using the Git client.

**Table 11-1** Procedure

Procedure	Description
Install the Git client	Install the Git client for your operating system. <ul style="list-style-type: none">• Git for Windows• Git for macOS X• Git for Linux
Create a repository	Create and open a new folder, and run the following command: <code>git init</code> A Git repository is created.
Clone a repository	Run the following command to create a clone of a local repository: <code>git clone /path/to/repository</code> If the repository is on a remote server, run the following command: <code>git clone username@host:/path/to/repository</code>

Procedure	Description
Local repository structure	<p>There are three components in a local repository: working directory, index, and HEAD.</p> <ul style="list-style-type: none">• Working directory contains the files that you are working on.• Index caches changes you have made.• HEAD points to the latest commit.
Add and commit changes	<p>Run the following command to add the changes to the index:</p> <pre>git add <filename> git add *</pre> <p>Run the following command to commit the changes:</p> <pre>git commit -m "Code submission information"</pre> <p>The changes are committed to the HEAD but not to the remote repository.</p>
Push changes	<p>The changes are in the HEAD of the local repository. Run the following command to push the changes to the remote repository:</p> <pre>git push origin master</pre> <p>You can replace master with any other branch to be pushed.</p> <p>If you have not cloned an existing repository, run the following command to connect the local repository to a remote server before the push:</p> <pre>git remote add origin <server></pre> <p>Then push the changes to the added server.</p>
Create a branch	<p>Branches enable you to develop features separately. When a repository is created, the master branch is the main branch by default. Develop features on other branches and then merge them to the main branch after the development.</p> <ol style="list-style-type: none">1. Create a branch named feature_x and check out the branch. <pre>git checkout -b feature_x</pre>2. Check out the main branch. <pre>git checkout master</pre>3. Push the main branch to the remote repository. (If the branch is not pushed, the branch can be seen only in your local repository.) <pre>git push origin <branch></pre>4. Delete the created branch. <pre>git branch -d feature_x</pre>

Procedure	Description
Update and merge branches	<ol style="list-style-type: none">1. Run the following command to update the local repository to the latest remote commits: <code>git pull</code> The remote changes are fetched and merged to your working directory.2. Run the following command to merge other branches to the current branch (for example, the master branch): <code>git merge <branch></code> <p>NOTE Automatic merges may fail and conflicts occur. In this case, you need to modify these files to manually merge the conflicts.</p> <ol style="list-style-type: none">3. After the modification, run the following command to add your changes. <code>git add <filename></code>4. Before the modification, you can run the following command to compare the source and target branches. <code>git diff <source_branch> <target_branch></code>
Create a tag	<p>You are advised to create tags for releases. For example, run the following command to create a tag named 1.0.0:</p> <pre>git tag 1.0.0 1b2e1d63ff</pre> <p>1b2e1d63ff is the first 10 characters of the commit ID to be tagged. Run the following command to obtain the commit ID:</p> <pre>git log</pre> <p>You can enter the first several characters of the commit ID as long as it can distinguish the commit from others.</p>
Replace local changes	<p>Run the following command to replace the unwanted local changes:</p> <pre>git checkout -- <filename></pre> <p>The files in the working directory are replaced by the latest content in the HEAD. Changes added to the index and new files are not affected.</p> <p>To discard all local changes and commits, fetch the latest commit from the server and reset the local main branch to the commit.</p> <pre>git fetch origin git reset --hard origin/master</pre>

11.2 Setting Password-Free Access via HTTPS

Background

The username and password are required each time you connect to CodeArts Repo using the HTTPS protocol. However, Git can help you implement password-free access with its credential storage. You are advised to install **Git 2.5** or a later version so that the function runs properly. The following describes the configuration methods on different OSs:

- [Setting Password-Free Access on Windows](#)
- [Setting Password-Free Access on macOS](#)
- [Setting Password-Free Access on Linux](#)

Prerequisites

- [The SSH keys and HTTPS password have been set.](#)
- You have to enter the username and password in CodeArts Repo each time you use the HTTPS protocol to perform operations such as git clone, git fetch, git pull, and git push.

Setting Password-Free Access on Windows

The following table describes how to set password-free access on Windows.

Table 11-2 Setting password-free access on Windows

Method	Description
Set the HTTPS password on the local computer	<ol style="list-style-type: none">1. Set the Git authentication mode. Open the Git client and run git config --global credential.helper store.2. Run the Git command to clone or push code for the first time, and enter the username and password as prompted.3. Open the .git-credentials file. If the username and password have been stored locally, the following information is displayed: <code>https://username:password@***.***.***.com</code>

Setting Password-Free Access on macOS

Install the **osxkeychain** tool to implement password-free access.

1. Check whether the tool has been installed.

```
git credential -osxkeychain
# Test for the cred helper
Usage: git credential -osxkeychain < get|store|erase >
```

If the following information is displayed, the tool has not been installed.

```
git: 'credential -osxkeychain' is not a git command. See 'git --help'.
```

2. Obtain the installation package.

```
git credential -osxkeychain
# Test for the cred helper
git: 'credential -osxkeychain' is not a git command. See 'git --help'.
curl -s -o \
https://github-media-downloads.s3.amazonaws.com/osx/git-credential-osxkeychain
# Download the helper
chmod u+x git-credential-osxkeychain
# Fix the permissions on the file so it can be run
```

3. Install **osxkeychain** in the directory where Git is installed.

```
sudo mv git-credential-osxkeychain\
"${dirname $(which git)}/git-credential-osxkeychain"
# Move the helper to the path where git is installed
Password:[enter your password]
```

4. Use **osxkeychain** to set Git to the password-free mode.

```
git config --global credential.helper osxkeychain
#Set git to use the osxkeychain credential helper
```

NOTE

The password needs to be entered the first time you perform Git operations. After that, **osxkeychain** will manage the username and password, and you do not need to enter password subsequently.

Setting Password-Free Access on Linux

Linux provides two password-free access modes:

- **cache:**

- Credentials are cached in memory and cleared after 15 minutes.

```
git config --global credential.helper cache
#Set git to use the credential memory cache
```

- Set the expiration time in **timeout**, in units of seconds.

```
git config --global credential.helper 'cache --timeout=3600'
# Set the cache to timeout after 1 hour (setting is in seconds)
```

- **store:**

Credentials are stored in a plain-text file (`~/git-credentials` by default) in the **home** directory on the disk. The credentials never expire unless you change the password on the Git server. The content of the **git-credentials** file is as follows:

```
https://username:password@*****.com
```

After saving the credentials in the preceding file, run the following command to implement pass-free access:

```
git config --global credential.helper store
```

Troubleshooting

If the message **SSL certificate problem: self signed certificate** is displayed when you download code using HTTPS, run the following command on the client:

```
git config --global http.sslVerify false
```

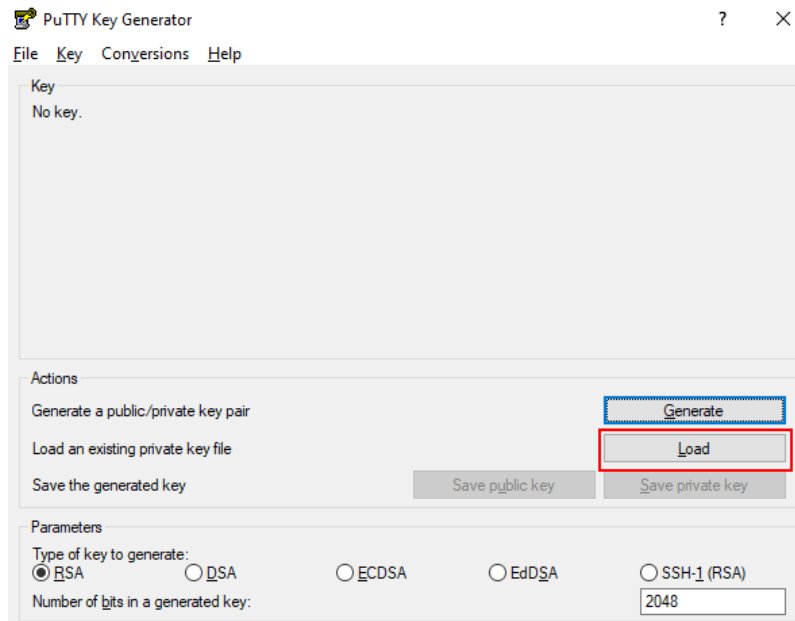
11.3 Using the TortoiseGit Client

Generating a PPK File

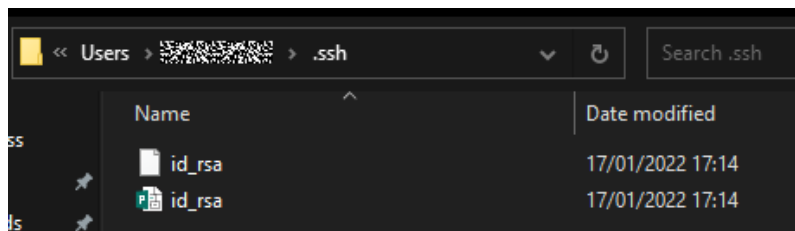
A PPK file is required for downloading and committing code on the TortoiseGit client. Assuming that an SSH key pair has been generated on the Git client, the methods to generate a PPK file are different in the following two scenarios:

- **The Public Key Has Been Added to Ssh-key in CodeArts Repo**

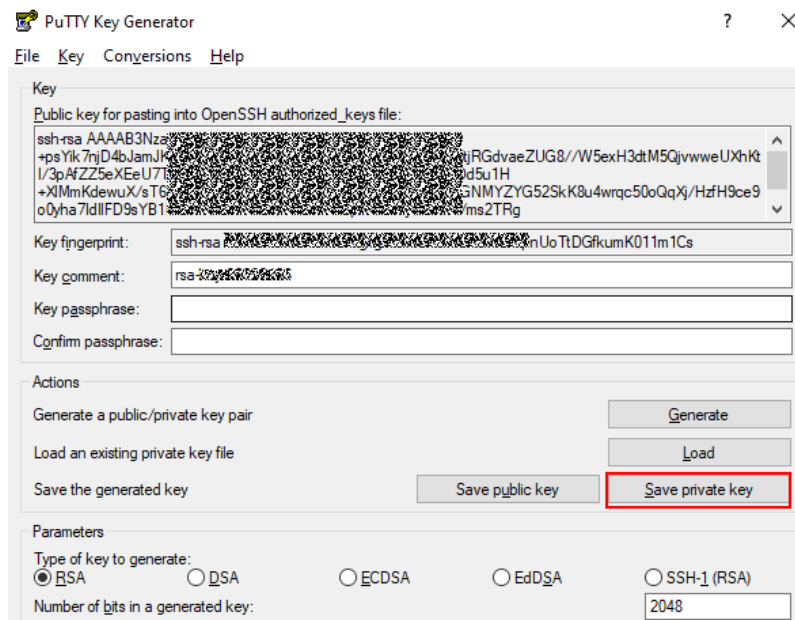
- a. On the **Start** menu, search for and select **PuttyGen**.
- b. Click **Load**.



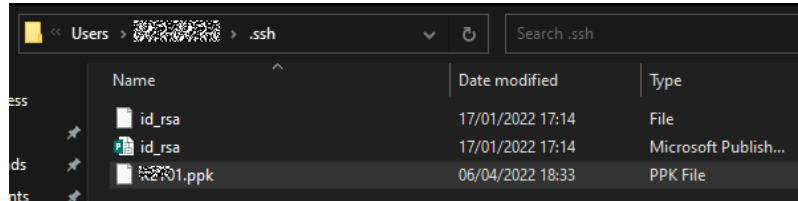
- c. Select the `id_rsa` file in the directory where the SSH key pair is stored and click **Open**.



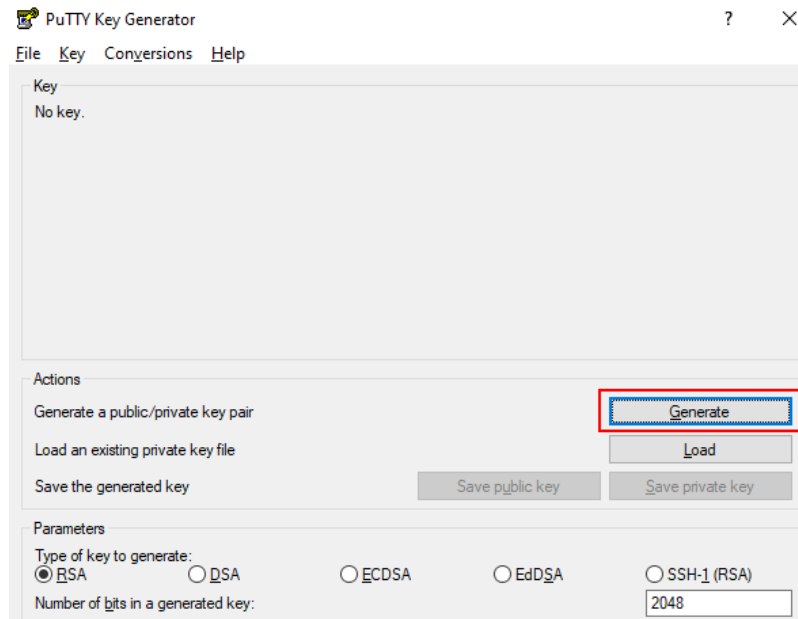
- d. Click **OK** and select **Save private key**.



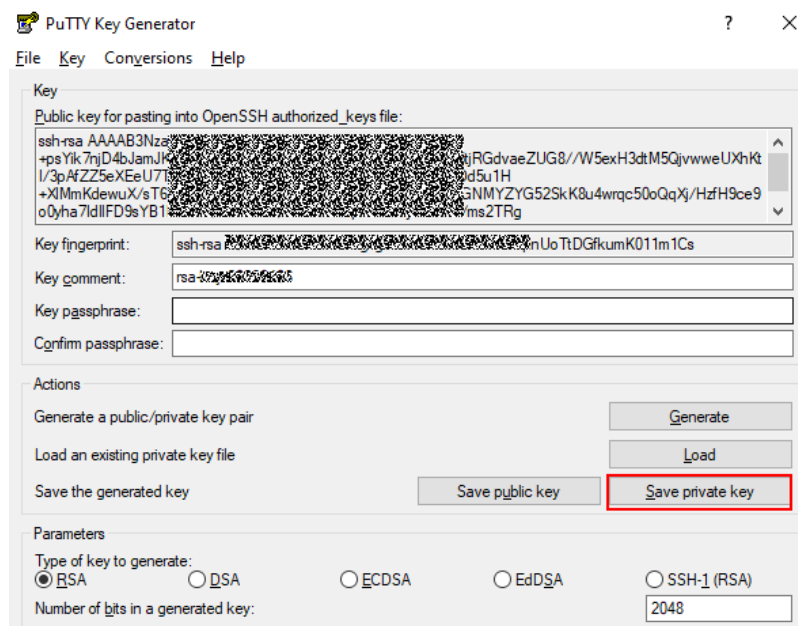
- e. Click **Yes** to generate a PPK file.
- f. Save the file to the directory where the SSH key pair is stored.



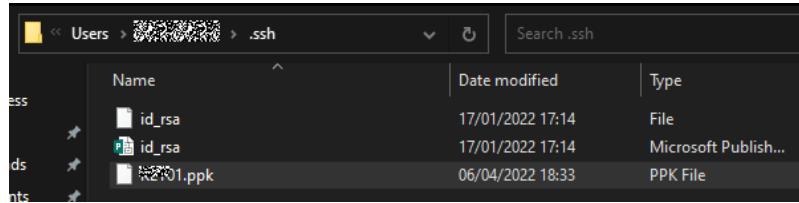
- **The Public Key Has Not Been Added to CodeArts Repo**
 - a. On the **Start** menu, search for and select **PuTTYGen**.
 - b. Click **Generate** to generate a key, as shown in the following figure.



- c. Click **Save private key** to save the generated key as a PPK file.

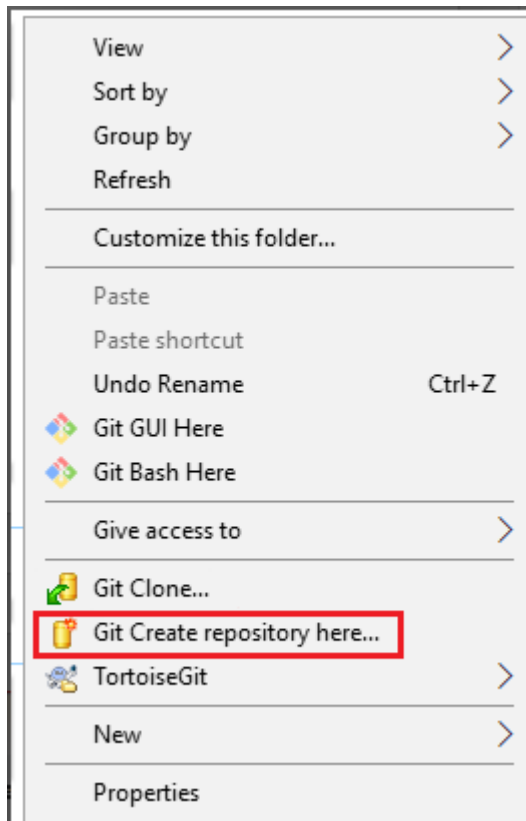


- d. Click **Yes** to generate a PPK file.
- e. Save the file to the directory where the SSH key pair is stored.



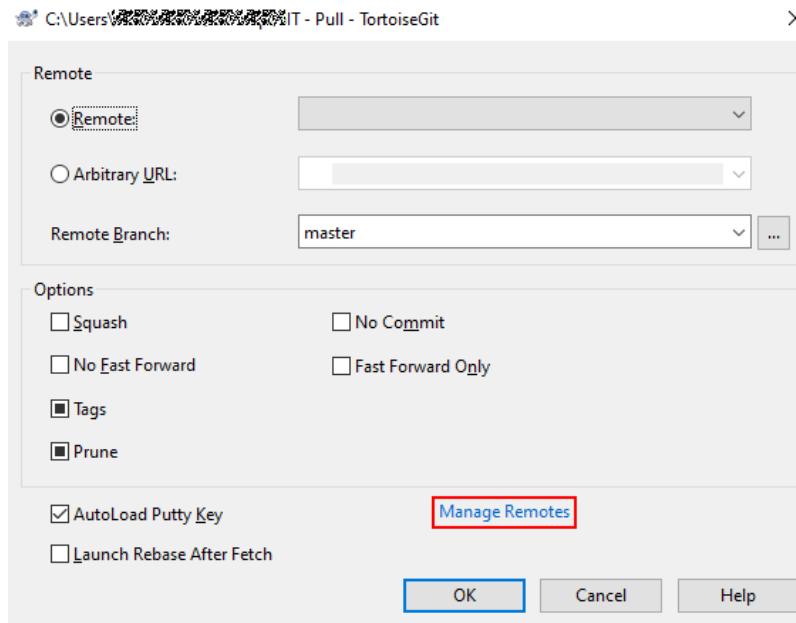
Creating a Git Version Repository

To create a repository for the first time, right-click in an empty directory on the local computer and choose **Git Create repository here...**

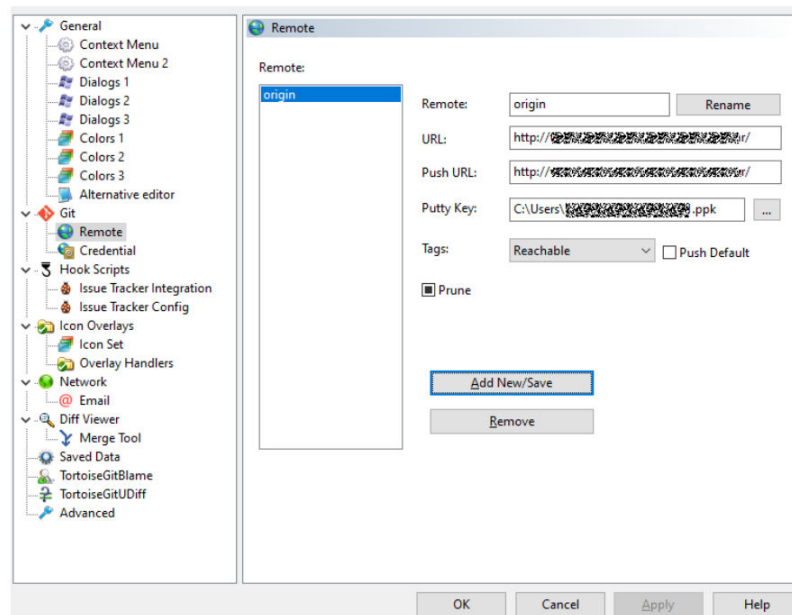


Cloning a Version Repository

1. Open the local Git repository directory (where **the repository is created**) and choose **TortoiseGit > Pull** on the right-click menu.
2. Click **Manage Remotes**.

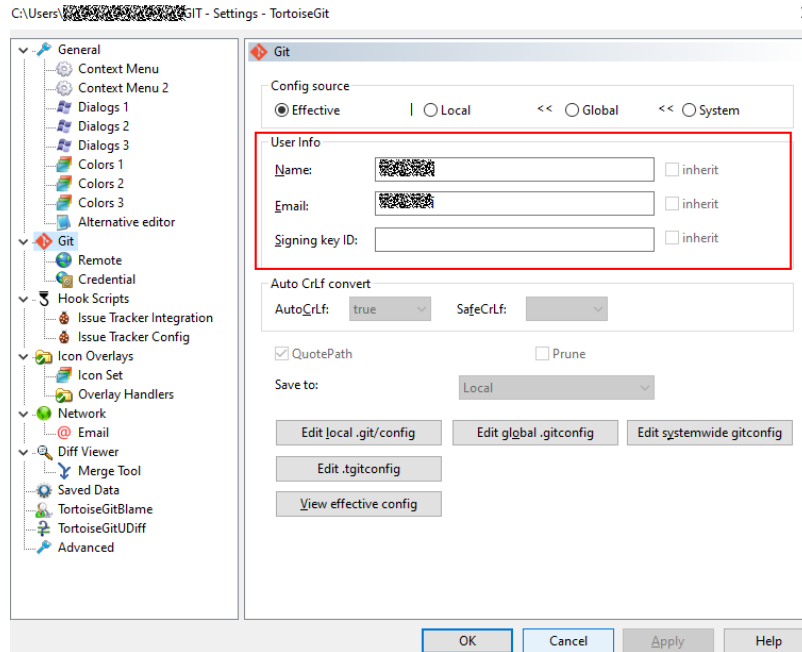


3. Specify the **URL**, select **the PPK file** for the **Putty** field, and click **OK**.



Push Version Repository

1. Configure the username, email address, and signature key ID (PPK file).
2. Right-click in the blank area and choose **TortoiseGit > setting**.
3. Select **Git**, and set **Name** and **Email**.



NOTE

If the push fails, run the following script to locate the fault and send the **git.log** file generated to the technical support:

```
#!/bin/bash
# this script will collect some logs for Coding.net
### how to use ###
# first enter your git repository
# then execute this bash, please make sure you have correct rights
echo "## git version #####" >> git.log
git version >> git.log
echo "## ping #####" >> git.log
ping code*****.com >> git.log
echo "## curl *****.com #####" >> git.log
curl -v https://code*****.com >> git.log 2>&1
echo "## ssh -vT git@*****.com #####" >> git.log
ssh -vT git@*****.com >> git.log 2>&1
echo "## git pull #####" >> git.log
GIT_CURL_VERBOSE=1 GIT_TRACE=1 GIT_TRACE_PACKET=1 git pull >> git.log 2>&1
```

11.4 Use Cases on the Git Client

11.4.1 Uploading and Downloading Code

1. Ensure that the network connection is up and running.

Enter **telnet *****.com 22** on the client.

If **command not found** is displayed, the network cannot access CodeArts Repo.

2. Check if the client is trusted by CodeArts Repo.

If the system prompts you to enter a password when you **pull** or **push** code, check whether the public key has been added to CodeArts Repo.

If the public key has been added, run **\$ ssh -vT git@*****.com** to check whether the trust relationship is established.

If the following information is displayed, the trust relationship is established.

```
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
Welcome to GitLab, 100314597!
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0
debug1: channel 0: free: client-session, nchannels 1
Transferred: sent 3536, received 3488 bytes, in 0.3 seconds
Bytes per second: sent 11491.6, received 11335.6
debug1: Exit status 0

MINGW64 /d/Gitlab
$
```

3. If the fingerprints of both parties are changed after the trust relationship is established, a public key authentication error is reported during commit attempts. In this case, perform the following operations:
 - a. Delete the lines related to `*****.com` from the `~/.ssh/known_hosts` file.
 - b. Enter **push**, **pull**, or **ssh -T git@*****.com**.
 - c. Enter **yes** when asked whether to trust the public key of the server.
4. The code download is successful. If the target branch of the push is protected, the code fails to be pushed.
5. Contact the repository administrator to **unprotect the branch**. The code can be pushed after the protection is canceled.

11.4.2 Committing Letter Case Changes in File Names to the Server

Background

When changes are made to the case of a file name and pushed to the server, the server does not recognize the changes.

For example, a file named **AppTest.java** is renamed as **apptest.java** on the Git client. When the change is pushed to the server, the name of the file in the remote server is still **AppTest.java**.

Procedure

Run the following commands in sequence:

```
git mv --force AppTest.java apptest.java
git add apptest.java
git commit -m "rename"
git push origin XXX (branch name)
```

11.4.3 Setting the Line Ending Conversion

Background

Different operating systems may use different line endings. Therefore, if you open a file created in an operating system different from yours, the file may be displayed incorrectly. This problem may also occur when you use version control systems.

11.5 Common Git Commands

Background

- Git is a free and open-source distributed version control system. It can manage projects of any size in an agile and efficient manner.
- With Git, you can clone a complete Git repository (including code and version information) from a server to a local computer, create branches, modify and commit code, and merge branches.

Commonly Used Commands

The following table describes the functions, formats, parameters, and examples of common Git commands.

Table 11-3 Common Git commands

Command	Function	Format	Parameter	Example
ssh-keygen -t rsa	Generate a key	ssh-keygen -t rsa -C [email]	email: indicates an email address.	Obtain the key file id_rsa.pub from the .ssh folder in drive C. ssh-keygen -t rsa -C "devcloud_key01@XXX.com"
git branch	Create a branch	git branch [new branchname]	new branch name: indicates the name of the new branch.	Create a branch: git branch newbranch

Command	Function	Format	Parameter	Example
git branch -D	Delete a branch	git branch -D [new branchname]	new branch name: indicates the name of the new branch.	Delete a local branch: git branch -D newbranch Delete a branch in the remote repository: git branch -rd origin/newbranch Remove branches that have been deleted in the remote repository: git remote prune origin
git add	Add a file to the index	git add [filename]	file name: indicates the name of the file to be added.	Add a file to the index: git add filename Add all modified and new files to the index: git add .

Command	Function	Format	Parameter	Example
git rm	Delete a local directory or file	git rm [filename]	filename: indicates the name of the file or directory to be deleted.	Delete a file or a directory: git rm filename
git clone	Clone a remote repository	git clone [VersionAddress]	Version Address: indicates the URL of the remote repository.	Clone a jQuery repository git clone https://github.com/jquery/jquery.git A directory is generated on the local computer. The name of the directory is the same as that of the cloned repository.

Command	Function	Format	Parameter	Example
git pull	Pull the branch in the remote repository to the local computer and merge it with a specified local branch	git pull [RemoteHost name] [RemoteBranchname]: [LocalBranch name]	-	Pull the next branch from the remote repository and merge it with the local master branch. git pull origin next:master
git diff	Compares files, branches, directories, or versions	git diff	-	Compare the current branch with the master branch: git diff master
git commit	Commit files	git commit	-	Add a commit message: git commit -m "commit message"
git push	Push files to the remote repository	git push [RemoteHost name] [LocalBranch name] [RemoteBranchname]	-	If the remote branch name is not specified, the local branch is pushed to the remote branch that it tracked (The two branches usually share a name). Such a remote branch will be created if it does not exist. git push origin master The local master branch is pushed to the master branch in the remote repository. If the latter does not exist, it will be created.

Command	Function	Format	Parameter	Example
git merge	Merge branches	git merge [branch]	branch : indicates the name of the source branch	Assuming that the current branch is the develop branch. The latest commit to the master branch is merged to the develop branch. git merge master
git checkout	Check out a branch	git checkout [branchname]	branchname: indicates the name of the branch to be switched to.	Check out the master branch: git checkout master
git log	List the log	git log	-	List all logs: git log --all
git status	Check the status	git status	-	git status
git grep	Search for a character string	git grep	-	Check whether there is any character string containing hello : git grep "hello"

Command	Function	Format	Parameter	Example
git show	Display objects or revisions	git show	-	<ul style="list-style-type: none">• git show v1 The revisions attached with the v1 tag are displayed.• git show HEAD Display the last commit of the current branch.• git show HEAD^ Display the first parent of the last commit of the current branch.• git show HEAD~4 Display the ancestor four generations prior to the last commit of the current branch.
git stash	Commands related to stashes	git stash	-	<ul style="list-style-type: none">• git stash Saves and restores the work progress.• git stash list Lists all stashes.• git stash pop Restore the latest stash and remove it from the stash list.• git stash apply Restore the latest stash but not remove it from the stash list.• git stash clear Clear all stashes.
git ls-files	View files	git ls-files	-	<ul style="list-style-type: none">• git ls-files -d View deleted files• git ls-files -d xargs git checkout Restore deleted files

Command	Function	Format	Parameter	Example
git remote	Perform operations on the remote repository	git remote	-	<ul style="list-style-type: none">• git push origin master:newbranch Create the master branch in the remote repository and push changes to it.• git remote add newbranch Create the master branch in the remote repository and push changes to it.• git remote show List the number of remote repositories• git remote rm newbranch Delete a new branch from the remote repository• git remote update Update branches of all remote repositories

11.6 Using Git LFS

Background

- Git Large File Storage (LFS) is supported on CodeArts Repo. It stores large file such as music, images, and videos outside a Git repository while users can still easily perform operations on these files as if they were within the repository. The Git extension allows more repository space and faster repository cloning, and reduces the impact of large files on the Git performance.
- If the size of a file to be uploaded exceeds 200 MB, use Git LFS.
- Get started with Git LFS:
 - [Installing Git LFS](#)
 - [Configuring File Tracking](#)
 - [Committing Large Files](#)
 - [Cloning a Remote Repository Containing Git LFS Files](#)
 - [More About Git LFS](#)

Installing Git LFS

The following table describes the installation on different operating systems.

Table 11-4 Installing Git LFS

Operating System	Installation Method
Windows	Download and install Git 1.8.5 or a later version. Run the following command in the CLI: <code>git lfs install</code>
Linux	Download the installation package from PackageCloud for your operating system and CPU architecture. Decompress the installation package, run the install.sh script to install the software, and then run the following command to check whether the installation is successful: <code>git lfs version</code>
macOS	Install the Homebrew software package management tool, and run the following commands: <code>\$ /usr/bin/ruby -e "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"</code> <code>\$ brew install git-lfs</code> <code>\$ git lfs install</code>

Configuring File Tracking

This section describes how to configure file tracking.

Table 11-5 Configuring file tracking

Scenarios	Method
Track all .psd files	Run the following command: <code>git lfs track "*.psd"</code>
Track a file	Run the following command: <code>git lfs track "logo.png"</code>
View tracked files	Run git lfs track or view the .gitattributes file. <code>\$ git lfs track</code> Listing tracked patterns *.png (.gitattributes) *.pptx (.gitattributes) <code>\$ cat .gitattributes</code> *.png filter=lfs diff=lfs merge=lfs -text *.pptx filter=lfs diff=lfs merge=lfs -text

Committing Large Files

The **.gitattributes** file should be pushed to the repository along with the large files. After the push, run **git lfs ls-files** to view the list of track files.

```
$ git push origin master
Git LFS: (2 of 2 files) 12.58 MB / 12.58 MB
Counting objects: 2, done.
Delta compression using up to 8 threads.
```

```
Compressing objects: 100% (5/5), done.  
Writing objects: 100% (5/5), 548 bytes | 0 bytes/s, done.  
Total 5 (delta 1), reused 0 (delta 0)  
To <URL>  
<SHA_ID1>..<SHA_ID2> master -> master  
$ git lfs ls-files  
61758d79c4 * <FILE_NAME_1>  
a227019fde * <FILE_NAME_2>
```

Cloning a Remote Repository Containing Git LFS Files

Run **git lfs clone** to clone a remote repository that contains **Git LFS** files to the local computer.

```
$ git lfs clone <URL>  
Cloning into '<dirname>'  
remote: Counting objects: 16,done.  
remote: Compressing objects: 100% (12/12),done.  
remote: Total 16 (delta 3), reused 9 (delta 1)  
Receiving objects: 100% (16/16),done.  
Resolving deltas: 100% (3/3),done.  
Checking connectivity...done.  
Git LFS: (4 of 4 files) 0 B / 100 B
```

More About Git LFS

For details, see the <https://git-lfs.github.com>.

11.7 Git Workflows

11.7.1 Overview

Create a Git workflow or branching policy that works best on your development scenarios for effective version control, project process management, and team collaboration.

There are four common Git workflows. The following sections describe their processes, advantages, disadvantages, and some usage tips.

- Centralized workflow
- Feature branch workflow
- GitFlow (recommended)
- Forking workflow

Development teams can integrate CodeArts Repo and the workflow that suits them best to efficiently manage code and secure code. This enables them to focus more on service development to achieve continuous integration and delivery, and fast iteration.

11.7.2 Centralized Workflow

The centralized workflow is suited to a development team that comprises around 5 members or has just migrated from SVN to Git. There is only one main branch called master by default (trunk in SVN), which is the single entry point of changes. However, this workflow is not recommended for teams who want to enjoy the benefits of Git and team collaboration.

Process

Developers clone the master branch from the central repository to their local computers, make changes to the code, and push changes to the remote master branch.

Advantages

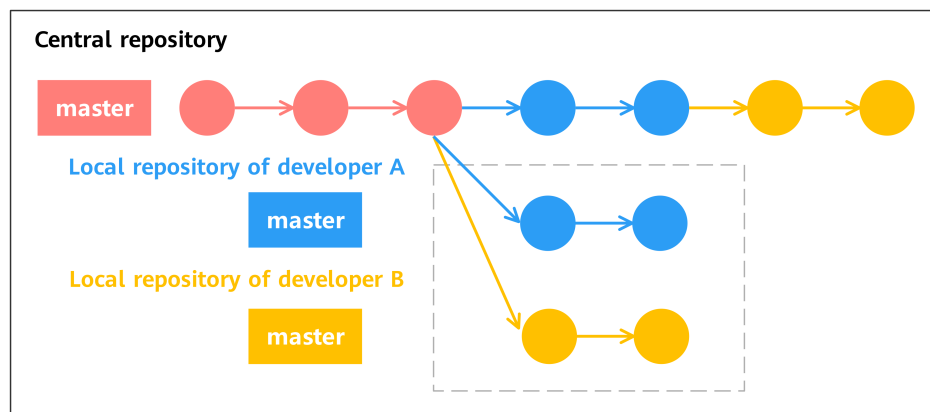
No branch interaction is involved.

Disadvantages

- Merge conflicts are frequent when the size of a development team is more than 10 members. Much time is spent on conflict resolution.
- The master branch is unstable due to frequent pushes to it, making it difficult to conduct integration tests.

Tips: Avoiding Conflicts and Unreadable Commit History

Before developing a new feature, developers must synchronize the local repository to the central one so that they can work on the latest version. After the development is complete, fetch updates from the central repository before rebasing their own commits. In this way, the commits are applied on top of changes that have been made and pushed to the central repository by other developers. The commit history is linear and clear. The following figure shows an example of the workflow.



1. Developers A and B pull code from the central repository at the same time.
2. Developer A completes the work and pushes it to the central repository.
3. When ready to push commits, developer B needs to first run **git pull --rebase** to apply commits on top of the changes made by developer A.
4. Developer B pushes the code to the central repository.

11.7.3 Branch Development Workflow

The core of the feature branch workflow is that every feature should be developed on a separate branch pulled off the master branch. This creates a work silo for every developer, ensures a stable master branch, and encourages team collaboration.

Process

Before developing a new feature, each developer should pull a new branch from the master branch and give it a descriptive name, for example, **video-output** or **issue-#1061**, to clearly state its purpose. By pushing local feature branches to the central repository, developers can share their code with each other without merging code into the master branch.

Advantages

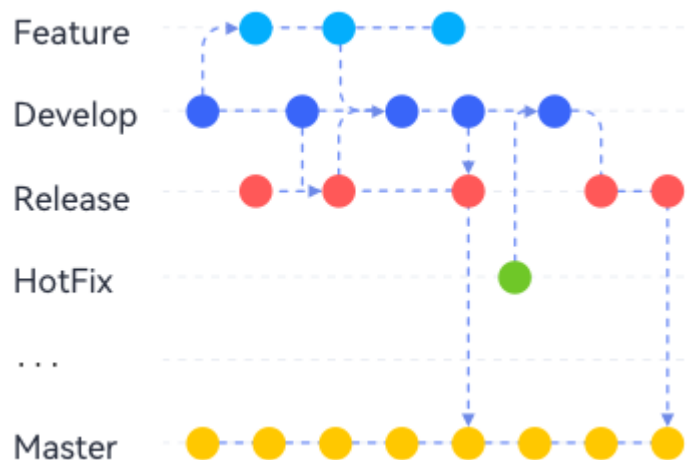
- Developers can create merge requests to have their code reviewed before merge.
- Pushes to the master branch are less frequent.

Disadvantages

Only the master branch is used to incorporate changes. The instability of the branch is further increased in large-scale development projects.

11.7.4 GitFlow

GitFlow is commonly seen in large-scale development projects. Each branch is dedicated to a specific purpose and policies are made to regulate the interaction between branches. The following figure shows the process of GitFlow.



Process

- **Master branch**
The master branch is the production branch where code is ready to deploy. It is the most stable branch because changes cannot be directly pushed to it. Developers can only merge other branches to the master branch. It is often set as a protected branch by default, on which only the project maintainer can operate.
- **Hotfix branch**
It is a temporary branch created off the master branch for fixing urgent bugs in a live production version. After the bug is fixed, the hotfix branch gets

merged into the master branch and tagged with a version number. The bug fix also needs to be merged to the develop branch.

- **Develop branch**

A develop branch is pulled from the master branch and used to merge features. It contains all the code ready to release for integration and system testing.

- **Release branch**

When a new release is coming up, developers create a release branch from the develop branch for release preparations, such as fixing minor bugs and producing documents. Adding new features is not allowed. They should be merged into the develop branch and wait for the next release. When the preparation is complete, the release branch is merged into the master branch and the commit is tagged with a version number. The changes made in the release branch also need to be merged to the develop branch.

- **Feature branch**

Feature branches are pulled from the develop branch for feature development. When the development is complete, they are merged into the develop branch. Feature branches do not interact with the master branch.

Developers add new features in either of the following ways:

- Integrate features after reviewed by a dedicated approver.
 - a. Developers push feature branches to the central repository in CodeArts Repo.
 - b. Developers then create merge requests for merging the feature branches into the develop branch, and assign the requests to the reviewer.

NOTE

CodeArts Repo supports MRs. You can choose source branches and target branches. Only repository administrators (project managers, repository creators, and developers granted with repository management permissions) can accept MRs.

- c. The approver reviews the merge requests. If the requests are approved, the feature branches are merged into the develop branch and deleted. Otherwise, the approver should explain the reasons of rejections.
- Integrate features after self-reviews.
 - a. Developers merge feature branches to the develop branch in the local repository and delete the feature branches.
 - b. The local develop branch is then pushed to the central repository in CodeArts Repo.

Advantages

- With a branch dedicated for release preparation, a development team can develop new features for a future release on the develop branch while improving the version for the upcoming release. Release is visualized, which means team members can have a clear view of the release status in commit graphs.
- Hotfix branches, which can be seen as temporary release branches created off the master branch, enable development teams to fix urgent bugs without

interrupting other works. You do not have to wait until next release but can quickly deploy fixes to the production version.

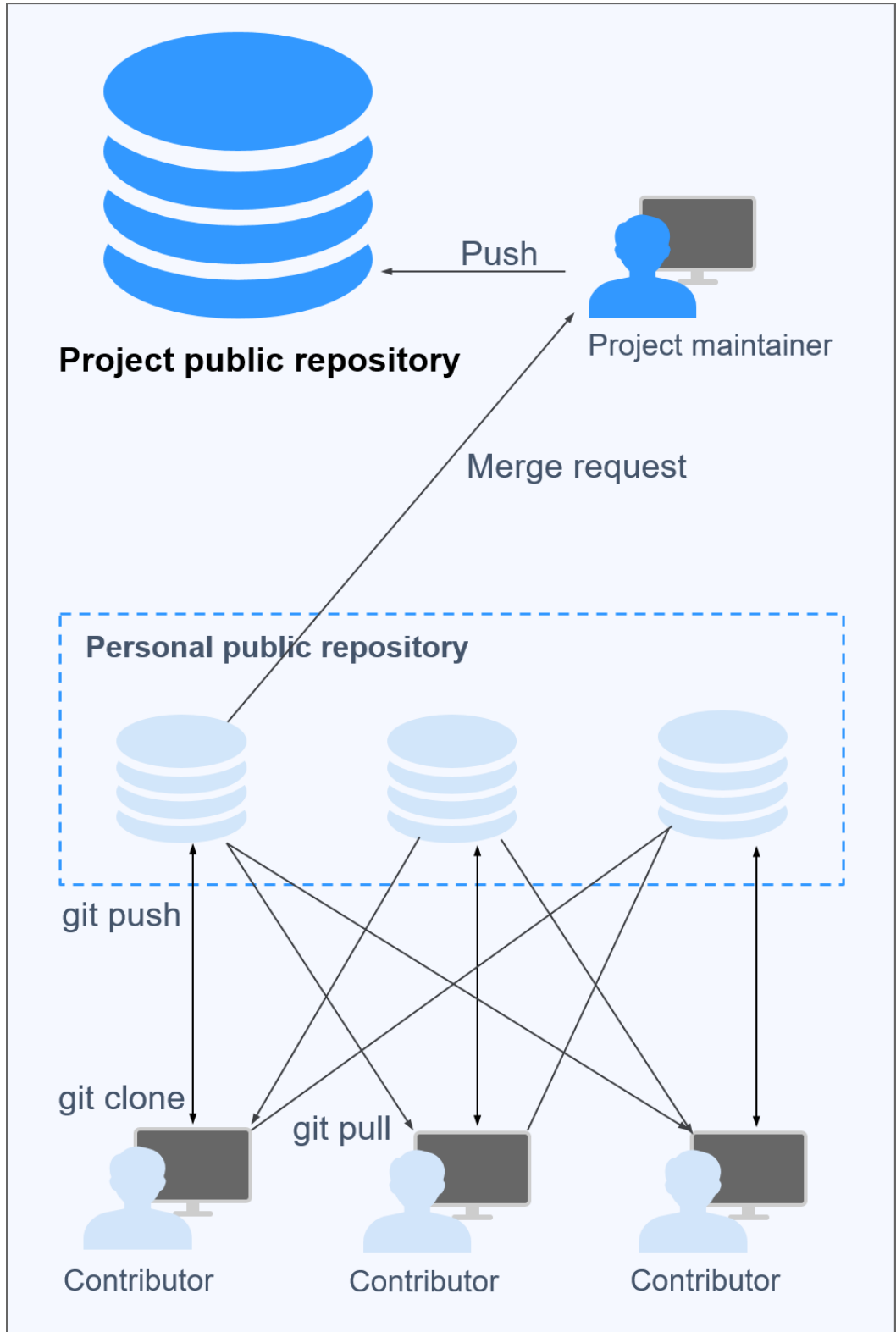
- Effective multi-branch mechanism allows for organized development process especially for large-scale projects.
- This workflow is more in line with the DevOps philosophies.

Disadvantages

- High learning thresholds.
- Impact will be greater if development teams do not comply with their specified workflow policies.

11.7.5 Forking Workflow

The forking workflow is suitable for outsourcing, crowdsourcing, crowdfunding, and open source projects. One of the features that distinguish this workflow is that every contracting developer has a personal public repository, which is forked from the project public repository. Developers can perform operations on the forks without the need of being authorized by the project maintainer. The following figure shows the process of the forking workflow.



Process

1. Developers fork the project public repository to create personal public ones.

2. The personal public repositories are cloned to their local computers for development.
3. After the development is complete, developers push changes to their personal public repositories.
4. Developers file merge requests to the project maintainer for merge to the project public repository.
5. The project maintainer pulls changes to the local computer and reviews the code. If the code is approved, it is pushed to the project public repository.

 **NOTE**

If the code written by a developer is not approved and therefore, not merged to the project public repository, other developers can still pull the code from the personal public repository of the developer for references.

Advantages

- Code collaboration is easier. Developers can share their code by pushing it to their personal public repositories for others to pull, unlike some workflows where developers cannot see others' work until it is merged into the project repository.
- Project maintainers do not have to grant permissions on project public repositories to every contributor.
- Merge requests serve as an important guard for code security.
- The three workflows introduced previously can be incorporated into the forking workflow based on project requirements.

Disadvantages

It takes more steps and time before the code of developers gets merged into the project repository.