CodeArts Repo

User Guide

Issue 01

Date 2025-07-25

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

The following figure shows how to use CodeArts Repo.

Figure 1-1 Process of CodeArts Repo

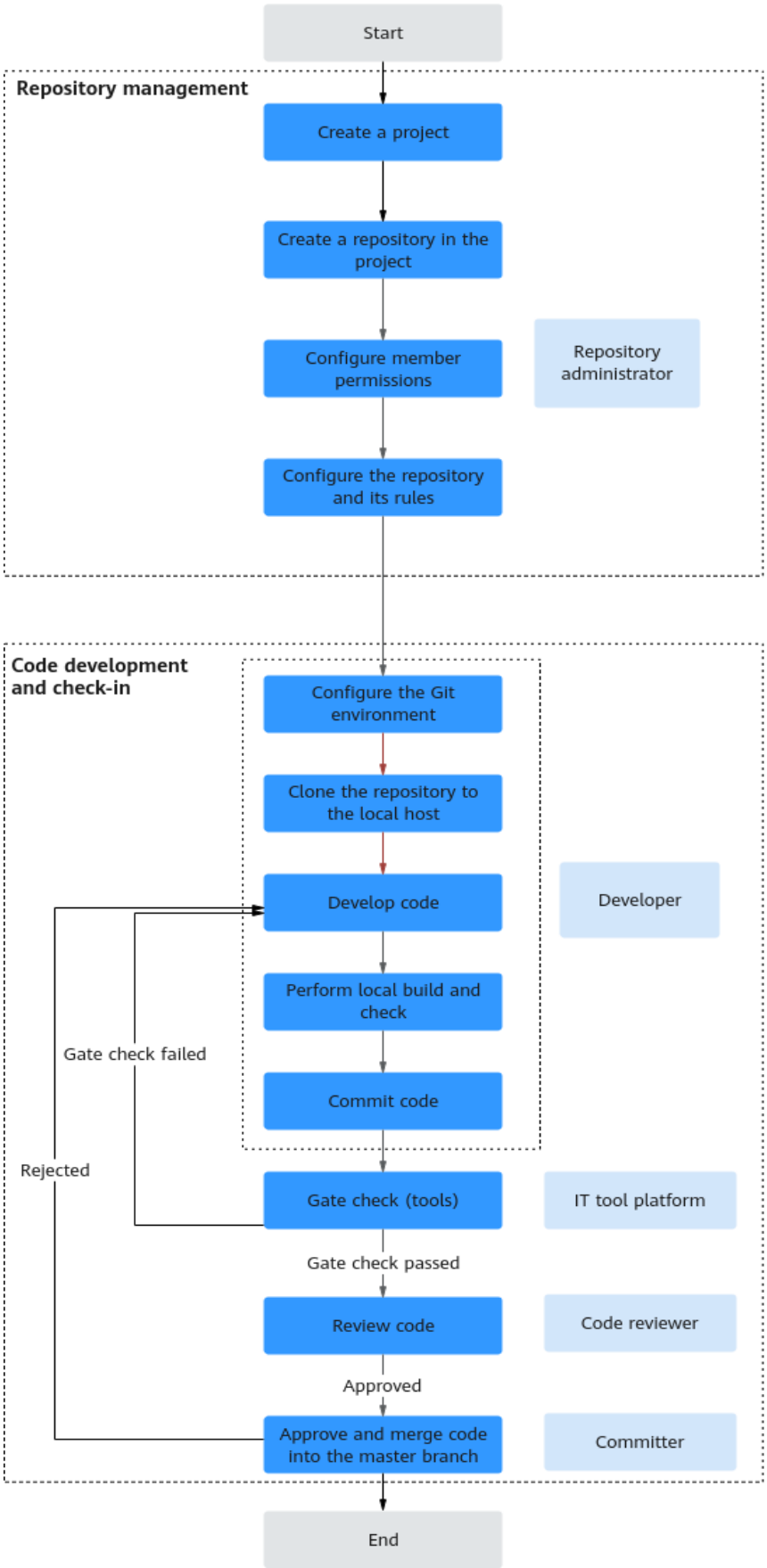


Table 1-1 Usage process

Process	Description
Create and Configure a CodeArts Project	All CodeArts Repo operations must be performed in a project. You need to create a CodeArts project. By doing this, you can create a CodeArts project, add project members, and manage members' permissions to use other CodeArts services.
Create a Repository in a Project	Repository administrators can create repositories. • **Creating a Custom Repository** • **Creating a Repository Using a Template** Repository administrators can also import repositories. Developers can develop code based on the imported repositories. • **Migrating a Git Repository Using a URL** • **Migrating a GitHub Repository** • **Migrating a GitLab Repository** • **Migrating a Self-Built GitLab Repository** • **Migrating a GitHub Repository** • **Migrating a Coding Repository** • **Migrating a Codeup Repository** • **Migrating a Bitbucket Repository** • **Migrating a Gerrit Repository** • **Migrating an SVN Repository**
Configure Member Permissions	• Repository administrators can configure CodeArts Repo permissions for project members of the repository. For details, see **Configuring Project-Level Permissions**. • Repository administrators can configure permissions for repository members. For details, see **Configuring Repo-Level Permissions**. • Project members can be synced to the repository or repository group. For details, see **Syncing Project Members to CodeArts Repo**.

Process	Description
Configure repository rules	CodeArts Repo allows repository administrators to set rules for repositories to ensure code security. Repository rules can be configured at the project level and repository level. • Project-level repository rules are as follows: **Configuring Project-Level Repository Settings**, **Configuring Protected Branch Rules**, **Configuring Protected Tags Rules**, **Configuring Code Commit Rules**, **Configuring Project-Level Merge Request Rules**, **E2E Settings**, **Configuring Webhook Settings**, **Configuring a Deploy Key for a Repository**, **Adding Watermarks to a Repository**, and **Syncing Project Members to CodeArts Repo**. • The repository group rules are as follows: **Repository Group Information**, **Configuring Repository Settings in a Repository Group**, and **Risky Operations**.
Configure the Git environment	Before cloning a repository to the local host for code development, developers need to install and configure the Git client and configure at least one type of key. • **Installing and Configuring Git** • **Configuring an SSH Key** • **Configuring an HTTPS Password** • **Configuring an Access Token** • **Configuring a GPG Public Key**

Process	Description
Clone a repository to a local PC	Before code development, developers need to clone the repository to the local host using an SSH key or HTTPS. • **Using the SSH Key to Clone a Repo to a Local PC** • **Using HTTPS to Clone a Repo to a Local Computer** If repository members do not develop code, they can download the repository to the local host for viewing. For details, see **Using a Browser to Download Code Package to a Local PC**.
Coding	You can clone a repository to the local host for code development. You can also develop code online. For details, see **Editing and Creating a Merge Request**.
Code commit	After developing and checking code on the local host, you can upload the code file to CodeArts Repo. • You can upload code to Repo through Git Bash. For details, see **Creating a Branch and Developing Code in Git Bash**. • You can upload code to Repo through Eclipse. For details, see **Committing Code in Eclipse and Creating a Merge Request**. • You can also use the **git-crypt** command on the Git client to transfer sensitive data to CodeArts Repo to keep your code secure. For details, see **Using git-crypt to Transmit Sensitive Data on the Git Client**. Code conflicts may occur when you upload code. For details, see **Resolving Code Conflicts in an MR**. After the code file is uploaded, you can view the commit history of the code. For details, see **Viewing Commit History**.

Process	Description
Code check	If the repository administrator selects **Automatically create Check task (free of charge)** when creating a repository, a CodeArts Check task will be automatically triggered for code specifications check after the code is committed.
Code review	After the code passes the check, the code reviewer can organize code review. They can provide review comments based on the configuration of the repository administrator. For details, see **Step 1**.
Approve and merge code in to the master branch	If the repository administrator sets the review comment gate, you need to pass the review comment gate by referring to **Passing of the Gate**. Then, the committer can merge the MR.

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 Free edition comes with a fixed number of users and duration. To use this edition, click **Try Free**. You do not need to make any payment.

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 Environment and Personal Settings

3.1 Installing and Configuring Git

Constraints

Table 3-1 Constraints

Item	Description
Security protocol compatibility	CodeArts Repo supports TLS 1.2 and TLS 1.3. If Git is the latest version, you can run the following command to specify the TLS version: `openssl s_client -connect test.com:443 -tls1_2` • **test.com** is the domain name **tls1_2** for Git upload and download in CodeArts Repo. • **tls1_2** indicates TLS 1.2.
CodeArts Repo functions	GitHub Desktop is not supported in CodeArts Repo.

Install Git Client

For details about the clients supported by Repo and the installation guide link, see **Table 3-2**.

Table 3-2 Compatible Git clients

Client Name	OS	Official Installation Guide Link
Git client	Windows	**Windows Git Client Installation Guide**

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

Windows Git client usernames can include letters, numbers, and common symbols. Characters with ASCII values less than or equal to 32, as well as ".", ",", ":", ";", "<", ">", """, "\", and "/" cannot be used at the beginning or end of the username. For easier management, you can use your CodeArts Repo username.

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

```
git config --global user.name your username
git config  --global user.email your_email_address
```

Related Document

For different Git client solutions, see **Compatible TLS Protocol Versions with CodeArts Repo**

3.2 Configuring an SSH Key

Introduction

An SSH key securely connects a local computer with your CodeArts Repo. For details, see **Procedure**.

You can configure an SSH key by referring to **Procedure** or Configuring an SSH Key.

Constraints

If you have more than one computer, you need to configure an SSH key for each so that all accounts on the computer can be connected through the SSH key.

Procedure

You can configure an SSH key by referring to Configuring an SSH Key or the following steps.

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
```

- **-t rsa** indicates that an RSA key is generated.
- **-b 4096** indicates the key length. (RSA keys of this length are more secure.)
- **-C your_email@example.com** indicates that a comment is added to the generated public key file to help identify the purpose of the key pair.

Enter the command for generating a 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.

- 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

Related Document

- If **Failed to add the key. Check whether the key is valid.** is displayed, the entered key is invalid. If the key is manually copied from the local host, redundant spaces may have been copied or the copied key is incomplete. In this case, reconfigure the key by referring to **Step 3**.
- 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.
- CodeArts Repo provides an API to obtain the SSH key list. For details, see **Obtaining the SSH Key List**.
- CodeArts Repo provides an API to obtain the SSH key list. For details, see **Adding an SSH Key**.

- CodeArts Repo provides an API to obtain the SSH key list. For details, see **Adding an SSH Key**.

3.3 Configuring an HTTPS Password

Introduction

An HTTPS password is a user credential used for pulling and pushing code using the HTTPS protocol. 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.

You can configure HTTPS passwords by referring to **Setting the HTTPS Password for the First Time**or **Configuring HTTPS Passwords** in *Best Practices*.

Constraints

Table 3-3 Constraints

Item	Description
Permission	• 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 accounts** cannot be bound to email addresses and the HTTPS protocol cannot be used.
Password format	An HTTPS password consists of two parts: • The HTTPS username consists of your tenant name and IAM username: *Tenant name*|*IAM username*. A tenant name is the account to which an IAM user belongs, that is, a Huawei Cloud account. • The HPPTS password consists of 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.

Item	Description
File size	The size of a single file to be pushed using HTTPS cannot exceed 200 MB. If you need to transfer a file larger than 200 MB, configure an SSH key by referring to **Configuring an SSH Key** and **use the SSH key to transfer the file**.

Setting the HTTPS Password for the First Time

By default, the HTTPS password is your IAM 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 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 **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 **Password** and **Confirm Password**, and click **OK**. A dialog box is displayed, indicating that the password is set successfully.

Step 3 Verify whether the HTTPS password takes effect by referring to **Verifying Whether Your HTTPS Password Takes Effect**. If the HTTPS password does not take effect, rectify the fault by referring to **Related Document**.

----End

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 Password**. Click **Change**, enter the old password, new password, and confirm it. Click **Save**. A dialog box is displayed, indicating that the password is set successfully.

You can also click **Reset**. You need to bind an email address for the first-time password reset. 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 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 Verify whether the HTTPS password takes effect by referring to **Verifying Whether Your HTTPS Password Takes Effect**. If the HTTPS password does not take effect, rectify the fault by referring to **Related Document**.

----End

Verifying Whether Your HTTPS Password Takes Effect

After setting the HTTPS password, you can run the following command in Git Bash to clone the repository that you have permission to access:

```
git clone https://username:password@exaple.com/repo_path.git
```

- **username** is the configured HTTPS username.
- **password** is the configured HTTPS password..
- **example.com/repo_path.git** is the HTTPS address of the repository to be cloned.

  If the code is successfully cloned with the command, the HTTPS password has been successfully set.

  As shown in the following figure, the repository **Test_Repo** is successfully cloned to the local host.

  Figure 3-1 Cloned repository

Related Document

- You can configure HTTPS passwords by referring to **Configuring HTTPS Passwords** in *Best Practices*.
- 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.

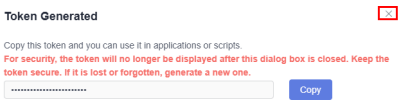
3.4 Configuring an Access Token

Introduction

Access tokens can keep your code resources secure. When you authorize a third party to access your repository, you can create an access token and set its validity period to avoid exposing your account and password. You can view the **Constraints** and configure access tokens by referring to **Configuring an Access Token**.

Constraints

Table 3-4 Constraints

Item	Description
Number of access tokens	A maximum of 20 tokens can be created for CodeArts Repo.
Validity period	One month by default. Max. one year. You can set the validity period of an access token as needed.
Single-time visibility	• As shown in the following figure, after the dialog box is closed, the access token is not displayed. Keep the access token properly. If you lose or forget the access token, you can generate a new one. 

Configuring an Access Token

Log in to CodeArts Repo, go to the repository list page, click your account name in the upper right corner, choose **This Account Settings** > **Repo** > **Access Tokens**, click **New Token**, set parameters according to the following table, save and copy the configured token. For details, see **Examples**.

Table 3-5 Description

Parameter	Description
Token Name	Custom name with a maximum of 200 characters.
Description	Optional. If the description is empty, **--** will be displayed in the list. Max. 200 characters.
Permissions	This parameter is selected by default and cannot be modified. Read/Write repo: Read from and write into repositories using HTTPS.
Expires	Time when a token expires. **The default expiration date is set to 30 days from the current date, including the current day. For example, if a token is created on July 3, the default expiration date will be 23:59:59 on August 2. The expiration date can be set to one year from the creation date and cannot be left empty.**

Related Document

After configuring the token, you can run the following command to clone the repository to the local host:

```
git clone https://private-token:Your Token@example.com/repo_path.git
```

- **Your Token** is the copied token.
- If the HTTPS clone address of the repository is **https://example.com/repo_path.com**, the content following **https://** is **example.com/repo_path.git**.

3.5 Configuring a GPG Public Key

Introduction

GNU Privacy Guard (GPG) is a method used for digital signature and authentication. When you push the local code to CodeArts Repo, a GPG public key ensures trusted sources and code integrity by signing and verifying Git code commits and tags in Git. You can view **Constraints** and configure a GPG public key by referring to **Procedure**.

The procedure for generating and using a GPG public key is as follows:

Step 1 Generate a GPG public key. Generate a key pair consisting of a public key and a private key by referring to **Configuring a GPG Public Key**, and configure the GPG public key to CodeArts Repo. The private key is kept confidential.

Step 2 Sign data. When developers want to sign code commits, they can use their own private keys to sign the code commits.

Step 3 Verify the signature. CodeArts Repo can verify the signature using the public key. If the verification is successful, the data has not been tampered with and comes from the user associated with the GPG public key.

----End

Constraints

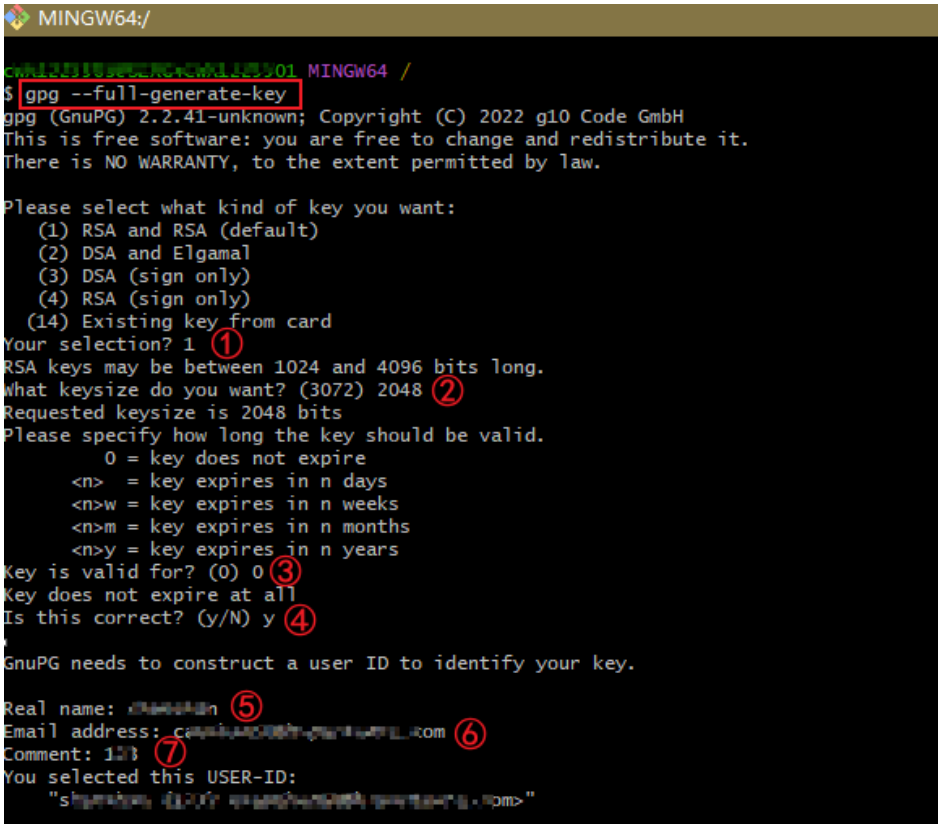
A GPG public key cannot be used repeatedly.

Procedure

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 3-2**.

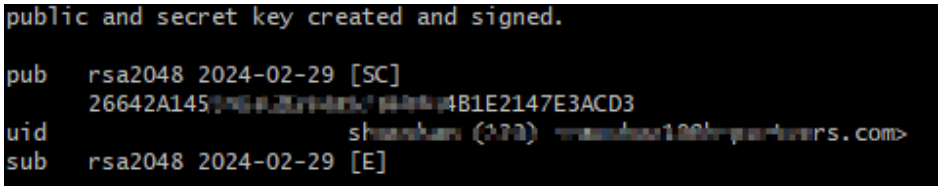
Figure 3-2 Generating a GPG Key Pair



Step 3 After confirming that the information is correct, type **O** and press **Enter** as prompted. In the displayed dialog box for entering and confirming the password, enter the correct password to generate a key.

If the information shown in **Figure 3-3** is displayed, the GPG key is generated successfully.

Figure 3-3 Successful generation of a GPG key

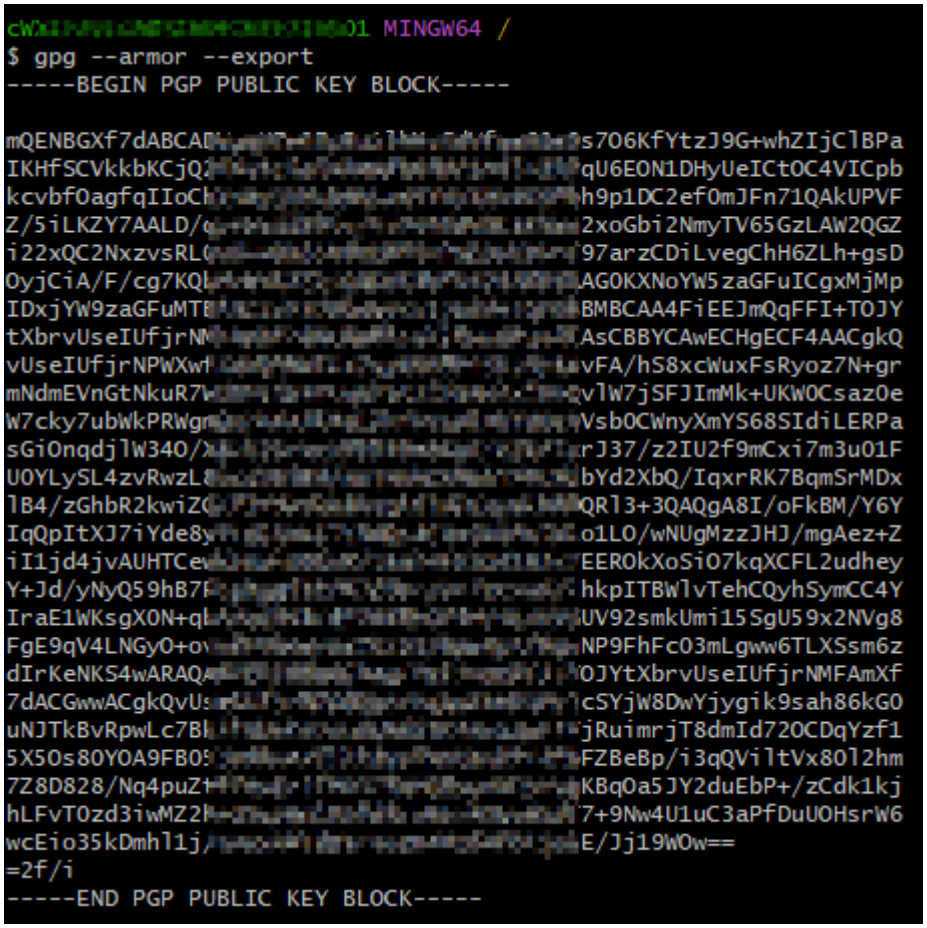


Step 4 Run the following command to export the public key:

```
gpg --armor --export
```

As shown in **Exporting the GPG public key**, copy the public key to the clipboard, including **-----BEGIN PGP PUBLIC KEY BLOCK-----** and **-----END PGP PUBLIC KEY BLOCK-----**.

Figure 3-4 Exporting the GPG public key



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 3-6 Parameters for creating a GPG public key

Parameter	Description
Title	Custom GPG public key name with a maximum of 200 characters.
GPG Public Keys	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**. If the GPG public key is created successfully, the GPG public key list page is displayed. You can delete the GPG public key when you do not need it.

If the public key fails to be added, check whether there are redundant spaces before and after the public key or whether it has already been added.

----End

3.6 Configuring Git LFS

Introduction

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.

Installing Git LFS

Table 3-7 Installing Git LFS on different OSs

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

3.7 Clearing the Repository Memory

When 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.

4 Accessing CodeArts Repo Homepage

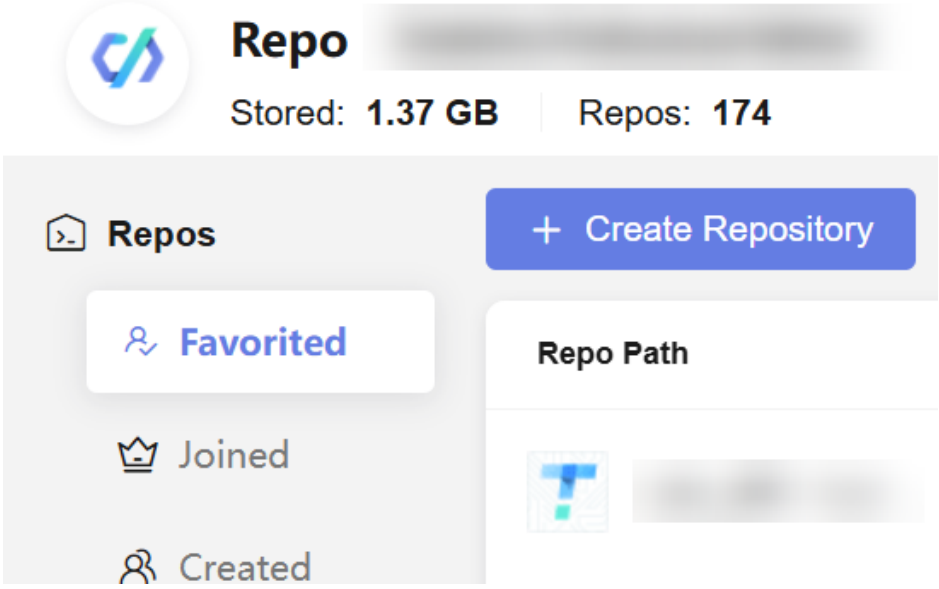
Step 1 **Log in to the Huawei Cloud console** with your Huawei Cloud account.

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

Step 3 View the repository list page of your joined repos or of a specified project.

- **View the list page of your joined repos**

  Click **Access Service**. Go to the CodeArts Repo homepage. This page displays the repos you have favorited.



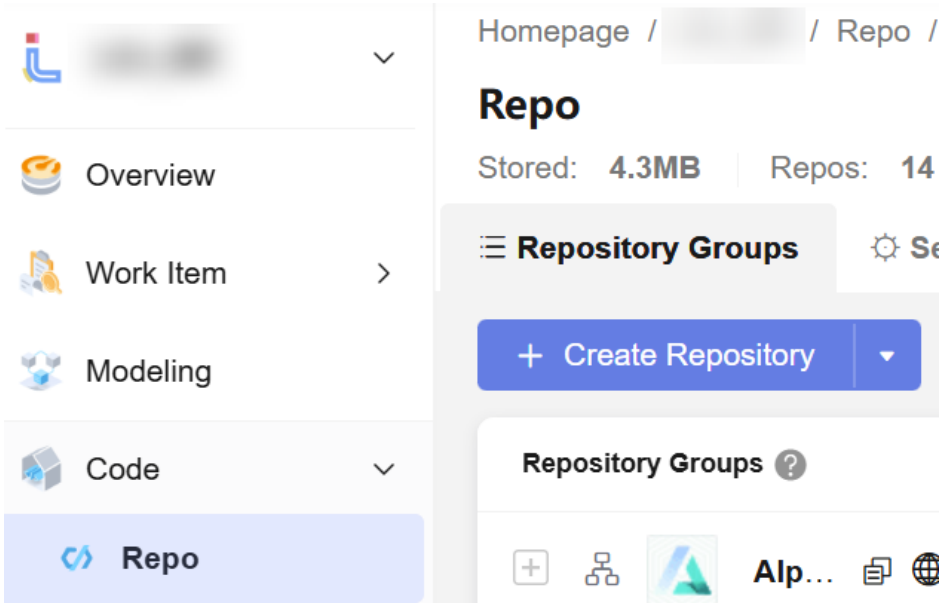
- **View the list page of a specified project**

  a. Click **Access Service**. Go to the CodeArts Repo homepage.

  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

5 Configuring Project-Level Settings for CodeArts Repo

5.1 Configuring Project-Level Repository Settings

Constraints

Project manager or **project administrator** can configure project-level settings. For details, see **Configuring Project-Level Permissions**.

Configuring the Repository Settings

If you want to set the same settings for all repositories in a project, click the project on the CodeArts Repo homepage, and choose **Settings** > **Repository Management** > **Repository Settings**. Set parameters by referring to **Table 5-1**.

Table 5-1 Repository settings description

Parameter	Description	Example Value
Force inherit	Optional. Once selected, all repository groups and repositories in the project use the following settings and cannot be changed. Exercise caution when selecting this option.	Select **Force inherit**.

Parameter	Description	Example Value
Do not fork a repository	Optional. Once selected, no one can fork the repo in the project.	Select **Do not fork a repository**. After this parameter is set, access any repository in the project and click **Fork** in the upper right corner. A dialog box is displayed in the upper right corner.
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.	Select **Pre-merge**. For details, see **Example**.
Branch Name Rule	Optional. After you enter a rule, all branch names must match the regular expression with max. 500 characters. If this field is left blank, any branch name is allowed. You must meet the following tag naming rules: • Max. 500 characters. • Do not start with **-**, **refs/heads/** or **refs/remotes/** nor end with **. / .lock**. Spaces and the following characters are not supported:. [\ < ~ ^: ? ( ) ' " |. • Must be unique.	**Branch_[a-zA-Z0-9_]+**: indicates that all branch names in the project must start with **Branch_** and can match any combination of uppercase or lowercase letters, digits, and underscores (_).

Parameter	Description	Example Value
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. • Max. 500 characters. • Do not start with **-**, **refs/heads/** or **refs/remotes/** nor end with **. / .lock**. Spaces and the following characters are not supported:. [\ < ~ ^: ? ( ) ' " |. • The tag name must be unique.	**Tag_[a-zA-Z0-9_]+**: indicates that all tag names must start with **Tag_** and can match any combination of uppercase or lowercase letters, digits, and underscores (_).

Related Document

- If the error message indicating a branch name does not match the rule is displayed when a branch is created, the new branch name **Test_Branch** does not match the branch name rule **Branch_**. Choose **Settings** > **Repository Management** > **Repository Settings** to view the latest branch name rule and create a branch based on it.
- If the error message indicating a tag name does not match the rule is displayed when a tag is created, the new tag name **Test_Tag** does not match the tag name rule **Tag_**. Choose **Settings** > **Repository Management** > **Repository Settings** to view the latest tag name rule and create a tag based on it.

5.2 Configuring Protected Branch Rules

Protected Branch Rules Overview

CodeArts Repo makes a code branch 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.

Constraints

Project manager or **project administrator** can configure project-level protected branch rules. For details, see **Configuring Project-Level Permissions**.

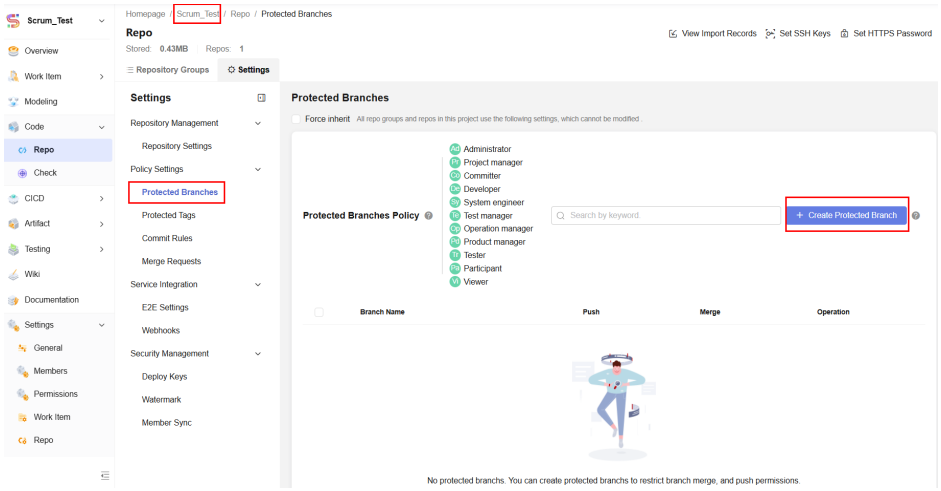
Configuring Protected Branch Rules

The process is as follows:

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

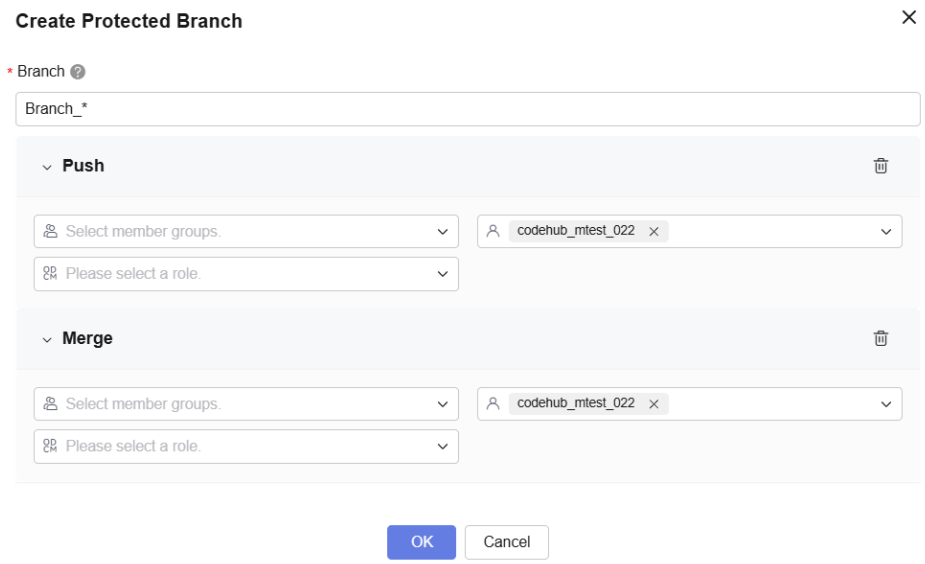
Step 1 On the CodeArts Repo homepage, access the **Scrum_Test** project, choose **Settings** > **Policy Settings** > **Protected Branches**, and click **Create Protected Branch**, as shown in the following figure.

Figure 5-1 Project-level protected branch page



Step 2 As shown in the following figure, **Branch_*** indicates that all branches starting with **Branch_** are protected branches, and only **codehub_mtest_022** can be **pushed** and **merged**.

If a branch contains a single slash (/), the branch cannot be matched using the wildcard * due to the fnmatch syntax rule.



----End

Related Operations

When you commit a code file, if an error message indicating you have no permission to commit to the target protected branch (DEV-23-8040) is displayed, the branch you are committing is a protected branch and you need to contact an administrator to **grant you the commit permission**.

5.3 Configuring Commit Rules

Introduction

CodeArts Repo supports verification and restriction rules for high-quality code commits. This section describes how to configure project-level commit rules and use common regular expressions.

Before configuring commit rules, check **Constraints**.

If you have the configuration permission, you can set the same commit rules for all repositories in a project based on **Table 5-2**. When configuring rules, you can refer to **Examples of Common Regular Expression**.

Constraints

Project manager or **project administrator** can set project-level commit rules.

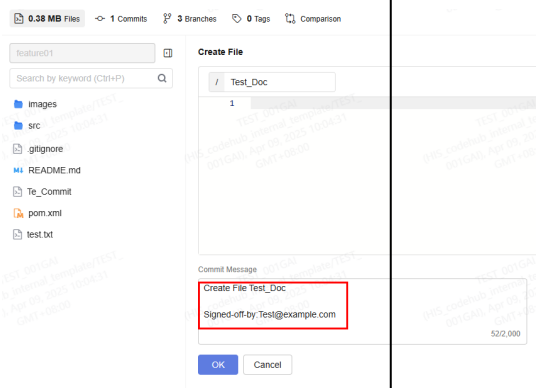
Configuring Project-Level Commit Rules

On the CodeArts Repo homepage, go to a project, and choose **Settings** > **Policy Settings** > **Commit Rules**.

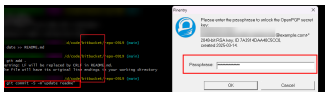
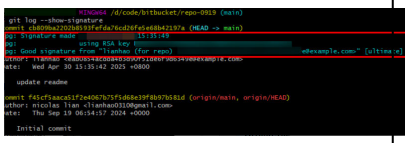
If you want to set the same commit rules for all repositories in a project, set parameters based on **Table 1** and select **Force inherit**.

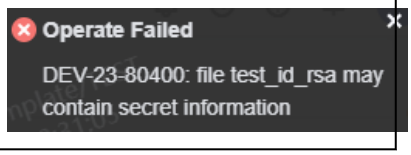
Table 5-2 Commit rules description

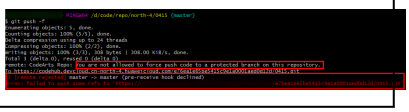
Parameter	Description	Example Value

Reject non-signed-off-by commits	This function records the developer responsible for code modifications, confirming their verification and accountability for the submitted content. This is a transparent way to track contributors and safeguard the quality of contributions.
	If this option is selected, only signed-off-by commits can be pushed to the repository.
	Currently, the following two modes are supported:
	• Online:
	When performing online commit in the CodeArts Repo, use the following format to compile and commit information:
	`commit message # Enter the customized commit information.`
	`# This is a blank line.`
	`Signed-off-by: User-defined signature # Enter the user-defined signature after Signed-off-by:`
	• **Git client**
	When committing code on the Git client, run the following command. **-s** indicates that the signature (Signed-off-by) is added to the commit message. You need to configure the signature and email address on the client in advance by referring to **Related Operations**.
	`git commit -s -m "<your_commit_message>"`
		Refer to the following example if this option is selected:
		• Online:
		When submitting a file online, you need to enter a commit message in the format shown in the following figure. Otherwise, an error message indicating non-signed-off commits will be rejected will be displayed.
		 
		• Git client:
		Run the following command on the local Git client to submit the change to the version control system. The commit message is **New File**.
		`git commit -s -m "New File"`
		You can run the **git show** command to check the signature result.
		 

Parameter	Description	Example Value

Reject commits not signed by GPG	If this option is selected, only commits signed by GPG can be pushed to the repository. For details, see **Configuring a GPG Public Key**.
	Git client signature mode:
	When committing code on the Git client, you need to add the **-S** parameter, indicating that the GPG (GNU Privacy Guard) signature is used to verify the identity of the committer.
	`git commit -s -m "your commit message"`
	When using tags on the Git client, you need to add the **-s** parameter, indicating that the GPG signature is used to verify the authenticity of the tag.
	`git tag -s -m "your tag message"`
		After this option is selected, run the following command and the commit message is **update readme**.
		`git commit -S -m "update readme"`
		After the command is executed successfully, you'll be instructed to enter the password of the GPG key, as shown in the following figure.
		 
		As shown in the following figure, you can run the following command to view the GPG signature submission records.
		 

Tags cannot be deleted	After this option is selected, tags cannot be deleted online or by running commands on the client.	After this option is selected, an error message indicating tags cannot be deleted is displayed when you delete any tag. .
Prevent committing secrets	The confidential file names include **id_rsa** and **id_dsa**. After this option is selected, an error is reported if the name of a new file submitted online or locally contains **rsa**, **id_rsa**, **dsa**, or **id_dsa**.	After this option is selected, if the name of a new file is **test_id_rsa**, an error message indicating the file may contain sensitive information is displayed. 

Parameter	Description	Example Value
Prevent **git push -f**	Configure whether to use the **git push -f** command on the client to commit code. It is recommended to enable this considering the **git push -f** command is used to push your local code repository to CodeArts Repo.	After this option is enabled, the following error message is displayed when you run the **git push -f** command locally: "CodeArts Repo: You are not allowed to force push code to a protected branch on this repository." 

If you want to set commit rules for a specified branch of a repository in a project, click **Create Commit Rule**. For details about the parameters, see **Table 5-3**.

Table 5-3 Parameters for commit rules

Parameter	Description	Example Value
Rule Name	Customize a rule name.	Test_Rule
Branch	Enter a complete rule name or create a regular expression. The input needs to be verified, including the branch name and regular expression.	**Branch_*** indicates that all branch names must start with **Branch_**.

Parameter	Description	Example Value
Commit Rule	Optional. • **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 commit message (max. 500 characters) must contain the work item number to implement E2E code tracing. • **Commit Message Negative Match**: This parameter is empty by default. If left blank, all messages can be committed. Every commit message (max. 500 characters) that matches the regex provided will be rejected. • **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**. • **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.	• **Commit Message Match**: **\d+\..***. **\d** indicates any decimal digit. **+** indicates that **\d** can appear once or multiple times. **.** indicates any single character except the newline character. **\.** indicates matching **.**. That is, the regular expression matches a string that starts with one or more digits, followed by a period (.) and then any number of other characters (except newline characters). For example, the regular expression can match strings in the format of **123.** and **456.abc**. Only commit messages that meet this match rule can be submitted. • **Commit Message Negative Match**: Leave this parameter blank. • **Commit Author**: **/([a-zA-Z]d){7}/**, indicating that a string consisting of a letter followed by a digit has appeared for seven consecutive times. For example, **a1b2c3d4e5f6g7d8** complies with the regular expression. • **Commit Author's Email**: **@my-company.com$**, indicating that any string ending with **@my-company.com** is matched. For example, **Test@my-company.com** complies with the regular expression. For more regular expression rules, see **Examples of Common Regular Expression**.

Parameter	Description	Example Value
	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.	
Basic Attributes	Optional. • **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. • **Single File Size (MB)**: If the size of the added or updated file exceeds the default value, the push will be rejected. You can change the default value.	• **File Name That Cannot Be Changed**: **(\.jar|\.exe)$**, indicating that any file name ending with **.jar** or **.exe** is matched. • **Single File Size (MB)**: 50.

Parameter	Description	Example Value
Binary Rules	Optional. To ensure repository performance, you are advised to select **Do not allow new binary files (privileged users excepted)**. 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 (privileged users excepted)** 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. • **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.) If a privileged user is no longer a repository member, an error message **Failed to verify the privileged user** is displayed when you click **Save**. Remove the privileged user who is not a repository member and save the settings.	Example: Select **Do not allow new binary files (privileged users excepted)**.
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.	Example: April 15, 2025

Examples of Common Regular Expression

Common regular expression examples are listed below.

Table 5-4 Examples

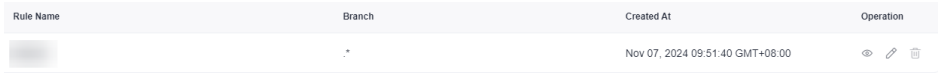
Rule	Example
Single a, b, or c	[abc]
Characters other than a, b, or c	[^abc]
Lowercase letters ranging from a to z	[a-z]
Characters outside the range of a to z	[^a-z]
Uppercase and lowercase letters in the range of a to z or A to Z	[a-zA-Z]
Any single character	.
Match a or b.	a|b
Any blank character	\s
Non-blank characters	\S
Arabic numeral characters	\d
Non-Arabic numeral characters	\D
Letters, digits, or underscores (_)	\w
Characters other than letters, digits, or underscores (_)	\W
Match the content in parentheses (not capture)	(?:...)
Match and capture the content in parentheses	(...)
No or one a	a?
No or more a's	a*
One or more a's	a+
Three a's	a{3}
More than three a's	a{3,}
3 to 6 a's	a{3,6}
Beginning of text	^
End of text	$
Word boundary	\b

Rule	Example
Non-word boundary	\B
Line breaker	\n
Carriage return character	\r
Tab key	\t
Null string	\0

Managing Commit Rules

Click in the row where the target commit rule is located to view details, as shown in the following figure. Click to modify the rule and save it again. Click to delete the commit rule.

Figure 5-2 Commit rule details



Related Operations

- Run the following command to configure a global username in Git:

```
git config --global user.name your name
```

- Run the following command to set a global email address in Git:

```
git config --global user.email your email address
```

5.4 Configuring Project-Level Merge Request Rules

Introduction

The merge request rules refer to the configuration of code merge conditions and modes in MR mode. Project-level MR rules can be inherited to the repositories and repository groups.

Before configuring a merge request rule, check **Constraints**. If you have the configuration permission, configure the rule based on **Configuring Merge Request Rules**.

Constraints

Table 5-5 Constraints on setting merge request rules

Item	Description
Permission	**Project manager** or **project administrator** can set project-level webhooks. For details, see **Configuring Project-Level Permissions**.

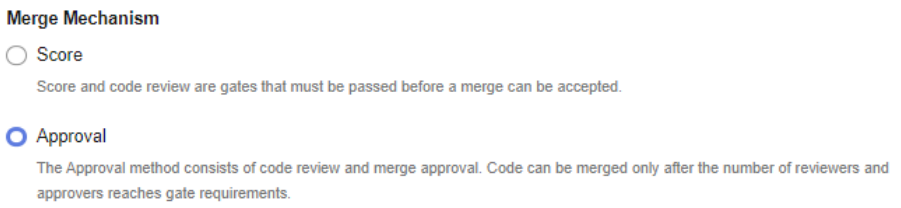
Configuring Merge Request Rules

You can select **Inherit from project**. The settings in the project are automatically inherited and cannot be changed.

You can also access the target project homepage of the code to be configured and choose **Settings** > **Policy Settings** > **Merge Requests**. You can create a merge request rule by referring to the following steps:

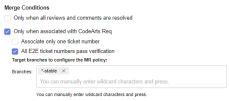
Step 1 Select a merge mechanism. There are two MR mechanisms: score and approval. The differences between the two mechanisms are as follows:

- Scoring mechanism: This mode includes only code review and is based on scoring (0—5). 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.

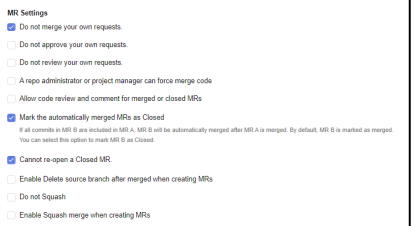


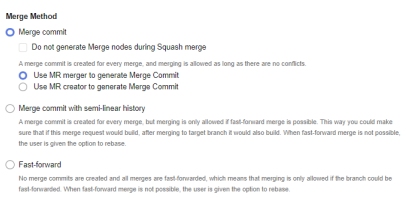
Step 2 After selecting a mechanism, set other parameters by referring to the **following table**. The configuration takes effect for the entire code repo.

Table 5-6 Parameters for setting the merge condition, settings, and mode

Parameter	Description	Example Value
Merge Conditions	Optional. There are two options: • 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. • Must be associated with CodeArts Req The options are as follows: 1. Associate only 1 ticket number. After this option is selected, one MR can be associated with only one ticket number. 2. All E2E ticket numbers pass verification After this option is selected, all associated E2E ticket numbers must pass the verification. 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**. **CAUTION** The **Use score as a merge gate** parameter is available for the Pro edition. • Use score as a merge gate. By default, the project manager, repository owner,	***-stable** indicates that the E2E trouble ticket number ending with **-stable** is matched. 

Parameter	Description	Example Value
	and project administrator can evaluate the MR. You can also add other roles. For permission configuration, see **Configuring Project-Level Permissions**. Selecting this option means that a committer, project manager, repository owner, or project administrator must evaluate the MR. 	

Parameter	Description	Example Value
MR Settings	Optional. Options: • 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 • Do not review your own requests • A repo administrator or project manager can force merge code • Allow code review and comment for merged or closed MRs • 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**.) • Cannot re-open a Closed MR This function is enabled by default. You can enable or disable it as required. • Enable "Delete source branch after merged" when creating MRs • Forbid squash merge (Forbid to select squash merge when creating a merge request) • Enable Squash merge for new MRs	

Parameter	Description	Example Value
Merge Method	Mandatory. The options are as follows: • 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. – **Do not generate Merge nodes during Squash merge**: If this parameter is selected, no merge node is generated during the squash merging. – **Use MR merger to generate Merge Commit**: If this parameter is selected, the commit information is recorded. **Use MR creator to generate Merge Commit**: If this parameter is selected, the commit information is recorded. • Merge commit with semi-linear history. 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	

Parameter	Description	Example Value
	branch can be correctly constructed after the merge is complete. • 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 possible, the user is given the option to rebase.	

Step 3 After setting the parameters, click **Submit** to save the configurations.

----End

If your merge mechanism is **Approval** and you want to set the merge policy for a specified branch or all branches in the repository, refer to **this section**.

Configuring Branch Policy

Go to the target repository homepage, choose **Settings** > **Policy Settings** > **Merge Requests**. In the **Branch Policies** area, click **Create Branch Policy**, and set parameters by referring to **the following table**.

Table 5-7 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.

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

Setting Branch Policies

1. The following is an example of the branch policy priority:
   - If a branch has both policies A and B in a repository, the latest created branch policy is used by default.
   - 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 policy B is used by default.
2. After the merger, approver, and reviewer are configured for a branch for the first time, the data is automatically loaded from the browser cache when a merge request is created. If the names of the merger, approver, and reviewer are changed and their repository permission is removed, a message is displayed on the page for creating a merge request, indicating the current username is inconsistent with that in the list of merger, approver, and reviewer. In this case, you only need to remove the user as prompted to update the cache data. The user will not be displayed in the subsequent merge request.

5.5 E2E Settings

Introduction

E2E settings in CodeArts Repo help you record the reasons for each code merge, including development requirements, trouble tickets, and completed work items. CodeArts Repo records the association information for future tracing.

Before configuring E2E settings, view **Constraints**. If you have the E2E setting permission, perform the configuration by referring to **Integrated Systems**,

Integration Policies, and **Automatic ID Rules Extraction**. You can also perform the configuration by referring to **E2E Settings Example**.

Constraints

Table 5-8 Constraints

Item	Description
Function constraint	The repositories of Kanban projects do not support E2E settings.
Permission constraint	**Project manager** or **project administrator** can set project-level webhooks. For details, see **Configuring Project-Level Permissions**.

Integrated Systems

CodeArts Repo integrates with CodeArts Req for seamless requirement management, allowing you to associate code commits, branches, and merge requests with work items. Association is enabled by default.

Work item types that can be associated with CodeArts Repo are listed in the following table.

Table 5-9 Parameters for associating work items with CodeArts Repo

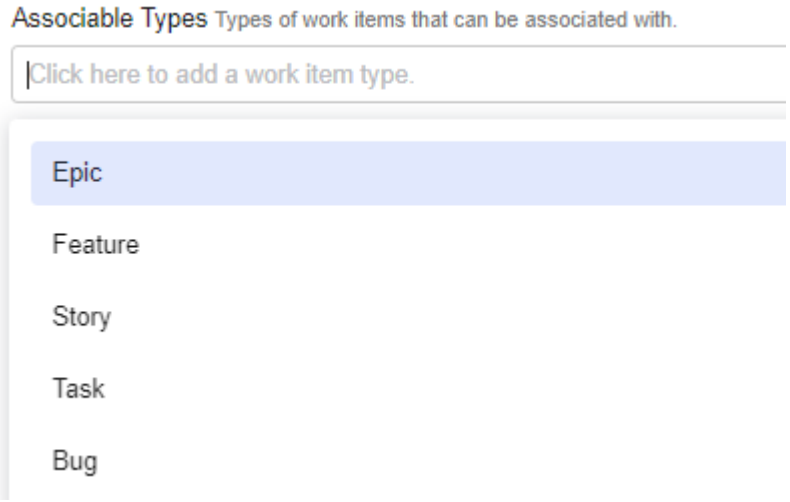
Project Type	Supported Work Items
Scrum	Epic, feature, story, task, and bug
IPD system device	Initial requirement (IR), system requirement (SR), allocated requirement (AR), and bug
IPD standalone software	IR, user story (US), task, and bug
IPD-self-operated software/cloud service project template	Epic, feature (FE), US, task, and bug

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. For example, the picture shows work items associable with a Scrum project. If you select **New**, it means work items in this state cannot be associated with MRs.

Associable Types: Types of work items that can be associated with. For example, if you select **Epic**, work items of this type can be associated with MRs. For details, see **Table 5-9**.



Apply to: Only the selected branch can be used. For example, if the regular expression is **Branch_***, the merge requests of all branches starting with **Branch_** comply with the settings of **Excluded States** and **Associable Types**. For details about the regular expression rules, see **Examples of Common Regular Expression**.

Automatic ID Rules Extraction

Constraints

- The prefix, suffix, and separator cannot contain each other. Otherwise, the extraction effect may be unsatisfying.
- If **Separator** is left empty, the values of **ID Prefix** and **ID Suffix** cannot be semicolon (;).
- 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.

Procedure:

Automatic ID extraction rules can automatically extract ID based on code commit messages. For details, see **the following table**.

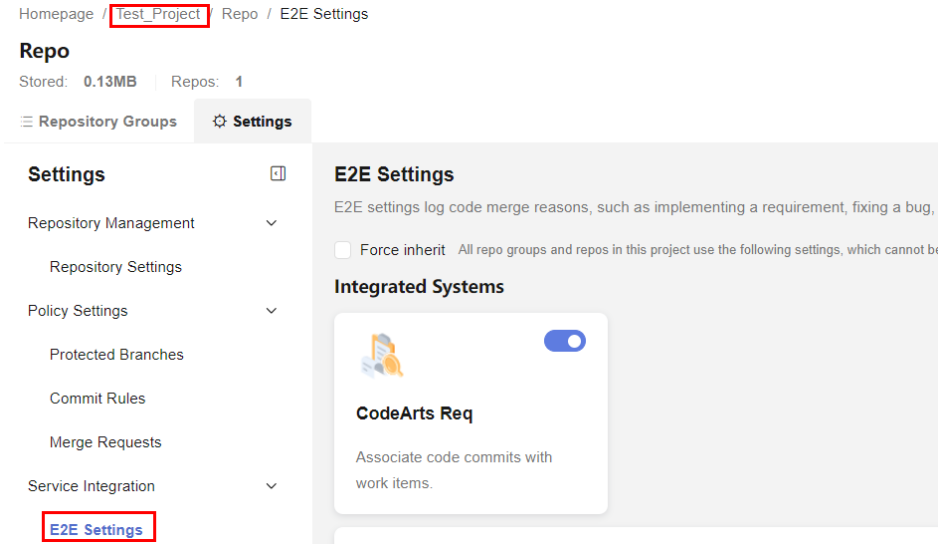
Table 5-10 Parameters for automatic ID extraction rules

Parameter	Description
ID Prefix	Optional. Max. 10 prefixes, with each prefix containing up to 200 characters.
Separator	Optional. The default value is **;**.
ID Suffix	Optional. Line breaks are used by default.

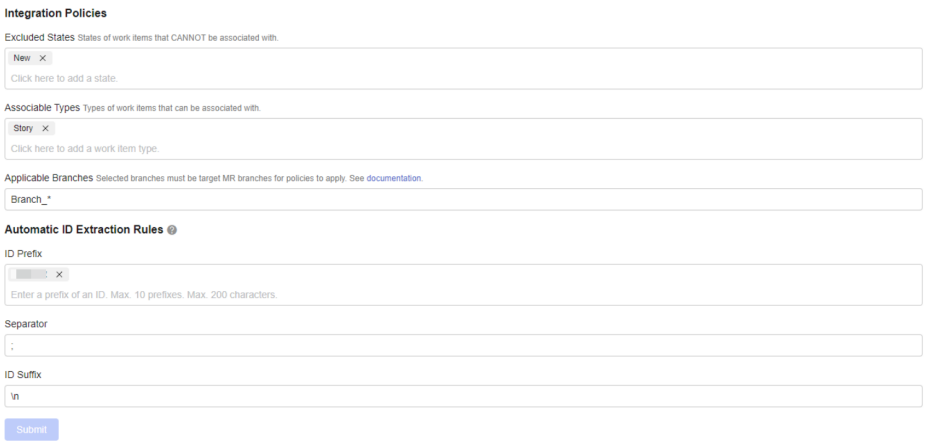
E2E Settings Example

This example uses a Scrum project.

Step 1 Go to a project to be configured and choose **Settings** > **Service Integration** > **E2E Settings**. The **E2E Settings** page is displayed.

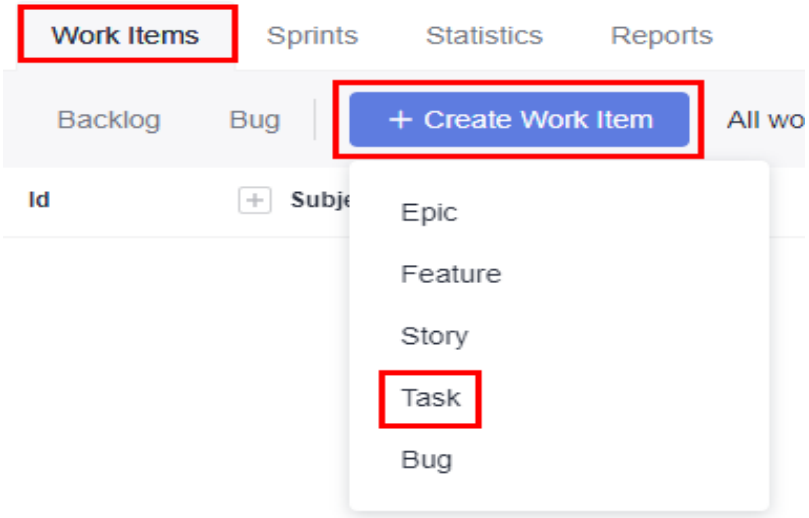


Step 2 Set the parameters as shown in the following figure. Set **Excluded States** to **New**, **Associable Types** to **Story**, **Applicable Branches** to **Branch_***, **ID Prefix** to **Requirement**, **Separator** to **;**, and **ID Suffix** to **\n**.



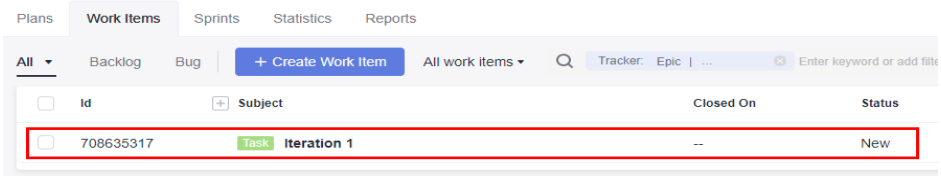
Step 3 Create a work item.

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



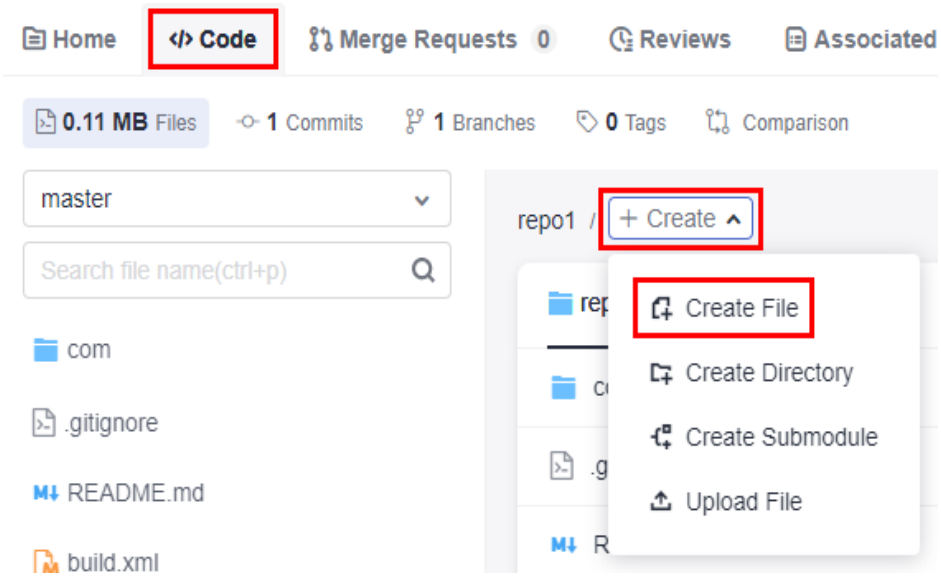
3. Enter a title, for example, **Sprint 1**.

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



Step 4 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.

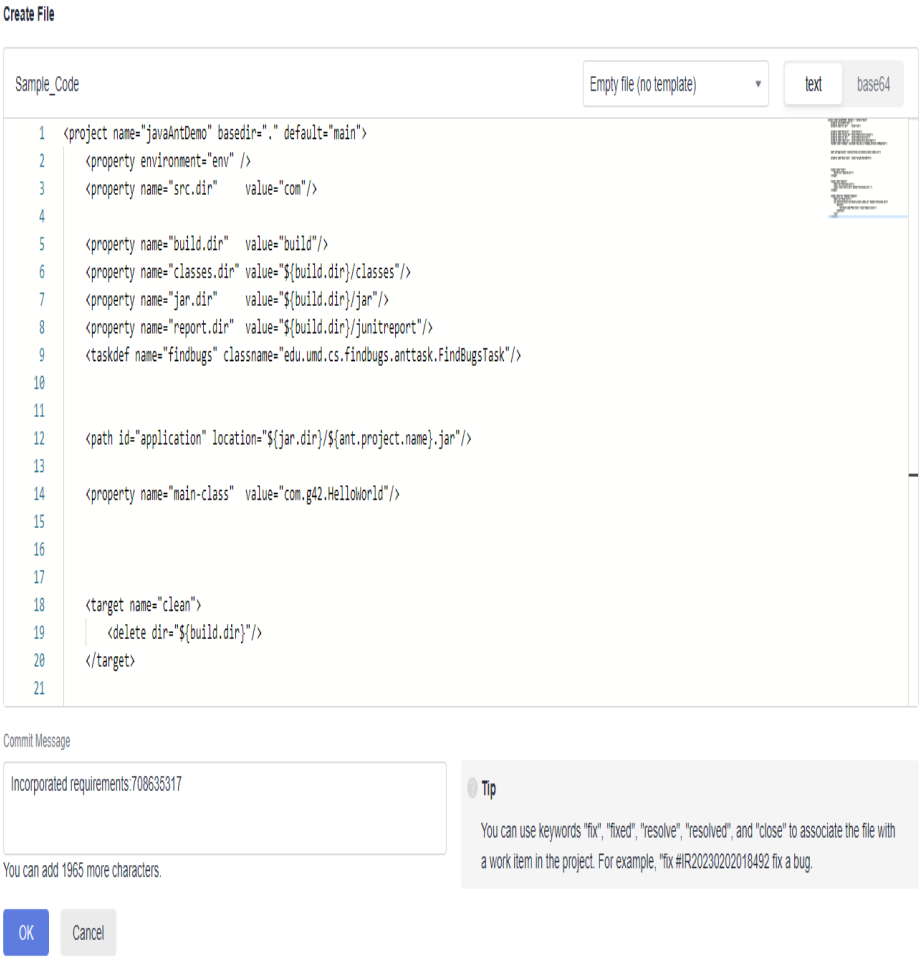


3. Enter the following information, retain the default values for other parameters, and click **OK**.

   File name: user-defined file name, for example, **Sample_Code**.

   File content: user-defined file content.

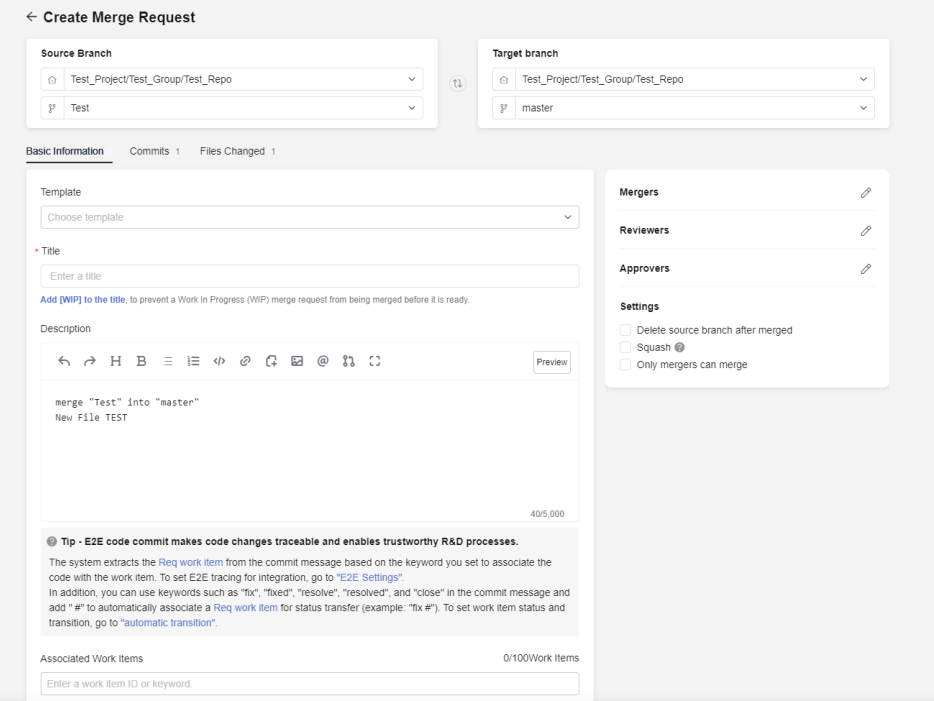
   Commit message: Enter the prefix and work item number in the E2E settings, for example, 708635317.



Step 5 Extract the ticket number when creating a merge request.

1. Switch to the **Merge Requests** tab and click **New**.
2. Set the source branch to **Dev** and the target branch to **master**. The Pro and Enterprise editions have the **Templates** function.

   At this point, the work item is automatically extracted to the merge request.



----End

5.6 Configuring Webhook Settings

Overview

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 subscribed event occurs, you can use a webhook to send a POST request to the URL of a third-party system to trigger corresponding operations (in the third-party system), such as popping up a notification window, building or updating images, or performing deployment.

Before configuring a merge request rule, check **Constraints**. If you have the configuration permission, configure the rule based on **Configuring Webhook Settings**. For details about the configuration example, see **Related Document**.

Constraints

Table 5-11 Constraints

Item	Description
Permission settings	**Project manager** or **project administrator** can set project-level webhooks. For details, see **Configuring Project-Level Permissions**.

Item	Description
Function constraint	A maximum of 20 webhooks can be created for a repository.

Configuring Webhook Settings

To configure webhooks, choose **Settings > Service Integration > Webhooks** on the repository details page. You can set parameters by referring to the following figure or table. For details about the configuration result, see **Related Document**.

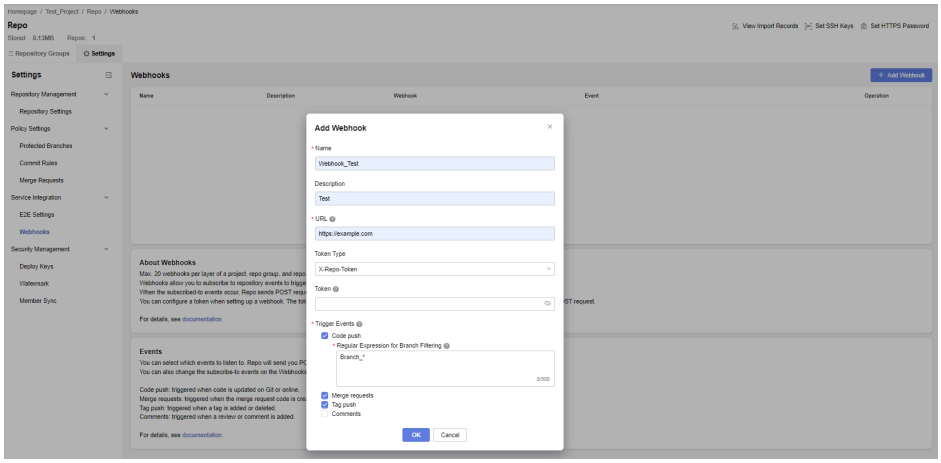


Table 5-12 Parameters for creating a webhook

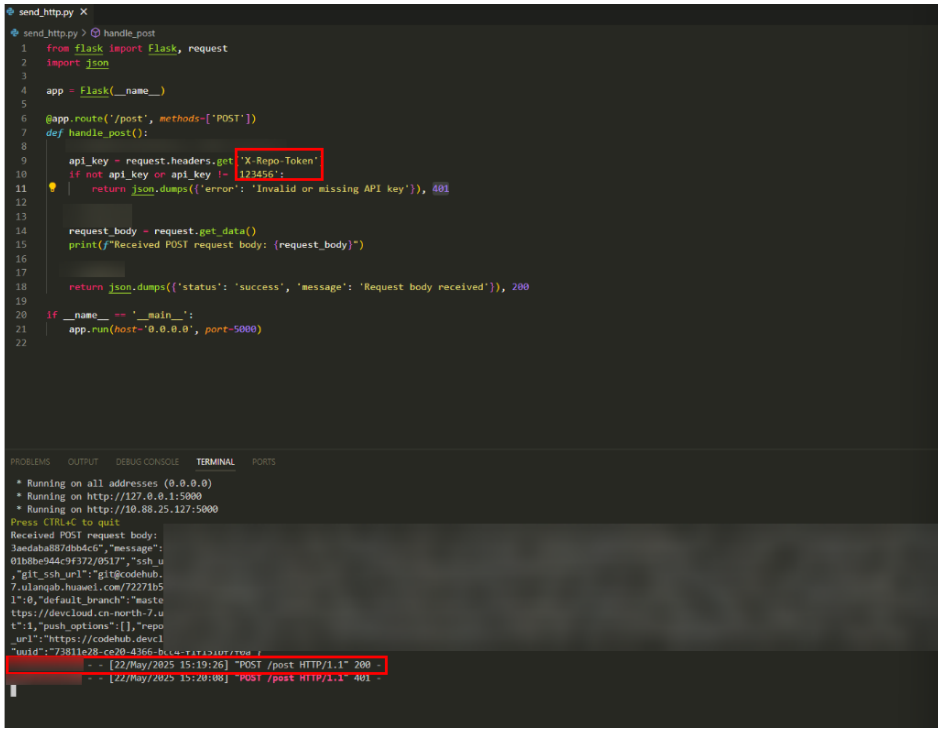
Parameter	Description	Example Value
Name	Custom name with up to 200 characters.	Webhook_Test
Description	Optional. Used to describe the webhook with up to 200 characters.	Test
URL	The URL and its response cannot contain sensitive data or the sensitive data must be encrypted and decrypted for security. Provided by the third-party CI/CD system.	https://example.com

Parameter	Description	Example Value
Token Type	Optional. Used for authentication of third-party services' webhook APIs. The authentication information should be placed in the HTTP request header as the key and used together with the token. Three options are available: • X-Repo-Token • X-Gitlab-Token • X-Auth-Token You can also enter a custom token type. 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.	X-Repo-Token
Token	Optional. After you set **Token Type**, this parameter is used as the value and is used together with the token type. Enter a custom value, with max. 2,000 characters. Used for third-party CI/CD system authentication. The authentication information is stored in the HTTP request header.	Token_Test

Parameter	Description	Example Value
Trigger Events	You can subscribe to the following events: • **Code push** After this option is selected, you need to enter the **Regular Expression for Branch Filtering** with up to 500 characters. **.*** indicates that all branches are matched. If the branch name matches the configured regular expression and the code is updated (code updates in sub-modules; online or offline code push in Git client), the event will be triggered. • **Merge requests** After this option is selected, updating the MR code, closing MRs, re-opening MRs, updating MR title or description, updating the merger and the work item, deleting source branches, or updating the Squash merge will trigger this event. • **Tag push** After this option is selected, both creating and deleting a tag will trigger this event. • **Comments** After this option is selected, this event will be triggered when a review is added to a code file, a file change in a code commit, or an MR, or when a comment is added to a repository's commit details or MR details.	• Select **Code Push** and set **Regular Expression for Branch Filtering** to **Branch_***. This indicates that the push of all branches starting with **Branch_** will trigger this event. • Select **Merge requests**. • Select **Tag push**.

Related Document

- After the configuration is complete, the webhook of push event is triggered. As shown in the following figure, **200** is returned. If the token value in the code example is different from that configured in the table, the authentication fails and the error code 401 is returned.

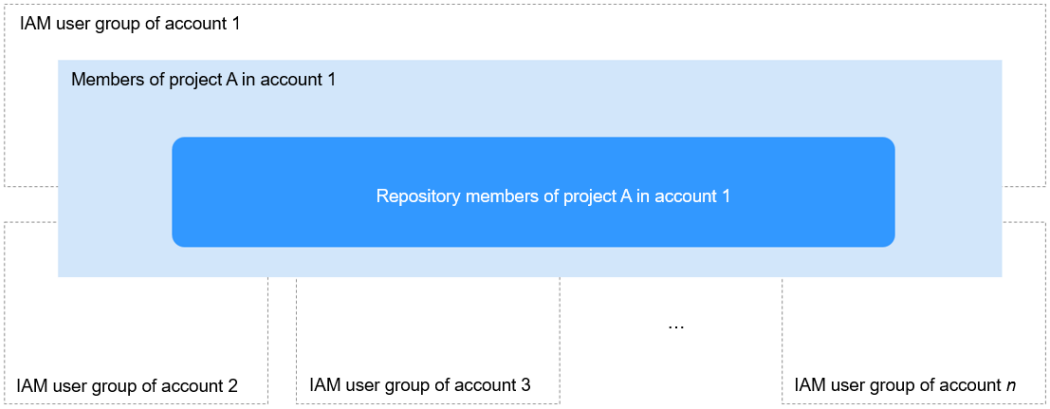


6 Managing Member Permissions

6.1 IAM Users, Project Members, and Repository Members

Relationships Between IAM Users, Project Members, and Repository Members

Repository members come from 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.



By default, project roles are mapped to repository roles. You can change the repository role of a member in the repository as needed.

By default, the project creator is both the project administrator and repo administrator.

6.2 Configuring Project-Level Permissions

Constraints

- The project manager and other users with management permissions can modify the default operation permissions of different roles in the project on this page.
- If the repository's visibility is set to read-only for project members, then all project members can view other resources except for the settings.
- If the repository's visibility is set to read-only for tenant members, all tenant members can view other resources except for the settings.
- If the repository's visibility is set to read-only for all visitors, then all visitors can view all resources except for the settings.

Configuring Project-Level Permissions

Step 1 Log in to the CodeArts Repo homepage. In the navigation pane on the left, choose **Settings** > **General** > **Permissions**. The **Permissions** page is displayed.

Step 2 Select the corresponding **Role** > **CodeArts Repo**, and click **Edit** to configure permissions. 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 a new role does not have the copied permissions of any existing role, the new role does not have any permissions. However, you can add permissions for a custom role as required, as shown in the following table.

A indicates that the role has the permission by default and cannot be removed. **B** indicates that the role has the permission by default and can be removed. **C** indicates that the role can be assigned with the permission. **D** indicates that the role cannot be assigned with the permission.

----End

Table 6-1 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

Role / Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Custom Role
Branch	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
Code	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
Repository group	Delete	B	D	D	D	D	D	D	D	D	D	C
Repository group	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
Member	Edit	B	D	D	D	D	D	D	D	D	D	C
Member	Delete	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
MR	Create	B	C	C	C	B	B	B	C	C	D	C
MR	Edit	B	D	D	D	C	B	C	D	D	D	C
MR	Comment	B	C	C	C	B	B	B	C	C	C	C
MR	Review	B	D	D	D	B	B	B	D	D	C	C
MR	Approve	B	D	D	D	C	B	C	D	D	D	C
MR	Merge	B	D	D	D	C	B	C	D	D	D	C
MR	Close	B	D	D	D	C	B	C	D	D	D	C
MR	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
Repository	fork(MR)	B	C	C	C	B	B	B	C	C	D	C
Repository	Delete	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
Repository	Set	B	D	D	D	D	D	D	D	D	D	C
Tag	Create	B	C	C	C	B	B	B	C	C	D	C
Tag	Delete	B	C	C	C	C	C	C	C	C	D	C

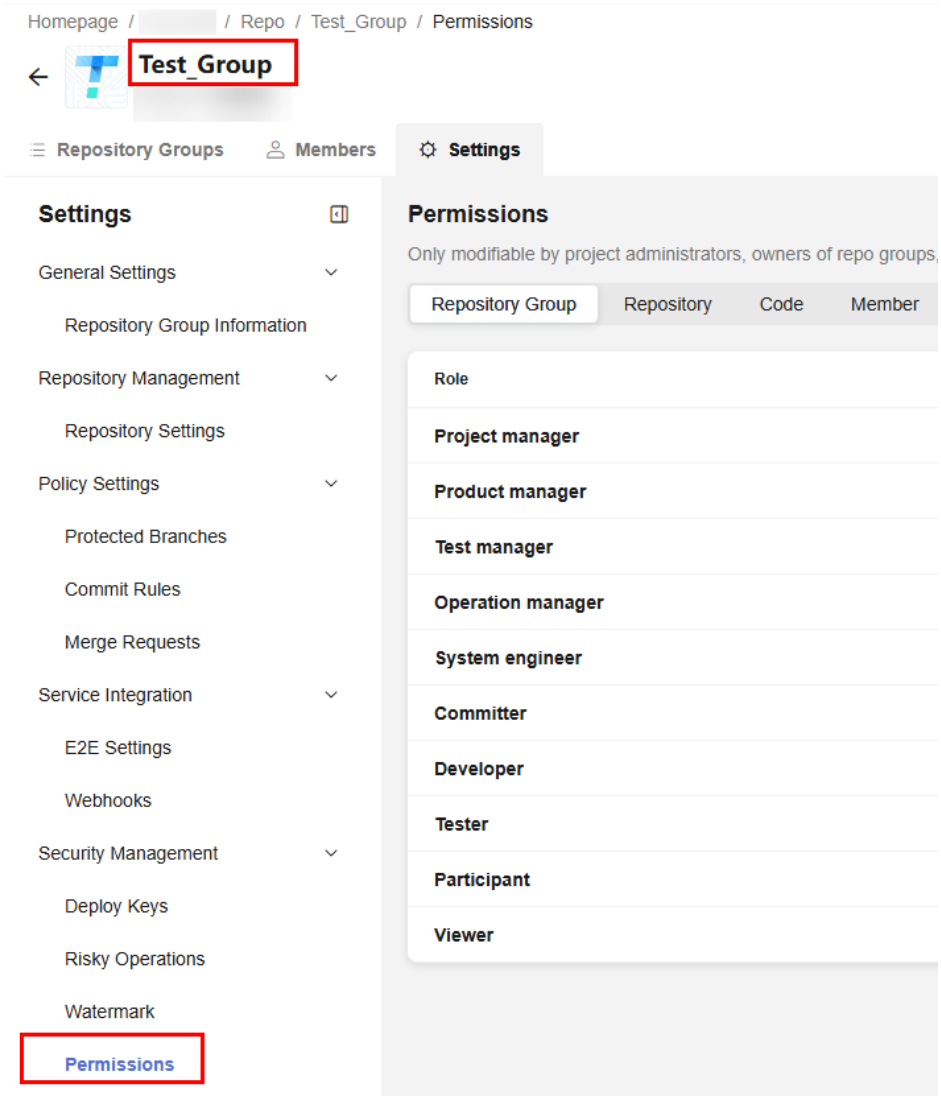
6.3 Configuring a Repository Group's Permissions

Constraints

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

Steps

Log in to the target repository group's page. In the navigation pane on the left, choose **Settings** > **Security Management** > **Permissions**.



Step 1 You can enable **Use Project Permissions** to use project-level permission configuration, which cannot be changed.

You can also click to sync project permissions. After the synchronization, you can modify the permissions.

You can also configure the permissions according to the following table. **A** indicates that the role has the permission by default and cannot be removed. **B** indicates that the role has the permission by default and can be removed. **C** indicates that the role can be assigned with the permission. **D** indicates that the role cannot be assigned with the permission.

Table 6-2 Permissions of the repository group roles

Role/Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Customized Role
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
	Setting	B	D	D	D	D	D	D	D	D	D	C
Repository	Create	B	C	C	C	B	B	B	C	C	D	C
	Fork	B	D	D	D	D	D	D	D	D	D	C
	Delete	B	C	C	C	B	B	B	C	C	D	C
	Setting	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
Member	Add	B	D	D	D	D	D	D	D	D	D	D
	Edit	B	D	D	D	D	D	D	D	D	D	D
	Delete	B	D	D	D	D	D	D	D	D	D	D
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

Role / Permission	Permission	Project Manager	Product Manager	Test Manager	Operation Manager	System Engineer	Committer	Developer	Tester	Participant	Viewer	Customized 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
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

----End

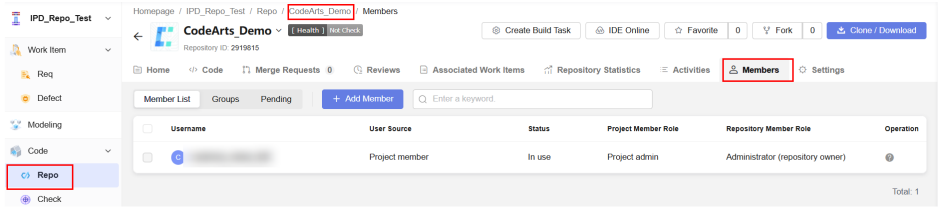
6.4 Configuring Repo-Level Permissions

Constraints

Table 6-3 Constraints on configuring repository permissions

Item	Description
Permission constraints	• The permission matrix of a repository can be modified only by project administrators, parent repository groups, and the owners of the repositories. • If you are the creator of an MR, you have the highest permission on it. If there is no other policy settings, you can edit, comment, review, approve, merge, close, re-open, or delete the MR, regardless of the permission matrix.
Function constraint	For public repositories, you have the code download permission and the comment permission of MR by default, which cannot be deleted or modified. Other permissions are the same as the default permissions of private 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 displayed code repository details page, click **Members** on the navigation bar to add members to the code repository.



Complete the member configuration of the code repository. On the navigation bar, 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 synchronize the custom role of the project. By default, the custom role does not have the repository operation permission. After the

synchronization, you can add the permission in **Table 6-4** as required. **A** indicates that the role has the permission by default and cannot be removed. **B** indicates that the role has the permission by default and can be removed. **C** indicates that the role can be assigned with the permission. **D** indicates that the role cannot be assigned with the permission.

Table 6-4 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

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
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

6.5 Syncing Project Members to CodeArts Repo

Overview

CodeArts Repo allows you to sync project members to your repository groups and repositories for better project managements. Choose between automatic or manual sync to suit your needs. Before configuration, you can refer to **constraints**. For details, see **Auto-Syncing Project Members to a Repository Group or Repository** or **Manually Adding Project Members to a Repository Group or Repository**.

Constraints

Table 6-5 Constraints on synchronizing project members

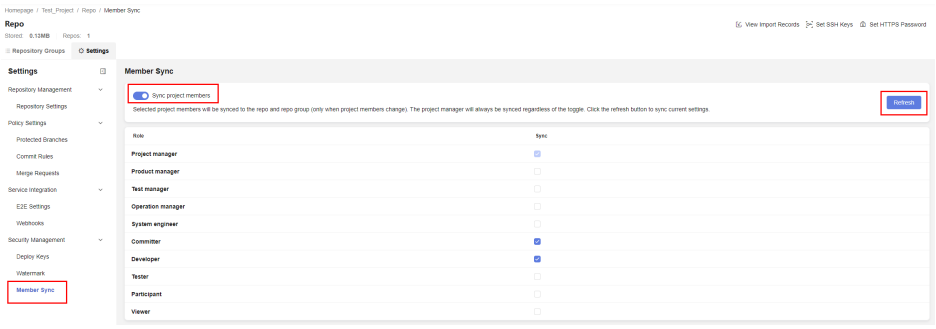
Item	Description
Permission constraint	The repo owner, repo administrator, and custom roles with member management permissions can change repo members. Other users can only view the repo member list. For details about permission configuration, see **Configuring Repo-Level Permissions**.
Function Constraint	• Before adding a repository 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**. • If the member list displayed after clicking the **Add Member** button is empty, no member is available to add to the repository. Add project members first. • In the member list, all members can be set to any project role and can be removed from the repository.

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 repository groups and repositories in the project.

Go to the **Test_Project** homepage. Click **Settings > Repo**. Choose **Security Management > Member Sync**, click **Sync project members**, and select the target

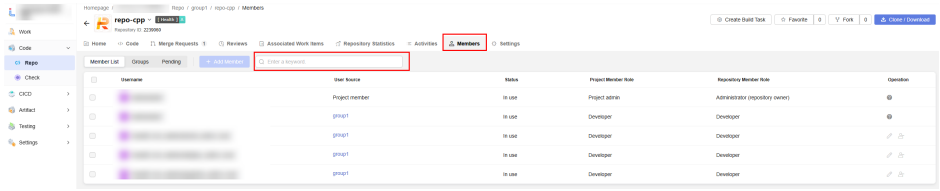
role. The project manager will always be synced to the repository group and repositories regardless of the toggle. Click the refresh button to sync all roles immediately. Automatic sync of updated project members is triggered only when **Sync project members** is enabled.



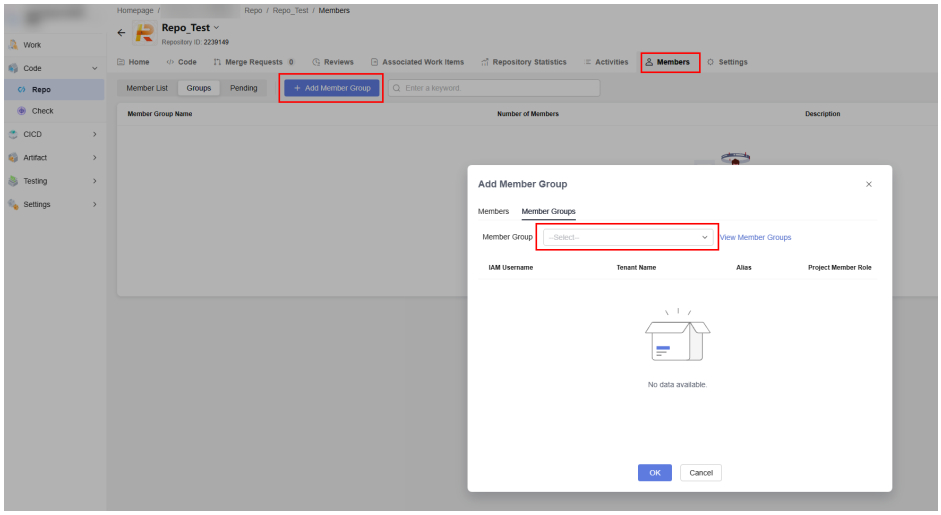
Manually Adding Project Members to a Repository Group or Repository

Use an account with repository setting permissions to go to the repository group or repository homepage, click the **Members** tab, and click **Add Member**. The page for adding members is displayed. You can add members in either of the following ways:

- On the **Members** tab page, enter a keyword and press **Enter** to search for a member.



- On the **Member Groups** tab page, select a member group from the drop-down list.



7 Creating a Repository

7.1 Creating Repos in Different Scenarios

You can create repositories based on your application scenarios. CodeArts Repo supports the following repository creation modes:

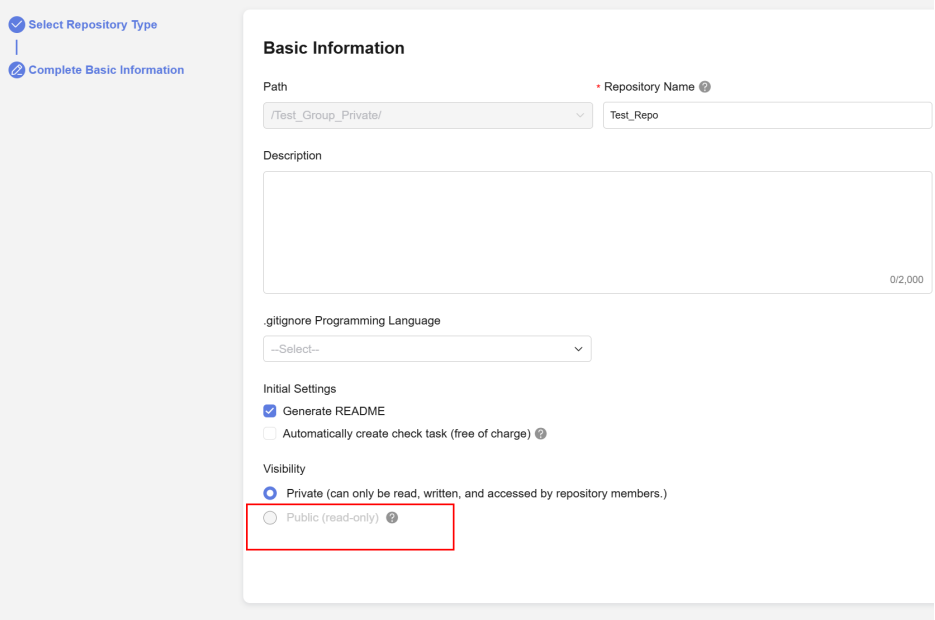
- **Create a custom repository**. You can create a custom repository to suit your needs. If your local code repositories have not been integrated into any version control system, such as Git, you can migrate them to CodeArts Repo by creating a custom repository.
- **Create a repository using a template**. You can quickly initialize a new repository and start version control by using the templates provided by CodeArts Repo.
- **Fork a repository**. You can initiate new projects based on historical ones while preserving the repository structure of the latter.

Before creating a repository, check whether the prerequisites for **creating a repository** are met.

7.2 Prerequisites for Creating a Repository

Before creating a repository in CodeArts Repo, ensure that:

1. CodeArts Repo must be enabled in the project. You need to create a project or select an existing one.
2. You have the permission to create a repository. If not, refer to **Configuring Repo-Level Permissions**.
3. Public repositories cannot be added to a private repository group. As shown in the following figure, when the repository group **Test_Group_Private** is private, **Public** in **Visibility** is unavailable.

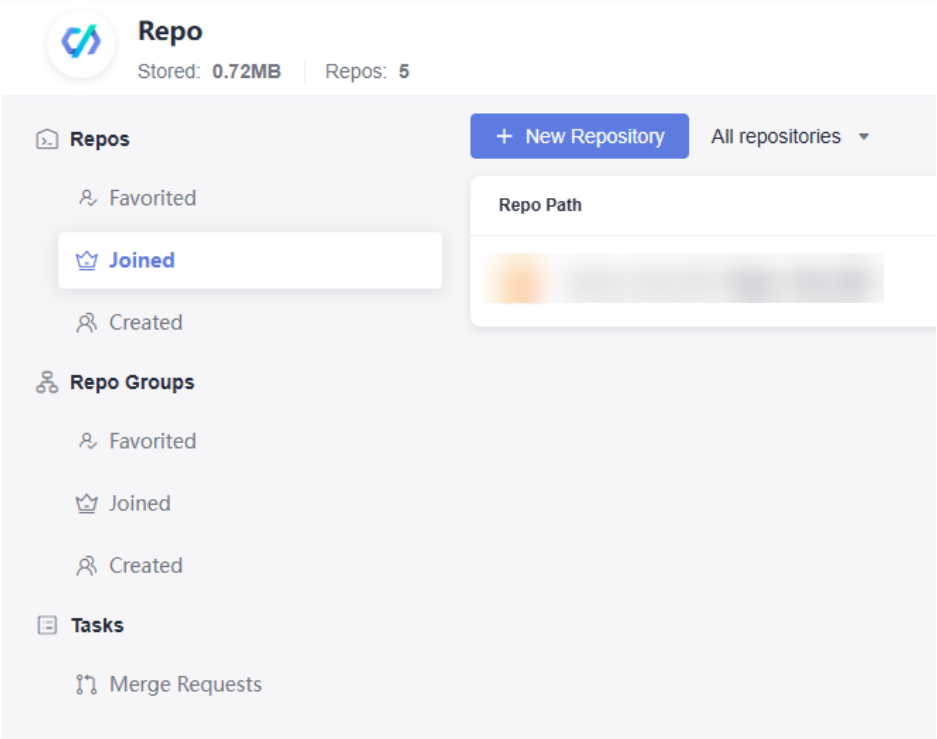


7.3 Creating a Custom Repository

Constraints

Before creating a custom repository, check the **prerequisites**.

Step 1 Enter the CodeArts Repo homepage as shown in the figure. 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**, click **Next**, and set parameters according to **table**.

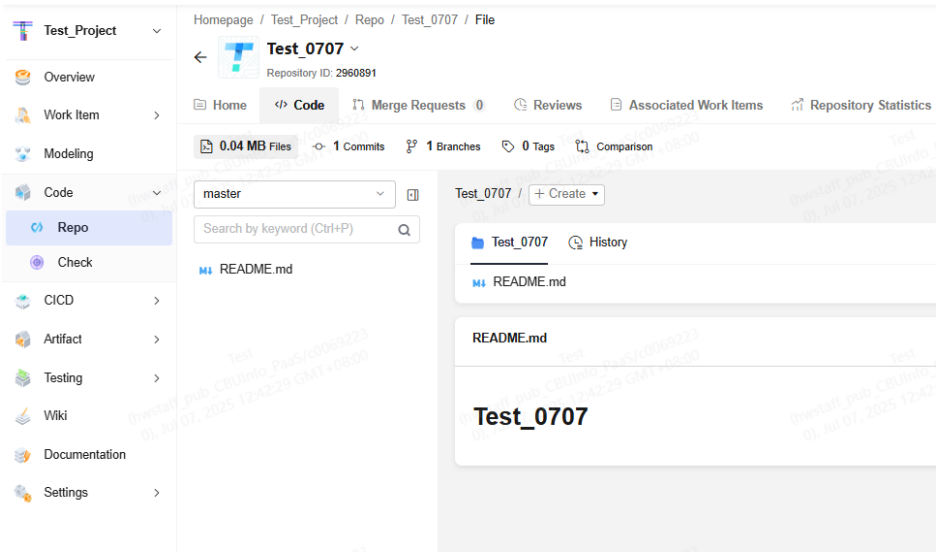
Table 7-1 Parameters for creating a repo

Parameter	Description
Path	Optional. Target repo group path of the new repo. You can select a repo group path from the drop-down list.
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 2,000 characters.
.gitignore Programming Language	Optional. It is recommended that you fill in this parameter and select the programming language for your code repository from the drop-down 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: • **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) You must have the CodeArts Check permissions. Otherwise, the configuration cannot take effect.** 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.

Parameter	Description
Visibility	Optional. You can select either of the following options as needed: • **Private**: Only repository members can access and commit code. • **Public**: The value can be **For project members**, **For tenant members**, or **For all guests**. 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. For project members: Project members can view and search for repositories in the repository list of a project and repository group. For tenant members: Tenant members can view and search for repositories in the repository list of a project and repository group. For all visitors: All visitors can view and search for repositories in the repository list of a project and repository group.
Open-Source License	This parameter is mandatory when **Visibility** is set to **Public**. Select an existing license from the drop-down list.

Step 3 Click **OK**. The **Code** page of the new repository is displayed. As shown in the following figure, the repository **Test_0707** is successfully created.

Figure 7-1 Repository **Test_0707**



----End

7.4 Creating a Repository Using a Template

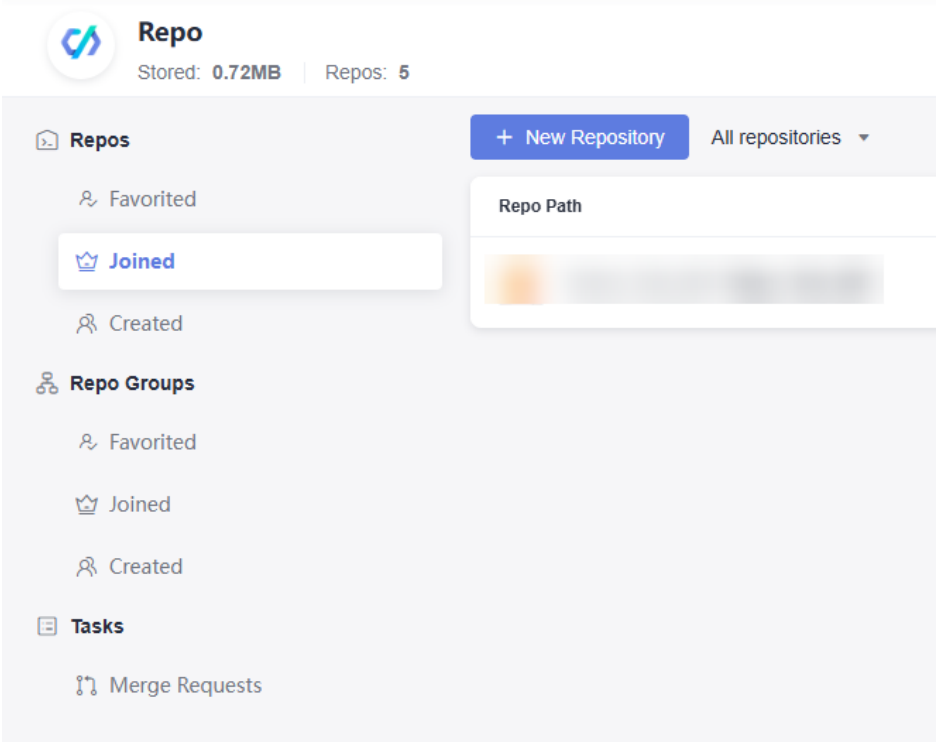
CodeArts Repo provides various repository templates for you to create repositories from. Before creating a repository, refer to **Constraints** and **Procedures**. You can view the official templates supported by CodeArts Repo in **Procedures**.

Constraints

Before creating a repository using a template, check whether the prerequisites are met by referring to **Prerequisites for Creating a Repository**.

Procedures

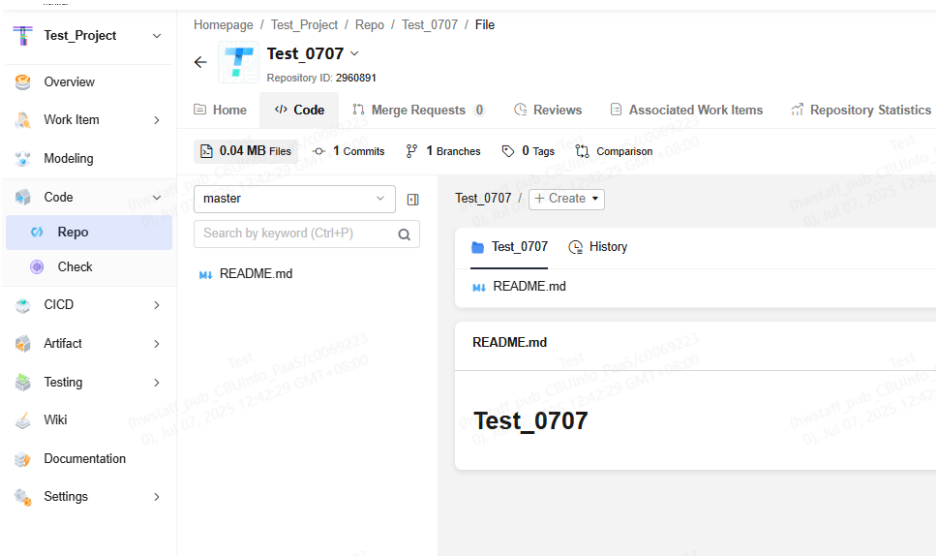
Step 1 Enter the CodeArts Repo homepage as shown in the figure. Click **New Repository**, and select an existing project from the **Project** drop-down list box or create a project.



Step 2 Select **Template**. You can select **CodeArts Templates** or **Custom Templates**. Custom templates can be set on the repository settings page. You can set official templates as custom ones.. After selecting a template, set parameters based on the **table**.

Step 3 Click **OK**. The **Code** page of the new repository is displayed. As shown in the following figure, the repository **Test_0707** is successfully created.

Figure 7-2 Repository **Test_0707**



----End

Official Repository Templates

Table 7-2 Repository templates

Key Provider	Description
Java Maven Demo	It uses the Maven template for building. Programming language: Java. Build tools: Maven 3.5.3 and JDK 1.8.
Java Ant Demo	It uses the Ant template for building. Programming language: Java. Build tools: Ant 1.10.3 and JDK 1.8.
Android Gradle Demo	It uses the Gradle (for Android) template for building. Programming language: Java. Build tools: Gradle and JDK 1.8.
Cpp Demo	It uses the CMake template for building. Programming language: C/C++. Build tools: CMake 3.10.1 and GCC 5.5.0.
Windows Cpp Demo	It uses the MSBuild template for building. Programming language: C++. Build tool: MSBuild 15.0.
Groovy Grails Demo	It uses the Grails template for building. Programming language: Groovy. Build tools: Grails 2.5.0 and JDK 1.8.
Python2 Demo	It uses the SetupTool template for building. Programming language: Python. Build tool: Python 2.7.
Python3 Demo	It uses the SetupTool template for building. Programming language: Python. Build tool: Python 3.6.
C Codesoucery Demo	It uses the Gnu-arm template for building. Programming language: C. Build tool: gnuarm201405 image.
Cpp LiteOS Demo	It uses the Gnu-arm template for building. Programming language: C++. Build tool: gnuarm-7-2018-q2-update.
Nodejs Webpack Demo	It uses the NPM template for building. Programming language: Node.js. Build tool: Node.js-8.11.2.
C# Demo	It uses the MSBuild template for building. Programming language: C#. Build tools: MSBuild15-dotnetframework 4.7.2.

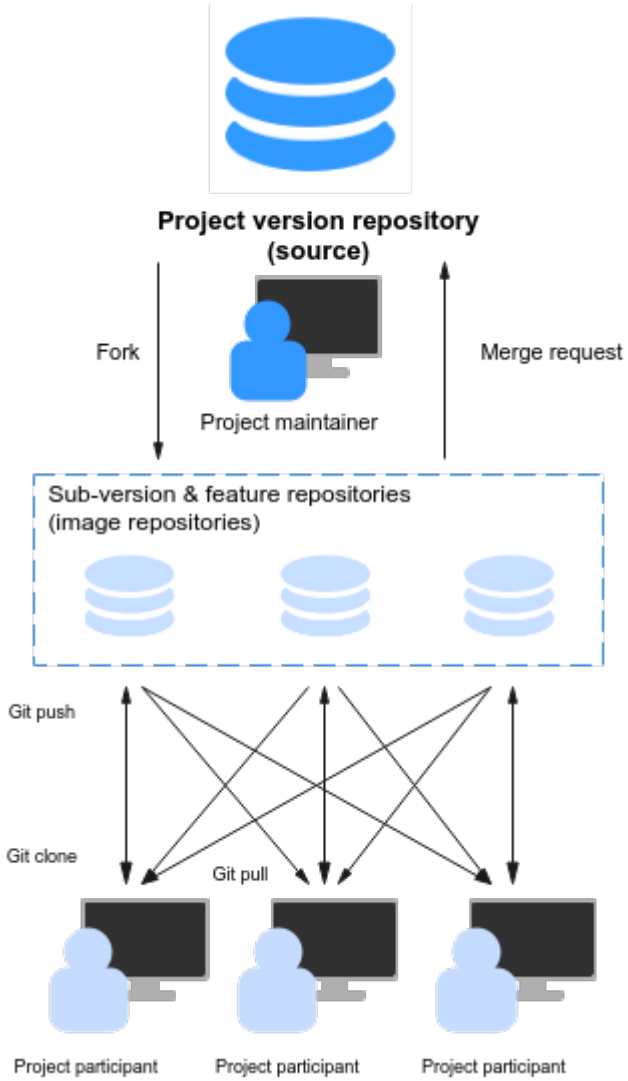
Key Provider	Description
Java Web Demo	This sample repository is used to demonstrate how to use CodeArts to build and deploy via pipelines on cloud.
Go web Demo	It uses the Hello World applet based on the Go language. Programming language: Go. Build tool: Go-1.10.3.
Java War demo	It uses the Hello World applet complied based on the Maven language and generates a .war package after using Maven for building. Programming language: Java. Build tools: Maven 3.5.3 and JDK 1.8.
Phoenix Mall	An example of an e-commerce website that helps developers quickly learn about CodeArts features and solve software development, testing, and deployment problems during microservice use.

7.5 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.

Constraints

Before forking a repository, you must have the necessary permissions; see **Configuring Repo-Level Permissions**.

Steps

Step 1 Access the repository list page. Click a repository name to go to the target repository.

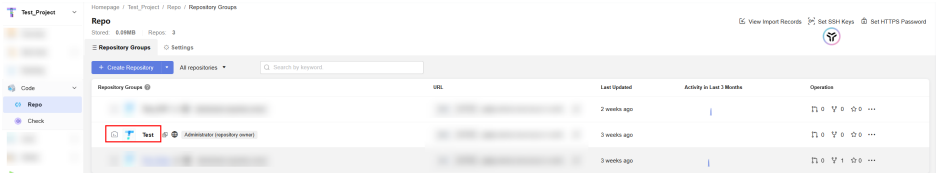
Step 2 Click **Fork** in the upper right corner. In the displayed **Fork Repository** window, set parameters by referring to the following table.

Table 7-3 Fork repository parameters

Parameter	Description
Project	Target project of the new repo. You can select a project name you have already joined from the drop-down list.
Path	Optional. Target repo group path of the new repo. You can select a repo group path from the drop-down list.
Repository	Start with a letter, digit, or underscore (_). You can use letters, digits, hyphens (-), underscores (_), and periods (.). Do not end with **.git**, **.atom**, or periods (.). The repository path (including the repository group name and repository name) cannot exceed 256 characters.
Visibility	Indicates the visible scope of the source repo. The options are as follows: • **Private**: Only members of this repository can access it and commit code. • Public The value can be **For project members**, **For tenant members**, or **For all guests**.

Parameter	Description
Synchronize commit rules of the source repository	Optional. Indicates whether to synchronize the commit rules of the source repo. This option is selected by default, which means the source repo commit rules are synchronized. You can choose **Settings > Policy Settings > Commit Rules** to view the rule settings of the source repo and determine whether to select it.

Step 3 Click **OK** to fork the repository. As shown in the following figure, after the fork is successful, the repository (group) page of the target project is automatically displayed, and **Test** is the forked repository.



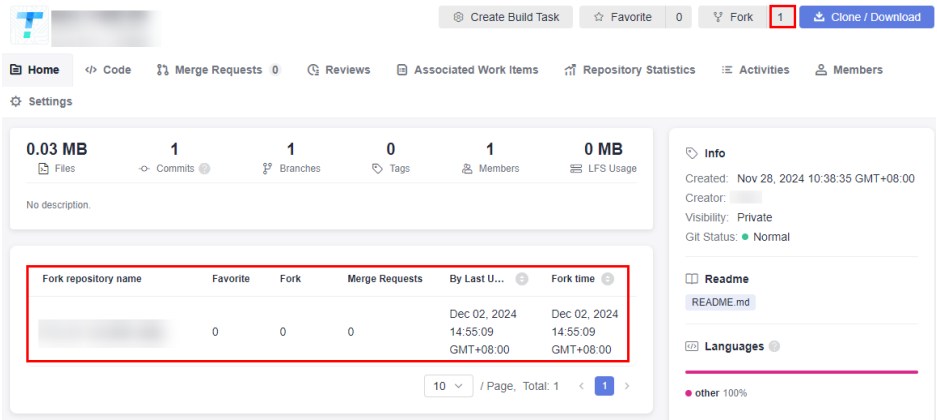
----End

Checking the Fork Status of the Current Repo

Step 1 Access the repository list page.

Step 2 Click the source repository name.

Step 3 Click the number after **Fork** in the upper right corner of the page to view the list of forked repositories, as shown in the following figure. The current source repo is forked only once. Click the name in the **Fork repository name** column to go to the **Code** page of the forked repo.



----End

Merging Changes of a Fork to the Source Repository

Step 1 Access the repository list page.

Step 2 Click the name of the forked repository.

Step 3 Click the **Merge Requests** tab.

Step 4 Click **Create MR**. On the **Create MR** page that is displayed, select a source branch and target branch to be merged. Set **Title** and **Description**, and click **Create MR**.

NOTE

Cross-repository merge requests belong to the source repository and can be viewed only on the **Merge Requests** page of the source repository. These merge requests cannot be viewed in the forked repository (that initiates these merge requests). Therefore, the reviewer, scorer, approver, and merger must be members of the source repository.

----End

8 Migrating Code and Syncing a Repository

8.1 Repository Migration Overview

CodeArts Repo allows you to import third-party repositories through URLs to better manage code resources.

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

- Migrate repositories to CodeArts Repo using URLs. For details, see **Migrating a Git Repository Using a URL**.
- Migrate GitHub repositories to CodeArts Repo using service endpoints. For details, see **Migrating a GitHub Repository**.
- Migrate GitLab repositories to CodeArts Repo using personal access tokens. For details, see **Migrating a GitLab Repository**.
- Migrate GitLab repositories to CodeArts Repo using personal access tokens. For details, see **Migrating a Self-Built GitLab Repository**.
- Migrate Gitee repositories to CodeArts Repo using personal access tokens. For details, see **Migrating a Gitee Repository**.
- Migrate Coding repositories to CodeArts Repo using personal access tokens. For details, see **Migrating a Coding Repository**.
- Migrate Codeup repositories to CodeArts Repo using personal access tokens, Apsara DevOps service endpoints, and organization IDs. For details, see **Migrating a Codeup Repository**.
- Migrate Bitbucket repositories to CodeArts Repo using workspace IDs, usernames, and passwords. For details, see **Migrating a Bitbucket Repository**.
- Migrate Gerrit repositories to CodeArts Repo using the server address, username, and password. For details, see **Migrating a Gerrit Repository**.
- Import repositories that are not managed by Git to CodeArts Repo. For details, see **Importing a Local Git Repository**. Clone third-party Git repositories to local repositories and import them to CodeArts Repo. For details, see **Importing a Local Third-Party Git Repository to CodeArts Repo**.

- Migrate SVN repositories to CodeArts Repo online. For details, see **Import an SVN repo to CodeArts Repo**. Migrate SVN repositories to CodeArts Repo using Git Bash. For details, see **Migrating an SVN Repository**. For details about how to solve the problems encountered during SVN repository migration, see **Related Document**.

8.2 Obtaining an Access Token

8.2.1 Obtaining an Access Token from GitHub

Constraints

You have a GitHub account.

Configuring an Access Token

Step 1 **Log in to GitHub**, click the profile picture 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.

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

----End

8.2.2 Obtaining an Access Token from GitLab

Constraints

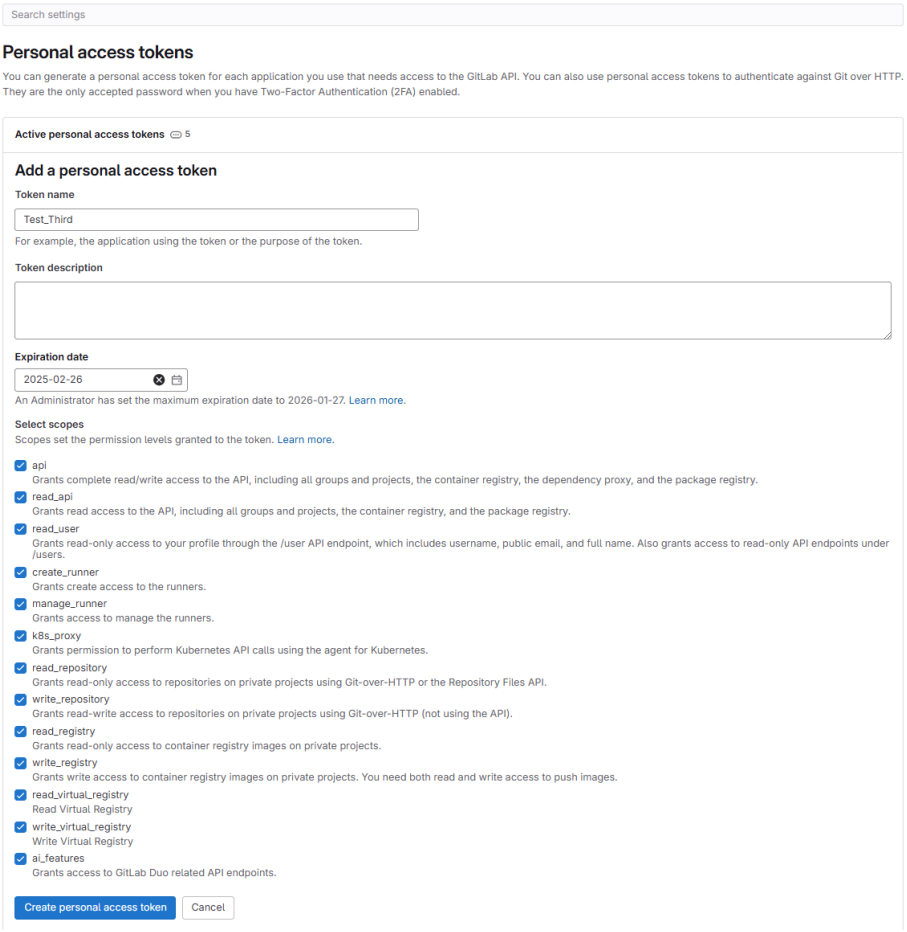
You have a GitLab account.

Configuring an Access Token

Step 1 **Log in to GitLab** with an account, click the profile picture in the upper left corner, and click **Preferences**. On the page that is displayed, choose **User Settings** > **Access Tokens**.

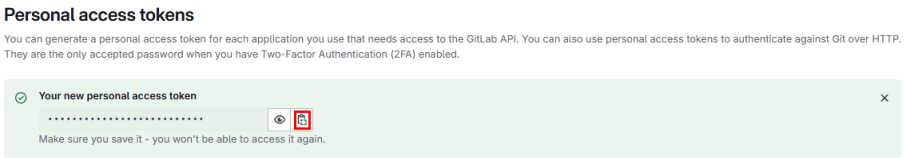
Step 2 As shown in the following figure, customize **Token name**, select **Expiration date**, select all permissions, and click **Create personal access t+oken** to complete the configuration of the GitLab personal access token.

Figure 8-1 Page for creating a personal access token in a GitLab repository



Step 3 After the preceding steps are performed, copy the personal access token s shown in the following figure.

Figure 8-2 Page for copying personal access token



----End

8.2.3 Obtaining an Access Token from Gitee

Constraints

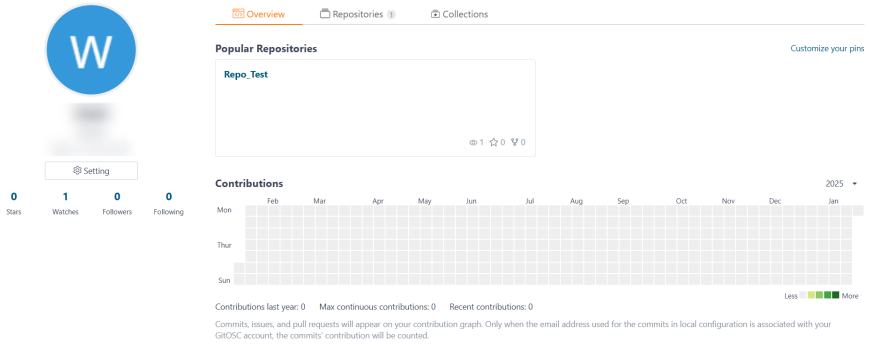
You need to have a Gitee account.

Configuring an Access Token

Step 1 Log in to Gitee, click the nickname in the upper right corner, and click **Profile**. The personal overview page is displayed.

On this page, click **Setting**. The **Account Information** page is displayed.

Figure 8-3 Personal overview page



Step 2 In the navigation pane on the left, choose **Security Settings** > **Personal access tokens**. In the upper right corner, click **Generate new token**. The page for creating a private token is displayed.

Step 3 Enter **Personal Token Description**, set **Token Expiration Time**, and select the permissions of the personal token.

Step 4 Click **Submit**, enter your login password for verification in the displayed dialog box, and the private token is generated.

Step 5 Click **Copy**, select **I have learned that the personal token is no longer displayed in clear text on the platform, and the token has been copied and saved**, and click **confirm and close**.

----End

8.2.4 Obtaining an Access Token from Coding

Requirement

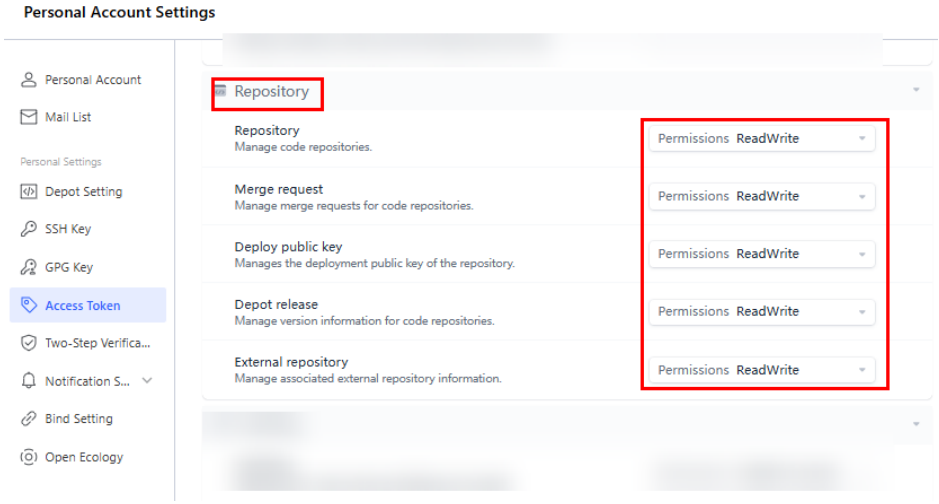
You already have a Coding account.

Configuring an Access Token

Step 1 Log in to Coding, click the profile picture in the lower left corner, choose **Personal Account Settings** , and click **Access Tokens** . On the current page, click **Create Token** in the upper right corner .

Step 2 Enter **Token Description** and set **Expiration time** , as shown in the following figure. Configure the repository permission, which can be set to **ReadWrite** .

Figure 8-4 Configuring repository permissions



Step 3 Configure **IP Whitelist** as needed. If you do not need to configure it, click **Confirm** . Then, perform identity authentication as prompted.

Step 4 Click to copy and save the current access token.

----End

8.2.5 Obtaining an Access Token from Codeup

Requirement

You already have a Codeup account and joined an organization.

Configuring an Access Token

Step 1 Log in to Codeup, click the profile picture in the upper right corner, and choose **My Settings**. In the navigation pane on the left, choose **Personal Access Tokens**, and click **Create Token** in the upper right corner.

Step 2 Set **Token Name**, **Description**, and **Expiration Time**. Set the repository permission for code management and set the permission to **Read-only**.

Step 3 Click to copy and save the current access token.

----End

Configuring the Apsara DevOps Access Point

- Standard domain name (standard): openapi-rdc.aliyuncs.com
- Primary domain name: The main domain name of your organization instance. For details, refer to the Apsara DevOps help documents.

Obtaining the Organization ID

Log in to Codeup, click the profile picture in the upper right corner, and choose **My Settings**. In the navigation pane on the left, click **Organizations**, and copy the value of **Organization ID**.

8.2.6 Obtaining a Password from Bitbucket

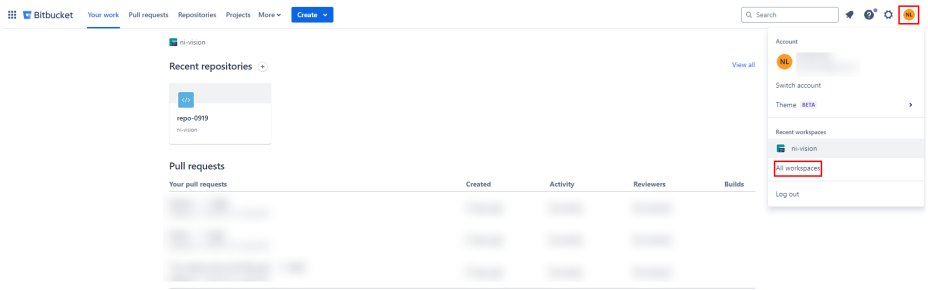
Requirement

You already have a Bitbucket account.

Obtaining a Workspace ID

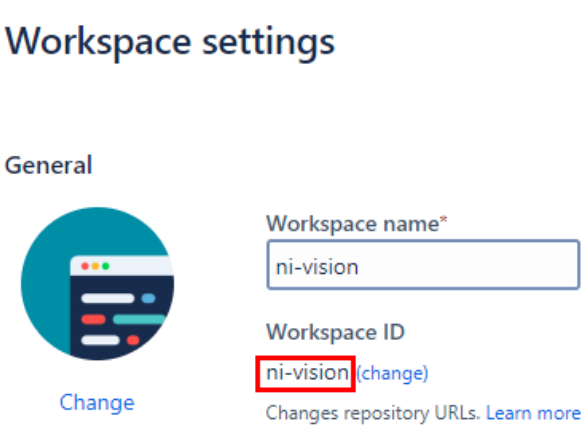
Step 1 Log in to Bitbucket, click the profile picture in the upper right corner, choose **All workspaces**, and select a workspace.

Figure 8-5 Logging in to the Bitbucket page



Step 2 After entering the space, click  in the upper right corner and choose **Workspace settings**.

Step 3 Copy and save the workspace ID. In the following figure, the workspace ID is **ni-vision**.



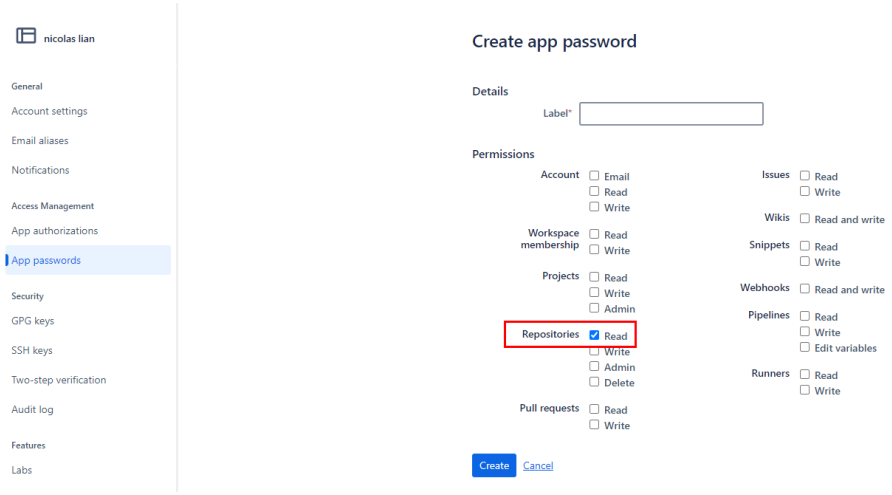
----End

Obtaining a Username and Password

Step 1 Log in to the **Bitbucket console** and click **Account settings** in the navigation tree on the left. **Username** indicates the username. Copy it and save the settings.

Step 2 **Log in to the Bitbucket password page**. If no password is displayed on the current page, click **Create app password** to create a password, as shown in the following figure.

Figure 8-6 Creating a new password



----End

8.3 Migrating a Third-Party Git Repository to CodeArts Repo

8.3.1 Migrating a Git Repository Using a URL

Constraints

- You already have a project or need to create a new one.
- You have the permission to create a repository. If not, refer to **Configuring Repo-Level Permissions**.
- The repository domain must be connected to the service node.
- 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 xiaolvyun.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 by referring to **Configuring Repo-Level Permissions**.

Steps

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 8-1**.

Table 8-1 Parameters for obtaining authorization

Parameter	Description
Source Repository URL	Mandatory. Specify the repo path to be imported. The source repo path must start with **http://** or **https://** and end with **.git**.
Verification to Access Source Repo	Mandatory. There are two cases: • 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**. After the parameters are set, the **Code** page for creating the repo is displayed.

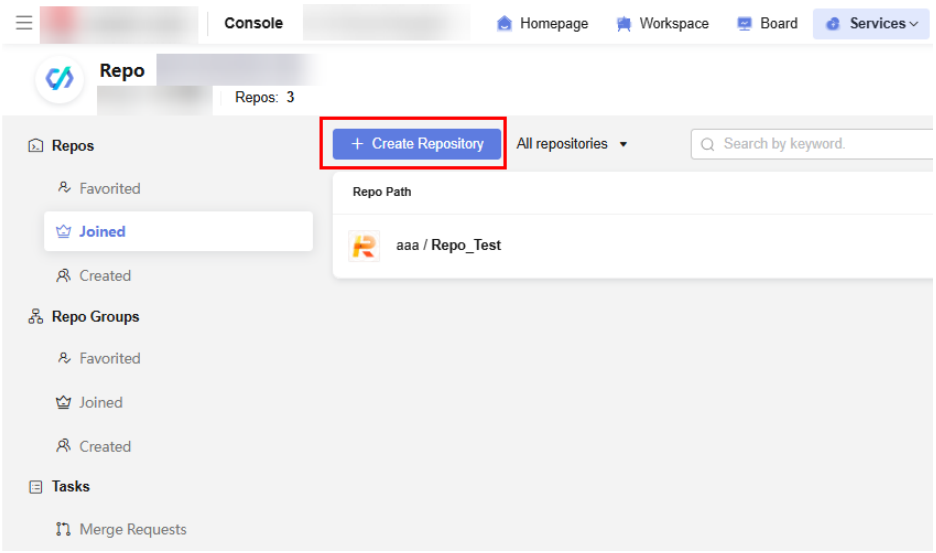
----End

Related Document

- 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/push function on the client to handle the problem. For details, see **Timed Out When Importing an External Repository**
- 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 repository and import the external repository again according to **Migrating a Git Repository Using a URL**.

8.3.2 Migrating 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 **Imported** 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**. The **Imported** page is displayed. Select repositories to be imported and click **Next**. Set basic information and repository synchronization by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing repositories**.

----End

Service Endpoint Authorization

Table 8-2 Service endpoint authorization

Parameter	Description
Service Endpoint Name	Mandatory. Enter a name with a maximum of 256 characters.

Parameter	Description
Authentication Mode	Mandatory. Select a value as required. • 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 on the endpoint creation page, 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**.

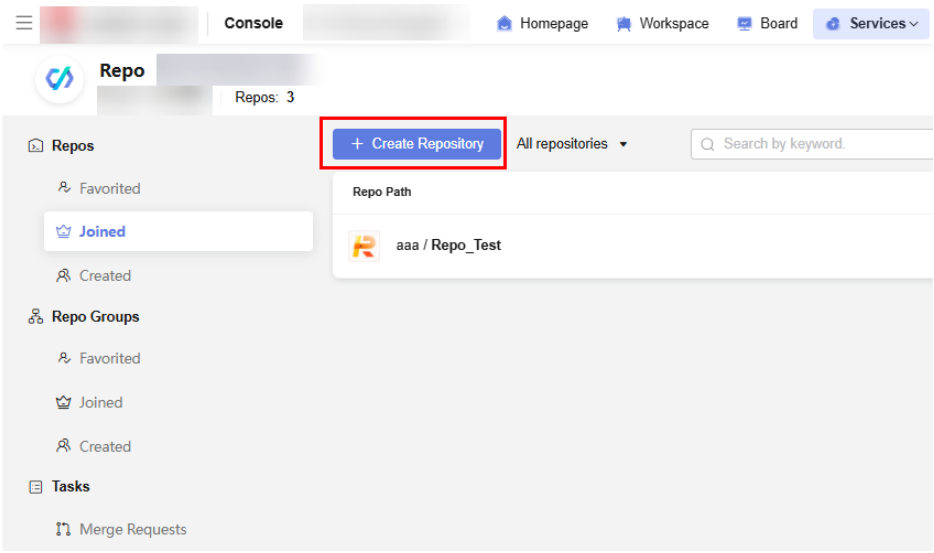
8.3.3 Migrating a GitLab Repository

Prerequisites

- Before importing a GitLab repository, you need to configure a personal access token for GitLab by referring to **Obtaining an Access Token from GitLab**.
- A maximum of 100 repositories can be imported at a time.

Steps

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.



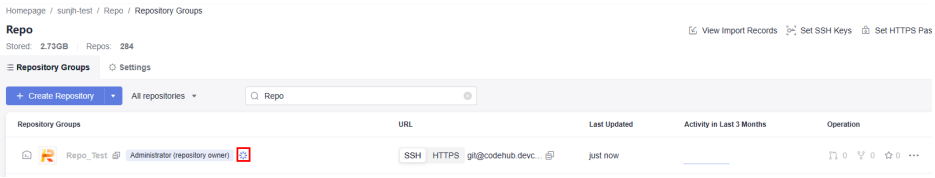
Step 2 Set the repository type to **Imported**, import from **GitLab**, and select **By personal access token** for **Authorization**.

If an error message indicating authorization failure is displayed, it means your access token is incorrect or invalid. In this case, reconfigure your access token in GitLab by referring to **Obtaining an Access Token from GitLab**.

Step 3 Click **Next**. The **Import** page is displayed. Select the repositories to be imported. Click **Next**. Set **Basic Information** and **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing repositories**.

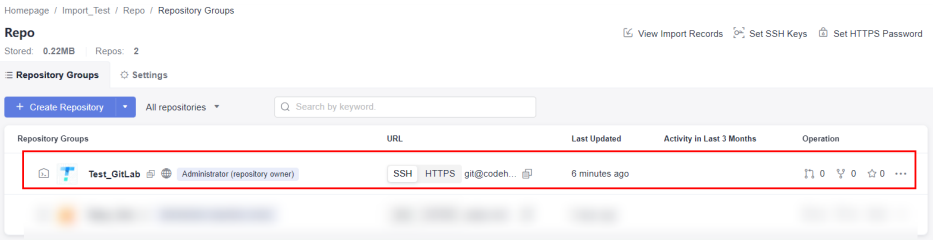
Step 4 Click **OK**. The repository list page of the project is displayed. If the following figure is displayed, the repository is being imported.

Figure 8-7 Repository being imported



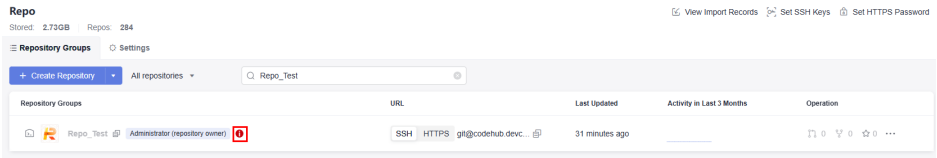
If the following figure is displayed, the GitLab repository is successfully imported.

Figure 8-8 GitLab repository successfully imported



If the following figure is displayed, the GitLab repository fails to be imported.

Figure 8-9 Repository import failed



----End

Video Tutorial

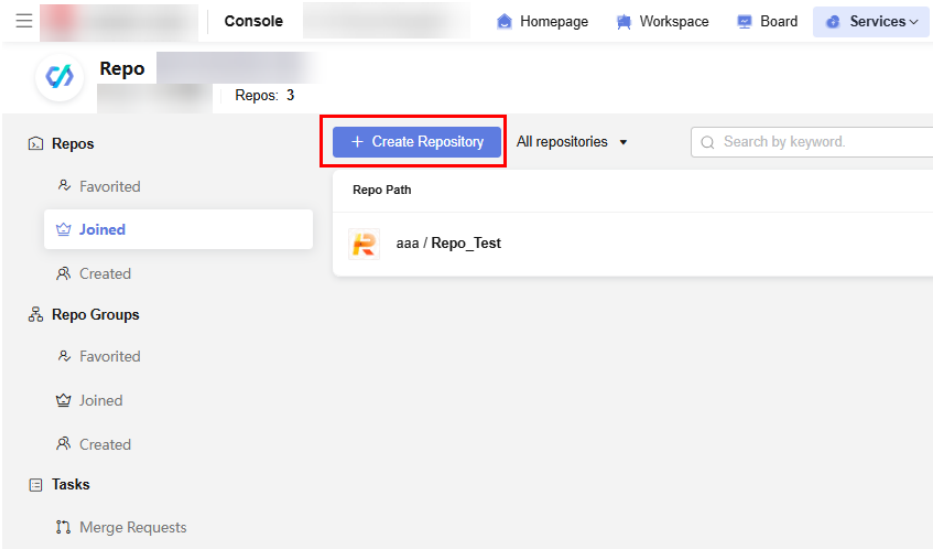
8.3.4 Migrating a Self-Built GitLab Repository

Prerequisites

- Currently, only self-hosted repositories of GitLab 11.0 or later can be imported.
- Before importing a repository from your own GitLab server , you need to configure a personal access token.

Steps

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 repository type to **Imported**, import from **Self-hosted GitLab**, and select **By personal access token** for **Authorization**. Click **Next**. The **Select Repo to Import** page is displayed.

Step 3 Select the repository to be imported and click **Next**. The **Basic Information** page is displayed. Enter the **Basic Information** and **Repo Sync Settings** of the

repositories by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

----End

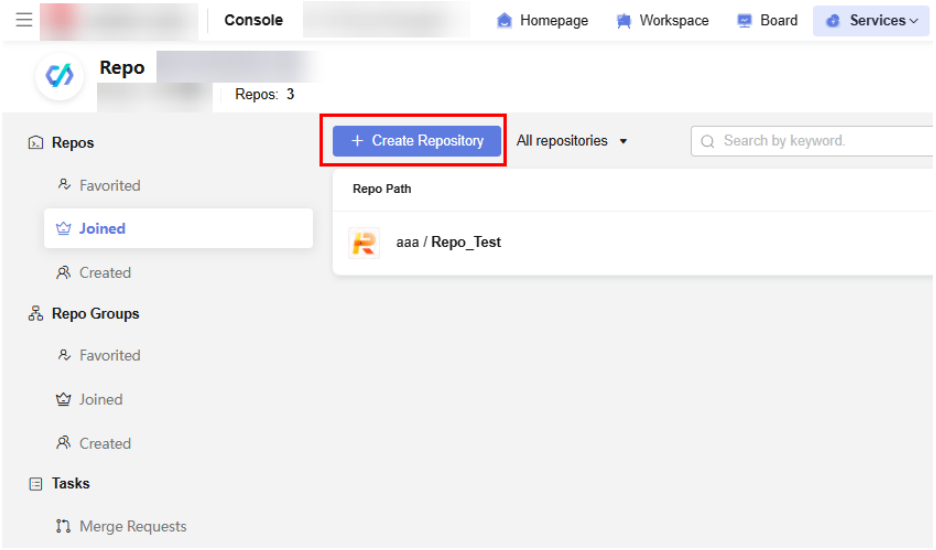
8.3.5 Migrating a GitHub Repository

Prerequisites

- Before importing a repository from your own Gitee server, you need to configure a personal access token by referring to **Obtaining an Access Token from Gitee**.

Steps

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.



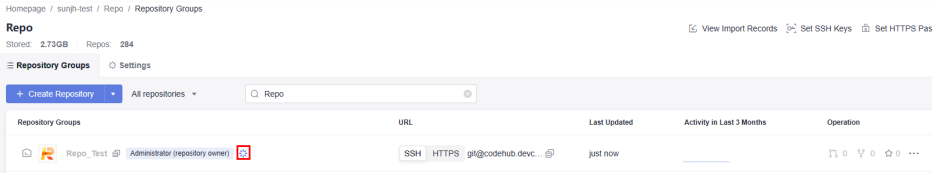
Step 2 Set **Repository Type** to **Imported**, **Import From** to **Gitee**, and **Authorization** to **By personal access token**. Enter the token and click **Next**.

If an error message indicating authorization failure is displayed, it means your access token is incorrect or invalid. In this case, reconfigure your access token in Gitee by referring to **Obtaining an Access Token from Gitee**.

Step 3 Select repositories to be imported and click **Next**. Enter **Basic Information** and set **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

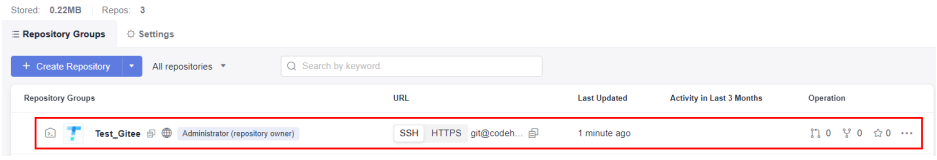
Step 4 Click **OK**. The repository list page of the project is displayed. If the following figure is displayed, the repository is being imported.

Figure 8-10 Repository being imported



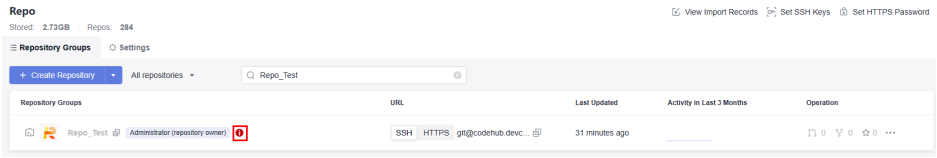
If the following figure is displayed, the Gitee repository is successfully imported.

Figure 8-11 Gitee repository successfully imported



If the following figure is displayed, the Gitee repository fails to be imported.

Figure 8-12 Repository import failure



----End

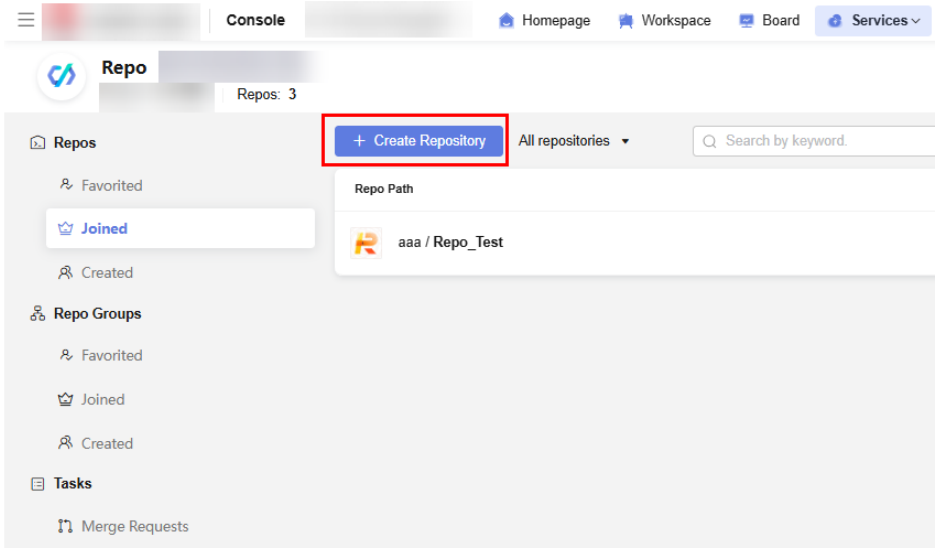
8.3.6 Migrating a Coding Repository

Prerequisite

Before importing a repository from your own Gitee server, you need to configure a personal access token by referring to **Obtaining an Access Token from Coding**.

Steps

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.



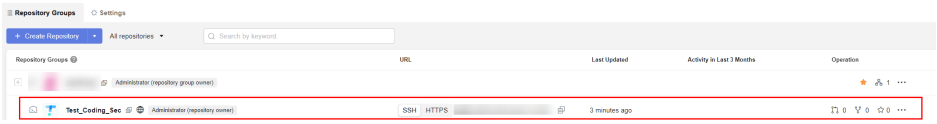
Step 2 Set the repository type to **Imported**, import from **Coding**, and select **By personal access token** for **Authorization**. If an error message indicating authorization failure is displayed, it means your access token is incorrect or invalid. In this case,

reconfigure your access token in Coding by referring to **Obtaining an Access Token from Coding**.

Step 3 Select repositories to be imported and click **Next**. Enter **Basic Information** and set **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

Step 4 Click **OK**. The repository list page of the project is displayed. If the following figure is displayed, the Coding repository is successfully imported.

Figure 8-13 Coding repository successfully imported



----End

8.3.7 Migrating a Codeup Repository

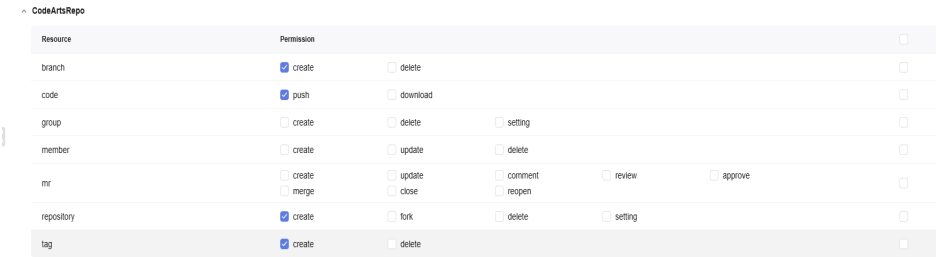
Prerequisites

- Configure the personal access token by referring to **Configuring an Access Token**.
- Obtain the Apsara DevOps URL by referring to **Configuring the Apsara DevOps Access Point**.
- Obtain the organization ID by referring to **Obtaining the Organization ID**.

Prerequisites

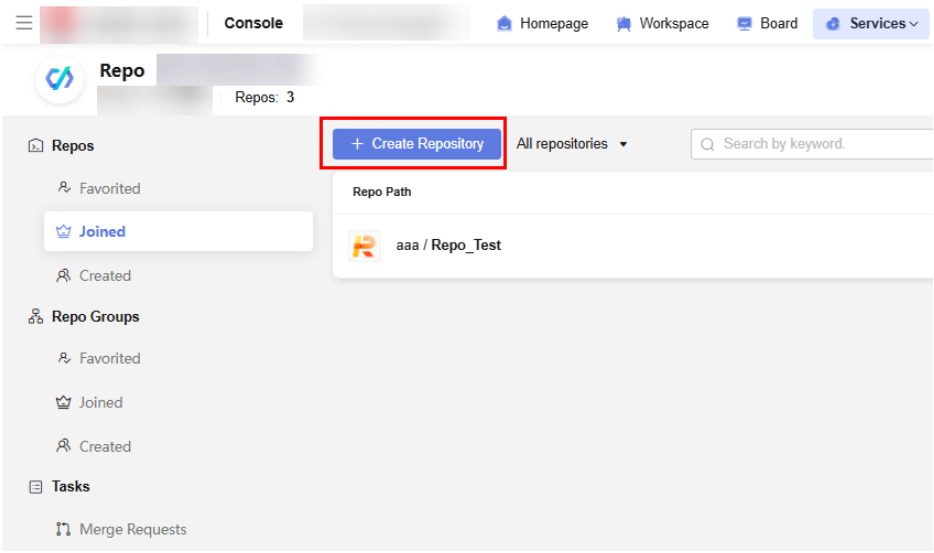
As shown in the following figure, you must have the permission to create repositories. For details, see **Configuring Project-Level Permissions**.

Figure 8-14 Permissions required



Steps

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 **Imported** and import from **Codeup**.

Step 3 Obtain and fill in the **Apsara DevOps URL** by referring to **Configuring the Apsara DevOps Access Point**. The format is **https://***Apsara DevOps URL*.

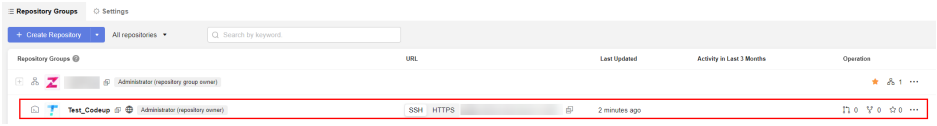
Step 4 Obtain and fill in the **Organization ID** by referring to **Obtaining the Organization ID**.

Step 5 Obtain the personal access token and fill in the **Authorization by Personal Access Token** box by referring to **Configuring an Access Token**. If an error message indicating authorization failure is displayed, it means your access token is incorrect or invalid. In this case, reconfigure your access token in Codeup.

Step 6 Select repositories to be imported, click **Next**, click **OK**, and the project repositories list is displayed. Set **Basic Information** and **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

Step 7 If the following figure is displayed, the Codeup repository is successfully imported.

Figure 8-15 The Codeup repository is successfully imported.



----End

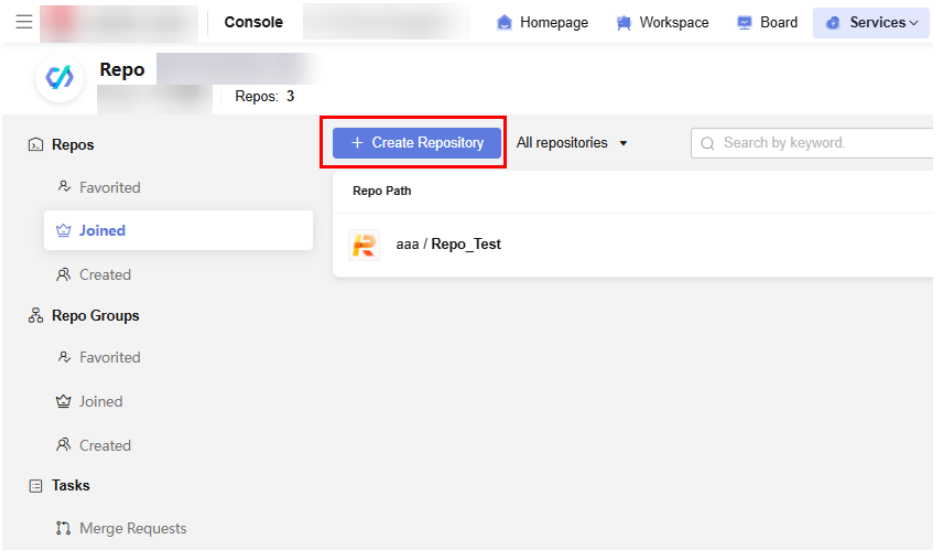
8.3.8 Migrating a Bitbucket Repository

Prerequisites

- Obtain the workspace ID by referring to **Obtaining a Workspace ID**.
- Obtain the username and password by referring to **Obtaining a Username and Password**.

Steps

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 **Imported** and import from **Bitbucket**.

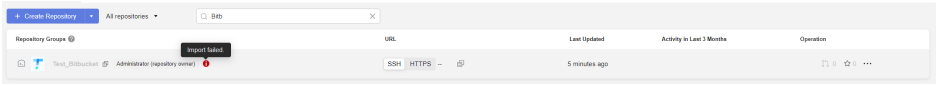
Step 3 Obtain and set **Workspace ID**, **Username**, and **Password** by referring to **Obtaining a Password from Bitbucket**.

Step 4 Select repositories to be imported and click **Next**. Enter **Basic Information** and set **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

Step 5 Click **OK**. The repository list page of the project is displayed.

Search for the repository name. If the following information is displayed as shown in the following figure, the Bitbucket repository fails to be imported.

Figure 8-16 Importing the Bitbucket repository failed



----End

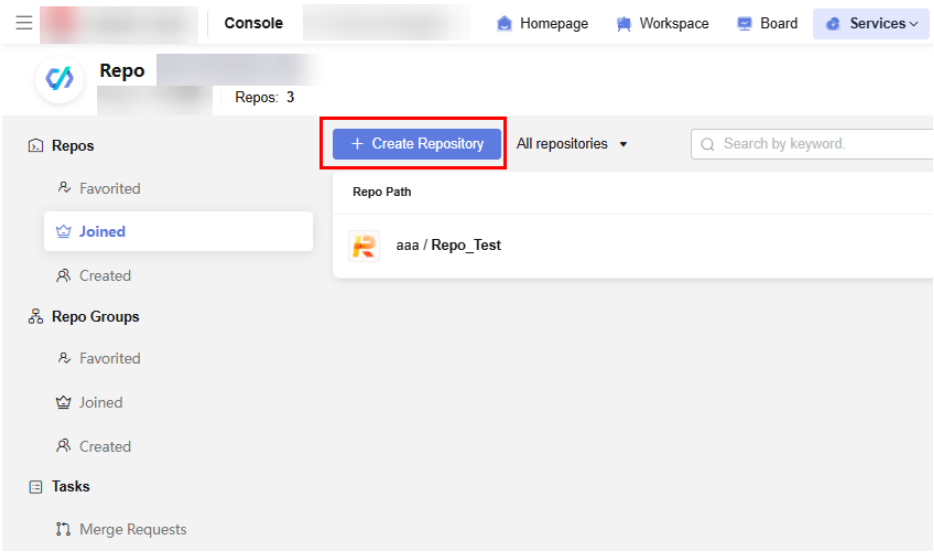
8.3.9 Migrating a Gerrit Repository

Prerequisite

Before importing a Gitee repository, you need to obtain the username and password.

Steps

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.



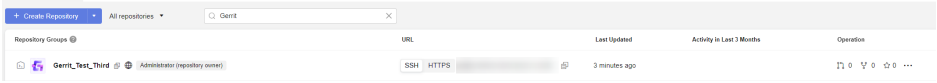
Step 2 Set the repo type to **Imported** and import from **Gerrit**.

Step 3 Set **Host Url**, which is the Gerrit server address, and set **Username** and **Password**.

Step 4 Select repositories to be imported and click **Next**. Enter **Basic Information** and set **Repo Sync Settings** by referring to **Configuring Basic Information for a New Repository** and **table 1 parameters for synchronizing a repository**.

Step 5 Click **OK**. The repository list page of the project is displayed.

Search for the repository name. If the following information is displayed as shown in the following figure, the Gerrit repository is successfully imported.



----End

8.4 Importing a Local Git Repository to CodeArts Repo

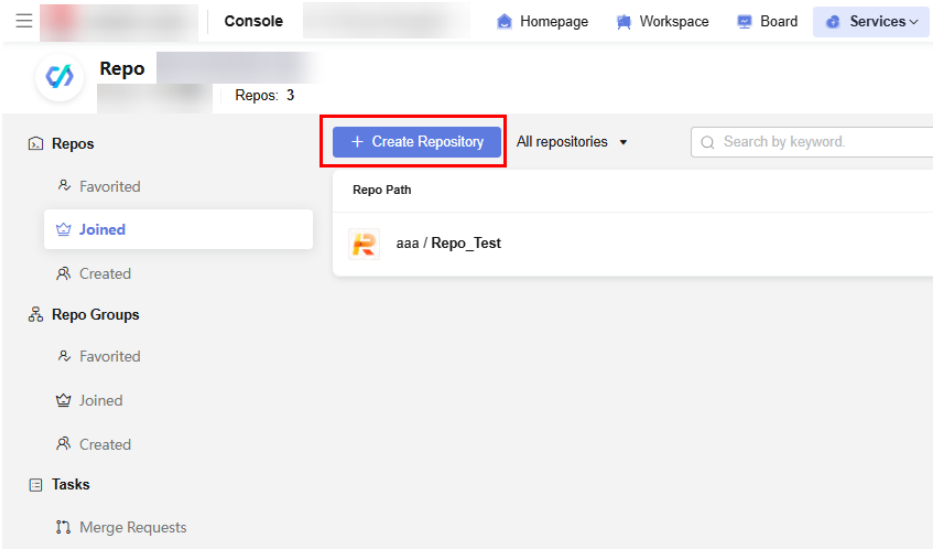
Constraints

- You need to create a new project or select an existing one.
- You have the permission to create a repository. If not, refer to **Configuring Repo-Level Permissions**.
- Before importing the repository, ensure that your CodeArts Repo has sufficient storage space. If it is almost full, clear the repository resources by referring to **Clearing the Repository Memory**.
- 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.

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 (master branch) 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.

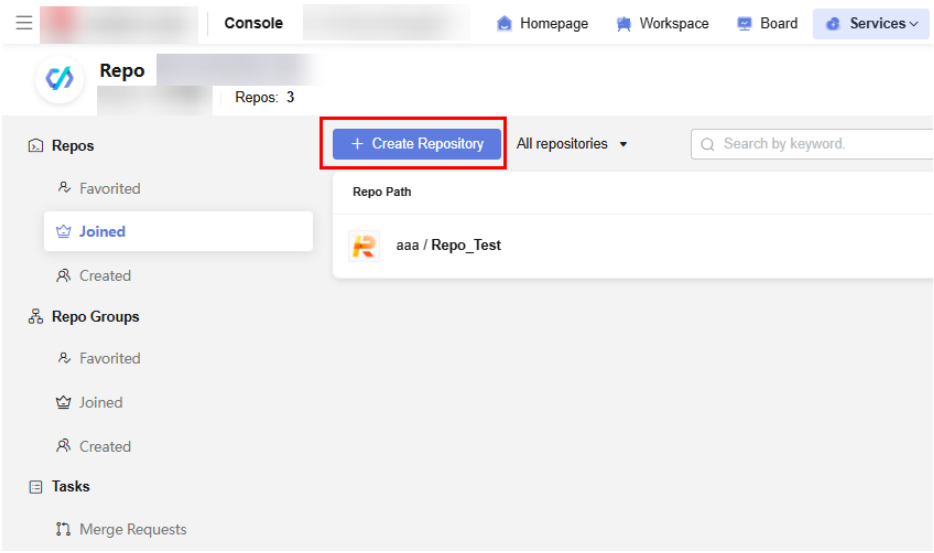
Step 7 Run the **git push -u origin master** command to push the local Git repository to the code repository created in CodeArts Repo.

----End

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 (master branch) 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

8.5 Migrating an SVN Code Repository

Constraints

- The repository domain must be connected to the service node.
- You need to create a new project or select an existing one.
- You have the permission to create a repository. If not, refer to **Configuring Repo-Level Permissions**.

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 8-3**.

Table 8-3 Parameters for importing the SVN repo

Parameter	Description
Source Repository URL	Mandatory. Specify the repo path to be imported. The source repo path must start with **http://**.
Verification to Access Source Repo	Mandatory. There are two cases: • 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

Related Document

- 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**.

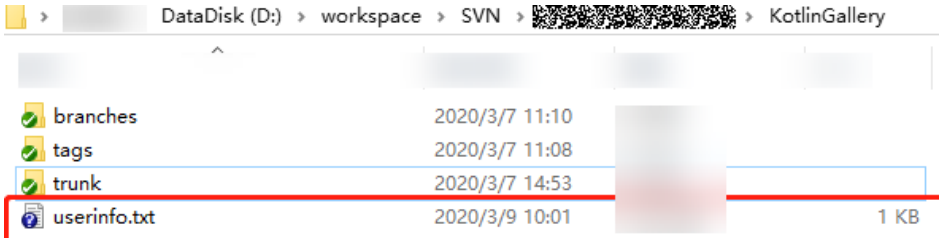
Using Git Bash to Import an SVN repo to CodeArts Repo

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:

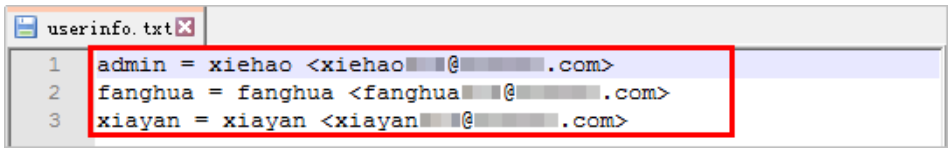
```
svn log --xml | grep "^<author" | sort -u | \awk -F '<author>' '{print $2}' | awk -F '</author>' '{print $1}' > userinfo.txt
```

   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.



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
   ```

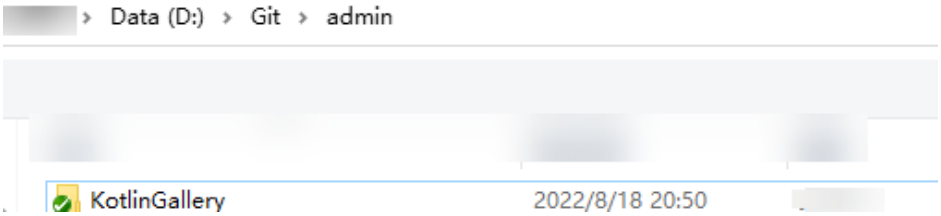
3. Start the Git Bash client in the directory and run the following command to clone a Git repo:

   ```
   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
```

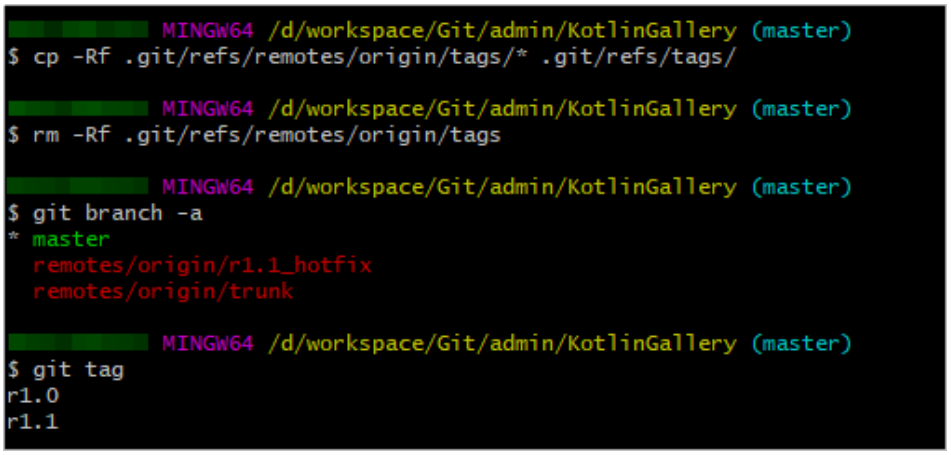
As shown in the preceding figure, all directory structures in the SVN are successfully migrated in the form of Git branches.

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
```



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

8.6 Syncing Repo Settings

Syncing Repo Settings

Table 8-4 Parameters for syncing repo settings

Parameter	Description
Branch	Mandatory. The options are as follows: • **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. 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.

Table 8-5 Repository synchronization settings

Parameter	Description
Branch	Mandatory. The options are as follows: • **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.

Parameter	Description
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.

8.7 Verifying the Import Permission

The following permission verification methods are supported:

- 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 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**.

8.8 Entering Basic Information for a Repository

8.8.1 Entering Basic Information for an Imported Repository

Table 8-6 Entering basic information for a new 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: • **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.

8.8.2 Configuring Basic Information for a New Repository

Table 8-7 Enter the basic information about the repository to be imported.

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.
Repository Name	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 (.).

Parameter	Description
Visibility	Optional. Indicates the visible scope of the source repo. The options are as follows: • **Private**: Only members of this repository can access it and commit code. • **Public (read-only) for project members** • **Public (read-only) for tenant members** • **Public (read-only) for all guests**
Scheduled Sync	Optional. If you select this option, the imported repo is an image repository, which 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. If no endpoint is used to import the repository, you do not need to configure **Scheduled Sync**.

9 Configuring Repository Settings

9.1 Configuring Repository Policies

9.1.1 Configuring Protected Branch Rules

Constraints

Table 9-1 Constraints on configuring protected branch rules

Item	Description
Permission constraint	To configure protected branch rules for a repository, you need to have the necessary permissions. For details, see **Configuring Repo-Level Permissions**.
Function constraint	Only one protected branch rule can be created for a branch. Otherwise, the error message indicating that the operation failed and the protected branch *xxx* already exists is displayed.

Configuring a Protected Branch

Go to the repository details page and choose **Settings** > **Policy Settings** > **Protected Branches**.

If you select **Inherit from project**, the settings of the project are automatically inherited and cannot be modified. If you do not select this option, see **Step 2**.

Managing a Protected Branch

Click in the row where the protected branch resides to modify the protected branch rule.

Click in the row where the protected branch resides to delete the protected branch rule.

9.1.2 Configuring Protected Tag Rules

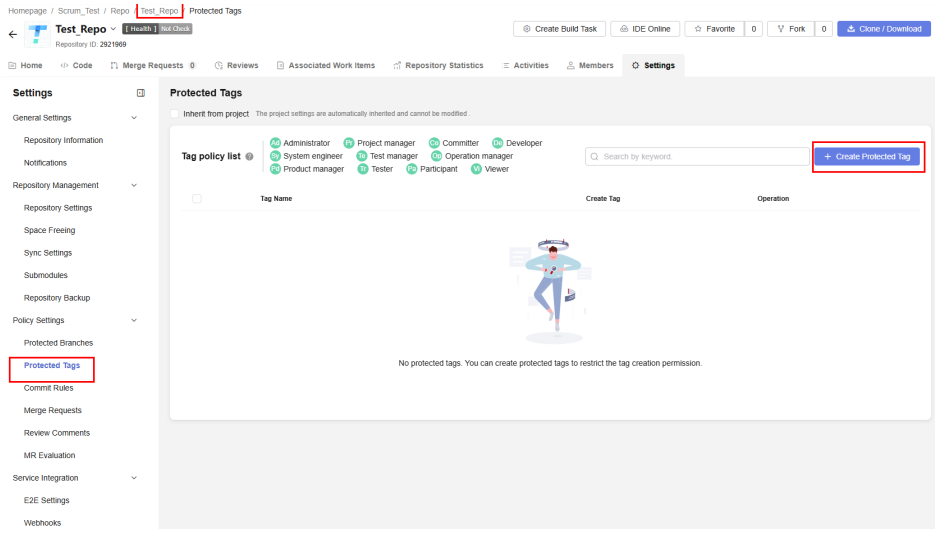
Constraints

Table 9-2 Constraints on configuring protected tag rules

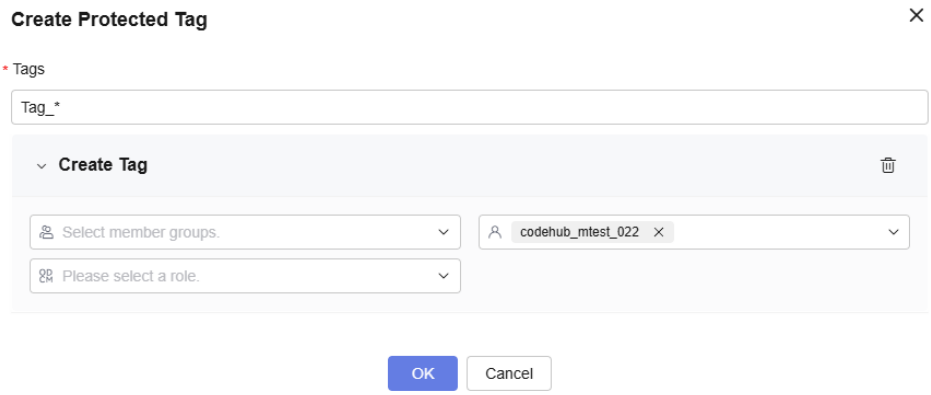
Item	Description
Permission constraint	To configure protected tag rules for a repository, you need to have the necessary permissions. For details, see **Configuring Repo-Level Permissions**.
Function constraint	Only one tag rule can be created for a tag. Otherwise, an error message is displayed, indicating that the protected tag name has already been used.

Creating Protected Tag Rules for a Repository

Step 1 Go to the homepage of the repository **Test_Repo**, choose **Settings** > **Policy Settings** > **protected tags**, and click **Create Protected Tag**.



Step 2 Set parameters by referring to **Table 2 Parameters for creating a protected tag**. As shown in the following figure,**Tag_*** indicates that all branches starting with **Tag_** are protected tags, and only **codehub_mtest_022** can be **created**.



----End

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.
Add Permissions	Mandatory. Roles allowed to create protected tags. You can select member groups, members, or roles from the drop-down list.

Managing Protected Tag Rules

Click in the row where the protected tag rule is located to change the role that is allowed to submit tags.

Click in the **Operation** column of a protected tag rule to delete it.

9.1.3 Configuring Code Commit Rules

Constraints

To configure commit rules for a repository, you need to have the necessary permissions. For details, see **Configuring Repo-Level Permissions**.

Creating Commit Rules for a Repository

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 repo homepage of the code to be configured and choose **Settings** > **Policy Settings** > **Commit Rules**.

Step 1 Set commit rules by referring to **Table 5-2**. This configuration takes effect for all branches in the repository.

Step 2 If you want to configure commit rules for a specified branch, click **Create Commit Rules**, and refer to **Table 5-3** for filling out the parameters. For common regular expression examples, see **Table 5-4**.

----End

If you want to manage the created code commit rules, see **Managing Commit Rules**.

9.1.4 Configuring Repository-Level Merge Request Rules

Constraints

To configure merge request rules for a repository, you need to have the necessary permissions. For details, see **Configuring Repo-Level Permissions**.

Configuring Merge Request Rules

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**.

For details, see **Step 1** to **Step 3**.

9.1.5 Configuring Review Comment Rules

Overview

Formulating clear review standards and rules safeguards the quality of code and eliminates potential vulnerabilities. In addition, unified review comment rules improve team cooperation efficiency by reaching consensus. Through code review, experienced developers can transfer their knowledge to newer team members.

Before configuring review comment rules, view **constraints** and configure review comment rules by referring to **Setting Review Comments**.

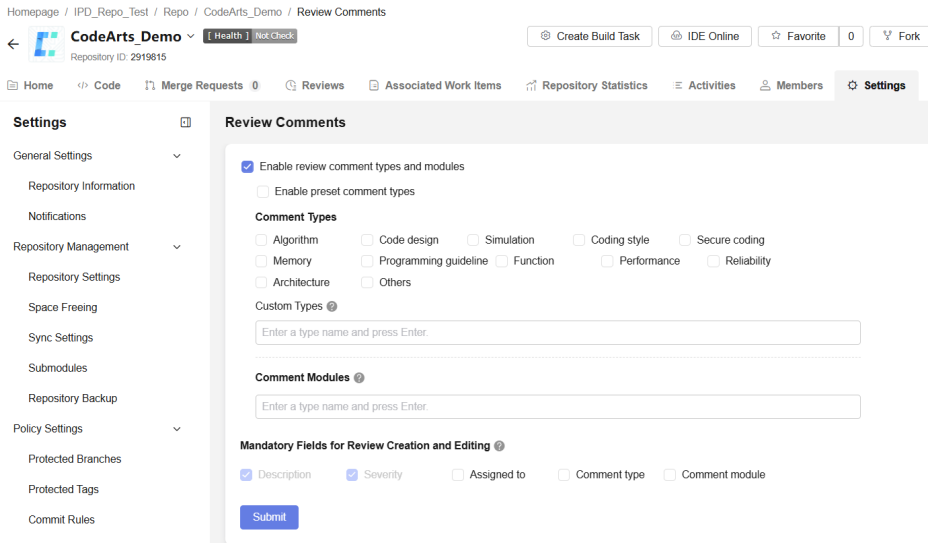
Constraints

Table 9-4 Constraints on configuring review comment rules

Item	Description
Package constraint	If your package is the professional or enterprise edition, you can configure review rules.
Permission constraint	You must have the set permission to configure review rules for the repository. For details, see **Configuring Repo-Level Permissions**.
Function constraint	The settings take effect only for the repository configured.

Setting Review Comments

Go to the details page of the target repository, choose **Settings** > **Policy Settings** > **Review**, and set review rules according to the following steps.



Step 1 If **Enable comment types and modules** is selected, you can continue to set review comment rules.

Step 2 If **Enable preset comment types** is selected, the preset review comment types are used. If not, go to step 3.

Step 3 Select the review comment types to be configured, set **Custom Types**, and press **Enter** to save the name. The name can contain a maximum of 200 characters and cannot contain colons (:). Review comment types should be separated by commas (,). A maximum of 20 review comment types can be created and must be unique.

Step 4 Set **Comment Modules**. Enter a name and press **Enter**. A maximum of 200 characters are allowed. The module names can be separated by commas (,). A maximum of 20 module names are allowed and must be unique.

Step 5 Select **Mandatory Fields for Review Creation and Editing** as required. Click **Submit** to save the settings.

----End

9.1.6 MR Evaluation

Overview

You can configure this feature to evaluate MRs by dimension on the merge request details page.

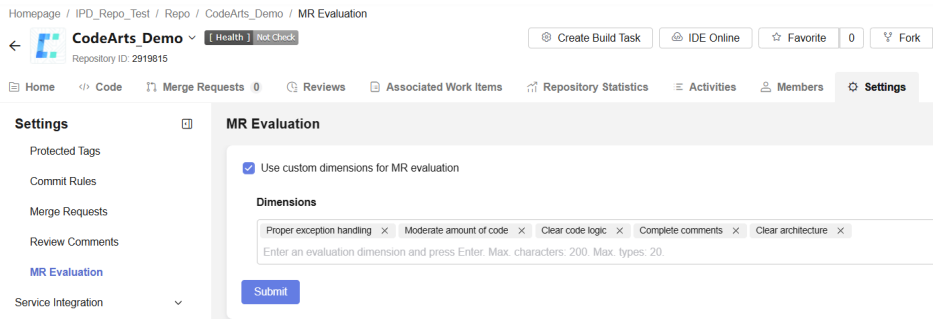
Constraints

Table 9-5 Constraints on configuring MR evaluation

Item	Description
Package constraint	If your package is the professional or enterprise edition, you can configure the MR evaluation.
Permission constraint	You must have the set permission to configure MR evaluation for the repository. For details, see **Configuring Repo-Level Permissions**.

Setting MR Evaluation

Step 1 Go to the target repository homepage. Choose **Settings** > **Policy Settings** > **MR Evaluation**.



Step 2 If you select **Use custom dimensions for MR evaluation**, you can set MR evaluation in multiple dimensions. There are default MR evaluation names in several dimensions. You can also enter a dimension name and press **Enter** to save it. The name can contain a maximum of 200 characters. A maximum of 20 dimensions can be created. If **Use custom dimensions for MR evaluation** is not selected, the single-dimension MR evaluation will be used.

----End

9.2 Configuring the Repository Settings

9.2.1 Configuring Repository Information

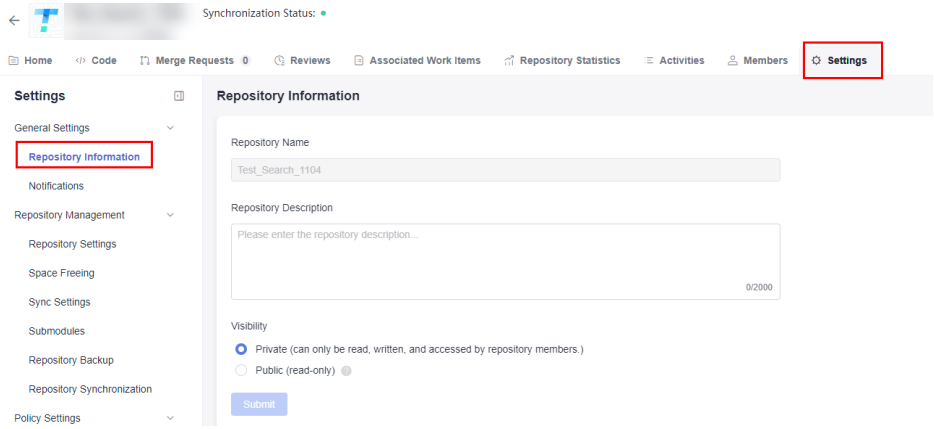
Constraints

- You must have the set permission on the repository to configure the repository information. For details, see **Configuring Repo-Level Permissions**.

Configuring Repository Information

Go to the repository homepage to be configured, as shown in the following figure. In the navigation pane, choose **Settings** > **General Settings** > **Repository Information**. You can modify the repository's description and visibility on this page.

Figure 9-1 Repository information settings page



9.2.2 Setting Notifications for Repositories and MRs

CodeArts Repo can push repository and merge request notifications by email or WeCom. You can enable the notifications as required.

- Configure the MR notification of the repository by referring to this section.

Requirement

Members in the repository can view this page. Only roles with the set permission in CodeArts Repo can configure the notification settings of the repository. For details, see **Configuring Repo-Level Permissions**.

Configuring Email Notifications

Go to the homepage of the repository to be configured. Choose **Settings** > **General Settings** > **Notifications** from the navigation bar. Set the notification

type to **Email**. For details about the email notification parameters, see **the following table**.

Table 9-6 Parameters for email notifications

Parameter	Description
Repository	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: • **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	Optional. You can select the following options as needed. • **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.

Configuring WeCom Notification Settings for the Repository

Table 9-7 Parameters for setting WeCom notifications

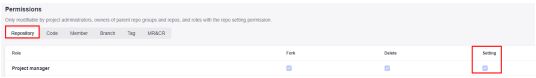
Parameter	Description
Webhook URL	Mandatory. Used to identify webhook address of the robot added to the CodeArts Repo member group with a maximum of 500 characters.

Parameter	Description
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. • **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. • **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**.

9.2.3 Configuring the Repository Settings

Constraints

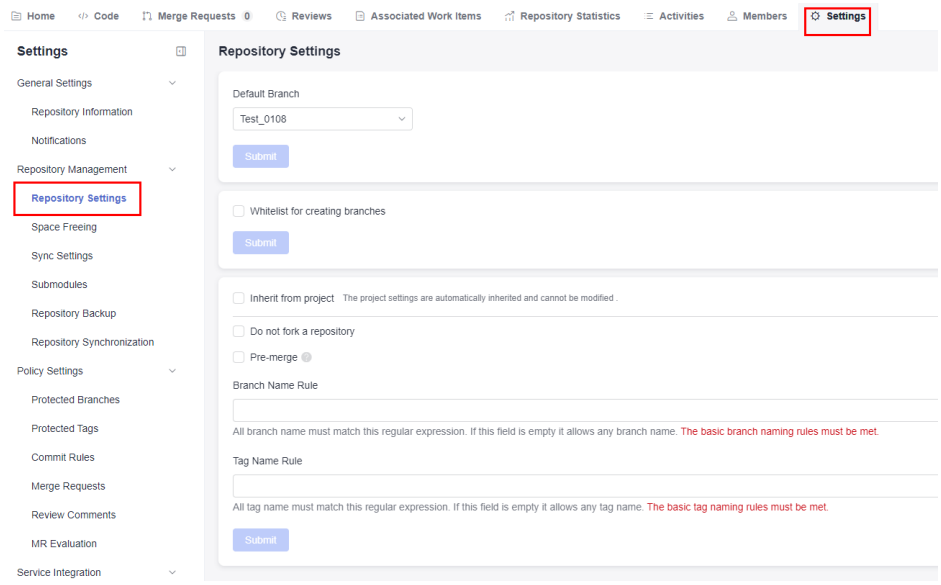
Table 9-8 Constraints on configuring repository permissions

Item	Description
Permission constraint	You must have the set permission on the repository. If you do not have the permission, configure the permission by referring to **Configuring Repo-Level Permissions**. 
Function constraint	If **Force inherit** is selected in the project-level **Repo Settings**, **Repo Settings** is not supported at the repository level.

Repository Settings

Step 1 If the project-level configuration is not inherited, set parameters by referring to this section. Go to the target repository homepage. Choose **Settings** > **Repository Management** > **Repository Settings**, as shown in the following figure.

Figure 9-2 Repository settings page



Step 2 Set **Default Branch** to the target branch. For example, if you set the default branch to **Test_0108**, the selected branch will be displayed as the default target branch on the **Code** page or when you create a merge request.

Step 3 If **Whitelist for creating branches** is selected, only repository members with the developer role can create branches in this whitelist. Non-developers will not be displayed and will not take effect even after configuration.

Step 4 If you select **Inherit from project**, the project settings are automatically inherited. If you do not select this option, set parameters by referring to **the table**.

Table 9-9 Repository settings description

Parameter	Description
Do not fork a repository	After this option is selected, the repository cannot be forked.
Pre-merge	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. For details, see **Example**.

Parameter	Description
Branch Name Rule	All branch names must match the regular expression with max. 500 characters. For details, see **Table 5-4**. If this field is left blank, any branch name is allowed. The rules must meet the following branch naming rules: • Max. 500 characters. • Do not start with **refs/heads** and **refs/remotes/** nor end with period (.), slash (/), and **.lock**. Spaces and the following characters are not supported:. [\ < ~ ^: ? ( ) ' " |. • The name of a new branch cannot be the same as that of an existing branch or tag.
Tag Name Rule	All tag names must match the regular expression specified by this parameter. For details, see **Table 5-4**. 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. The rules must meet the following tag naming rules: • Max. 500 characters. • Do not start with **refs/heads/**, **refs/remotes/** nor end with **. / .lock.** Spaces and the following characters are not supported:. [\ < ~ ^: ? ( ) ' " |. • The name of a new tag cannot be the same as that of an existing branch or tag.

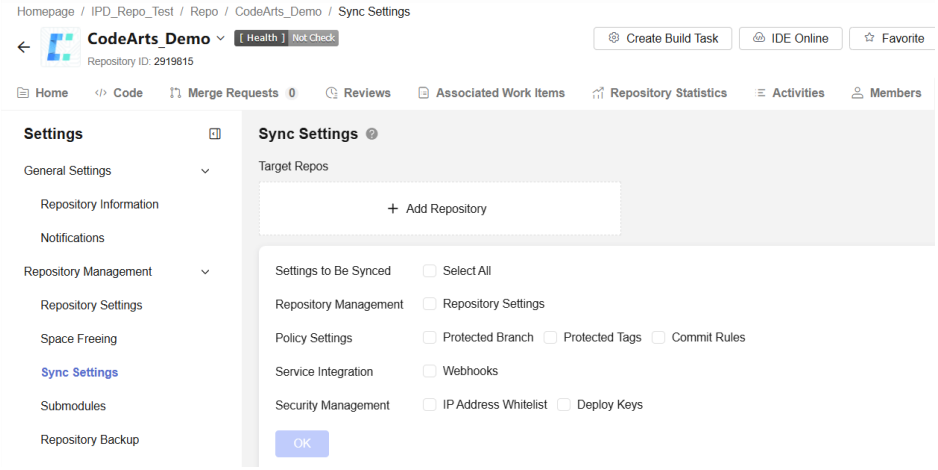
----End

9.2.4 Configuring Repository Synchronization

Overview

CodeArts Repo allows you to sync the settings of the current repository to other repositories as you want.

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



Constraints

Table 9-10 Constraints on configuring synchronization settings

Item	Description
Permission constraint	If you have the set permission on the target repository, you can configure the synchronization settings. For details about how to configure permissions, see **Configuring Repo-Level Permissions**.
Function constraint	This function supports cross-project synchronization but does not support cross-region synchronization.

Configuring Synchronization Between Repos

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.

Related Document

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.

9.2.5 Setting Submodule

Overview of Configuring a Submodule

A submodule is a Git tool used to manage shared repositories for higher team efficiency. Submodules allow you to keep a shared repository as a subdirectory of your repository. You can isolate and reuse repositories, and pull latest changes from or push commits to shared repositories.

When the repository **Test_Fir** needs to contain and use the repository **Test_Sec** (a third-party library or a library developed for multiple parent projects), and you want to treat them as two independent projects and want to use **Test_Sec** in **Test_Fir**, you can use the submodule function of Git. 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 repo 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**.

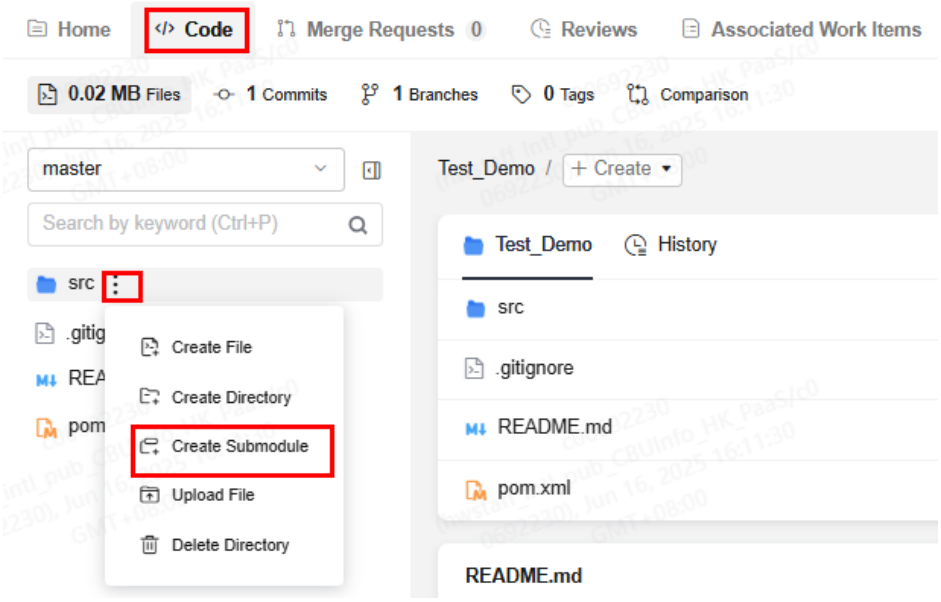
Constraints

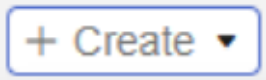
If the content of the **Test_Fir** repository will be used as a submodule of the **Test_Sec** repository, you need to have both the set and code commit permissions for the **Test_Sec** repository. For details, refer to **Configuring Repo-Level Permissions**.

Adding a Submodule on the Repo Page

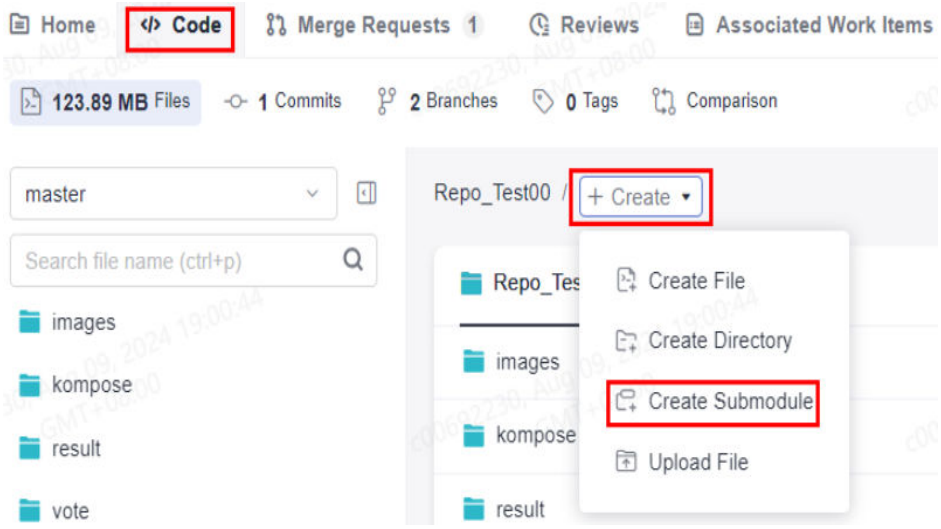
You can add a submodule with any of the following methods as needed.

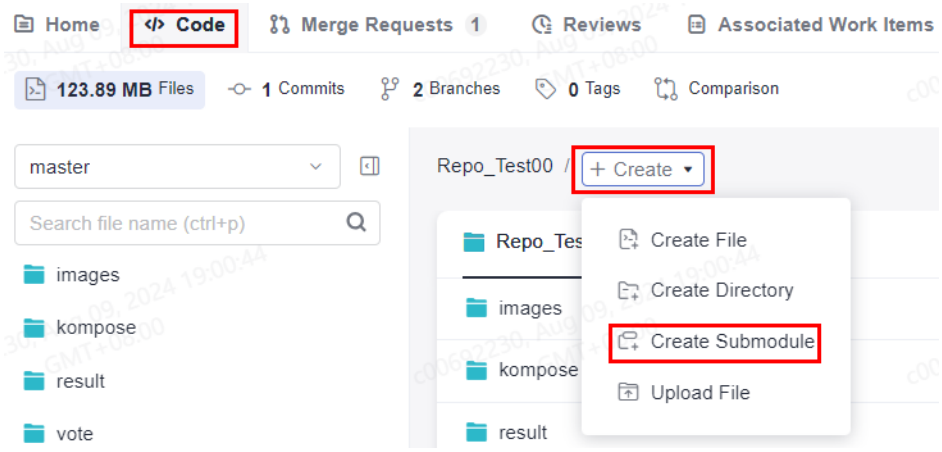
- Go to the repository homepage to which a submodule is to be added, and click **Code**, as shown in the following figure. Click the button, select **Create Submodule**, and set parameters by referring to **the table**.



- Go to the repository homepage to which a submodule is to be added, and click **Code**, as shown in the following figure. Click the 

button, select **Create Submodule**, and set parameters by referring to **the table**.





- Access the repository homepage to which a submodule is to be added, choose **Settings** > **Repository Management** > **Submodules**, click **Create Submodule**, and set parameters by referring to **the following table**.

  Configure the following parameters and click **OK**. 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 .

Table 9-11 Parameters of creating a submodule

Parameter	Description
Submodule Repo Path	Mandatory. Select a repository as the sub-repository. You can select a cross-project repository.
Submodule Repo Branch	Mandatory. Select the target branch of the submodule to be synchronized to the parent repository.
Submodule File Path	Mandatory. Path of the submodule file in the repository. Use **/** to separate levels.
Commit Message	Remarks for creating a submodule. You can find the operation in the file history. The value contains a maximum of 2,000 characters.

Adding a Submodule on Git

Step 1 Add a submodule.

```
git submodule add <repo> [<dir>] [-b <branch>] [<path>]
```

Example:

```
git submodule add git@***.***.com:****/WEB-INF.git
```

Step 2 Pull 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 **.gitsubmodule** 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
   ```

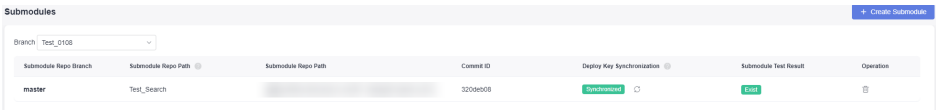
----End

Editing a Submodule

You must have the set permission to edit submodules. For details, see **Configuring Repo-Level Permissions**.

As shown in the following figure, **Submodule Repo Path** indicates the Git address of the sub-repository. You can click **Commit ID** to view the historical commits of the sub-repository. Click in the **Deploy Key Synchronization** column to synchronize the deploy key of the parent repository to the child repository. The child repository will inherit the deploy key of the parent repository. If the **Submodule Test Result** column shows **Exist**, it indicates that the sub-repository is available. To delete the submodule, click in the **Operation** column.

Figure 9-3 Submodule settings page



9.2.6 Pre-Merging an MR

Overview

Pre-merging an MR generates a temporary merge node in the repository. When the code in an MR has not been merged, you can download the pre-merge code via plugin scripts such as custom webhooks or pipelines for code building.

Advantages of pre-merge

In the MR merging process involving dozens or hundreds of servers for code building, the local client pre-merging result may be different from that generated by the server and yield inaccurate build. Enabling MR pre-merge can solve this real-time issue, as the build script commands are simpler compared to local pre-merging scripts, making it easier for developers to get started.

Constraints

- Enable **Pre-merge**.
- You need to have the set permission.

Example

The following example shows the difference between the scripts with **Pre-merge** enabled and unenabled. The former one is more simple and efficient.

- **Pre-merge**: When an MR is created, a temporary merge node will be generated on the server. You can download the code that has been temporarily merged. Steps are provided as follows:

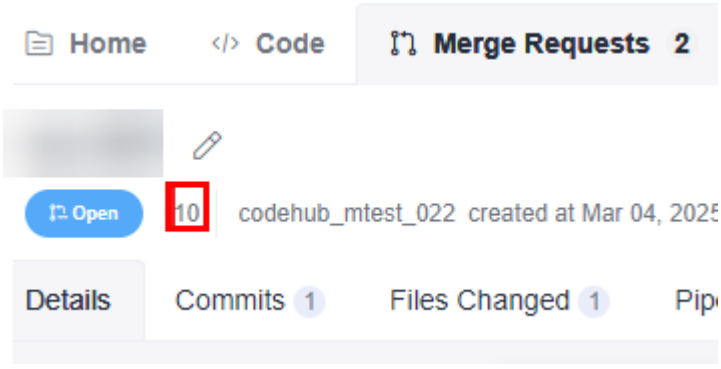
Step 1 Initialize the local repository. In the command, **repo_url** indicates the address of the target repository.

```
git init
git remote add origin ${repo_url}
```

Step 2 Pulls the temporary merge node from the server to the local branch. As shown in the following figure, **10** under the merge request header is displayed as **repo_MR_iid**. **merge-requests/${repo_MR_iid}/merge** indicates the temporary merge node, and **${repo_MR_iid}/merge** indicates the local branch.

```
git fetch origin +refs/merge-requests/${repo_MR_iid}/merge:refs/remotes/origin/${repo_MR_iid}/merge
```

Figure 9-4 MR IID



Step 3 Check out the branch and the pre-merged code is obtained.

```
git checkout ${repo_MR_iid}/merge
```

----End

If MR **pre-merge** is not enabled, you need to download the code of the MR source and target branches on the client and merge on your build executor. To update the information, perform the following operations:

Step 1 Initialize the local repository. In the command, **repo_url** indicates the address of the target repository.


```
git init
git remote add origin ${repo_url}
```

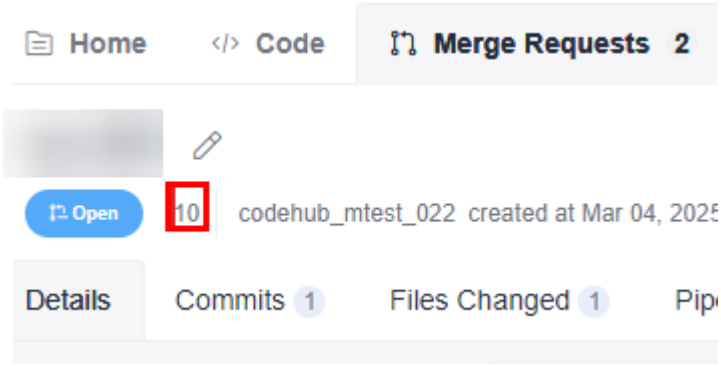
Step 2 Pull and check the target branch of the MR. The **repoTargetBranch** represents the target branch of the MR.

```
git fetch origin +refs/heads/${repoTargetBranch}:refs/remotes/origin/${repoTargetBranch}
git checkout ${repoTargetBranch}
```

Step 3 Pull the source branch of the MR to the local branch. In the following figure, **10** under the MR header is shown as **repo_MR_iid**. **merge-requests/${repo_MR_iid}/head** indicates the source branch of the MR, and **${repo_MR_iid}/head** the local branch.

```
git fetch origin +refs/merge-requests/${repo_MR_iid}/head:refs/remotes/origin/${repo_MR_iid}/head
```

Figure 9-5 MR IID



Step 4 Merge locally and the pre-merged code is generated.

```
git merge refs/remotes/origin/${repo_MR_iid}/head --no-edit
```

----End

9.3 Backing Up a Repository

Constraints

- You must have the set permission to back up repositories. For details, see **Configuring Repo-Level Permissions**.
- Ensure the repository connectivity.
- You need to enable the destination region for backup.

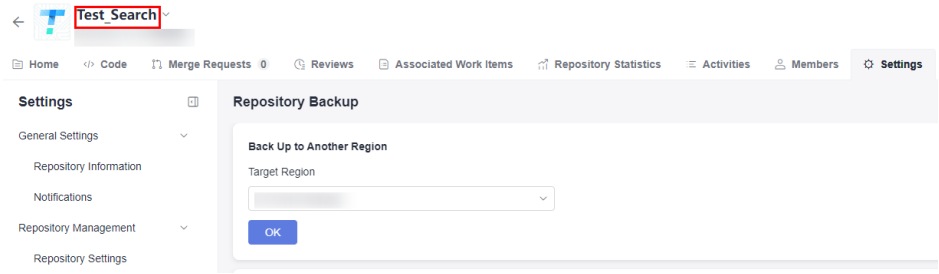
Steps

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:

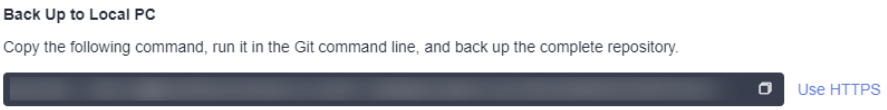
- **Back Up to Another Region**: As shown in the following figure, select the region to be backed up, indicating that the current repository will be backed up to another region.

Figure 9-6 Backup to another region



- **Back up to Local PC**: Back up the repository to your local PC. As shown in the following figure, copy the clone address using HTTPS or SSH. After the clone command is generated, paste it to the local Git client and run the command.

Figure 9-7 Backup to local PC



9.4 Repository Integration with Other Services

9.4.1 E2E Settings

Constraints

Table 9-12 Constraints on repositories' E2E settings

Item	Description
Function constraint	The repositories of Kanban projects do not support E2E settings.
Permission constraint	You must have the set permission on repositories to configure E2E settings. For details, see **Configuring Repo-Level Permissions**.

Configuring E2E Settings

If you select **Inherit from project**, the settings of the project are automatically inherited and cannot be modified. If this parameter is not selected, go to the target repository homepage, choose **Settings** > **Service Integration** > **E2E Settings**, and set parameters based on **E2E Settings**.

9.4.2 Webhook Settings

Constraints

You must have the set permission on repositories to configure webhook settings. For details, see **Configuring Repo-Level Permissions**.

Configuring a Webhook

Go to the repository homepage to be configured, choose **Settings** > **Service Integration** > **Webhooks**, click **Add Webhook**, and configure the webhook accordingly. For details, see **Configuring Webhook Settings**.

9.5 Viewing Activities

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

- **All**: displays all operation records of the repository including push, merge request, review comments, and members up to the present.
- **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**: displays all review comments of the repository. You can click the commit ID 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.

9.6 Viewing Repository Statistics

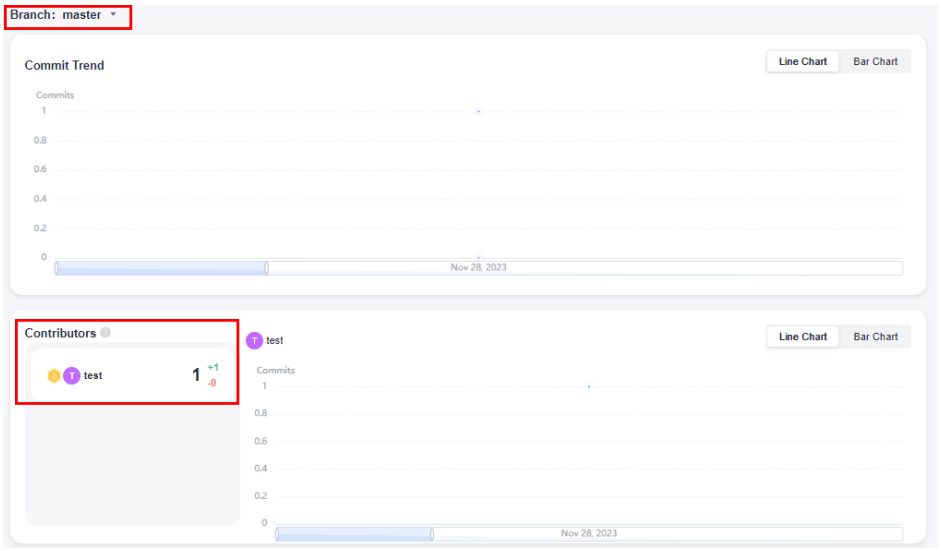
Constraints

- 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.
- Commits (an operation that combines two or more historical development records) of the merge node are not counted.

Viewing Repository Statistics

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

- **Overview**: **Repo Used**, **LFS used**, **Members**, **Commits**, **Branches**, **Tags**, **Merge Requests**, and **Review Comments**.
- **Languages**: displays the distribution of each language in the current branch of the repository. Repository members can trigger code contribution statistics and language ratio statistics. The language distribution of the repository is calculated based on the number of code lines in the file of the default branch. Hidden files and **README.md** files are not counted. CodeArts Repo collects statistics on more than 100 languages.
- **Commit Trend**: You can select the branch to be collected in **Branch**. The statistics can be displayed in **Line Chart** or **Bar Chart**.
- **Contributors**: Statistics on the contribution of code committers in a branch (number of commits and number of code lines). 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). As shown in the following figure, the committer **test** added one line of code to the **master** branch and didn't delete any line of code.



- **Commits Overview**: Statistics on the code commit frequency and number of contributors by week, day, and hour in either **Line Chart** or **Bar Chart**.

10 Hierarchical Repository Management

10.1 Creating a Repository Group

Constraints

Table 10-1 Constraints on creating a repository group

Item	Description
Permission constraint	You have the permission to create a repository group. If not, refer to **Configuring Repo-Level Permissions**.
Function constraint	• **A repository group supports a maximum of five levels of directories.** • You can only create a repository group either in a new project or an existing one.

Overview

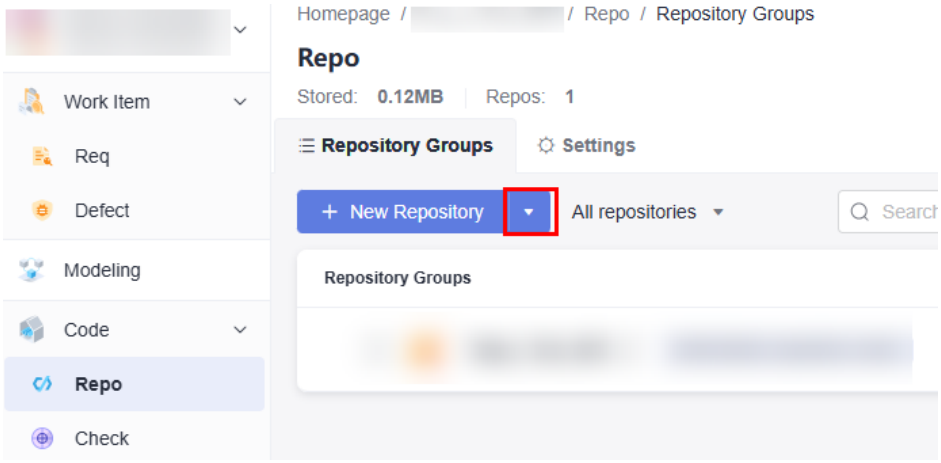
A repository group consists of one or more repositories. You can configure and manage repository rules for repositories or child repository groups in a repository group, including commit rules and member permissions.

Creating a Repository Group

Step 1 You can create a repository group in either of the following ways:

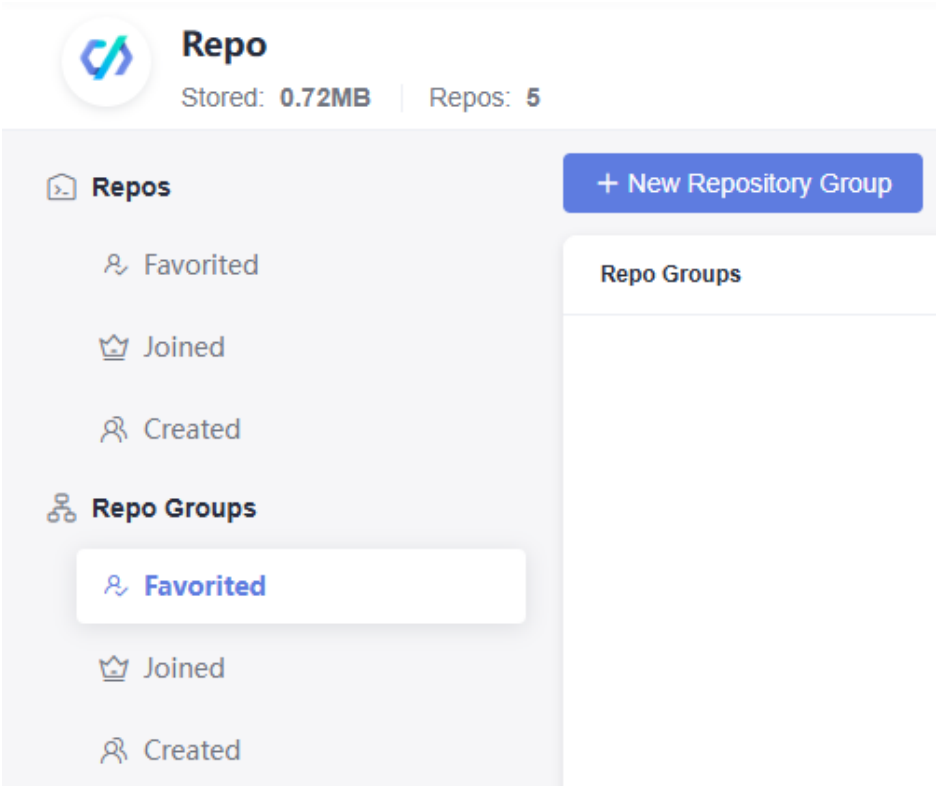
- As shown in the following figure, go to a project or parent organization, click the drop-down list box next to **Create Repository** and select **Create Repository Group**. The page for creating a repository group is displayed.

Figure 10-1 Project-level CodeArts Repo homepage



- Go to the CodeArts Repo homepage, click any line under **Repository Groups**, and click **Create Repository Group**, as shown in the following figure.

Figure 10-2 CodeArts Repo homepage



Step 2 Enter the basic information according to the following table and click **OK**. A repository group supports a maximum of five levels of directories.

Table 10-2 Parameters for creating a repository group

Parameters	Description
Project	• If you access the page from a project or parent organization, this parameter is set by default. • If you access the page from the CodeArts Repo homepage, select an existing project from the **Project** drop-down list or **Create Project**.
Path	Optional. The default value is **/**, indicating that the repo group is the first-level repo group after the repo group path is created. You can also select an existing repo group path from the drop-down list.
Name	Mandatory. 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	Optional. Describe your repository group. Max. 2,000 characters.
Visibility	Mandatory. You can choose **Private** (default) or **Public**. • **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.

----End

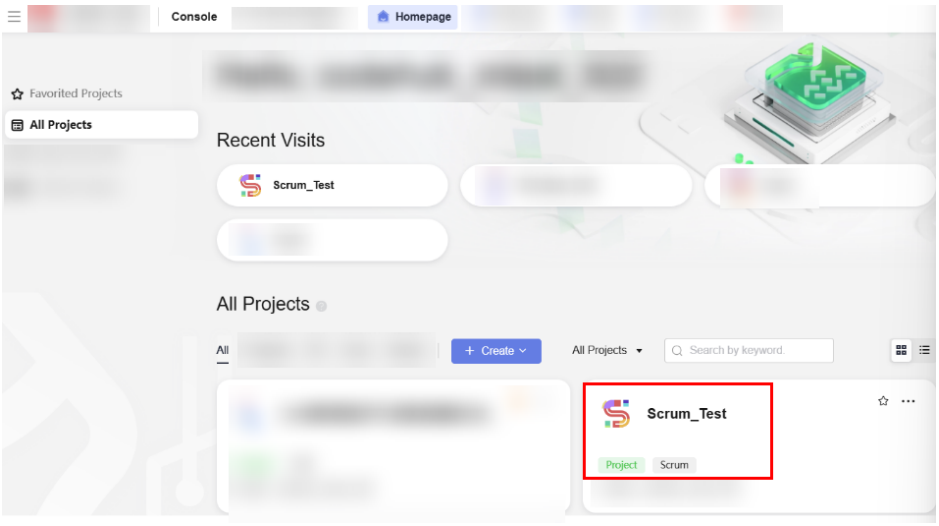
10.2 Using Repository Groups

10.2.1 Viewing the Repository Group List

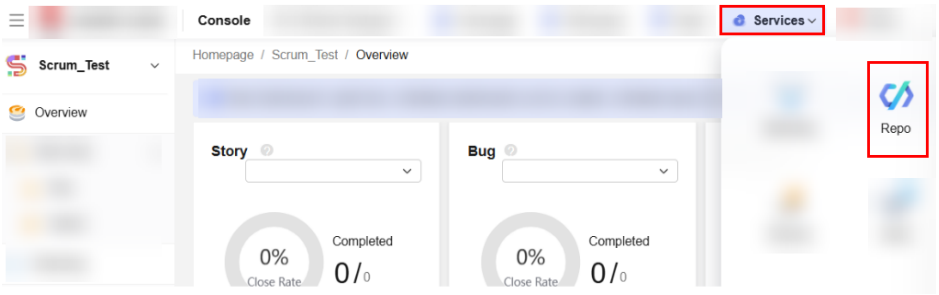
Accessing the Repository Group List

To view the **Joined** repository group list of a project, perform the following operations:

Step 1 On the CodeArts homepage, click the **Scrum_Test** project to be viewed.

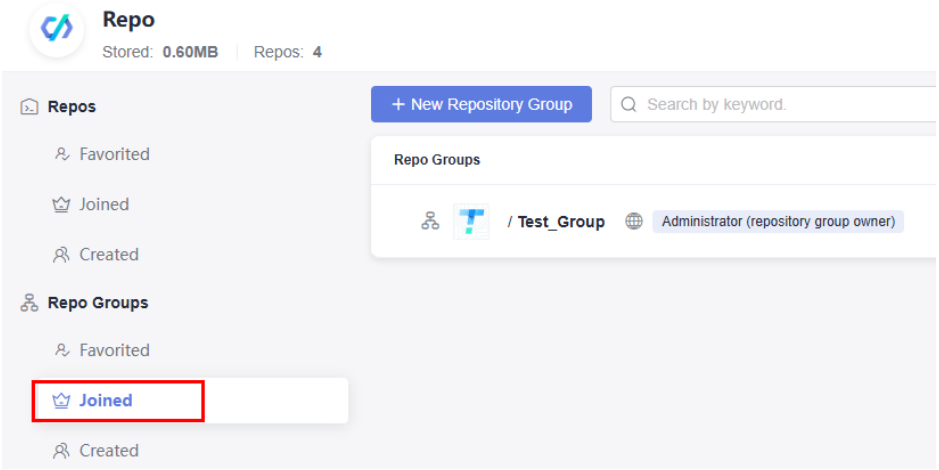


Step 2 On the **Scrum_Test** homepage, click **Services** in the navigation bar and choose **CodeArts Repo**.



Step 3 Choose **Repo Groups** > **Participated**. The repository groups that you have joined are displayed.

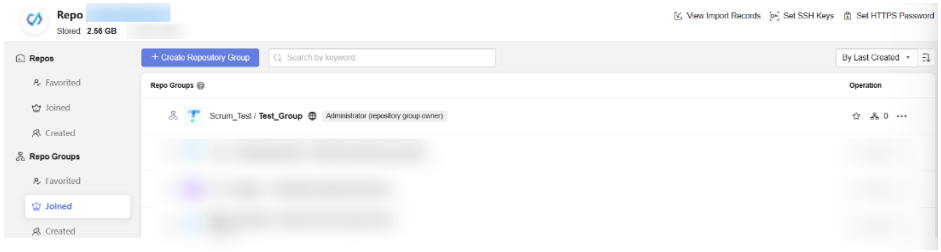
Figure 10-3 CodeArts Repo homepage



----End

Viewing Details on the Repository Group List Page

As shown in the following figure, you can create and configure a repository group on the repository group list page.



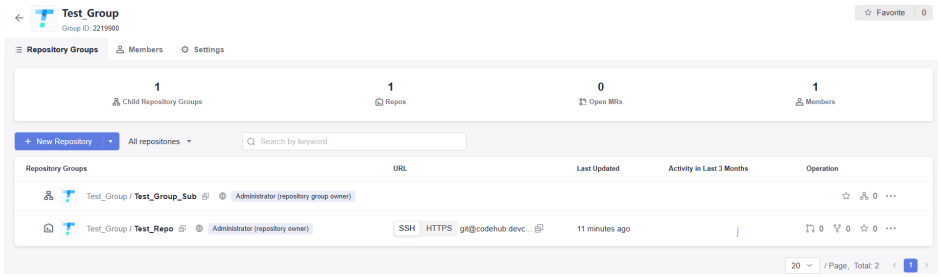
- + New Repository Group: Click this icon to access the page for creating a repository group.
- ☆: Click this icon to follow the repository group. You can click **Favorited** in the left navigation pane to view the repository group.
- : Click this icon on the right of a repository group to access the 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.

10.2.2 Viewing Repository Group Details

Viewing Repository Group Information on the Repository (Group) Tab Page

Click the name of the repository group to be viewed, for example, **Test_Group**. The repository (group) tab page of the repository group is displayed, as shown in the following figure. **Test_Group_Sub** is the child repository group of repository group **Test_Group**, and **Test_Repo** is the repository of repository group **Test_Group**.

Figure 10-4 Repo group details page



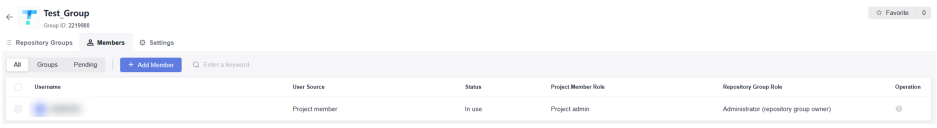
This tab page displays the **Test_Group**'s number of child repository groups, number of repositories, number of open MRs, and number of members in the repository group. You can perform the following operations:

- Click **Create Repository** to create a repository under **Test_Group**.
- Click the drop-down list box next to **Create Repository** to create a child repository group under **Test_Group**.
- Click **All repositories** to view all **Test_Group**'s repositories, unlocked repositories, and locked repositories in the current repository group. Enter the name of any repository in the search box on the right of **All repositories** and the searched repository will be displayed.
- Click in the row where the repository group **Test_Group_Sub** is located to favorite it. Click on the right of **Test_Group_Sub** to go to its details page.
- Click in the row where **Test_Group_Sub** is located to view the repositories, members, and settings. You can also create a new child repository group and a new repository under the group.
- Click in the row where **Test_Repo** is located to copy the SSH or HTTPS address of the repository. Click in the row where the repository is located to view all its merge requests. Click to go to the repository homepage. Click to favorite it.
- Click in the row where **Test_Repo** is located.
  - View Associated Items : Click this icon to go to the **Associated Items** tab page of the repository.
  - Manage Members : Click this icon to go to the **Members** tab page of the repository.
  - Delete Repository : Click this icon to delete **Test_Repo**.

Viewing Repository Group's Member Information on the Members Page

Click **Members** in the navigation bar to access the **Members** tab page of the repository group, as shown in the following figure.

Figure 10-5 Repository group member tab page

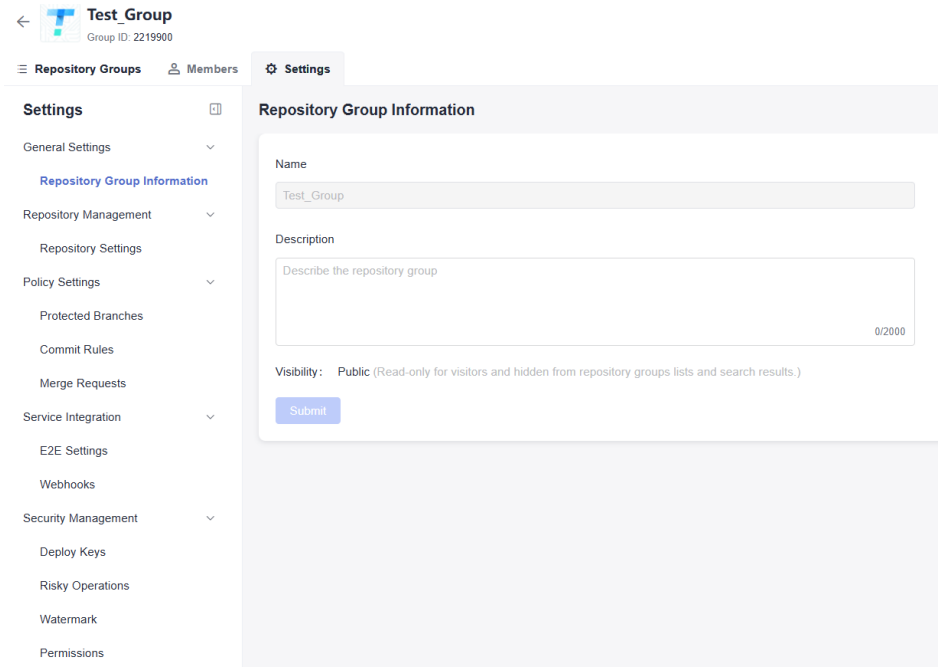


Viewing Repository Group Information on the Settings Tab Page

Constraints: Only the project manager and repository group administrator can view and configure the information on the **Setting**. For details about how to configure the permission to view the **Settings** page, see **Configuring Repo-Level Permissions**.

Click **Settings** in the navigation bar, as shown in the following figure.

Figure 10-6 Repo group settings tab page



10.2.3 Managing Repository Group Members

Before managing repository group members, view **Constraints**. If you meet the conditions, you can perform the following operations.

- **Adding a Member or Member Group to a Repository Group**
- **Editing Repository Group Members**
- **Removing a Member from a Repository Group**
- **Viewing Repository Group Member Information**

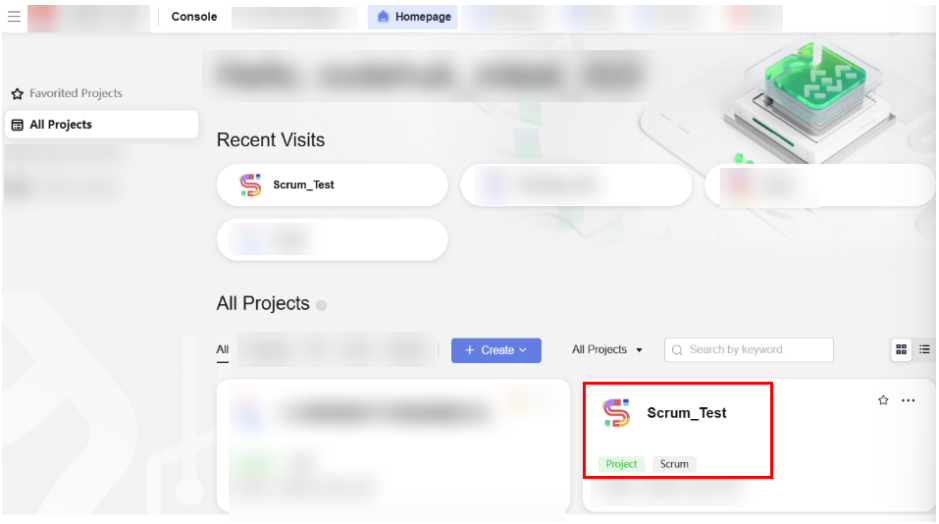
Constraints

Table 10-3 Constraints on repository group member management

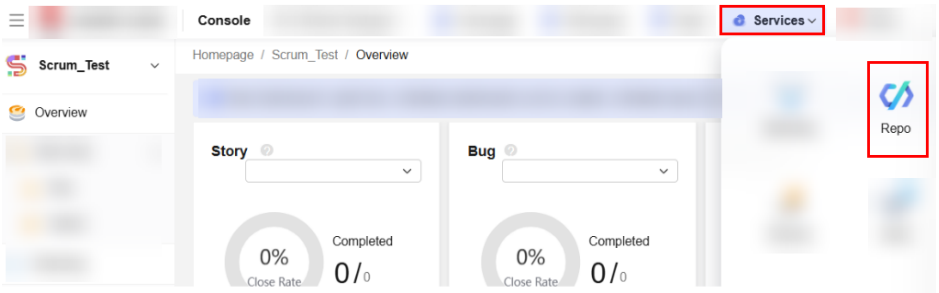
Item	Description
Function constraints	• Members in a parent repository group are inherited to its child repository groups or child repositories by default and cannot be deleted. • When a role in a project changes and is the same as that of the repository in the project, the role of the repository is updated accordingly. • For members inherited from the repository group or added through the member group, the role is subject to the latest role update. • 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. • If you want to delete a member added to the repository group through a member group, go to the member group to delete the member. • This member is from the upper-layer repository group and can only be deleted in that repository group.
Permission constraints	• As the administrator of this repository, the repository owner has full permissions for the repository by default 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 by default and cannot be deleted or modified.

Adding a Member or Member Group to a Repository Group

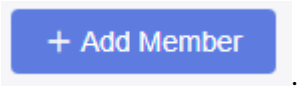
Step 1 On the CodeArts homepage, click the **Scrum_Test** project to be viewed as shown in the following figure.



Step 2 On the **Scrum_Test** homepage, click **Services** in the navigation bar and choose **CodeArts Repo**.



Step 3 Click the repository group name **Test_Group** to go to its details page.

Step 4 Click **Members** and click 

.

Step 5 On the **Members** tab page, search for the member to be added, select the member, and click **OK** to add the member to the repository group.

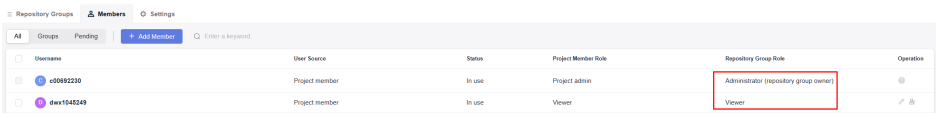
You can also click the **Member Groups** tab, select the member group to be added from the drop-down list, and click **OK** to add the member group to the repository group.

----End

Editing Repository Group Members

Constraint: As shown in the following figure, the repository group owner has the highest permissions of the repository group, child repository groups, and repositories, and the permissions cannot be edited. The viewer role cannot be edited, and the edit button is unavailable.

Figure 10-7 Administrator (repository group owner)

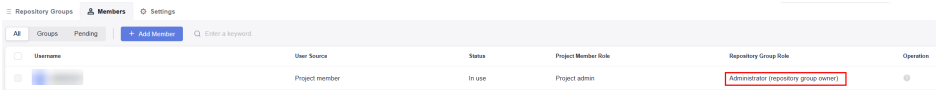


Click in the row where the member is located to change its role.

Removing a Member from a Repository Group

Constraint: As shown in the following figures, the owner of a repository group has the highest permissions of the repository group, child repository groups, and repositories, and the permissions cannot be removed.

Figure 10-8 Administrator (repository group owner)



Click in the row where the member is located to remove the member from the repository group. Click in the row where the member group is located to delete the repository group from the repository group.

Viewing Repository Group Member Information

Only repository group members can view the **Members** tab of the 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**, as shown in the following figure.

On this page, you can view the repository group member information listed in the following table.

Table 10-4 Viewing repository group member information

Parameter	Description
All	Displays the **Username**, **User Source**, **Status**, **Project Member Role**, **Repository Group Role**, and **Operation** of all members in the repository group.
Groups	Displays the **Member Group Name**, **Number of Members**, **Description**, and **Operation** of all member groups in the repository group.
Pending	Displays the **Pending** members who are about to join the repository group. The information includes **Username**, **Status, 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**.

----End

10.3 Managing Repository Groups

10.3.1 Repository Group Information

Constraints

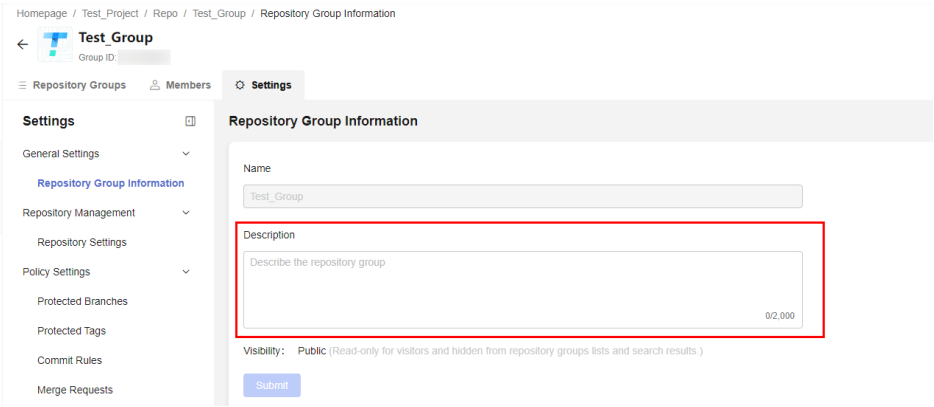
Table 10-5 Constraints on repository group information

Item	Description
Permission constraint	All members in the repository group can view the page. Only the project administrator and repository group creator have the permission to set repository group information. For details, see **Configuring a Repository Group's Permissions**.

Item	Description
Function constraints	• This setting takes effect only for the repository group configured. • The **Name** and **Visibility** of the repository group cannot be modified.

Modifying Repository Group Information

The project administrator or repository group creator accesses the repository group to be set, choose **Settings** > **General Settings** > **Repository Group Information** in the navigation pane, and modify **Description**.



10.3.2 Configuring Repository Settings in a Repository Group

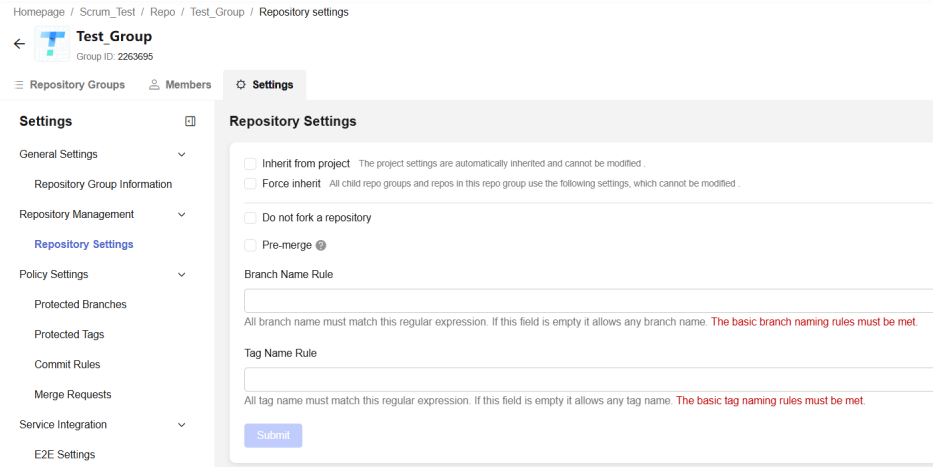
Constraints

Table 10-6 Constraints

Item	Description
Permission constraint	Repository group members who have the **setting** permission for repository groups can configure permissions on this page. For details, see **Configuring a Repository Group's Permissions**. Repository group members can only view this page.
Function constraint	If you select **Inherit from project**, the settings of the project will be used and cannot be changed.

Configuring Repository Settings

Step 1 Log in to the repository group **Test_Group** as a repository group member who has the set permission, and choose **Settings** > **General Settings** > **Repository Settings** in the navigation pane.



Step 2 If **Force inherit** is selected, the settings of the current repository group take effect for all the repositories in this repository group and its child repository groups.

If **Inherit from project** is not selected, you can configure repository settings by referring to **Configuring the Repository Settings**. For details about common problems during the configuration, see **Related Document**.

----End

10.3.3 Configuring Policy Settings for a Repository Group

10.3.3.1 Configuring Protected Branch Rules for a Repository Group

You can configure protected branch rules for all repositories in a repository group. Before the configuration, view **Constraints** and configure protected branches for the repository groups by referring to **Configuring Protected Branches for a Repository Group**.

Constraints

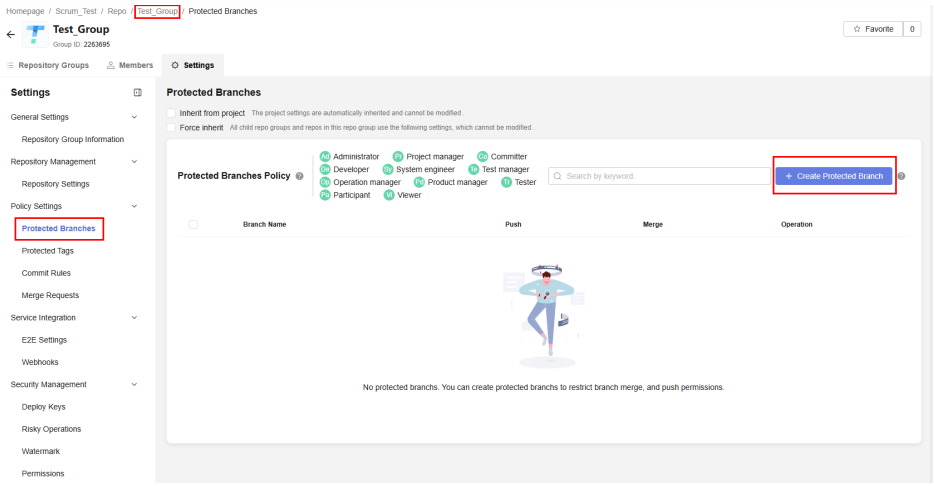
Table 10-7 Constraints on configuring protected branch rules

Item	Description
Permission constraint	To configure protected branch rules for repository under a repository group, you need to have the set permission for repository groups. For details, see **Configuring a Repository Group's Permissions**.

Item	Description
Function constraint	• Only one protected branch rule can be created for a branch. Otherwise, the error message indicating that the operation failed and the protected branch *xxx* already exists is displayed. • If you select **Inherit from project**, the settings of the project are automatically inherited and cannot be modified. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.

Configuring Protected Branches for a Repository Group

Step 1 Go to the details page of the repository group **Test_Group** to be configured, choose **Settings** > **Policy Settings** > **Protected Branches**, and click **Create Protected Branch**.



Step 2 If you do not select this option, see **Step 2**.

----End

10.3.3.2 Configuring Protected Tags for a Repository Group

You can configure protected tag rules for all repositories in a repository group. Before the configuration, view **Constraints** and configure protected tag rules for the repository groups by referring to **Configuring Protected Tag Rules for a Repository Group**.

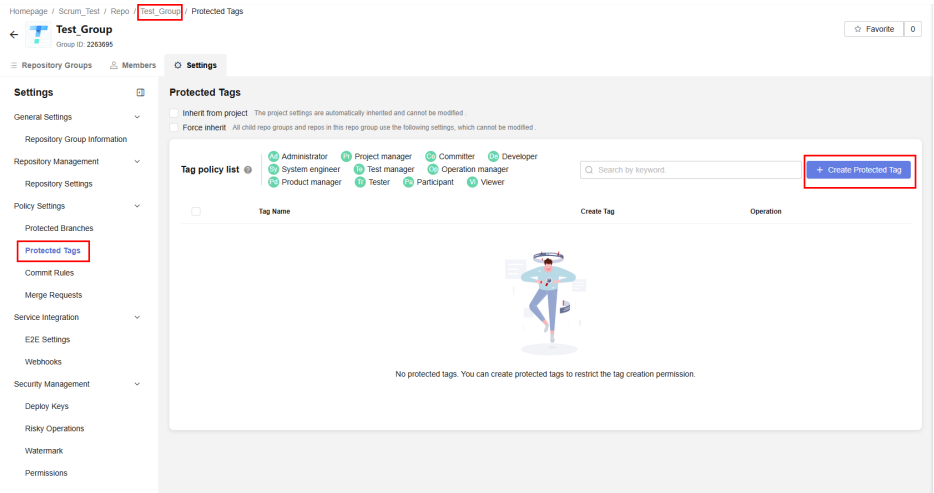
Constraints

Table 10-8 Constraints on configuring protected tags

Item	Description
Permission constraint	Repository group members who have the set permission for repository groups can configure permissions on this page. For details, see **Configuring a Repository Group's Permissions**. Repository group members can only view this page.
Function constraints	• Only one tag rule can be created for a tag. Otherwise, an error message is displayed, indicating that the protected tag name has already been used. • If you select **Inherit from project**, the settings of the project will be used and cannot be changed. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.

Configuring Protected Tag Rules for a Repository Group

Step 1 Go to the details page of the repository group **Test_Group** to be configured, choose **Settings** > **Policy Settings** > **Protected Branches**, and click **Create Protected Tag**.



Step 2 If you do not select this option, see **Step 2**.

----End

10.3.3.3 Configuring Commit Rules for a Repository Group

You can configure code commit rules for all repositories in a repository group. Before the configuration, view **Constraints** and configure protected tag rules for the repository groups by referring to **Configuring Protected Tag Rules for a Repository Group**.

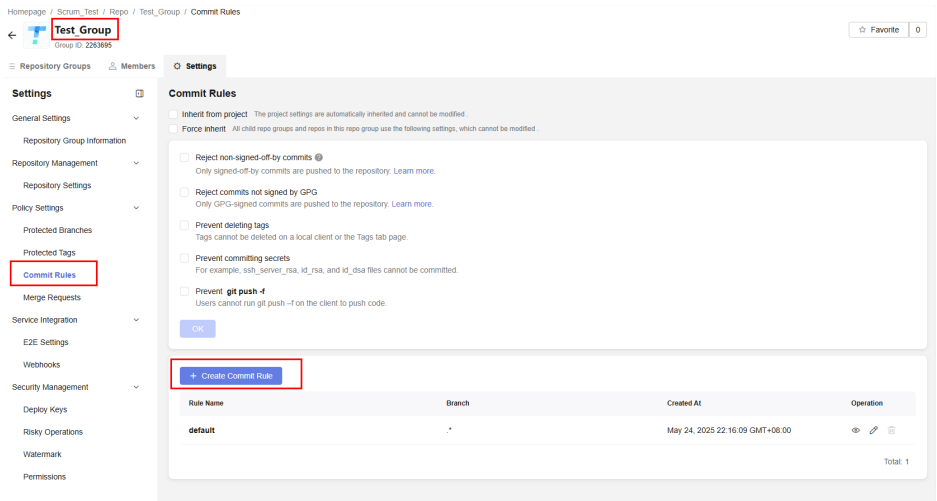
Constraints

Table 10-9 Constraints on configuring commit rules

Item	Description
Permission constraint	Repository group members who have the **setting** permission for repository groups can configure permissions on this page. For details, see **Configuring a Repository Group's Permissions**. Repository group members can only view this page.
Function constraints	• If you select **Inherit from project**, the settings of the project will be used and cannot be changed. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.

Steps

Step 1 Go to the repository group **Test_Group** to be configured.



Step 2 If you do not inherit project settings, set commit rules by referring to **Table 5-2**. The configuration takes effect for all branches in the repository.

Step 3 If you want to configure commit rules for a specified branch, click **Create Commit Rules**, and refer to **Table 5-3** for filling out the parameters. For common regular expression examples, see **Table 5-4**.

----End

10.3.3.4 Configuring Merge Request Rules for a Repository Group

You can configure merge request rules for all repositories in a repository group. Before the configuration, view **Constraints** and configure protected tag rules for the repository groups by referring to **Steps**.

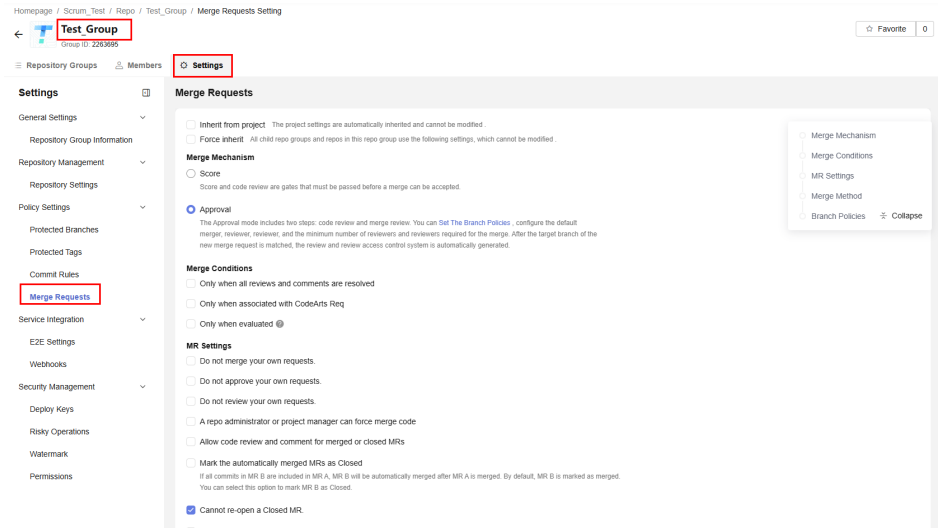
Constraints

Table 10-10 Constraints on setting merge request rules

Item	Description
Permission constraint	**Project manager** or **project administrator** can set project-level webhooks. For details, see **Configuring Project-Level Permissions**.
Function constraints	• If you select **Inherit from project**, the settings of the project will be used and cannot be changed. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.

Steps

Step 1 Go to the repository group **Test_Group** to be configured.



Step 2 If you do not inherit project settings, refer to **step 1** to understand the differences between score and approval mechanisms. Then select a merge mechanism, and configure merge request rules by referring to **Step 2** and **Step 3**.

If your merge mechanism is **Approval** and you want to set the merge policy for specified branched or branches with match rules in the repository, refer to **this section**.

----End

10.3.4 Repository Group Integration with Other Services

10.3.4.1 E2E Settings

Before configuring E2E settings, view **Constraints**. If you have the E2E setting permission, perform the configuration by referring to **Integrated Systems**, **Integration Policies**, and **Automatic ID Rules Extraction**. You can also perform the configuration by referring to **E2E Settings Example**.

Constraints

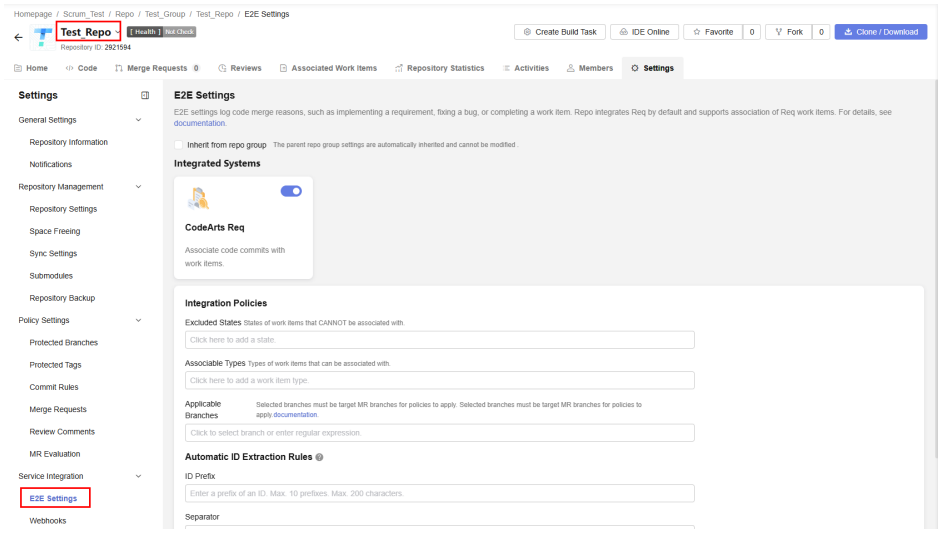
Table 10-11 Constraints on repository groups' E2E settings

Item	Description
Function constraint	The repository groups and repositories of Kanban projects do not support E2E settings. If the project type is **Kanban**, this page cannot be viewed on the repository group settings tab page.

Item	Description
Function constraints	• If you select **Inherit from project**, the settings of the project will be used and cannot be changed. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.
Permission constraint	You must have the set permission on repository groups to configure E2E settings. For details, see **Configuring a Repository Group's Permissions**.

E2E Settings

Step 1 Go to the repository group **Test_Group** to be configured.



Step 2 If you do not inherit project settings, see **E2E Settings**.

----End

10.3.4.2 Webhooks

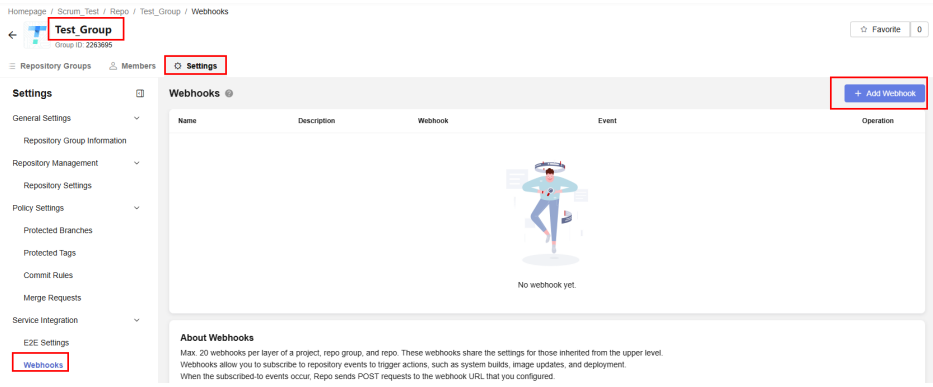
Constraints

Table 10-12 Constraints on configuring webhooks

Item	Description
Permission settings	**Project manager** or **project administrator** can set project-level webhooks. For details, see **Configuring Project-Level Permissions**.
Function constraints	• A maximum of 20 webhooks can be created for a repository. • If you select **Inherit from project**, the settings of the project will be used and cannot be changed. • If **Force inherit** is selected, all child repository groups and repositories in the repository group will use the following settings and cannot be changed.

Procedure

Step 1 Go to the repository group **Test_Group** to be configured and click **Add Webhook**.



Step 2 You can set parameters by referring to **Table 5-12**. For the example results, see **Related Document**.

----End

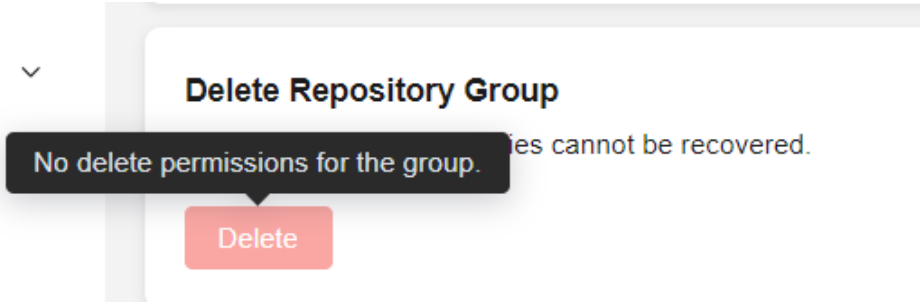
10.3.5 Risky Operations

Constraints

Only the repository group owner has the permission for risky operations. You can contact the repository group owner to perform these operations.

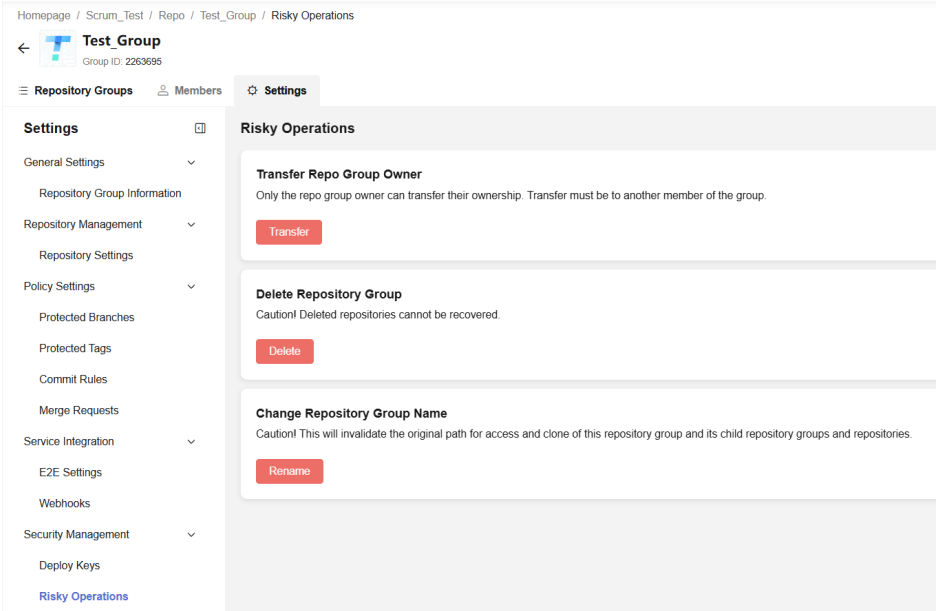
As shown in the following figure, if you do not have the permission, the setting page will not be displayed or the configuration button is unavailable.

Figure 10-9 Display of pages without operation permission



Configuring a Repository Group's Risky Operations

To configure risky operations, choose **Settings** > **Security Management** > **Risky Operations** on the repository group details page.



The following operations are supported:

- **Transfer Repository Group Ownership**: Only the repository group owner can perform this. You can transfer the current repository group to another member in the repository (but not to a viewer).
- **Delete Repository Group**: Only members with the permission to delete a repository group can perform this operation. Enter **DELETE** to delete the repository group. **Once you delete the repository group, all content in the**

repository group will be permanently and irrecoverably deleted. Exercise caution when performing this operation.

- **Change Repository Group Name**: Only the repository owner can change the repository group name. This will invalidate the original path for accessing and cloning this repository group and its child repository groups and repositories. Exercise caution when performing this operation. If CodeArts Build, CodeArts Check, CodeArts Deploy, and CodeArts IDE have been enabled, you need to check and modify related configurations.

11 Viewing Repository Information

11.1 Viewing the Repository List

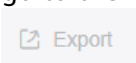
Constraints

Table 11-1 Permission constraint

Item	Description
Permission constraint	Only the tenant space owner or tenant administrator can view this page.

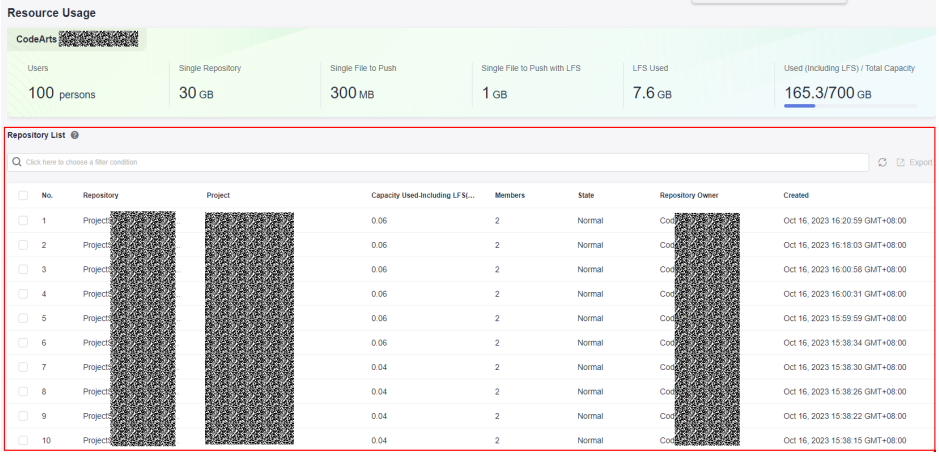
Viewing the Repository List

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. Enter the name of any repository in the search box on the right of **All repositories** and the searched repository will be displayed.
- On the CodeArts homepage, click a project, and click **Code > Repo** in the navigation pane on the left to enter the repo list page. Enter the name of any repository in the search box on the right of **All repositories** and the searched repository will be displayed.
- If you have subscribed to the new CodeArts package or purchased any CodeArts Repo package, go to the CodeArts homepage, click your profile picture, and choose **All Account Settings > Repo > Resource Usage**. In the **Repository List** area, click a project to go to the repo list page, as shown in the **following figure**. Alternatively, you can click the storage usage and the number of repos in the upper left corner of the CodeArts Repo homepage to go to the **Resource Usage** page directly. If you are a tenant member, click 

in the upper right corner to export the selected resource usage.

Figure 11-1 Resource usage



On the top navigation bar, choose **Services** > **Repo**. Choose **Repos > Joined** to view all repositories that you participate in. If you click ☆ in the row where a repository is located, you can favorite the repository. You can choose **Repo > Favorited** to view the repositories that you favorite. To view the repo you created, choose **Repos > Created**.

11.2 Viewing Repository Details

Constraints

Table 11-2 Permission constraint

Item	Description
Permission constraint	If the repository is a private repository, only repository members can view the repository details page. If the repository is a public repository, the repository can be read-only for project members, tenant members, or all visitors.

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

Table 11-3 Description

Page	Function Description
Home	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.

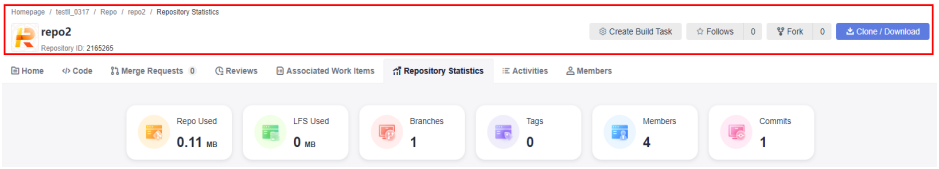
Page	Function Description
Code	• File list: You can create files, directories, and submodules, upload files, modify files and blame, and view commit history. • Submit: You can view commit records and repository network diagrams. • Branch: Branches can be managed on the console. • Tag: Tags can be managed 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: • **All**, **Groups**, **Pending**, and **Add Member** are located on the **Members** tab page of repository details. – **Username**, **User Source**, **Status**, **Project Member Role**, **Repository Member Role**, and **Operation** 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**, **Status**, **Project Member Role**, **Repository Member 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. If you do not have the build permission, only the **Create Build Task** button is displayed on the repository page. If you have the build permission and have created a Build task, the status above the repository page is displayed based on the build task execution status, including running, building, failed, and successful.
- **Favorite**: Click to favorite the repository. The favorited 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 figure shows 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 part 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.



- Code check status display: If you have the code check permission, the status is displayed next to the repository name. If no, no code check status is displayed next to the repository name.

11.3 Viewing Repository Homepage

The **Home** tab page displays the basic information about the repository, as shown in the following figure. You can view the repository information according to the table below.

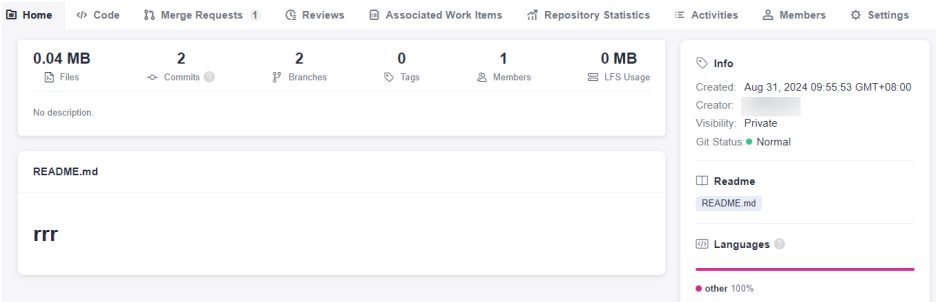
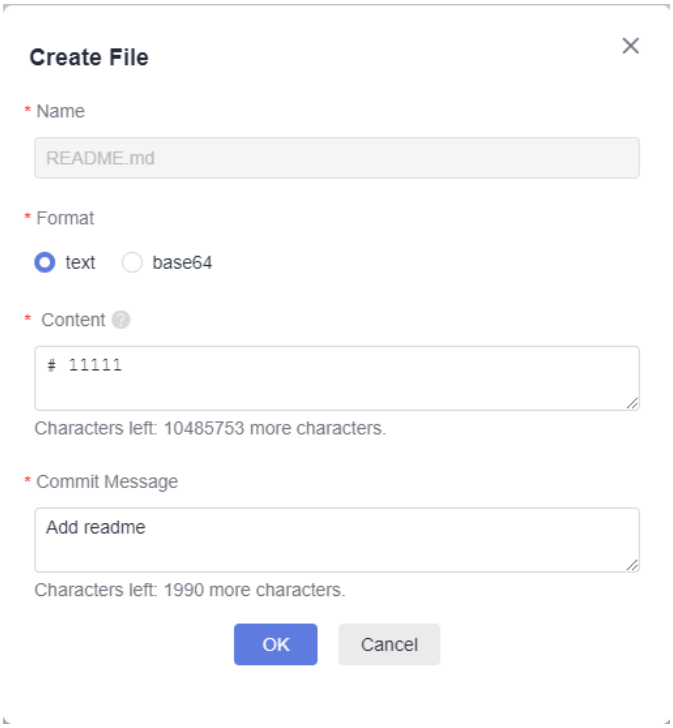


Table 11-4 Parameter description

Parameter	Description
Files	Capacity of the current repository. The preceding figure shows that 0.04 MB capacity has been used by the current repo. • The repository capacity includes the LFS usage. The capacity of a single repository cannot exceed the limit of a single repository in your purchased CodeArts package. To view the capacity, go to the repository list page of CodeArts Repo, click the alias in the upper right corner, and choose **All Account Settings > Repo > Resource Usage**. If the capacity exceeds the limit, the repository cannot be used or 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	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.
Branches	Displays the number of branches in the current repository. You can click the branch icon or the number above it to go to the **Branches** page **Code** tab page and manage branches.
Tags	Displays the number of tags in the current repository. You can click the icon to go to **Tags** page under the **Code** tab page and manage tags.
Members	Displays the number of members in the current repository. You can click the icon to go to the **Members** tab page and manage members.
LFS Usage	Collects statistics on the LFS usage of the current repository.
Repository description	The description entered during repository creation.

Parameter	Description
README.md	You can preview README files. If no README file exists in the repository, click **Create Readme** to create a README file. **Name**: The default file name is **README.md**. **Format**: The options are as follows: • **text**: indicates text data or a text string. • **base64**: Base64 is a method of representing binary data based on 64 printable characters. **Content**: The value can be customized. • If the format is **text**, enter common text. • If the format is **base64**, enter Base64-encoded content that can pass the encoding verification. **Commit Message**: Enter the commit information about the file or folder, which can be customized. 
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.

12 Cloning or Downloading a Repository to a Local PC

12.1 Differences Between Cloning and Downloading a Repository

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

1. Clone a repo to a local PC.

   Using the SSH key or HTTPS protocol to clone a repo means the process of copying 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.

NOTE

Currently, CodeArts Repo supports cloning one repository at a time. If you want to clone multiple repositories to the local host at a time, you can use Shell or batch processing commands.

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

Using Git Bash to Clone a Repo to a Local Host

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

Step 1 Access CodeArts Repo homepage.

Step 2 Go to the homepage of the code repository 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 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 12-1 Successful cloning of the repository using the SSH key

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, no SSH key for accessing Repo has been configured. Configure an SSH key first. For details, see **Configuring an SSH Key**.

----End

Using TortoiseGit 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 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

12.3 Using HTTPS to Clone a 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 homepage 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 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:

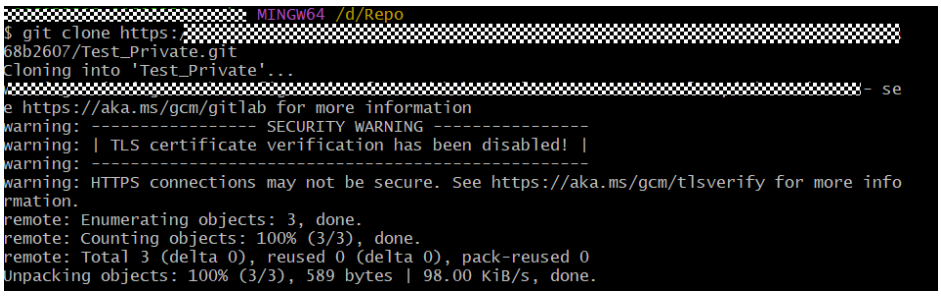
```
git clone code repo HTTPS link
```

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 CodeArts Repo's code repo list page, click the alias 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 the code repo fails to be cloned, rectify the fault based on the **description**.

Figure 12-2 Successful cloning of the repository using HTTPS



----End

Related Document

- 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/'': Recy 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 **fetal: Authentication failed for 'https:/xxx.git/'**, indicating that your password is incorrect. You can log in to the repository list page, click the alias 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 when performing **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
```

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

Constraints

- 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 need to have the download permission to download the repository source code on the page.

Downloading a Code Package

CodeArts Repo supports both cloning code repositories and packaging repository code for local download. The downloadable package contains the content of a specified branch.

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

13 Uploading Code Files to CodeArts Repo

13.1 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 MR**. You can view the file differences and commit records of the two branches in the lower part of the page.

NOTICE

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

13.2 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 MR** to a file named **hello_MR.txt**.


```
echo 'hello MR' > hello_MR.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. **Note: The newline character of the MAC file is /r, which cannot be identified by Git. You are advised to convert the newline character to the Linux or Windows format on the local host and then submit the file. Otherwise, the number of lines in the MAC file on the comparison page is different from that in the MR.**

```
git add .
```

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

```
git commit -m 'hello MR'
```

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

Related Document

- 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 refspec 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. For details, see **Error "Merge branch 'master' of https://xx.com. Please Enter a commit" Is Reported When Pulling Code Using the Git Pull Command** in
- 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' does not appear to be a git repository..." is displayed when you perform step 6, the cause is that the repository name origin does not exist remotely. For details, see **Error "'origin' does not appear to be a git repository..." Is Reported When the Git Push Command Is Executed** in .
- 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 "failed to push some refs to '....git'" error is reported when you try to submit a merge request, see **Resolving Merge Request Conflicts**.
- 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.

13.3 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.

- For the first push:

a. Create a repository on the local computer, that is, the local repository.
b. **Commit** the update to the local repository.
c. Pull the code from the server to the local repository, merge the code, and push the repository to the server. The remote commit is complete.

- If it is not the first push:
  a. Commit the modified code to the local repository.
  b. Pull the code from the server to the local repository, merge the code, and push the repository to the server.

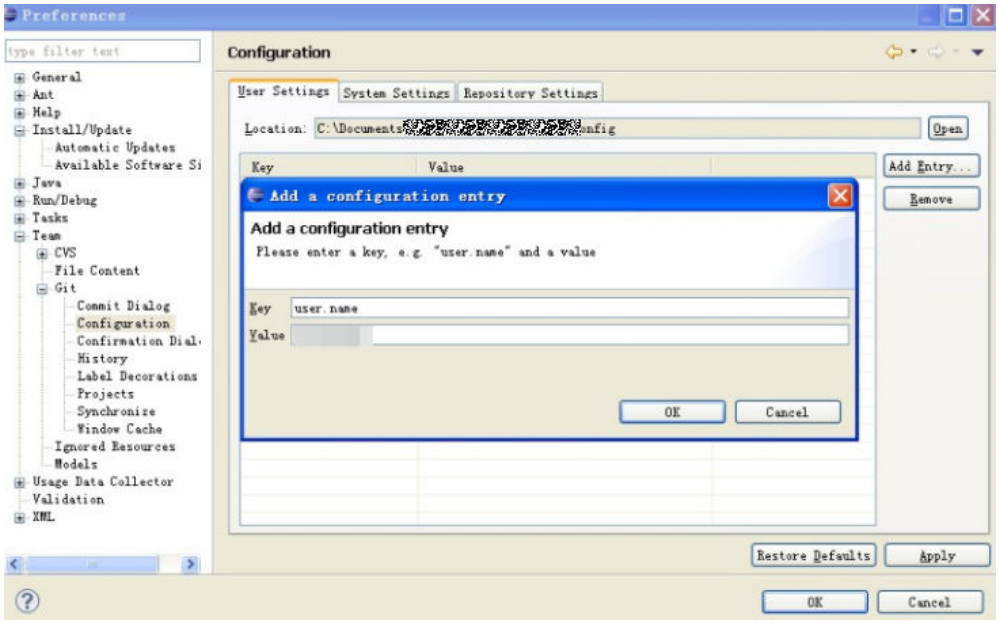
Step 1: Installing EGit on Eclipse

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.

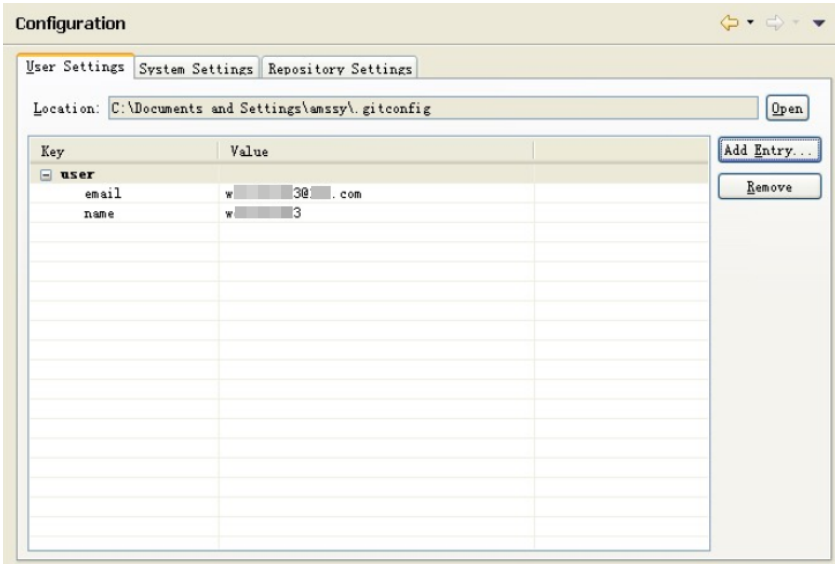
Step 2: Configuring EGit on Eclipse

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



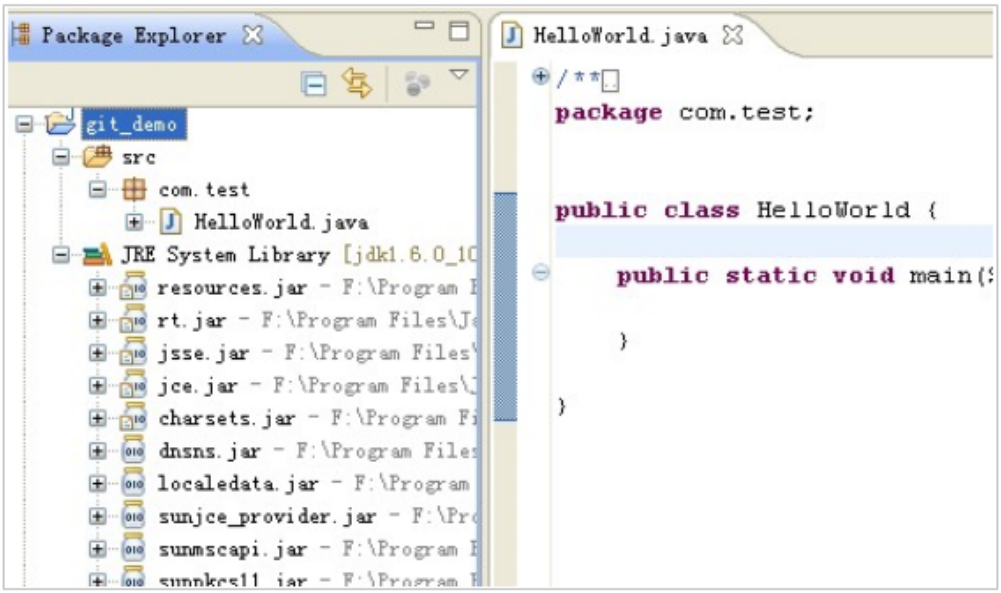
2. Click **OK**.

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

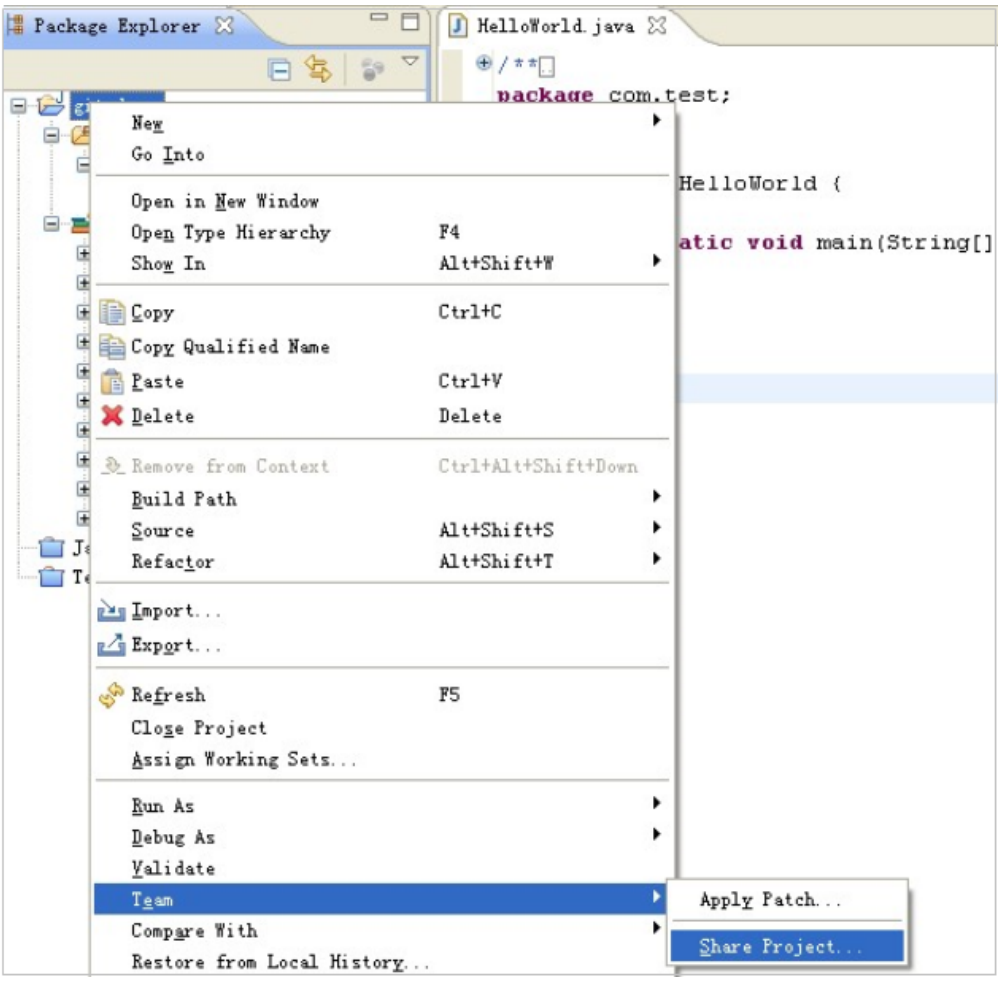


Step 3: Creating a Project and Committing Code to the Local Git Repository

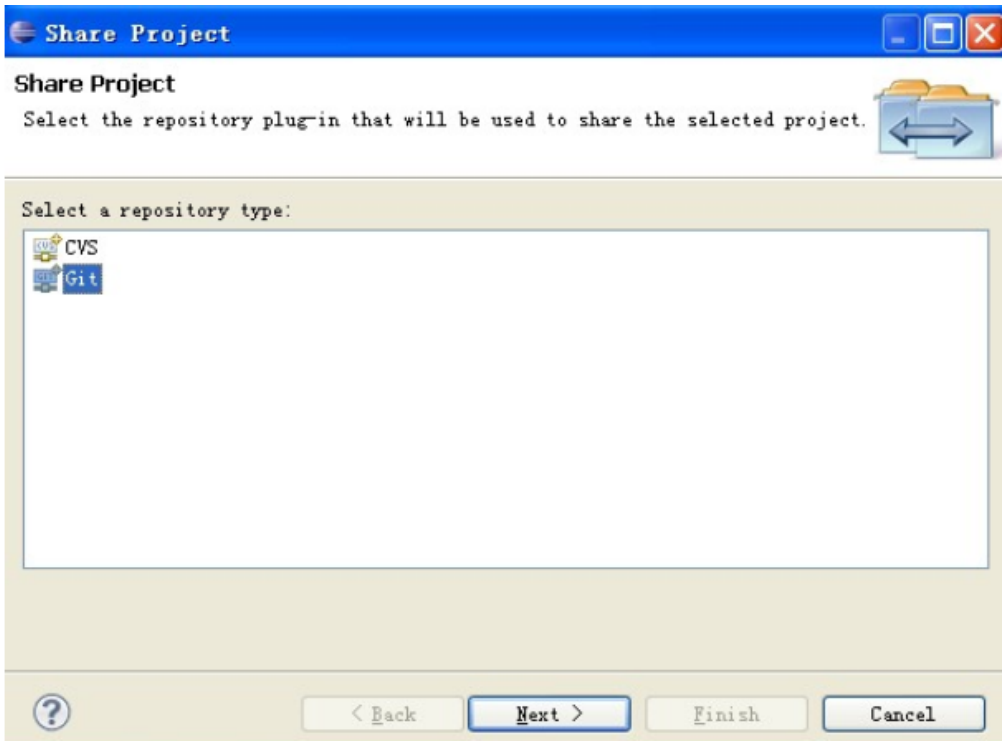
1. Create the **git_demo** project and the **HelloWorld.java** class.



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



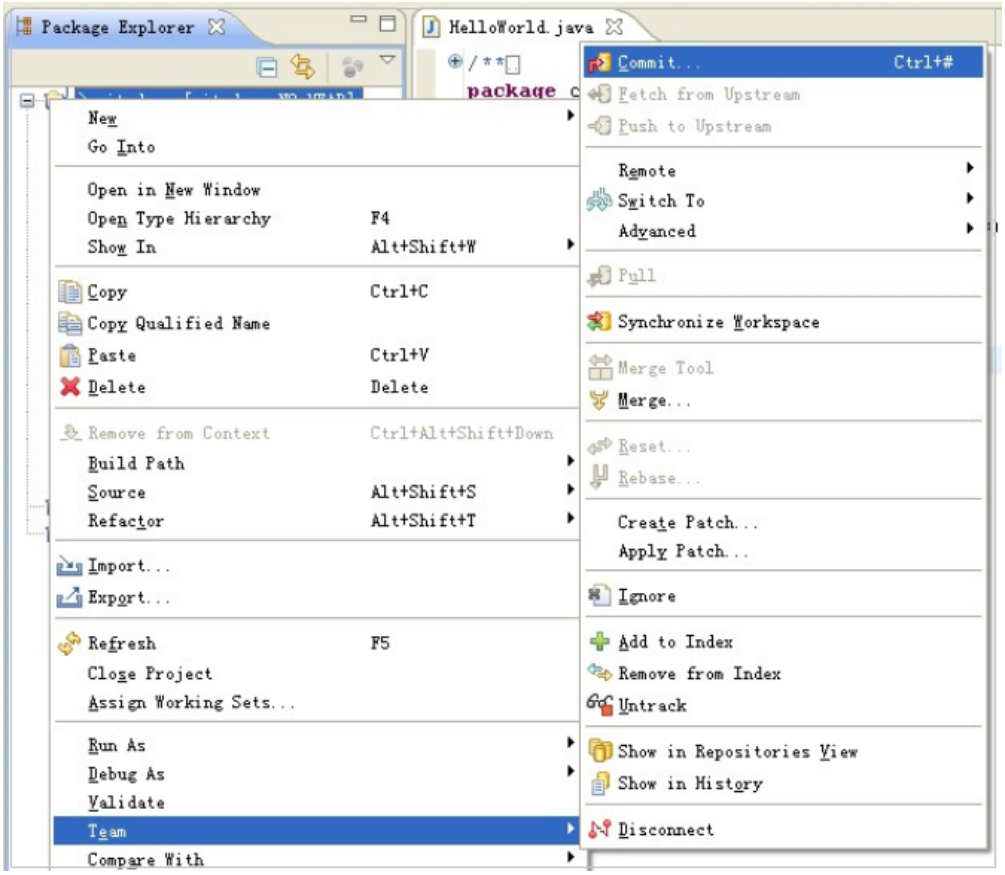
3. In the **Share Project** window displayed, select **Git**.



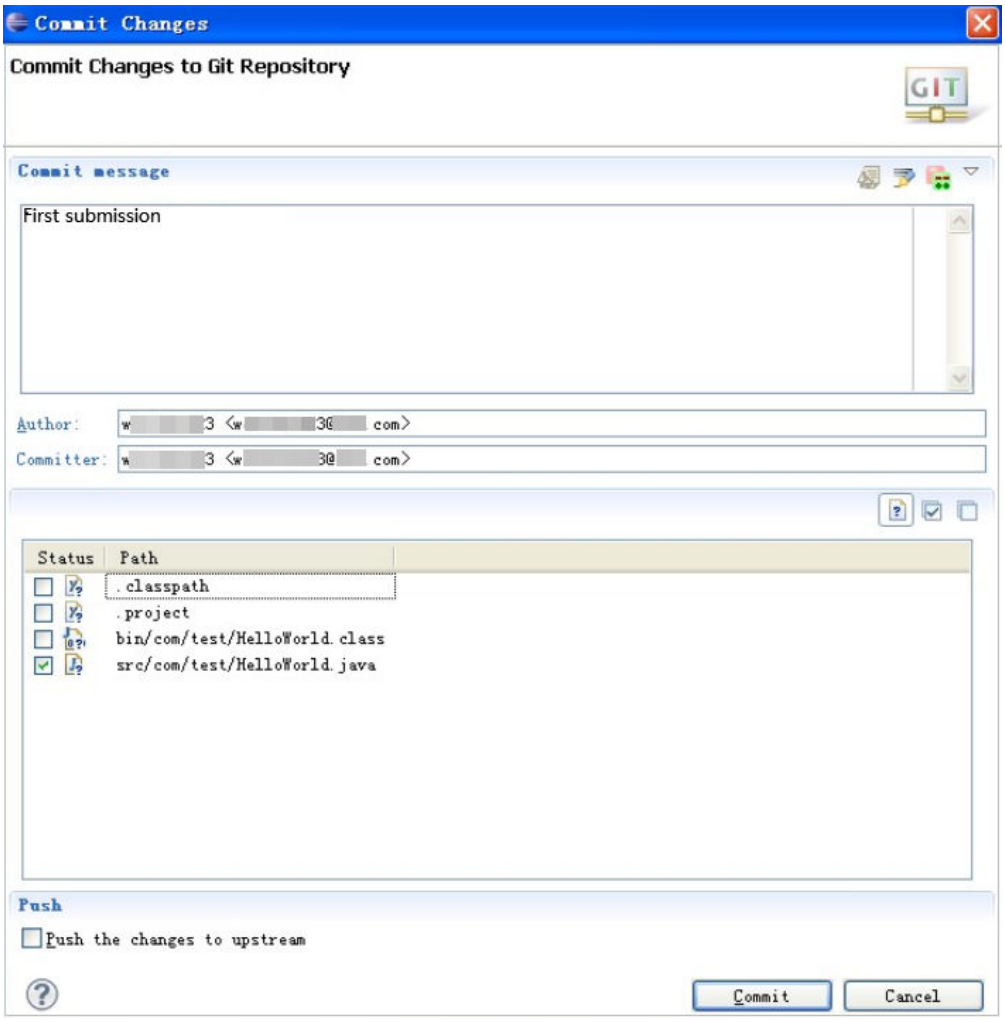
4. Click **Next >**. In the **Configure Git Repository** dialog box, select **Use or create repository in parent folder of project** and click **Create Repository**.
5. Click **Create Repository** to create a Git repository.

   The directory is in the **untracked** status, indicated by a question mark (**?**).

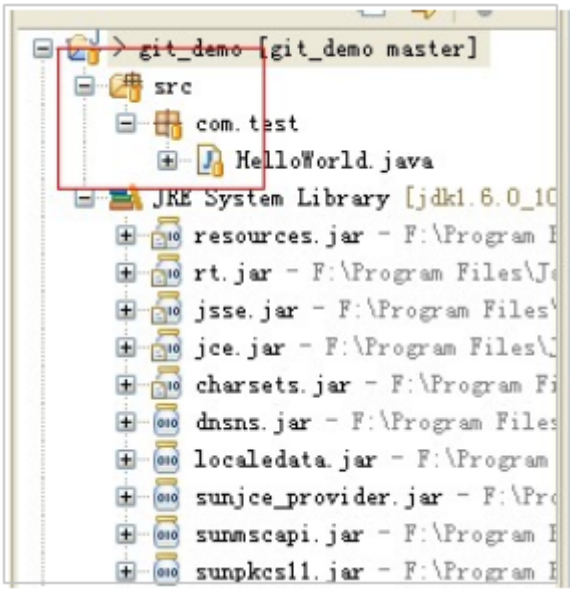
   Choose **Team** > **Commit...** to commit code to the local repository.



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

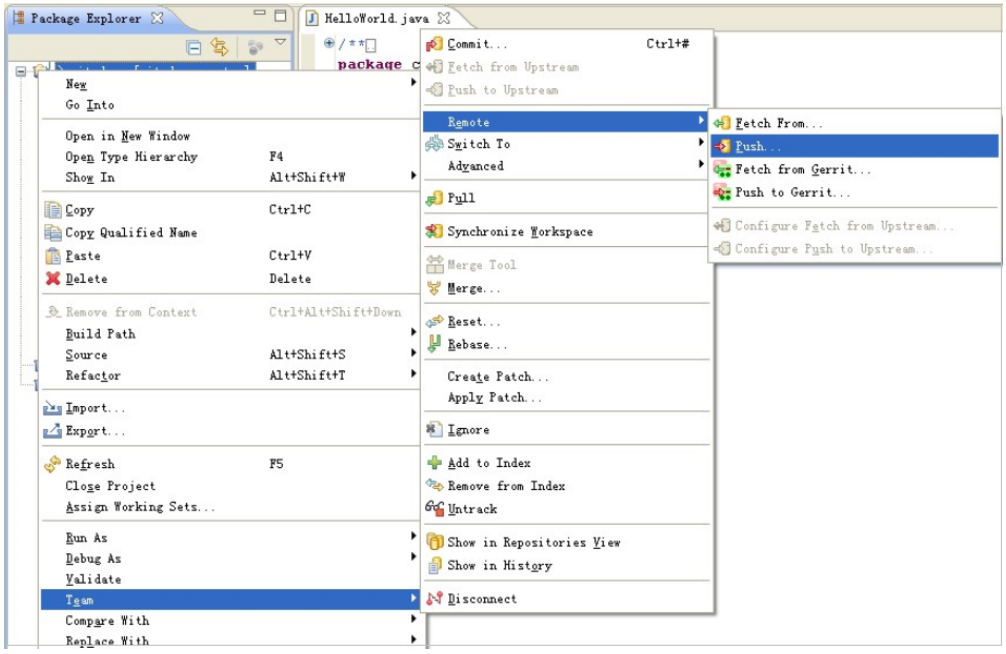


7. Click **Commit** to commit the code to the local repository.

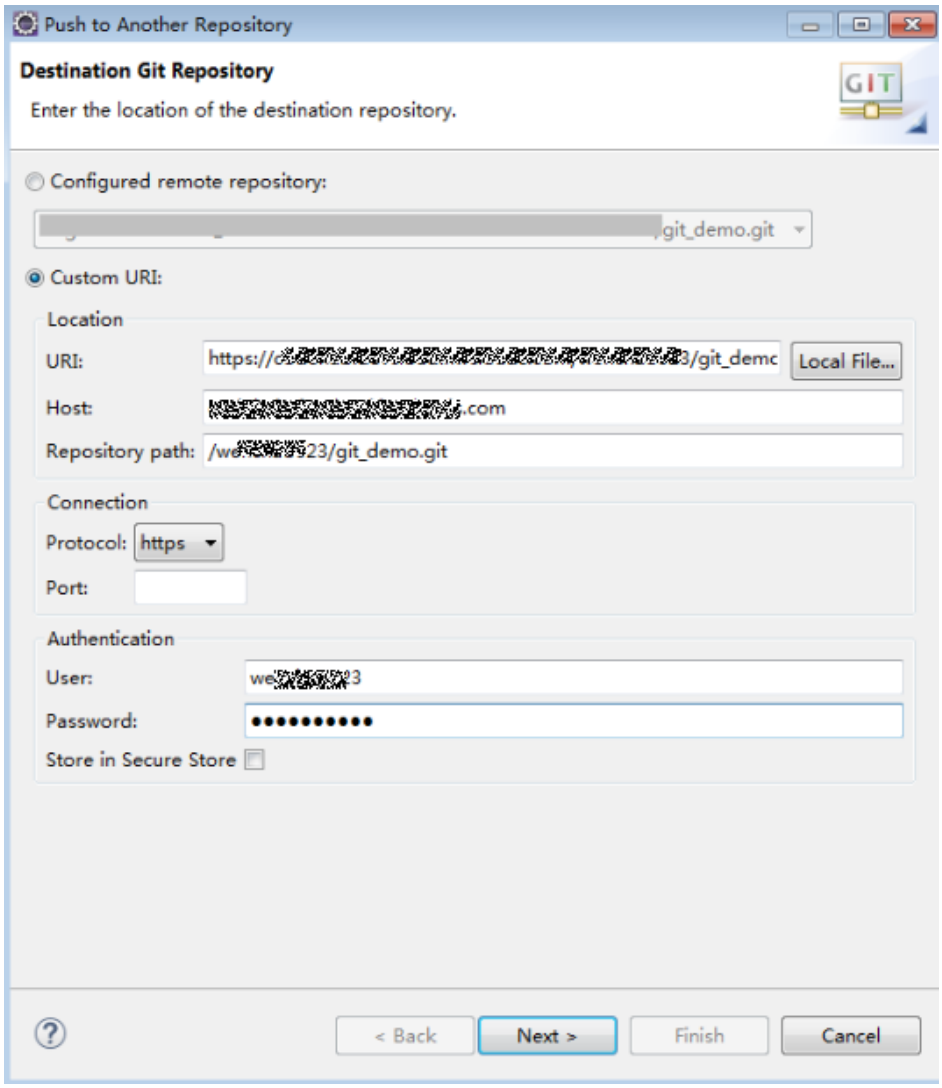


Step 4: Committing Code in the Local Repository to the Remote Git Repository

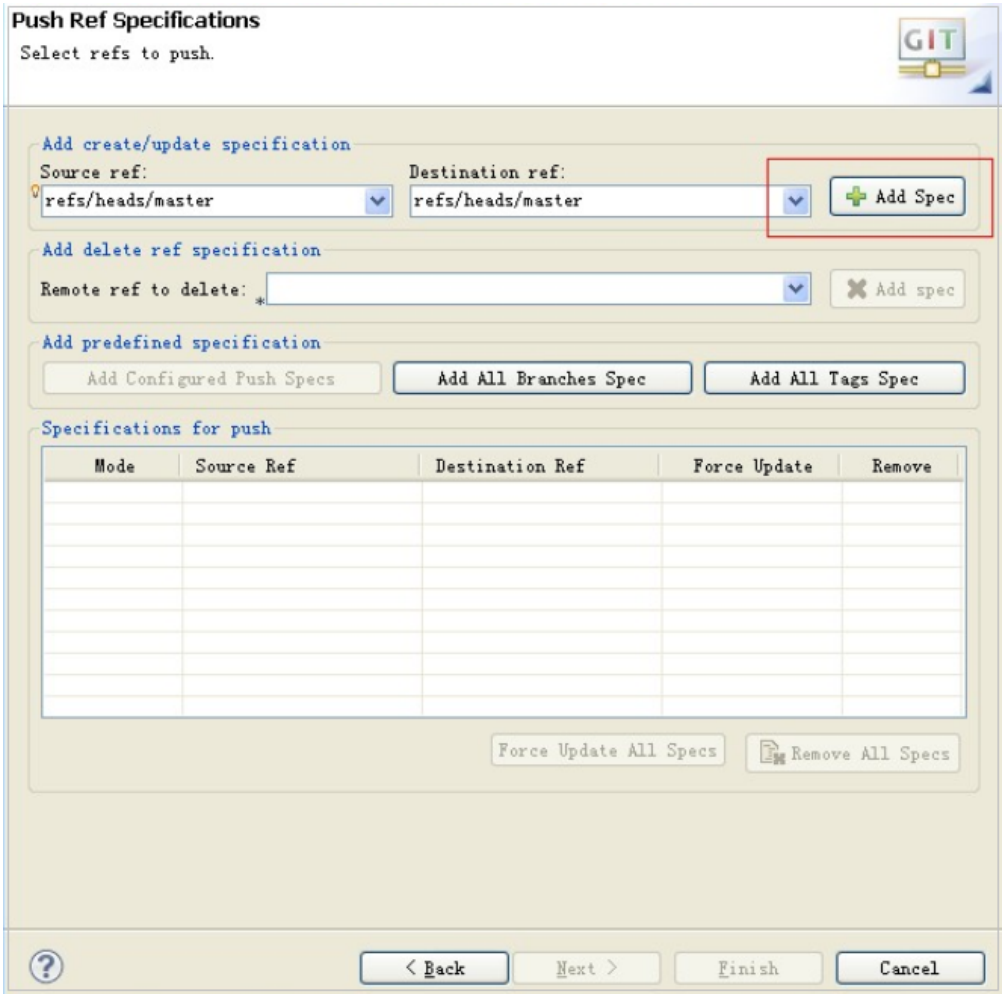
1. Create a repositor in CodeArts Repo. For details, see **Overview**.
   Go to the repository details page and copy the repository URL.
2. Choose **Team** > **Remote** > **Push...** to push the code to the remote repository.



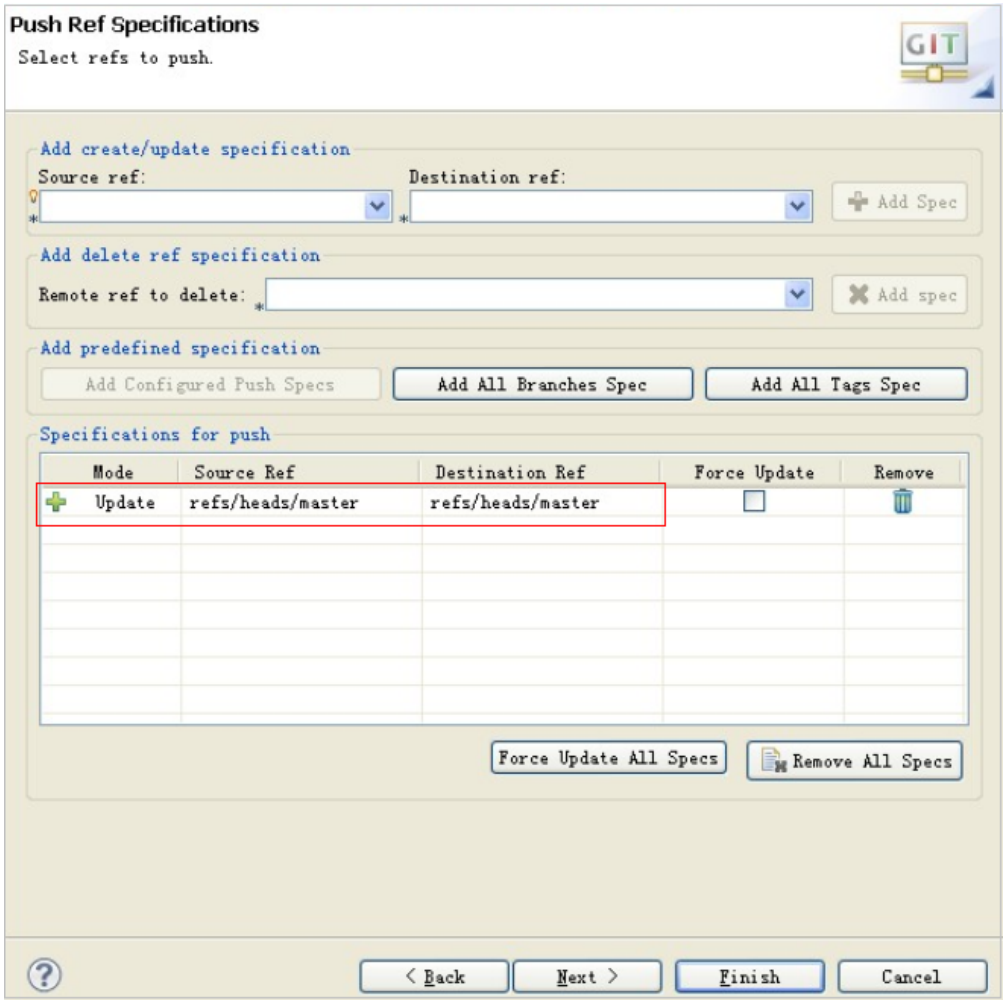
3. In the **Push to Another Repository** dialog box, set the parameters.



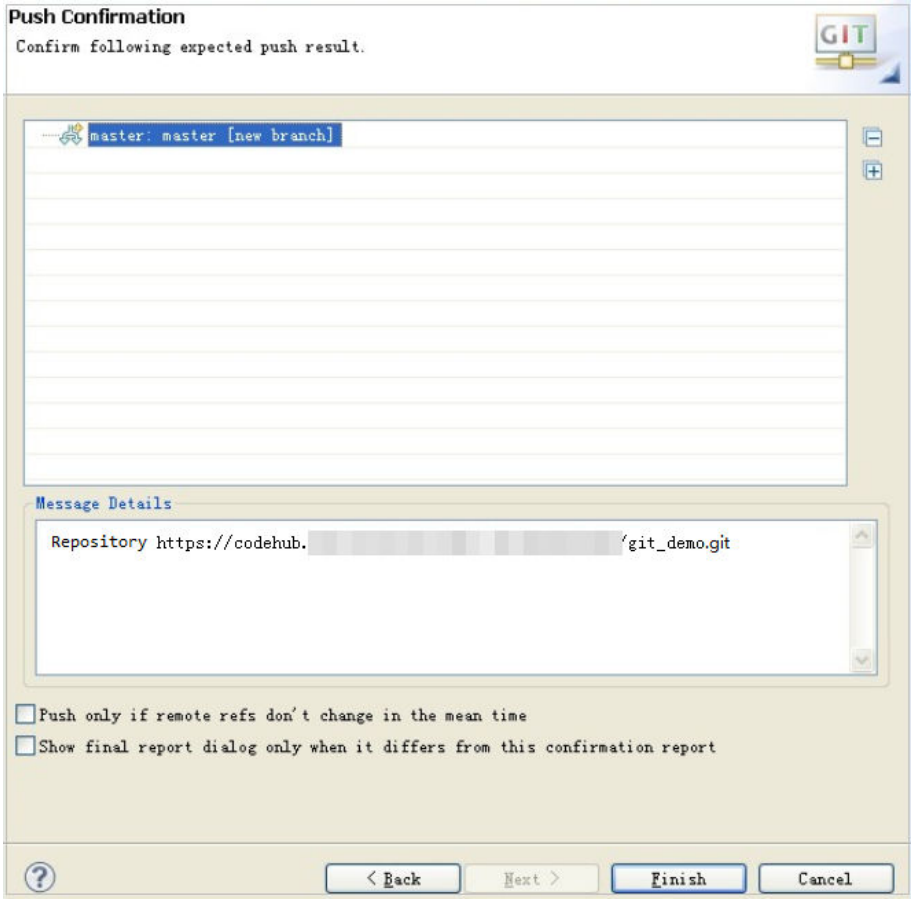
4. Click **Next**. The **Push Ref Specifications** dialog box is displayed.



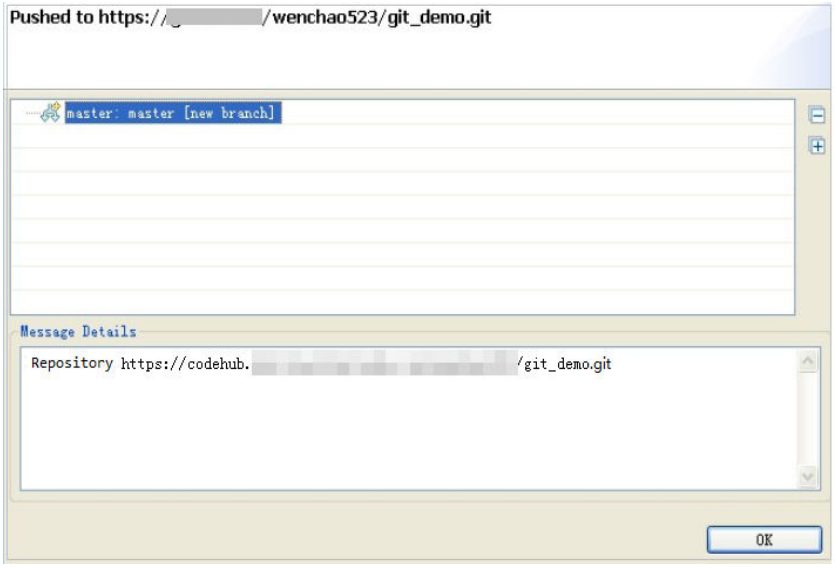
5. Click **Add Spec**.



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**. 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. On the lower part of the **Merge Requests** page, you can view the file difference of the two branches and the commit records of the branch to be merged.

13.4 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. It 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, 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:

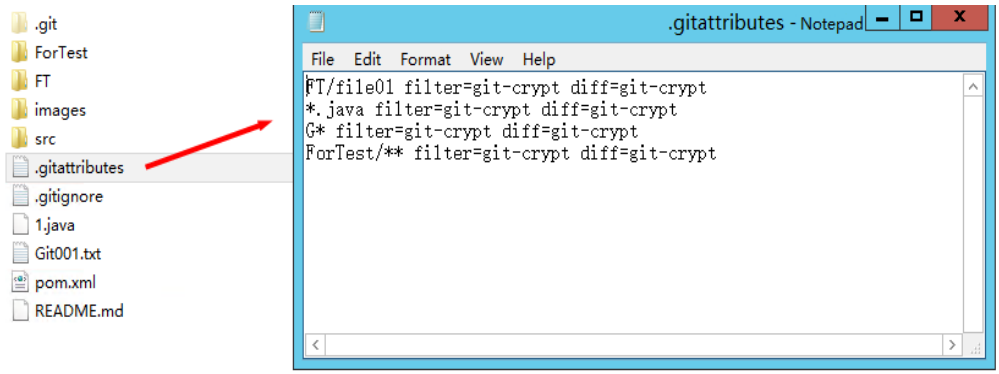
1. Create a file named **.gitattributes** in the root directory of the repository.

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 whose names start with G are encrypted.
ForTest/** filter=git-crypt diff=git-crypt  # Files in the ForTest folder are encrypted.
```



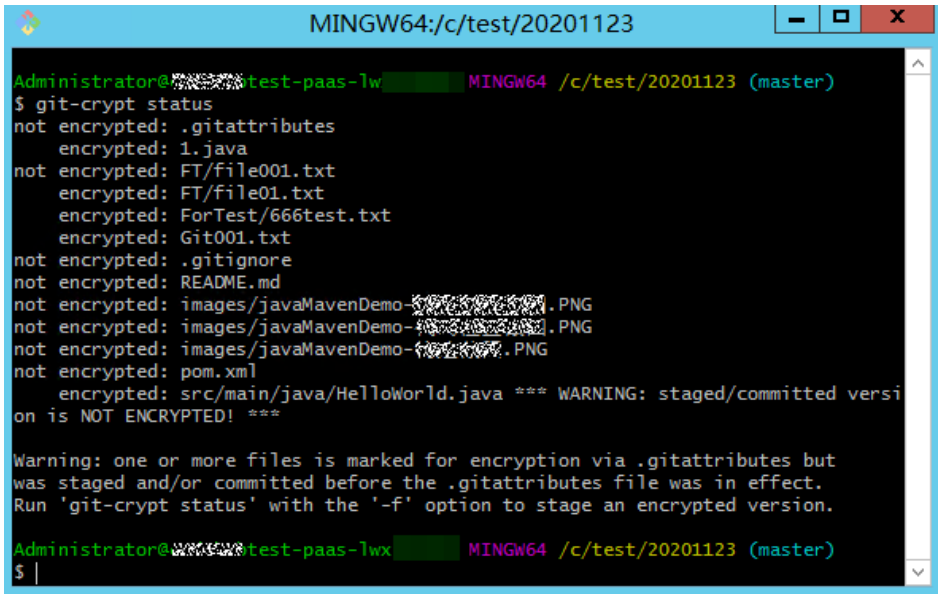
NOTE

- 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 these files even if you download them to the local computer.

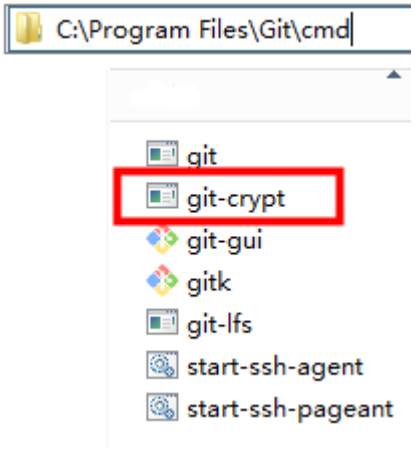
NOTE

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.

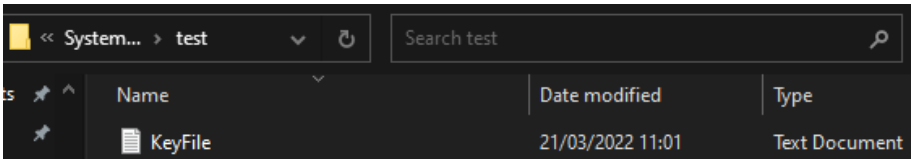
The **-f** (enforce) option is risky in getting teams to work together, so use it with caution.

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.
```

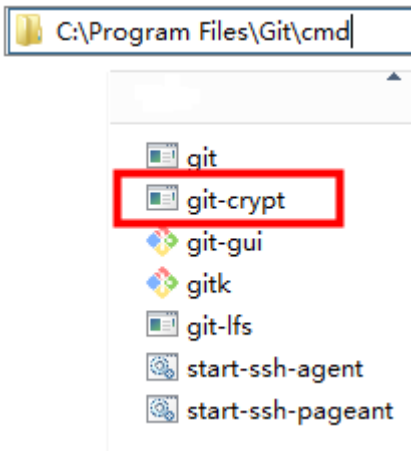
----End

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 *GnuPG*
	download sig	Simple installer for the current *GnuPG*
	download sig	Simple installer for *GnuPG 1.4*
OS X	**Mac GPG**	Installer from the gpgtools project
	GnuPG for OS X	Installer for *GnuPG*
Debian	**Debian site**	GnuPG is part of Debian
RPM	**rpmfind**	RPM packages for different OS
Android	**Guardian project**	Provides a GnuPG framework
VMS	**antinode.info**	A port of GnuPG 1.4 to OpenVMS
RISC OS	**home page**	A port of GnuPG to RISC OS

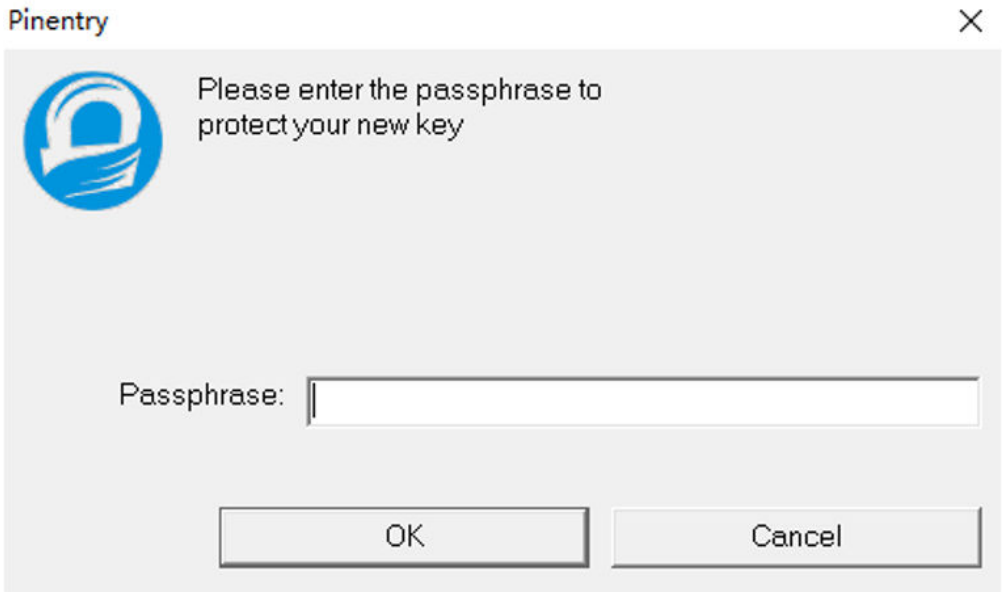
Double-click to start the installation. Click **Next** to complete the installation.

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- 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.

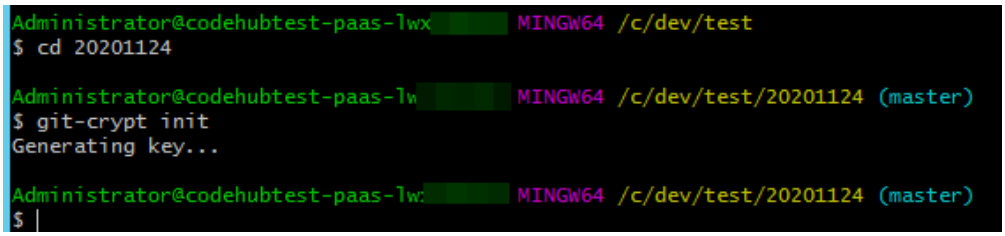
4. If the following information is displayed, the GPG key pair is generated successfully.

```
public and secret key created and signed.

pub   rsa3072 2020-11-24 [SC] [expires: 2022-11-24]
      0DD                                    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:

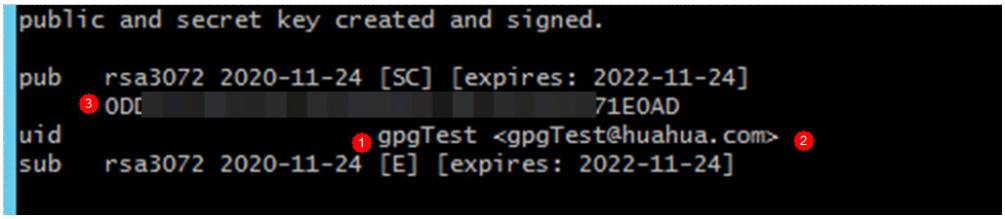
```
git-crypt init
```

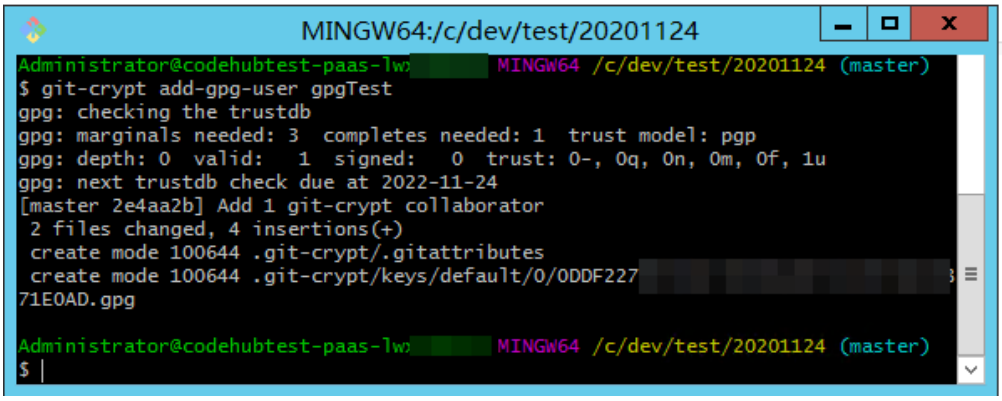
2. Run the following command to add a copy of the key to your repository. The copy has been encrypted using your public GPG key.

```
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.



After the command is executed, a message is displayed, indicating that the **.git-crypt** folder and two files in it are created.



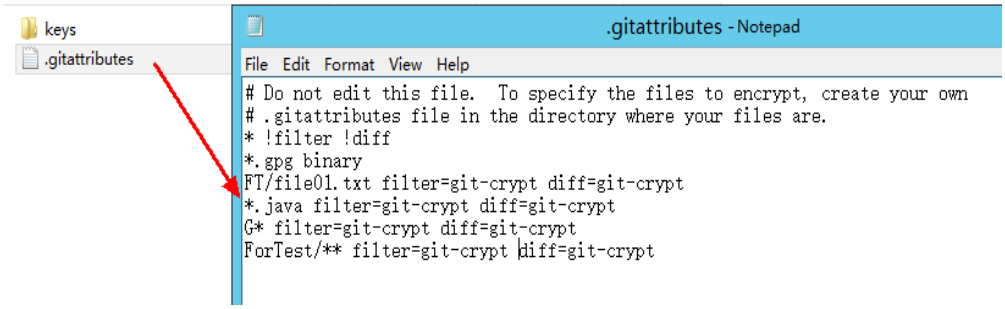
Step 6 Configure the encryption scope for the repository.

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 whose names start with G are encrypted.
ForTest/** filter=git-crypt diff=git-crypt  # Files in the ForTest folder are encrypted.
```

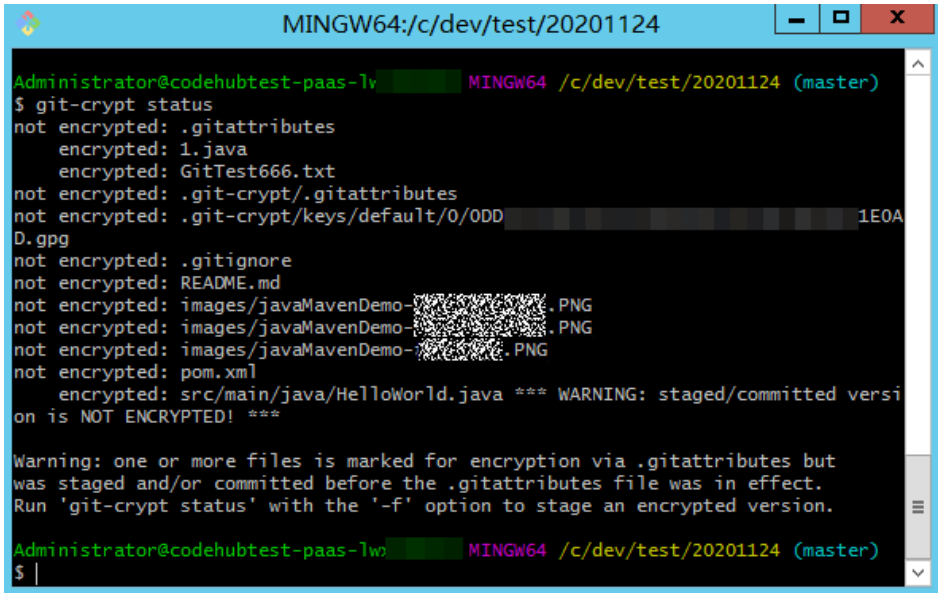



3. Copy the **.gitattributes** file to the root directory of the repository.

Step 7 Encrypt the file.

Open Git Bash in the root directory of the repository and run the following command to encrypt the file. The encryption status of the file is displayed.

```
git-crypt status
```



After the encryption, you can still open and edit the encrypted files in plaintext in your local repository because your local repository has a key.

You can run the **add**, **commit**, and **push** commands to push the repository to CodeArts Repo. In this case, the encrypted files are pushed together.

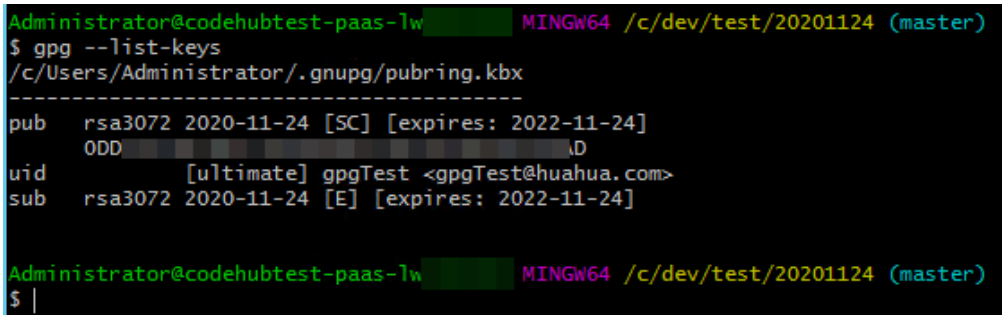
Encrypted files are stored in CodeArts Repo as encrypted binary files and cannot be viewed directly. If you do not have a key, you cannot decrypt it even if you download it to the local computer.

NOTE

git-crypt status encrypts only the files to be committed this time. It does not encrypt the historical files that are not modified this time. Git displays a message for the unencrypted files involved in this setting (see **Warning** in the preceding figure). If you want to encrypt all files of a specified type in the repository, run the **git-crypt status -f** command.

In team cooperation, **-f** (forcible execution) has certain risks and may cause the members' work output to remain unchanged. Exercise caution when using **-f**.

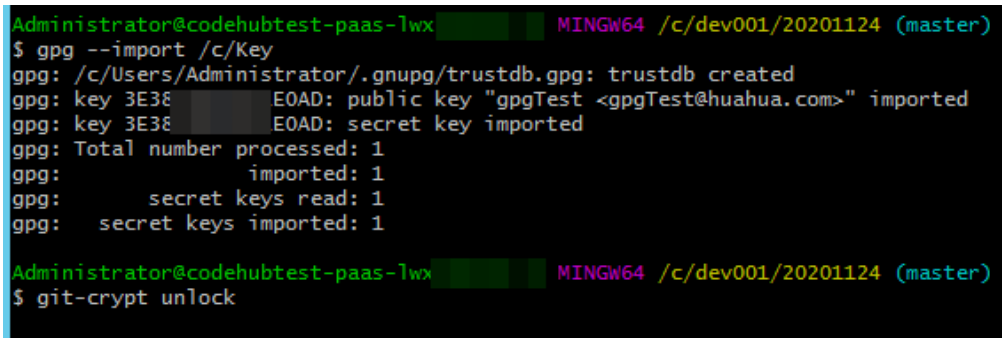
Step 8 Export the key.

1. Lists the currently visible keys. You can view the name, email address, and fingerprint of each key.
   ```
   GPG --list-keys
   ```
   
   ```
   Administrator@codehubtest-paas-lw      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]
         0DD                                      AD
   uid           [ultimate] gpgTest <gpgTest@huahua.com>
   sub   rsa3072 2020-11-24 [E] [expires: 2022-11-24]


   Administrator@codehubtest-paas-lw      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-lwx      MINGW64 /c/dev001/20201124 (master)
   $ gpg --import /c/Key
   gpg: /c/Users/Administrator/.gnupg/trustdb.gpg: trustdb created
   gpg: key 3E38        EOAD: public key "gpgTest <gpgTest@huahua.com>" imported
   gpg: key 3E38        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-lwx      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 your team needs to set different confidential levels for encrypted files, the **GPG encryption** can be used. This encryption mode allows users 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's 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 GPG using the brew package manager.

  ```
  brew install git-crypt
  ```

- Install GPG on macOS.

  Run the following command to install GPG using the brew package manager.

  ```
  brew install GPG
  ```

13.5 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 the repository file list.

Viewing the Commit History on the Repo Activities Page

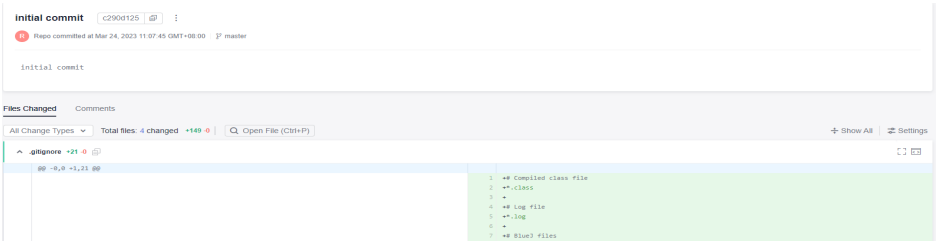
Go to the homepage of the code repository to be viewed and click **Activities** to view all activities of the repository. To view the activities within a specified period, select a period on the right. To view the activities of a repo member, select the repo member from the drop-down list box on the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

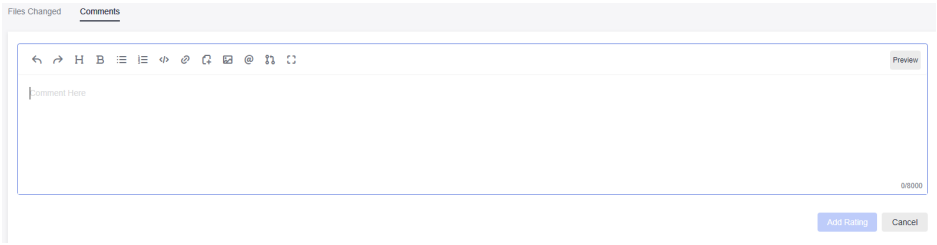
NOTE

- 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.

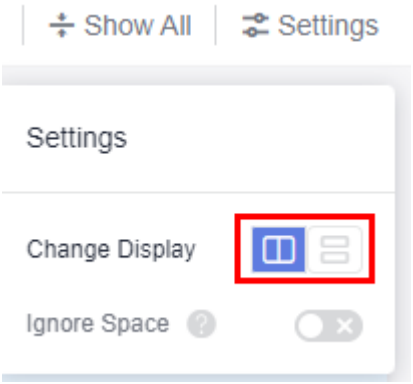
You can view the commit history on the **History** tab page of the **Files** or **Activities** tab. Click a commit record to view the committer, commit number, parent node, number of comments, and code change comparisons.



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.



14 Collaborating on a Workflow

14.1 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**

14.2 Working on a Centralized Workflow

Centralized workflows are suitable for small teams that have just transformed from SVN to Git. Centralized workflows revolve around a central repo. Developers clone the repositories from the central repo and push the code back to the central repo upon completion.

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** and **Eclipse**.

Step 5 **Create a merge request**.

Step 6 **Resolve review comments**.

Step 7 Committers **merge the MR**.

----End

14.3 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.

Working on a Function Branch 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** and **Eclipse**.

Step 4 **Create a merge request**.

Step 5 **Resolve review comments**.

Step 6 Committers **merge the MR**.

----End

15 Committing Code to CodeArts Repo and Managing a Merge Request

15.1 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.

15.2 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.

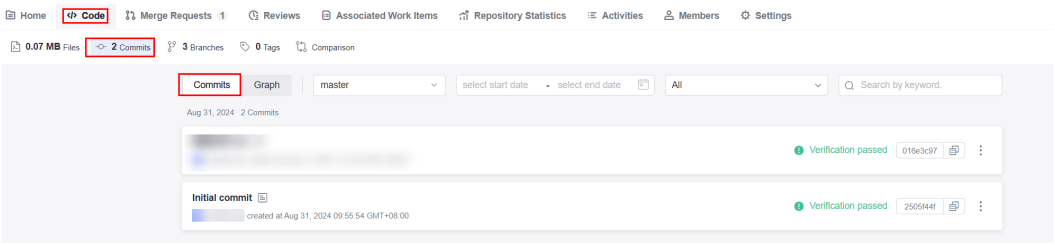
To better understand this function, perform the following operations:

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**.

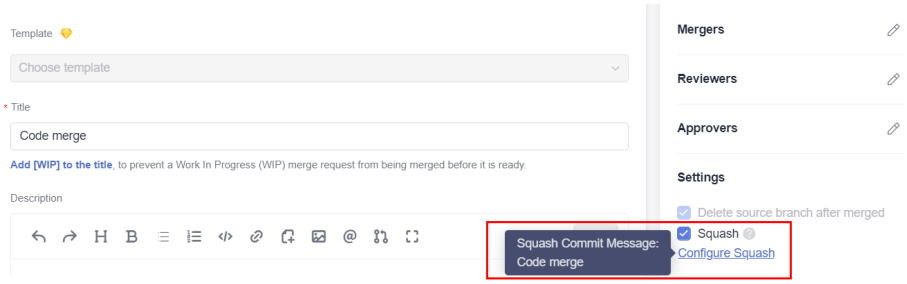
Step 3 Check the effect before Squash is enabled. Click the **Dev** branch and choose **Code > Commits > Commits** to view the commit information.



Step 4 Create and merge a request.

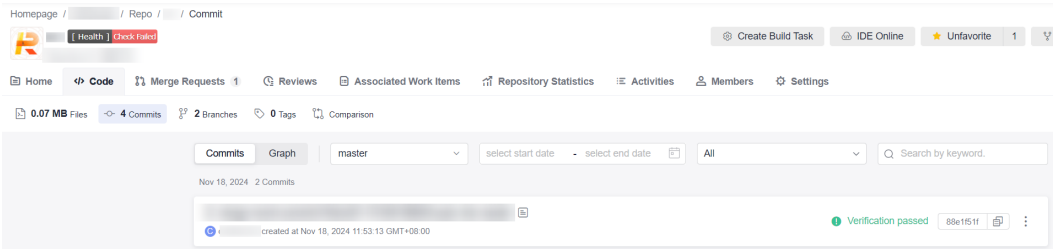
1. Set the source branch to **Dev** and target branch to **master**. Create a merge request.

   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.

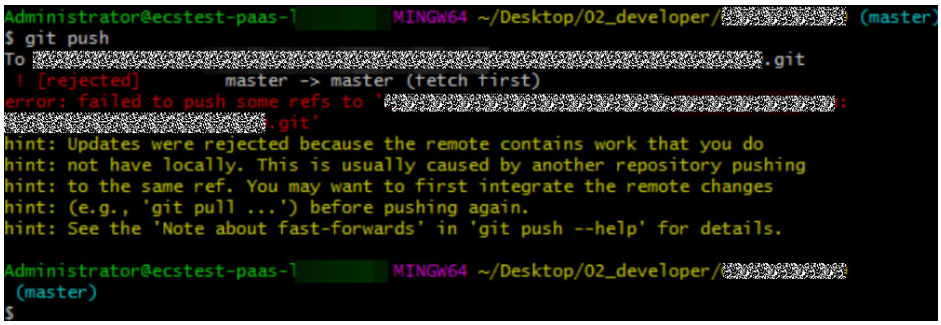
Step 5 Check the effect after **Squash** is enabled. After the request is successfully merged as shown in the following figure, click the **Code**, **Commits**, and **Commits** tabs, select the **master** branch. Compared with **Step 4**, the committed content has been merged.



----End

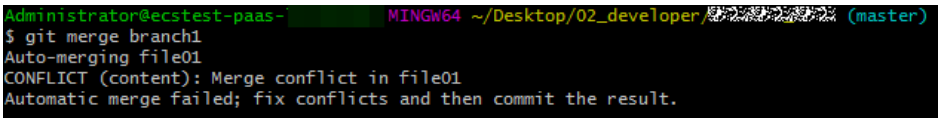
15.3 Resolving Code Conflicts in an MR

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.



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).

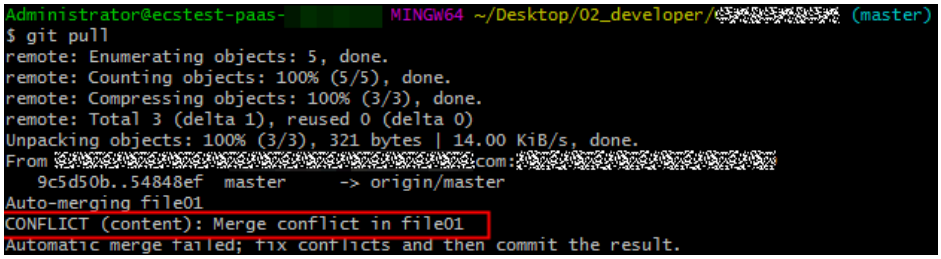
- Max. 50 conflict files can be solved, and the size of a single conflict file cannot exceed 200 KB.

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.



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.

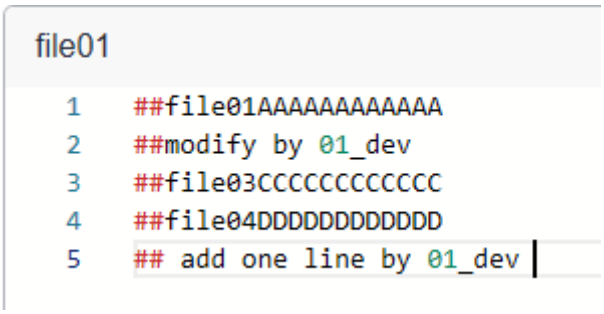
A company uses CodeArts Repo and Git to manage a project. A function (the **file01** file is modified) of the project is jointly developed by developer 1 (01_dev) and developer 2 (02_dev). The two developers encounter the following situation.

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

   file01

   ```
   1  ##file01AAAAAAAAAAA
   2  ##file02BBBBBBBBBBB
   3  ##file03CCCCCCCCCCC
   4  ##file04DDDDDDDDDDD
   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.



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

```
##file01AAAAAAAAAAAA
## modify by 02_dev
##file03CCCCCCCCCCCC
##file04DDDDDDDDDDDD
## add by 02_dev
```

4. 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**.
5. 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.

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.

6. 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.

```
##file01AAAAAAAAAAAA
## modify by 02_dev
##modify by 01_dev
##file03CCCCCCCCCCCC
##file04DDDDDDDDDDD
## add by 02_dev
## add one line by 01_dev
```

7. 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  ##file01AAAAAAAAAAAA
2  ## modify by 02_dev
3  ##modify by 01_dev
4  ##file03CCCCCCCCCCCC
5  ##file04DDDDDDDDDDD
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.

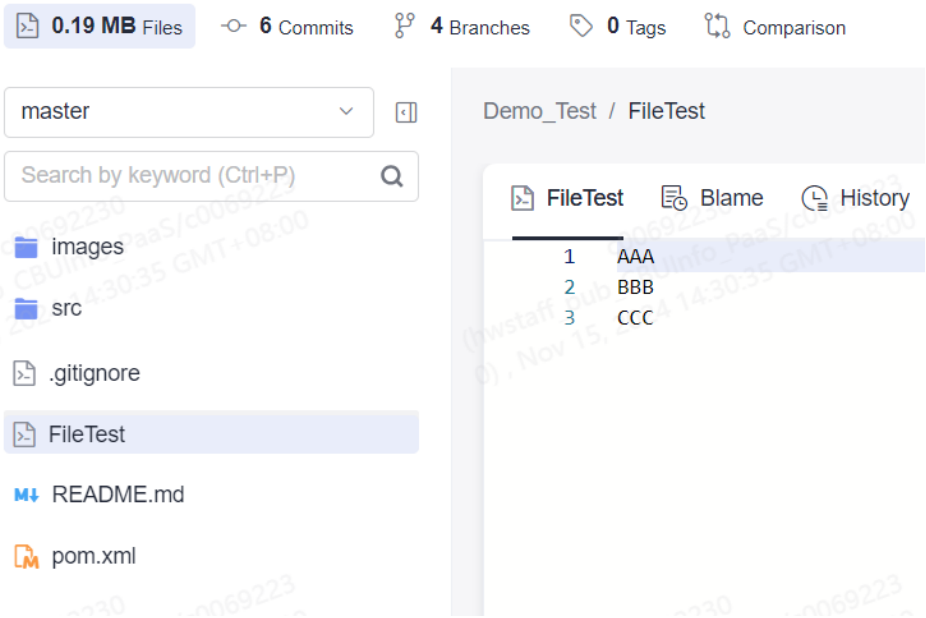
Resolving a Merge Conflict

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.

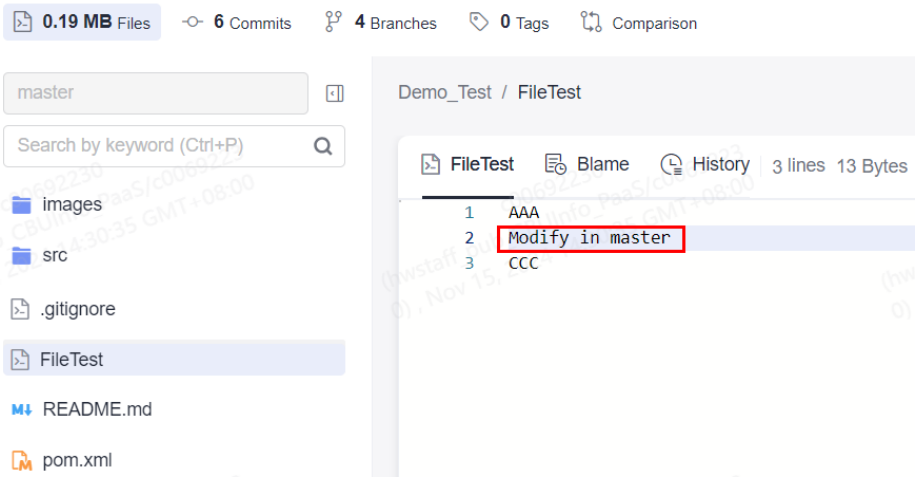
Figure 15-1 Creating the **FileTest** file in the master branch



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.

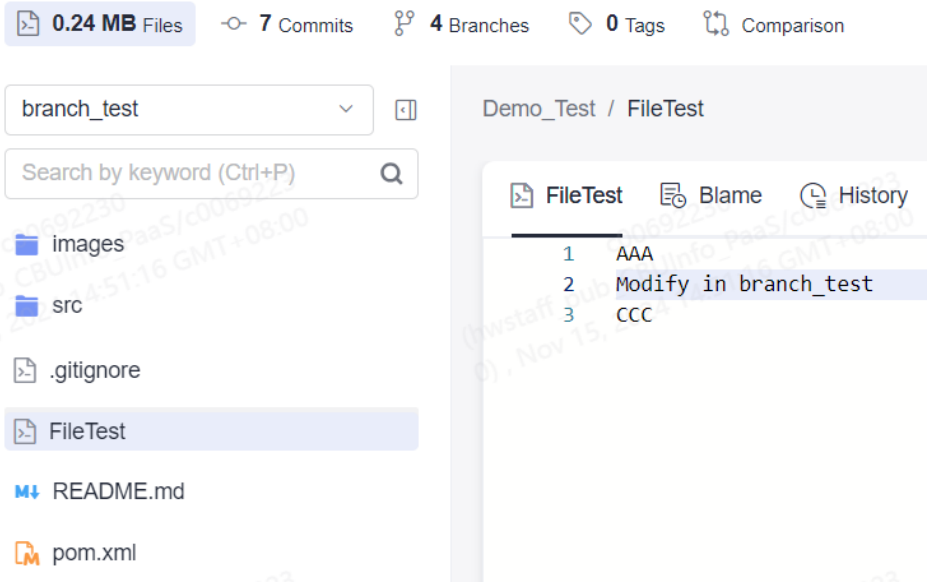
Step 4 In the master branch, modify **FileTest** as shown in the following figure, and enter the commit message **Update FileTest in master**.

Figure 15-2 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.

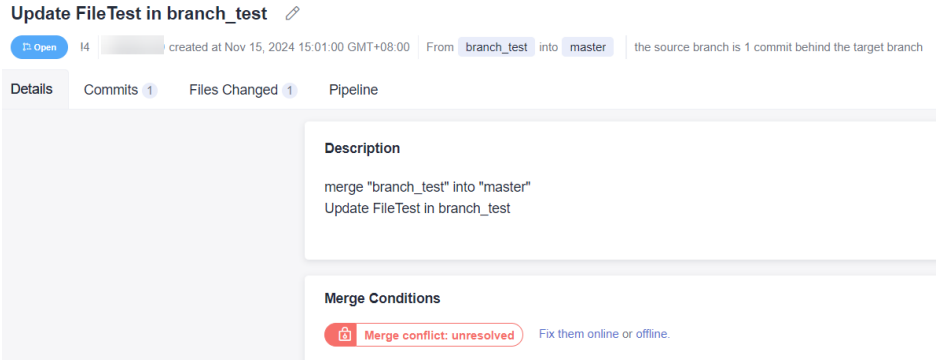
Figure 15-3 Modifying the **FileTest** file in the **branch_test** branch



Step 6 Switch to the **branch_test** branch, click **Create MR** 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**.

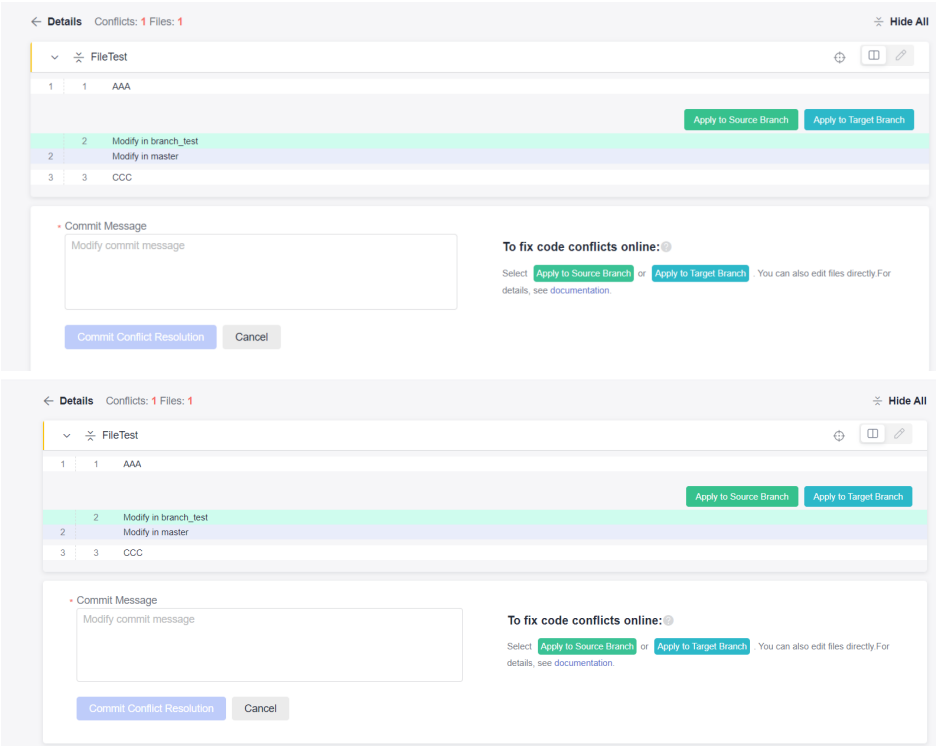
Figure 15-4 Creating a merge request



Step 7 Perform the following operation to resolve the conflict:

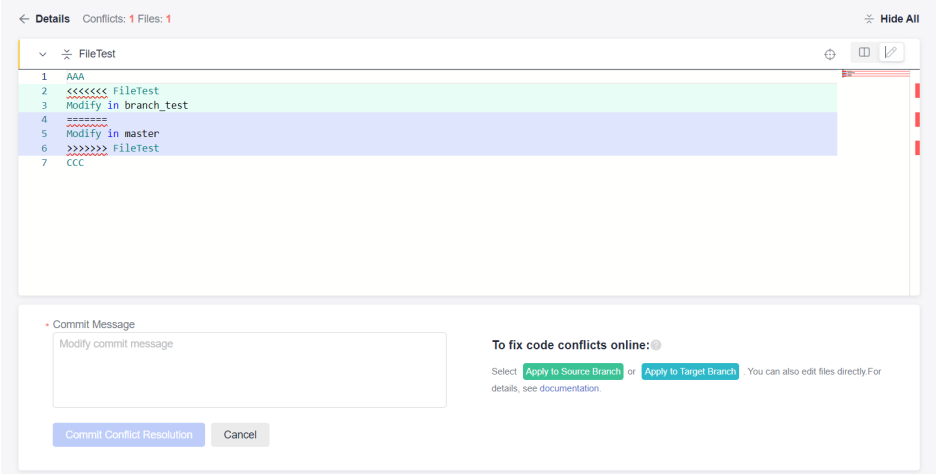
- **Resolving a conflict on CodeArts Repo** (recommended for small code volume)

  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.



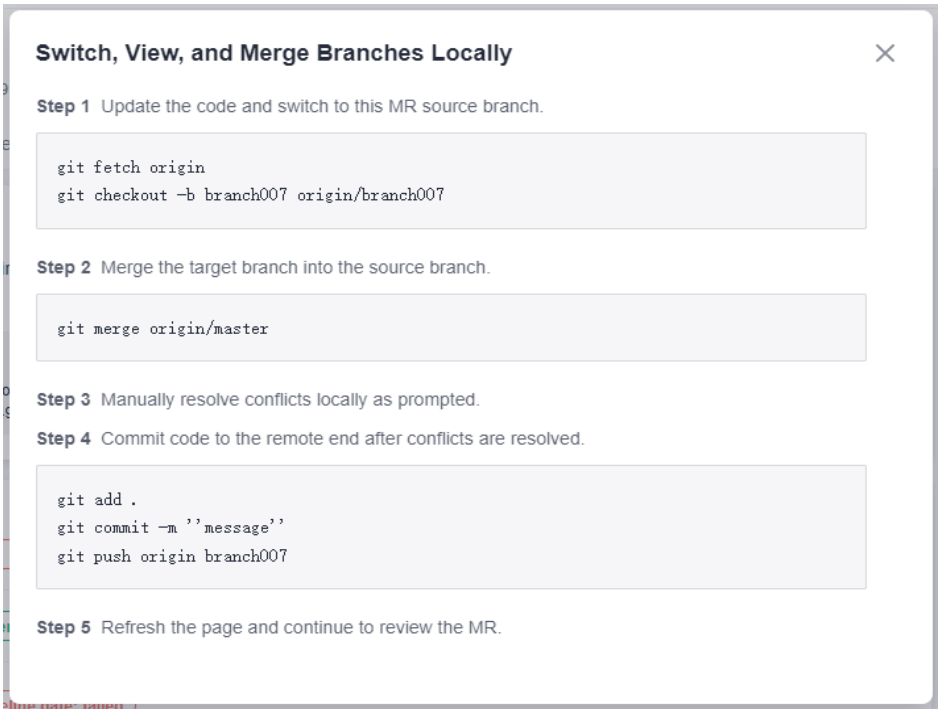
  b. If there are many conflicts, click to go to the page shown in the following figure to edit or resolve them online. The lines where the **<<<<**, **>>>>**, and **====** signs are located display conflict and separator which lines need to be deleted when modifying the code to solve the conflict.

Figure 15-5 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

15.4 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

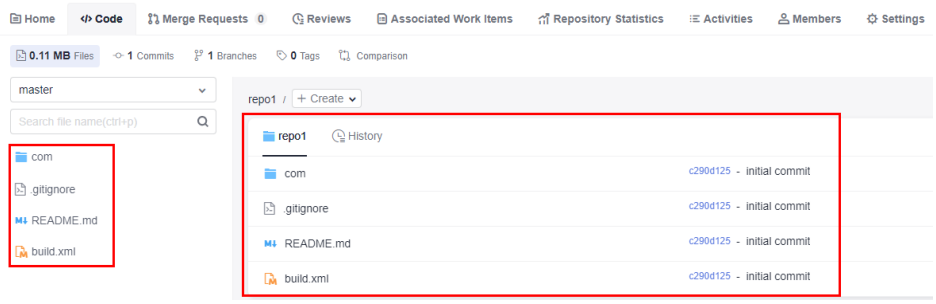
After you have resolved the issue raised in the review comments, you can switch the status of the review comments from **Unresolved** to **Resolved** in **Details** > **Review Comments** of the **Merge Request**. In this case, the status of the review comments is displayed as **Review comment gate: passed**.

16 Managing Code Files

16.1 Managing Files

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

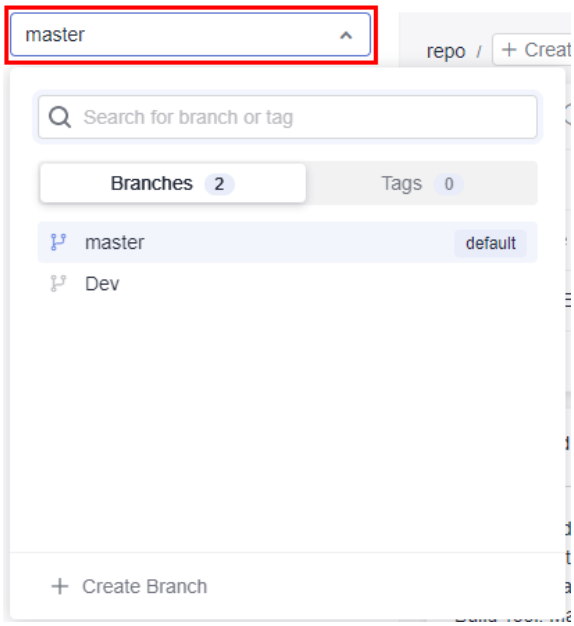
When you access the repository details console, you will be on the **Files** subtab of the **Code** tab page. You can switch to different branches and tags to view the files in the corresponding version. As shown in the following figure, the file list under the main branch is displayed on the left, the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



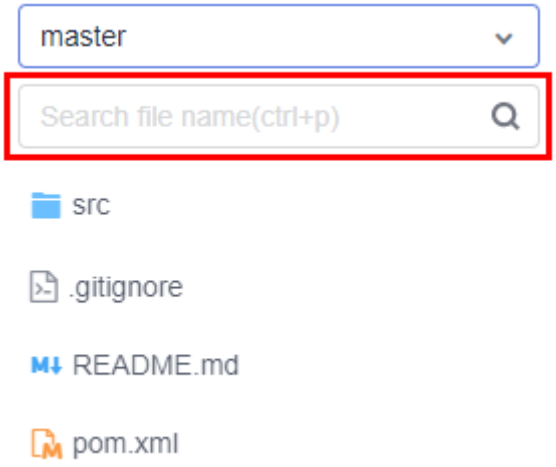
File List

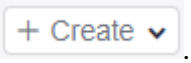
The file list is on the left of the **Files** tab page of the repository. The file list provides the following functions:

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**. The **Commit Message** field is equivalent to

the **-m** message in **git commit** and can be used to view associated work items.

- **Creating a directory**

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

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

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

- **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.

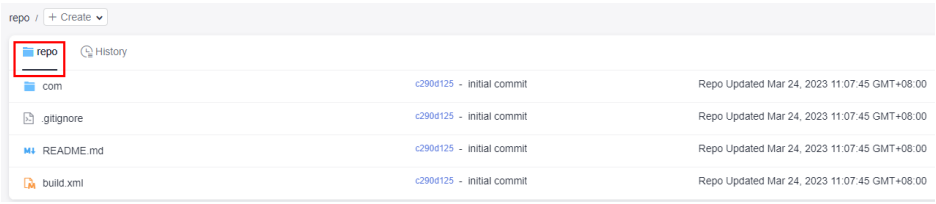
4. Move the cursor to the file name and click  to change the file name.

   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.



It displays the following information:

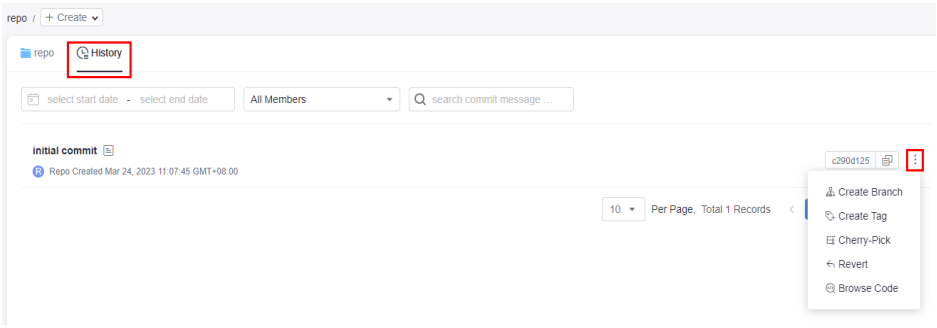
- *File*: name of a file or folder.
- *Commit message*: message of the last commit to the file or folder (**-m** in the **commit** command). You can click the message to display the commit record.
- *Creator*: creator of the last commit to the file or folder.
- *Update time*: last update time of the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

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.

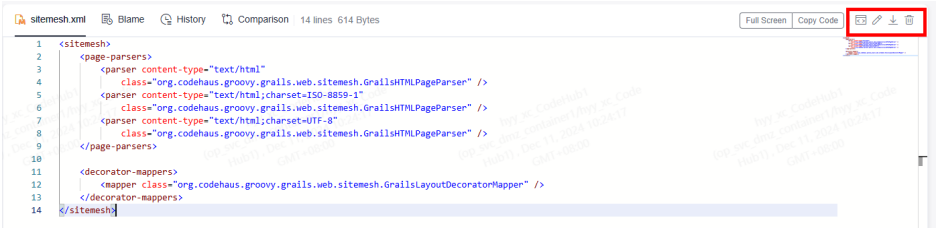


On this page, you can perform the following operations on the commit history:

- 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 16-1 Screen description

Screen Function	Function Description
File Capacity	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 to add a review comment. Max. 5000 review comments for a single MR.

- **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

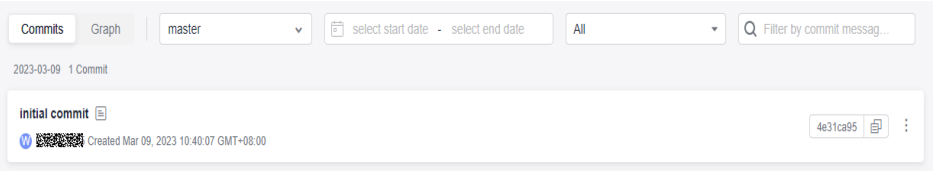
  The comparison result shows the impact of merging from the left repository version to the right repository version on the files in the right repository. If you want to know the differences between the two file versions, you can adjust the left and right positions, compare them again, and learn all the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

16.2 Managing Commits

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

Commits

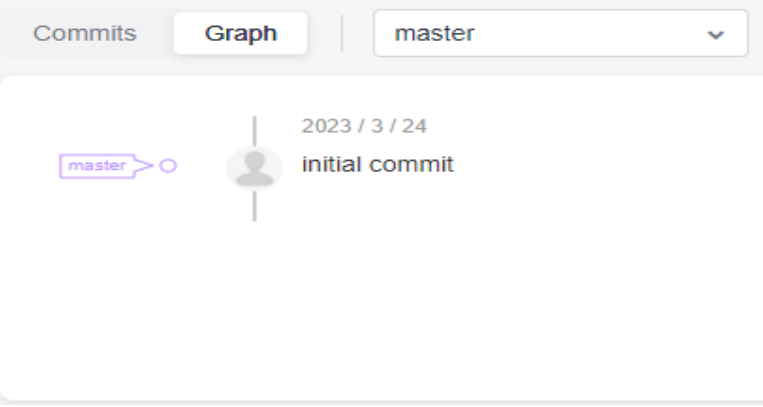
This tab displays the entire commit records of a branch or tag in the current repository. You can filter records by time segment, committer, commit message, or commit ID.



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.

16.3 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.

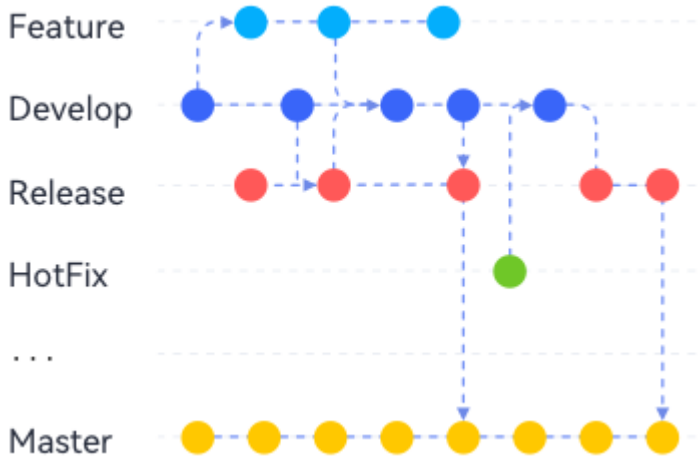


Table 16-2 Suggestions on using GitFlow branches

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.	• 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...
When to Merge Other Branches into This Branch	• 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.	• 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.	After a child feature branch is developed and tested, it is merged into the parent feature branch.	When a version is to be released, the develop branch is merged into this branch.	-

Branch	Master	Develop	Feature_1\2...	Release	HotFix_1\2...
When to Merge This Branch to Other Branches	-	• 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.	• 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 Branch**. In the dialog box that is displayed, select the version (branch or tag) based on which you want to create a branch, and fill in the branch name. Do not start the branch name with a hyphen (-), period (.), **refs/heads/**, **refs/remotes/**, **/**, or end with a period (.), slash (/), or **.lock**. Do not use two consecutive periods or the sequence @{. Do not use spaces and the following special characters: [\ < ~ ^ : ? * ! ( ) ' " | $ & **Note**: The name cannot be the same as another branch or tag name.

Step 5 You can set **Description** for the new branch and select **CodeArts Req Work Items** to associate with. For details about the work item types that can be associated with, see **E2E Settings**.

Step 6 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.

You can also manage default branches, merge requests, and protected branches.

Common Git Commands for Branches

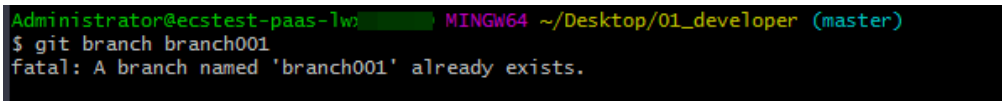
- **Creating a branch**

  ```
  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.

- **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.

- **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.

- **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>*.)

- **Merging a branch**

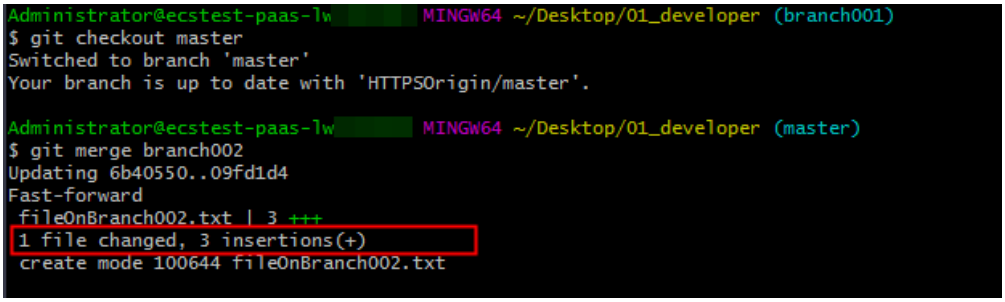
  When a development task on a branch is complete, the branch needs to be merged into another branch to synchronize the latest changes.

  ```
  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.

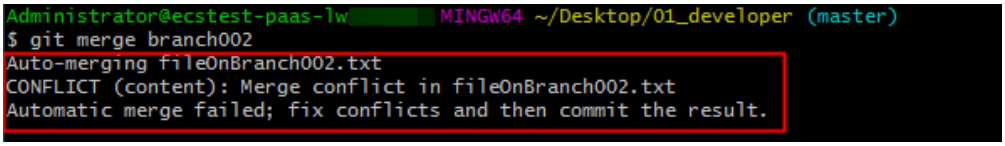
  ```
  git checkout master             # Switch to the master branch.
  git merge branch002             # Merge branch002 into the master branch.
  ```


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

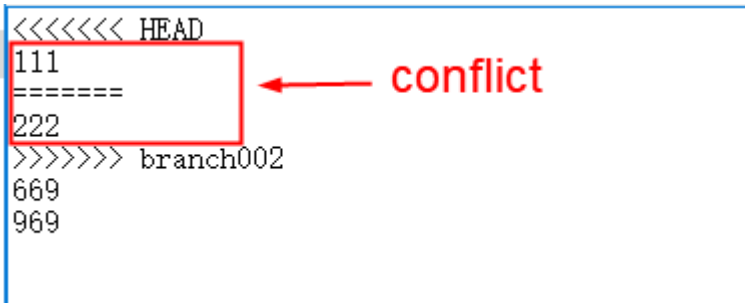


NOTE

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



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.

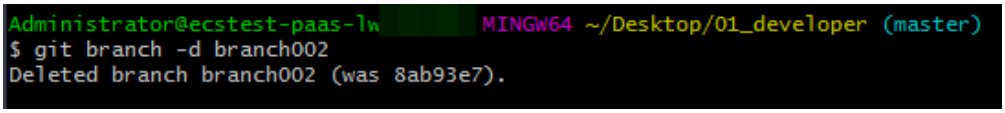


- **Deleting a local branch**

  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.



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

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

  Example:

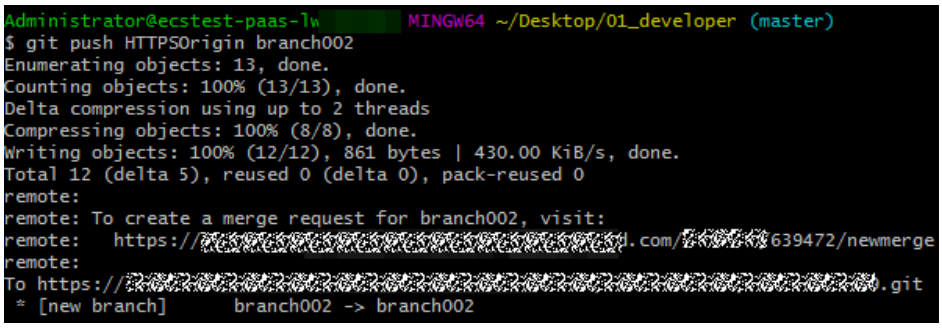
  git push HTTPSOrigin -d branch002 # Delete **branch002** from the remote repository whose alias is **HTTPSOrigin**. The following information shows that the deletion is successful.

- **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.
```



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.

16.4 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**.

Creating a Tag for the Latest Commit on the Console

Step 1 Access the repository list.

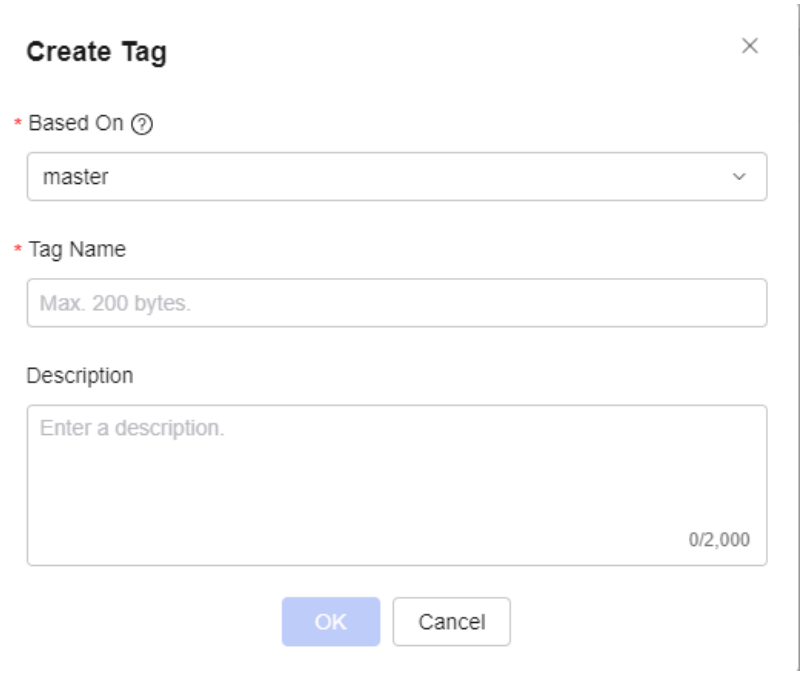
Step 2 Click a repository to go to the details page.

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. 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 **@{**.

If you set **Description**, an annotated tag will be generated. Otherwise, a lightweight tag will be created. 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.

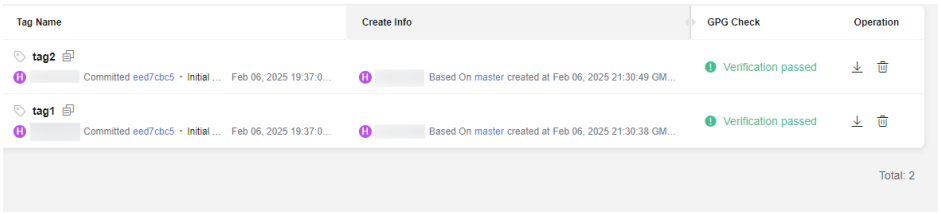
Step 3 In the historical commit list, click ⋮ next to a commit record and select **Create Tag**. The dialog box for creating a tag for the historical version is displayed. If you set **Description**, an annotated tag will be generated. Otherwise, a lightweight tag will be created.

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

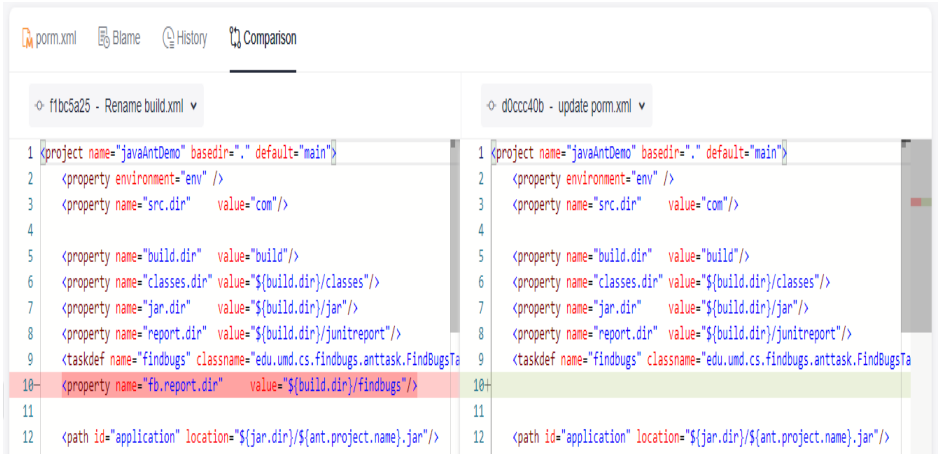
----End

Managing Tags on the Console

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



  - 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. 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.
  - Click  to delete a tag from CodeArts Repo. (To delete the tag from the local repository, perform the **clone**, **pull**, or **-d** operation.)
- You can create a branch based on a tag.
- On the console, click the **Files** tab and click the file name of the target file. Click the **Comparison** tab to compare commit records of the file.



Tag Classification

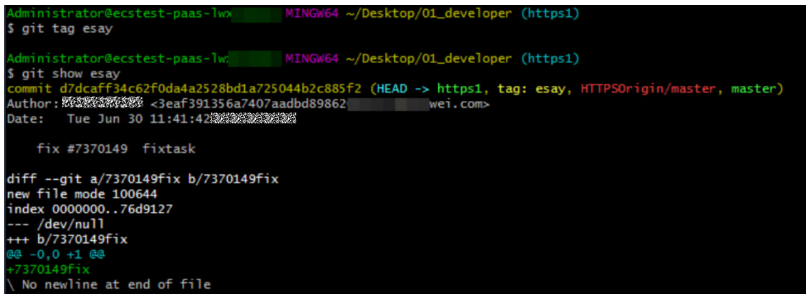
Both types of tags can identify versions. **Annotated tags** contain more information and are stored in a more stable and secure structure in Git. They are more widely used in large projects.

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>
```

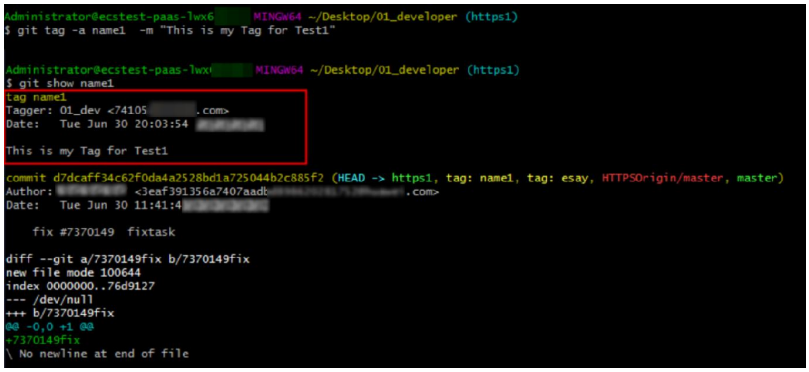
  The following figure shows the information of a lightweight tag. You can find that it is an alias of a commit.



- **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.



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              #Create a tag in the repository of the current version.
  ```

- **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:

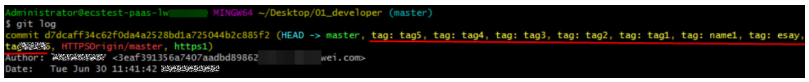
  ```
  git log             # The historical commit information is displayed. Obtain the commit ID (only the first several digits are required), as shown in the following figure. Press q to return.
  ```


  ```
  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.".
  ```


- **FAQs About Creating a Tag**
  - 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.

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

- **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
  ```

- **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: 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.

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

- **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

This section will help you understand the process of retrieving a historical version using tags. You can choose Git user commands as needed to perform operations in specific scenarios. It is not recommended to copy the entire process.

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.
   ```

   ```
   [redacted]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.
   ```

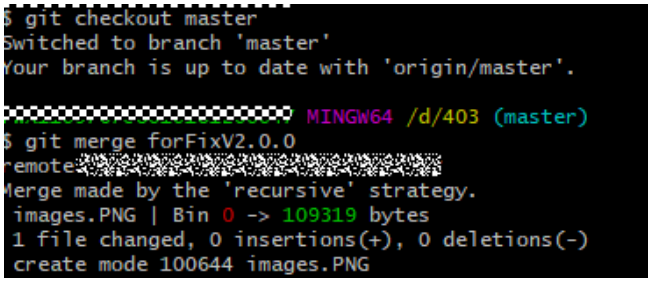
   ```
   [redacted] MINGW64 /d/403 ((V2.0.0))
   $ git switch -c  forFixV2.0.0
   Switched to a new branch 'forFixV2.0.0'
   ```

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

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


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

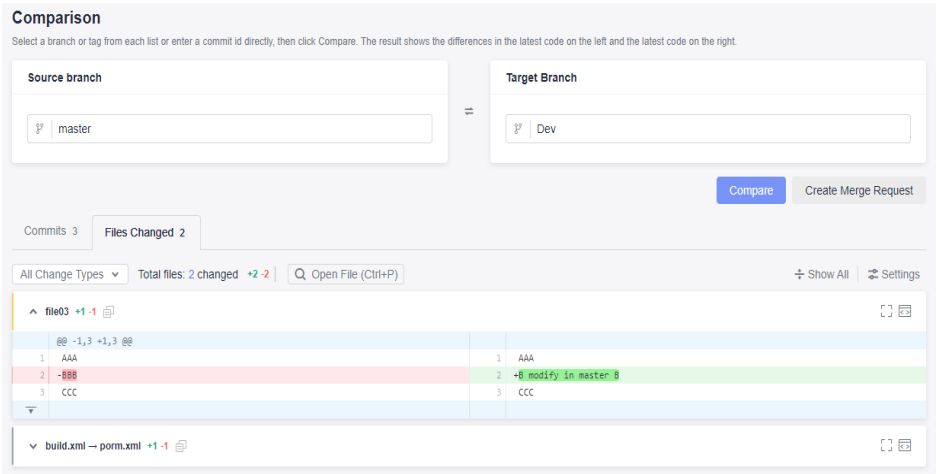
```
git checkout master                # Switch to the master branch.
git merge forFixV2.0.0             # Merge the changes based on the historical version into the master branch.
```



16.5 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 following figure shows the file changes of the source branch **master** and target branch **dev**.



17 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, adjust repository visibility, and record audit logs. For details, see the following sections. Only the users who have the permission to set repository groups or repositories can perform these operations. For details about the set permission, see **Configuring Repo-Level Permissions**.

17.1 Configuring a Deploy Key for a Repository

Constraints

- 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.

Configure 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 Key**.

17.2 Risky Operations

Risky Operations on Repositories

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**. The following operations are supported:

- **Transfer Repository Ownership**: Only the repo owner can perform this operation. You can transfer the current repository to another member in the repository.
- **Delete Repository**: Only members who have the permission to delete repositories can perform this operation. Enter **Delete** to delete the repository. Once you delete the code repository, all its content will be permanently deleted. This operation cannot be undone. Please exercise caution.
- **Rename Repository**: Only the repository owner can change the repository name. This will invalidate the original path for access and clone. Please exercise caution.
- **Transfer Repo**: Only members who have the permission to delete repositories can perform this operation. This will invalidate the original path for access and clone. Please exercise caution.

17.3 Adding Watermarks to a Repository

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

Adding a Watermark to a Repository in a Project

Go to the homepage of the target project, and choose **Settings > Security Management > Watermark**. If this is enabled, the repositories will be displayed with the watermarks of the account and time.

If you want all code repositories under the project to inherit this configuration, select **Inherit from project**.

Adding Watermarks to a Repository

Go to the homepage of the target repository and choose **Settings > Security Management > Watermark Settings**. If **Inherit project settings** is selected, all configurations will be the same as the project-level configurations and cannot be modified.

Click **View details** to go to the project-level configuration page and check whether the watermark function is enabled in the project-level configuration.

17.4 Locking a Repository

- Overview

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

- Constraints

Repository members with the set permission can perform this operation.

- Locking a Repository

Go to the repository homepage to be configured, choose **Settings > Security Management > Repository Locking**, and enable the watermark setting function to lock the repository. After the repository is locked, it is read-only, and no one can commit code to any branch, create comments, or perform other new operations.

17.5 Configuring an IP Whitelist

You can access and configure the tenant-level IP address whitelist if you are a tenant or an IAM user assigned as the **te_admin** role by the tenant. For details, see **Configuring Repo-Level Permissions**.

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 can be used. 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  in the row where the IP address whitelist is located.

Table 17-1 Parameters for creating an IP address whitelist

Parameter	Description
IPv4	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: • **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.
Description	Optional.
Access Control	Optional. Select the corresponding options as needed. • **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.

FAQs About Configuring IP Address Whitelist

- If you want to import the repository of another tenant but they have configured an IP address whitelist, the import will fail. In this case, contact

your administrator to obtain the proxy IP address and ask the target tenant to add the proxy IP address to the IP address whitelist.

- To set an IP address whitelist for all repositories of your 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.

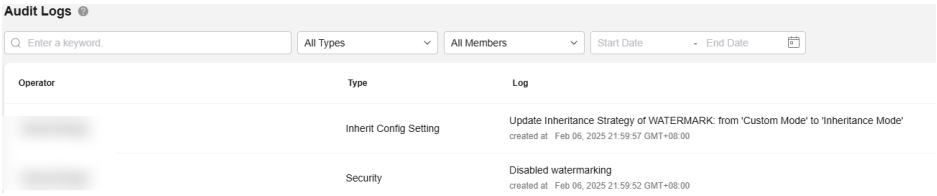
17.6 Audit Logs

17.6.1 Repository Audit Logs

CodeArts Repo allows you to modify repository attributes. It records information such as general settings, policy settings and E2E settings of the repositories. Each audit log contains the operator, operation type, and operation content.

As shown in the following figure, you can filter and view data by time. The operator, type, and information about audit logs are displayed.

Figure 17-1 Viewing audit logs



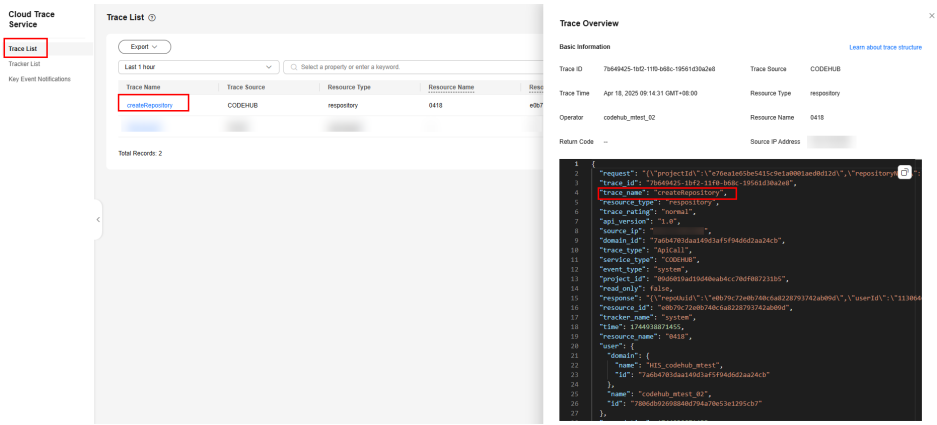
17.6.2 CTS Audit Logs

Constraints

You need to enable Cloud Trace Service (CTS) in advance.

Viewing CTS Audit Logs

Currently, CodeArts Repo reports repository creation and deletion events to CTS. You can view CTS audit logs by referring to . The figure shows the **repository creation** event.



17.7 Adjusting Repository Visibility

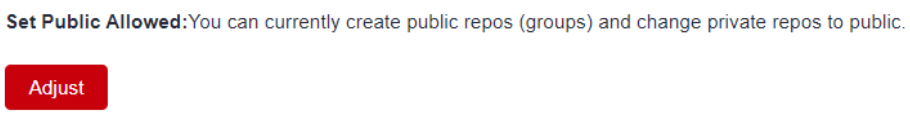
Constraints

You can adjust the openness of a repository only when you are a tenant or an IAM user with the te_admin role assigned by the tenant. For details, see **Configuring Repo-Level Permissions**.

Adjusting Repository Visibility

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.



- 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.