

CodeArts Repo

User Guide

Issue 01
Date 2024-11-11



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. 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 Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property 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 Cloud 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 Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: <https://www.huaweicloud.com/intl/en-us/>

Contents

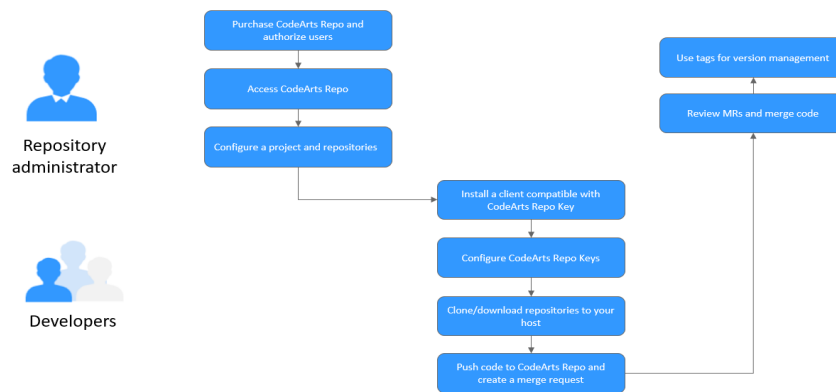
1 Process of CodeArts Repo.....	1
2 Purchasing CodeArts.....	2
3 Accessing CodeArts Repo Homepage.....	4
4 Environment and Personal Settings.....	5
4.1 Installing and Configuring Git.....	5
4.2 Key.....	6
4.3 Configuring an SSH Private Key.....	6
4.4 Configuring HTTPS Password.....	8
4.5 Configuring an Access Token.....	9
4.6 Configuring a GPG Public Key.....	10
4.7 Configuring Git LFS.....	13
5 Migrating Code and Syncing a Repository.....	14
5.1 Repository Migration Overview.....	14
5.2 Migrating a Third-Party Git Repository to CodeArts Repo.....	14
5.2.1 Importing a Git Repository Using a URL.....	14
5.2.2 Importing a GitHub Repository.....	16
5.3 Importing a Local Git Repository to CodeArts Repo.....	17
5.4 Migrating an SVN Code Repository.....	18
5.5 Syncing Repo Settings.....	23
5.6 Verifying the Import Permission.....	23
5.7 Obtaining an Access Token.....	24
5.8 Entering Basic Information for a Repository.....	24
6 Creating a Repository.....	26
6.1 Creating Repos in Different Scenarios.....	26
6.2 Creating a Repository.....	26
6.3 Creating a Repository Using a Template.....	28
6.4 Forking a Repository.....	28
7 Viewing Activities.....	32
8 Viewing Repository Statistics.....	33
9 Configuring Repository Settings.....	35

9.1 Configuring Repository Policies.....	39
9.1.1 Configuring Protected Branch Rules.....	39
9.1.2 Configuring Protected Branch Rules.....	40
9.1.3 Configuring Code Commit Rules.....	40
9.1.4 Review Comments.....	41
9.1.5 MR Evaluation.....	43
10 Hierarchical Repository Management.....	44
10.1 Creating a Repository Group.....	44
10.2 Using Repository Groups.....	45
10.2.1 Viewing the Repository Group List.....	45
10.2.2 Viewing Repository Group Details.....	46
10.2.3 Viewing the Repository Group Homepage.....	47
10.2.4 Managing Members of a Repository Group	47
10.3 Configuring Repository Groups.....	49
10.3.1 Repository Group Information.....	49
10.3.2 Repository Settings.....	49
10.3.3 Risky Operations.....	51
10.3.4 Permission Management.....	51
11 Configuring a Repository.....	57
11.1 Configuring Repository Settings.....	57
11.2 Viewing the Repository List.....	57
11.3 Viewing Repository Details.....	58
11.4 Viewing Repository Homepage.....	60
11.5 Backing Up a Repository.....	63
12 Managing Repo Member Permissions.....	64
12.1 IAM Users, Project Members, and Repository Members.....	64
12.2 Configuring Project-Level Permissions.....	65
12.3 Configuring Repo-Level Permissions.....	68
12.4 Syncing Project Members to CodeArts Repo.....	71
13 Cloning or Downloading Code Repo to a Local PC.....	72
13.1 Differences Between Cloning and Downloading a Repository.....	72
13.2 Using the SSH Key to Clone a Repo to a Local PC.....	73
13.3 Using HTTPS to Clone Code from CodeArts Repo to a Local Computer.....	74
13.4 Using a Browser to Download Code Package to a Local PC.....	75
14 Uploading Code Files to CodeArts Repo.....	77
14.1 Editing and Creating a Merge Request.....	77
14.2 Creating a Branch and Developing Code in Git Bash.....	77
14.3 Committing Code in Eclipse and Creating a Merge Request.....	80
14.4 Using git-crypt to Transmit Sensitive Data on the Git Client.....	91
14.5 Viewing Commit History.....	100

15 Developing a Workflow.....	102
15.1 Workflow Overview.....	102
15.2 Centralized Workflow.....	102
15.3 Feature Branch Workflow.....	103
16 Creating and Configuring a CodeArts Project.....	105
16.1 Configuring Project-Level Commit Rules.....	105
16.2 Configuring Project-Level Repo Settings.....	115
16.3 E2E Settings.....	124
16.4 Webhook Settings.....	129
17 Committing Code to CodeArts Repo and Creating a Merge Request.....	132
17.1 Setting Repo-Level Merge Request Rules.....	132
17.2 Configuring Merge Request Notifications.....	137
17.3 Resolving Review Comments and Merging Code.....	140
18 Managing Merge Requests.....	143
18.1 Detailed Description of Review Comments Gate.....	143
18.2 Resolving Code Conflicts in an MR.....	144
18.3 Creating a Squash Merge.....	152
19 Managing Code Files.....	154
19.1 Managing Files.....	154
19.2 Managing Commits.....	159
19.3 Managing Branches.....	159
19.4 Managing Tags.....	170
19.5 Managing Comparison.....	176
20 Security Management.....	177

1 Process of CodeArts Repo

The following figure shows how to use CodeArts Repo.



2 Purchasing CodeArts

Prerequisites

To perform this operation, you must have any of the following permissions:

- **Tenant Administrator**
- **DevCloud Console FullAccess** and **BSS Administrator**
- **DevCloud Console FullAccess** and **BSS Finance**
- **DevCloud Console FullAccess** and **BSS Operator**
- Custom policies that contain all permissions in **DevCloud Console FullAccess** as well as the **bss:order:view**, **bss:order:pay**, and **bss:order:update** permissions.

Purchasing a CodeArts Package

CodeArts uses yearly/monthly billing, and provides the free, basic, professional, and enterprise edition packages to meet the requirements of different user scales. For details about these packages, see [Package Overview](#).

Step 1 Go to the [Buy CodeArts](#) page.

Step 2 Set the region, edition, number of users, required duration, and auto-renewal, agree to the agreement, and click **Next**.

NOTE

- You are advised to select the nearest region based on your physical region where your services are located to reduce network latency. The purchased package takes effect only in the corresponding region and cannot be used across regions.
- The number of users and required duration in the Free edition are fixed and cannot be changed.

Step 3 Confirm the order content. If you need to modify it, click **Previous**. If the content is correct, click **Pay**.

Step 4 Follow the prompts to complete the payment.

Step 5 Check the package purchase record back on the CodeArts console.

If the purchase fails, rectify the fault by referring to [Billing FAQs](#).

----End

Purchasing Resource Extension

CodeArts provides parallel job extension for multiple services. For details, see [Resource Extension](#).

Step 1 Go to the [Buy CodeArts Resource Extension](#) page.

Step 2 Set the region, product, type, required duration, and auto-renewal, agree to the agreement, and click **Next**.

NOTE

- The configuration items for the selected product and type are displayed. Select the configuration items you need.
- Select a region where you have purchased CodeArts Basic or higher edition, or you cannot purchase resource extensions.

Step 3 Confirm the order content. If you need to modify it, click **Previous**. If the content is correct, click **Pay**.

Step 4 Follow the prompts to complete the payment.

Step 5 Check the resource extension purchase record back on the CodeArts console.

If the purchase fails, rectify the fault by referring to [Billing FAQs](#).

----End

Purchasing a Value-added Feature

CodeArts provides value-added features such as CodeCheck enhanced package. For details, see [Value-added Features](#).

Step 1 Go to the [CodeArts value-added feature purchase](#) page.

Step 2 Set the region, product, quantity, required duration, and auto-renewal, agree to the agreement, and click **Next**.

NOTE

- To purchase the CodeCheck enhanced package, select a region where you have purchased CodeArts Pro or Enterprise edition.

Step 3 Confirm the order content. If you need to modify it, click **Previous**. If the content is correct, click **Pay**.

Step 4 Follow the prompts to complete the payment.


Step 5 Check the value-added feature purchase record back on the CodeArts console.

If the purchase fails, rectify the fault by referring to [Billing FAQs](#).

----End

3 Accessing CodeArts Repo Homepage

Step 1 [Log in to the Huawei Cloud console](#) with your Huawei Cloud account.

Step 2 Click  in the upper left corner of the page and choose **Developer Services > CodeArts Repo** from the service list.

Step 3 You can access CodeArts Repo in either of the following ways:

- **From the homepage**

Click **Try Now**. This page displays the build task list of the current user.

- **From the project page**

- a. Click **Try Now**.
- b. On the navigation bar, click **Homepage**.
- c. Click the name of the project to be viewed.
- d. Choose **Code > Repo**. The repository list page of the project is displayed.

----End

4 Environment and Personal Settings

4.1 Installing and Configuring Git

For details about the clients supported by Repo and the installation guide link, see [Table 4-1](#).

Table 4-1 Compatible Git clients

Client Name	OS	Official Installation Guide Link
Git client	Windows	Windows Git Client Installation Guide
	Linux	Linux Git Client Installation Guide
	Mac	Mac Git Client Installation Guide
TortoiseGit	Windows	Windows TortoiseGit Client Installation Guide

After installing the Windows Git client, configure the user name and email address. Enter the following command in Git Bash:

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

 NOTE

- Currently, CodeArts Repo cannot be managed using GitHub desktop.
- The username can contain letters, digits, and common characters, but cannot contain characters no more than 32 in the ASCII code table. It cannot start or end with `.`, `:`, `<`, `>`, `"`, `\\` and `\`. Any of the preceding characters appears at the beginning or end will be ignored. To facilitate management, you can set this parameter to the username of CodeArts Repo.
- CodeArts Repo supports TLS1.2 and TLS1.3. If Git is of the latest version, run the following command to specify the TLS protocol version: In the preceding command, **test.com** is the domain name for Git upload/download in CodeArts Repo, and **tls1_2** indicates that the TLS protocol version is TLS1.2. For details about different Git client solutions, see [TLS protocol versions compatible with CodeArts Repo](#).

```
openssl s_client -connect test.com:443 -tls1_2
```

4.2 Key

The Repo code repository supports SSH and HTTPS access protocols. You can use either of the following methods for configurations.

- The SSH key is a secure connection method between the local computer and your Repo. Different users usually use different computers. Therefore, before connecting to Repo via SSH, you need to generate your own SSH key on your computer and add the public key to Repo. Once the SSH key is configured on the local computer and the public key is added to Repo, all repos under this account can use the key to connect to the computer.
- An HTTPS password is a user credential used for pulling and pushing code using the HTTPS protocol.

NOTICE

- In CodeArts Repo, the size of a single file to be pushed using HTTPS cannot exceed 200 MB. To transfer a file larger than 200 MB, use the SSH mode.
 - Only accounts that can be bound to email addresses can use the HTTPS protocol.
-
- GNU Privacy Guard (GPG) is a method used for digital signature and authentication. When you push the local code to CodeArts Repo, the GPG public keys ensure trusted sources and code integrity by signing and verifying Git code commits and tags in Git.

4.3 Configuring an SSH Private Key

Step 1 Run Git Bash to check whether an SSH key has been generated locally. Run the following command in Git Bash:

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

- If **No such file or directory** is displayed, no SSH key has been generated on your computer. Go to [Step 2](#).
- If a character string starting with **ssh-rsa** is returned, an SSH key has been generated on your computer. If you want to use the generated key, go to [Step 3](#). If you want to generate a new key, go to [Step 2](#).

Step 2 Generate an SSH key. Run the following command to generate a key in Git Bash:

```
ssh-keygen -t rsa -b 4096 -C your_email@example.com
```

In the preceding command, **-t rsa** indicates that an RSA key is generated, **-b 4096** indicates the key length (which is more secure), and **-C your_email@example.com** indicates that comments are added to the generated public key file to help identify the purpose of the key pair.

If you select the ED25519 algorithm, run the following command to generate a key in Git Bash:

```
ssh-keygen -t ed25519 -b 521 -C your_email@example.com
```

In the preceding command, **-t ed25519** indicates that an ED25519 key is generated, **-b 521** indicates the key length (which is more secure), and **-C your_email@example.com** indicates that comments are added to the generated public key file to help identify the purpose of the key pair.

Enter the command for generating the key and press Enter. The key is stored in **~/.ssh/id_rsa** by default, and the corresponding public key file is **~/.ssh/id_rsa.pub**.

Step 3 Copy the SSH public key to the clipboard. Run the corresponding command based on your operating system to copy the SSH public key to your clipboard.

- Windows:

```
clip < ~/.ssh/id_rsa.pub
```
- Mac:

```
pbcopy < ~/.ssh/id_rsa.pub
```
- Linux (xclip required):

```
xclip -sel clip < ~/.ssh/id_rsa.pub
```

Step 4 Log in to Repo and go to the code repository list page. Click the alias in the upper right corner and choose **This Account Settings > Repo > SSH Keys**. The **SSH Keys** page is displayed.

You can also click **Set SSH Keys** in the upper right corner of the code repository list page. The **SSH Keys** page is displayed.

Step 5 In **Key Name**, enter a name for your new key. Paste the SSH public key copied in [Step 3](#) to **Key** and click **OK**. The message "The key has been set successfully. Click Return immediately, automatically jump after 3s without operation" is displayed, indicating that the key is set successfully.

----End

NOTE

- After an SSH key is configured on a computer and the public key is added to CodeArts Repo, all repos under the account can use the SSH key to connect to the computer. Different users usually use different computers. Therefore, before connecting to CodeArts Repo in SSH mode, you need to configure an SSH key on your computer.
- When you configure an SSH key, the following message is displayed: **SSH Key Already Exists**, indicating that the key has been added to the account or another account. Solution: Generate a new SSH key locally by referring to the preceding steps and configure the generated key in CodeArts Repo.

4.4 Configuring HTTPS Password

When you push code to or pull code from CodeArts Repo, the repository verifies your identity and permissions. HTTPS password is an identity authentication mode for remote access to CodeArts Repo. You only need to set this once.

- **HTTPS username**

It consists of your tenant name and IAM username: *Tenant name/IAM username*.

- **HTTPS password**

8 to 32 characters with at least three types of the following characters: digits, uppercase letters, lowercase letters, and special characters, and cannot be the same as the username of the HTTPS password or the reverse-order username.

Setting the HTTPS Password for the First Time

By default, the HTTPS password is your login password which can be synced in real time. You can also perform the following steps to set the initial password.

Step 1 Go to the repo list page of CodeArts Repo, click the nickname in the upper right corner, and choose **This Account Settings > Repo > HTTPS Password**.

You can also go to the repo list page and click **Set HTTPS Password** in the upper right corner.

Step 2 Click **Reset** to go to the password resetting page if you are setting the password for the first time. Click **Set new password**, fill in the **New Password** and **Confirm Password**, and click **OK**. A dialog box is displayed, indicating that the password is set successfully.

Step 3 Regenerate the repository credential locally and check the IP address whitelist when the new password is created. Otherwise, you cannot use CodeArts Repo repositories.

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

Step 4 Check whether the HTTPS password takes effect by referring to [Verifying Whether Your HTTPS Password Takes Effect](#).

----End

NOTE

- If your account is upgraded to a HUAWEI ID, the tenant-level function of **Use Huawei Cloud login password** is no longer supported (the function is still valid for IAM users).
- **Federated users** cannot be bound to email addresses and do not support the HTTPS protocol.
- If the message "SSL certificate problem" is displayed when you perform step 3, run the **git config --global http.sslVerify false** command on the Git client to disable the SSL verification function of Git.

Changing the HTTPS Password

- Step 1** Go to the repo list page of CodeArts Repo, click the alias in the upper right corner, and choose **This Account Settings > Repo > HTTPS Password**.

You can also go to the repo list page and click **Set HTTPS Password** in the upper right corner.

- Step 2** Click **Set new password**. The page for resetting the password is displayed. Click **Change**, fill in **Old Password**, **New Password**, and **Confirm Password**, and click **OK**. A dialog box is displayed, indicating that the password is set successfully.

You can also click **Reset**. You need to bind an email address if you are setting the password for the first time. Set **Verification Code**, **New Password**, and **Confirm Password**, and click **Save**. A dialog box is displayed, indicating that the password is set successfully.

- Step 3** Check whether the HTTPS password takes effect by referring to [Verifying Whether Your HTTPS Password Takes Effect](#).

----End

NOTE

If the message "No backend available: service IAM" is displayed when you bind an email address, contact your administrator to bind an email address for you, return to the HTTPS password resetting page, and refresh it.

Verifying Whether Your HTTPS Password Takes Effect

After setting the HTTPS password, you can run the **git clone https://username:password@example.com/repo_path.git** command in Git Bash to clone the code repository to which you have access. **Username** indicates the HTTPS username, **Password** indicates the HTTPS password, and **example.com/repo_path.git** indicates the HTTPS address of the code repository to be cloned. If the code is successfully cloned with the command, the HTTPS password has been successfully set.

4.5 Configuring an Access Token

Log in to the CodeArts Repo service repository list page, click the nickname in the upper right corner, choose **This Account Settings > Repo > Access Token**, click **New Token**, and set parameters based on the following table.

Table 4-2 Description

Parameter	Description
Token Name	Mandatory Custom name with a maximum of 200 characters.
Description	Optional. If the description is empty, -- is displayed in the list. Max. 200 characters.

Parameter	Description
Permissions	This parameter is selected by default and cannot be modified. Read/Write repo: Read from and write into repositories using HTTPS.
Expired	Mandatory Time when a token expires. NOTE 30 days after current date by default, including current date. For example, if a token is created on July 3, the default expiration date is 23:59:59 on August 2. The expiration date can be set to a maximum of one year and cannot be empty.

After the parameters are set, the token is successfully generated. Copy and use it in the application or script.

NOTICE

- For security reasons, the token will not be displayed after this dialog box is closed. Keep the token safe. If you lose or forget it, generate a new one.
 - A maximum of 20 tokens can be created for CodeArts Repo.
-

4.6 Configuring a GPG Public Key

Perform the following steps to generate and configure a GPG public key in CodeArts Repo:

- [Download the GPG key generation tool from gpg4win official site.](#)
- [Generate a GPG Key Pair.](#)
- [Check whether the GPG key is generated successfully.](#)
- [Copy the GPG key pair to the clipboard.](#)
- [Go to the GPG key configuration page.](#)
- [Set parameters for creating a GPG public key.](#)
- [Check whether the GPG public key is successfully configured.](#)

Step 1 Download the GPG key generation tool from [gpg4win official site](#).

Step 2 Run the `gpg --full-generate-key` command on the local Git client, select the encryption algorithm, key length, expiration time, and correctness in sequence as prompted, and enter a username, email address, and comment, as shown in [Figure 4-1](#).

Figure 4-3 Exporting the GPG public key

```

cwx3iPpVtGmE7GmHPCNtX71M001 MINGW64 /
$ gpg --armor --export
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBGXF7dABCAD...s706KFYtzJ9G+whZIjC1BPa
IKHF5CVkbbKCjQ2...qu6EON1DHyUeICt0C4VICpb
kcvbf0agfqIIoCh...h9p1DC2ef0mJFn71QAkUPVF
Z/5iLKZY7AALD/c...2xoGb12NmyTV65GzLAW2QGZ
i22xQC2NxzvsRL...97arzCDiLvegChH6ZLh+gsD
OyjCiA/F/cg7KQl...AG0KXNoYw5zaGFuICGxMjMp
IDXjYw9zaGFuMTE...BMBCAA4FiEEJmQqFFI+TOJY
tXbrvUseIUfjrNM...AsCBBYCAwECHgECF4AACgkQ
vUseIUfjrNPWxw...LavFA/hS8xcWuxFsRyoz7N+gr
mNdmEVnGtNkuR7...v1w7jSFJImMk+UKW0Csaz0e
w7cky7ubwkPRWg...Vsb0CWnyXmYS68SIdiLERPa
sGiOnqdj1w340/X...J37/z2IU2f9mCx17m3u01F
U0YLySL4zvRwzL...bYd2XbQ/IqxrRK7BqmSrMDx
1B4/zGhbR2kwiZ...QR13+3QAQgA8I/oFkBM/Y6Y
IqQpItXJ7iYde8...o1LO/wNUgMzzJHJ/mgAez+Z
iI1jd4jvAUHTCe...EEROkXoSIO7kqXCFL2udhey
Y+Jd/yNyQ59hB7...hkpITBw1vTehCQyhSymCC4Y
IraE1WKsgXON+q...UV92smkUmi15SgU59x2NVg8
FgE9qV4LNGyO+...NP9FhFc03mLgww6TLX5sm6z
dIrKeNKs4wARAQ...OJYtXbrvUseIUfjrNMFAMxf
7dACGwwACgkQvU...cSYjw8DwYjygiK9sah86kG0
uNJTkBVrPwLc7B...jRuimrjT8dmId720CdQYzf1
5X50s80Y0A9FB0...FZBeBp/i3qQV1tVx8012hm
7Z8D828/Nq4puZ...KBqOa5JY2duEbP+/zCdk1kj
hLFvT0zd3iwMZ2...7+9Nw4U1uC3aPFduU0HsrW6
wcEio35kdMh11j...E/Jj19W0w==
=2f/i
-----END PGP PUBLIC KEY BLOCK-----

```

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

Step 6 Click **New GPG Public Key**. On the page that is displayed, set the following parameters.

Table 4-3 Parameters for creating a GPG public key

Parameter	Description
Title	Mandatory Custom GPG public key name with a maximum of 200 characters.
GPG Public Keys	Mandatory Paste the GPG public key copied from Step 4 to this text box.
Description	Optional. Enter a maximum of 200 characters. If the description is empty, -- will be displayed in the list.

Step 7 Click **OK**. The GPG public key is created successfully and the GPG public key list page is displayed.

 NOTE

- A GPG public key cannot be used repeatedly. If you fail to add a GPG public key, check whether you have added the public key and whether there are redundant spaces before and after the public key.
- After the public key is added successfully, you can view the added public key on the **GPG Public Keys** page. If the public key is no longer used, you can delete it.

----End

4.7 Configuring Git LFS

Git Large File Storage (LFS) is an extension of Git and is used to manage large binary files in Git repositories. Git LFS stores large files outside Git repositories to prevent Git repositories from becoming too large and slow. It supports most common binary file formats, including image, video, and audio. You can manage large files and code separately, and use Git's versioning function to track and manage them. Git LFS can also lock files and control versions to avoid conflicts when multiple users are editing large files at the same time.

To use Git LFS, you need to install the Git LFS client and enable the Git LFS extension in Git repositories. You need to add large files to the Git LFS tracking list for better management.

Table 4-4 Installing Git LFS

OS	Official Installation Guide Link
Windows	Windows Git-LFS installation guide
Linux	Linux Git-LFS installation guide
macOS	macOS Git-LFS installation guide

5 Migrating Code and Syncing a Repository

5.1 Repository Migration 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 a third-party git repository.](#)
- [Importing a local git repository.](#)
- [Migrating an SNV code repository.](#)

5.2 Migrating a Third-Party Git Repository to CodeArts Repo

5.2.1 Importing a Git Repository Using a URL

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Set repo type to **Import** and import from **Git Url**. For details about how to set parameters, see [Table 5-1](#).

Table 5-1 Parameters for obtaining authorization

Parameter	Description
Source Repository URL	<p>Mandatory. Specify the repo path to be imported. The source repo path must start with http:// or https:// and end with .git.</p> <p>NOTE</p> <ul style="list-style-type: none">• If the repo file is too large or the network quality is poor, it may take more than 30 minutes to import the repo file. If the import times out, you are advised to use the clone or push function on the client. For details, see External Repository Import Times Out.• The repository domain must be connected to the service node.
Verification to Access Source Repo	<p>Mandatory. There are two cases:</p> <ul style="list-style-type: none">• If the imported source repository is open to all visitors, select Not needed.• If the imported source repository is private, select Needed. Currently, two authentication modes are supported: By service endpoint and By username and password. For details about how to set parameters, see Verifying the Import Permission.

Step 3 Click **Next**. On the **Basic Information** page, set parameters by referring to the parameter [table](#).

Step 4 Set the parameters for **syncing a repo** by referring to [table 1](#).

----End

 **NOTE**

- After the parameters are set, the **Code** page for creating the repo is displayed.
- On the repository list page, if the new repository name is in gray with a red exclamation mark next to it, the repository fails to be imported. The possible cause is that the username, password, or access token is incorrect. You can delete the code repository and perform the preceding steps to import the external repo again.
- Currently, Git supports the following external import sources: bitbucket.org, code.aliyun.com, coding.net, git.qcloud.com, gitee.com, github.com, gitlab.com, visualstudio.com and xiaolv Yun.baidu.com.
- After a code repo is created, only the creator can access the repo. Other project members need to be manually added to the repo and assigned with permissions. Therefore, you need to manually add members to the repository and configure access permissions for the new members.

5.2.2 Importing a GitHub Repository

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Set the repo type to **Import** and import from **Github**.
- Step 3** Choose an authorization mode. You can grant permissions **By service endpoints** (see [Service Endpoint Authorization](#)) or **By personal access tokens** (see [Obtaining an Access Token](#)).
- Step 4** Click **Next**. On the **Select Repository** page, select the repo to be imported and click **Next**. On the **Basic Information** page, enter the basic information by referring to the [Entering Basic Information for a Repository](#) table and set the parameters for syncing a repo information by referring to [Table 1 Parameters for syncing repo settings](#).

----End

Service Endpoint Authorization

Table 5-2 Service endpoint authorization

Parameter	Description
Service Endpoint Name	Mandatory. Enter a name with a maximum of 256 characters.
Authentication Mode	Mandatory. Select a value as required. <ul style="list-style-type: none">If you select OAuth, click Authorize and OK, and the GitHub login page is displayed. Enter the GitHub login account and password, and click Authorize huaweidevcloud to complete the authorization. After the authorization is successful, Authorized successfully is displayed, and Service endpoint created successfully is displayed in the upper right corner. You can select the created endpoint from the drop-down list box.If you select By personal access token, use an account with the repo administrator permissions to create an access token on GitHub. For details, see Obtaining an Access Token.

5.3 Importing a Local Git Repository to CodeArts Repo

Importing a Local Git Repository

If your repo has not been incorporated into any version system, such as Git or SVN, perform the following operations in the root directory of the source code to import the local code repository to CodeArts Repo.

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Set **Repository Type** to **Common**, enter parameters, deselect **Generate README** and **.gitignore Programming Language**, and a code repository is created. The homepage of the code repository is displayed.
- Step 3** Run the **git init** command to create an empty Git repo directory on the local PC.
- Step 4** Run the **git add *** command to add the file to the version library.
- Step 5** Run the **git commit -m "init commit"** command to create an initial commit.
- Step 6** Run the **git remote add origin Remote repo address** command.
- Step 7** Run the **git push -u origin master** command to push the local Git repository to the code repository created in CodeArts Repo.



----End

NOTICE

After a code repo is created, only the creator can access the repo. Other project members need to be manually added to the repo and assigned with corresponding permissions. Therefore, you need to manually add members to the repository and configure access permissions for the new members.

NOTE

If the repo capacity of CodeArts Repo is about to be used up, go to the code repo details page and perform the following operations to clear code repo resources:

- Choose **Code > Branches**, select unnecessary branches, and click  to delete them.
- Choose **Code>Tags**, select unnecessary tags, and click  to delete them.
- Choose **Settings > Repo Management > Space Freeing** and clear the cache data.
- Choose **Settings > Repo Management > Submodules** and delete unnecessary submodules.

Importing a Local Third-Party Git Repository to CodeArts Repo

If you clone the code from a third-party Git repository to the local host and modify the code repository, you can perform the following steps to import the modified Git code repository to CodeArts Repo:

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Set **Repository Type** to **Common**, enter parameters, deselect **Generate README** and **.gitignore Programming Language**, and a code repository is created. The homepage of the code repository is displayed.
- Step 3** Run the **git commit -m "init commit"** command to create an initial commit.
- Step 4** Run the **git remote add origin Remote repo address** command.
- Step 5** Run the **git push -u origin master** command to push the local Git repository to the code repository created in CodeArts Repo.
- End

5.4 Migrating an SVN Code Repository

Import an SVN repo to CodeArts Repo

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Set repo type to **Import** and import from **SVN**. For details about how to set parameters, see [Table 5-3](#).

Table 5-3 Parameters for importing the SVN repo

Parameter	Description
Source Repository URL	<p>Mandatory. Specify the repo path to be imported. The source repo path must start with http://.</p> <p>NOTE</p> <ul style="list-style-type: none">If the repo file is too large or the network quality is poor, it may take more than 30 minutes to import the repo file. If the import times out, you are advised to clone or push the client. For details, see Using Git Bash to Import an SVN repo to CodeArts Repo.Online import is simple, and branches and tags in the SVN can be moved. If you want to continue development based on the code repository, use the Git Bash client to import the code repository. For details, see Using Git Bash to Import an SVN repo to CodeArts Repo.The repository domain must be connected to the service node.

Parameter	Description
Verification to Access Source Repo	Mandatory. There are two cases: <ul style="list-style-type: none">• If the imported source repository is open to all visitors, select Not needed.• If the imported source repository is private, select Needed. Currently, two authentication modes are supported: By service endpoint and By personal access token. For details about how to set parameters, see Verifying the Import Permission.

Step 3 Click **Next**. On the **Basic Information** page, set parameters by referring to the parameter [table](#).

Step 4 Set the parameters for **syncing a repo** by referring to [table 1](#).

----End

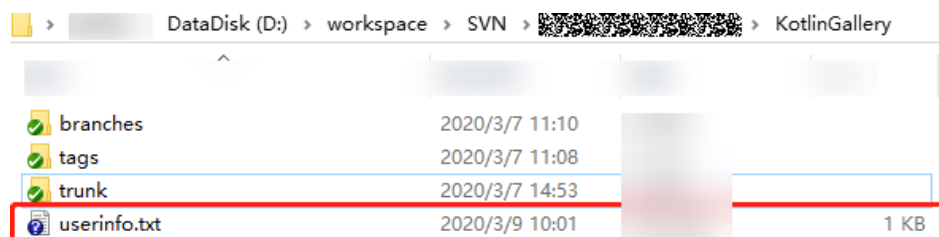
Using Git Bash to Import an SVN repo to CodeArts Repo

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:

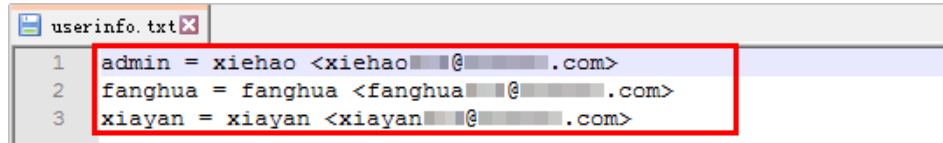
```
svn log --xml | grep "^<author" | sort -u | \awk -F '<author>' '{print $2}' | awk -F '</author>' '{print $1}' > userinfo.txt
```

After the command is executed, the **userinfo.txt** file is generated in the **KotlinGallery** directory, as shown in the following figure.



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 **userinfo.txt** so that in each line, SVN *author* = Git *author nickname* <*email address*>. The following figure shows the mapping format.



```
1 admin = xiehao <xiehao@... .com>
2 fanghua = fanghua <fanghua@... .com>
3 xiayan = xiayan <xiayan@... .com>
```

Step 2 Create a local Git repository.

1. Run the **git init** command to create an empty Git repo directory on the local PC.
2. Copy the **userinfo.txt** file in [step 1](#) to the directory and run the following command to switch to the directory:

```
cd Destination directory address
```

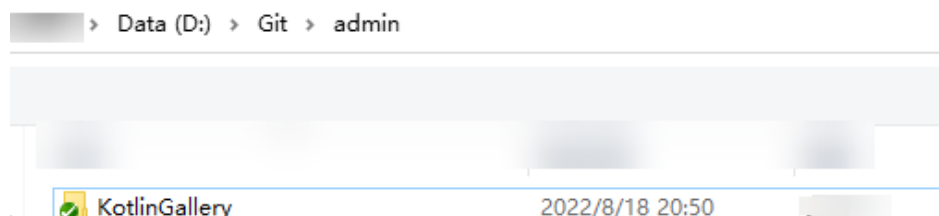
3. Start the Git Bash client in the directory and run the following command to clone a Git repo:

```
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	Indicates that the SVN metadata is not imported to the Git repo. In this way, the size of the Git code repo is reduced, but some SVN historical information may be lost.
--authors-file=userinfo.txt	Indicates that the specified user information file is used for author information mapping.
--trunk=trunk	Indicates that the trunk branch in SVN repo is used as the main branch of Git code repo.
--tags=tags	Indicates that the tags directory in the SVN code repo is used as the tag of the Git code repo.
--branches=branches	Indicates that the branches directory in the SVN code repo is used as the branch of the Git code repo.

After the command is executed successfully, a Git code repo named **KotlinGallery** is generated locally.



4. 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 directory structures in the SVN are successfully migrated in the form of Git branches.

Step 3 Correct local branches.

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 repository to CodeArts Repo.

1. Set the SSH key of the code repo. For details, see [Configuring the SSH Key](#).
2. Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
3. Set **Repository Type** to **Common**, enter related parameters, deselect **Generate README** and set **.gitignore Programming Language** to create a code repository. The homepage of the code repository is displayed.
4. Choose **Clone/Download** > **Clone with HTTPS** in the upper right corner and copy the HTTPS address.
5. Run the following command to associate the local code repo with CodeArts Repo and push the master branch to the code repo of CodeArts Repo: When running commands, enter the HTTPS account and password of CodeArts Repo.

```
git remote add origin HTTPS address of the new code repo
git push --set-upstream origin master
```

After the push is successful, go to the code repo homepage and choose **Code** > **Branches** to view the master branch in the current code repo.

6. Run the following command to push other local branches to CodeArts Repo:

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

After the push is successful, go to the code repo homepage and choose **Code** > **Branches**. The r1.1_hotfix branch is added to the code repo.

7. Run the following command to push tags from the local host to CodeArts Repo:

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

After the push is successful, go to the code repo homepage and select the **code** > **Tags**. The r1.0 and r1.1 tags exist in the code repo.

----End

5.5 Syncing Repo Settings

Syncing Repo Settings

Table 5-4 Parameters for syncing repo settings

Parameter	Description
Branch	Mandatory. The options are as follows: <ul style="list-style-type: none">• Default branch. The master branch automatically created when a code repo is created, for example, the master branch.• All branches. All branches in the code repo, including the default branch and other custom branches.
Schedule	Optional. NOTICE If you select this option, the imported image repository cannot commit code and can only be synced from the source repository periodically. The code is automatically refreshed every 24 hours. The refreshed content is the content of the source repository 24 hours ago.

5.6 Verifying the Import Permission

CodeArts Repo supports two permission verification modes: service endpoint verification and username and password authorization.

- Verifying permissions through service endpoints

Step 1 Link name Mandatory. Enter a name with a maximum of 256 characters.

Step 2 Git repository URL Mandatory enter the URL of the source repository to be imported.

Step 3 Username. This parameter is mandatory when the source repository is private. Username for cloning HTTPS code, for example, GitHub login name.

Step 4 Password or access token. This parameter is mandatory when the source repository is private. Password or access token for cloning HTTPS code, for example, the login password of GitHub or the access token created in GitHub. For details about how to obtain the value of this parameter, see [Obtaining an Access Token from GitHub](#).

----End

- Verifying permissions through username and password

You can also select **By username and password**. For details about how to set **Username**, see [setting username](#). For details about how to set **Password or access token**, see [setting password or access token](#).

5.7 Obtaining an Access Token

Obtaining an Access Token from GitHub

- Step 1** [Log in to GitHub](#), click the avatar in the upper right corner, and choose **Settings** > **Developer settings**.
- Step 2** Choose **Personal access tokens** > **Personal access tokens (classic)** > **Generate new token (classic)** and enter key information, as shown in [the following figure](#).

Figure 5-1 Enter key information about the new token.

The screenshot shows the GitHub interface for generating a new classic personal access token. Key elements include:

- Note:** A text input field containing the word "Test".
- Expiration:** A dropdown menu set to "No expiration" with a tooltip that says "The token will never expire!".
- Scopes:** A list of scopes with checkboxes, all of which are checked. The scopes are: "repo" (Full control of private repositories), "repo:status" (Access commit status), "repo_deployment" (Access deployment status), "public_repo" (Access public repositories), "repo:invite" (Access repository invitations), and "security_events" (Read and write security events).

- Step 3** Enter the necessary information to create a token. The new token page is displayed. Copy and save the token for it is temporary.

----End

5.8 Entering Basic Information for a Repository

Table 5-5 Entering basic information for a repository

Parameter	Description
Path	Optional. The default value is /, indicating that the repository does not belong to any repo group path. You can also select an existing repo group path from the drop-down list.

Parameter	Description
Repository Name	Mandatory. Name of the repo to be imported. Start with a letter, digit, or underscore (_), and use letters, digits, hyphens (-), underscores (_), and periods (.). Do not end with .git, .atom, or periods (.).
Description	Optional. Add a description for the repo. The description can contain a maximum of 2,000 characters.
Initial Settings	Optional. If you have enabled CodeArts Check, you are advised to select this option. After the repository is created, you can view the check task of the repository in the CodeArts Check task list.
Visibility	Optional. Indicates the visible scope of the source repo. The options are as follows: <ul style="list-style-type: none">• Public The value can be For project members, For tenant members, or For all guests.• Private: Only members of this repository can access it and commit code.

6 Creating a Repository

6.1 Creating Repos in Different Scenarios

CodeArts Repo supports the following methods of creating a repository:

- [Create a custom repository.](#)
- [Create a repository using a template.](#)
- [Fork a repository.](#)

You can create a repository as needed.

6.2 Creating a Repository

Step 1 Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.

Step 2 Select **Common** and set parameters based on the [following table](#).

Table 6-1 Parameters for creating a repo

Parameter	Description
Repository Name	Mandatory. Start with a letter, digit, or underscore (_). You can use letters, digits, hyphens (-), underscores (_), and periods (.). Do not end with .git, .atom, or periods (.).
Description	Optional. The value contains a maximum of 2000 characters.

Parameter	Description
.gitignore Programming Language	Optional. It is recommended that you fill in this parameter and select the programming language for your code repository from the dropdown list. This can effectively prevent unnecessary files from being tracked, thus keeping your repo clean and maintainable.
Initial Settings	Optional. The options are as follows: <ul style="list-style-type: none">• Generate README It is recommended that you select this option. After the file is generated, you can edit the README file to include information such as the project's architecture and compilation purpose, which will help others quickly understand the repo.• Automatically create Check task (free of charge) It is recommended that you select this option. After a code repository is created, you can view the check task of the repo in the CodeArts Check task list.
Visibility	Optional. You can select either of the following options as need: <ul style="list-style-type: none">• Private: Only repository members can access and commit code.• Public The value can be For project members, For tenant members, or For all guests. NOTE Repos can be set to Private or Public . Go to the details page of a code repo, choose Settings > General Settings > Repository Information , and modify the visibility for the repo.
Open-Source License	This parameter is mandatory when Visibility is set to Public . Select an existing license from the drop-down list.

----End

6.3 Creating a Repository Using a Template

- Step 1** Go to the CodeArts Repo homepage, click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.
- Step 2** Select **Template**. You can select **CodeArts Templates** or **Custom Templates**. You can set the official template as your custom template in the repo settings. After selecting a template, set parameters based on the [table](#).

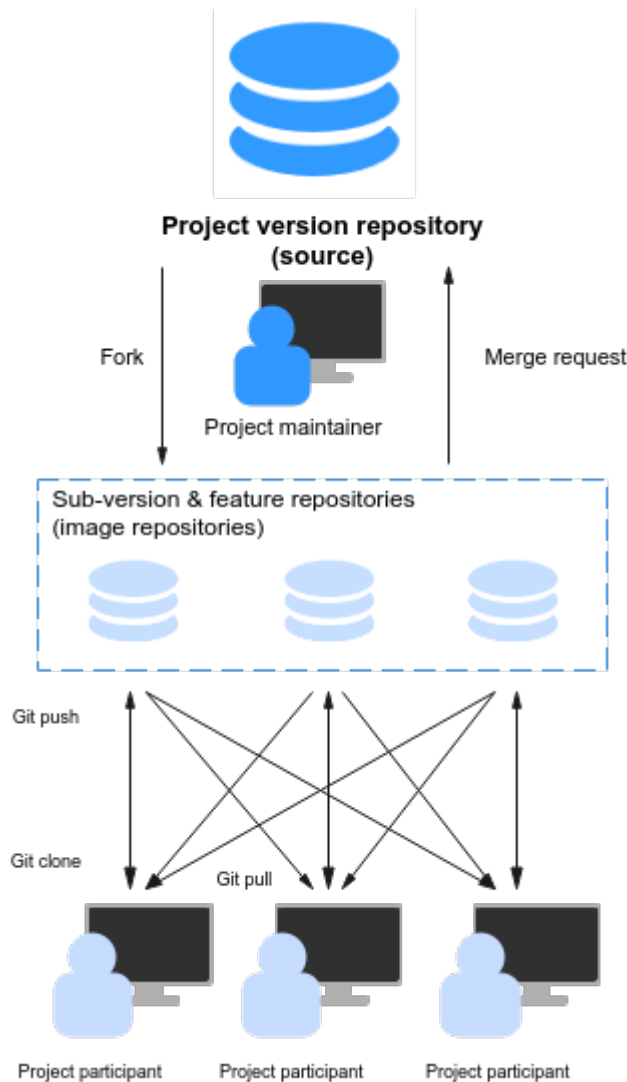
----End

6.4 Forking a Repository

Application Scenarios

The fork function can be used in large-scale projects with multiple sub-projects. You can fork a repository (an image) based on a repository and merge the CRs in the image to the source repository. When there is no merge, the modification of both the image repository and source repository will not affect each other.

As shown in the following figure, the complex development process occurs only in the image repository and does not affect the project version repo (source repo). Only the confirmed new features can be merged back to the project version repo. Therefore, fork is a team collaboration mode.



Differences Between Forking a Repository and Importing an External Repository

Both forking or importing a repo is a process of 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.

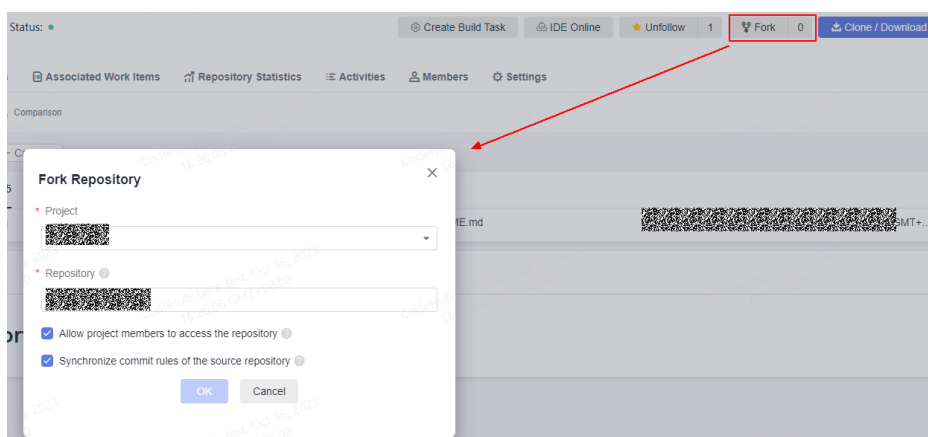
- An import also generates a repository copy based on the current version of the source repository. The difference is that you can pull the default branch of the source repository to the repository copy at any time to obtain the latest version, but you cannot apply for merging changes made on the repository 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 displayed **Fork Repository** dialog box, select a target project, enter a repository name, and decide whether to select **Synchronize commit rules of the source repository**.



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

----End

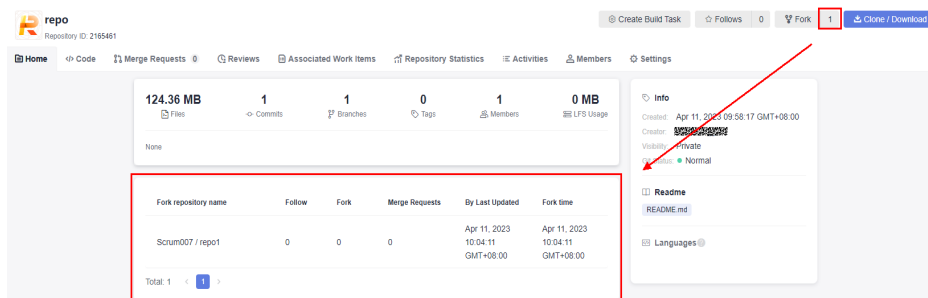
Viewing the List of Forked Repositories

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.

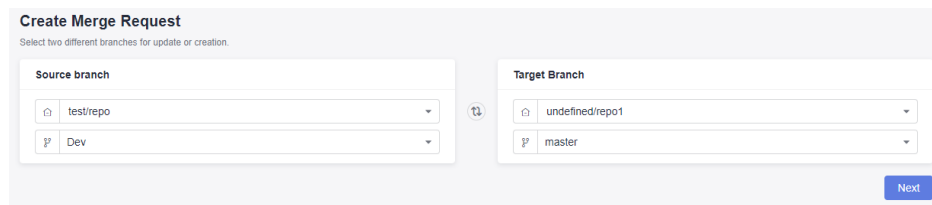
You can click the name of a forked repository to access the repository.



----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 **Create MR**.
- 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 **OK**. 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, refer to the process of 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, scorers, approvers, and mergers must be members of the source repository.

7 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.

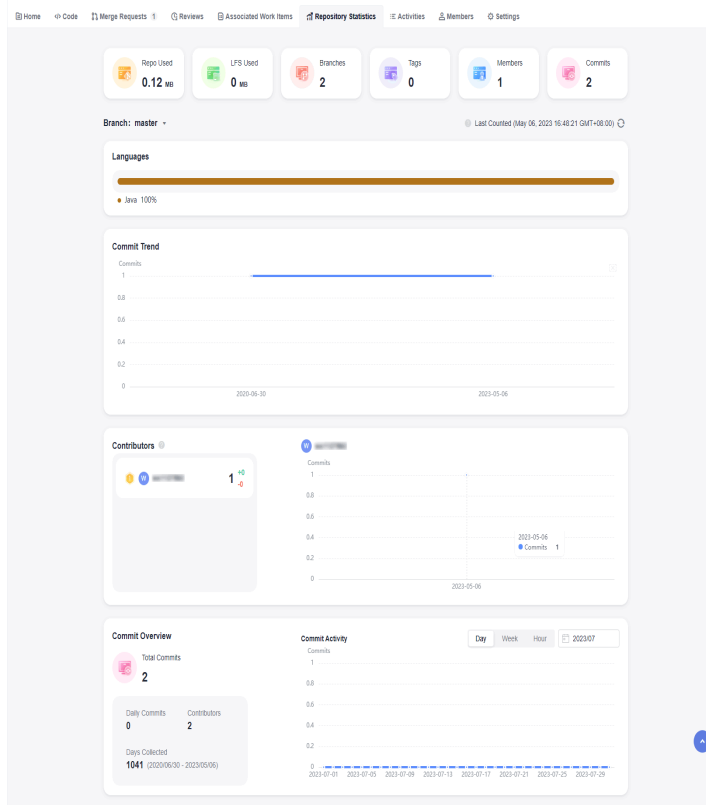
8 Viewing Repository Statistics

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

- Repository members can trigger statistics collection by contributor and language.
- 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 ("+" indicates the added code lines and "-" indicates the deleted code lines).
- Commits (an operation that combines two or more historical development records) of the merge node are not counted.



9 Configuring Repository Settings

Configuring Repo-level Settings

If **Force inherit** is selected in the project-level **Repo Settings**, **Repo Settings** is not supported for specific repos.

If the project-level configuration is not inherited, set parameters by referring to [the following table](#).

Table 9-1 Parameters for repo-level settings

Parameter	Description
Default Branch	Optional. The master branch is set as the default branch when the code repo is created.
Whitelist for creating branches	Optional. By default, this parameter is not selected. If this parameter is selected, the whitelist for developers to create branches is enabled. Only developers can be added to the whitelist. Non-developers will not be displayed and will not take effect even after configuration.
Do not fork a repository	Optional. Once selected, no one can fork the repo in the project.
Pre-merge	Optional. Once this is selected, the server automatically generates the pre-merge code of the MR. 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.

Parameter	Description
Branch name rule	Optional. All branch names must match the regular expression with max. 500 characters. If this field is left blank, any branch name is allowed. The rules must meet the following tag naming rules: <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .
Tag Name Rule	Optional. All tag names must match the regular expression specified by this parameter. If this field is left blank, any tag name is allowed. The basic tag naming rules must be met. <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .

Configuring a Submodule

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 repo or a repo developed by yourself for multiple parent projects) in project A, and use them as two separate projects. Submodules allow you to use a Git repo as a subdirectory of another Git repo. This means that you can clone another repo into your own project while keeping commits independent.

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**.

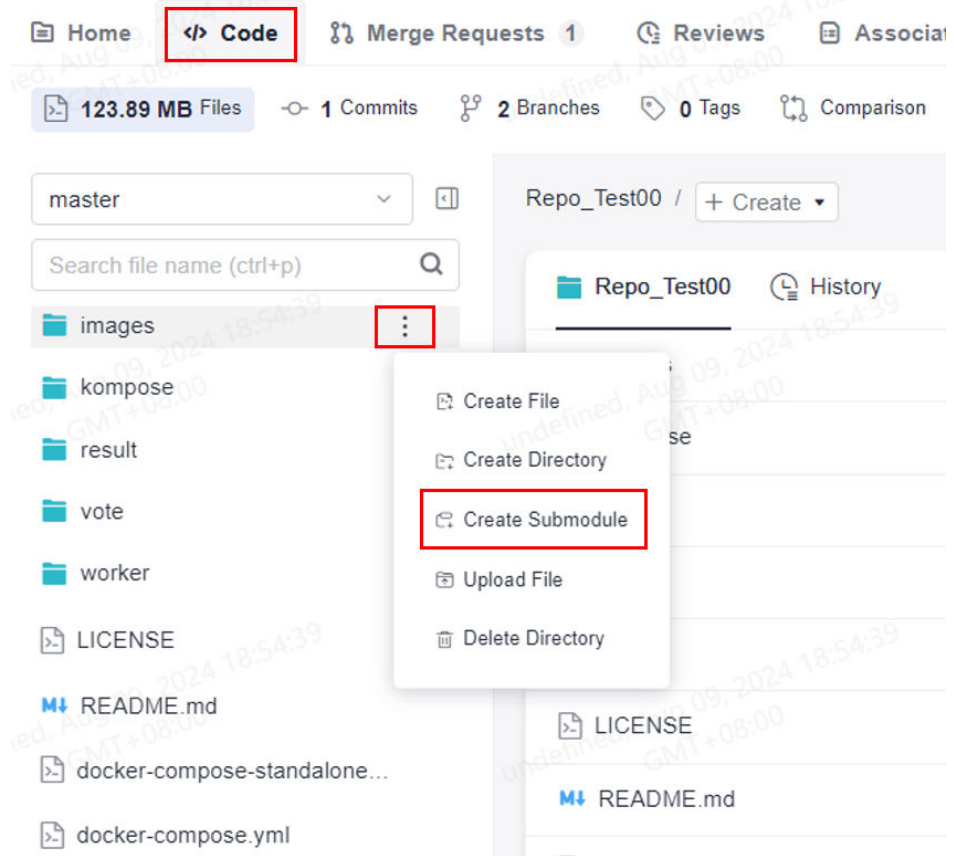
Using the Console

- **Creating a submodule**

- **Entry 1:**

You can add a submodule to a folder in the repository file list.

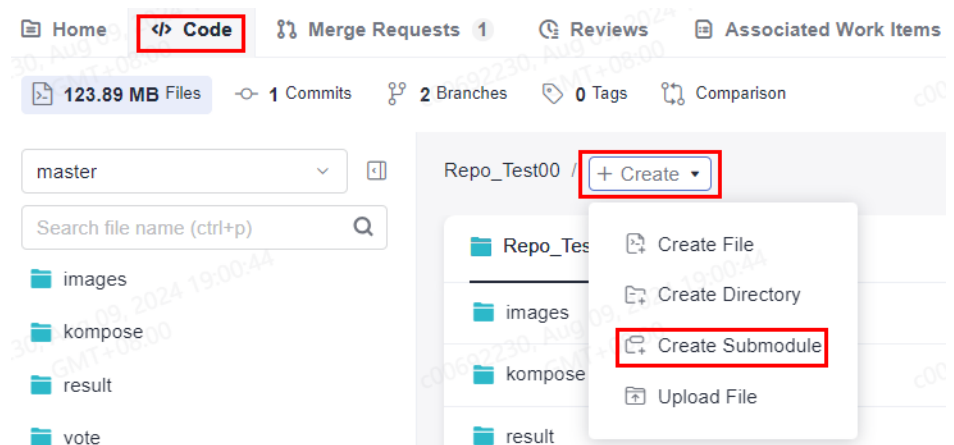
Click  and select **Create Submodule**, as shown in the following figure.



- **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 submodule

Parameter	Description
Submodule Repo Path	Select a repository as the submodule.
Submodule Repo Branch	Select the target branch of the submodule to be synchronized to the parent repository.
Submodule File Path	Path of the submodule file in the repository. Use / 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 (child 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

9.1 Configuring Repository Policies

9.1.1 Configuring Protected Branch Rules

Configuring Project-Level Protected Branch Rules

CodeArts Repo makes code branches more secure by preventing anyone other than the administrator from committing code, preventing anyone from forcibly committing code, or from deleting the branch. You can set this branch to be protected. The procedure is as follows: On the CodeArts Repo homepage, go to the project homepage, choose **Settings > Policy Settings > Protected Branch**, click **Create Protected Branch**, and set parameters as follows.

Step 1 Enter a branch name. This parameter is mandatory. Enter a complete branch name or a branch name with wildcard characters. If a branch contains a single slash (/), the branch cannot be matched using the wildcard * due to the fnmatch syntax rule.

Step 2 You can set the push or merge permission for the administrator/project manager, committer, and developer. These two permissions cannot be granted at the same time because the protected branch cannot be forcibly pushed or merged into the code. You can create, edit, and delete protected branches in batches.

----End

If you want all repo groups and repo in this project to use the preceding settings, select **Force inherit**.

9.1.2 Configuring Protected Branch Rules

Setting Repo-Level Protected Tag Rules

Go to the repo homepage, choose **Settings > Policy Settings > Protected Tags**, click **Create Protected Tag**, and set parameters by referring to the following table.

Table 9-3 Parameters for creating a protected tag

Parameter	Description
Tags	Mandatory. Enter a complete tag or a tag with a wildcard as required. Only one branch can be added at a time. Batch adding is not supported. The value must start with refs/heads/ and end with * . Special characters are not allowed in other positions.
Allowed to Create	Mandatory. Roles allowed to create protected tags . You can select a role with permission to create protected tags from the drop-down list box.

9.1.3 Configuring Code Commit Rules

Configuring Repo-Level Commit Rules

CodeArts Repo allows you to create verification and restriction rules for code commits to ensure code quality. You can select **Inherit from project** to automatically inherit and use the project settings. The settings cannot be modified.

You can also access the CodeArts Repo homepage. Choose **Settings > Policy Settings > Commit rules**, and click **Create Commit Rule**. For details about the parameters, see [Table 16-8](#).

Table 9-4 Parameters on the Commit Rules page

Parameter	Description
Reject non-signed-off-by commits	<p>Only signed-off-by commits are 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>
Reject commits not signed by GPG	<p>Only GPG-signed commits are pushed to the repository.</p> <p>Configure a GPG key:</p> <pre>git config --global user.signingkey "your GPG private key"</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>When running the tag command on the Git client, you need to add the <code>-s</code> parameter.</p> <pre>git tag -s -m "Your tag message"</pre> <p>You need to configure the signature and email address on the client in advance.</p>
Tags cannot be deleted	After this option is selected, tags cannot be deleted on the page or by running commands on the client.
Prevent committing secrets	Example: <code>id_rsa</code> and <code>id_dsa</code> files.
Prevent git push -f	<p>Indicates whether users can run the git push -f command on the client to push code.</p> <p>git push -f indicates that the current local code repository is pushed to and overwrites the code in CodeArts Repo.</p> <p>In general cases, you are not advised using this command.</p>

9.1.4 Review Comments

On the repository details page, choose **Settings > Policy Settings > Review Comments**. It can standardize the review comments and configure review comment templates.

The settings take effect only for the repository configured.

All repository members can view this page. For details about whether a repository member has repository setting permissions, refer to [Permissions](#).

Setting Review Comments

Step 1 Select **Enable review comment types and modules** as needed.

Step 2 Configure review comment categories.

- Enable preset comment types

If you select **Enable preset comment types**, you can directly use the preset review comment categories.

- Custom types

You can customize review comment categories. Enter a type name and press **Enter** to save the settings.

NOTE

Enter a category name and press **Enter**. The name cannot contain colons (:) and can contain a maximum of 200 characters. A maximum of 20 category names separated with commas (,) can be entered and must be unique.

Step 3 Enter a category name in the text box under **Comment Modules**.

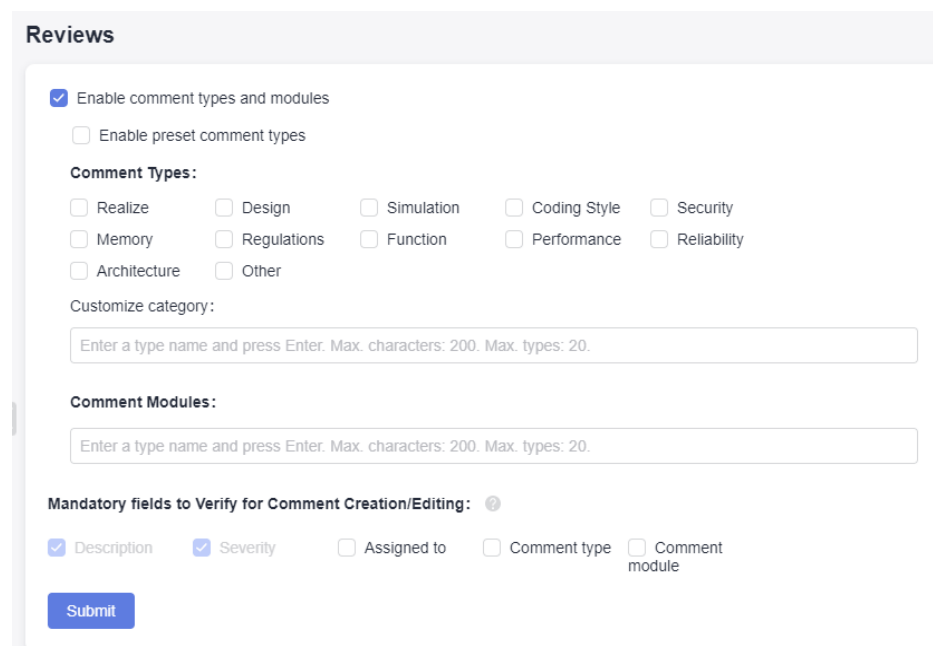
NOTE

Enter a module name and press **Enter**. The name can contain a maximum of 200 characters. A maximum of 20 category names separated with commas (,) can be entered and must be unique.

Step 4 Set **Mandatory Fields to Verify for Comment Creation/Editing** as required.

Step 5 Click **Submits**.

----End



Reviews

Enable comment types and modules

Enable preset comment types

Comment Types:

Realize Design Simulation Coding Style Security

Memory Regulations Function Performance Reliability

Architecture Other

Customize category:

Enter a type name and press Enter. Max. characters: 200. Max. types: 20.

Comment Modules:

Enter a type name and press Enter. Max. characters: 200. Max. types: 20.

Mandatory fields to Verify for Comment Creation/Editing:

Description Severity Assigned to Comment type Comment module

Submit

9.1.5 MR Evaluation

This function is used to set MR evaluation dimensions. After the dimensions are set, you can evaluate the dimensions on the MR details page.

Setting MR Evaluation

Step 1 Select **Enable MR User-defined Evaluation Dimension Classification**. You can add evaluation dimensions.

Enter a dimension name and press **Enter** to save the settings. The name can contain a maximum of 200 characters. A maximum of 20 dimensions can be created.

 **NOTE**

If **Enable MR User-defined Evaluation Dimension Classification** is not selected, the single-dimension MR evaluation is performed.

Step 2 Click **Submits**.

----End



MR Evaluation

Enable MR User-defined Evaluation Dimension Classification

Evaluation Dimension

Clear architecture ✕ Moderate amount of code ✕ Complete comments ✕ Clear code logic ✕ Proper exception handling ✕

Enter a type name and press Enter. Max. characters: 200. Max. types: 20.

Submit

10 Hierarchical Repository Management

10.1 Creating a Repository Group

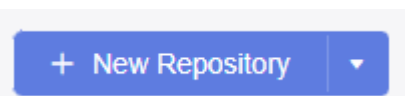
Overview

A repository group consists of one or more repositories. You can configure and manage repository rules for repositories or child repository groups in a repository group, including commit rules and member permissions.

 **NOTE**

A maximum of five levels of repo groups can be created.

Creating a Repository Group



In the project or parent organization, click the icon, and select **New Repository Group** from the drop-down list box. On the displayed page, enter basic information according to the following table and click **OK**. A maximum of three levels of repository groups can be created.

Table 10-1 Parameters for creating a repository group

Parameters	Mandatory	Remarks
Responsible Project	Yes	<ul style="list-style-type: none">The repository group must exist in a project.If your account does not have a project, click Create Project in the drop-down list box to create a Scrum project. NOTE You can create a project only when you create a repository group on the CodeArts Repo homepage.
Path	No	The repository group path corresponds to the groupid parameter of the API for creating a repository. If groupid is empty, there is no corresponding repository group for the repository in the project. Select a repository group path as required. The repository group path range is the root organization paths of all first-level repository groups and child repository groups.
Name	Yes	Start with a letter, digit, or underscore (_), and use letters, digits, hyphens (-), underscores (_), and periods (.). Do not end with .git, .atom, or periods (.). Max. 256 characters.
Description	No	Describe your repository group. Max. 2000 characters.
Visibility	Yes	You can choose Private (default) or Public . <ul style="list-style-type: none">Private Only repository group members can access. The child repository groups and repositories in a private repository group can only be private.Public Read-only for visitors via referral link and hidden from repo lists and search results. Child repository groups and repositories in a public repository group can be private or public.

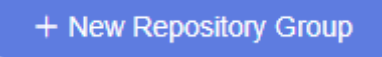







10.2 Using Repository Groups

10.2.1 Viewing the Repository Group List

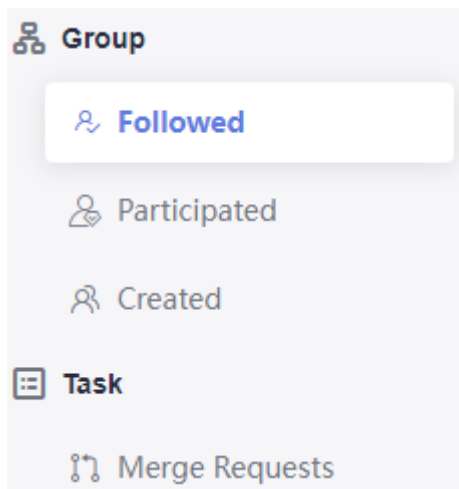
You can use either of the following methods to access the repository group list page of CodeArts Repo:

On the CodeArts homepage, choose **Services** > **Repo**. The repositories that you participated in are displayed by default. Click any menu under **Group** to go to the repository group list page.

You can create and configure a repository group.

- : Click this icon to access the page for creating a repository group.
- : Click this icon to follow the repository group. You can view the repository group in followed repository group list.
- : Click this icon on the right of a repository group to access the homepage of child repository groups.
- : Click this icon next to the parent repository group. The **Repositories**, **Members**, **Settings**, and **New Child Repository Group** icons are displayed.
 - : Click this icon to access the repository (group) list page.
 - : Click this icon to access the repository group members page.
 - : Click this icon to access the **Repository Group Information** page on the **Settings** tab page.
 - : Click this icon to access the page for creating a child repository group.

On the personal homepage, you can view **Followed**, **Participated**, and **Created**. In the upper right corner, you can filter repository groups using **By Last Created** and **By Last Updated**.



10.2.2 Viewing Repository Group Details

Click a repository group name in the repository group list to go to its details page. CodeArts Repo provides various console operations. The details are described in the following table.

Table 10-2 Description


Function	Description
Repository (Groups)	The number of repository groups, repositories, open MRs, and members are displayed. You can also create repositories and view unlocked ones.

Function	Description
Members	On the repository group member management page, members can be added and repository group member roles can be adjusted.
Settings	Entry for the repository group settings. All members of the repository group can view the settings, but only the project administrator or repository group owner can modify the settings.

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

: Click this icon next to SSH or HTTPS to obtain the repository address.



: Click this icon to follow the repository group.

: Click this icon under a repository group to view the number of repositories, access each repository, view and set repository group members, create child repository groups or repositories, create repositories based on templates, and import external repositories. Click this icon under a repository to associate work items, manage members, and delete the repository.

10.2.3 Viewing the Repository Group Homepage

The repository group homepage displays its basic information.

Table 10-3 Parameters

Parameter	Description
Child Repository Groups	Number of child repository groups.
Repositories	Number of repositories.
Opening Merge Request	Number of opening MRs.
Members	Number of members in a repository group. Click  to go to the Members tab page for management.
New Repository	Click  to go to the New Repository page and create a repository.
All Repositories	Both locked and unlocked repositories included.

10.2.4 Managing Members of a Repository Group

CodeArts Repo allows you to add members or member groups to a repository group.

- **All, Groups, Pending, and Add Member** are located on the **Members** tab page of repository group details.
 - **Username, User Source, Project Member Role, Repository Group Role, and Operation** of all members in the repository group are displayed in **All**.
 - **Member Group Name, Number of Members, Description, and Operation** of all member groups in the repository group are displayed in **Groups**.
 - **Pending** displays the members to be reviewed in the repository group, including the **Username, Project Member Role, Repository Group Role, and Operation**. A member to be reviewed can be set to **Agree** or **Reject** by a user who has permission to add members.
 - You can add members or member groups on the **Add Member** page.

 **NOTE**

Members in a parent repository group are unconditionally inherited to its child repository groups or child repositories and cannot be deleted.

When a project role changes, the repository role that is consistent with this role is updated synchronously. The priority of the member role inherited by the repository group or added to the member group is subject to the latest update.

As the administrator of this repository, the repository owner has full permissions for the repository and cannot be removed or edited.

The project administrator has the highest permission in the project and automatically becomes an administrator in this repository. They have full permissions for this repository and cannot be deleted or modified.

The repository group creator has all permissions for the repository group and its child repository groups and repositories. The creator cannot be deleted or modified.

This member is added in a member group and can only be deleted in the group.

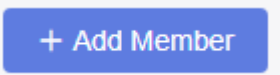
This member is from the upper-layer repository group and can only be deleted in that repository group.

Adding a Member or Member Group to a Repository Group

Step 1 Go to the CodeArts homepage and click the target project name to access the project.

Step 2 Choose **Services > Repo**.

Step 3 Find the parent organization of repository group and go to the repository group homepage.

Step 4 Click **Members** and click the  icon. The **Add Member** dialog box is displayed.

Step 5 In the **Add Member** dialog box, click **Members**, select a target member, Click **OK**.

Step 6 In the **Add Member** dialog box, click **Member Groups**, select a target member group from the drop-down list box, and click **OK**.

----End

10.3 Configuring Repository Groups

10.3.1 Repository Group Information

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

The settings take effect only for the repository group configured.

All members in a repository group can view this page, but only the project administrator and repository group creator have the setting permission.

By default, the repository group name cannot be changed.

Repository group description is used to describe repository group information.

10.3.2 Repository Settings

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

The default branch is selected when you enter the current repository group or create an MR. Each new repository in a repository group has a default branch - master, which can be changed at any time.

The settings take effect only for the repository group configured.

All repository members can view this page. For details about whether a repository member has repository setting permissions, refer to the **Permissions** page. After the setting is complete, click **Submit**.

Table 10-4 Description

Parameter	Description
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.

Parameter	Description
Branch name rule	<p>All branch names must match the regular expression specified by this parameter. If this parameter is left empty, any branch name is allowed. The value must comply with the basic branch naming rules and contain a maximum of 500 characters. Example: ^feature-[0-9a-zA-Z]+</p> <ul style="list-style-type: none">• Max. 500 characters.• The name cannot start with -, refs/heads/, or refs/remotes/, and cannot contain spaces or special characters such as <code>[\<~^:?!()"]</code>. It cannot end with <code>./</code> or <code>.lock</code>.• The name of a new branch cannot be the same as that of an existing branch or tag.
Tag name rule	<p>All tag names must match the regular expression specified by this parameter. If this parameter is left empty, any tag name is allowed. The tag name must comply with the basic tag naming rules and contain a maximum of 500 characters. Example: ^TAG*\$</p> <ul style="list-style-type: none">• Max. 500 characters.• The name cannot start with -, refs/heads/, or refs/remotes/, and cannot contain spaces or special characters such as <code>[\<~^:?!()"]</code>. It cannot end with <code>./</code> or <code>.lock</code>.• 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-merge

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 **Pre-merge** is selected, 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 **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.

Commands

The pre-merge commands on the server are as follows:

```
git init
git remote add origin {repo_url clone or download URL}
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 or download URL}
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.

10.3.3 Risky Operations

On the repository group details page, choose **Settings > Risky Operations**.

All members of a repository group can view this page, but only the project administrator and repository group owner can modify it.

Currently, the following operations are available:

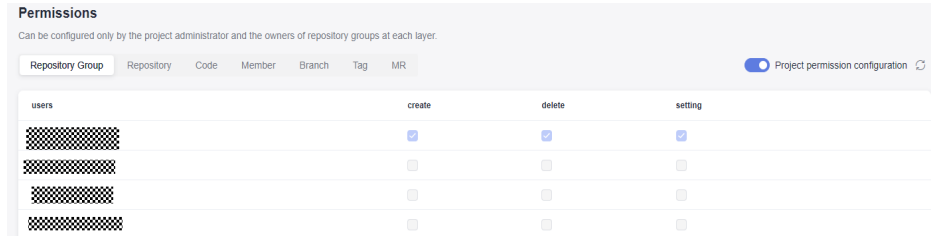
- **Deleting Repository Group:** This will delete all its child repository groups and resources. Caution! The deleted repository group cannot be restored.
- **Rename Repository Group:** This will invalidate the original repository group path and repository path and may cause unexpected situations.

NOTE

- Caution! Changing the repository group name will invalidate the original clone path of this repository group. Check and update related configurations.
- If a pipeline is configured for a repository in a repository group, the pipeline cannot be triggered after the repository group name is changed. In this case, you need to update the pipeline configuration (execution plan and pipeline source). For details, see > "Configuring a Pipeline".
- After renaming a repository group, you need to check and modify related configurations for CodeArts Build, CodeArts Check, CodeArts Deploy, and CodeArts IDE Online.

10.3.4 Permission Management

Permissions is on the **Settings** tab page of repository group details.



You can configure permissions for each role according to the following table.

NOTE

The repository group permission matrix can be modified only by the project administrator and the owners of repository groups at each layer.


If a repository member is inherited from a repository group, the member uses the role of the repository group by default. After changing the role of the repository member in the repository, click  in the **Operation** column of the repository member on the **Member List** tab page. The permission of the role is changed to that of the repository group role.

Table 10-5 Permissions of the repository group roles

Role/ Function	Permission	Project Manager	Committer	Developer	System Engineer	Test Manager, Tester, Participant, Operation Manager, and Product Manager	Viewer	Customized Role
Repository Group	Create	B	B	B	B	C	D	C
	Delete	B	D	D	D	D	D	C
	Setting	B	D	D	D	D	D	C
Repository	Create	B	B	B	B	C	D	C
	Fork	B	B	B	B	C	D	C
	Delete	B	D	D	D	D	D	C

Role/ Function	Permission	Project Manager	Committer	Developer	System Engineer	Test Manager, Tester, Participant, Operation Manager, and Product Manager	Viewer	Customized Role
	Setting	B	D	D	D	D	D	C
Code	Commit	B	A	A	A	C	D	C
	Download	B	A	A	A	C	D	C
Member	Add	B	D	D	D	D	D	C
	Edit	B	D	D	D	D	D	C
	Delete	B	D	D	D	D	D	C
Branch	Create	B	B	B	B	C	D	C
	Delete	B	B	B	B	C	D	C
Tag	Create	B	B	B	B	C	D	C
	Delete	B	C	C	C	C	D	C
MR	Create	B	B	B	B	C	D	C
	Edit	B	B	C	C	D	D	C
	Comment	B	B	B	B	C	C	C
	Review	B	B	B	B	D	C	C

Role/ Function	Per miss ion	Proje ct Man ager	Com mitt er	Devel oper	Syste m Engi neer	Test Mana ger, Tester, Partici pant, Opera tion Mana ger, and Produc t Mana ger	Viewer	Customize d Role
	Appr ove	B	B	C	C	D	D	C
	Mer ge	B	B	C	C	D	D	C
	Clos e	B	B	C	C	D	D	C
	Re- open	B	B	C	C	D	D	C

 **NOTE**

- A: indicates that the role has the permission by default and the permission cannot be removed.
- B: indicates that the role has the permission by default and the permission can be removed.
- C: indicates that the role can have the permission assigned.
- D: indicates that the role cannot have the permission assigned.

Permissions is on the **Settings** tab page of repository details.

You can configure permissions for each role according to the following table.

Table 10-6 Permissions of the repository roles

Role/ Function	Per miss ion	Proje ct Man ager	Com mitt er	Devel oper	Syste m Engi neer	Test Mana ger, Tester, Partici pant, Opera tion Mana ger, and Produc t Mana ger	Viewer	Customize d Role
Repository	Fork	B	B	B	B	C	D	C
	Dele te	B	D	D	D	D	D	C
	Setti ng	B	D	D	D	D	D	C
Code	Com mit	B	A	A	A	C	D	C
	Dow nloa d	B	A	A	A	C	D	C
Member	Add	B	D	D	D	D	D	C
	Edit	B	D	D	D	D	D	C
	Dele te	B	D	D	D	D	D	C
Branch	Crea te	B	B	B	B	C	D	C
	Dele te	B	B	B	B	C	D	C
Tag	Crea te	B	B	B	B	C	D	C
	Dele te	B	C	C	C	C	D	C
MR	Crea te	B	B	B	B	C	D	C
	Edit	B	B	C	C	D	D	C

Role/ Function	Per miss ion	Proje ct Man ager	Com mitt er	Devel oper	Syste m Engi neer	Test Mana ger, Tester, Partici pant, Opera tion Mana ger, and Produc t Mana ger	Viewer	Customize d Role
	Com men t	B	B	B	B	C	C	C
	Revi ew	B	B	B	B	D	C	C
	Appr ove	B	B	C	C	D	D	C
	Mer ge	B	B	C	C	D	D	C
	Clos e	B	B	C	C	D	D	C
	Re- open	B	B	C	C	D	D	C

 **NOTE**

- A: indicates that the role has the permission by default and the permission cannot be removed.
- B: indicates that the role has the permission by default and the permission can be removed.
- C: indicates that the role can have the permission assigned.
- D: indicates that the role cannot have the permission assigned.

11 Configuring a Repository

11.1 Configuring Repository Settings

Configuring Synchronization Between Repos

CodeArts Repo allows you to synch the current repo settings to other repos across projects but not regions.

This function is used for a repository forked based on the repository because the settings are not automatically copied during forking.

If you have enabled **Inherit from project**, you cannot synch settings.

Only members with the **Set** permission can perform this operation. Members in the repo can view this page.

Go to the repo homepage and choose **Settings > Repo Management > Sync Settings >**. Click **Add Repository**. In the dialog box that is displayed, select the target repository.

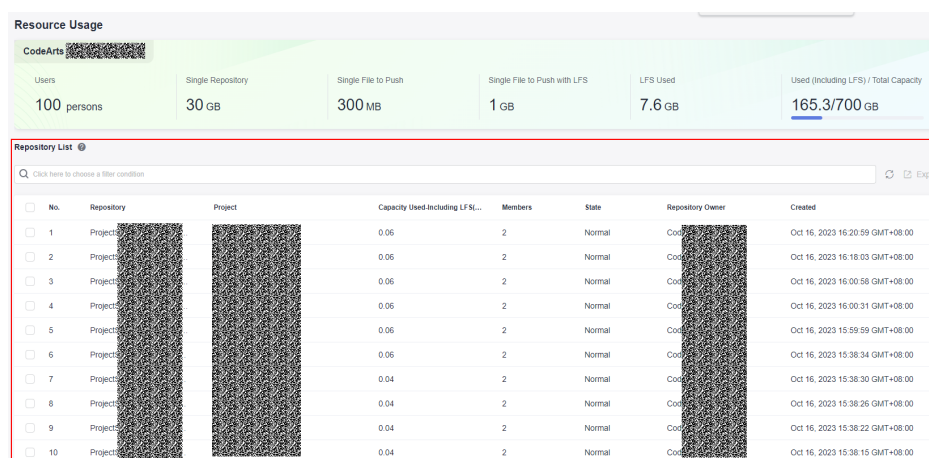
NOTE

- Ensure that the network connection is normal before synchronizing a repository.
 - CodeArts Repo supports accessing other public platforms' to code repos.
 - For private repository platforms on the intranet, ensure that the network connection between CodeArts Repo and your repository is normal.
- Common Failure Causes
 1. Failed to sync **Commit Rules**: No commit rules are set for the source repository.
 2. Failed to sync **Protected Branches**: The branch names of the source repository and target repository are different.


11.2 Viewing the Repository List

You can view your repo list in three ways: **Favorited**, **Joined**, and **Created**. You can access the repository list in the following ways:

- On the CodeArts homepage, click **Services** > **Repo** under the navigation bar. The repository list page of CodeArts Repo is displayed, showing all your code repositories.
- On the CodeArts homepage, click a project, and click **Code** > **Repo** in the navigation pane on the left to enter the repo list page.
- If you have subscribed to the new CodeArts package or purchased any CodeArts Repo package, go to the CodeArts homepage, click your avatar, and choose **All Account Settings** > **Repo** > **Resource Usage**. On the **Repo List** tab page, click a project to go to the repo list page, as shown in the **following figure**. On the CodeArts Repo homepage, click the storage usage and repos in the upper left corner to go to the **Resource Usage** page directly.

Figure 11-1 Resource Usage

Choose **Repos** > **Joined** to view all repositories that you have participated in. If

you select  in the row where a repo is located, you will favorite the repo. You can choose **Repos** > **Favorited** to view the repo that you have favorited. To view the repo you created, choose **Repos** > **Created**.

11.3 Viewing Repository Details

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

Table 11-1 Description

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

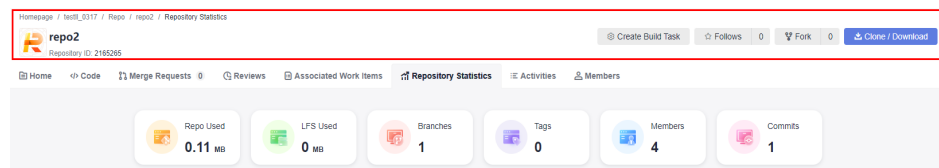
Page	Function Description
Code	<ul style="list-style-type: none">• File list: You can create files, directories, and submodules, upload files, modify files and blame, and view commit history.• Submit: You can view commit records and repository network diagrams.• Branch: Branches can be managed on the 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.
Activities	You can view the dynamic information about the repository.
Members	Member management is supported. The details are as follows: <ul style="list-style-type: none">• All, Groups, Pending, and Add Member are located on the Members tab page of repository details.<ul style="list-style-type: none">– Username, User Source, Project Member Role, and Repository Member Role of all members in the repository are displayed in All.– Member Group Name, Number of members, Description, and Operation of all member groups in the repository are displayed in Groups.– Pending displays the members to be reviewed in the repository, including their Username, Alias, Enterprise User, Project Member Role, Repository Group Role, and Operation. A user with permission to add members can set a member to be reviewed as either Agree or Reject.– You can add members or member groups to a repository on the Add Member page.
Settings	Entry for setting the repository. All repository members can view this page. For details about whether a repository member has repository setting permissions, refer to the Permissions page.

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

- **Create Build Task:** Create a build task.
- **Follow:** Click to follow the repository. The followed repositories are pinned on top.
- **Fork:** displays the number of forks of the repository. Click this button, the **Fork Repository** page is displayed.
- **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.



- Rules for displaying the code check status:
 - If you have the code check permission, the code check status is displayed next to the repository name.
 - If you do not have the code check permission, no code check status is displayed next to the repository name.
 - Rules for displaying the build status:
 - If you do not have the build permission, only the **Create Build Task** button is displayed on the top of the repository page.
 - If you have the build permission, and no build task is set, the **Create Build Task** button is displayed on the top of the repository page.
- If a build task is set, the following statuses are displayed on the top of the repository page: **Build task running**, **Building**, **Build failed**, and **Build successful**.

11.4 Viewing Repository Homepage

The **Home** tab page displays the basic information about repo, as shown in the following figure.

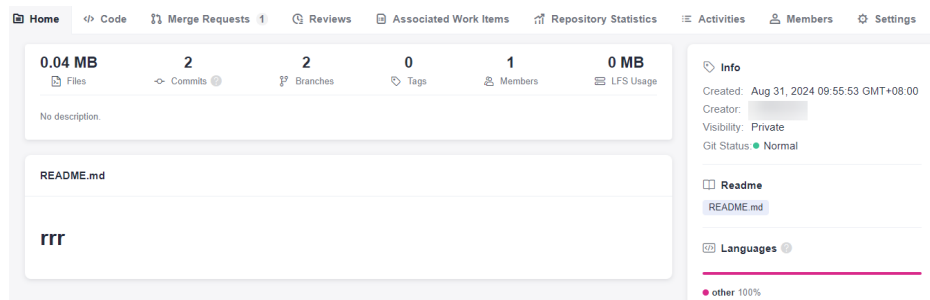
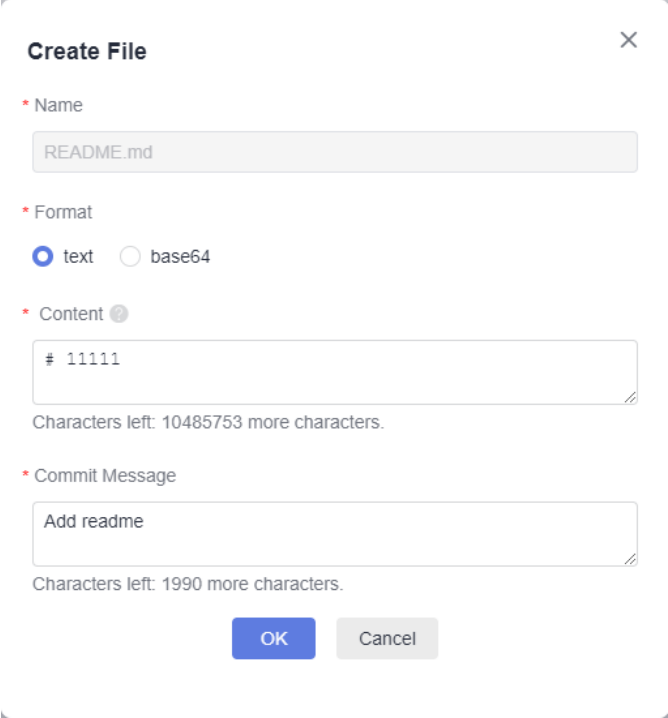


Table 11-2 Parameter description

Parameter	Description
Files	<p>Capacity of the current repository The preceding figure shows that 0.04 MB capacity has been used by the current repo.</p> <p>NOTE</p> <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	<p>Number of commits in the default branch of the repo. You can click the number or icon to go to the Commits page under the Code tab page and view the commit details. This example indicates that there are two commits.</p>
Branches	<p>Displays the number of branches in the current repository. You can click the branch icon or the number above it to go to the Code tab page and manage Branches.</p>
Tags	<p>Displays the number of tags in the current repository. You can click the icon to go to the Code tab page and manage tags.</p>
Members	<p>Displays the number of members in the current repository. You can click the icon to go to the Members tab page and manage members.</p>
LFS Usage	<p>Collect statistics on the LFS usage of the current repository.</p>
Repository description	<p>The description entered during repository creation.</p>

Parameter	Description
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.
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.

11.5 Backing Up a Repository

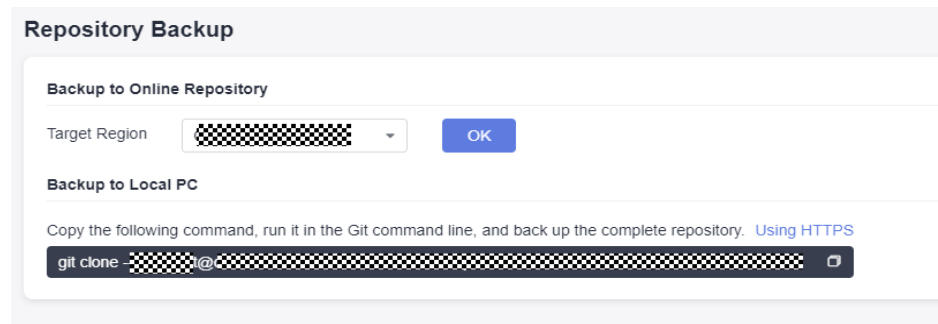
To configure remote backup, choose **Settings > Repository Management > Repository Backup** on the repository details page.

The repository can be backed up in either of the following modes:

- **Back up to Online Repository:** Back up the repository to another region. This mode imports a repository from a region to another region.
- **Back up 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.)

All repository members can view this page. For details about whether a repository member has repository setting permissions, refer to [Permissions](#).



12 Managing Repo Member Permissions

12.1 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.

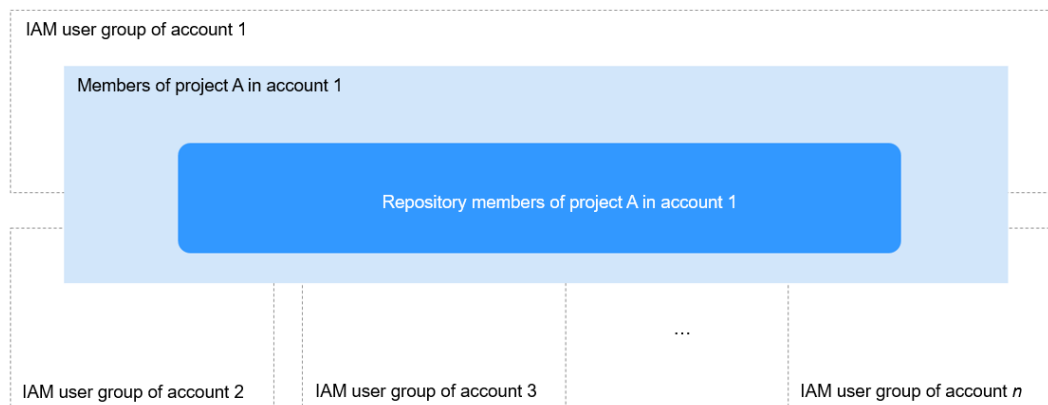


Table 12-1 Mapping between project roles and repository roles

Project Role	Repository Role
Project Manager	Project manager (default)
Product Manager	Product manager (default)
Test Manager	Test manager (default)
Operation Manager	Operation manager (default)


Project Role	Repository Role
System Engineer	System engineer (default)
Committer	Committer (default)
Developer	Developer (default)
Tester	Tester (default)
Participant	Participant (default)
Viewer	Viewer (default)
Custom Role	Custom role (default)

By default, the project creator is both the **project administrator** and repo **administrator (repo owner)**.

12.2 Configuring Project-Level Permissions

- Step 1** Log in to the CodeArts Repo homepage. In the navigation pane on the left, choose **Settings > General > Service Permissions**. The **Service Permissions** page is displayed.
- Step 2** Select the corresponding **Role > CodeArts Repo**, and click **Edit** to configure permissions.

NOTE

1. The project manager and other users with management permissions can modify the default operation permissions of different roles in the project on this page.
2. You can click  in the **Role** column to create a role. The new role name cannot be the same as a system role name. However, the new role can copy the permissions of an existing role. If the permissions of an existing role are not copied to a new role, the new role does not have any permissions. However, you can add permissions for a custom role as required, as shown in [Table 1](#).

----End

Table 12-2 Configuring Project-Level Role Permissions

Role / Permission	Permission	Project Manager	Product Manager	Test manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
Branch	Create	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	C	C	C	B	B	B	C	C	D	C
Code	Commit	B	C	C	C	A	A	A	C	C	D	C
	Download	B	C	C	C	A	A	A	C	C	D	C
Repository group	Create	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	D	D	D	D	D	D	D	D	D	C
	Settings	B	D	D	D	D	D	D	D	D	D	C
Members	Add	B	D	D	D	D	D	D	D	D	D	C
	Edit	B	D	D	D	D	D	D	D	D	D	C
	Delete	B	D	D	D	D	D	D	D	D	D	C
MR	Create	B	C	C	C	B	B	B	C	C	D	C

Role / Permission	Permission	Project Manager	Product Manager	Test manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
	Edit	B	D	D	D	C	B	C	D	D	D	C
	Comment	B	C	C	C	B	B	B	C	C	C	C
	Review	B	D	D	D	B	B	B	D	D	C	C
	Approve	B	D	D	D	C	B	C	D	D	D	C
	Merge	B	D	D	D	C	B	C	D	D	D	C
	Close	B	D	D	D	C	B	C	D	D	D	C
	Re-open	B	D	D	D	C	B	C	D	D	D	C
Repository	Create	B	C	C	C	B	B	B	C	C	D	C
	fork(MR)	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	D	D	D	D	D	D	D	D	D	C
	Set	B	D	D	D	D	D	D	D	D	D	C

Role / Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
Tag	Create	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	C	C	C	C	C	C	C	C	D	C

NOTE

- A: indicates that the role has the permission by default and the permission cannot be removed.
- B: indicates that the role has the permission by default and the permission can be removed.
- C: indicates that the role can have the permission assigned.
- D: indicates that the role cannot have the permission assigned.

12.3 Configuring Repo-Level Permissions

Only modifiable by the project administrator and the owners of parent repository groups and repositories. After confirming that you are an administrator, go to the CodeArts Repo homepage and click the name of the code repository to be set. On the code repository details page that is displayed, click **Member** in the navigation tree to add members to the code repository. Complete the member configuration of the code repository. In the navigation pane, choose **Settings**. On the settings page that is displayed, choose **Security Management > Permissions**. If **Inherit from project** is enabled, the permissions of members in the current role list will be the same as those of the project, and the current permission configuration will be overwritten.


Click  on the right to sync custom roles of the project. By default, custom roles do not have the permission to perform operations on Repo. After the synchronization, you can add permissions listed in [Table 12-3](#) as required.

Table 12-3 Configuring permissions for repo roles

Role / Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
Repository	fork	B	C	B	C	B	B	B	C	C	D	C
	Delete	B	D	D	D	D	D	D	D	D	D	C
	Set	B	D	D	D	D	D	D	D	D	D	C
Code	Commit	B	C	C	C	A	A	A	C	C	D	C
	Download	B	C	C	C	A	A	A	C	C	D	C
Members	Add	B	D	D	D	D	D	D	D	D	D	C
	Edit	B	D	D	D	D	D	D	D	D	D	C
	Delete	B	D	D	D	D	D	D	D	D	D	C
Branch	Create	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	C	C	C	B	B	B	C	C	D	C
Tag	Create	B	C	C	C	B	B	B	C	C	D	C
	Delete	B	C	C	C	C	C	C	C	C	D	C

Role / Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
MR	Create	B	C	C	C	B	B	B	C	C	D	C
	Edit	B	D	D	D	C	B	C	D	D	D	C
	Comment	B	C	C	C	B	B	B	C	C	C	C
	Review	B	D	D	D	B	B	B	D	D	C	C
	Approve	B	D	D	D	C	B	C	D	D	D	C
	Merge	B	D	D	D	C	B	C	D	D	D	C
	Close	B	D	D	D	C	B	C	D	D	D	C
	Re-open	B	D	D	D	C	B	C	D	D	D	C

 **NOTE**

- A: indicates that the role has the permission by default and the permission cannot be removed. B: indicates that the role has the permission by default and the permission can be removed. C: indicates that the role can have the permission assigned. D: indicates that the role cannot have the permission assigned.
- Download and comment permissions are fixed in the public repos' settings, while other permissions mirror those of the private repos

12.4 Syncing Project Members to CodeArts Repo

CodeArts Repo allows you to sync project members to your repo groups and repositories for better project managements. Choose between automatic or manual sync to suit your needs.

Before adding a repo group and repo member, ensure that the member has been added to the project. For details about project member management, see [project-level member management](#).

The repo owner, repo administrator, and custom roles with member management permissions can change repo members. Other users can only view the repo member list.

Auto-Syncing Project Members to a Repository Group or Repository

CodeArts Repo supports syncing project members within one click. After this function is enabled, project members of the selected role can be automatically synced to all repo groups and repos in the project.

Go to the project homepage. Click **Settings**. Choose **Security Management > Member Sync**, click **Sync project member**, and select the target members. The project manager will always be synced regardless of the toggle. Click the refresh button to sync all the current settings.

NOTICE

Automatic sync of updated project members is triggered only when **Sync Project Members** is enabled.

Manually Adding Project Members to a Repository Group or Repository

Go to the repo group or repository homepage, click the **Members** tab, and click **Add Member**. The page for adding members is displayed in either of the following ways:

- On the **Members** tab page, enter a keyword and press **Enter** to search for a member.
- On the **Groups** tab page, select a member group from the drop-down list.

NOTE

- In the member list, all members can be set to any project role and can be removed from the repository.
- If the repository-level member list is empty, the repository does not have members other than the owner. Add project members first.

13 Cloning or Downloading Code Repo to a Local PC

13.1 Differences Between Cloning and Downloading a Repository

Both cloning and downloading the code repository are ways of obtaining the code repo, but their operations and outcomes vary.

1. Clone a repo to a local PC.

Use the SSH key or HTTPS protocol to clone a repo: copy the contents of the entire repo to the local computer and create a local repo. The local repository contains the complete history of code commits, branches, and tags for version controls and modifications. Currently, CodeArts Repo supports cloning code repositories using Git Bash and TortoiseGit clients. Before cloning repos in CodeArts Repo with an SSH key, [configure the SSH key for accessing CodeArts Repo](#).

2. Download a repo.

Download one or more files or folders in the repo to a local computer. This does not contain complete code commit history, branches, or tags. Version control and modification cannot be performed. Currently, CodeArts Repo allows you to download code using a browser.

Therefore, if you need to control and modify the version of the code repo, you need to use the SSH key or HTTPS protocol to clone the code repo. If you only need to obtain one or more files of the code repo, you can use a browser to download the code repo.

NOTICE

- If you want to clone the code repo and develop code locally, go to the home page of the code repo to be cloned from CodeArts Repo, choose **Branches** > **Create Branch**, and create a development branch based on the master branch.
- Currently, CodeArts Repo supports cloning only one code repo at a time. If you want to clone multiple code repo to the local host at a time, you can download multiple repo using Shell or batch processing commands.

13.2 Using the SSH Key to Clone a Repo to a Local PC

Using Git Bash to Clone a Repo to a Local Host

The SSH key is a secure identity authentication method used to access a remote server. Using an SSH key to clone a code repository avoids the username and password each time for higher efficiency of cloning a code repository.

Step 1 [Access CodeArts Repo homepage](#).

Step 2 Go to the repo homepage of the code to be cloned, create a personal branch, click **Clone/Download**, and copy the SSH address.

Step 3 On the local Git Bash client, run the following command to access the address of the code repository to be cloned. This command indicates that the cloned code repository will be cloned to the Repo folder in drive D. You can change the address as needed.

```
cd D:/Repo
```

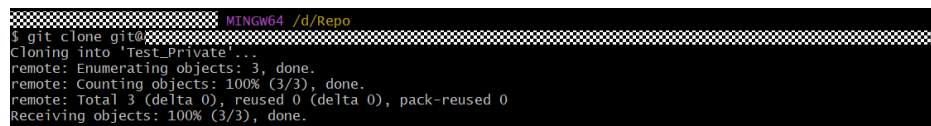
Step 4 Run the following command to clone the repository to the directory:

```
SSH address of git clone code repo
```

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

If [the following figure](#) is displayed, the repo is cloned successfully.

Figure 13-1 Successful cloning of the repository using the SSH key



```
MTNGW64 /d/Repo
$ git clone git@...:
Cloning into 'Test_Private'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

If Git Bash reports error **git@test.com: Permission denied.fatal: Could not read from remote repository.Please make sure you have the correct access rights and the repository exists.** in step 3, the SSH key for accessing Repo has not been configured. Configure the SSH key first. For details, see [Configuring an SSH Private Key](#).

----End

Use the SSH key to clone the code repo to the local host in TortoiseGit.

- Step 1** [Access CodeArts Repo homepage](#).
 - Step 2** Go to the home page of the code repo to be cloned, click **Clone/Download**, and copy the SSH address.
 - 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 displayed dialog box, paste the SSH address copied in [step 2](#) to the URL text box, select **Load PuTTY key**, and select the private key file generated during TortoiseGit installation.
 - Step 5** Click **OK**. If you clone the code repository on the TortoiseGit client for the first time, the system asks you whether to trust the remote repository. Click **Yes**.
- End

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

Using Git Bash to Clone a Repo to a Local Host

- Step 1** [Access CodeArts Repo homepage](#).
- Step 2** Go to the home page of the code repo to be cloned, click **Clone/Download**, and copy the HTTP address.
- Step 3** On the local Git Bash client, run the **cd D:/Repo** command to go to the address of the code repository to be cloned. The following command indicates that the cloned code repository will be cloned to the Repo folder in drive D.
- Step 4** Run the following command to clone the repository to the directory:

```
git clone code repo HTTPS link
```

If you clone code repo for the first time, you need to enter the username and password. There are two types of usernames and passwords. Select one of the following methods based on your configuration:

- To view the username and password, log in to and go to the Repo code repo list page, click the nickname in the upper right corner, and choose **This Account Settings > Repo > HTTPS Password** to obtain your username and password. If you have forgotten the password, you can reset the HTTPS password.
- Token username and password. The token username is **private-token**, and the token password is the configured token. If the token is lost or forgotten, generate a new token by referring to [Configuring an Access Token](#).

If [the following figure](#) is displayed, the repo is cloned successfully. If code repo fails to be cloned, rectify the fault based on the [description](#).

Figure 13-2 Successful cloning of the repository using the HTTPS

```
MINGW64 /d/Repo
$ git clone https://68b2607/Test_Private.git
Cloning into 'Test_Private'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 589 bytes | 98.00 KiB/s, done.
```

----End

NOTE

- When **step 3** is executed, Git Bash reports error **fatal: unable to access 'https://test.com/Test_Private.git/': SSL certificate problem: unable to get local issuer certificate**. Before running the git clone command, run the following command so that Git does not verify the SSL certificate when cloning the code repository using HTTPS:
`git config --global http.sslVerify false`
- During the execution of **step 3**, Git Bash reports error **fatal: unable to access 'https://test.com/Remote_Test.git/': Failed to connect to test.com port 443 after 21161 ms: Couldn't connect to server**, indicating that the network is disconnected. Contact your local network administrator.
- When **step 3** is executed, Git Bash reports error **fatal: unable to access 'https://xxx.git/': Recv failure: Connection was reset**, indicating that the domain name resolution is incorrect. For details about the solution, see FAQs.
- When **step 3** is executed, Git Bash reports error **fatal: destination path 'Test_Private' already exists and is not an empty directory**, indicating that the **Test_Private** code repository has been cloned to this path and is not empty. Solution: Switch to a new empty directory and execute **step 3** again.
- When **step 3** is executed, Git Bash reports error **fatal: Authentication failed for 'https://xxx.git/'**, indicating that your password is incorrect. You can log in to the Repo code repo list page, click the nickname in the upper right corner, and choose **This Account Settings > Repo > HTTPS Password**. Obtain your username and password. If you have forgotten the password, you can reset the HTTPS password.
- Error "The requested URL returned error: 401" Is Reported When HTTPS Is Used to Clone Code in CentOS This is because of the Git version dis-match.
- If you want to embed the access token into the HTTPS download link, run the following command in **step 3**: In the preceding command, password indicates your configured token. If the token is lost or forgotten, you can generate a new token by referring to **Configuring an Access Token**. *{project_name}* indicates the project name, and *{repository_name}* indicates the name of the code repository to be cloned.
`git clone https://private-token:password@codehub.test.com/{project_name}/{repository_name}.git`

13.4 Using a Browser to Download Code Package to a Local PC

CodeArts Repo not only supports code repository cloning, but also allows you to package and download the repo code a local PC.

Step 1 [Access CodeArts Repo homepage](#).

Step 2 Go to the homepage of the repository to be cloned, switch to the branch to be downloaded, and click **Clone/Download**.

Step 3 In the dialog box that is displayed, click the required code package type to download it.

----End


 **NOTE**

- After the branch is switched, the downloaded package is the content of the specified branch.
- 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.

14 Uploading Code Files to CodeArts Repo

14.1 Editing and Creating a Merge Request

Go to the repo homepage, click **Code** to go to the code homepage, and create a branch based on the code branch to be merged. Select the branch to be modified, edit the code, and create a merge request.

- To add a code file, click **Create** to create a code file or upload one from your local PC. After modifying a branch, click **Create Merge Request** on the right of **Code**, select the branch to be merged, and click **Next**. The **Create Merge Request** page is displayed, the title is mandatory..
- To modify a code file online, click the file name on the **Code** page and click  . Edit and save the file. Click **Create Merge Request**. You can view the file difference comparison and commit record of the two branches in the lower part of the page.

NOTICE

- A branch name cannot start with hyphen (-), period (.) refs/heads/refs/remotes/ nor end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " |
-

14.2 Creating a Branch and Developing Code in Git Bash

- Step 1** Go to a local repo directory and open Git Bash. Run the following command to create a branch **feature1** based on the master branch and switch to the **feature1** branch:

```
git checkout -b feature1
```

Step 2 The following steps simulate writing the string **hello CR** to a file named **hello_cr.txt**.

```
echo 'hello CR' > hello_cr.txt
```

Step 3 Add all modified files in the current directory to the temporary storage area of Git and prepare to commit them to the version library.

```
git add .
```

Step 4 Commit the modified code to the local code repo and add a piece of commit information.

```
git commit -m 'hello cr'
```

Step 5 View the details of the latest commit.

```
git log -1
```

Step 6 Run the following command to push the local branch **feature1** to the origin branch of your remote repo and establish a tracing relationship between the local branch and the remote branch:

```
git push --set-upstream origin feature1
```


NOTE

- If **connect to host *****.com port 22: Connection timed out** is displayed in step 6, your network is restricted and you cannot access CodeArts Repo. Contact your local network administrator.
- If you add the local path to the repository of CodeArts Repo after creating a commit, you cannot change the path of the commit code. You can only delete the file locally or roll back the commit to forcibly commit the code.
- 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 7 Go to the homepage of the repository where a merge request is to be created, choose **Merge Requests > Create MR**, and select the source and target branches. In the lower part of the page for creating a merge request, you can view the details of the files in the two branches and the commit records of the branch to be merged.

----End

NOTICE

- To commit local code to CodeArts Repo, you need to run the **git push** command. The **git commit** command only saves the modifications in the local repo. Each commit operation generates a new commit record, recording the modified content, author, and time. The commit operation only saves the code changes to the local repo and does not sync them to the remote repo.
- When you push code to CodeArts Repo, the message "You are not allowed to push code to protected branches on this project" is displayed. This is because the branch is protected and you do not have the permission to push code to the branch. Solution: Go to the repository details page as the repository owner or project administrator, choose **Settings > Policy Settings > Protected Branches**, and click  to cancel the protection for the branch.
- The message "src refs spec master does not match any" is displayed during code push. The reason is that you do not run the **git add** and **git commit** commands to add files from the workspace to the temporary storage area in sequence. Solution: Before running the **git push** command, run the **git add** and **git commit** commands to submit the modified file to the temporary storage area, and then run the **push** command to push the file to the cloud repository.
- When you push code to CodeArts Repo, the message "error: failed to push some refs to 'https://codehub'" is displayed. The code in the CodeArts Repo is inconsistent with that in the local repo. As a result, the code commit is rejected. Solution: Run the **git pull** command to pull the code from the remote repository of CodeArts Repo, merge the code with the local repository, and run the **git push** command to push the code to CodeArts Repo.
- The **git pull** command fails to pull code, and the message "**Merge branch 'master' of https://codehub/testMaven Please enter a commit message to explain why this merge is necessary**" is displayed. The code in CodeArts Repo is different from the code in your local repository. Therefore, when **git pull** is executed, the remote code will be merged to the local code. The dialog box displayed asks you to confirm the merge and enter a commit message. Solution: See [What Can I Do If Code Fails to Be Pulled Using git pull with Error Message "Merge branch'master' of https://xx.com Please Enter a commit"?](#)
- If the error message "unable to auto-detect email address" is displayed when you perform step 4, it means the username and email address are not set. You can run the following command to configure your personal information:

```
git config --global user.name {your name}
git config --global user.email {your email address}
```
- If the error message " **'origin on' does not appear to be a git repository...**" is displayed in step 6, it means the repository name origin does not exist remotely. For details about the solution, see [The error message "origin" does not appear to be a git repository... is displayed when the git push command is executed.](#)
- If the error message "**Not a git repository**" is displayed when you perform step 3, the cause is that you are not in the current code repository directory. In this case, run the **cd** command to go to the repository directory.
- If the error message "failed to push some refs to '...git'" is reported when you commit a merge request, see [Resolving a Merge Request Conflict.](#)

- In step 6, the message "**Connection reset by test port 22 fatal: Could not read from remote repository.**" is displayed, it means the network is unstable and the request is reset. If this problem occurs occasionally, the network may be faulty.
-

14.3 Committing Code in Eclipse and Creating a Merge Request

If EGit is installed on your local Eclipse, you can commit the local Git repository code to CodeArts Repo. CodeArts Repo supports only Eclipse 4.4 and later versions.

NOTE

- 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. After remote commit is complete.
- 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.

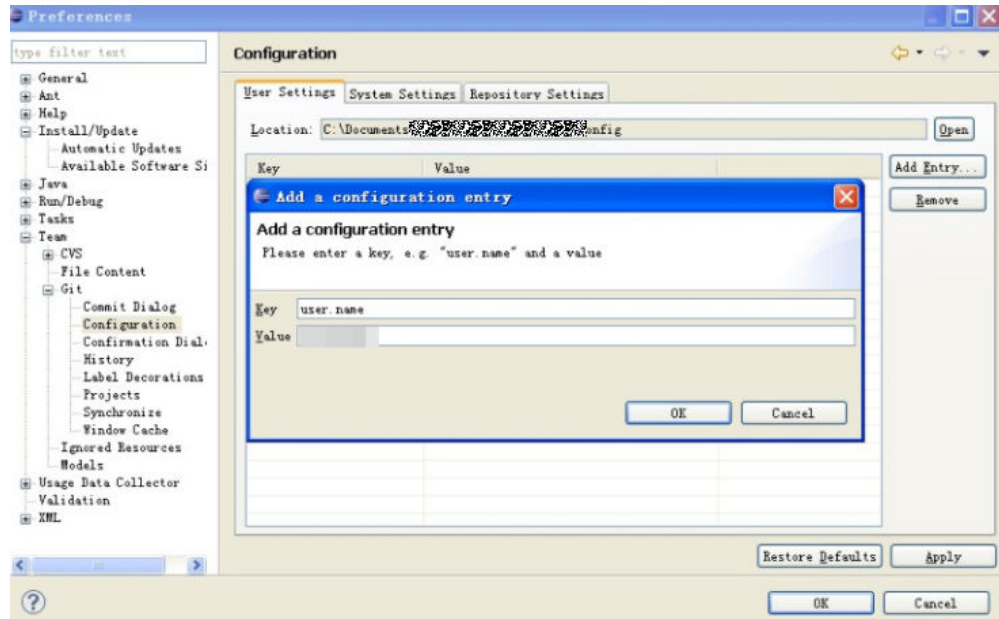
Step 1: Installing EGit on Eclipse

Perform the following steps to install Eclipse 4.4:

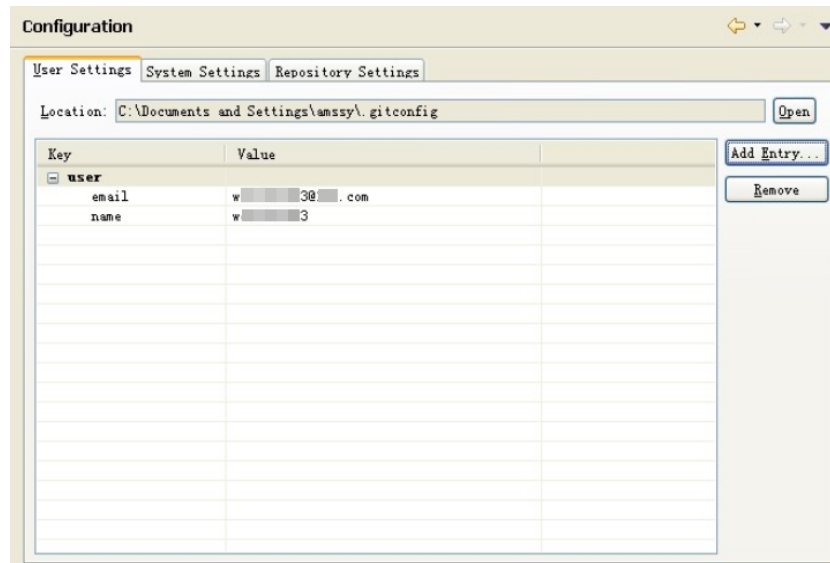
On the Eclipse toolbar, choose **Help > Install New Software....** In the **Install** window that is displayed, click **Add...**, enter **EGit** in **Name** and **EGit plug-in address** in **Location**, click **OK**, and click **Next >** until the installation is complete. After the installation is complete, restart Eclipse.

Step 2: Configuring EGit on Eclipse

1. On the Eclipse toolbar, choose **Window > Preferences > Team > Git > Configuration** and enter the **User Settings** information.

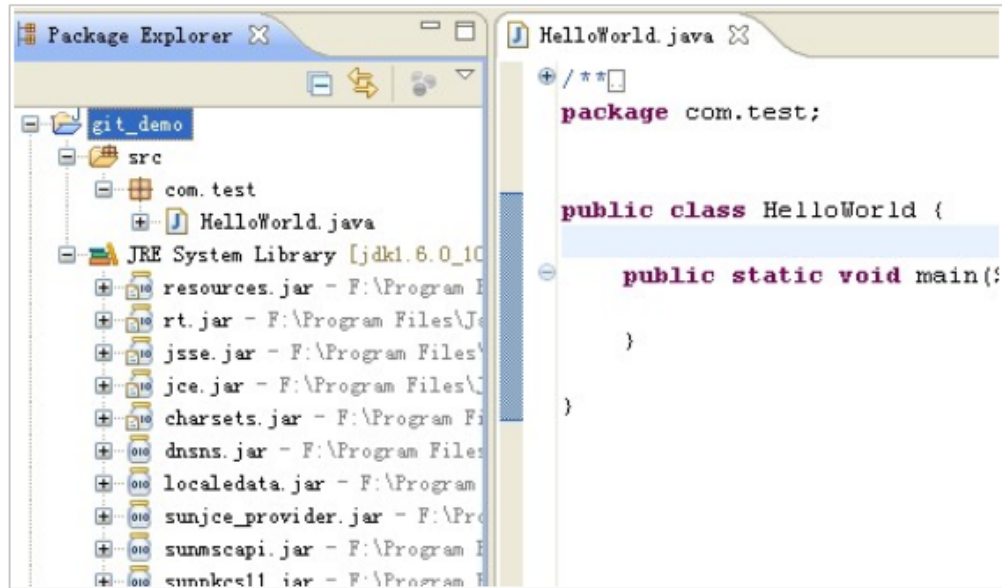


2. Click **OK**.
email indicates the bound email address. If the username is not set previously, set it in this step.

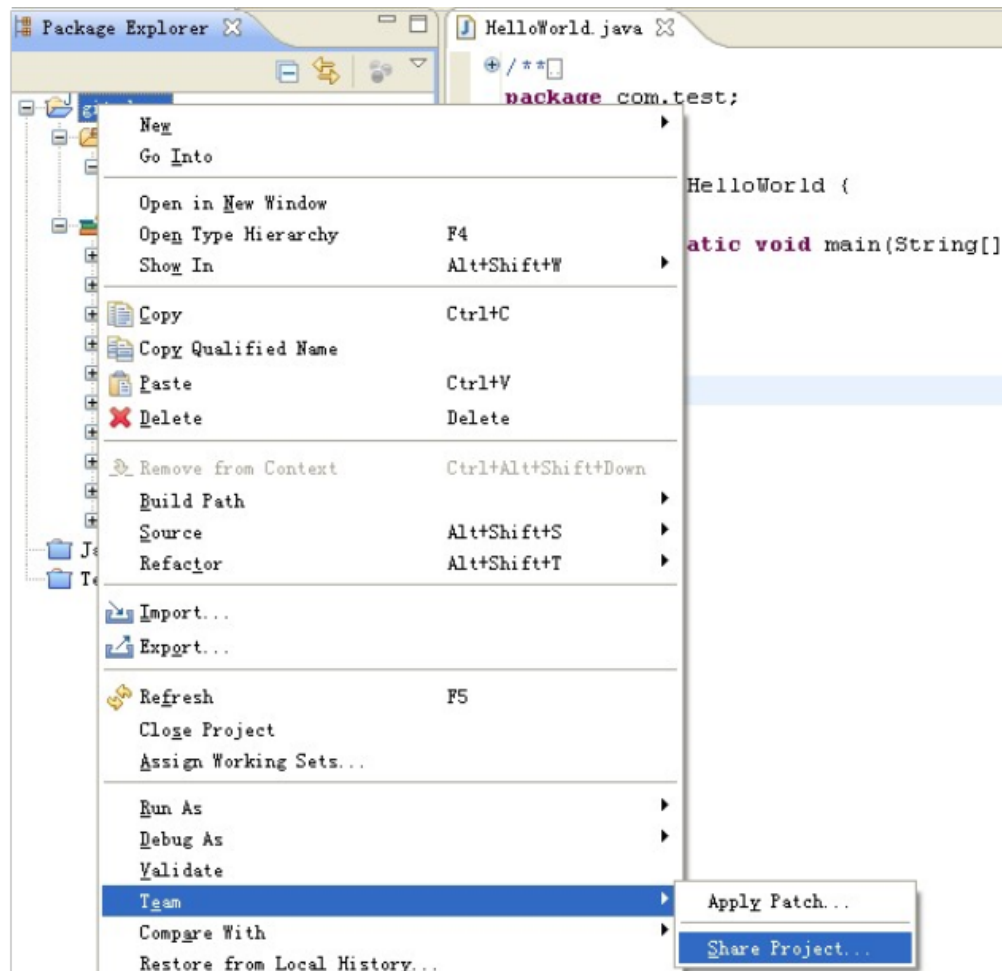


Step 3: Creating a Project and Committing Code to the Local Git Repository

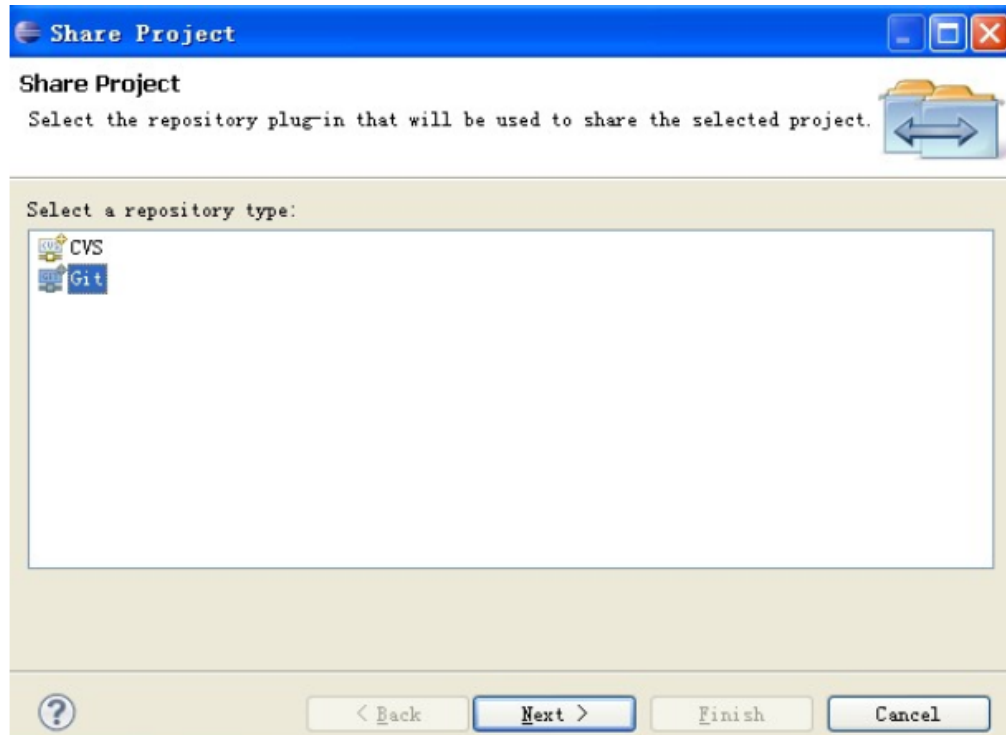
1. Create the **git_demo** project and the **HelloWorld.java** class.



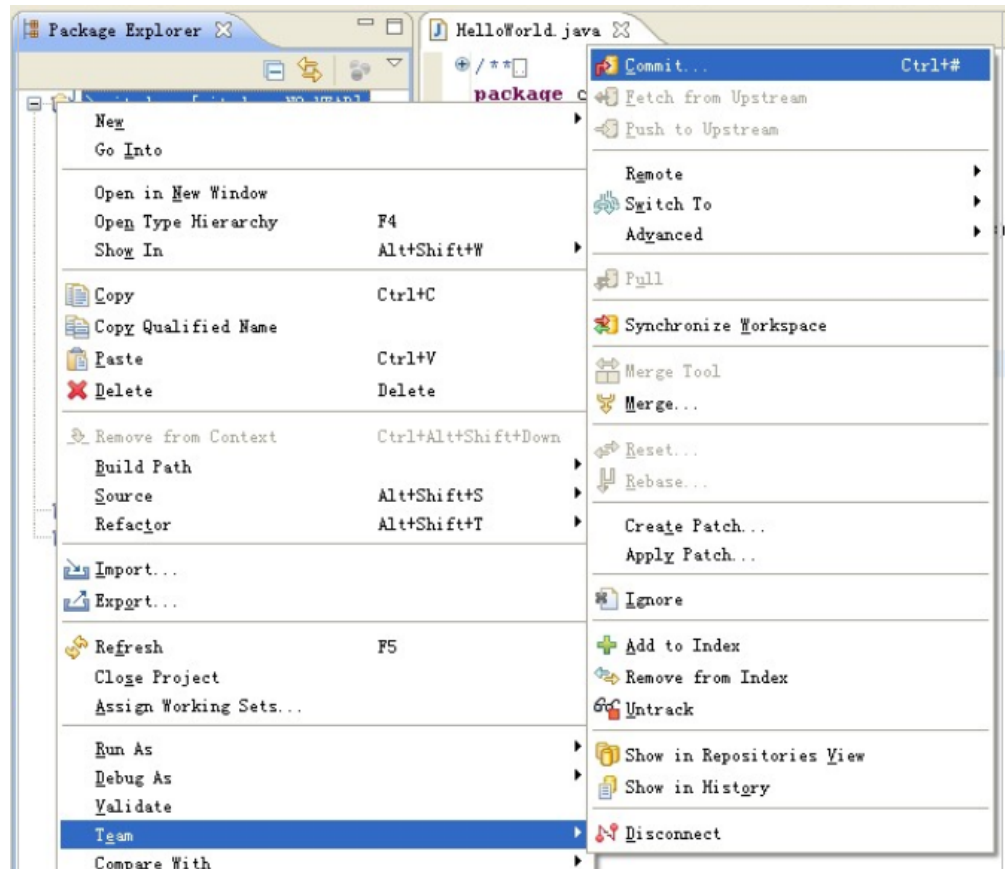
2. Share the **git_demo** project with the local repository.



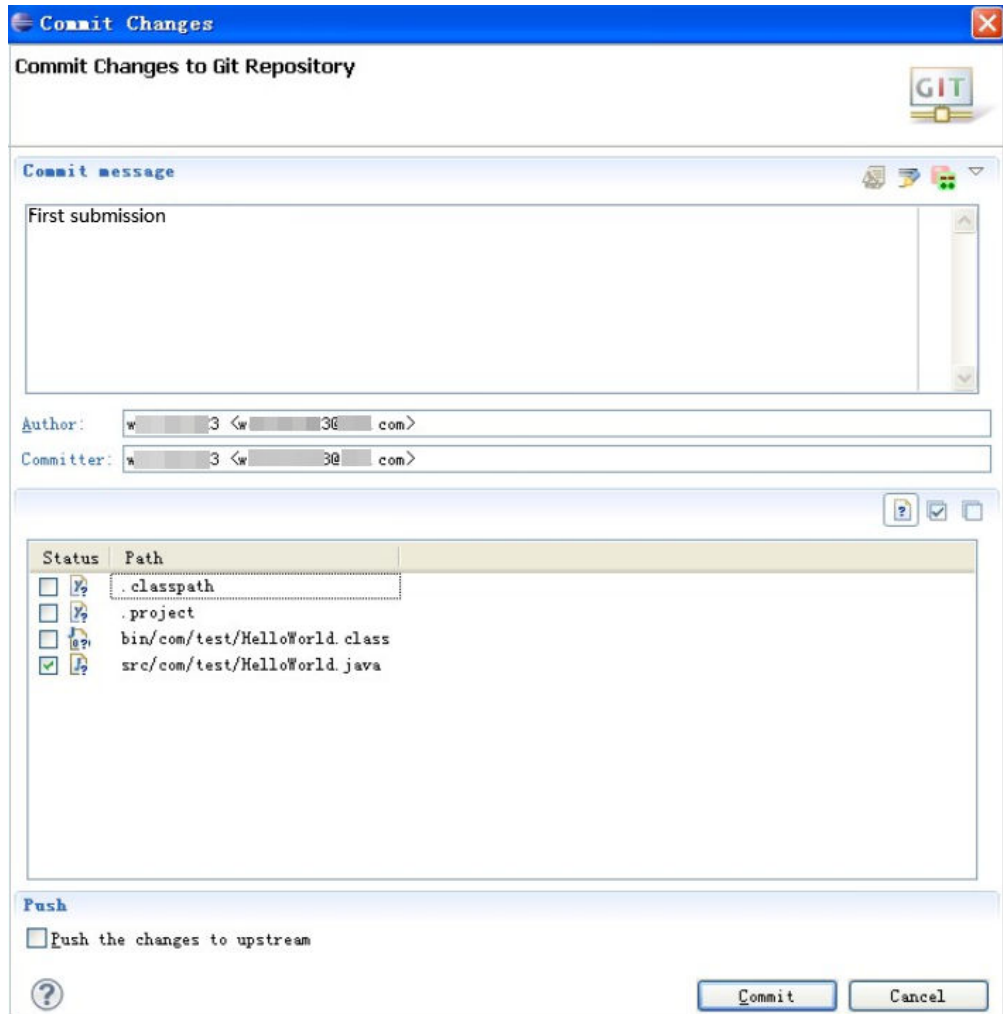
3. In the **Share Project** window displayed, select **Git**.



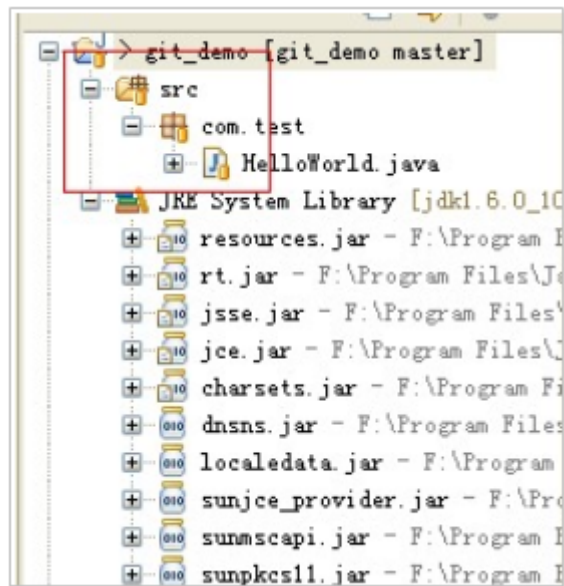
4. Click **Next >**. In the **Configure Git Repository** dialog box, select **Use or create repository in parent folder of project** and click **Create Repository**.
5. Click **Create Repository** to create a Git repository.
The directory is in the **untracked** status, indicated by a question mark (?).
Choose **Team > Commit...** to commit code to the local repository.



6. In the **Commit Changes** dialog box displayed, set the commit message.

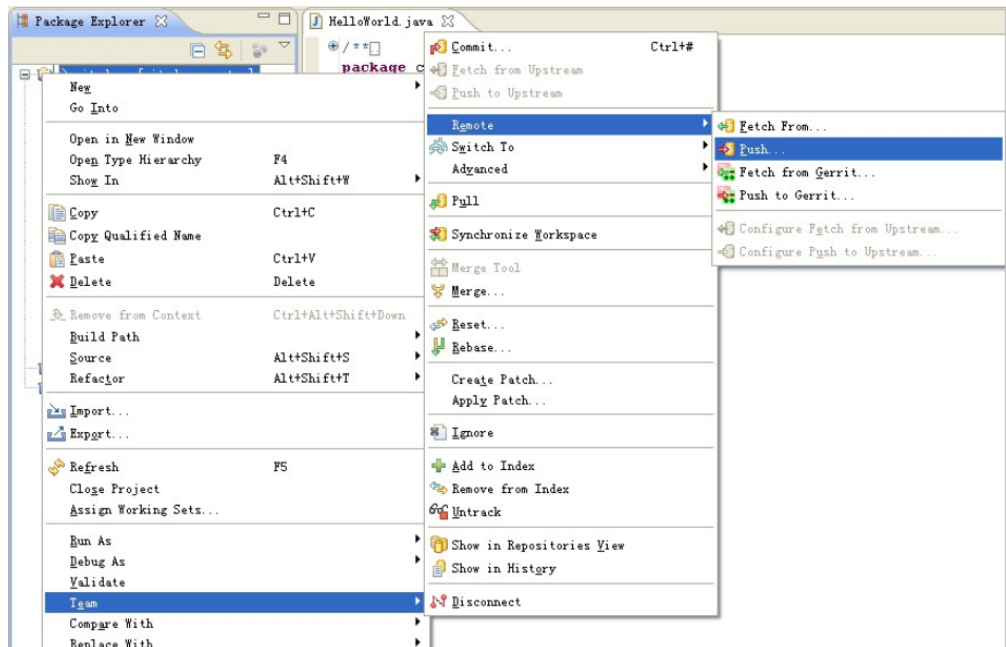


7. Click **Commit** to commit the code to the local repository.

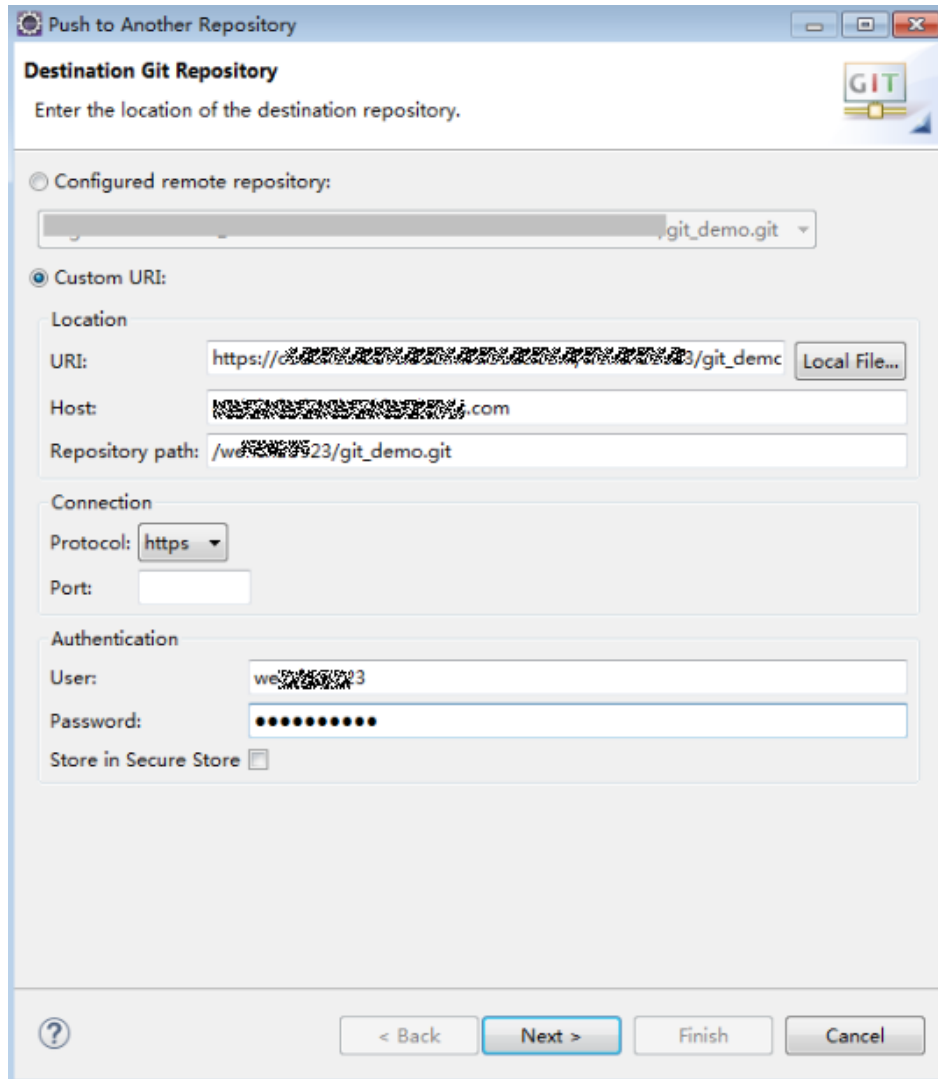


Step 4: Committing Code in the Local Repository to the Remote Git Repository

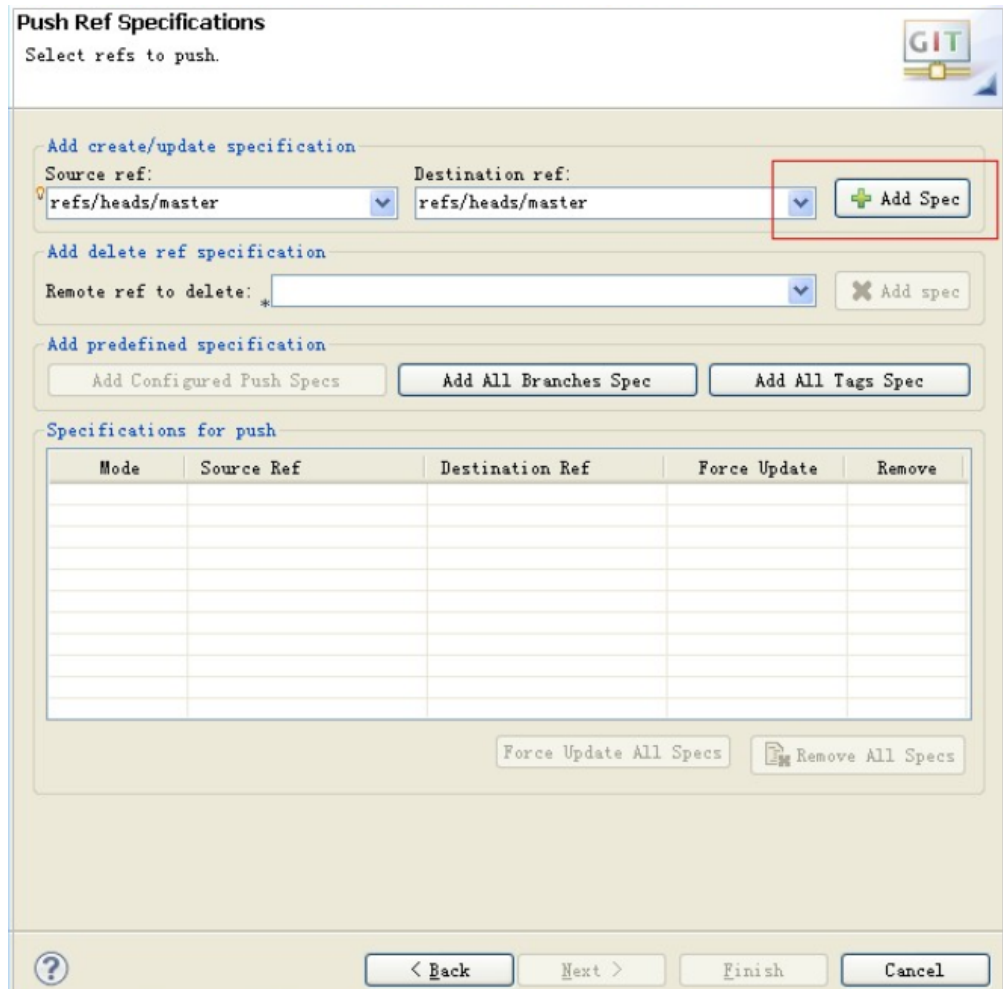
1. Create a repository in CodeArts Repo. For details, see [Overview](#).
Go to the repository details page and copy the repository URL.
2. Choose **Team > Remote > Push...** to push the code to the remote repository.



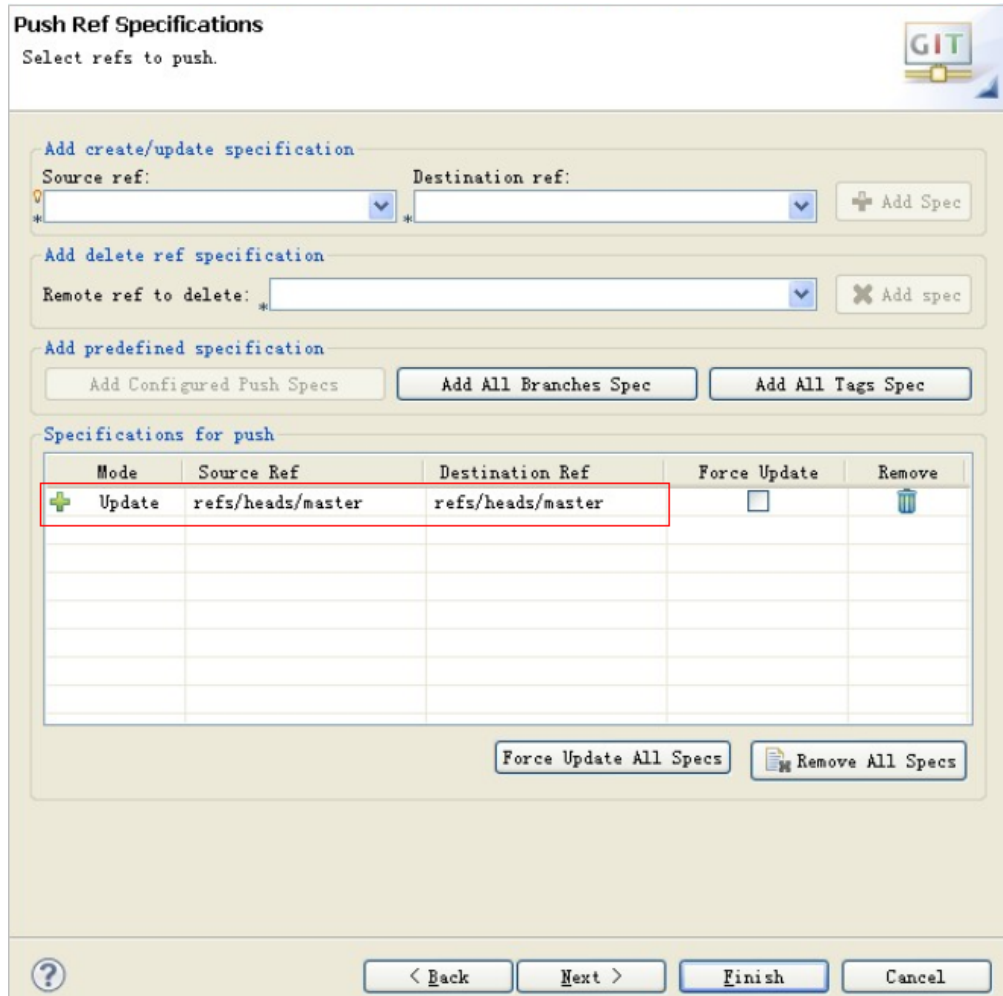
3. In the **Push to Another Repository** dialog box, set the parameters.



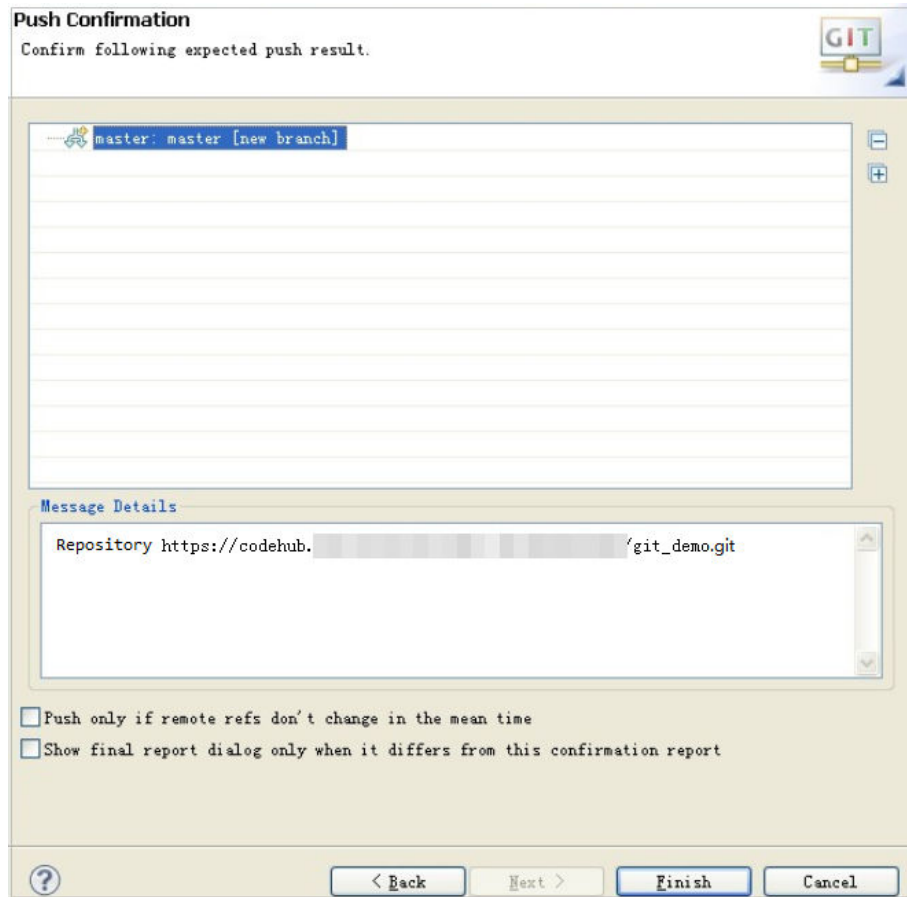
4. Click **Next**. The **Push Ref Specifications** dialog box is displayed.



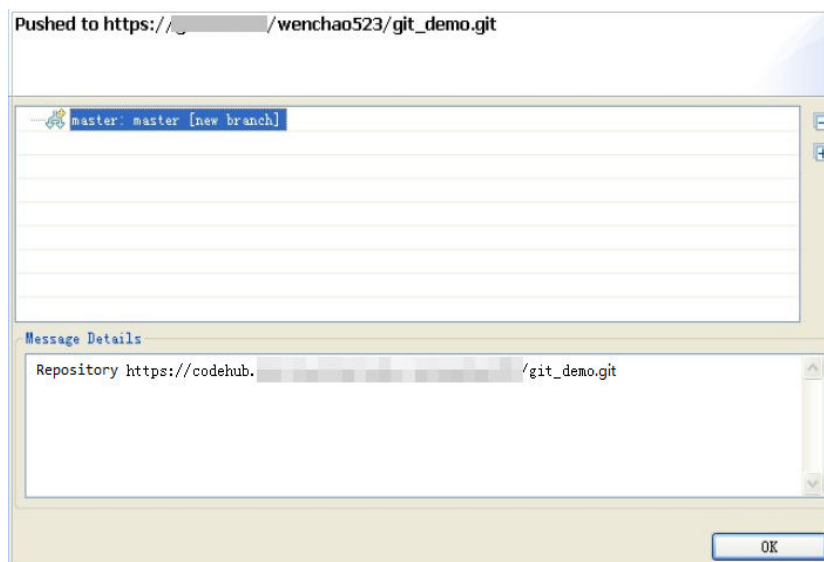
5. Click **Add Spec**.



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.

 NOTE

When Eclipse connects to the repository of CodeArts Repo using HTTPS, the message "Transport Error: cannot get remote repository refs. XXX.git: cannot open git-upload-pack" is displayed. This is because the EGit plug-in configuration in Eclipse is incorrect. Solution: Right-click in Eclipse and choose **Windows > Preferences > Team > Git > Configuration > User Settings** from the shortcut menu. Click **Add Entry**, set **Key** to **http.sslVerify**, and set **Value** to **false**.

Step 5: Creating a Merge Request in CodeArts Repo

Go to the homepage of the repository where a merge request is to be created, choose **Merge Requests > Create MR**, and select the source and target branches. 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.

14.4 Using git-crypt to Transmit Sensitive Data on the Git Client

About git-crypt

git-crypt is a third-party open-source software that can transparently encrypt and decrypt files in the Git repository. The **git-crypt** command can be used to encrypt and store specified files and file types. Developers can store encrypted files (e.g. confidential information or sensitive data) in the same repository as shareable code, and the repository can be pulled and pushed just like a normal repository, with the contents of the encrypted files visible only to those who have the corresponding file key, but with no restriction on participants' ability to read or write to unencrypted files.

Using Key Pairs for Encryption and Decryption on Windows

Step 1 [Download and install the latest Git client for Windows](#), download the latest [git-crypt for Windows](#), and save the downloaded **.exe file** to the **cmd** folder in the Git installation directory.

Step 2 Run the following commands to generate a key pair locally:

1. Open **Git Bash** and go to the local repository.
2. Run the following command to create the **.git-crypt** folder in the Git repository. The folder contains the key and configuration file required for encrypting the file.

```
git-crypt init
```
3. Run the following command to export the key file to the **C:/test** directory and name the file **KeyFile**:

```
git-crypt export-key /c/test/keyfile
```
4. After the preceding steps are performed, you can go to the path of the exported key file to check whether the key is successfully generated. The computer containing the key file can decrypt the corresponding encrypted file.

Step 3 Run the following command to configure the encryption range 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
```

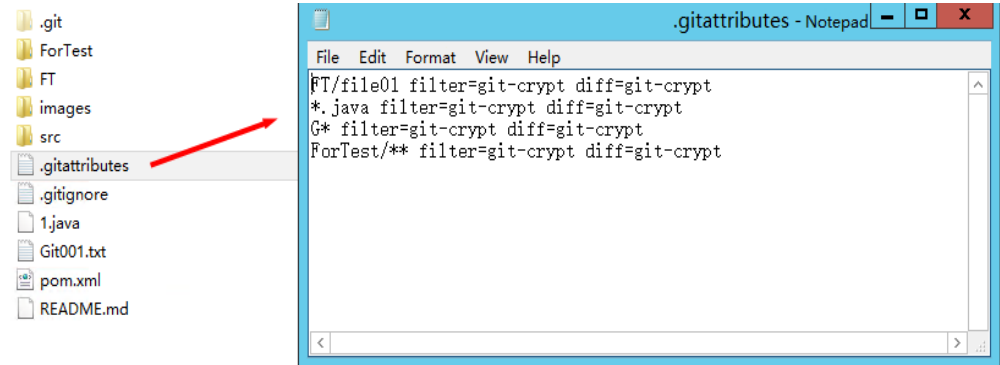
Four examples are as follows:

```
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.
```



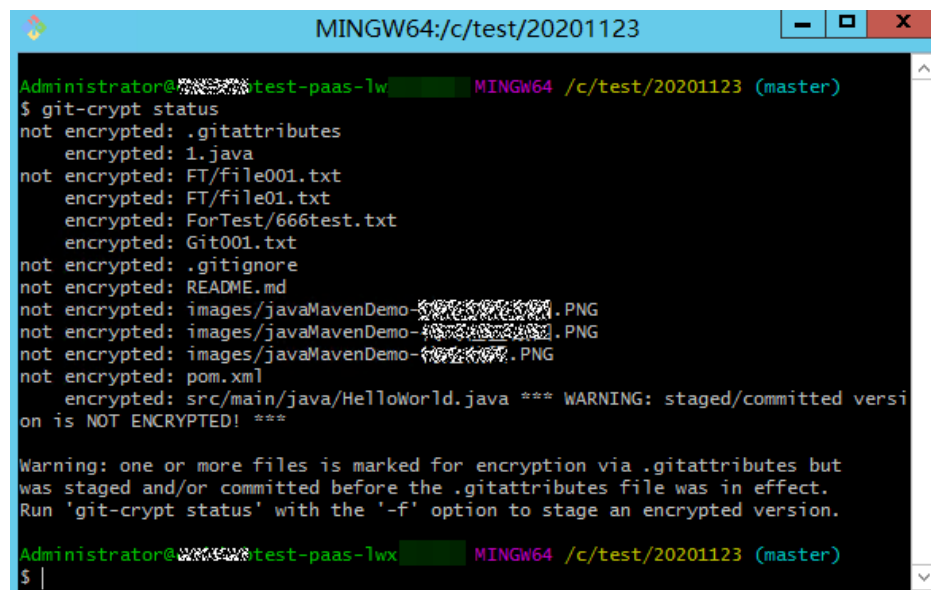
NOTE

- 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 4 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.

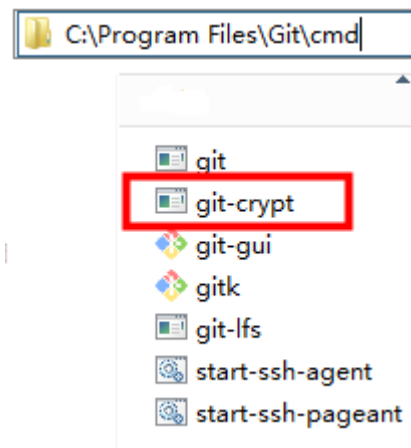
NOTE

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.

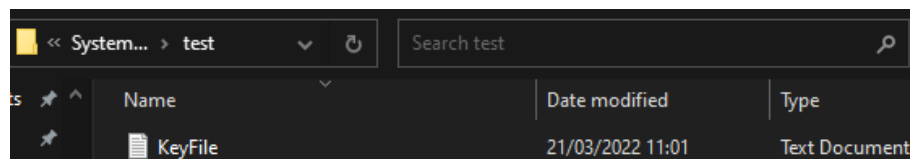
The **-f** (enforce) option is risky in getting teams to work together, so use it with caution.

Step 5 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.
```

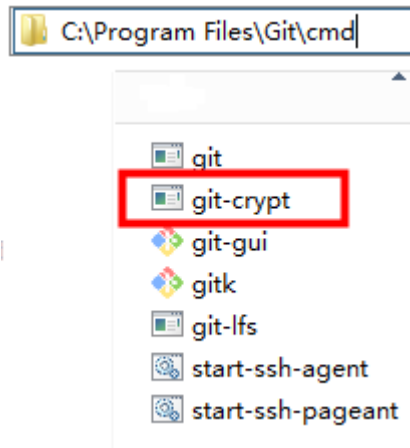
----End

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.



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.

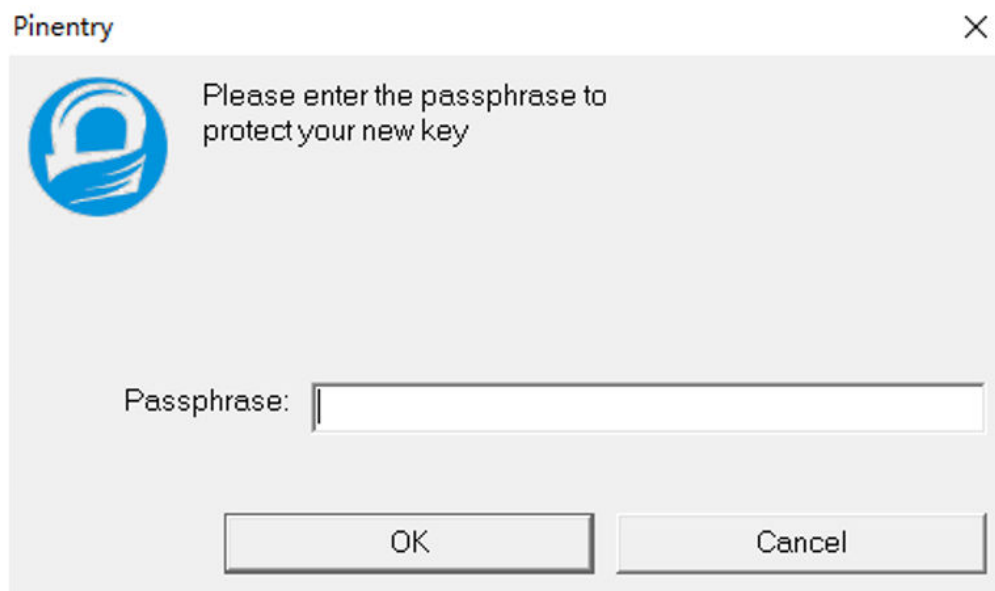
gpg: directory '/c/Users/Administrator/.gnupg' created
gpg: keybox '/c/Users/Administrator/.gnupg/pubring.kbx' created
Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

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. For information security, you are advised to set a new password.

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-lw: MINGW64 /c/dev/test
$ cd 20201124

Administrator@codehubtest-paas-lw: MINGW64 /c/dev/test/20201124 (master)
$ git-crypt init
Generating key...

Administrator@codehubtest-paas-lw: 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-lw: 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-lw: 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
```

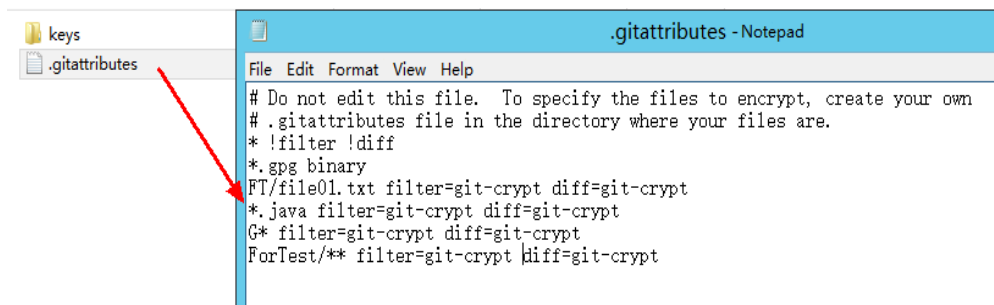
Four examples are as follows:

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
```

```
Administrator@codehubtest-paas-1w... MINGW64 /c/dev/test/20201124 (master)
$ git-crypt status
not encrypted: .gitattributes
  encrypted: 1.java
  encrypted: GitTest666.txt
not encrypted: .git-crypt/.gitattributes
not encrypted: .git-crypt/keys/default/0/0DD
D. pgp
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@codehubtest-paas-1w... MINGW64 /c/dev/test/20201124 (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 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-1w [redacted] 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 [redacted]
uid   [ultimate] gpgTest <gpgTest@huahua.com>
sub   rsa3072 2020-11-24 [E] [expires: 2022-11-24]

Administrator@codehubtest-paas-1w [redacted] 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
```

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 machine, download and install git-crypt and GPG based on Git.
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 [step 8](#). 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. You will be prompted to enter the key password during the import.

```
GPG --import /c/key
```

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 [redacted] MINGW64 /c/dev001/20201124 (master)
$ gpg --import /c/Key
gpg: /c/Users/Administrator/.gnupg/trustdb.gpg: trustdb created
gpg: key 3E38 [redacted] EOAD: public key "gpgTest <gpgTest@huahua.com>" imported
gpg: key 3E38 [redacted] 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 [redacted] 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. The combination of **CodeArts Repo**, **Git**, and **git-crypt** can be used 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

Install 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.

[Download 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.

[Download the source code.](#)

```
./configure  
make  
make install
```

- Install git-crypt using the Debian package.

[Download the source code.](#)

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
```

14.5 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 **Activities** tab page of a repo or on the **History** tab page of a repo file list.

Viewing the Commit History on the Repo Activities Page

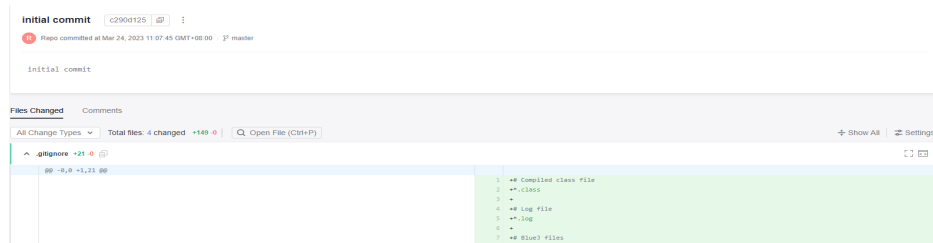
Go to the homepage of the code repository to be viewed and click **Activities** to view all activities of the repositories. To view the activities within a specified period, select a period on the right. To view the activities of a repo member, select the repo member from the drop-down list box on the right.

- **All:** If no time range or member is selected, the operation records of all members in the repo are displayed.
- **Push:** If no time range or member is selected, the push records of all members in the repository are displayed, for example, code push, branch creation, and branch deletion.
- **Merge Request:** If no time range or member is selected, the merge request records of all members in the repository are displayed. You can click the sequence number of a merge request to view details, such as creating, closing, re-opening, and merging MRs.
- **Review:** This tab displays all review comments of the repository. You can click the commit ID to view details such as adding or deleting comments.
- **Member:** This tab displays the management records of all members in the repository, for example, adding or removing members and editing member permissions.

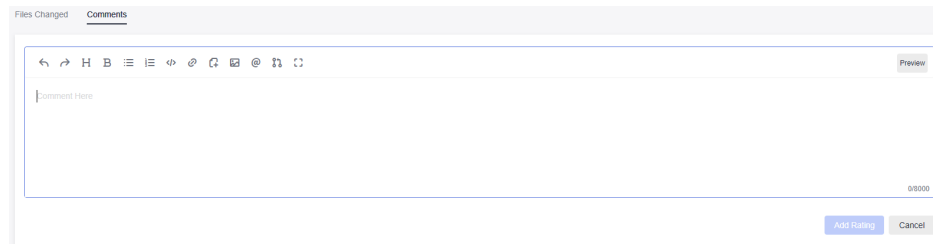
NOTE

- 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.

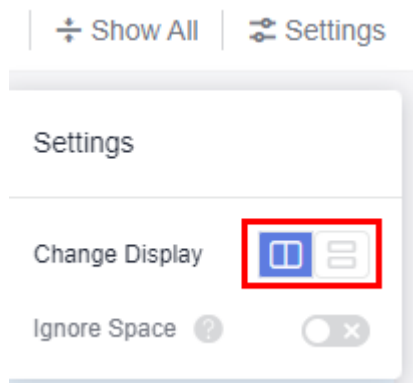
You can view the commit history on the **History** tab page of the **Files** or **Activities** tab. You can click a commit record to view the committer, commit number, parent node, number of comments, and code change comparisons.



You can comment on a commit or reply to 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.



15 Developing a Workflow

15.1 Workflow Overview

Git workflows can be used for versioning, project process management, and collaborative development, improving project management and collaborative development capabilities. It is necessary to pick your own Git workflow based on your team requirements and workflow for continuous integration, continuous delivery, and fast iteration.

There are several common Git workflows. The following sections describe their processes, advantages, disadvantages, and some usage tips.

- [Centralized workflow](#)
- [Feature branch workflow](#)

15.2 Centralized Workflow

Centralized workflows are suitable for small teams that have just transformed from SVN to Git. Centralized workflows are developed in a central repo. Developers clone the repository from the central repo and push the code to the central repo after the development.

Advantages

- Central management. In a centralized workflow, all code repositories are stored in a central repository, facilitating code management and maintenance.
- Efficient collaboration. Team members can share and collaborate with each other through the central repository.
- Secure and reliable. The central repository can be backed up and restored for code security and reliability.

Disadvantages

- Dependency on the central repo: All code depends on the central repo. If the central repo is faulty, the development work of the entire team will be affected.

- Code conflicts: All code is managed in the central repository. Conflicts may occur when team members modify code. Therefore, you need to manually resolve conflicts to ensure code correctness.
- Permission management: All code is managed in the central repository. Therefore, the permissions of team members need to be managed to ensure code security and reliability.
- Not suitable for large project teams: For large project teams, centralized workflows may make it difficult to manage and maintain the central repository, affecting development efficiency and code quality.

Centralized Workflow Process

- Step 1** Create a code repository. In CodeArts Repo, you can [create a custom repository](#), [create a repository using a template](#), and [fork an existing repository](#). You can also [import a local repository](#), [import a Git repository](#), or [import an SVN repository](#).
 - Step 2** Clone a code repository. Currently, CodeArts Repo supports code cloning from CodeArts Repo to a local computer by using [an SSH key](#) and [HTTPS](#).
 - Step 3** [Create a local branch and compile code](#) or [create a branch online and compile code](#).
 - Step 4** Commit the modified code file to the cache. Currently, Repo supports code commit with [Git Bash](#) or [Eclipse](#).
 - Step 5** Developers [create a merge request](#).
 - Step 6** [Resolve review comments](#).
 - Step 7** Committers [merge the MR](#).
- End

15.3 Feature Branch Workflow

This function allows teams to independently develop new functions or fix bugs without affecting the master branch (usually **master** or **main**). The core of this workflow is to use branches to manage different development phases, improving team collaboration efficiency and code quality.

Advantages

- Parallel development: Team members can independently develop new functions or fix problems without affecting the master branch.
- Code isolation: Each branch is independent. That is, the change of a branch does not affect other branches, reducing the risk of code conflicts.
- Fast iteration: By creating and merging branches, teams can quickly iterate new functions or fixes, accelerating software development.
- Easy management: Branches can be created and merged using the Git command line tool or GUI, making versioning more intuitive and convenient.
- Code review: Code review before merging branches helps ensure code quality and knowledge sharing among team members.

- Rollback and cancellation: Code can be quickly restored to the previous state when a problem occurs during development.

Disadvantages

- Complex merge: When multiple function branches need to be merged back to the master branch, complex merge conflicts may occur. In this case, you need to resolve the merge conflicts.
- Resource consumption: Maintaining multiple function branches may consume more computing resources and storage space.
- Branch management: Effective branch management policies are required to prevent too many branches or disordered relationships between branches.

Workflow

- Step 1** Create a code repository. In CodeArts Repo, you can [create a custom repository](#), [create a repository using a template](#), and [fork an existing repository](#). You can also [import a local repository](#), [import a Git repository](#), or [import an SVN repository](#).
 - Step 2** [Create a local branch and compile code](#) or [create a branch online and compile code](#).
 - Step 3** Commit the modified code file to the cache. Currently, Repo supports code commit with [Git Bash](#) or [Eclipse](#).
 - Step 4** Developers [create a merge request](#).
 - Step 5** [Resolve review comments](#).
 - Step 6** Committers [merge the MR](#).
- End

16 Creating and Configuring a CodeArts Project

16.1 Configuring Project-Level Commit Rules

On the CodeArts Repo homepage, go to the project homepage and choose **Settings > Policy Settings > Repository Settings**. For details about how to set the parameters, see [Table 16-1](#).

Table 16-1 Project-level parameters

Parameter	Description
Force inherit	Optional. Once selected, all repo groups and code repos in the project use the following settings and cannot be changed. Exercise caution when selecting this option.
Do not fork a repository	Optional. Once selected, no one can fork the repo in the project.
Pre-merge	Optional. Once this is selected, the server automatically generates the pre-merge code of the MR. 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.

Parameter	Description
Branch Name Rule	Optional. All branch names must match the regular expression with max. 500 characters. If this field is left blank, any branch name is allowed. The rules must meet the following tag naming rules: <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .
Tag Name Rule	Optional. All tag names must match the regular expression specified by this parameter. If this field is left blank, any tag name is allowed. The basic tag naming rules must be met. <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .

Configuring Project-Level Protected Branch Rules

CodeArts Repo makes code branches more secure by preventing anyone other than the administrator from committing code, preventing anyone from forcibly committing code, or from deleting the branch. You can set this branch to be protected. The procedure is as follows: On the CodeArts Repo homepage, go to the project homepage, choose **Settings > Policy Settings > Protected Branch**, click **Create Protected Branch**, and set parameters as follows.

- Step 1** Enter a branch name. This parameter is mandatory. Enter a complete branch name or a branch name with wildcard characters. If a branch contains a single slash (/), the branch cannot be matched using the wildcard * due to the fnmatch syntax rule.
- Step 2** You can set the push or merge permission for the administrator/project manager, committer, and developer. These two permissions cannot be granted at the same time because the protected branch cannot be forcibly pushed or merged into the code. You can create, edit, and delete protected branches in batches.

----End

If you want all repo groups and repo in this project to use the preceding settings, select **Force inherit**.

Configuring Project-Level Commit Rules

CodeArts Repo supports verification and restriction rules for high-quality code commits. On the CodeArts Repo homepage, go to the project homepage, choose **Settings > Policy Settings > Commit Rules**, and click **Create Commit Rule**. For details, see [Table 16-2](#).

Table 16-2 Parameters for project-level commit rules

Parameter	Description
Rule Name	Mandatory. Custom rule name.
Branch	Enter a complete rule name or create a regular expression. This parameter is mandatory. The input needs to be verified, including the branch name and regular expression.

Parameter	Description
Commit Rule	<p data-bbox="922 297 1043 331">Optional.</p> <ul data-bbox="922 342 1430 1839" style="list-style-type: none"><li data-bbox="922 342 1430 674">• Commit Message Match: This parameter is empty by default. If left blank, all messages can be committed. Every commit message that matches the regex can be committed. You can also set that the committed information must contain the work item number to implement E2E code tracing with max. 500 characters.<li data-bbox="922 685 1430 920">• Commit Message Negative Match: This parameter is empty by default. If left blank, all messages can be committed. Every commit message that matches the regex provided in it, will be rejected with max. 500 characters.<li data-bbox="922 931 1430 1335">• 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}){1,9})</code><li data-bbox="922 1435 1430 1839">• 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: Commit author's email: <code>@my-company.com\$</code>

Parameter	Description
Basic Attributes	<p>Optional.</p> <ul style="list-style-type: none">• File Name That Cannot Be Changed: This parameter is left empty by default, indicating that a file with any name 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 2,000 characters. Example: File name that cannot be changed: (\.jar .exe)\$• 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. <p>NOTE When a repository is created, the max. size of a single file in the default commit rule is 50 MB for recommendation. The max. file size is 200 MB.</p>
Binary Rules	<p>Optional.</p> <p>This is not selected by default. Do not allow new binary files (privileged users excepted) is selected by default. After Allow changes to binary files is selected, binary files in the modify state will not be 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)• Repo File Whitelist (files that can be directly imported to the database. This field supports a maximum of 2,000 characters.)• Privileged Users (Max. 50 privileged users.) <p>NOTE If the privileged user is not a repository member, the system displays a message indicating that the privileged user fails to be verified when you click Save. In this case, remove the privileged user to save the information.</p>

Parameter	Description
Effective Date	Optional. Before being committed, all commits created after the effective date must match the hook settings. If this parameter is left empty, all commits are checked regardless of the commit date.

Common Regular Expression Examples

[Common regular expression examples](#) are listed below.

Table 16-3 Examples

Rule	Example
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+

Rule	Example
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
Null string	\0

Configuring Project-Level Merge Request Rules

A merge request rule consists of three parts: merge mechanism, merge condition, MR setting, and merge method.

Table 16-4 Parameters of the merge mechanism

Parameter	Description
Merge Mechanism	<p data-bbox="922 342 1369 376">Mandatory. There are two options.</p> <ul data-bbox="922 387 1425 797" style="list-style-type: none"><li data-bbox="922 387 1425 618">● Score: Code review is included. You can set minimum merge score and the score ranges from 0 to 5. The code can be merged only when the score and mandatory review meet pass conditions. Set a minimum score when using this mechanism.<li data-bbox="922 629 1425 797">● Approval: It consists of code review and merge approval. Code can be merged only after the number of reviewers reaches gate requirements. <p data-bbox="962 808 1034 837">NOTE</p> <ul data-bbox="986 848 1425 1055" style="list-style-type: none"><li data-bbox="986 848 1425 904">● By default, Approval is used. You can manually switch to Score.<li data-bbox="986 916 1425 1055">● After the merge mechanism is changed, the workflows of the MRs are changed. However, the early created MRs retain the previous merge mechanism.

Table 16-5 Parameters of merge conditions

Parameter	Description
Merge Conditions	<p>Optional. The options are as follows:</p> <ul style="list-style-type: none">• Select Only when all reviews and comments are resolved. If Must resolve is selected, Review comment gate: failed will be displayed and the Merge button is dimmed. If it is only a common review comment, the Resolved button does not exist, and the comment will not be intercepted by the merge condition.• If you select Only when associated with CodeArts Req, all associated E2E ticket numbers must pass the verification. One MR can be associated with only one ticket number. You can add multiple branches to configure the merge request policy. You can manually enter wildcard characters, for example, *-stable or production/*, and press Enter to confirm.

Table 16-6 MR settings 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.
Do not approve your own requests	If you select this parameter, the Approve button is dimmed and you cannot approve your own MRs. You need to ask another person with the approve permission to approve your MRs.
Do not review your own requests	If you select this parameter, the Review button is dimmed and you cannot review your own MRs. You need to ask another person with the approve permission to approve your MRs.

Parameter	Description
A repo administrator can force 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.
Allow code review and comment for merged or closed MRs	After this parameter is selected, you can continue to review and comment on the merged 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, automatically merged MRs are marked as merged.• If this parameter is selected, MRs that are automatically merged are marked as closed.
Cannot re-open a Closed MR	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. This parameter is used for process control to prevent review history from being tampered with.
Delete source branch by default after the MR is merged	After the merge, the source branch is deleted. <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	After this parameter is selected, the Squash button is unavailable, and the entry for using this button is unavailable in the MR.

Parameter	Description
Enable Squash merge for new MRs	When squash merge in Git, all changes from the branch being merged are "squashed" into a single commit, which is then appended to the current branch as a "merge commit". This can simplify the branch history. 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 commit records, while squash merge has only one commit record.

Configuring Project-Level Member Sync

On the project homepage, choose **Code > Repo**, and choose **Settings > Security Settings > Member Sync**.

After **Sync Project Members** is enabled, select the roles to be synced. Selected project members will be synced to the repo and repo group. The project manager will always be synced regardless of the toggle. Click the refresh button to sync all the current settings.

Adding, deleting, or modifying a project member will be automatically synced.

Sync Project Members is disabled for historical projects but enabled for new projects, and the **Project manager**, **Committer**, and **Developer** are selected by default.

16.2 Configuring Project-Level Repo Settings

On the CodeArts Repo homepage, go to the project homepage and choose **Settings > Policy Settings > Repository Settings**. For details about how to set the parameters, see [Table 16-7](#).

Table 16-7 Project-level parameters

Parameter	Description
Force inherit	Optional. Once selected, all repo groups and code repos in the project use the following settings and cannot be changed. Exercise caution when selecting this option.
Do not fork a repository	Optional. Once selected, no one can fork the repo in the project.

Parameter	Description
Pre-merge	Optional. Once this is selected, the server automatically generates the pre-merge code of the MR. 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	Optional. All branch names must match the regular expression with max. 500 characters. If this field is left blank, any branch name is allowed. The rules must meet the following tag naming rules: <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .
Tag Name Rule	Optional. All tag names must match the regular expression specified by this parameter. If this field is left blank, any tag name is allowed. The basic tag naming rules must be met. <ul style="list-style-type: none">• Max. 500 characters.• Do not start with refs/heads/refs/remotes/ or end with . / .lock. Spaces and the following characters are not supported: [\ < ~ ^: ? () ' " .

Configuring Project-Level Protected Branch Rules

CodeArts Repo makes code branches more secure by preventing anyone other than the administrator from committing code, preventing anyone from forcibly committing code, or from deleting the branch. You can set this branch to be protected. The procedure is as follows: On the CodeArts Repo homepage, go to the project homepage, choose **Settings > Policy Settings > Protected Branch**, click **Create Protected Branch**, and set parameters as follows.

- Step 1** Enter a branch name. This parameter is mandatory. Enter a complete branch name or a branch name with wildcard characters. If a branch contains a single slash (/), the branch cannot be matched using the wildcard * due to the fnmatch syntax rule.

Step 2 You can set the push or merge permission for the administrator/project manager, committer, and developer. These two permissions cannot be granted at the same time because the protected branch cannot be forcibly pushed or merged into the code. You can create, edit, and delete protected branches in batches.

----End

If you want all repo groups and repo in this project to use the preceding settings, select **Force Inherit**.

Configuring Project-Level Commit Rules

CodeArts Repo supports verification and restriction rules for high-quality code commits. On the CodeArts Repo homepage, go to the project homepage, choose **Settings > Policy Settings > Commit Rules**, and click **Create Commit Rule**. For details, see [Table 16-8](#).

Table 16-8 Parameters for project-level commit rules

Parameter	Description
Rule Name	Mandatory. Custom rule name.
Branch	Enter a complete rule name or create a regular expression. This parameter is mandatory. The input needs to be verified, including the branch name and regular expression.

Parameter	Description
Commit Rules	<p>Optional.</p> <ul style="list-style-type: none">• Commit Message Match: This parameter is empty by default. If left blank, all messages can be committed. Every commit message that matches the regex can be committed. You can also set that the committed information must contain the work item number to implement E2E code tracing with max. 500 characters.• Commit Message Negative Match: This parameter is empty by default. If left blank, all messages can be committed. Every commit message that matches the regex provided in it, will be rejected with max. 500 characters.• 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}){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: Commit author's email: <code>@my-company.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 Changed: This parameter is left empty by default, indicating that a file with any name 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 2,000 characters. Example: File name that cannot be changed: (\.jar \.exe)\$● 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. <p>NOTE When a repository is created, the max. size of a single file in the default commit rule is 50 MB for recommendation. The max. file size is 200 MB.</p>
Binary Rules	<p>Optional.</p> <p>This is not selected by default. Do not allow new binary files (privileged users excepted) is selected by default. After Allow changes to binary files is selected, binary files in the modify state will not be 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)● Repo File Whitelist (files that can be directly imported to the database. This field supports a maximum of 2,000 characters.)● Privileged Users (Max. 50 privileged users.) <p>NOTE If the privileged user is not a repository member, the system displays a message indicating that the privileged user fails to be verified when you click Save. In this case, remove the privileged user to save the information.</p>

Parameter	Description
Effective Date	Optional. Before being committed, all commits created after the effective date must match the hook settings. If this parameter is left empty, all commits are checked regardless of the commit date.

Table 16-9 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}

Rule	Examples
Beginning of text	^
End of text	\$
Word boundary	\b
Non-word boundary	\B
Line breaker	\n
Carriage return character	\r
Tab key	\t
Null string	\0

Configuring Project-Level Merge Request Rules

A merge request rule consists of three parts: merge mechanism, merge condition, MR setting, and merge method.

Table 16-10 Parameters of the merge mechanism

Parameter	Description
Merge Mechanism	<p>Mandatory. There are two options.</p> <ul style="list-style-type: none">• Score: Code review is included. You can set minimum merge score and the score ranges from 0 to 5. The code can be merged only when the score and mandatory review meet pass conditions. Set a minimum score when using this mechanism.• Approval: It consists of code review and merge approval. Code can be merged only after the number of reviewers reaches gate requirements. <p>NOTE</p> <ul style="list-style-type: none">• By default, Approval is used. You can manually switch to Score.• After the merge mechanism is changed, the workflows of the MRs are changed. However, the early created MRs retain the previous merge mechanism.

Table 16-11 Parameters of merge conditions

Parameter	Description
Merge Conditions	<p>Optional. The options are as follows:</p> <ul style="list-style-type: none">• Select Only when all reviews and comments are resolved. If Must resolve is selected, Review comment gate: failed will be displayed and the Merge button is dimmed. If it is only a common review comment, the Resolved button does not exist, and the comment will not be intercepted by the merge condition.• If you select Only when associated with CodeArts Req, all associated E2E ticket numbers must pass the verification. One MR can be associated with only one ticket number. You can add multiple branches to configure the merge request policy. You can manually enter wildcard characters, for example, *-stable or production/*, and press Enter to confirm.

Table 16-12 MR setting 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.
Do not approve your own requests	If you select this parameter, the Approve button is dimmed and you cannot approve your own MRs. You need to ask another person with the approve permission to approve your MRs.
Do not review your own requests	If you select this parameter, the Review button is dimmed and you cannot review your own MRs. You need to ask another person with the approve permission to approve your MRs.

Parameter	Description
A repo administrator can force 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.
Allow code review and comment for merged or closed MRs	After this parameter is selected, you can continue to review and comment on the merged 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, automatically merged MRs are marked as merged.• If this parameter is selected, MRs that are automatically merged are marked as closed.
Cannot re-open a Closed MR	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. This parameter is used for process control to prevent review history from being tampered with.
Delete source branch by default after the MR is merged	After the merge, the source branch is deleted. <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	After this parameter is selected, the Squash button is unavailable, and the entry for using this button is unavailable in the MR.

Parameter	Description
Enable Squash merge for new MRs	When squash merge in Git, all changes from the branch being merged are "squashed" into a single commit, which is then appended to the current branch as a "merge commit". This can simplify the branch history. 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 commit records, while squash merge has only one commit record.

Configuring Project-Level Member Sync

On the project homepage, choose **Code** > **Repo**, and choose **Settings** > **Security Settings** > **Member Sync**.

After **Sync Project Members** is enabled, select the roles to be synced. Selected project members will be synced to the repo and repo group. The project manager will always be synced regardless of the toggle. Click the refresh button to sync all the current settings.

Adding, deleting, or modifying a project member will be automatically synced.

Sync Project Members is disabled for historical projects but enabled for new projects, and the **Project manager**, **Committer**, and **Developer** are selected by default.

16.3 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 extraction rules (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.

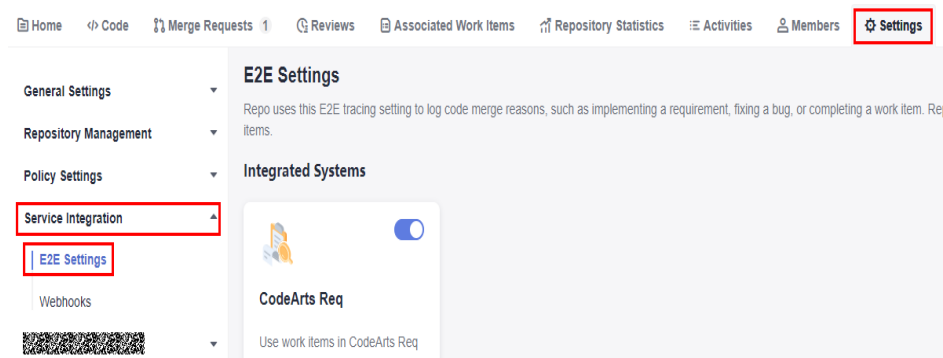
NOTE

- 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

[Click here to add a state.](#)

Associable Types Types of work items that can be associated with, e.g. Story/Task/Bug

[Click here to add a work item type.](#)

Applicable Branches Branches to comply with preceding restrictions

branch

Automatic ID Rules Extraction

ID Prefix Incorporated requirements Enter a prefix of

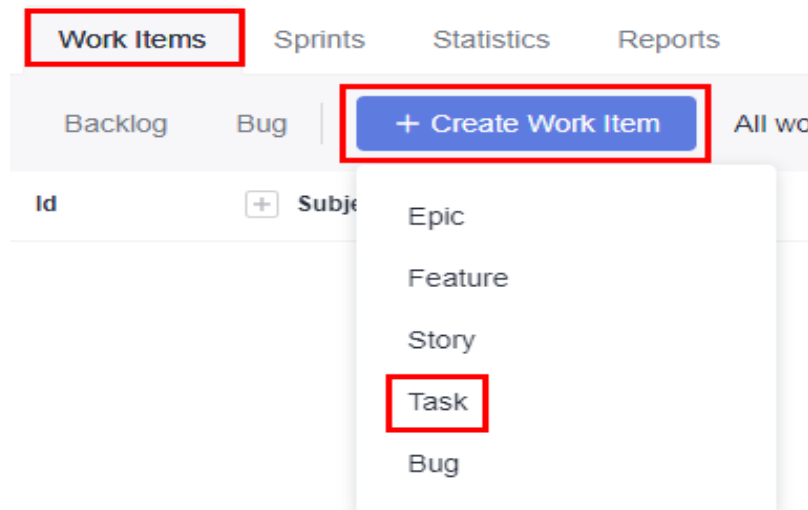
Separator Specify the separator (; by default).

ID Suffix Specify the suffix (line break by default).

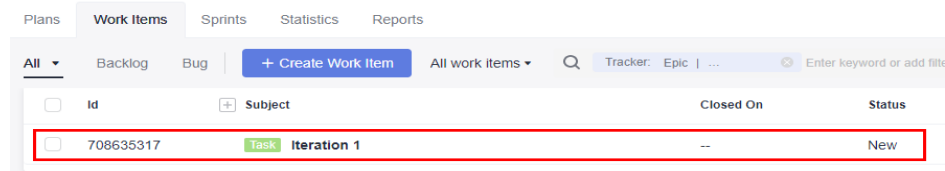
Submit Save and Enable

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.

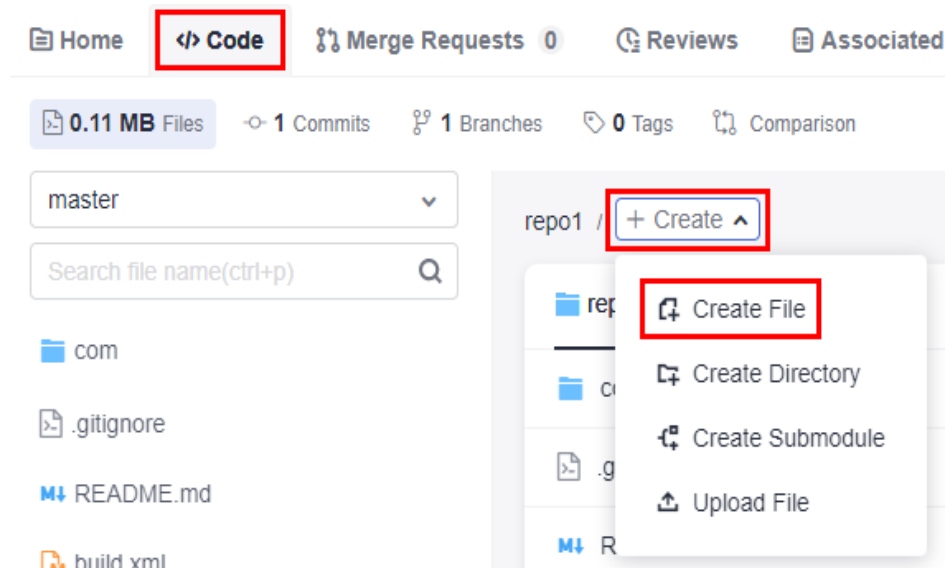


3. Enter a title, for example, Sprint 1.
Retain the default values for other parameters. Click **Save**.



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

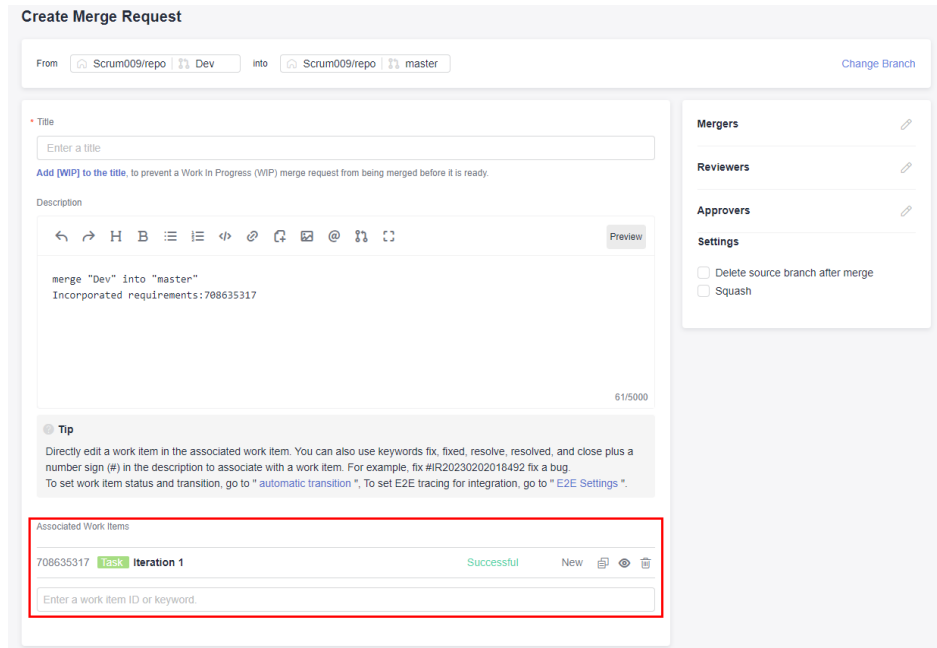
You can add 1965 more characters.

Tip
You can use keywords "fix", "fixed", "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

16.4 Webhook Settings

Configuring Webhook Settings

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.

To configure webhooks, you can choose **Settings > Service Integration > Webhooks** on the repository details page.

The settings take effect only for the repository configured. Members within the repo can view this page.

Table 16-13 Parameters for creating a webhook

Parameter	Description
Name	Custom name.
Description	Description of the webhook.
URL	Mandatory. Provided by the third-party CI/CD system.

Parameter	Description
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.
Trigger Events	<p>The system can subscribe to the following events:</p> <ul style="list-style-type: none">• Code push<ul style="list-style-type: none">– If Code push 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 the regular expression.</p> <ul style="list-style-type: none">– 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. <ul style="list-style-type: none">• Tag push This event is triggered when a tag is created or deleted.• Merge requests<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 comment 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.

17 Committing Code to CodeArts Repo and Creating a Merge Request

17.1 Setting Repo-Level Merge Request Rules

The MR configuration refers to the configuration of code merge conditions and modes in MR mode. Project-level MR rules can be inherited to the repos and repo groups.

You can select **Inherit from project**. The settings in the project are automatically inherited and cannot be changed. You can also access the repo homepage of the code to be configured and choose **Settings > Policy Settings > Merge Requests**. There are two MR mechanisms: score and approval. The differences between the two mechanisms are as follows:

- Scoring mechanism: This mode includes only code review and is based on scoring. Code can be merged only when the score meets the gate conditions.
- The approval method consists of code review and merge approval. Code can be merged only after the number of reviewers and approvers reaches gate requirements.

After selecting a mechanism, set other parameters by referring to the [following table](#). The configuration takes effect for the entire code repo.

Table 17-1 Parameters for setting the merge condition and mode

Parameter	Description
Merge Conditions	<p data-bbox="954 342 1374 376">Optional. There are two options:</p> <ul data-bbox="954 387 1433 1429" style="list-style-type: none"><li data-bbox="954 387 1433 757">• 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.<li data-bbox="954 768 1433 1429">• Must be associated with CodeArts Req The options are as follows:<ol data-bbox="991 846 1433 1429" style="list-style-type: none"><li data-bbox="991 846 1433 981">1. Associate only 1 ticket number. After this option is selected, one MR can be associated with only one ticket number.<li data-bbox="991 992 1433 1149">2. All E2E ticket numbers pass verification After this option is selected, all associated E2E ticket numbers must pass the verification.<li data-bbox="991 1160 1433 1429">3. Branches to configure the MR policy: You can add multiple branches to configure the merge request policy. You can manually enter wildcard characters (for example, *-stable or production/*), and press Enter.

Parameter	Description
MR Settings	<p data-bbox="954 300 1190 331">Optional. Options:</p> <ul data-bbox="954 344 1430 1538" style="list-style-type: none"><li data-bbox="954 344 1430 577">• 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.<li data-bbox="954 591 1321 651">• Do not approve your own requests<li data-bbox="954 665 1417 696">• Do not review your own requests<li data-bbox="954 710 1390 770">• A repo administrator or project manager can force merge code<li data-bbox="954 784 1414 844">• Allow code review and comment for merged or closed MRs<li data-bbox="954 857 1430 1128">• Mark the automatically merged MRs as Closed If all commits in MR A are included in MR A, MR B is automatically merged after MR A is merged. By default, the B MR is marked as merged. You can use this parameter to mark the B MR as closed.)<li data-bbox="954 1142 1425 1270">• Cannot re-open a Closed MR This function is enabled by default. You can enable or disable it as required.<li data-bbox="954 1283 1369 1344">• Enable "Delete source branch after merged" when creating MRs<li data-bbox="954 1357 1422 1462">• Forbid squash merge (Forbid to select squash merge when create merge request)<li data-bbox="954 1476 1374 1536">• Enable Squash merge for new MRs

Parameter	Description
Merge Method	<p data-bbox="954 300 1347 360">Mandatory. The options are as follows:</p> <ul data-bbox="954 376 1426 1989" style="list-style-type: none"><li data-bbox="954 376 1426 1198">• Merge commit 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.<ul data-bbox="991 689 1426 1198" style="list-style-type: none"><li data-bbox="991 689 1426 853">- Do not generate Merge nodes during Squash merge: If this parameter is selected, no merge node is generated during the squash merging.<li data-bbox="991 869 1426 1025">- Use MR merger to generate Merge Commit: If this parameter is selected, the commit information is recorded.<li data-bbox="991 1041 1426 1198">- Use MR creator to generate Merge Commit: If this parameter is selected, the commit information is recorded.<li data-bbox="954 1214 1426 1742">• Merge commit with semi-linear history. 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.<li data-bbox="954 1758 1426 1989">• Fast-forward 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

Parameter	Description
	possible, the user is given the option to rebase.

Configure Branch Policy

If you select the **Approval** mechanism as the **Merge Mechanism** and want to configure a merge policy for a branch, go to the repo homepage to be configured, choose **Settings > Policy Settings > Merge Requests**, click **Create Branch Policy**, and set parameters by referring to the [following table](#).

Click **Create Branch Policy** to set a merge policy for a specified branch or all branches in the repository.

Table 17-2 Parameters for creating a branch policy

Parameter	Description
Branch	Mandatory. Select a branch from the drop-down list. You can select all branches.
Reviewers Required	Mandatory. The default value is 0, indicating that the review gate can be passed without being reviewed by the reviewer.
Approvals Required	Mandatory. The default value is 0, indicating that the approval gate can be passed without being approved by the approver.
Reset approval gate	Optional. This option is selected by default, indicating that MR approval gate is reset when code is re-pushed to the source branch of the MR.
Reset review gate	Optional. This option is selected by default, indicating that the MR review gate is reset when code is re-pushed to the source branch of the MR.
Add approvers/reviewers only from the following ones	Optional. 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.
Enable pipeline gate	Optional. 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	Optional. The list of mandatory mergers can be configured. When a merge request is created, the list is automatically synchronized to the merge request.
Approvers	Optional. The list of mandatory approvers can be configured. When a merge request is created, the list is automatically synchronized to the merge request.

Parameter	Description
Reviewer	Optional. The list of mandatory reviewers can be configured. When a merge request is created, the list is automatically synchronized to the merge request.

NOTE

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.

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.

17.2 Configuring Merge Request Notifications

You can push notifications about repositories and merge requests through emails or WeCom, and enable either or both of these two modes as needed. Repository members can view this page. Only roles with the **Set** permission can set repository notifications.

- For details about how to configure email notifications, see [Configuring Email Notifications](#).
- For details about how to configure WeCom notifications for repositories, see [Table 17-4](#).

Configuring Email Notifications

Table 17-3 Parameters for email notifications

Parameter	Description
Repository	<p>Optional. Set the email notification you want to receive. Four options are available. By default, Freeze Repo and Close Repo are selected and cannot be changed. If a repo is frozen or closed, an email will be sent to the repo owner and project administrator. The other two options are as follows, and you can select when to receive email notifications:</p> <ul style="list-style-type: none">• Delete Repo: When a member deletes the repository.• Capacity Warning: When the capacity usage exceeds the threshold. You can select 60%, 80%, or 90% from the drop-down list.

Parameter	Description
Merge Request	<p>Optional. You can select the following options as needed.</p> <ul style="list-style-type: none">• Open: When a merge request is created or re-created, an email will be sent to the selected roles. By default, the following roles are selected: Scorer, Approver, Reviewer, and Merger.• Update: When the code of the branch associated with the merge request is updated, an update email is pushed. The following roles are selected by default: Scorer, Approver, and Reviewer.• Merge: An email will be pushed when a merge request is committed. The MR creator is selected by default. You can also select Merger.• Review: An email will be sent to notify the merge request review. The MR creator is selected by default.• Approve: An email will be sent to notify the merge request approval. The MR creator is selected by default.• Comment: The email of new review comments will be sent to the selected role. The MR creator is selected by default.• Resolve Comment: An email will be sent to the selected role to resolve the review comments. The MR creator is selected by default.

 **NOTE**

If you have enabled the email notification in CodeArts Repo but cannot receive any email notification, go to [CodeArts notifications](#) to check whether the email configuration and email notification are enabled.


Configuring WeCom Notification Settings for the Repository

Table 17-4 Parameters for setting WeCom notifications

Parameter	Description
Webhook URL	Mandatory. Used to identify webhook address of the robot added to the CodeArts Repo member group with a maximum of 500 characters.
Repository	Optional. Select the following two options based on your need. By default, the following two options are selected. You can also select the email recipients. <ul style="list-style-type: none">• Delete Repo: When a member deletes the repository.• Capacity Warning: When the capacity usage exceeds the threshold. You can select 60%, 80%, or 90% from the drop-down list.
Merge Request	Optional. You can select the following options as needed. <ul style="list-style-type: none">• Status Change Notifications are pushed through the WeCom bot when the MR is opened, updated, or merged. By default, the following options are selected: Open and Merge.• Review and Approval: You can select Review or Approve.• Review Comments: By default, Comment is selected. You can also select Resolved comment.

17.3 Resolving Review Comments and Merging Code

Passing the Review Comment Gate

If the merge request gate is enabled for the target repository, the **Only when all reviews and comments are resolved** option is selected. The reviewers or approvers can move the cursor to the code line to which the review comment is to be added in **Files Changed** of the **Merge Requests** tab and click the  icon to add reviews. Alternatively, the reviewers or approvers can directly add comments in **Details > Comments** of the **Merge Requests**.

After you resolve the review comments, on the **Details > Review Comments** page of the merge request, the review comment status changes from **Unresolved** to

Resolved. Review comment gate: Passed is displayed, indicating that the merge request initiator has resolved all review comments and can merge the MR.

Pipeline Gate

If CodeArts Pipeline gates are enabled for the target repository, select **Enable pipeline gate**. Perform the following steps:


- Step 1** Go to the target repo homepage. In the navigation pane on the left, choose **CICD > Pipeline**.
- Step 2** Click **Create Pipeline**, enter the following information, click **Next**, and select the target template.
 - **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** After a task is created, the system automatically switches to the **Task Orchestration** tab page. Click **More** and select **Execution Plan**.
- Step 4** Enable **Merge Request** and select one of the following trigger events based on your needs:
 - **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 commit event will also be triggered.
 - **Reopen**: triggered upon MR reopening.
- Step 5** Configure other information about the pipeline task and click **Save and Execute**.
- Step 6** Return to CodeArts Repo and wait for the event selected in **Execution Plan** to be triggered for the repository to execute the CodeArts Pipeline task.

----End

Go to the merge request details page. If the message **Merge into pipeline gate: passed** is displayed, the latest commit or pre-merge commit successfully starts the pipeline.

Associating Gate Control by E2E Ticket Number

If E2E ticket number association is enabled for the target repository, select **Must be associated with CodeArts Req**. Perform the following steps to associate the E2E ticket number:

- Step 1** Go to the target repo, switch to the **Merge Requests** tab page, and click a target merge request name to access it.
- Step 2** On the **Details** page, click the  icon next to **Associated Work Items** to search for and select the target work item.

Step 3 Click **OK**. The E2E ticket number is associated.

----End

When the merge request is successfully associated with a work item, **E2E ticket number associated** is displayed.

 **NOTE**

- If the system displays a message indicating that the capacity of a single repository exceeds 2 GB and the merge is not allowed, check whether there are Git commit cache files submitted.
- A maximum of 100 work items can be associated with an MR.

18 Managing Merge Requests

18.1 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.

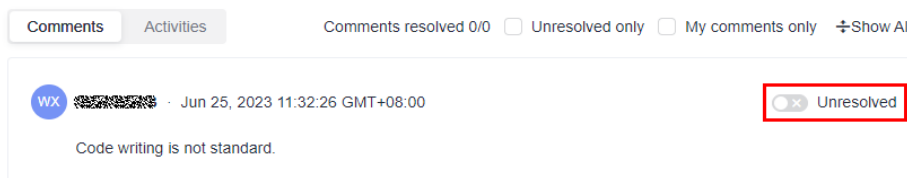


- **Review comment gate: failed:** It is displayed when the review comments in the Merge request are not resolved.



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**.



18.2 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 ...
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to '...'
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.
```

- Max. 50 conflict files can be solved, and the size of a single conflict file cannot exceed 200 KB.

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

NOTE

The **git pull** command 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  ##file01AAAAAAAAAAAA
2  ##file02BBBBBBBBBBBB
3  ##file03CCCCCCCCCCCC
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 |
```

- 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
##file04DDDDDDDDDDDD
## add by 02_dev
=====
##modify by 01_dev modify by 01_dev
##file03CCCCCCCCCCCCC
##file04DDDDDDDDDDDD
## 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  ##file04DDDDDDDDDDDDD
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.

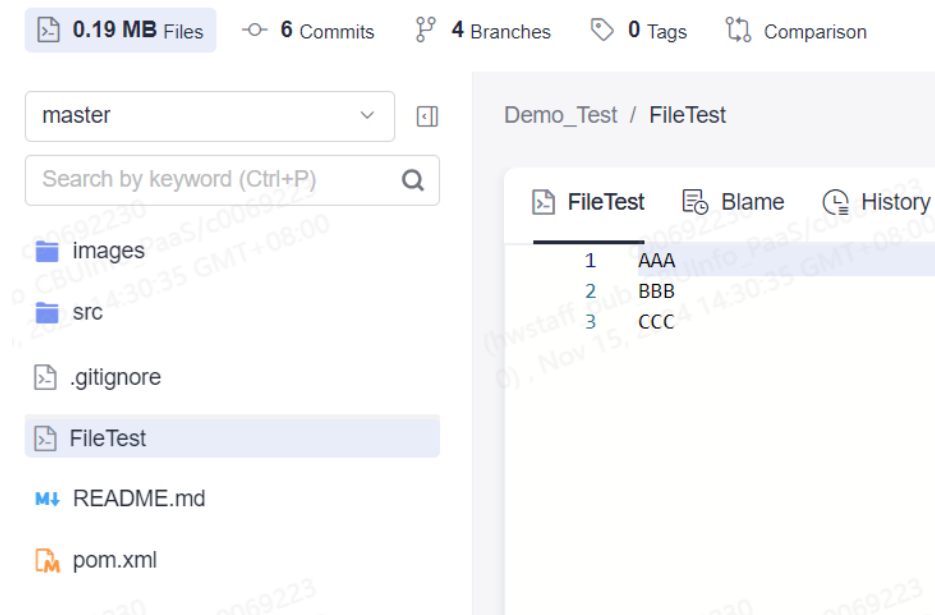
Resolving a Merge Conflict

CodeArts Repo supports branch management. When branches are merged, conflicts may occur. This case describes how to resolve a merge request conflict by reproducing it.

Step 1 Create a repo named **Demo_Test**.

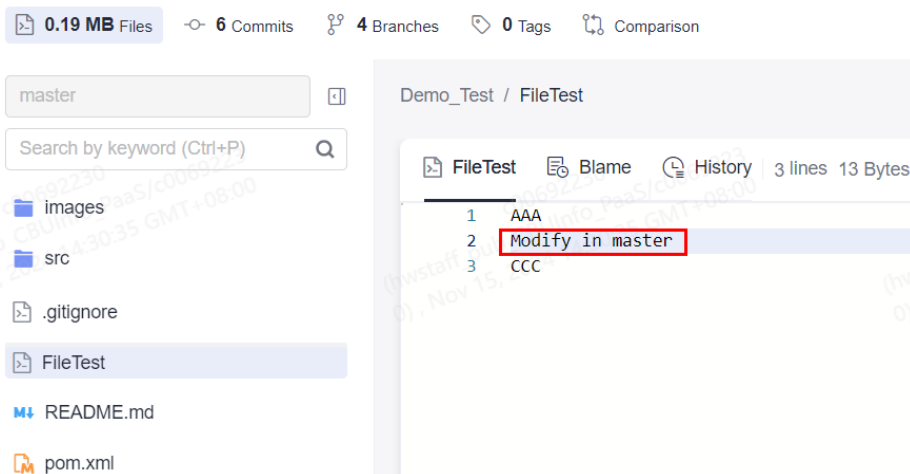
Step 2 Create a file named **FileTest** based on the master branch. The following figure shows the content of the file.

Figure 18-1 Creating the **FileTest** file in the master branch

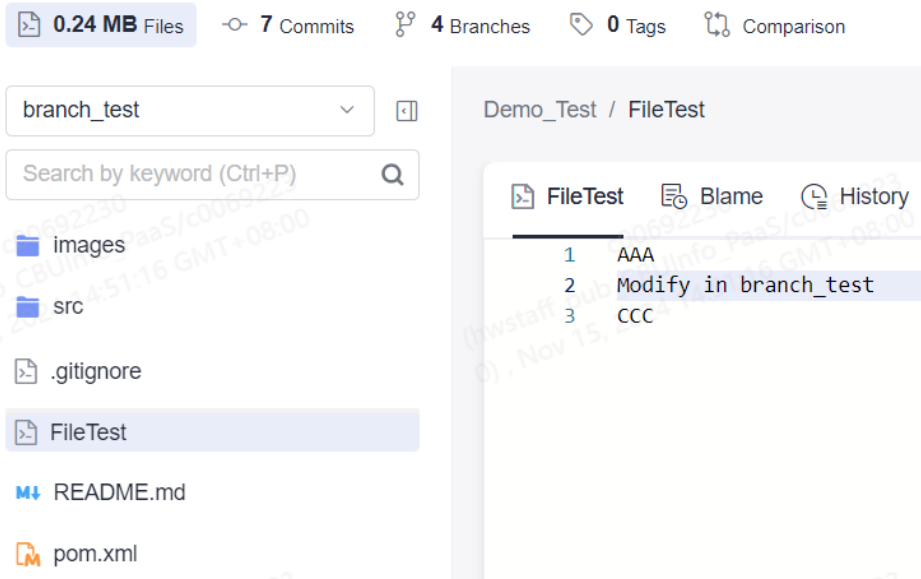


Step 3 Create the **branch_test** branch based on the **master** branch. In this case, the content in the **master** branch is the same as that in the **branch_test** branch. The following describes how to make the content of the two branches different.

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

Figure 18-2 Modifying the **FileTest** file in the **master** branch

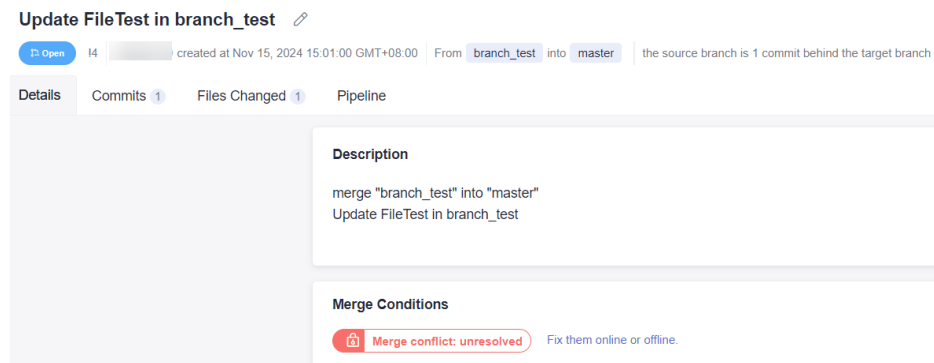
Step 5 Switch to the **branch_test** branch and modify the **FileTest** content, as shown in the following figure. Fill in the commit information as **Update FileTest in branch_test**. Now the content of the **master** branch is different from that of the **branch_test** branch and this is when a code conflict occurs.

Figure 18-3 Modifying the **FileTest** file in the **branch_test** branch

Step 6 Switch to the **branch_test** branch, click **Create Merge Request** in the upper right corner, and merge the **branch_test** branch into the **master** branch.

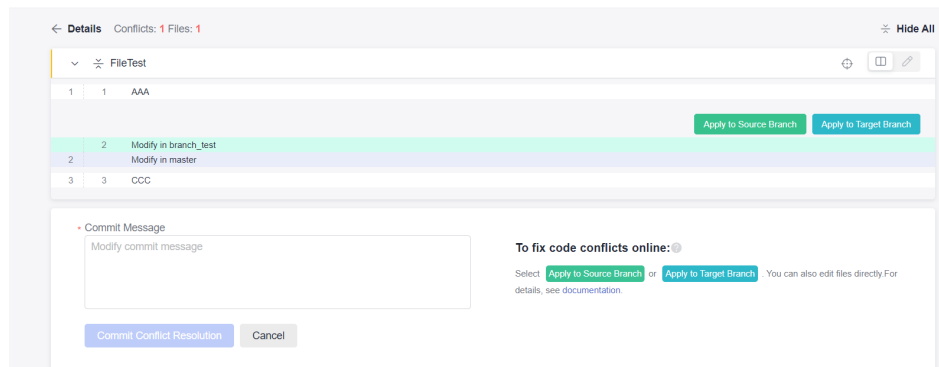
The **Details** page of the merge request is displayed as shown in the following figure. **Merge conflict: unresolved** is displayed, and you are recommended to **Fix them online or offline**.

Figure 18-4 Creating a merge request

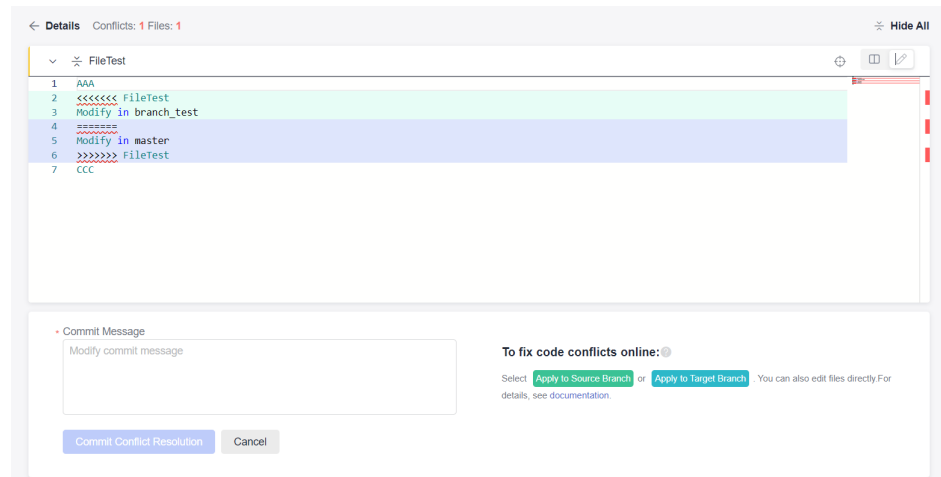


Step 7 Perform the following operation to resolve the conflict:

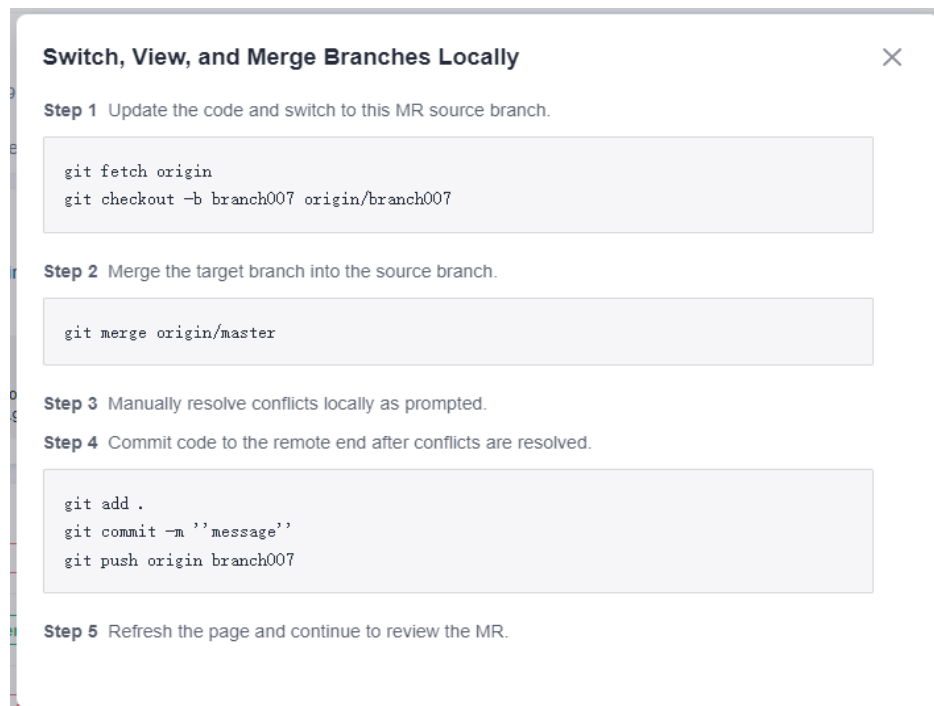
- **Resolving a conflict on CodeArts Repo** (recommended for small code volume)
 - a. Click **Fix them online** and the following page is displayed. To resolve the conflict, select **Apply to Source Branch** or **Apply to Target Branch**. If you select **Apply to Source Branch**, the content of the **branch_test** branch is applied to the **master** branch. If you select **Apply to Target Branch**, the content of the **master** branch is applied to the **branch_test** branch.



- b. If there are many conflicts, click to go to the page shown in the following figure to edit or resolve them online. The lines where the <<<<, >>>>, and ===== signs are located display conflict and splitter which need to be deleted when modifying the code to solve the conflict.

Figure 18-5 Solving conflicts online

- **offline** (recommended for large-scale projects)
Click **offline** and follow the prompted instructions as shown in the following figure.

**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** Use either of the **following methods to resolve the conflict**. Click **Merge** to merge the branches and the system displays a message indicating that the merge is successful. There is no difference in the content of the **branch_test** source branch and that of the **master** target branch.

----End

18.3 Creating a Squash Merge

Squash merge is to merge all change commit information of a merge request into one to simplify the commit information. 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.

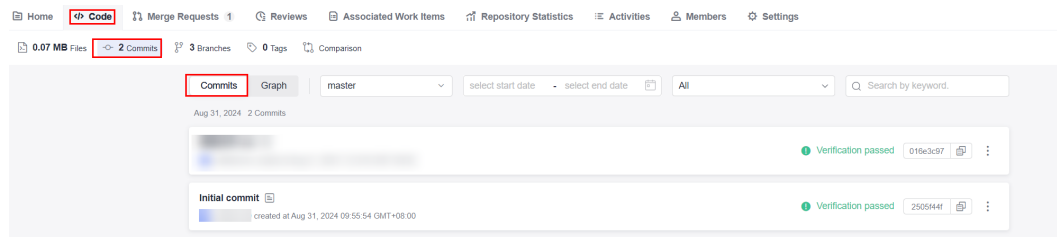
To better understand this function, perform the following operations:

Step 1 Create a repo and a branch.

The repo name is **repo**, and the branch name is **Dev**.

Step 2 Dev branch: Create two files and name them **Function_1** and **Function_2**.

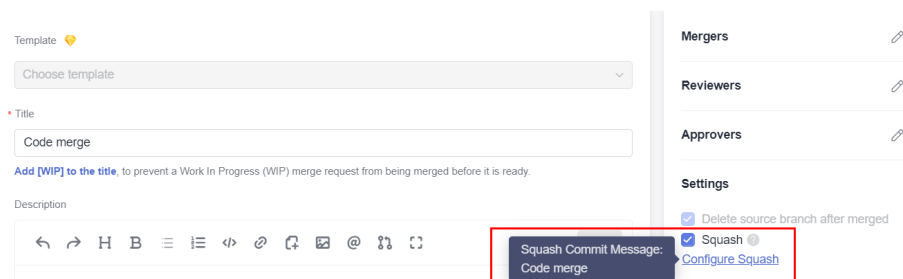
Step 3 Check the effect before Squash is enabled. Click the **Dev** branch and choose **Code > Commits > Commits** to view the commit information.



Step 4 Create and merge a request.

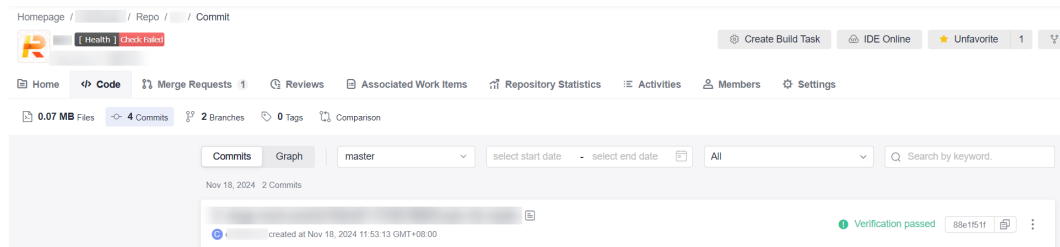
1. Set the source branch to **Dev** and target branch to **master**. Create a merge request.

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



2. After the merge request is reviewed and approved, the request can be merged.

Step 5 Check the effect after **Squash** is enabled. After the request is successfully merged as shown in the following figure, click the **Code**, **Commits**, and **Commits** tabs, select the **master** branch. Compared with **Step 4**, the committed content has been merged.



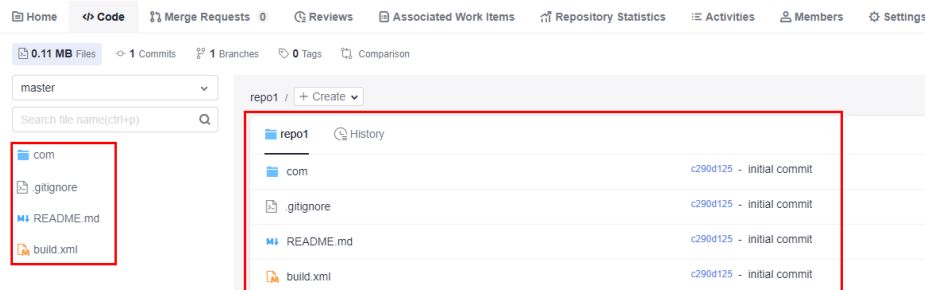
----End

19 Managing Code Files

19.1 Managing Files

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

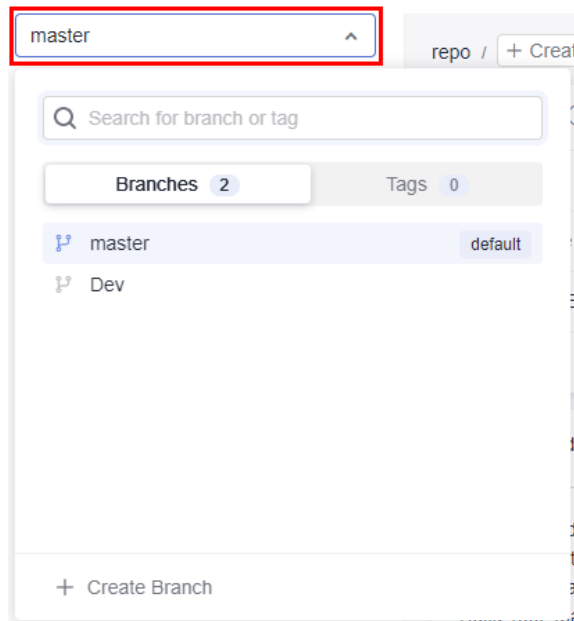
When you access the repository details console, you will be on the **Files** subtab of the **Code** tab page. You can switch to different branches and tags to view the files in the corresponding version. As shown in the following figure, the file list under the main branch is displayed on the left, the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




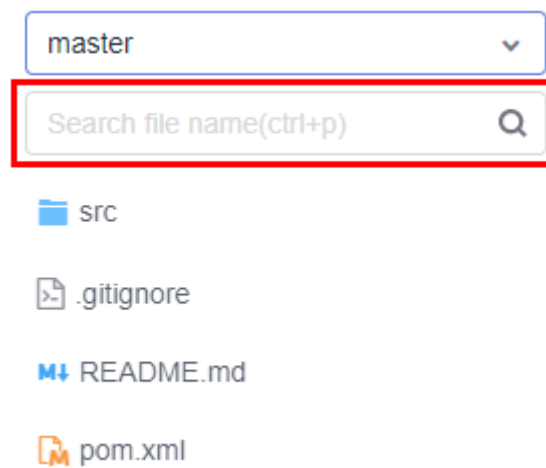
File List

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 to view associated work items by referring to 11.7.

– **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**.


– **Create 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
repo		
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:

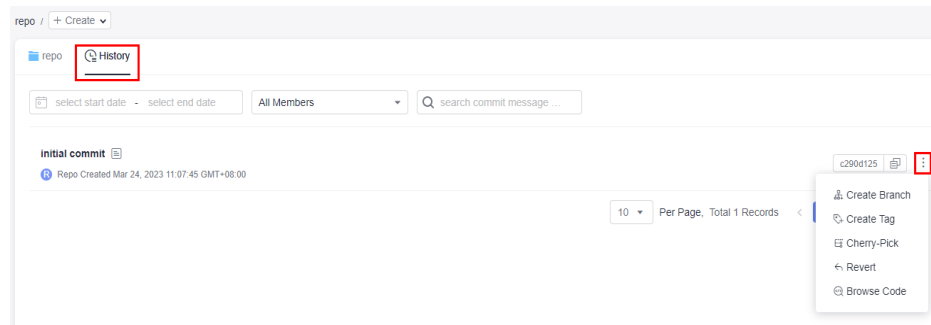
- **File:** name of a file or folder.
- **Commit message:** message of the last commit to the file or folder (**-m** in the **commit** command). You can click the message to display the commit record.
- **Creator:** creator of the last commit to the file or folder.
- **Update time:** last update time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

 NOTE


Commit messages are required for the edit and delete operations. These operations are similar to **-m** in the **git commit** command and can be used to view associated work items by referring to 11.7.

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.(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 19-1 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. (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.**
- **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**

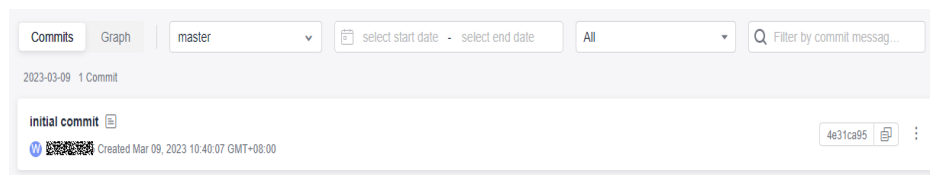
The comparison result shows the impact of merging from the left repository version to the right repository version on the files in the right repository. If you want to know the differences between the two file versions, you can adjust the left and right positions, compare them again, and learn all the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

19.2 Managing Commits

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

Commits

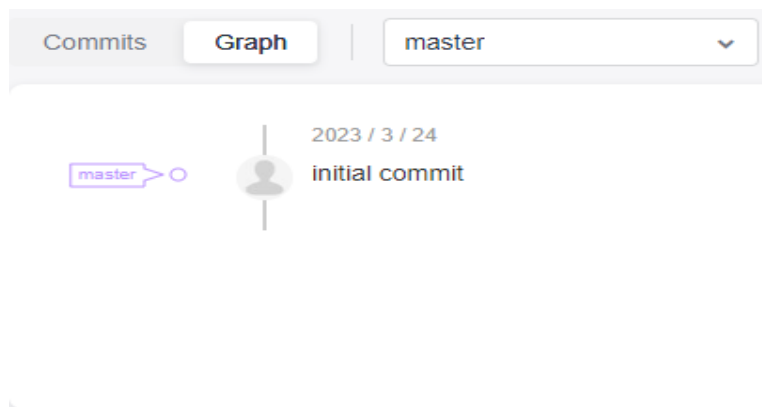
This tab displays the entire commit records of a branch or tag in the current repository. You can filter records by time segment, committer, commit message, or commit ID.



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.

19.3 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.

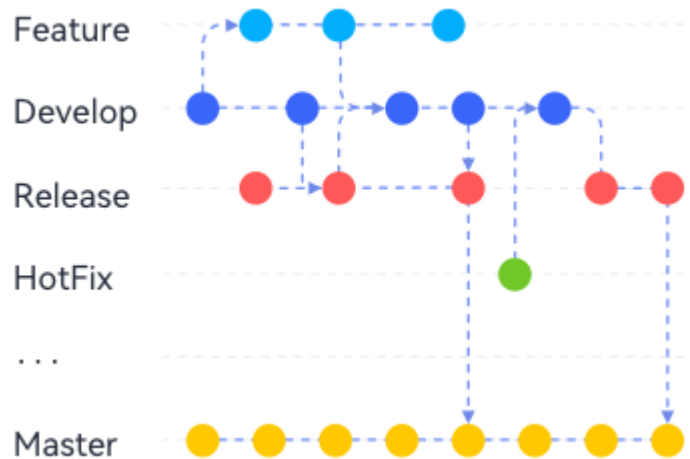


Table 19-2 Suggestions on using GitFlow branches

Branch	Master	Develop	Feature_1\ 2...	Release	HotFix_1\ 2..
Description	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.
Validity	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 these special characters 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 last month. Branches are sorted by the last commit time in descending order.
 - **Inactive**: displays the branches that have not been developed in the last month. 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.
- You can click  to download its compressed package.
- You can click  to access the **Merge Requests** tab page and create a merge request.
- You can 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 Settings
- Default Branch
- 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'

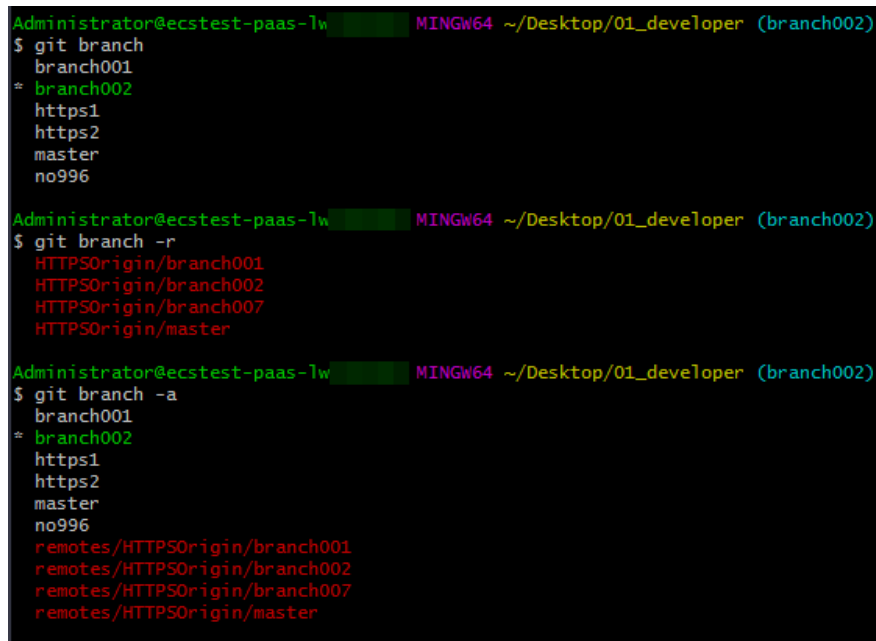
Administrator@ecstest-paas-lw MINGW64 ~/Desktop/01_developer (branch002)
$
```

- **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-1w MINGW64 ~/Desktop/01_developer (branch002)
$ git branch
branch001
* branch002
https1
https2
master
no996

Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (branch002)
$ git branch -r
HTTP5origin/branch001
HTTP5origin/branch002
HTTP5origin/branch007
HTTP5origin/master

Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (branch002)
$ git branch -a
branch001
* branch002
https1
https2
master
no996
remotes/HTTP5origin/branch001
remotes/HTTP5origin/branch002
remotes/HTTP5origin/branch007
remotes/HTTP5origin/master
```

- **Merging a branch**

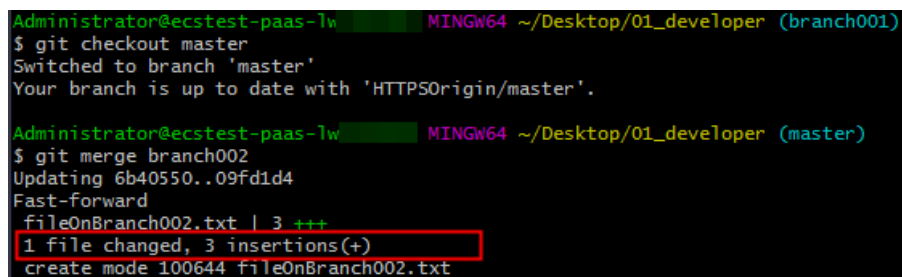
When a development task on a branch is complete, the branch needs to be merged into another branch to synchronize the latest changes.

```
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-1w MINGW64 ~/Desktop/01_developer (branch001)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'HTTP5origin/master'.

Administrator@ecstest-paas-1w MINGW64 ~/Desktop/01_developer (master)
$ git merge branch002
Updating 6b40550..09Fd1d4
Fast-forward
 file0nBranch002.txt | 3 +++
1 file changed, 3 insertions(+)
create mode 100644 file0nBranch002.txt
```

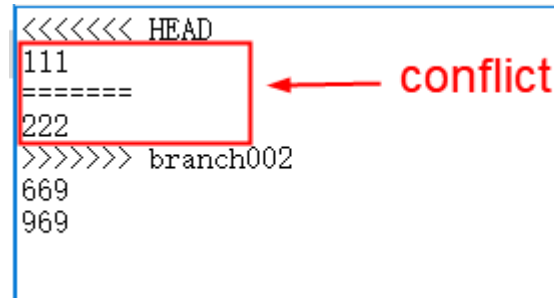
 NOTE

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
```



- **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@*****.com
```

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

19.4 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. (What are annotated tags?)

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. (What are annotated tags?)

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 17:00:00 [Download] [Delete]
Version1.0	Repo Committed c290d125 · Initial commit Mar 24, 2023 11:07:45 GMT+08:00	Based On master Created Apr 13, 2023 17:00:00 [Download] [Delete]

- 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.

- 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.

ftbc5a25 - Rename build.xml	d0ccc40b - update pom.xml
1 <project name="javaAntDemo" basedir="." default="main">	1 <project name="javaAntDemo" basedir="." default="main">
2 <property environment="env" />	2 <property environment="env" />
3 <property name="src.dir" value="com"/>	3 <property name="src.dir" value="com"/>
4	4
5 <property name="build.dir" value="build"/>	5 <property name="build.dir" value="build"/>
6 <property name="classes.dir" value="\${build.dir}/classes"/>	6 <property name="classes.dir" value="\${build.dir}/classes"/>
7 <property name="jar.dir" value="\${build.dir}/jar"/>	7 <property name="jar.dir" value="\${build.dir}/jar"/>
8 <property name="report.dir" value="\${build.dir}/junitreport"/>	8 <property name="report.dir" value="\${build.dir}/junitreport"/>
9 <taskdef name="findbugs" classname="edu.umd.cs.findbugs.anttask.FindBugsTa	9 <taskdef name="findbugs" classname="edu.umd.cs.findbugs.anttask.FindBugsTa
10 <property name="fb.report.dir" value="\${build.dir}/findbugs"/>	10+
11	11
12 <path id="application" location="\${jar.dir}/\${ant.project.name}.jar"/>	12 <path id="application" location="\${jar.dir}/\${ant.project.name}.jar"/>

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@ecs-test-paas-lw6: ~$ git tag esay
Administrator@ecs-test-paas-lw6: ~$ git show esay
commit d7dcaff34c62f0da4a2528bd1a725044b2c885f2 (HEAD -> https://, tag: esay, HTTPSOri
Author: Administrator <3eaf391356a7407aadbd89862@ecs-test-paas-lw6.com>
Date: Tue Jun 30 11:41:42 2024 +0800

    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@ecs-test-paas-lw6: ~$ git tag -a name1 -m "This is my Tag For Test1"
Administrator@ecs-test-paas-lw6: ~$ git show name1
tag name1
Tagger: Administrator <74105@ecs-test-paas-lw6.com>
Date: Tue Jun 30 20:03:54 2024 +0800
This is my Tag For Test1

commit d7dcaff34c62f0da4a2528bd1a725044b2c885f2 (HEAD -> https://, tag: name1, tag: esay, HTTPSOri
Author: Administrator <3eaf391356a7407aadbd89862@ecs-test-paas-lw6.com>
Date: Tue Jun 30 11:41:42 2024 +0800

    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@weid.com>
Date: Mon Jun 29 09:14:01 2025
```

```
git tag -a historyTag -m "Tag a historical version." 6a5b7c8db # Add tag historyTag to the historical version whose commit ID starts with 6a5b7c8d, 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 d9caff34c62f0d4a2528bd1a725044b2c85f2 (HEAD -> master, tag: tag5, tag: tag4, tag: tag3, tag: tag2, tag: tag1, tag: name1, tag: esay, tag: tag0)
Author: Administrator <3eaf391356a7407aad8db9862@weid.com>
Date: Tue Jun 30 11:41:43 2025
```

- **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-lwx MINGW64 ~/Desktop/01_developer (master)
$ git show myTag1
tag myTag1
Tagger: 01_dev <74105@weid.com>
Date: Tue Jun 30 11:41:43 2025

This is a tag for show you~!

commit 53538093c56de4df204b12ca4841926eef630bbd (tag: myTag1)
Author: 02_dev <yuhu@weid.com>
Date: Tue Jun 30 11:41:43 2025

    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.

- **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://[redacted].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.
```

```
Administrator@ecstest-paas-lw MINGW64 /d/403 (master)
$ git checkout V2.0.0
Note: switching to 'V2.0.0'.
```

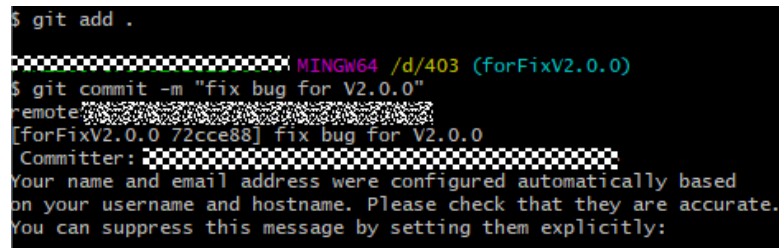
2. Create a branch based on the current working directory and switch to it.

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

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

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

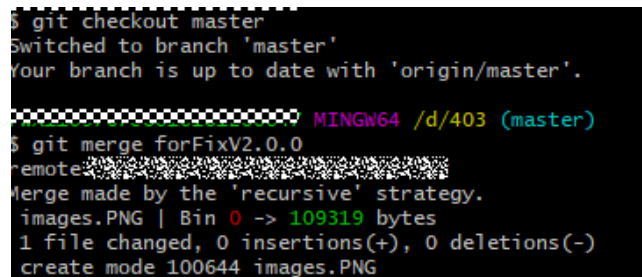
```
git add . # Add the changes to the temporary zone of the new branch.  
git commit -m "fix bug for V2.0.0" # Save the changes to the repository of the branch.
```



```
$ 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: [redacted]  
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:
```

- 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.
```



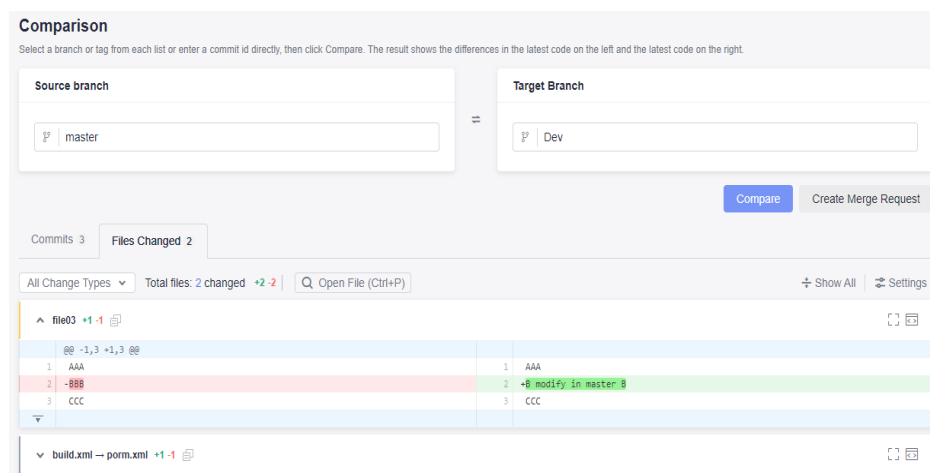
```
$ git checkout master  
Switched to branch 'master'  
Your branch is up to date with 'origin/master'.  
MINGW64 /d/403 (master)  
$ git merge forFixV2.0.0  
remote: [redacted]  
Merge made by the 'recursive' strategy.  
 images.PNG | Bin 0 -> 109319 bytes  
 1 file changed, 0 insertions(+), 0 deletions(-)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

19.5 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.



The screenshot shows the 'Comparison' interface in CodeArts Repo. It features two input fields for 'Source branch' (set to 'master') and 'Target Branch' (set to 'Dev'). Below these fields are 'Compare' and 'Create Merge Request' buttons. The interface displays a list of files with change indicators. For 'file03', it shows a diff between two versions: version 1 has 'AAA' and 'CCC', while version 2 has 'AAA', 'CCC', and a new line '+B modify in master B'. For 'build.xml', it shows a change from 'pom.xml' to 'pom.xml'.

NOTE

After comparing branches, you can create a merge request as needed.

20 Security Management

For higher security, CodeArts Repo allows you to add IP addresses to the whitelist, change the repository owner, delete the repository, change the repository name, add watermarks, lock the repository, and record audit logs. For details, see the following sections. Only the users who have the permission to **set** repo groups or repositories can perform these operations. For details about how to **set** permissions, see [Configuring Repo-Level Permissions](#).

Configuring a Deploy Key

For security purposes, some repositories can only be cloned and downloaded and do not support other change operations such as merge code. You can configure a deploy key for a read-only repository. To generate a deploy key locally, choose **Settings > Security Management > Deploy Key** on the repository details page. On the **Deploy Key** page, click **Add Deploy Key**. For details about how to generate an SSH key locally, see step 1 to step 3 in [Configuring an SSH Private Key](#).

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 code 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.

Risky Operations

CodeArts Repo allows you to change the repository owner, delete a repository, and change the repository name, but these operations are also risky. Exercise caution when performing these operations.

To configure risky operations, choose **Settings > Security Management > Risky Operations** on the repository details page. The following operations are supported:

- **Transfer repository ownership:** You can transfer the current repository to another member in the repository (but not to a viewer).

- **Delete repository:** Once you delete the repository, all content in the repository will be permanently deleted. This operation cannot be undone. Please exercise caution.
- **Rename repository:** This will invalidate the original path for access and clone. Please exercise caution.

Adding Watermark to a Repository

CodeArts Repo allows you to add watermarks to repositories to protect intellectual property rights.

To set watermarks, choose **Settings > Security Management > Watermark** on the repository details page.

After the watermark is enabled, the repository displays the following watermark content: account + time.

Locking a Repository in CodeArts

You can lock a repository to prevent anyone from damaging its upcoming versions.

To lock a repository, choose **Settings > Security Management > Repository Locking** on the repository details page. Repo members with the settings permission can perform this operation.

If the watermark is enabled, the repository is locked and read-only. No one can commit code to any branch, create comments, or perform other related operations.

Setting an IP Address Whitelist for CodeArts Repo

In CodeArts, you can set the IP address range and access for the IP address whitelist to restrict users' access, upload, and download permissions, enhancing repository security. The IP address whitelist takes effect only for repos whose visibility is **Private**, **Read-only for project members**, and **Read-only for tenant members**.

To configure the IP address whitelist, you can choose **Settings > Security Management > IP Address Whitelist** on the repository details page. IPv4 and IPv6 are supported. The [following table](#) lists the three formats of IP address whitelists.


Click **Add IP Whitelist** and set parameters by referring to the following table. To modify an IP address whitelist, click the  in the row where the IP address whitelist is located.

Table 20-1 Parameters for creating an IP address whitelist

Parameter	Description
IPv4	<p>If you select this option, you can specify an IP address, set an IP address range, or set a route in CIDR format. The differences are as follows:</p> <ul style="list-style-type: none">• IP address: The IP address will be added to the whitelist. For example, you can add the IP address of your personal computer to the whitelist.• IP address segment: If you have multiple servers and the IP address segments are consecutive or your IP addresses dynamically change in a network segment, you can add an IP address segment. Example: 100.*.*.0 - 100.*.*.255.• CIDR: When your server is on a LAN and uses CIDR routing, you can specify a 32-bit egress IP address of the LAN and the number of bits of a specified network prefix. Requests from the same IP address are accepted if the network prefix is the same as the specified one.
IPv6	<p>If you select this option, you can specify an IP address and an IP address range. For details, see IP address and IP address segment.</p>
Description	Optional.
Access Control	<p>Optional. Select the corresponding options as needed.</p> <ul style="list-style-type: none">• Allowed to access the repository: Only whitelisted IP addresses and the repository owner can access the repository.• Allowed to download code: If this option is selected, IP addresses in the whitelist 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

To set an IP address whitelist for all repositories of a tenant, log in to the repository list page of CodeArts Repo, click the alias in the upper right corner, and choose **All Account Settings > Repo > IP Address Whitelist**. The configuration rules are the same as the preceding configuration.

CodeArts Repo Audit Logs

CodeArts Repo allows you to modify repository attributes. It records information such as code commits and merge requests about the code repository. Each audit log contains the operator, operation type, and operation content. You can filter and view audit logs by time.

Adjusting Repository Visibility

CodeArts Repo allows you to adjust the visibility of repositories.

On the CodeArts homepage, click the profile picture and choose **All Account Settings**. In the navigation pane on the left, choose **Repo > Repo Visibility Adjustment**, and click **Adjust** to adjust the visibility of the code repo of a tenant.

- If the page shown in the following figure is displayed, you can create a public code repository (group) and set the visibility of the code repository to public.

Set Public Allowed:You can currently create public repos (groups) and change private repos to public.

 Adjust

- If the page shown in the following figure is displayed, the public code repository (group) cannot be created and the visibility of the code repository cannot be set to public.

Set Public Not Allowed:You can currently only create private repos (groups) and cannot change private repos to public.

 Adjust