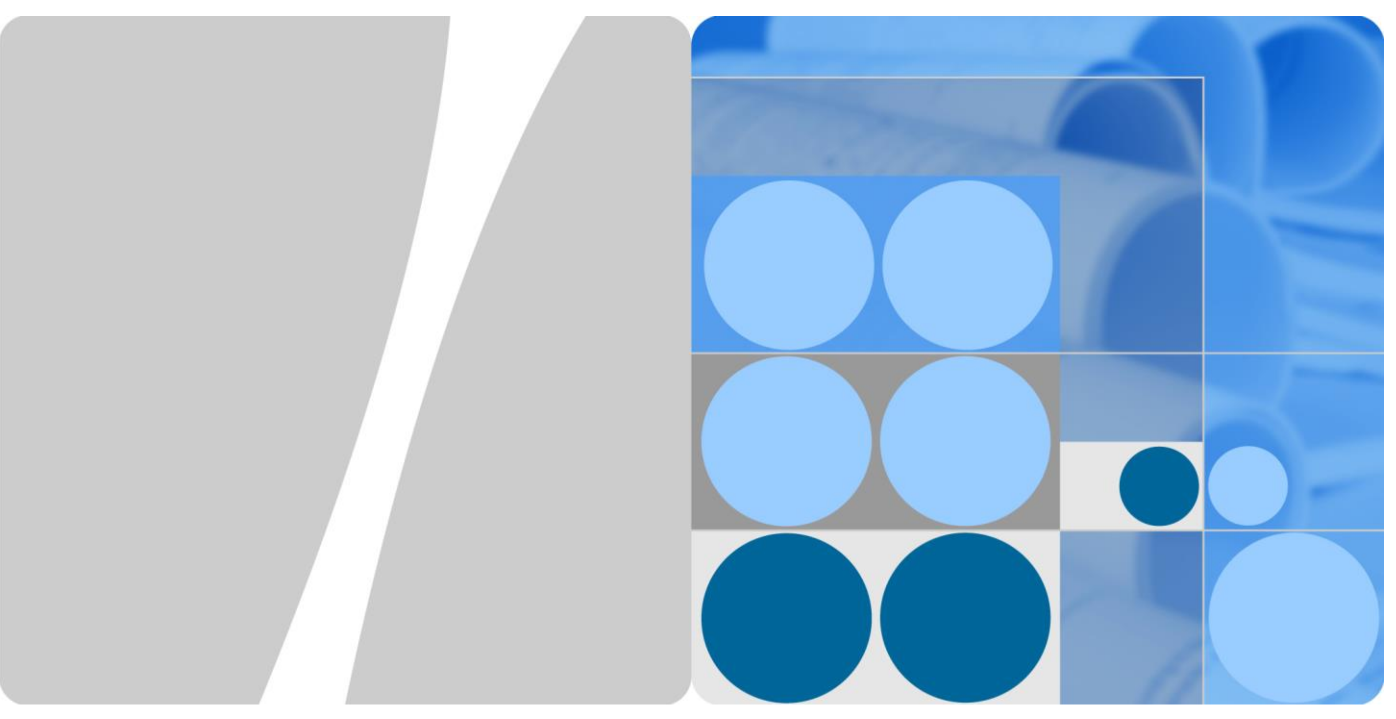


代码托管

用户指南

发布日期 2024-08-30

目 录

1 服务概述......5

2 Git 客户端安装配置......9

2.1 Git 客户端安装与配置 9

2.2 Windows Git Bash 客户端 10

2.3 Windows TortoiseGit 客户端 10

2.4 Linux Git 客户端......12

2.5 Mac Git 客户端 12

3 设置代码托管仓库的 SSH 密钥/HTTPS 密码/访问令牌 14

3.1 概述 14

3.2 SSH 密钥......15

3.3 HTTPS 密码 17

3.4 访问令牌 18

4 迁移到代码托管仓库 21

4.1 概述 21

4.2 将基于 Git 的远程仓库导入代码托管 21

4.3 将本地代码上传到代码托管 23

5 创建代码托管仓库......25

5.1 概述 25

5.2 创建普通仓库 26

5.3 按模板新建仓库 29

5.4 导入外部仓库 31

5.5 Fork 仓库......32

6 关联代码托管仓库......36

7 克隆/下载代码托管仓库到本地 39

7.1 概述 39

7.2 使用 SSH 协议克隆代码托管仓库到本地 39

7.3 使用 HTTPS 协议克隆代码托管仓库到本地 42

7.4 从浏览器下载代码包 45

8 使用代码托管仓库......46
8.1 查看仓库列表......46
8.2 查看仓库详情......47
8.3 查看仓库首页......49
8.4 管理代码文件......51
8.4.1 文件管理......51
8.4.2 提交管理......55
8.4.3 分支管理......56
8.4.4 标签管理......64
8.4.5 对比管理......70
8.5 管理合并请求......71
8.5.1 合并请求管理......71
8.5.2 解决合并请求的代码冲突......78
8.5.3 评审意见门禁详解......84
8.5.4 流水线门禁详解......85
8.5.5 E2E 单号关联门禁详解......86
8.5.6 星级评价门禁详解......88
8.5.7 检视门禁详解......89
8.5.8 审核门禁详解......90
8.6 查看仓库的评审记录......91
8.7 查看关联工作项......93
8.7.1 概述......93
8.7.2 Commit 关联......96
8.8 查看仓库的统计信息......100
8.9 查看仓库的动态......101
8.10 管理仓库成员......102
8.10.1 IAM 用户、项目成员与仓库成员的关系......102
8.10.2 配置成员管理......102
9 配置代码托管仓库......104
9.1 基本设置......104
9.1.1 仓库信息......104
9.1.2 通知设置......105
9.2 仓库管理......108
9.2.1 仓库设置......108
9.2.2 仓库加速......109
9.2.3 同步设置......109
9.2.4 子模块设置......111
9.2.5 仓库备份......114
9.2.6 同步仓库......114

9.3 策略设置 115
9.3.1 检视意见 115
9.4 服务集成 116
9.4.1 E2E 设置 116
9.4.2 Webhook 设置 120
9.5 模板管理 122
9.5.1 合并请求模板 122
9.5.2 检视评论模板 122
9.6 安全管理 123
9.6.1 部署密钥 123
9.6.2 IP 白名单 124
9.6.3 风险操作 126
9.6.4 水印设置 126
9.6.5 锁定仓库 127
9.6.6 审计日志 128
9.6.7 权限管理 128
10 提交代码到代码托管仓库 134
10.1 创建提交 134
10.2 加密传输与存储 136
10.3 查看提交历史 147
10.4 在 Eclipse 提交代码到代码托管 148
11 更多 Git 知识 160
11.1 Git 客户端使用 160
11.2 使用 HTTPS 协议设置免密码提交代码 163
11.3 TortoiseGit 客户端使用 165
11.4 Git 客户端示例 170
11.4.1 Git 客户端上传下载代码 170
11.4.2 Git 客户端修改文件名大小写后，如何提交到远端 171
11.4.3 Git 客户端设置系统的换行符转换 171
11.4.4 Git 客户端提交隐藏文件 172
11.4.5 Git 客户端提交已被更改的文件 172
11.5 Git 常用命令 173
11.6 Git LFS 使用 176
11.7 Git 工作流 178
11.7.1 Git 工作流概述 178
11.7.2 集中式工作流 178
11.7.3 分支开发工作流 179
11.7.4 Git flow 工作流 180
11.7.5 Forking 工作流 181

1 服务概述

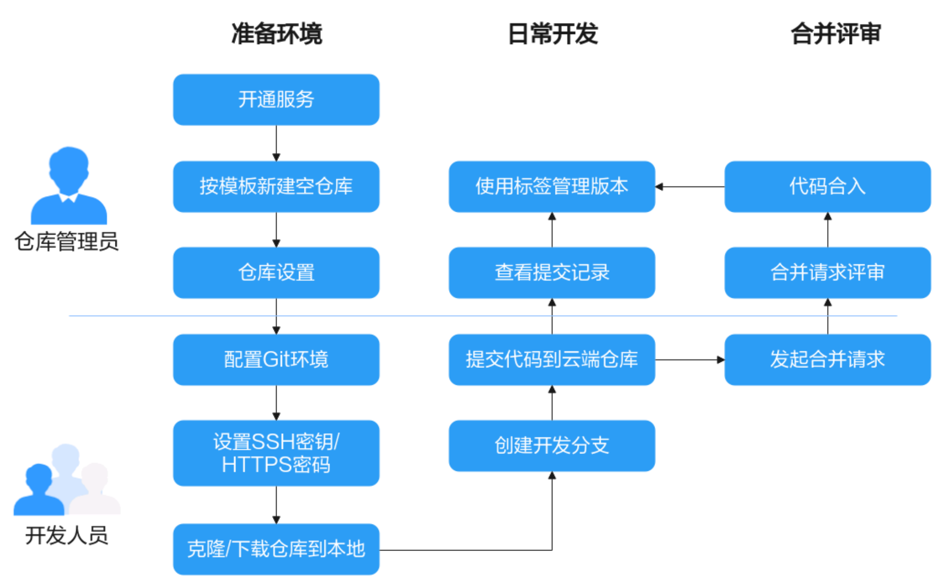
代码托管（CodeArts Repo）是遵循 Git 的基本运行模式的分布式版本管理平台，其具备安全管控、成员/权限管理、分支保护/合并、在线编辑、统计服务等功能，旨在解决软件开发者在跨地域协同、多分支并发、代码版本管理、安全性等方面的问题。

如果您计划开始一个新项目，那么您可以选择使用代码托管内置的仓库模板创建仓库并开始开发，流程请参见在代码托管仓库开始研发项目。

如果您本地正在开发一个项目，想使用代码托管服务来管理版本，可以将项目迁移到代码托管仓库，流程请参见将本地项目迁移到代码托管仓库。

在代码托管仓库开始研发项目

如果您全新开始一个项目，那么您可以选择使用代码托管服务为您提供的仓库模板来创建项目并开始开发，其使用流程如下。



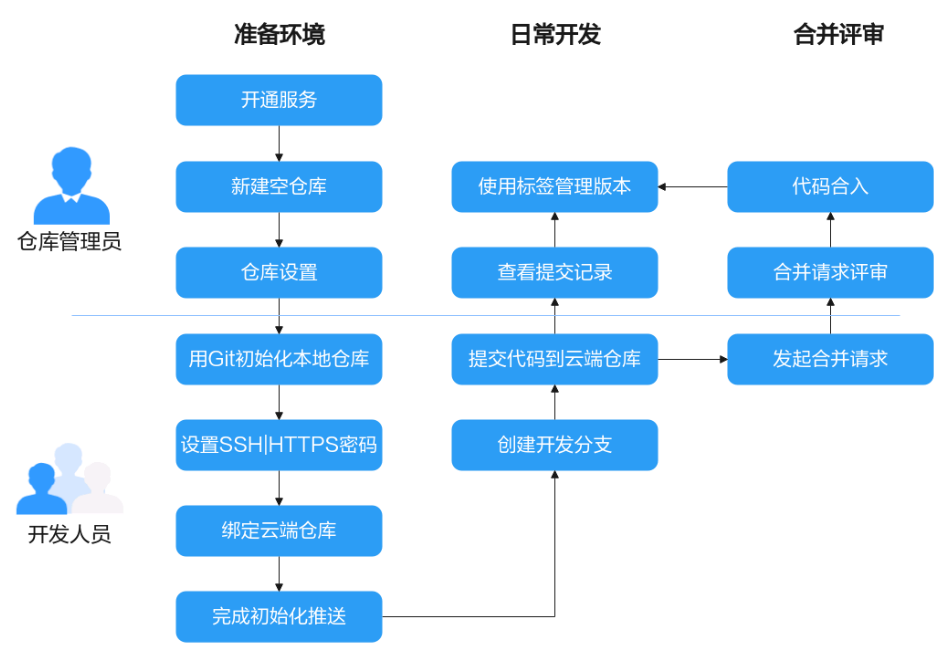
所涉及到的操作或知识如下：

- 按模板新建仓库
- 配置成员管理

- 配置代码托管仓库
- Git 客户端安装配置
- 克隆/下载代码托管仓库到本地
- 分支管理
- 标签管理
- 提交代码到代码托管仓库
- 合并请求管理
- Fork 仓库

将本地项目迁移到代码托管仓库

如果您本地正在开发一个项目，想使用代码托管服务来管理版本，那么您可以将本地仓库绑定代码托管仓库并完成初始化推送，之后便可以使用分布式版本管理方式来继续开发您的项目，其使用流程如下。



所涉及到的操作或知识如下：

- 创建普通仓库
- 配置成员管理
- 配置代码托管仓库
- Git 客户端安装配置
- 关联代码托管仓库
- 克隆/下载代码托管仓库到本地
- 分支管理
- 标签管理
- 提交代码到代码托管仓库
- 合并请求管理

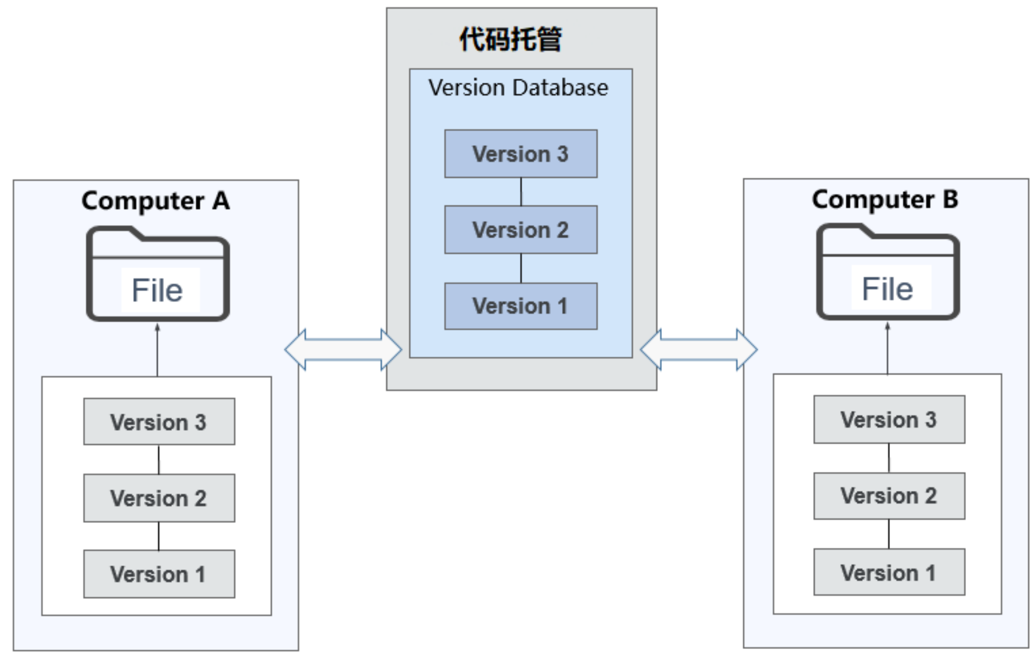
- Fork 仓库

分布式版本管理

您的本地计算机与代码托管服务中各有一个完整的代码仓库。

所有版本信息可同步到本地计算机，这样就可以在本地计算机查看所有版本历史。

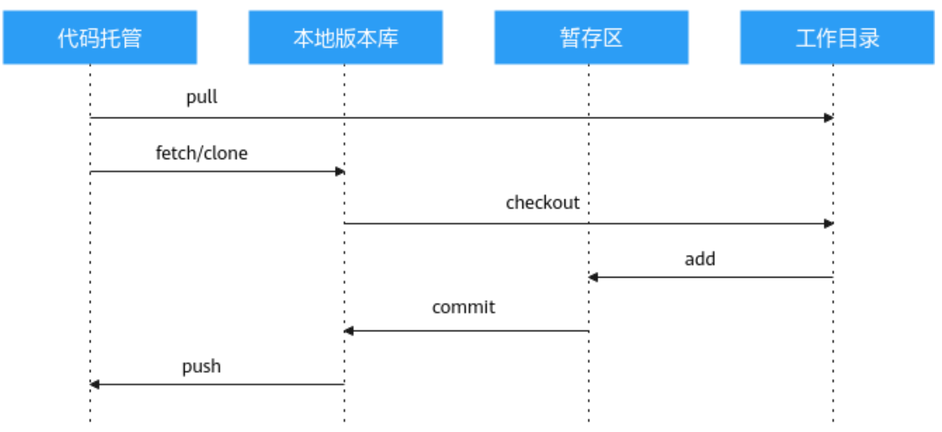
可以离线在本地计算机提交，只需在连网时推送到代码托管仓库即可。



基本运行模式

代码托管（CodeArts Repo）是基于 Git 的一种云端仓库服务，其遵循 Git 的工作模式。

- Git 本地仓库中的数据有三种状态，分别是“已修改”、“已暂存”和“已提交”。当您对仓库中的文件做出修改后，该文件状态为“已修改”，您可以通过 **add** 命令将该修改追加到本地的暂存区，此时状态为“已暂存”，再通过 **commit** 命令将修改提交到本地版本库进行管理，每次提交都会生成对应的版本和版本号，通过版本号可以进行版本的切换、回滚。同一版本中还可以同时存在多个分支、标签，每个分支、标签、每次提交又相当于独立的版本可以使用 **checkout** 进行检出。
- 代码托管作为云端仓库服务，其除了具备 Git 本地仓库的基本特性外，还作为各个本地仓库的远程版本库，并具备可配置的安全策略、鉴权等。
- 代码托管服务的仓库与 Git 本地仓交互的场景主要有以下四种：
  - Clone：直接将代码托管仓库的分支克隆到本地，作为本地仓库。
  - Push：将本地仓库的修改推送到代码托管仓库。
  - Fetch：从代码托管仓库抓取版本到工作区。
  - Pull：从代码托管仓库抓取版本到工作区并尝试与当前分支合并，如果失败，需要手动解决文件冲突。


代码托管
本地版本库
暂存区
工作目录
pull
fetch/clone
checkout
add
commit
push

2 Git 客户端安装配置

Git 客户端安装与配置

Windows Git Bash 客户端

Windows TortoiseGit 客户端

Linux Git 客户端

Mac Git 客户端

2.1 Git 客户端安装与配置

代码托管基于 Git 工具，开发人员的本地环境需要安装 Git Bash 或 TortoiseGit 等 Git 客户端工具，实现与代码托管服务的连接。后续章节介绍 Git Bash、TortoiseGit 的安装与简易配置，其中 Git 客户端支持在 Windows、Linux、Mac 操作系统中运行。

如果您已经安装过 Git 客户端并且已经配置了签名和邮箱，可跳过以下章节。

- Windows Git Bash 客户端
- Windows TortoiseGit 客户端
- Linux Git 客户端
- Mac Git 客户端

说明

- 代码托管暂不支持使用 github desktop 进行管理。
- 用户名可以由字母、数字、常用符号组成，但是不支持以 ASCII 码表中小于等于 32 的字符以及 ‘.’、‘,’、‘:’、‘;’、‘<’、‘>’、‘"’、‘\\’、‘\’ 字符作为开头和结尾，如果首尾出现以上字符，则会直接被忽略。如为方便管理，可以考虑配置成与代码托管服务相同的用户名。
- 邮箱请按照标准邮箱格式填写。

2.2 Windows Git Bash 客户端

如果您不熟悉 Git 命令，推荐使用 Windows TortoiseGit 客户端的可视化操作界面，如果您熟悉常用的 Git 命令，Git Bash 将会是您 Windows 上更加简洁、高效的客户端。

1. 安装 Git Bash 客户端。
   a. 打开 Git Bash 官网，根据您的操作系统位数下载 32 位/64 位的安装包。
   b. 双击运行安装包，在弹出的安装窗口中依次单击“下一步（Next）”，最后单击“安装（Install）”完成安装。
2. 打开 Git Bash 客户端。

   单击 Windows“开始”图标，在“开始”搜索栏中输入“**Git Bash**”，单击回车即可打开 Git Bash 客户端，建议将其固定到 Windows 的任务栏中。
3. 配置 Git Bash 客户端。

   配置用户名和邮箱，在 Git Bash 中输入以下命令行：

```
git config --global user.name 您的用户名
git config --global user.email 您的邮箱
```

   配置好之后可以使用以下命令行查看配置：

```
git config -l
```

说明

- git config 命令的--global 参数，用了这个参数，表示您这台机器上所有的 Git 仓库都会使用这个配置，您也可以对某个仓库指定不同的用户名和 Email 地址。

2.3 Windows TortoiseGit 客户端

如果您不熟悉常用的 Git 命令，那么 TortoiseGit 客户端将是您更好的选择。

前提条件

1. 打开 TortoiseGit 官网，根据您的操作系统位数下载 32 位/64 位的安装包。
2. 双击运行安装包，在弹出的窗口中依次单击“Next”，然后单击“Install”即可完成安装，最后单击“Finish”即会运行第一次启动引导。
3. 在弹出的第一次启动引导中，会有 Language 语言选择、Git 可执行路径配置（自动填充可用的 Git 路径）、配置用户名和邮箱，保持默认依次单击 Next 完成即可。

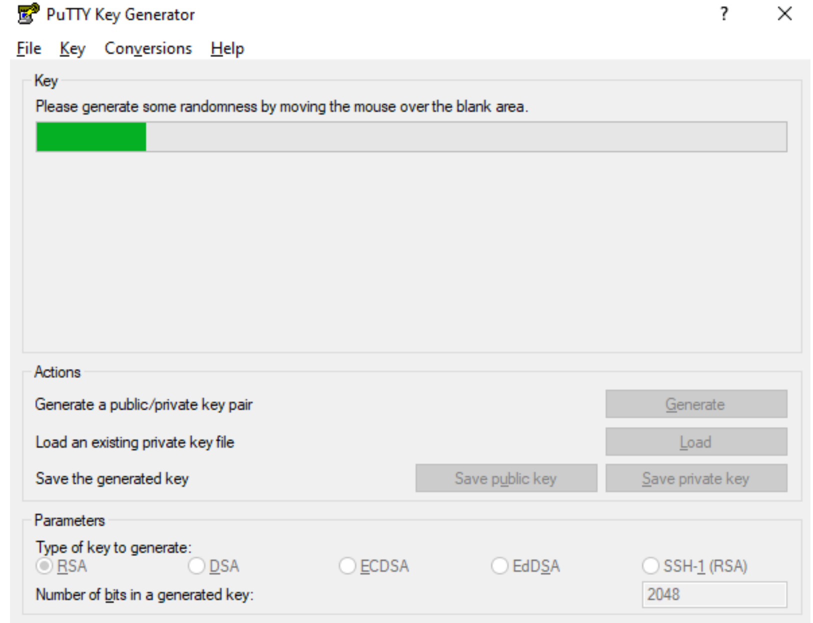
语言包（可选）

TortoiseGit 的安装包默认为英文，可以从 TortoiseGit 官网下载语言包（Language Packs）。

配置

TortoiseGit 同样需要一个密钥来和代码托管服务端进行鉴权认证，密钥生成步骤如下：

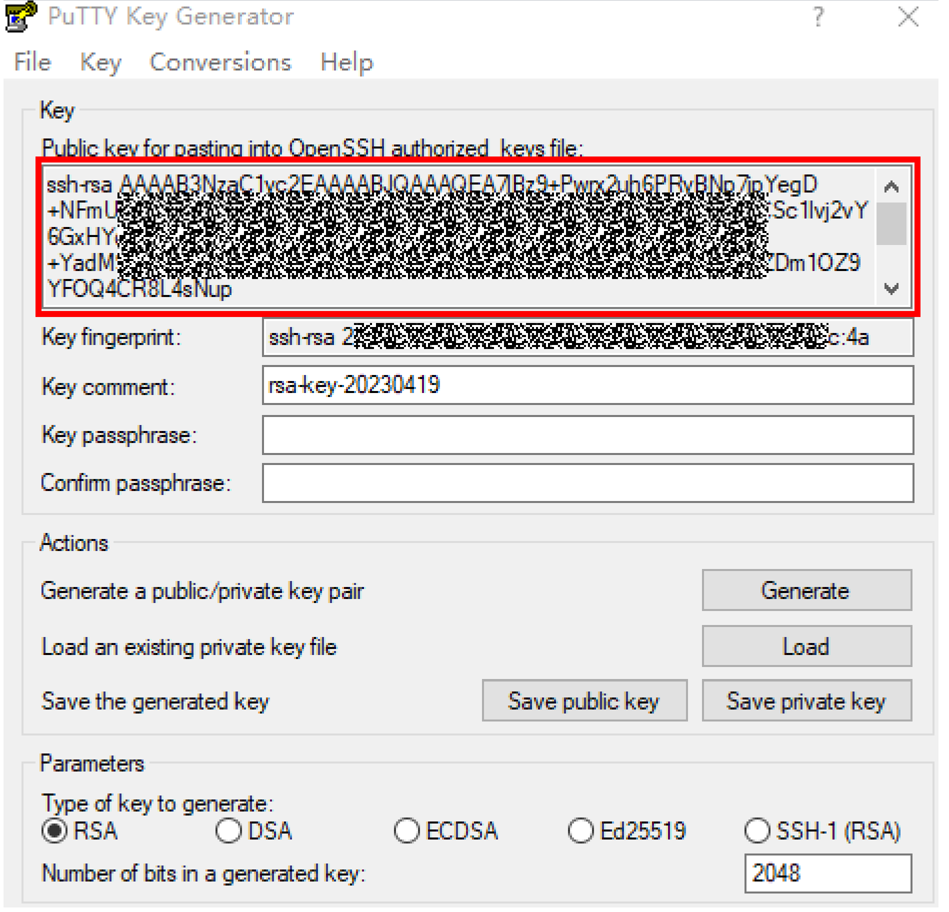
1. 单击 Windows“开始”图标搜索“PuttyGen”并打开，在打开的窗口中单击“Generate”，即可生成密钥。



说明

PuTTYgen 是 TortoiseGit 自带的一款功能强大并且小巧易用的公钥私钥生成工具软件，安装完 TortoiseGit 后即可在 Windows 开始图标搜索到 PuTTYgen。需要注意的是，TortoiseGit 与 PuTTY 都有自带 PuTTYgen，是不会冲突的。

2. 密钥生成后就可以分别将公钥、私钥进行存储。
   - 单击“Save private key”在弹出的窗口中输入文件名保存为私钥文件。
   - 单击“Save public key”在弹出的窗口中输入文件名保存为公钥文件。
3. 复制下图红框中的公钥并将公钥绑定到代码托管仓库。



4. 将私钥绑定到本地客户端。

   单击 Windows“开始”图标搜索“Pageant”并打开，单击“Add Key”，并选择您生成的私钥文件即可。

2.4 Linux Git 客户端

- Debian/Ubuntu 系统

  在终端中输入以下命令行安装：

```
apt-get install git
```

- Fedora/Centos/Redhat 系统

  在终端中输入以下命令行安装：

```
yum install git
```

- 更多操作系统请参见 Git 官网。

2.5 Mac Git 客户端

- Mac 上安装 Git 最简单的方法是安装 Xcode Command Line Tools；

- 在 Mavericks（10.9）或更高版本的系统中，在 Terminal 里尝试首次运行 Git 命令即可，如果没有安装过命令行开发者工具，将会提示您安装；
- 如果您想安装更新的版本，可以使用二进制安装程序，官方维护的 OSX Git 安装程序可以在 Git 官网下载。

3 设置代码托管仓库的 SSH 密钥/HTTPS 密码/访问令牌

概述

SSH 密钥

HTTPS 密码

访问令牌

3.1 概述

什么是 SSH 密钥/HTTPS 密码

当您需要将代码推送到代码托管仓库或从代码托管仓库下拉代码时，代码托管仓库需要验证您的身份与权限，SSH 和 HTTPS 是对代码托管服务进行远程访问的身份验证方式。

- **SSH 密钥**是在本地计算机与您账号下的代码托管服务之间建立安全连接。

  不同的用户通常使用不同的电脑，在使用 SSH 方式连接代码仓库前需要在自己电脑生成自己的 SSH 密钥，并设置到代码托管服务中。

  在一台本地计算机上配置了 SSH 密钥并添加公钥到代码托管服务中后，所有该账号下的代码仓库与该台计算机之间都可以使用该密钥进行连接。

- **HTTPS 密码**是 HTTPS 协议方式下载、上传时使用的用户凭证。

  在本产品中，HTTPS 协议所支持的单文件推送大小不超过 200M，需传输大于 200M 时，请使用 SSH 方式。

  因为无法绑定邮箱，所以无法使用 HTTPS 协议。

说明

使用其中任何一种方式都可以进行代码的上传下载，密钥（密码）的设置根据您选择的连接方式设定即可。

3.2 SSH 密钥

什么是 SSH 密钥

当您需要将代码推送到代码托管仓库或从代码托管仓库下拉代码时，代码托管仓库需要验证您的身份与权限，SSH 是对代码托管服务进行远程访问的身份验证方式。

- SSH 密钥是一种加密的网络传输协议，在电脑与您账号下的代码托管服务之间建立安全连接。
- 在一台电脑上配置了 SSH 密钥并添加公钥到代码托管服务中后，所有该账号下的代码仓库与该台电脑之间都可以使用该密钥进行连接。
- 不同的用户通常使用不同的电脑，在使用 SSH 方式连接代码仓库前需要在自己电脑生成自己的 SSH 密钥，并设置到远程仓库中。

生成并设置您的 SSH 密钥

以下介绍生成公钥和绑定的方法。

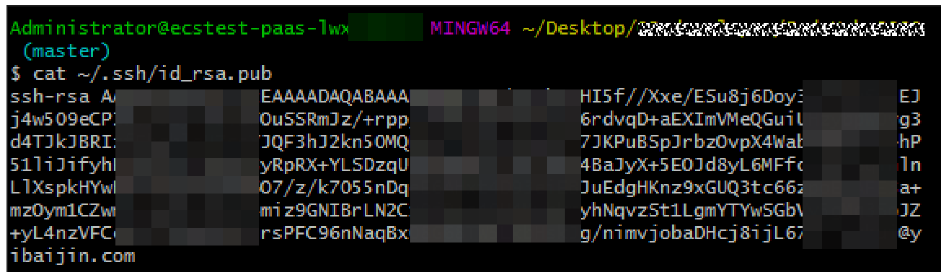
步骤 1 安装 Windows Git Bash 客户端。

步骤 2 检查您的计算机是否已经生成了密钥。

在本地 Git 客户端中执行命令，尝试显示 ssh 密钥。

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

- 如果提示“No such file or directory”如下图，则说明您这台计算机没有生成过 SSH 密钥，请从步骤 3 向下执行以生成并配置 SSH 密钥。
- 如果至少返回了一组密钥如下图，则说明您这台计算机已经生成过 SSH 密钥，如果想使用已经生成的密钥请直接跳到步骤 4，如果想重新生成密钥，请从步骤 3 向下执行。



步骤 3 生成 SSH 密钥。

在本地 Git 客户端中执行命令以生成新的 SSH 密钥：

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

```
Administrator@ecstest-paas-lwx MINGW64 ~/.ssh
$ ssh-keygen -t rsa -C ""
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): ①
/c/Users/Administrator/.ssh/id_rsa already exists.
Overwrite (y/n)? y ②
Enter passphrase (empty for no passphrase): ③
Enter same passphrase again:
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:NOsGrzQ6mHGUUUaNGWTd4a97GkC2gH+PJoUTudJHosM
The key's randomart image is:
+---[RSA 3072]----+
|      .+B* ...   |
|       *o.o..    |
|      + * + .    |
|     o + % . .   |
|    . E S %   .  |
|     = = @ * .   |
|    o o o + +    |
|       . o   o.  |
|            oo   |
+----[SHA256]-----+

Administrator@ecstest-paas-lwx MINGW64 ~/.ssh
```

请按照下面提示操作，上图所示为密钥已生成成功。

1 首先会提示您输入密钥的存储地址，一般直接回车使用默认即可。

2 如果您本地路径下已经有密钥，会询问您是否将其覆盖，填写“n”即选择不覆盖，会退出本次密钥生成操作，填写“y”继续生成密钥，本案例填写“y”并回车。

3 接下来会提示您为密钥设置密码和重新输入密码，如果不想设置密码，直接回车即可。

须知

- 如果设置密码，则生成的私钥文件是 AES-128-CBC 加密后存储的。（建议使用）
- 如果直接回车，不输入密码，则生成的私钥文件 id_rsa 是明文存储在本地的，请妥善保管。

步骤 4 复制 SSH 公钥到剪切板。

根据您的操作系统，选择相应执行命令，可将 SSH 公钥复制到您的剪切板，以 Windows 为例，无回显则为复制成功

- **Windows：**

```
clip < ~/.ssh/id_rsa.pub
```

- **Mac：**

```
pbcopy < ~/.ssh/id_rsa.pub
```

- **Linux (xclip required):**

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

步骤 5 登录您的代码托管服务仓库列表页，单击右上角昵称，单击“个人设置 > SSH 密钥管理”，进入页面。

步骤 6 在“SSH 密钥管理”页面，单击“添加 SSH 密钥”，弹出“添加 SSH 密钥”页面。

步骤 7 在**“标题”**中为您的新密钥起一个名称、将您在步骤 4 中复制的 SSH 公钥粘贴进**“密钥”**中，单击**“确定”**，页面会提示您操作成功。

说明

- 同一个 SSH 密钥不能重复添加，如果添加失败，请检查您是否已经添加过这个密钥、粘贴时前后是否有多余的空格。
- 添加成功后，可以在“SSH 密钥管理”页面查看到您添加的密钥，当您确认不再使用时，可以将其删除。
- SSH 密钥与仓库部署密钥的区别为，前者与用户/计算机关联，后者与仓库关联；SSH 密钥对仓库有读写权限，部署密钥对仓库是只读权限。

----结束

验证 SSH 密钥是否绑定成功

当 SSH 密钥绑定成功后，您可以在客户端对您有访问权限的仓库进行一次 SSH-clone 操作，如果克隆成功了，则说明密钥设置成功。

说明

如果是第一次使用 ssh 克隆仓库到本地，客户端会弹出“The authenticity of host *.*.com can't be established. RSA key...（yes/no）？”的提示，输入 yes 后表示信任方可继续。

3.3 HTTPS 密码

什么是 HTTPS 密码

当您需要将代码推送到代码托管仓库或从代码托管仓库拉取代码时，代码托管仓库需要验证您的身份与权限，HTTPS 是对代码托管服务进行远程访问的身份验证方式。

- **HTTPS 用户名**

  包含租户名/IAM 用户名，请完整输入，如果需要将用户名添加到 URL 中，请将'/'转义成 '%2F'。

说明

当主账号（账号和用户名一样）设置 HTTPS 密码时可以只写账号。

- **HTTPS 密码**
  - 请输入 8 到 32 位密码，数字、大小写字母及特殊字符至少包含三种，不能与用户名或者倒序的用户名相同。
  - HTTPS 密码是 HTTPS 协议方式下载/上传时使用的用户凭证。每个开发者，只需要设置一次密码，与仓库无关。
  - HTTPS 密码要妥善保存，不要外传，并定期更换，以免出现安全风险。如果忘记用户名密码，单击修改，设置新的 HTTPS 密码即可。

修改 HTTPS 密码

首次登录时需要设置初始密码，您也可以随时更改 HTTPS 密码，其步骤如下。

步骤 1 登录您的代码托管服务仓库列表页，单击右上角昵称，单击“个人设置 > HTTPS 密码”，进入页面。

步骤 2 单击“自行设置密码”，进入重设密码页面。（如果您之前自主设置过 HTTPS 密码并正在使用，直接单击“修改”。）

说明

- 如果您为第一次设置密码则可单击“**设置密码**”。
- 如果您需要修改密码则可单击“**修改**”，系统将向您个人邮箱发送验证码。

步骤 3 填写新密码与邮箱验证码，单击“**保存**”，页面会提示您操作成功。

步骤 4 密码重设完成后，需要在本地重新生成仓库凭证并检查 IP 白名单，否则不能与代码托管仓库交互。

删除该本地存储的凭证（以 Windows 为例“控制面板 > 用户账户 > 管理 Windows 凭据 > 普通凭据”），并使用 HTTPS 方式再次克隆，在弹出的窗口中输入正确的账号、密码即可。

说明

如果界面提示“**SSL certificate problem**”，请在 Git 客户端执行如下指令进行配置：

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

----结束

说明

- 本产品中 HTTPS 协议所支持的单文件推送大小不超过 200MB，需传输大于 200MB 时，请使用 SSH 方式。

验证 HTTPS 密码是否生效

当设置好 HTTPS 密码后，您可以在客户端对您有访问权限的仓库进行一次 HTTPS-clone 操作，会弹出对话框要求你输入账号、密码，填写后克隆成功，则说明密码设置成功。

说明

- 您也可以使用 HTTPS 协议设置免密码提交代码，请参考如何使用 HTTPS 协议设置免密码提交代码？
- 在使用登录密码进行 HTTPS 克隆代码时，仅支持三段式的 IAM 账号密码认证，对于租户账号的两段式的账号方式登录的密码不支持。
- 账号需要有“编程访问”权限才能正常认证通过。

3.4 访问令牌

什么是访问令牌

当第三方应用需要调用 API 接口时，代码托管仓库需要验证第三方应用的身份与权限，访问令牌是对调用 API 接口的第三方应用进行身份验证的方式。

访问令牌是指用户在代码托管仓库生成的个人凭证 Token，生成的 Token 可通过 HTTPS 方式读写仓库。

生成访问令牌

> **说明**
>
> 生成 Token 数量上限为 20 个，用于读写访问仓库，最长使用期限为 1 年。

步骤 1 登录您的代码托管服务仓库列表页，单击右上角昵称，单击“个人设置 > 代码托管 > 访问令牌”，进入页面。

步骤 2 在**访问令牌**页面，单击“新建 Token”，根据下表填写基本信息。

表3-1 参数说明

参数	说明
Token 名称	可自定义，用于与其他 Token 区分。字符上限为 200。
描述	描述可为空，字符上限为 200。当描述为空时，列表显示--。
权限	读/写 API：授予使用 Https 方式，读写访问仓库权限。
创建时间	Token 创建成功的时间。时间格式为 YYYY/MM/DD。
失效时间	Token 失效的时间。默认为当前日期的 30 天之后，包含当天共 30 天。例如 10 月 8 号新建的 Token，失效日期默认为 11 月 6 号 23 点 59 分 59 秒。失效日期最长为 1 年，且不可为空。时间格式为 YYYY/MM/DD。
操作	可以执行删除 Token 操作。

步骤 3 单击“**保存**”，Token 生成成功。

步骤 4 单击“**复制**”，保存到本地。

> **说明**
>
> Token 生成后，只会显示一次，之后无法查看，可将生成的 Token 自行妥善保管。如遗失或忘记，可重新生成使用。

----结束

使用访问令牌

步骤 1 进入代码托管仓库，您可使用 **git clone** 命令将仓库下载到本地 。通过克隆存储库，您可以将其文件的副本下载到本地计算机中，并保留与远程存储库的 Git 连接。

步骤 2 单击“**克隆/下载**”，单击“**用 HTTPS 克隆**”，复制访问方式中的 HTTPS 链接。

步骤 3 打开 Git Bash 客户端进入您的目录下，输入以下命令进行仓库克隆。

```
git clone 您的 HTTPS 下载链接
```

出现如下弹框，您可输入 Token 的用户名和密码。其中，Token 的用户名为“**private-token**”，密码为生成的 Token。

图3-1 输入 Token 的用户名

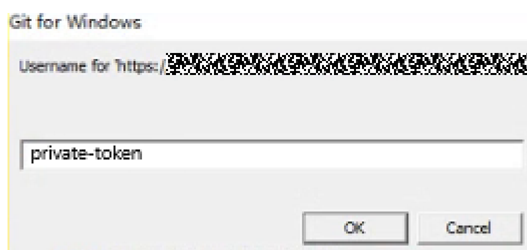


图3-2 输入 Token 密码

Git for Windows
Password for
'htt .com
OK
Cancel

步骤 4 单击 OK，即可完成代码托管仓库的克隆下载。

```
MINGW64 /d/test-2023
$ git clone htt
run
Cloning into '1121'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 227 bytes | 7.00 KiB/s, done.
```

----结束

说明

代码仓库支持通过将访问令牌嵌入 HTTPS 下载链接，嵌入格式如下：

https://username:password@codehub.xxx.com/{project_name}/{repository_name}.git

- username：private-token。
- password：通过生成的 Token 复制的 token。
- {project_name}：项目名称。
- {repository_name}：仓库名称。

```
user@DESKTOP-          MINGW64 ~/Downloads
$ git clone https://private-token:p                    @codehub.devcloud.
c             com/            /Web-Demo.git
Cloning into 'Web-Demo'...

remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 45 (delta 4), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (45/45), 172.03 KiB | 1.04 MiB/s, done.
```

4 迁移到代码托管仓库

概述

将基于 Git 的远程仓库导入代码托管

将本地代码上传到代码托管

4.1 概述

本章主要介绍如何将您的仓库迁移到代码托管服务（CodeArts Repo）中，请结合你目前的仓库存储方式选择以下迁移方案：

- 将基于 Git 的远程仓库导入代码托管。
- 将本地代码上传到代码托管。

4.2 将基于 Git 的远程仓库导入代码托管

背景信息

代码托管服务支持您将基于 Git 的远程仓库导入。

这里基于 Git 的远程仓库指的是 GitHub 这类存储服务中的云端仓库。

方式一：在线导入

这种方式可以直接将您的远程仓库导入到代码托管中，全程在线完成，但导入速度会受到源仓库的网络条件的影响。

1. 在代码托管仓库列表页，单击“新建仓库”，选择“导入仓库”，弹出“填写外部仓库信息”页面。
2. 填写**“源仓库路径”**，设置**“源仓库访问权限”**，如果源仓库是开源库（公仓），请勾选“不需要用户名/密码”，如果源仓库是私有仓库，请勾选“需要用户名/密码”。
3. 单击**“下一步”**，进入“创建仓库”页面，填写仓库基本信息。

4. 单击“确定”按钮，完成仓库导入，跳转到仓库列表页。

详细操作可参考导入外部仓库。

方式二：将 Git 仓库克隆到本地，再关联并推送到代码托管

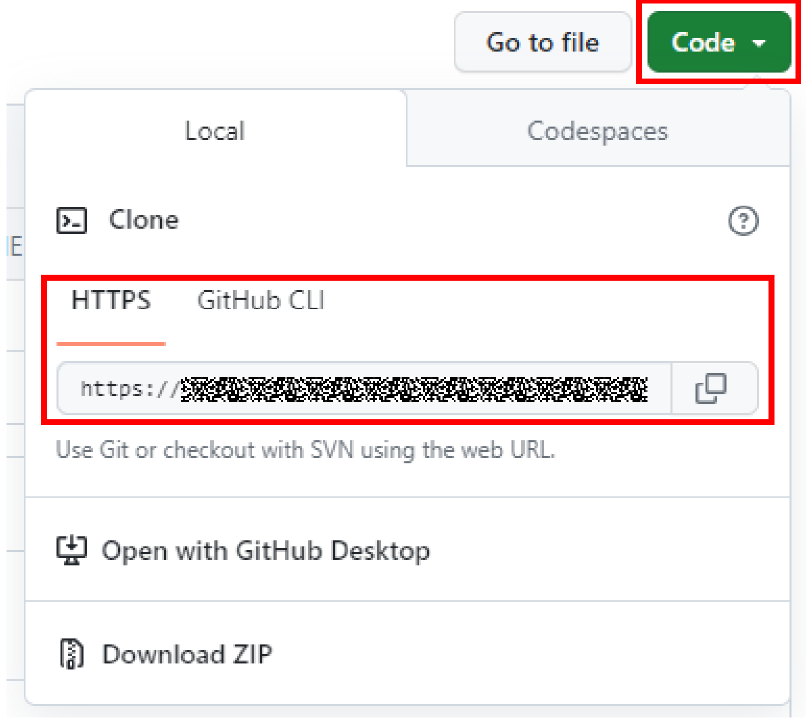
当因为网络等原因，无法直接在线导入时，可以使用以下方法，将远程仓库克隆到本地，再关联、推送到代码托管中。

步骤 1 安装与配置 Git 客户端。

步骤 2 从源仓库地址下载仓库。

下面以 GitHub 为例：

1. 在浏览器中打开并进入 GitHub 代码仓地址。
2. 单击右侧“code”，选择“HTTPS”，单击右侧 图标。



3. 在本地打开 Git Bash 客户端，执行以下命令将仓库克隆到本地计算机，再使用 cd 指令进入仓库目录。

```
git clone --bare 源仓库地址
```

步骤 3 将本地仓库关联并推送到代码托管。

1. 在代码托管服务中新建普通仓库，在“**权限设置**”里，不要勾选“允许生成 README 文件”。
2. 进入 1 中新建的仓库详情页，单击“**克隆/下载**”，单击“**用 SSH 克隆**”或“**用 HTTPS 克隆**”，再单击 按钮，取得仓库地址。

   本示例中以 HTTPS 地址为例。

3. 在本地源代码的根目录下，打开 Git Bash 客户端，执行以下命令将本地的仓库推送到新建的代码托管仓库中。

```
git push --mirror 新建的代码托管仓库的地址
```

指令执行时，会提示您输入代码托管仓库的 HTTPS 账号和密码，正确输入即可。（如何获取 HTTPS 账号、密码？）

```
Administrator@-test MINGW64 ~/Desktop/GitFile/.git (BARE:master)
$ git push --mirror https://

Enumerating objects: 1466, done.
Counting objects: 100% (1466/1466), done.
Delta compression using up to 2 threads
Compressing objects: 100% (1043/1043), done.
Writing objects: 100% (1466/1466), 38.73 MiB | 1.28 MiB/s, done.
Total 1466 (delta 402), reused 1466 (delta 402), pack-reused 0
remote: Resolving deltas: 100% (402/402), done.
To https://

 * [new branch]      master -> master

Administrator@-test MINGW64 ~/Desktop/GitFile/.git (BARE:master)
$ |
```

如果您的源仓库有分支和标签，也会一并推送到代码托管仓库。

----结束

推送成功后，到代码托管仓库内验证迁移是否完整。（如何浏览代码托管仓库？）

4.3 将本地代码上传到代码托管

背景信息

代码托管服务支持您将本地的代码进行 Git 初始化并上传到代码托管仓库。

操作步骤

步骤 1 在代码托管服务中，创建一个空仓库。

- 不选择**“选择 gitignore”**。

- 不勾选 **“允许生成 README 文件”**。

步骤 2 在本地，准备好将要上传的源代码。

- 如果原来没有纳入过任何的版本系统，则在源代码的根目录，执行以下 git 命令（以 Git Bash 为例）：

  a. 初始化 Git 仓库：

```
git init
```

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

  b. 将文件加入版本库：

```
git add *
```

  c. 创建初始提交：

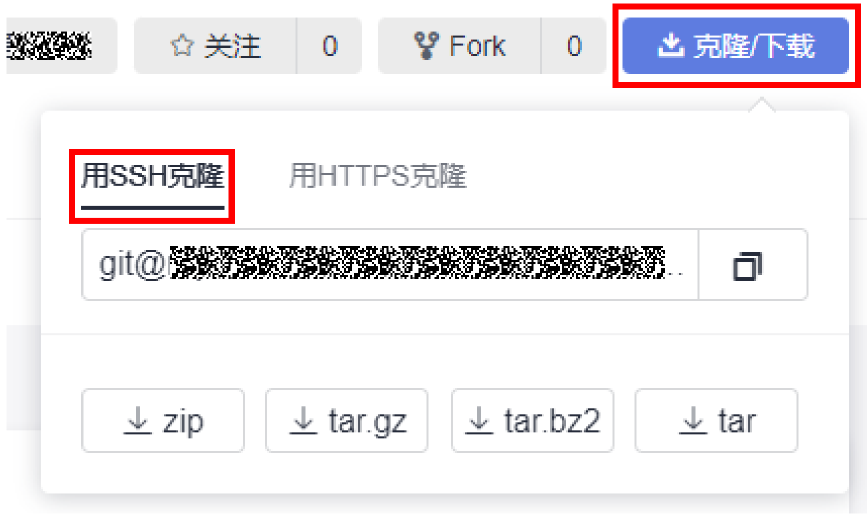
```
git commit -m "init commit"
```

步骤 3 设置本地仓库的远程服务器地址：

- 如果原来从其它地方 clone 的 git 仓库，则添加一个新的 remote，命令行参考如下：

```
git remote add new git@***.***.com:testtransfer/Repo1.git #（new 后面为仓库地址）
```

仓库地址在仓库详情页，获取方式如下图：



- 如果是一个刚刚初始化的仓库，则添加一个名为 origin 的 remote，命令行参考如下：

```
git remote add origin git@***.***.com:testtransfer/Repo1.git   #（origin 后面为仓库地址）
```

步骤 4 推送全部代码到代码托管仓库：

```
git push new master      #（对应步骤 3 的第一种情况）
git push origin master   #（对应步骤 3 的第二种情况）
```

----结束

5 创建代码托管仓库

概述
创建普通仓库
按模板新建仓库
导入外部仓库
Fork 仓库

5.1 概述

目前代码托管服务提供以下几种仓库创建方式：

- 创建普通仓库，适用于本地有仓库，需要将本地仓库同步到代码托管仓库的场景。
- 按模板新建仓库，使用代码托管服务提供的模板创建，适用于本地没有仓库，希望按模板初始化一个仓库的场景。
- 导入外部仓库，用于将其它云端仓库导入到代码托管服务中，也可以将代码托管服务中一个区域的仓库导入到另一个区域（仓库备份），导入后的仓库与源仓库彼此独立。
  - 适用场景一：Gitee、Github 仓库迁移、项目迁移到代码托管服务。
  - 适用场景二：使用软件开发生产线的用户，希望将项目迁移到其它区域。
- Fork 仓库基于目前已有的代码托管仓库复制，复制出的仓库可以将修改内容合并回源仓库。
  - 适用场景一：希望基于历史项目开展新项目，又不想破坏历史项目仓库结构。
  - 适用场景二：组织内项目开源。

须知

- 仓库容量包含 LFS 使用量，单个仓库的容量不能超出 2GB，超出时仓库将不能正常使用，且仓库无法扩容。
- 当仓库超出容量上限时，仓库处于冻结状态，这种情况建议您删除该仓库，在本地控制好容量之后重新推送即可。

仓库常用设置

- 仓库设置
- IP 白名单
- 了解更多

5.2 创建普通仓库

适用于本地有仓库，需要将本地仓库同步到代码托管仓库的场景。在代码托管服务控制台创建空仓库的步骤如下：

步骤 1 登录控制台，单击页面左上角☰，搜索“代码托管”，单击“立即使用”，进入代码托管服务仓库列表页。

步骤 2 单击“新建仓库”按钮，跳转到“归属项目和仓库类型”页面，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

说明

- 代码仓库必须挂到项目下，通过项目维度查看仓库看板。

步骤 3 仓库类型默认选择“普通仓库”。

步骤 4 单击“下一步”按钮，跳转到“基本信息”页面，填写仓库基本信息。

表5-1 新建“普通仓库”参数说明

字段名称	是否必填	备注说明
代码仓库名称	是	请以大小写字母、数字、下划线开头，可包含大小写字母、数字、中划线、下划线、英文句点，但不能以.git、.atom 或.结尾，仓库名称至少 2 个字符，总长度不超过 255 个字符。
描述	否	为您的仓库填写描述，限制 2000 个字符。
选择 gitignore	否	会根据您的选择生成.gitignore 文件。
类型	否	为您的仓库内容勾选相应的仓库类型。

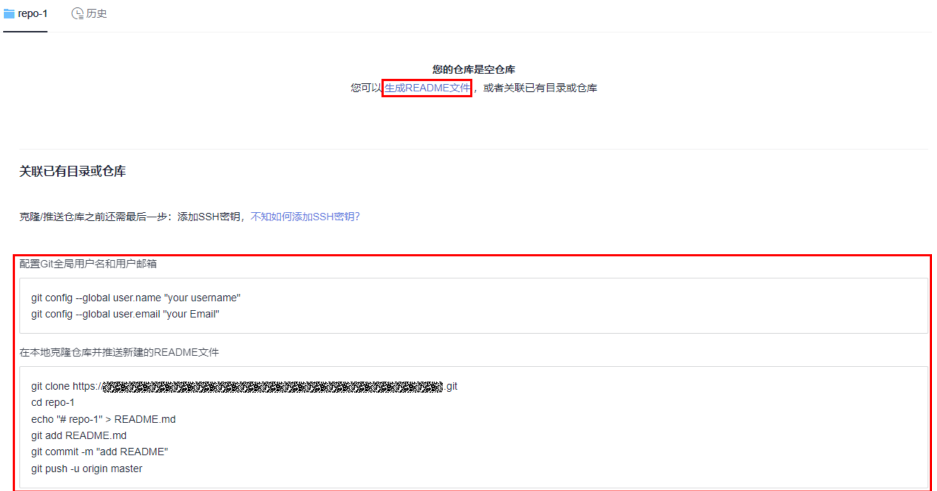
字段名称	是否必填	备注说明
设置	否	可选择： • **允许生成 README 文件。** 您可以通过编辑 README 文件，记录项目的架构、编写目的等信息，相当于对整个仓库的一种注释。 • **自动创建代码检查任务（免费）。** 仓库创建完成后在代码检查任务列表中，可看到对应仓库的检查任务。 **说明** • "将项目开发人员自动添加为该仓库成员" 功能下线，不再自动将项目经理和开发人员添加为仓库成员。仅默认将项目创建者、项目管理员加入仓库。
是否公开	是	可选择： • 私有(仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码)。 • 公开只读(仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中)
添加开源许可证	否	如果仓库设置为公开，可下拉选择已有的许可证。

步骤 5 单击“确定”按钮，完成仓库新建，跳转到仓库列表。

----结束

如何关联已有目录或仓库

如果在新建普通仓库时，没有勾选**“允许生成 README 文件”**，您可以单击仓库**“代码”**页签中的**“生成 README 文件”**，生成一个新的 README 文件，或者关联已有目录或仓库，具体操作如下。



前提条件

- 以下命令行操作需要您在 Git 客户端执行，安装 Git 客户端并配置 Git 全局用户名和用户邮箱，详情请参考 Git 客户端安装配置。
- 设置 SSH 密钥，详情请参考 SSH 密钥。

操作步骤

说明

以下命令已在您新建仓库中自动生成，您可以在仓库“**代码**”页签界面中复制获取。

步骤 1 在本地克隆仓库并推送新建的 README 文件。

```
git clone 您的 HTTPS 下载链接
cd taskecho "# 仓库名" > README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

步骤 2 关联已有代码目录到仓库。

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

步骤 3 关联已有的 Git 仓库。

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

----结束

5.3 按模板新建仓库

使用代码托管服务提供的模板新建仓库，适用于您新建一个仓库，并希望按模板初始化一个仓库的场景。

操作步骤

步骤1 登录控制台，单击页面左上角☰，搜索“代码托管”，单击立即使用，进入代码托管服务仓库列表页。

步骤2 单击“新建仓库”按钮，跳转到“归属项目和仓库类型”页面，在“归属项目”下拉框中选择选择已有的项目或者“新建项目”。

说明

- 代码仓库必须挂到项目下，通过项目维度查看仓库看板。

步骤3 仓库类型选择“模板仓库”。

步骤4 单击“下一步”按钮，跳转到**“按模板新建”**页面。

步骤5 选择想要的模板，支持在搜索栏中模糊查询。

步骤6 单击“下一步”按钮，进入**“基本信息”**页面，填写仓库基本信息。

表5-2 按模板新建仓库的参数说明

字段名称	是否必填	备注说明
代码仓库名称	是	请以大小写字母、数字、下划线开头，可包含大小写字母、数字、中划线、下划线、英文句点，但不能以.git、.atom 或.结尾，仓库名称至少 2 个字符，总长度不超过 255 个字符。
描述	否	为您的仓库填写描述，限制 2000 个字符。
设置	否	• **自动创建代码检查任务（免费）。** 仓库创建完成后在代码检查任务列表中，可看到对应仓库的检查任务。 **说明** • "将项目开发人员自动添加为该仓库成员" 功能下线，不再自动将项目经理和开发人员添加为仓库成员。仅默认将项目创建者、项目管理员加入仓库。 • 如果模板含“自动创建流水线”，则不涉及该参数。 • 项目成员自动添加到仓库的功能，可基于成员组的成员动态同步能力实现。
是否公开	是	可选择： • 私有(仓库仅对仓库成员可见，仓库成员可访问仓库或者提交

字段名称	是否必填	备注说明
		代码)。 • 公开只读(仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中)
添加开源许可证	否	如果仓库设置为公开，可下拉选择已有的许可证。

步骤 7 单击“确定”按钮，完成仓库新建。

----结束

说明

按模板新建时，仓库的类型会根据选择的模板的仓库类型自动配置。

按模板新建的仓库将包含模板预置的仓库文件结构。

自动创建流水线

代码仓库在按模板新建代码仓库时，可配置**“自动创建流水线”**，请注意部署服务中使用的主机需修改为自己的真实环境，才能够成功执行流水线。

步骤 1 进入代码托管服务仓库列表页。

步骤 2 单击“新建仓库”按钮，跳转到“归属项目和仓库类型”页面。

步骤 3 “归属项目”下拉框单选择已有的项目或者“新建项目”。

说明

- 仓库必须存在项目下。
- 如果在项目内新建仓库则默认选择该项目，页面会隐去“归属项目”这个字段。

步骤 4 仓库类型选择“模板仓库”。

步骤 5 单击“下一步”按钮，跳转到**“选择仓库模板”**页面。

步骤 6 在左侧导航栏**“自动创建流水线”**选项，选择**“是”**，过滤出可以自动创建流水线的模板。

自动创建流水线

所有

是

否

步骤 7 根据需要选择模板，单击**“下一步”**，填写仓库基本信息，然后单击**“确定”**。

步骤 8 创建完成后，进入流水线列表页面，即可看到通过该仓库自动创建好的流水线。

----结束

5.4 导入外部仓库

用于将其它云端仓库导入到代码托管服务中，也可以将代码托管服务中一个区域的仓库导入到另一个区域（仓库备份），导入后的仓库与源仓库彼此独立。

步骤 1 登录控制台，单击页面左上角☰，搜索“代码托管”，单击立即使用，进入代码托管服务仓库列表页。

步骤 2 单击“新建仓库”按钮，跳转到“归属项目和仓库类型”页面，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

说明

- 代码仓库必须挂到项目下，通过项目维度查看仓库看板。
- 如果在项目内新建仓库则默认选择该项目。

步骤 3 仓库类型选择“导入仓库”，跳转到“导入外部仓库”页面。

须知

- 导入时，源仓端口限制为：80、443，以及大于 1024 的端口。
- 目前完全支持的源站地址包括：Git。如果使用其它源站地址导入失败，请联系技术支持确认源站白名单。

步骤 4 单击“下一步”按钮，进入“创建仓库”页面，填写仓库基本信息，参数填写请参考表 5-2 和表 5-3。

表5-3 “同步仓库设置”参数填写说明

字段名称	是否必填	备注说明
分支设置	是	可选择同步源仓库的**默认分支**或**全部分支**。
增加定时同步	否	同步分为手动同步和定时同步，同步分支配置后不可更改。如果勾选**“增加定时同步”**功能： • 每天自动从源仓库导入仓库的默认分支。 • 仓库将成为只读镜像仓库，不能写入，并且只同步当前创建仓库的默认分支对应的第三方仓库的分支。

说明

同步分为手动同步和定时同步，同步分支配置后不可更改，可参考同步仓库。

步骤 5 单击“确定”按钮，完成仓库导入，跳转到仓库列表页。

----结束

说明

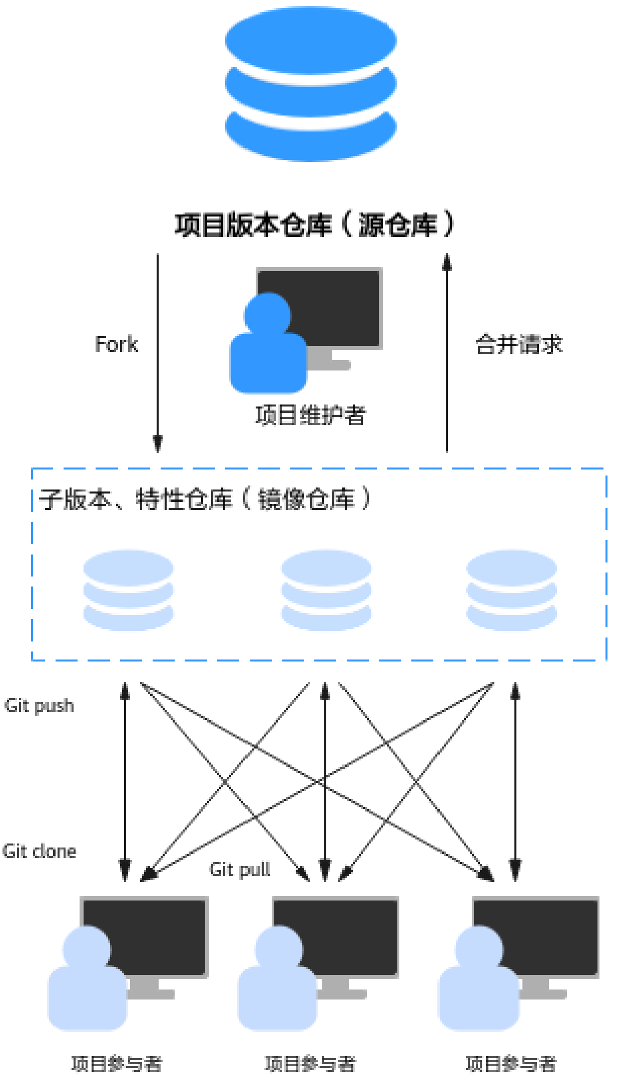
- 仓库导入超时时间为 30min。如果导入超时，建议使用客户端 clone/push 来处理。
- 导入的内容中不包含 Git LFS 对象。
- 该功能需要保证被导入的仓库域名和服务节点网络连通。

5.5 Fork 仓库

Fork 的应用场景

Fork 可基于某个仓库镜像出一个相同的仓库，并能将镜像仓库中的修改请求合并回源仓库，当合并未发生时，镜像仓、源仓库的修改都不会对彼此产生影响。

可见 Fork 特别适用于大型项目包含众多子项目时的开发场景，如下图所示，复杂的开发过程都只发生在镜像仓中，并不会影响到项目版本仓库(源仓库)，只有确认完成的新特性才会请求合并回项目版本仓库，所以可以认为 Fork 是一种团队协作模式。



Fork 仓库与导入外部仓库的区别

Fork 仓库与导入外部仓库的本质都是仓库的复制，其主要区别在于操作后源仓库与复制出仓库的联动关系不同，详细如下：

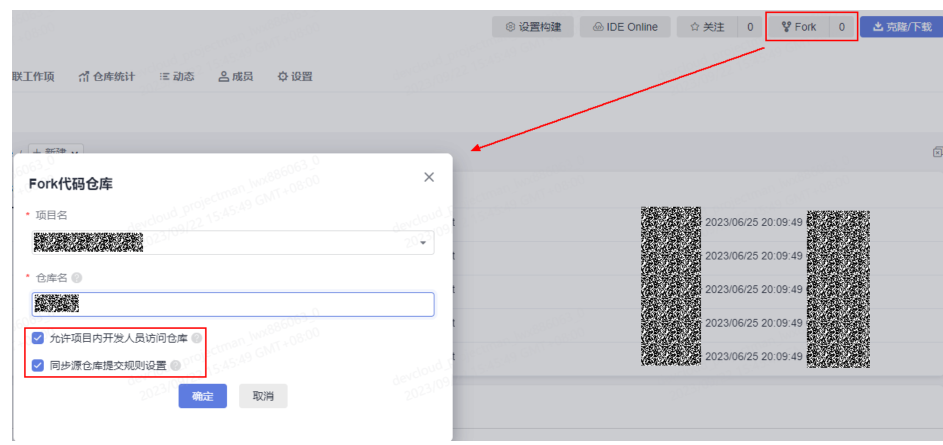
- **Fork 仓库**
  - Fork 仅应用于代码托管平台内的仓库间复制。
  - Fork 仓库时，会基于源仓库的当前版本复制出一个内容相同的副本仓库，您在副本仓库的修改，可以申请合并（可以理解为一种跨仓库的分支合并）回源仓库，但副本仓库不能再获取源仓库的更新。
- **导入外部仓库**
  - 导入外部仓库不仅可以将其它版本管理平台的仓库进行导入（主要针对基于Git 存储的托管平台），也可以导入代码托管服务自己的仓库。
  - 导入外部仓库时，也会基于源仓库的当前版本复制出一个内容相同的副本仓库，所不同的是，副本仓库不能向源仓库提交合并申请，但是副本仓库可以随时拉取源仓库的默认分支，以起到获取最新版本的作用。

如何 Fork 仓库

步骤 1 进入代码托管服务仓库列表页。

步骤 2 单击目标仓库名称，进入目标仓库。

步骤 3 单击页面右上角的“Fork”按钮，弹出**“Fork 代码仓库”**窗口，选择目标项目、填写仓库名称以及勾选是否**“同步源仓库提交规则设置”**。



步骤 4 单击“确定”按钮，即可完成 Fork 仓库操作。

----结束

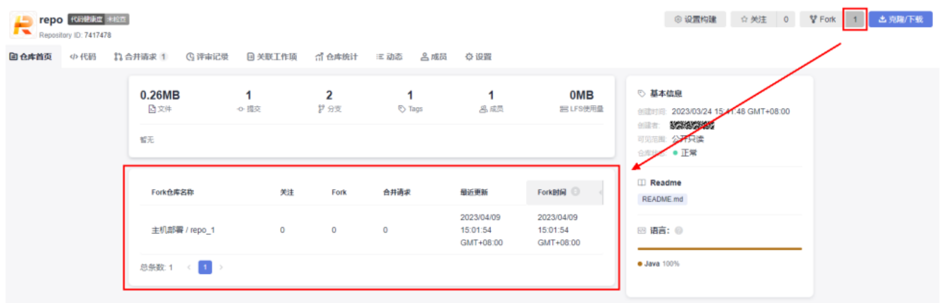
查看 Fork 仓库列表

步骤 1 进入代码托管服务仓库列表页。

步骤 2 单击源仓库名称，进入源仓库。

步骤 3 如下图所示，单击页面右上角“Fork”旁的按钮，可查看 Fork 仓库列表。

单击 Fork 仓库的名称可进入该仓库。



----结束

如何将 Fork 仓库中的修改合入源仓库

步骤 1 进入代码托管服务仓库列表页。

步骤 2 单击 Fork 仓库名称，进入 Fork 仓库。

步骤 3 单击**“新建合并请求”**，切换到合并请求页签。

步骤 4 单击“新建”，弹出**“新建合并请求”**页面。

“源分支”为本仓库作为请求合并的分支。

“目标分支”为该仓库的源仓库被合入的分支。

步骤 5 单击“下一步”，进入到新建合并请求页面，其后面的操作流程与仓库内部的新建合并请求完全一致，请参考新建合并请求。

----结束

说明

跨仓库的合并请求隶属于源仓库，只能在源仓库的“合并请求”页签中看到，在 Fork 仓库（请求发起方仓库）中看不到，因此选择的检视人、评审人、审核人及合并人均为源仓库的人员。

6 关联代码托管仓库

如果开发者之前将项目文件存放在本地计算机，在开始使用代码托管服务时，则需要将本地项目文件初始化成 Git 仓库，并将其与代码托管服务提供的仓库进行关联。

前置条件

请确保已安装 Git 客户端，并且 Git 客户端的 HTTPS 密钥已绑定至代码托管服务。

操作步骤

步骤 1 新建代码托管仓库。

如果根据您本地代码库选择 gitignore，会帮助你将一些非开发文件屏蔽掉而不受 Git 纳管。

步骤 2 将本地仓库初始化成 Git 仓库。

在您的仓库中打开 Git Bash 客户端，执行以下命令。

```
git init
```

初始化成功如下图，此时当前文件夹已经是本地 Git 仓库了。

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

步骤 3 绑定代码托管仓库。

1. 进入代码托管仓库，获取仓库地址。
2. 在本地使用 remote 命令，将本地仓库与代码托管仓库进行绑定。

   ```
   git remote add 仓库别名 仓库地址
   ```

   示例为：

   ```
   git remote add origin git@*****/java-remote.git  #复制使用时 注意换成您自己的仓库地址
   ```

   一般用 origin 作为仓库别名，因为当您从远程仓库 clone 到本地时，默认产生的别名就是 origin，您也可以使用任意别名。

   如果提示仓库名重复，更换一个即可。

   无回显即为绑定成功。

步骤 4 将代码托管仓库 master 分支拉取到本地库。

此步骤主要是避免冲突。

```
git fetch origin master  #复制使用时 注意需要将 origin 替换为您仓库的别名
```

步骤 5 将本地代码文件提交到 master 分支。

依次执行：

```
git add .
git commit -m "您的提交备注"
```

下图为成功的执行。

```
Administrator@ecstest-paas-lwx6      MINGW64 ~/Desktop/liu'Code/java (master)
$ git add .

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

步骤 6 将本地 master 分支与代码托管仓库 master 分支进行绑定。

```
git branch --set-upstream-to=origin/master master  #复制使用时 注意是否需要将 origin 替换为您仓库的别名
```

成功执行如下图所示，会提示您已经绑定成功。

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

步骤 7 合并代码托管仓库与本地仓库的文件，并存储在本地。

```
git pull --rebase origin master  #复制使用时 注意是否需要将 origin 替换为您仓库的别名
```

成功执行如下图所示，提示您已经将合并后的仓库放在工作区与版本库。

```
Administrator@ecstest-paas-lwx      MINGW64 ~/Desktop/liu'Code/java (master)
$ git pull --rebase origin master
From [illegible]est00001/java-remote
 * branch            master     -> FETCH_HEAD
Successfully rebased and updated refs/heads/master.
```

步骤 8 将本地仓库推送覆盖代码托管仓库。

因为之前已经进行了绑定，直接 push 即可。

```
git push
```

成功后，再直接拉取 pull，验证代码托管仓库与本地仓库版本相同，如下图。

```
Administrator@ecstest-paas- MINGW64 ~/Desktop/liu'Code/java (master)
$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 427 bytes | 427.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
To .git
   0ca3cd3..bafb729  master -> master

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

----结束

7 克隆/下载代码托管仓库到本地

概述
使用 SSH 协议克隆代码托管仓库到本地
使用 HTTPS 协议克隆代码托管仓库到本地
从浏览器下载代码包

7.1 概述

除了文件管理外，基于 Git 的代码托管服务，还支持将仓库文件下载到本地进行文件的操作。

首次将仓库克隆/下载到本地的方式主要分为以下三种：

- 使用 SSH 协议克隆代码托管仓库到本地。
- 使用 HTTPS 协议克隆代码托管仓库到本地。
- 从浏览器下载代码包。

7.2 使用 SSH 协议克隆代码托管仓库到本地

前提条件

操作前应确保您的网络可以访问代码托管服务，请参考验证网络连通性。

使用 SSH 协议在 Git Bash 客户端克隆代码

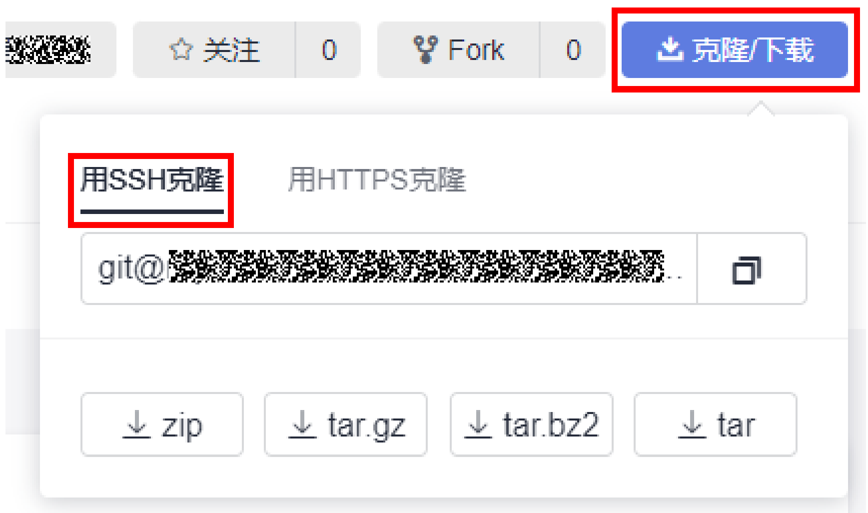
本节内容描述如何使用 Git Bash 客户端克隆代码托管服务的仓库到本地环境中。

步骤 1 下载并安装 Git Bash 客户端。

步骤 2 设置 SSH 密钥。

步骤 3 获取仓库地址。（没有仓库？如何新建仓库？）

在仓库主页中，单击“克隆/下载”按钮，获取 SSH 地址，通过这个地址，可以在本地计算机连接代码托管仓库。



步骤 4 打开 Git Bash 客户端。

在本地计算机上新建一个文件夹用于存放代码仓库，在空白处单击鼠标右键，打开 Git Bash 客户端。

说明

克隆仓库时会自动初始化，无需执行 init 命令。

步骤 5 输入如下命令，克隆代码托管仓库。

```
git clone 仓库地址
```

命令中“仓库地址”即步骤 3 中获取的 SSH 地址。

如果您是第一次克隆仓库，会询问您是否信任远程仓库，输入“yes”即可。

执行成功后，您会看到多出一个与您在代码托管服务新建的仓库同名的文件夹，并且其中有一个隐藏的.git 文件夹，则说明克隆仓库成功。

步骤 6 此时您位于仓库上层目录，执行如下命令，进入仓库目录。

```
cd 仓库名称
```

进入仓库目录，可以看到此时 Git 默认为您定位到 master 分支。

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

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

----结束

说明

客户端在 git clone 代码仓库时失败的原因排查：

- 确保您的网络可以访问代码托管服务。

请在 git 客户端使用如下测试命令验证网络连通性。

```
ping xxx.com
```

如果返回内容含有“Could not resolve hostname **********.com: Name or service not known”，则您的网络被限制，无法访问代码托管服务，请求助您本地所属网络管理员。

- 请检查建立的 SSH 密钥配对关系，必要时重新生成密钥并到代码托管控制台进行配置。
- 只有开启 IP 白名单的机器才可以在 Git 客户端克隆。

使用 SSH 协议在 TortoiseGit 客户端克隆代码

本节内容描述如何使用 TortoiseGit 客户端克隆代码托管服务的仓库到本地环境中。

步骤 1 下载并安装 TortoiseGit 客户端。

步骤 2 获取仓库地址。（没有仓库？如何新建仓库？）

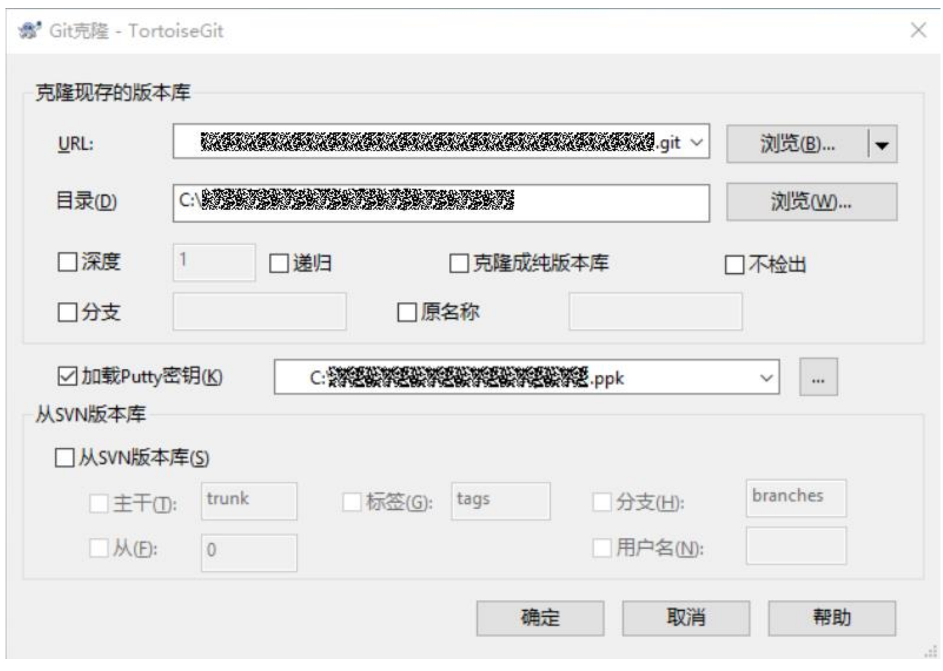
在仓库主页中，单击“克隆/下载”按钮，获取 SSH 地址，通过这个地址，可以在本地计算机连接代码托管仓库。

说明

您可在代码托管服务仓库列表中“仓库地址”下获取 SSH 地址。

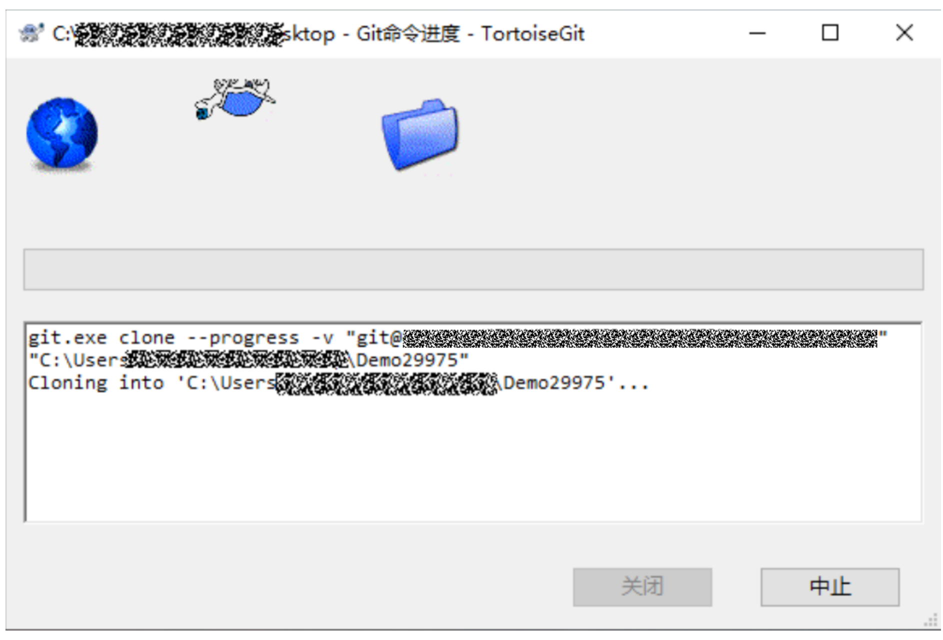
步骤 3 进入您的本地仓库目录下，右键选择“Git 克隆”菜单选项。

步骤 4 在弹出的窗口中将上述复制的 SSH 地址粘贴到 URL 输入框中，勾选“加载 Putty 密钥”并选择私钥文件，最后单击“确定”，如下图所示。



步骤 5 单击“确定”之后即开始克隆仓库，如果您是第一次克隆 TortoiseGit 客户端会询问您是否信任远程仓库，单击“是”即可。

步骤 6 克隆用时受仓库大小影响，克隆的动作如下图所示。



----结束

使用 SSH 协议在 Linux 或 Mac 中克隆仓库

在配置完Linux Git 客户端或Mac Git 客户端环境后，Linux 或 Mac 上 Git 客户端的克隆操作和使用 SSH 协议在 Git Bash 客户端克隆代码的操作完全一样。

7.3 使用 HTTPS 协议克隆代码托管仓库到本地

使用 HTTPS 协议从 Git Bash 客户端克隆代码

本节内容描述如何使用 Git Bash 客户端克隆代码托管服务的仓库到本地环境中。

须知

本产品中 HTTPS 协议所支持的最大单次推包大小为 200MB，需传输大于 200MB 时，请使用 SSH 方式。

步骤 1 下载并安装 Git Bash 客户端。

步骤 2 设置 HTTPS 密码。

步骤 3 进入代码托管仓库列表页，单击仓库列表中的“仓库名”进入仓库详情页，单击右侧导航栏“**克隆/下载**”按钮，单击**“用 HTTPS 克隆”**，复制访问方式中的 HTTPS 链接，如下图所示。

步骤 4 打开 Git Bash 客户端进入您的目录下，输入以下命令进行仓库克隆，其中第一次克隆需要您填写用户名（账号名/用户名）和 HTTPS 密码。

```
git clone 您的 HTTPS 下载链接
```

步骤 5 用户名（账号名/用户名）和 HTTPS 密码输入完成后，即可完成仓库克隆。

步骤 6 此时您位于仓库上层目录，执行如下命令，进入仓库目录。

```
cd 仓库名称
```

进入仓库目录，可以看到此时 Git 默认为您定位到 master 分支。

----结束

说明

客户端在 git clone 代码仓库时失败的原因排查：

- 确保您的网络可以访问代码托管服务。

  请在 git 客户端使用如下测试命令验证网络连通性。

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

  如果返回内容含有“Could not resolve hostname **********.com: Name or service not known”，则您的网络被限制，无法访问代码托管服务，请求助您本地所属网络管理员。
- 请确认 HTTPS 密码，必要时重新设置密码。
- 只有开启 IP 白名单的机器才可以在 Git 客户端克隆。

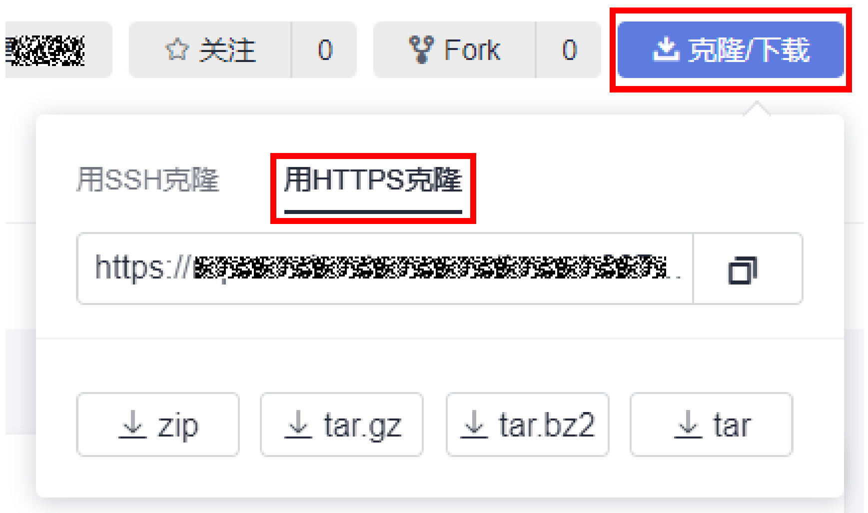
使用 HTTPS 协议在 TortoiseGit 客户端克隆代码

本节内容描述如何使用 TortoiseGit 客户端克隆代码托管服务的仓库到本地环境中。

步骤 1 下载并安装 TortoiseGit 客户端。

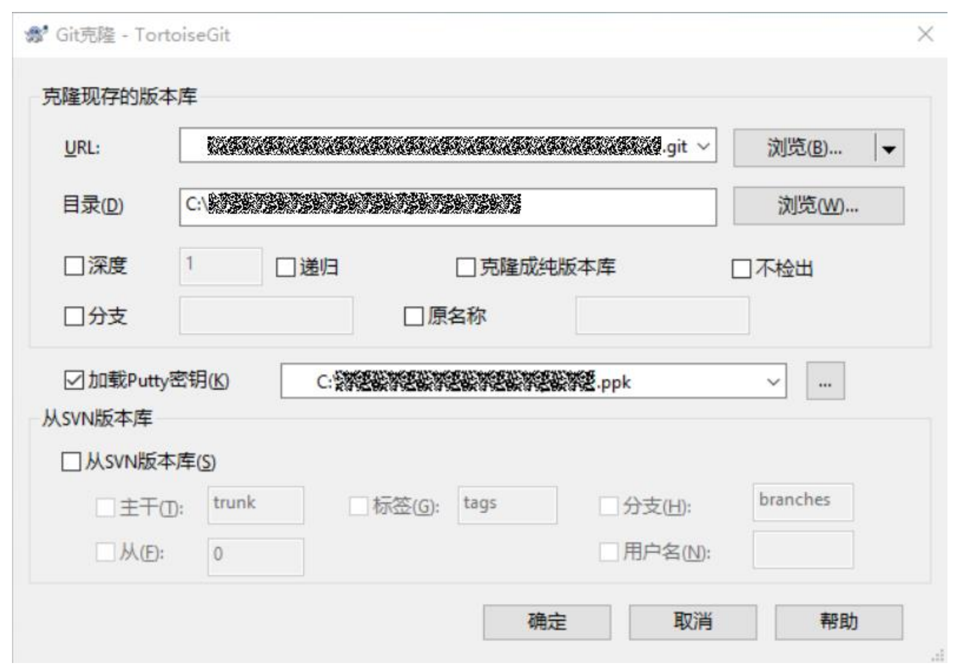
步骤 2 设置 HTTPS 密码。

步骤 3 进入代码托管仓库列表页，单击仓库列表中的“仓库名”进入仓库详情页，单击右侧导航栏“**克隆/下载**”按钮，单击**“用 HTTPS 克隆”**，复制访问方式中的 HTTPS 链接，如下图所示。



步骤 4 进入您的目录下，右键在弹出的菜单选项中选择“Git 克隆”，如下图所示。

步骤 5 在弹出的窗口中将上述复制的 HTTPS 地址粘贴到 URL 输入框中，然后单击“确定”，如下图所示。



步骤 6 单击“确定”之后即开始克隆仓库，如果您是第一次进行克隆 TortoiseGit 客户端会要求您输入用户名和 HTTPS 密码。

步骤 7 开始克隆之后稍后即可完成。

----结束

使用 HTTPS 协议在 Linux 或 Mac 中克隆仓库

在配置完Linux Git 客户端或Mac Git 客户端环境后，Linux 或 Mac 上 Git 客户端的克隆操作和使用 HTTPS 协议从 Git Bash 客户端克隆代码的操作完全一样。

7.4 从浏览器下载代码包

除了克隆以外，代码托管服务同时支持将仓库代码打包下载到本地。

使用直接下载方式获取的代码仓库文件与代码托管仓库没有关联关系，不能将内容回推代码托管仓库。

其操作步骤如下：

步骤 1 进入代码托管服务仓库列表页。

步骤 2 进入您的仓库。（如何新建仓库？）

步骤 3 单击“克隆/下载”按钮，在弹出的窗口中单击需要的代码包类型即可直接在浏览器中下载。

----结束

说明

- 如果仓库设置 IP 白名单，则只有 IP 白名单内的机器才可以在界面下载仓库源码，如果仓库没有设置 IP 白名单，则均可在界面下载仓库源码。
- 目前支持的下载包格式有：zip、tar.gz、tar.bz2、tar。
- 下载的代码包是对应的代码托管仓库的 master 分支内容。

8 使用代码托管仓库

查看仓库列表
查看仓库详情
查看仓库首页
管理代码文件
管理合并请求
查看仓库的评审记录
查看关联工作项
查看仓库的统计信息
查看仓库的动态
管理仓库成员

8.1 查看仓库列表

仓库列表是您使用代码托管服务的入口。

在这里您可以完成新建仓库、配置仓库、获取仓库地址等操作。

- 个人首页：支持查看**“我关注的”**、**“我参与的”**及**“我创建的”**等分类的仓库，单击目标仓库名称可进入该仓库。支持查看**“我创建的”**、**“待我合并的”**、**“待我检视的”**及**“待我审核的”**等分类的合并请求，单击目标合并请求名称可进入该合并请求。

说明

如果您进入某个项目下的代码托管服务，则该功能将隐藏。

- 新建仓库：支持“普通新建”、“按模板新建”、“导入外部仓库”等新建方式。
- 仓库筛选：支持**“所有仓库”**、**“未锁定仓库”**、**“已锁定仓库”**等筛选方式，配置仓库的锁定请参考锁定仓库。
- 关注按钮 ☆ ：可以切换仓库的关注状态。
- 关联工作项：所关联工作项的列表，其可设置与需求管理中工作项的联动，提升效率。
- 成员管理：也可以单独调整某个成员的权限。
- 删除仓库：单击 ··· 图标，在展开选项中，单击“删除仓库”按钮，按提示输入“YES”或者“yes”，单击“确认”按钮，即完成仓库删除。

说明

此操作无法撤销并且已删除的仓库无法恢复，请再三确认！

8.2 查看仓库详情

在仓库列表中单击仓库名称可进入该仓库的详情页面，代码托管服务提供了丰富的控制台操作，详情如下。

表8-1 页签说明

功能页签	功能说明
仓库首页	用于展示仓库的容量、提交次数、分支数量、标签数量、成员数量、LFS 使用量、创建时间、创建者、可见范围、仓库状态、readme 文件、语言、语言占比等信息。
代码	• 文件列表：支持新建文件、新建目录、新建子模块、上传文件、在线修改文件、修改追溯和查看提交历史等操作。 • 提交：支持查看提交记录及仓库网络图。

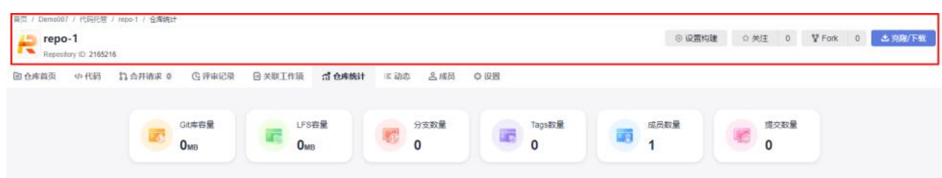
功能页签	功能说明
	● 分支：支持在控制台管理分支。 ● Tags：支持在控制台管理标签。 ● 对比：支持通过对比查看分支之间或标签版本之间发生的代码变化。
合并请求	支持在控制台管理分支的合并请求。
评审记录	支持查看合并请求的评审记录与 Commit 的评审记录。
关联工作项	所关联工作项的列表，其可设置与需求管理中工作项的联动，提升效率。
仓库统计	仓库提交记录的可视化图表，主要展现了代码贡献度等信息。
动态	支持查看仓库动态信息。
成员	支持对成员的管理，具体信息如下： ● **成员列表”**、**“成员组列表”** **“待审核”**和**“添加成员”**位于仓库详情的“**成员**”页签下。 – **“成员列表”**用于展示仓库中所有成员**“用户名”**、**“用户来源”**、“**项目角色**”、“**仓库角色**”和“**操作**”。 – “**成员组列表**”用于展示仓库所有成员组的**“成员组名称”**、**“成员数量”**、**“描述”**和**“操作”**。 – “**待审核**”用于展示即将加入仓库中**待审核**成员，包括**“用户名”**、**“用户昵称”**、**“企业用户”**、**“项目角色”**、**“仓库角色”**和**“操作”**。“**待审核**”成员可被拥有“**添加成员**”权限的人设置为“**同意**”或“**拒绝**”。 – “**添加成员**”用于仓库**添加成员**或**添加成员组**。
设置	此仓库的设置入口。仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”。

另外仓库详情页框架上还提供以下功能的快捷入口：

- 设置构建：新建编译构建任务入口。
- 关注：单击可关注该仓库，关注的仓库会在仓库列表置顶。
- Fork：会显示目前仓库有几个 Fork 出的仓库，单击弹出“Fork 代码仓库”页面。
- 克隆/下载：可获取仓库的 SSH 地址、HTTPS 地址，也可以直接下载代码压缩包。

说明

- 代码托管**“吸顶”**功能，当用户的仓库界面长度大于窗口长度，向下滑动鼠标滚轮后，仓库页签置顶，下图中红框位置被折叠，便于查看仓库信息，向上滑动鼠标滚轮后，界面恢复。



- 代码检查状态显示规则：
- 当用户有代码检查权限时，仓库名称后显示代码检查状态。
- 当用户无代码检查权限时，仓库名称后不显示代码检查状态。
- 构建状态显示规则：
- 当用户无构建权限时

  仓库页面上方只显示“设置构建”按钮。
- 当用户有构建权限时

  如果没有设置构建，则仓库页面上方显示“设置构建”按钮。

  如果设置了构建，则会根据构建任务执行情况在仓库页面上方显示的状态有：“运行构建任务”、“构建进行中”、“构建执行失败”、“构建执行成功”。

8.3 查看仓库首页

“仓库首页”页签用于展示仓库的基础情况。

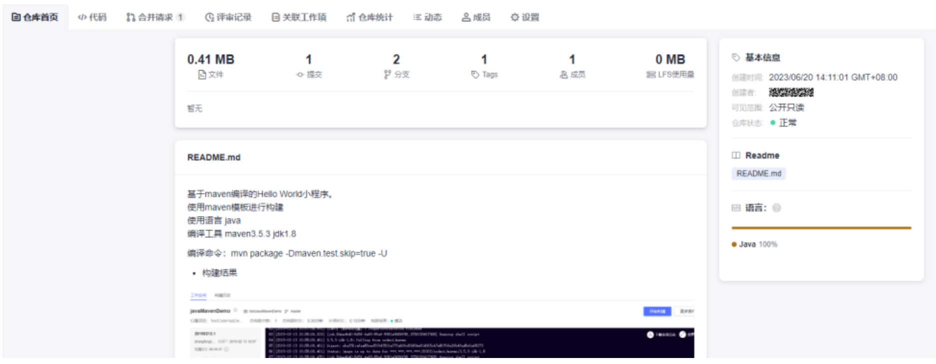


表8-2 字段说明

字段	说明
仓库容量	截止当前仓库的容量。 **说明** ● 仓库容量包含LFS使用量，单个仓库的容量不能超出2GB，超出时仓库将不能正常使用，且仓库无法扩容。 ● 当仓库超出容量上限时，仓库处于冻结状态，这种情况建议您删除该仓库，在本地控制好容量之后重新推送即可。
仓库提交次数	统计仓库的提交次数，单击图标可跳转到**“代码”**页签下的**“提交”**，查看提交详情。
仓库分支	统计仓库的分支数量，单击图标可跳转到**“代码”**页签下的**“分支”**，进行分支管理。

字段	说明
仓库标签	统计仓库的标签数量，单击图标可跳转到**“代码”**页签下的**“Tags”**，进行标签管理。
仓库成员	统计仓库的成员数量，单击图标可跳转到**“成员”**页签，进行成员管理。
LFS 使用量	统计仓库的 LFS 使用量。
仓库简介	显示创建仓库时填写的仓库描述信息。
Readme 文件预览	支持预览 Readme 文件。如果仓库中无 Readme 文件，可单击**“新建 Readme”**进行创建。 **文件名称**：默认为 README.md。 **格式**：可选择以下两种类型。 ● text：代表文本数据或文本字符串。 ● base64：Base64 是一种基于 64 个可打印字符来表示二进制数据的方法。 **内容**：支持自定义。 ● 当格式为 text 时，填写的内容应为普通文本。 ● 当格式为 base64 时，填写的内容应为 Base64 编码并通过编码校验。 **提交信息**：填写此文件或文件夹的提交信息，可自定义。
基本信息	显示仓库的创建时间、创建者、仓库的可见范围及仓库状态。
Readme	显示仓库的 Readme 文件，单击文件名称可跳转到**“代码”**页签

字段	说明
	下查看文件内容。
语言	显示仓库各语言的占比，统计单位为文件大小。

8.4 管理代码文件

8.4.1 文件管理

代码托管服务提供了对文件的编辑、追溯、对比等功能。

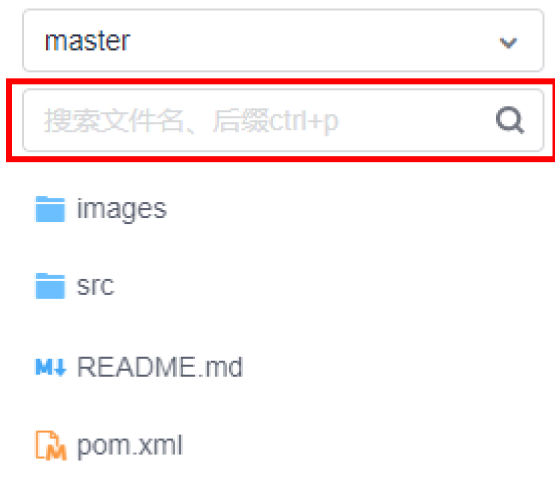
当您进入仓库详情控制台中，系统将为您定位到“代码”页签下的**“文件”**子页签，在这里您可以切换到不同的分支、标签，查看对应版本中文件的情况，如下图，左侧为主分支下的文件列表，右侧为可切换的页签：仓库名称（分支或标签版本的文件详情）、历史（分支或标签版本）。

文件列表

文件列表位于该仓库**“文件”**页签的左侧，文件列表包含以下功能：

1. 单击分支名称，切换分支、标签：切换后的分支、标签后会显示对应版本的文件目录。

2. 单击 检索图标：单击弹出搜索页面，可对文件列表进行搜索定位。



3. 单击 图标，可扩展功能如下：

须知

新建文件/重命名文件/新建目录/新建子模块支持创建多级目录，多级目录以/分隔，如'java/com'。

- **新建文件**。

  在代码托管仓库控制台新建文件，等同于“文件的新建 → add → commit → push”操作，会生成提交记录。

  在**“新建文件”**页面，填写文件名称，选择目标模板类型，选择编码类型，填写文件内容及提交信息后，单击**“确定”**完成新文件的创建。

说明

“提交信息”字段相当于 git commit 中的-m 消息，可以用于查看关联工作项。

- **新建目录**。

  在代码托管仓库控制台新建目录，其实是一次“文件夹结构的新建 → add → commit→ push”，会生成提交记录。

  新建目录后在目录的最深层会默认新建一个.gitkeep 文件，这是因为 Git 不允许提交空文件夹。

  在 **“新建目录”** 页面，填写目录名称，及提交信息后，单击 **“确定”** 完成新目录的创建。

- 新建子模块。
- **上传文件**。

  在代码托管仓库控制台上传文件，其实是一次“文件的新建 → add → commit→ push”，会生成提交记录。

  在 **“上传文件”** 页面，选择上传的目标文件，填写提交信息后，单击 **“确定”** 完成新文件的上传。

说明

鼠标停留在文件夹名称处，单击显示的 ⋮ 图标，可该文件夹下进行以上操作。

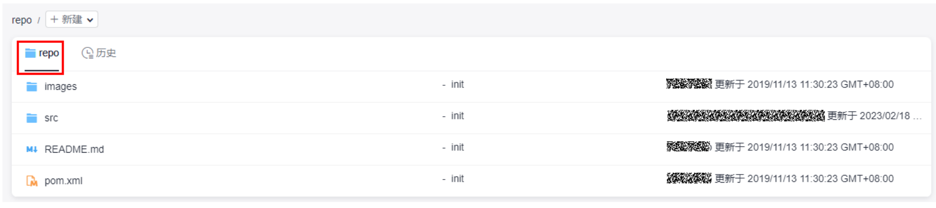
4. 鼠标停留在文件名称处，单击显示 图标即可修改文件名称。

   在代码托管仓库控制台修改文件名称，其实是一次“文件的名称修改 → add → commit→ push”，会生成提交记录。

5. 单击文件名称可将该文件内容显示于页面右侧，可对文件进行修改文件内容、追溯文件修改记录、查看历史记录、对比等操作。

仓库名称页签：查看分支或标签版本的文件详情内容

“仓库名称”页签位于仓库详情中，其默认状态显示主分支的文件详情内容，如下图所示。



其中包含字段：

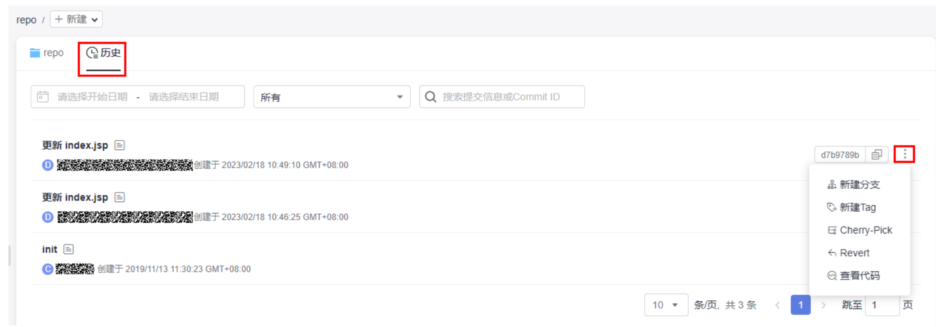
- 文件：文件或文件夹的名称。
- 提交信息：此文件或文件夹的上次提交信息（commit 的-m），单击可定位到此次提交记录。
- 创建者：此文件或文件夹的上次提交创建者。
- 更新时间：此文件或文件夹的上次更新时间。

说明

编辑、删除操作需要填写提交信息，相当于 git commit 中的-m 消息，其可以用于查看关联工作项。

历史页签：查看分支或标签版本的提交历史

“历史”页签位于仓库详情中，其显示分支或标签版本的提交历史，如下图所示。



在这个页面，可以对提交历史做如下操作：

- 单击“提交记录名称”，可以跳转到该次提交的详情中。
- 单击 ⋮ 可扩展功能如下：
  - 新建分支。
  - 新建 Tag：可针对此次提交补打标签。（什么是标签？）
  - Cherry-Pick：把此次提交作为最新的提交覆盖到某条分支上，这是一种版本找回方式。
  - Revert：还原此次提交。
  - 查看代码。

管理仓库文件

单击文件名称，可对该文件进行管理，功能如下：

说明

当您将浏览器窗口最大化时，上图下拉菜单中的功能会平铺展示。

- **文件名称**：查看文件详细内容。

表8-3 界面说明

界面功能	功能说明

界面功能	功能说明
文件容量	显示此时该文件的容量大小。
全屏显示	将该文件窗口扩展为全屏。
复制源代码	复制所展开文件内容到剪切板。
查看原始数据	可查看该文件的原始数据。
编辑	在线编辑文件。
下载	直接将此文件下载到本地。
删除	单独删除文件。
文件内容	显示文件的全部内容。
图标	单击可添加检视意见。

- **修改追溯**：查看文件的修改历史并追溯。
  在这个页面，修改者与修改内容相互对应，单击 **“提交信息名称”** 可以跳转到该次提交的详情中。
- **历史**：查看文件的提交历史。
  在这个页面，可以对提交历史做如下操作：
  - 单击“提交记录名称”，可以跳转到该次提交的详情中。
  - 单击 ⋮ 可扩展功能如下：
    - 新建分支。
    - 新建 Tag：可针对此次提交补打标签。（什么是标签？）
    - Cherry-Pick：把此次提交作为最新的提交覆盖到某条分支上，这是一种版本找回方式。
    - Revert：还原此次提交。
    - 查看代码。
- **对比**：提交的差异对比。
  在代码托管控制台对比出的差异，其展现形式优于 Git Bash 客户端，可以在界面选择不同提交批次，进行差异对比。

说明

本服务中的差异对比，其对比结果其实是显示您从左侧仓库版本向右侧仓库版本合并时对右侧仓库内文件所产生的影响，所以如果您想全面了解两个文件版本的差异，可以调整左右位置后再次对比，结合两次结果了解全部差异。

8.4.2 提交管理

在仓库详情的 **“代码”** 页签下的 **“提交”** 子页签，可以查看仓库的提交记录及仓库网络图。

提交记录

展示截至当前仓库某条分支或标签的整个提交记录，可根据选择具体的时间段、提交者、提交信息或 Commit 进行筛选记录。

仓库网络

仓库网络是以流向图的形式展现了某条分支或标签的整个提交（Commit）历史（包括动作、时间、提交者、提交系统生成备注和手动填写备注）以及提交历史的关系。

支持切换分支或标签查看，单击提交节点或提交备注信息，均可跳转到对应的提交记录中。

说明

相对于文件子页签中的历史而言，提交网络具备展现提交之间关系的优势。

8.4.3 分支管理

分支是版本管理工具中最常用的一种管理手段，使用分支可以把项目开发中的几项工作彼此隔离开来使其互不影响，当需要发布版本之前再通过分支合并将其进行整合。

在代码托管服务/Git 仓库创建之初都会默认生成一条名为 master 的分支，一般作为最新版本分支使用，开发者可以随时手动创建自定义分支以应对实际开发中的个性场景。

基于 Git 分支的经典工作模式

在基于分支的代码管理工作模式中，“Git-Flow”在业界被更多人认可，同时也被广泛应用，如果您的团队目前还没有更好的工作模式，可以先从尝试使用“Git-Flow”开始。

Git-Flow 是一种基于 Git 的代码管理工作模式，它提供了一组分支使用建议，可以帮助团队提高效率、减少代码冲突，其具备以下特性。

- **并行开发：**支持多个特性与 bug 修复并行开发，因其可以同时在不同分支中进行，所以在代码写作时互不受影响。
- **团队协作：**多人开发过程中，每条分支（可以理解为每个子团队）的开发内容可以被单独记录、合并到项目版本中，当出现问题时还可被精确检出并单独修改而不影响主版本的其它代码。
- **灵活调整：**通过 HotFix 分支的使用，支持各种紧急修复的情况，而不会对主版本以及各个团队的子项目产生干扰影响。

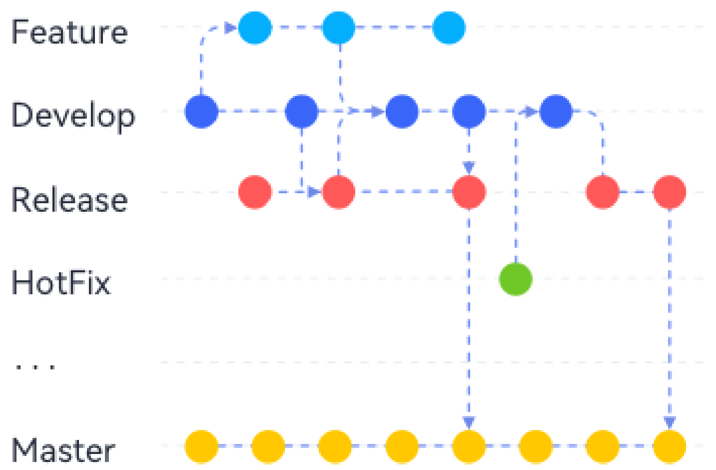


表8-4 Git-Flow 工作模式中分支的使用建议

分支名	Master	Develop	Feature_1\2...	Release	HotFix_1\2...
说明	核心分支，配合标签，用于归档历史版本，要保证其中的版本都是可用的。	开发主分支，用于平时开发的主分支，应永远是功能最新最全的分支。	新特性开发分支，用于开发某个新特性，可以几条并行存在，每条对应一个或一组新特性。	发布分支，用于检出某个要发布的版本。	快速修复分支，用于当现网版本发现 bug 时，拉出来单独用于修复这些 bug 的分支。
有效性	长期存在	长期存在	临时	长期存在	临时
何时被创建	项目仓库建立之初	在 master 分支被创建之后	• 收到新特性任务时，基于 develop 分支创建 • 当正在开发的新特性任务被拆分成出子任务时，基于	项目首次发布前，基于 develop 分支创建	当 master、bug 版本中发现问题时，基于对应版本（一般是 master 分支）创建

分支名	Master	Develop	Feature_1\2...	Release	HotFix_1\2..
			对应的父feature分支创建		
何时直接在此分支上开发	从不	一般不建议	当被创建出来，开始开发新特性时	从不	当被建立出来时
何时被其它分支合入	• 项目版本封版时，被develop或release分支合入 • 已发布版本中发现的bug被修复后，被对应的hotfix分支合入	• 新特性开发完成后，feature分支合入到此分支 • 当项目启动开发一个新版本时，被上一次历史发布版本（release、或master）合入	子feature分支开发、测试完成后，会合入到父feature分支	当需要发布一个版本时，被develop分支合入	-
何时合入到其它分支	-	• 当要发布版本时，合入到release分支 • 当需要归档版本时合入到master分支	当该分支上的新特性开发、测试完成时，合入到develop分支	• 当完成一次版本发布，将该版本归档时，合入到master分支 • 当要基于某一个发布版本，开始开发一个新版本时，合入到develop分支，起到初始化版本的作用	当其对应的bug修复任务完成时，会将其作为修复补丁合入master、develop分支
何时结	-	-	其对应的特	-	其对应的

分支名	Master	Develop	Feature_1\2...	Release	HotFix_1\2...
束生命周期			性已经验收(发布、稳定)后		bug 修复，已经验收(发布、稳定)后

说明

另外在使用 Git-Flow 工作模式时，业界普遍遵循如下规则：

- 所有开发分支从 develop 分支拉取。
- 所有 hotfix 分支从 master 分支拉取。
- 所有在 master 分支上的提交都必须要有标签，方便回滚。
- 只要有合并到 master 分支的操作，都需要和 develop 分支合并，保证同步。
- master 分支和 develop 分支是主要分支，并且都是唯一的，其它派生分支每个类型可以同时存在多个。

在控制台上新建分支

步骤 1 进入仓库列表。

步骤 2 单击仓库名称进入仓库详情。

步骤 3 切换到“代码”页签下的**“分支”**子页签，进入分支列表页面。

步骤 4 单击“新建”按钮，在弹出的窗口中选择要基于哪个版本（分支或标签）进行创建，填写新分支的名称，并且可关联现有工作项。

说明

分支名称须满足如下要求：

- 不支持以“-”、“.”、“refs/heads/”、“refs/remotes/”、“/”开头。
- 不支持空格和[\ < ~ ^ : ? * ! ( ) ' " | $ & ;等特殊字符。
- 不支持以“.”、“/”、“.lock”结尾。
- 不能有两个连续的点“..”。
- 不能包含序列“@{”。

当仓库中已存在同名称的分支/Tag 或者上级分支/Tag 时，则该分支不可被创建。

步骤 5 单击“确定”按钮，即可完成分支的新建。

----结束

在控制台中管理分支

在控制台的分支列表中可以进行如下操作。

- 分支筛选
  - **我的分支：**显示我创建的所有分支，按最近提交时间排序，最新提交的分支将更靠前。

  – **活跃分支**：显示过去一个月内存在开发活动的分支，按最近提交时间排序，最新提交的分支将更靠前。
  – **过时分支**：显示过去一个月内没有任何活动的分支，按最近提交时间排序，最新提交的分支将更靠前。
  – **所有分支**：显示所有分支，**“默认分支”**将显示在最前面，其余分支按最近提交时间排序，最新提交的分支将更靠前。
- 单击某个“分支名称”，可跳转到该分支的文件列表，可查看该分支的内容、历史等信息。
- 单击某个分支的提交 ID，可跳转到该分支的最新一次提交记录详情中，可查看本次提交的内容。
- 单击分支名称前面的勾选框，单击**“批量删除”**，可批量删除分支。
- 单击某个 ，可对该分支进行工作项关联操作。
- 单击某个 ，可定位到“对比”子页签，可以对将此分支与其它分支进行差异对比。
- 单击某个 ，可下载该分支的压缩包到本地。
- 单击某个 ，可以跳转到“合并请求”页签，可对该分支创建分支合并请求。
- 单击某个 ，可跳转到仓库设置中设置该分支为保护分支。
- 单击某个 ，可以按提示操作，将该分支进行删除。

须知

只有开启 IP 白名单的机器才可以从界面下载源码压缩包。

如果您误删了分支，可提交工单联系技术支持处理。

另外在控制台中您还可以对分支进行相关的设置：

- 默认分支管理

关于分支的常用 Git 命令

- **新建分支**

```
git branch <分支名称>            #在本地仓库基于目前的工作区，创建一条分支
```

示例如下：

```
git branch branch001            #在本地仓库基于目前的工作区，创建一条名为 branch001 的分支
```

无回显则为创建成功，如下图为使用了重复名称创建，更换一个名称再创建即可。

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

- **切换分支**

切换分支可以理解为将该分支的文件内容检出到当前的工作目录。

```
git checkout <分支名称>        #切换到指定分支
```

示例如下：

```
git checkout  branch002        #切换到名为 branch002 的分支
```

下图为切换成功：

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

- **新建并直接切换到新建的分支**

  有一种快速的操作办法，可以直接新建并切换到新建出来的分支，其用法如下：

```
git checkout -b <分支名称>     #在本地仓库基于目前的工作区，创建一条分支，并直接切换到该分支
```

示例如下：

```
git checkout -b branch002      #在本地仓库基于目前的工作区，创建一条名为 branch002 的分支，并直接切换到该分支
```

下图为该命令执行成功：

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

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

- **查看分支**

  您可以使用对应的命令查看本地仓库的分支、远程仓库的分支，亦或是全部，这组命令只是将分支名称罗列，如果想查看分支内的具体文件，请使用切换分支。

```
git branch                          #查看本地仓库分支
git branch -r                       #查看远程仓库分支
git branch -a                       #同时查看本地仓库与远程仓库的分支
```

如下图，分别依次执行了以上三种命令，Git 清晰的将本地仓库与远程仓库中的分支以不同的样式展现（远程仓库分支展现形式 remote/远程仓库别名/分支名）。

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

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

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

- **合并分支**

  当一条分支上的开发任务完成了，有时需要将其合并到另一条分支，以做功能归档，合并分支可以理解为将另一条分支最新的修改同步到当前所处分支，其用法如下：

```
git merge <要合并过来的分支的名称>        #将一条分支合并到当前的分支中
```

  分支合并前一般要先切换到被合入的分支，下面以将 branch002 合入到 master 分支为例进行演示：

```
git checkout master                    #切换到 master 分支
git merge branch002                    #将名为 branch002 的分支合入到 master 分支
```

  如下图是上面命令的执行效果，可以看到合并成功，合并了一个文件的变化，其变化是新增了 3 行内容。

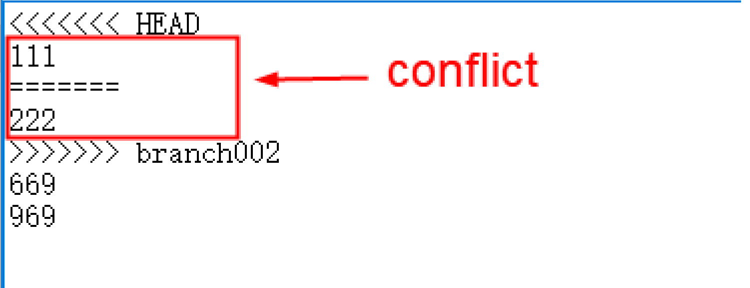
```
Administrator@ecstest-paas-lw        MINGW64 ~/Desktop/01_developer (branch001)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'HTTPSOrigin/master'.

Administrator@ecstest-paas-lw        MINGW64 ~/Desktop/01_developer (master)
$ git merge branch002
Updating 6b40550..09fd1d4
Fast-forward
 fileOnBranch002.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 fileOnBranch002.txt
```

说明

有时在合并时会提示出现了文件修改冲突（如下图“fileOnBranch002.txt”这个文件在合并时冲突了）。

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

解决方法是打开这个冲突的文件，手动编辑以将有冲突的代码（如下图）进行取舍，解决后保存，再进行一次 add 和 commit 操作以将其结果存储进本地仓库。



这与本地仓库提交到远程仓库时产生文件冲突的解决方法是类似的，您可以参考解决合并请求的代码冲突了解其工作原理。

在实际开发中，如果团队使用了合理的合作模式，可以基本杜绝这种情况的产生。

- **删除本地分支**

```
git branch -d <分支名>
```

  示例如下：

```
git branch -d branch002          #删除本地仓库中，名为 branch002 的分支，下图为执行成功
```

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

- **删除远程仓库分支**

```
git push <远程仓库地址或别名> -d <分支名>
```

示例如下：

```
git push HTTPSOrigin -d branch002        #从别名为 HTTPSOrigin 的远程仓库中删除名为
branch002 的分支，下图为删除成功
```

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

- **将本地新建的分支推送到远程仓库**

```
git push <远程仓库地址或别名> <分支名>
```

示例如下：

```
git push HTTPSOrigin branch002        #将本地名为 branch002 的分支，推送到别名为
HTTPSOrigin 的远程仓库，下图为推送成功
```

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

说明

如果推送失败请检查连通性：

- 确保您的网络可以访问代码托管服务。

  请在 Git 客户端使用如下测试命令验证网络连通性。

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

  如果返回内容含有“connect to host **********.com port 22: Connection timed out”，则您的网络被限制，无法访问代码托管服务，请求助您本地所属网络管理员。

- 请检查建立密钥配对关系，必要时重新生成密钥并到代码托管控制台进行配置，请确认 SSH 密钥或 HTTPS 密码正确配置。

8.4.4 标签管理

标签（tag）是 Git 提供的帮助团队进行版本管理的工具，您可以使用 Git 标签标记提交，从而将项目中的重要的版本管理起来，以便日后精确检索历史版本。

标签会指向一个 commit，就像一种引用，无论后续版本怎么变化，它永远指向这个 commit 不会变化，相当于一个被永远保存的版本快照（只有手动删除时才会被剔除版本库）。

在使用 Git 进行代码管理时，您可以根据每次提交（commit）的 ID 去查找、追述历史版本，这个 ID 是一长串编码（如下图中所示），相对于熟知的“V 1.0.0”这样的版本号，CommitID 不便于记忆，同时也不具备可识别性，这时就可以给重要的版本打上标

签，给它一个相对友好的名称（比如“myTag_V1.0.0”、“首个商业化版本”）以便更容易记住和追溯它。

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

    fix #7369022 fix a bug
```

如何在控制台为最新的提交创建标签？

步骤 1 进入仓库列表。

步骤 2 单击仓库名称进入仓库详情。

步骤 3 单击“代码”页签下的“**Tags**”子页签，在这里可以看到标签列表。

步骤 4 单击“新建”按钮，弹出新建标签页面如下图，选择要基于哪条分支或标签的最新版本进行标签的创建。

说明

标签名称须满足如下要求：

- 不支持以“-”、“.”、“refs/heads/”、“refs/remotes/”、“/”开头。
- 不支持空格和[\ < ~ ^ : ? * ! ( ) ' " | $ & ;等特殊字符。
- 不支持以“.”、“/”、“.lock”结尾。
- 不能有两个连续的点“..”。
- 不能包含序列“@{”。

如果在描述中输入信息会生成附注标签（描述相当于 -m 后的内容），不输入则生成轻量标签。（什么是附注标签？）

当仓库中已存在同名称的分支/Tag 或者上级分支/Tag 时，则该 Tag 不可被创建。

步骤 5 单击“确定”按钮，即可基于某个分支的最新版本生成标签，页面跳转到标签列表。

----结束

如何在控制台为历史版本创建标签？

步骤 1 进入仓库列表。

步骤 2 单击仓库名称进入仓库详情。在“代码”页签中，单击“文件 > 历史”页签。

步骤 3 在历史列表中的某条提交记录中，单击 ⋮ ，选择**“新建 Tag”**，弹出为历史版本新建标签的弹窗。

说明

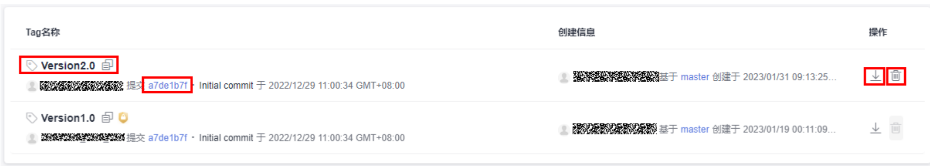
如果在描述中输入信息会生成附注标签（描述相当于 -m 后的内容），不输入则生成轻量标签。（什么是附注标签？）

步骤 4 单击“确定”按钮，即可基于某个指定历史版本生成标签，页面跳转到标签列表。

----结束

在控制台管理标签

- 在控制台的标签列表中，可查看该远程仓库中的全量标签并进行如下操作。



  – 单击“标签名”，跳转到该标签对应版本的文件列表。
  – 单击**“提交号”**，跳转到该次提交（commit）的详情页面。
  – 单击 ⤓ ，可下载 tar.gz 或 zip 格式的被标签版本的文件包。
  – 单击 🗑，可以将此标签从代码托管仓库删除（想从本地删除请 clone、pull 或本地手动-d 删除）。

须知

如果仓库设置 IP 白名单，则只有 IP 白名单内的机器才可以在界面下载仓库源码，如果仓库没有设置 IP 白名单，则均可在界面下载仓库源码。

- 在控制台创建分支时，您可以选择基于某个标签去创建分支。
- 在控制台中，单击**“代码”**页签，单击目标文件的“文件名称”，单击文件的**“对比”**页签，可在该文件的提交记录之间做差异对比。

标签的分类

Git 提供的标签类型分为两种：

- **轻量级标签：**仅是一个指向特定 commit 的引用，可以理解为给特定 commit 起了一个别名。

```
git tag <你给标签起的名称>
```

如下图是一个轻量标签被查看详情时的显示内容，可以看到它其实就是一次 commit 的别名。

- **附注标签：**指向一个特定的 commit，但在 Git 中被作为一个完整对象存储，相比于轻量标签，附注标签可以为标签附上说明，类似代码的注释功能，方便注解标签。在标签信息的记录中，除包括标签名、附注标签说明外，同时包含了打标签者名字、电子邮件地址、打标签时间/日期。

```
git tag -a <你给标签起的名称> -m <"你给标签编写的说明">
```

如下图是一个附注标签被查看详情时的显示内容，它指向了一次 commit，相对于轻量标签它包含了更多的信息。

说明

两种标签都可进行版本标识，**附注标签**包含了更多的信息，同时其在 Git 中也以更稳定安全的结构被存储，被更多的应用于大型企业与项目中。

关于标签的常用 Git 命令

- **新建轻量标签**

```
git tag <你给标签起的名称>        #为当前最新的提交打上轻量标签
```

示例如下：

```
git tag myTag1                  #为当前最新的提交上名称为 myTag1 的轻量标签
```

- **新建附注标签**

```
git tag -a  <你给标签起的名称> -m <"你给标签编写的说明">          #为当前最新的提交打上附注标签
```

示例如下：

```
git tag -a myTag2 -m "This is a tag." #为当前最新的提交打上名称为 myTag2 的附注标签，标签的备注信息为 This is a tag.
```

- **为历史版本打标签**

也可以对于历史版本打标签，只要使用给 git log 命令获取到历史版本的 commit ID 就行，以附标签为例，其操作如下。

```
git log                  #会显示历史提交信息,获取 commitID 如下图,只取前几位即可，按 q 返回
```

```
git tag -a historyTag -m "Tag a historical version." 6a5b7c8db    #为 commitID 为 6a5b7c8d 开头的历史版本打上一个标签，名称为 historyTag，备注为 Tag a historical version.
```

说明

- 执行完新建标签操作，如果无回显则是创建成功，如果有回显一般是标签名称重复（回显如下图），更换标签名称重新执行即可。
- Git 支持为一次 commit 打上多个标签，其在 log 中显示如下图，标签名不能重复。

- **查看本地仓库的标签列表**

将目前仓库内的标签的名称全部显示出来，在使用时可对其添加参数达到进行过滤的效果 。

```
git tag
```

- **查看指定标签详情**

```
git show <你想查看的标签的名称>
```

示例如下：

将名称为 myTag1 的标签详细信息和其指向的 commit 的信息显示出来，其执行回显示例如下。

```
git show myTag1
```

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

This is a tag for show you~!

commit 53538093c56de4df204b12ca4841926eef630bbd (tag: myTag1)
Author: 02_dev <yuhu .com>
Date:

    fix #7369022 fix a bug

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

- **将本地标签推送到远程仓库**
  - 默认情况下，将本地仓库推送（git push）到远程仓库时，不会把标签一起推送；当从远程仓库同步内容到本地时（clone、pull），会自动将远程仓库的标签同步到本地仓库，所以如果想将本地标签分享项目里的其他人时，需要使用单独的 Git 命令，其用法如下。

    ```
    git push <远程仓库地址或别名> <你想推送的标签的名称>      #将指定标签推送到远程仓库
    ```

    示例如下：

    将名为 myTag1 的本地标签推送到别名为 origin 的远程仓库。

    ```
    git push origin myTag1
    ```

  - 当您需要将本地所有新增标签推送到远程仓库时，可使用如下命令

    ```
    git push  <远程仓库地址或别名> --tags
    ```

说明

当您在远程仓库建立了一个标签，又在本地仓库建立了一个同名的标签，这时在推送时会失败（出现标签冲突），只能删除其一，再次推送。

如何在远程仓库查看全量标签？

- **删除本地标签**

  ```
  git tag -d <你要删除的标签的名称>
  ```

  其应用示例如下图，删除本地名为 tag1 的标签，删除成功。

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

- **删除远程仓库标签**

  如同标签的创建需要单独手动推送，标签的删除，也需要手动推送，其具体用法如下。

  ```
  git push <远程仓库地址或别名> :refs/tags/<你要删除的标签的名称>
  ```

  示例如下，图为删除成功。

  ```
  git push HTTPSOrigin :refs/tags/666          #删除别名为 HTTPSOrigin 的远程仓库的名为 666 的标签
  ```

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

如何使用标签找回历史版本

当您要查看某个标签指向版本的代码时，可以将其检出到工作区。由于被检出的版本仅隶属于标签，而不属于任何分支，因此该代码可以编辑，但是不能 add、commit。您可以基于工作区新建一条分支，在此分支上修改代码，并将此分支合入主干。具体的操作步骤如下所示。

1. 通过标签检出历史版本。

```
git checkout V2.0.0              #将被标签为 V2.0.0 的版本检出到工作区
```

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

2. 基于当前的工作区新建一条分支并切换到其中。

```
git switch -c  forFixV2.0.0        #新建一条名为 forFixV2.0.0 的分支，并切换到其中
```

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

3. （可选）如果修改了新建的分支的内容，需要将修改内容提交到该分支的版本库中。

```
git add .                          #将修改添加到新分支的暂存区
git commit -m "fix bug for V2.0.0"  #将修改内容存入该分支的版本库
```

```
$ git add .

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

4. 切换到 master 分支，并将新建立的分支合入（本示例中为 forFixV2.0.0 分支）。

```
git checkout master               #切换到 master 分支
git merge forFixV2.0.0             #将基于历史版本的修改 合入到 master 分支
```

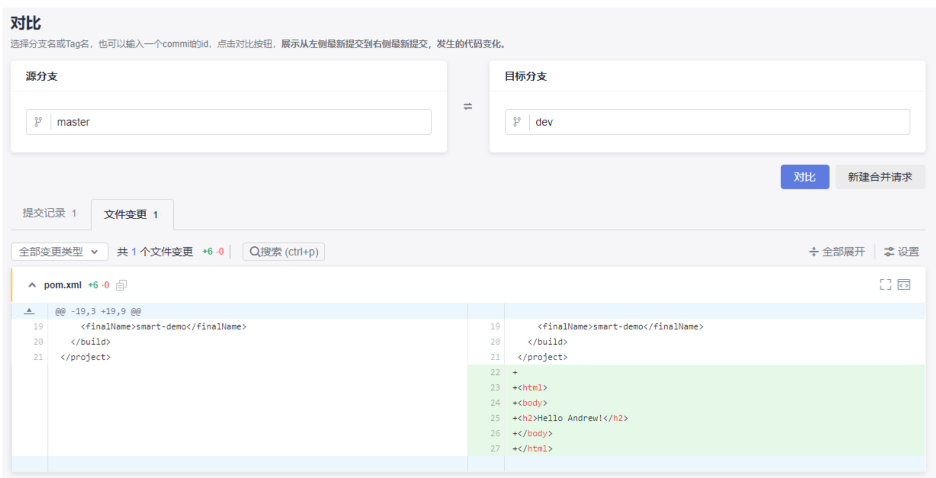
```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

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

说明

以上命令旨在帮助您理解通过标签找回历史版本的过程原理，请根据原理自行裁剪增补 Git 命令以完成您在特定场景下需要的操作，不建议全流程直接复制使用。

8.4.5 对比管理

在仓库详情的**“代码”**页签下的**“对比”**子页签，支持通过对比查看分支之间或标签版本之间发生的代码变化。



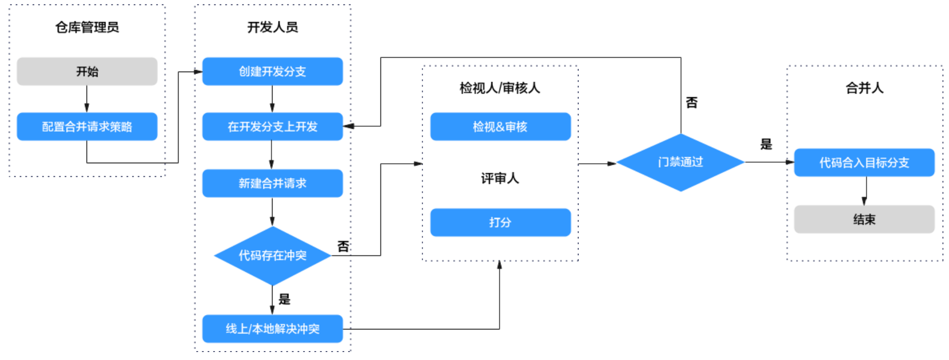
说明

分支之间对比后可根据需要新建合并请求。

8.5 管理合并请求

8.5.1 合并请求管理

代码托管服务支持多分支开发，并为分支合并建立了可配置的审核规则，当一个开发者发起一次合并请求时，可选择部分仓库成员参与到代码审视中，以确保合并代码的正确性。



合并请求列表

在仓库详情的“合并请求”页签中，可以看到“合并请求列表”页面。

- 可以切换、查看不同状态的合并请求。
- 通过单击请求标题可以进入合并请求详情页。
- 可以查看请求的简要信息，包括：涉及的分支、创建时间、创建人。
- 提供了多条件维度的查找功能。
- 在左上方有新建合并请求入口。

说明

开启中：代表该请求已进入检视或合并阶段，分支未合并。

已合并：代表该请求已经完成审核，并完成分支合并的动作。

已关闭：代表该请求被取消，分支未产生实际合并。

所有：显示所有状态的合并请求。

新建合并请求

当您在开发分支上完成了功能开发，并需要发起合并请求时，请按照以下流程操作。

步骤 1 进入目标仓库详情页。

步骤 2 切换到“合并请求”页签。

步骤 3 单击“新建”按钮，选择要合并的分支。

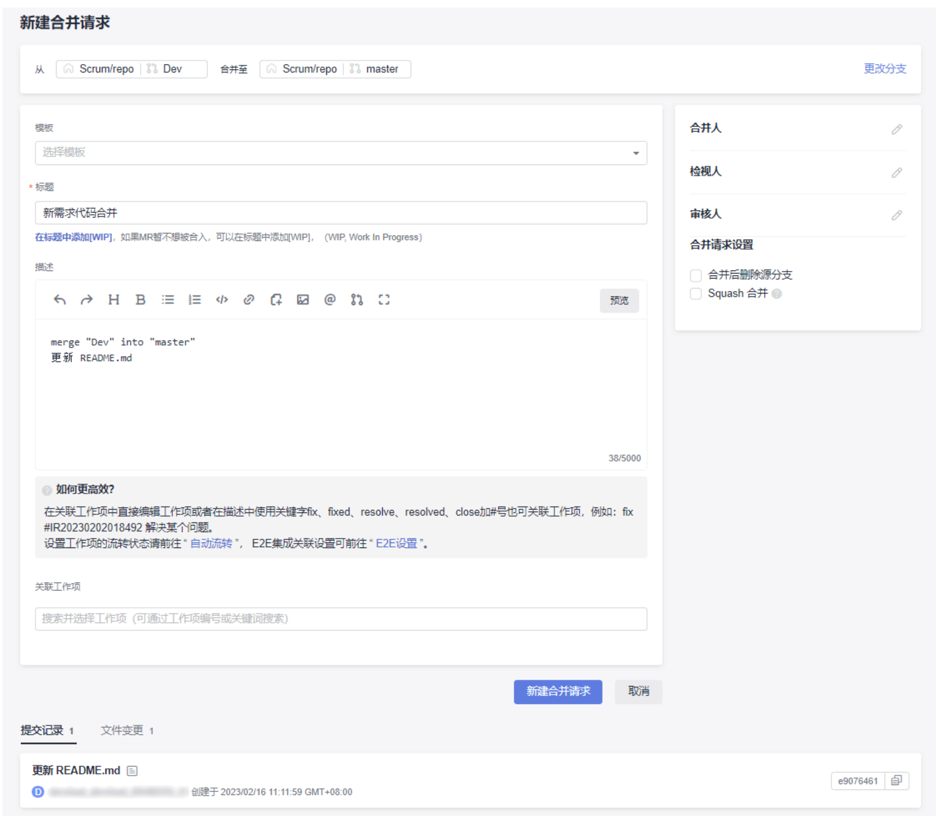
如上图在本示例中将刚完成开发任务的 Dev 分支合并到 master 分支中。

说明

支持源分支选择 Fork 仓库的分支。

步骤 4 单击“下一步”按钮，此时系统会检测两条分支是否有差异。

- 如果分支没有差异，系统会做出提示，且不能新建合并请求。
- 如果分支存在差异，则进入“新建合并请求”页面。



在“新建合并请求”页面的下方可以看到两条分支的文件差异对比详情、要合并分支的提交记录。

步骤 5 根据下表参数说明，填写页面信息。

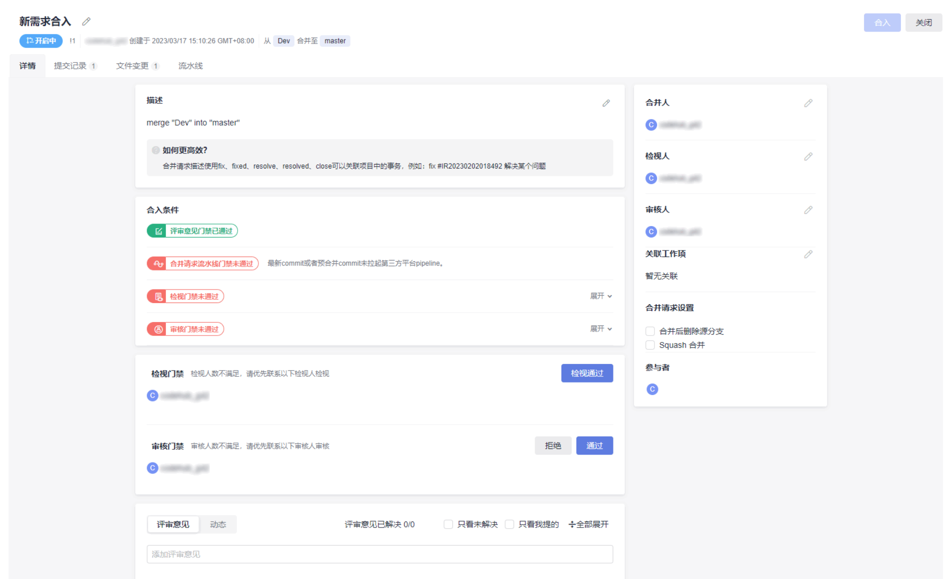
表8-5 参数说明

参数	说明
更改分支	单击可返回上一步更改需要合并的分支。
模板	如果仓库管理员或所有者已为该仓库创建合并请求模板，您可以直接选择使用模板。
标题	输入合并请求的标题。
描述	会结合分支合并情况与要合并分支的提交（commit）备注生成默认值，您可以根据项目情况进行修改。 **说明** 该输入框采用 markdown 格式，字符数限制在 5000 字符以内，即将超出上限时，使用顶部的操作按钮会按照 markdown 的语法替换内容。
关联工作项	可选择将合并动作关联到某个工作项，以起到自动改变工作项状态的作用。
合并人	在合并请求满足合入要求时，一般是所有审核人审核通过、所有问题都被解决（可设置不解决也能合并），合并人有权限执行合并操作（单击按钮）、也有权限关闭合并请求。

参数	说明
检视人	被指定参与合并分支检视，可以提出问题给发起人。
审核人	被指定参与合并分支评审，可以给出审核意见（审核通过、拒绝），也可以提出问题给发起人。
合并后删除源分支	可选择是否合并后删除源分支，初始会带入合并请求设置中预设状态。
Squash 合并	开启 Squash 合并，可使基本分支的历史记录保持干净，并带有有意义的提交消息，而且在必要时可以更简单地恢复，详情请参考 Squash 合并。

步骤 6 单击“新建合并请求”按钮，可以完成合并请求的提交，页面会跳转到该“合并请求详情页”。

在合并请求详情中，可以看到合入条件达成的状态、合并人、检视人、审核人、所关联的工作项等信息，同时可以查看可留下评审意见，可标注评审意见为待解决状态，并可看到该合并涉及的所有动态。



- “提交记录”：可以看到源分支的相关提交记录。
- “文件变更”：可以看到此次合并的变更内容，并可具体筛选出新增、修改、删除、重命名等变更种类。
- “流水线”：可以看到门禁流水线的信息。

----结束

说明

- 当发起分支合并请求时，其相关人员（审核人、合并人）会收到提醒邮件。审核人不能为合并请求创建者。

- 单个文件差异超过 5000 行、差异文件个数超过 100 个时，建议使用客户端合并后，推送到代码托管。

对合并请求进行检视、审核与合入

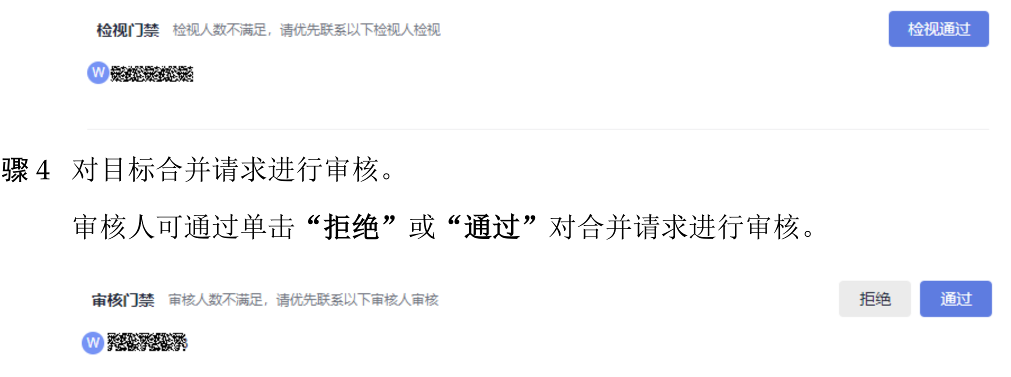
当检视人、审核人、合并人收到系统的分支合并请求通知邮件时，请按以下步骤进行操作。

步骤 1 进入目标仓库详情页。

步骤 2 切换到“合并请求”页签，单击目标合并请求名称，查看详情。

步骤 3 对目标合并请求进行检视。

检视人与审核人均可对合并请求进行检视并给予检视意见，如果无修改意见，检视人可单击**“检视通过”**完成检视。



步骤 4 对目标合并请求进行审核。

审核人可通过单击**“拒绝”**或**“通过”**对合并请求进行审核。

步骤 5 通过合并请求门禁。

表8-6 合入条件说明

合入条件	说明
代码合并冲突	当源分支代码与目标分支代码产生合并冲突时，需要先解决冲突才可进行下一步操作，解决代码冲突可参考解决合并请求的代码冲突。
评审意见门禁	当发起人解决所有检视人或审核人的评审意见后，门禁显示通过，更多门禁详情请参考评审意见门禁详解。
流水线门禁	当最新 commit 或者预合并 commit 拉起并成功执行流水线时，门禁显示通过，更多门禁详情请参考流水线门禁详解。
E2E 单号未关联	当合并请求关联工作项后，门禁显示通过，更多门禁详情请参考E2E 单号关联门禁详解。
星级评价未通过	当指定人员进行星级评价后，门禁显示通过，更多门禁详情请参考星级评价门禁详解。
检视门禁	当已检视的检视人数达到最小检视人数时，门禁显示通过，更多门禁详情请参考检视门禁详解。

合入条件	说明
审核门禁	当已审核的审核人数达到最小审核人数时，门禁显示通过，更多门禁详情请参考审核门禁详解。

步骤 6 对目标合并请求进行合入。

当发起人通过以上合入条件后，合并人可单击“合入”按钮进行合入，反之，合并人可单击**“关闭”**将请求关闭。

----结束

Squash 合并

Squash 合并是将合并请求的所有变更提交信息合并为一个，并保留干净的历史记录。当用户在处理功能分支只关注当前提交进度，而不关注提交信息时，可使用 squash merge。

说明

当勾选 **Squash 合并**，可将**源分支**的多个连续变更记录合并为一个提交记录（**Squash 提交信息**），提交到**目标分支**。

- 如果合并请求中的变更记录只有一个提交记录，则勾选 **Squash 合并**后，**目标分支**中的提交记录为**源分支**的提交记录。
- 如果合并请求中的变更记录有多个提交记录，则勾选 **Squash 合并**后，**目标分支**中的提交记录为 **Squash 提交信息**。

为了您更深入了解此功能，下面进行实际操作：

步骤 1 新建仓库。

仓库名称命名为“repo”。

步骤 2 新建分支。

分支名称命名为“Dev”。

步骤 3 新建提交。

将“新建一个文件”作为一次 Commit 提交。

Dev 分支：新建两个文件并分别命名为“功能一”及“功能二”。

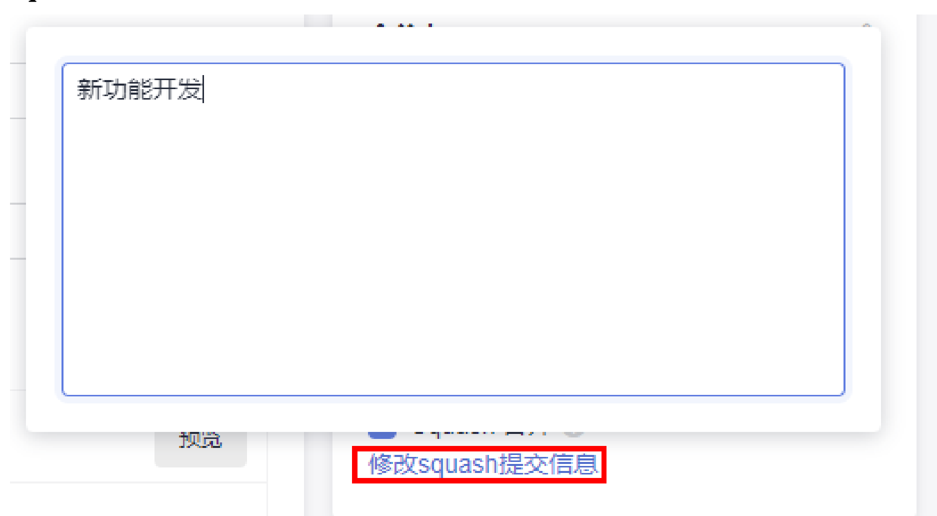
步骤 4 查看开启**“Squash 合并”**前的效果。

切换到**“Dev 分支”**下的**“代码 > 提交 > 提交记录”**界面，查看提交信息。

步骤 5 新建并合入合并请求。

1. 源分支为 Dev，目标分支为 master，修改以下修改即可新建合并请求。

   Dev 分支：合并请求标题命名为“代码合入”，勾选**“Squash 合并”**并**“修改 squash 提交信息”**为“新功能开发”。



2. 完成对合并请求进行检视、审核后，即可合入请求。

步骤 6 查看开启**“Squash 合并”**后的效果。

请求合入成功后，切换到**“master 分支”**下的**“代码 > 提交 > 提交记录”**界面，与步骤 4 对比，提交信息已被合并。

----结束

8.5.2 解决合并请求的代码冲突

在多人团队使用代码托管服务时，不可避免的会出现两个人同时修改了一个文件的情况，这时在推送（push）代码到代码托管仓库时就会出现代码提交冲突并推送失败，如下图就是因为本地仓库与远程仓库文件修改的冲突所产生的推送失败。

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

Administrator@ecstest-paas-1          MINGW64 ~/Desktop/02_developer/
 (master)
$
```

说明

- 不同版本的 Git、不同编译工具的 Git 插件所返回提示的内容不完全一致，但所表达的意思基本一致。
- 只要在返回提示的内容中解读出，推送失败、另一个仓库成员，两个信息，一般即为产生了提交冲突。
- Git 在文件合并时是比较智能的，对于同一个文件不同位置的修改内容会自动合并，只有在同一个文件同一个位置被同时修改时（本地仓与远程仓的当前版本有差异），才会产生冲突。
- 在分支合并时，有时也会产生冲突，这时的判定方式与解决办法与提交远程仓库时的冲突基本一样，如下图是本地分支 branch1 向 master 分支合并时产生了冲突（file01 文件的修改冲突了）。

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

- 可解决合并请求的文件冲突数量不得超过 50 个，且单文件冲突内容的大小不得超过 200KB。

如何解决代码提交冲突？

当代码提交冲突产生时，您可以将远程代码仓库拉取（pull）到本地仓库的工作区，这时 Git 会将可以合并的修改内容进行合并，并将不能合并的文件内容进行提示，开发者只需要对提示的冲突内容进行修改即可再次推送到远程仓库（add → commit → push）,这时冲突就解决完毕了。

如下图所示，在做拉取（pull）操作时，Git 提示您，一个文件合并时产生了冲突。

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

在修改冲突文件时应该考虑清楚，必要时要与冲突方联系协商解决，避免覆盖他人代码。

说明

git pull 可以理解为 git fetch 的操作 + git merge 的操作，其详细说明如下：

```
git fetch origin master #从远程主机的master分支拉取最新内容
git merge FETCH_HEAD    #将拉取下来的最新内容合并到当前所在的分支中
```

在 merge 的时候，会将有冲突不能合并的内容做出提示。

示例：冲突的产生与解决

下面模拟一个情景来帮助理解冲突的产生和解决的过程，情景如下。

某公司的一个项目使用代码托管服务和 Git 工具来管理，这个项目有一个功能（假设此功能涉及的修改文件是 file01）由开发者 1 号（以下用 01_dev 表示）和开发者 2 号（以下用 02_dev 表示）共同开发，项目上线前一周，大家都在修改代码，产生了如下情况。

1. file01 存储在远程仓库，此时文件内容如下。

   file01

   ```
   1  ##file01AAAAAAAAAAAA
   2  ##file02BBBBBBBBBBBB
   3  ##file03CCCCCCCCCCCC
   4  ##file04DDDDDDDDDDDD
   5  
   ```

2. 01_dev 在本地仓库修改了文件 file01 的第二行等内容，并已经成功推送到了远程仓库，此时 01_dev 的本地仓库和远程仓库的文件内容如下。

   file01

   ```
   1  ##file01AAAAAAAAAAAA
   2  ##modify by 01_dev
   3  ##file03CCCCCCCCCCCC
   4  ##file04DDDDDDDDDDDD
   5  ## add one line by 01_dev
   ```

3. 此时 02_dev 也在本地仓库修改了文件 file01 的第二行等内容，在推送远程仓库时 Git 提示 file01 文件上产生了冲突，02_dev 的本地仓库文件内容如下，此时与远程仓库的冲突很明显。

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

4. 02_dev 将远程仓库的代码拉取到本地，发现文件第二行开始的冲突并马上联系 01_dev 进行冲突的解决。

5. 打开冲突的文件（如下图所示），发现都对第 2 行进行了修改，也都在最后一行添加了内容，Git 将第二行开始的内容识别为冲突。

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

=======
##modify by 01_dev
##file03CCCCCCCCCCCC
##file04DDDDDDDDDDDD
## add one line by 01_dev
>>>>>>> af5daac097230b2f8f
```

modify by 02_dev

modify by 01_dev

commit ID

说明

Git 很智能的将两个人的修改同时显示出来，并用“=======”分割开来

- “<<<<<<<HEAD” 与 “=======” 中间的是冲突位置中对应的本地仓库的修改。
- “=======” 与 “>>>>>>>” 中间的是冲突位置中对应的远程仓库的修改（也就是刚拉取下来的内容）。
- “>>>>>>>” 后面是本次的提交 ID。
- “<<<<<<<HEAD”、“=======”、“>>>>>>>”、提交 ID 并非实际编写的代码，解决冲突时注意删除。

6. 两人商量的解决方案是保留两个人的修改内容，由 02_dev 负责修改，修改后 02_dev 的本地仓库文件内容如下图，同时保留了两个人的修改和新增内容。

```
##file01AAAAAAAAAAAA
## modify by 02_dev
##modify by 01_dev
##file03CCCCCCCCCCCC
##file04DDDDDDDDDDDD
## add by 02_dev
## add one line by 01_dev
```

7. 这样 02_dev 就可以重新推送（add → commit → push）本次合并后的更新到远程仓库，推送成功后，远程仓库文件内容如下。此时冲突解决

file01

```
1   ##file01AAAAAAAAAAAA
2   ## modify by 02_dev
3   ##modify by 01_dev
4   ##file03CCCCCCCCCCCC
5   ##file04DDDDDDDDDDDD
6   ## add by 02_dev
7   ## add one line by 01_dev
```

说明

在上面的示例中，使用 txt 文本方式进行的演示，在实际开发中不同的文本编辑器、编程工具的 Git 插件中，对冲突的展示会略有不同。

如何避免冲突的产生？

代码提交、合并冲突经常发生，但只要在代码开发前，做好仓库预处理工作，就能有效的避免冲突的产生。

在示例：冲突的产生与解决中，开发者 02（02_dev）成功的解决了提交远程仓库时遇到的冲突问题，此时他的本地仓库与远程仓库的最新版本内容是一样的，但是开发者 01（01_dev）本地仓库和远程仓库仍然是有版本差异的，此时如果直接推送本地仓库（push），仍然会产生冲突，那么如何避免呢？

方式一（推荐新手使用）：

如果开发者本地的仓库不常更新使用，在做本地修改时，可以重新 clone 一份远程仓库的内容到本地，修改后再次提交，这样简单直接的解决了版本差异问题，但缺点是如果仓库较大、更新记录较多，clone 过程将耗费一定的时间。

方式二：

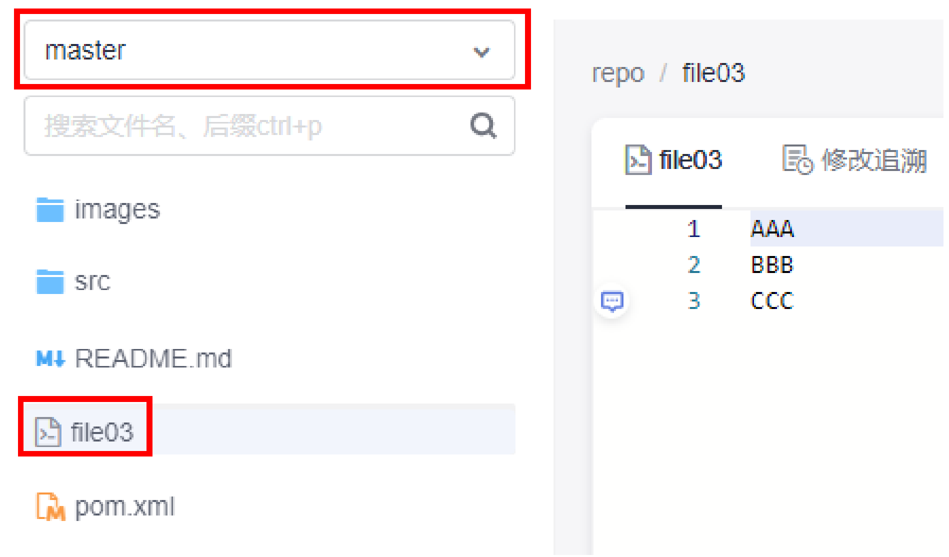
如果开发者每天都要对本地仓库进行修改，则建议在本地新建一条开发分支进行代码修改，在要提交远程仓库时，切换到 master 分支并将远程仓库的最新 master 分支内容拉取到本地，在本地进行分支合并，对产生的冲突进行修复，成功将内容合并到 master 分支后，再提交到远程仓库。

如何在代码托管服务的控制台上解决分支合并冲突？

代码托管服务支持分支管理，当在进行分支合并时，有时也会产生冲突，下面模拟一次产生了冲突的分支合并请求，并将其解决。

步骤 1 新建一个仓库。

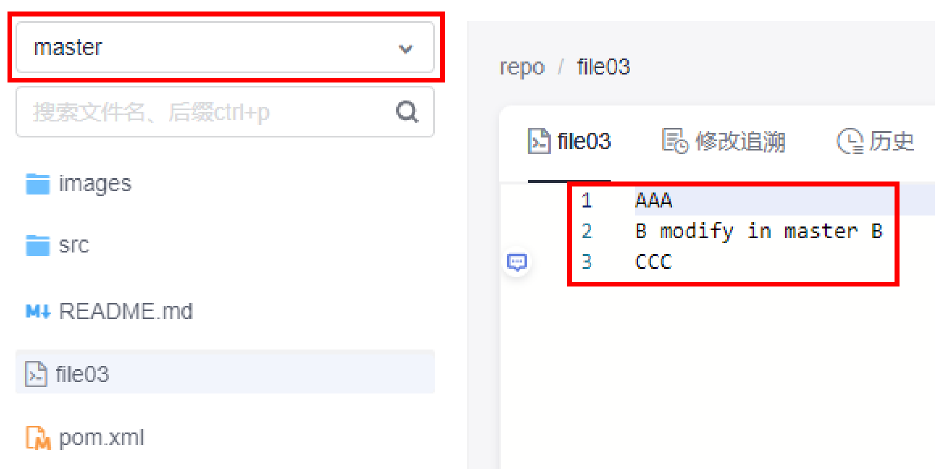
步骤 2 在仓库中新建一个文件，在本案例中，在 master 分支上新建一个名为 file03 的文件，其内容初始编写如下。



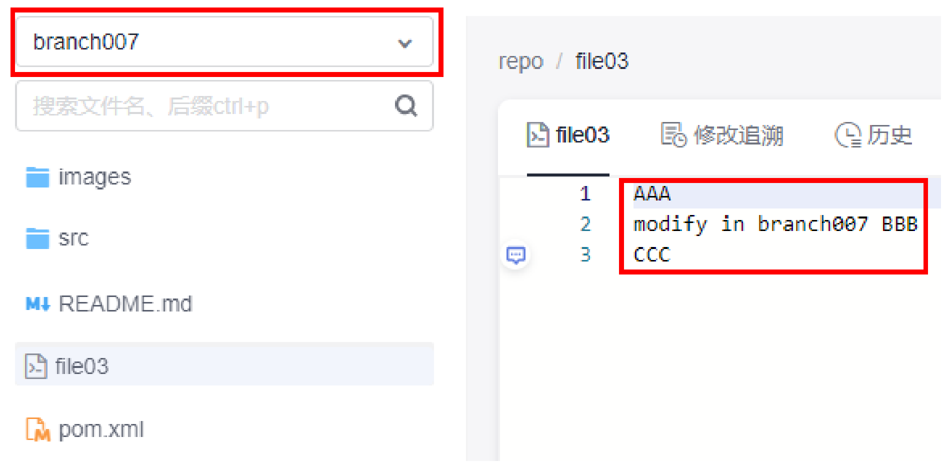
步骤 3 基于 master 分支新建一个分支，在本案例中，将其命名为“branch007”。

新建成功后，在 master 分支、branch007 分支中的内容是一模一样的，下面需要让它们产生差异。

步骤 4 在 master 分支中，将 file03 的内容修改为如下图所示，将提交信息填写为“modify in master”。



步骤 5 切换到 branch007 分支，并将 file03 文件修改为如下图所示，将提交信息填写为“modify in branch007”。此时切换分支即可直观的看到两个分支上已经产生了差异，也就是冲突。



步骤 6 新建一个合并请求，选择将 branch007 分支合入到 master 分支，单击“新建合并请求”按钮即可提交一条分支合并请求。

此时将自动跳转到“合并请求详情”页面，您也可以在“合并请求列表”中单击请求的名称进入此页面，如下图所示，代码托管服务提示您**“代码合并冲突未解决”**，并建议您“在线解决冲突”或“本地解决冲突”

步骤 7 下面根据提示，解决冲突：

- **在线解决冲突**（推荐在代码量较小或涉及冲突的代码量较小的情况下使用）
  a. 单击提示内容“在线解决冲突”，弹出如下图页面非常直观的展示了代码冲突。

  在

  b. 当情况较复杂，简单的直接覆盖无法解决问题时，可单击 进入“手动编辑”模式，如下图所示，可以看到跟示例：冲突的产生与解决中的冲突展现格式很像。

  c. 在上述页面中手动修改代码以解决冲突，并进行提交即可。

说明

提交时注意需要填写提交信息。

上图中“<<<<” 、“>>>>”、“====”等所在行是冲突展现与分割符，在修改代码解决冲突时，要注意将其删除。

- **本地解决冲突**（推荐在大型项目中使用）

单击提示内容“本地解决冲突”，即弹出指导内容如下图所示，按照步骤操作即可。

说明

代码托管服务会根据您的分支名自动生成适合您的 Git 命令，您只需要复制并在本地仓库执行即可。

步骤 8 使用上述两种方法之一，解决了冲突之后，此时可以单击“合入”按钮，进行分支合并的操作了，系统会提示您合并成功。

（可选）您也可以使用分支合并评审流程。

此时 master、branch007 两个分支的内容是一样的了，您可以切换分支进行查看验证。

----结束

8.5.3 评审意见门禁详解

门禁的开启/关闭

步骤 1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤 2 配置门禁。

- 勾选合入条件下的**“评审问题全部解决才能合入”**，单击**“提交”**保存设置，门禁开启。
- 取消勾选合入条件下的**“评审问题全部解决才能合入”**，单击**“提交”**保存设置，门禁关闭。

----结束

门禁触发的效果

该合并请求的检视人或审核人可在合并请求的**“文件变更”**中，将鼠标置于代码行，单击图标添加评审意见，也可在合并请求的**“详情 > 评审意见”**中直接添加评审意见。

- **评审意见门禁已通过**：当合并请求中无评审意见，或者所有评审意见均无需解决或已被解决时显示。

  合入条件

- **存在未解决的评审意见**：当合并请求中的评审意见未被解决时显示。

  合入条件

门禁的通过

当您已解决评审意见中提出的问题后，可在合并请求的**“详情 > 评审意见”**中需要将评审意见的状态由**“未解决”**切换成**“已解决”**，此时门禁将显示为**“评审意见门禁已通过”**。

8.5.4 流水线门禁详解

说明

流水线门禁仅支持合入机制为**“审核机制”**的合并请求。

门禁的开启/关闭

步骤1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤2 单击**“新建”**，为目标分支设置分支策略。

步骤3 配置门禁。

- 勾选策略下的**“开启流水线门禁”**，单击**“确定”**保存设置，门禁开启。
- 取消勾选策略下的**“开启流水线门禁”**，单击**“确定”**保存设置，门禁关闭。

----结束

门禁触发的效果

- **合并请求流水线门禁已通过**：当最新 commit/预合并 commit 成功拉起流水线时显示。

  合并请求流水线门禁已通过 Pipeline #7043001 success for dee57f85

- **合并请求流水线门禁未通过**：当该仓库无关联的流水线任务，或者最新 commit/预合并 commit 未成功拉起流水线时显示。

  合并请求流水线门禁未通过

门禁的通过

步骤 1 单击菜单**“持续交付 > 流水线”**，进入流水线服务。

步骤 2 单击**“新建流水线”**，填写以下信息。

- 名称：自定义名称。
- 流水线源：选择**“Repo”**。
- 代码仓：选择需要创建合并请求的目标代码仓。
- 默认分支：选择合并请求的目标分支。

步骤 3 单击**“下一步”**，根据需求选择目标模板，单击**“确定”**。

步骤 4 任务创建成功后会自动跳转任务详情中的**“工作流”**页签，切换到**“执行计划”**页签。

步骤 5 开启合并请求下仓库的**“启用合并请求事件触发”**，根据实际情况勾选以下触发事件。

- 新建：合并请求创建时触发。
- 更新：合并请求内容或设置更新时触发。
- 合并：合并请求合入时触发，该事件会同时触发代码提交事件。
- 重新打开：合并请求重新打开时触发。

步骤 6 完成流水线任务其他信息配置，单击**“保存”**。

步骤 7 返回代码托管服务，触发**“执行计划”**中已勾选的事件让仓库拉起流水线任务即可。

----结束

8.5.5 E2E 单号关联门禁详解

门禁的开启/关闭

步骤 1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤 2 配置门禁。

- 勾选合入条件下的**“必须与 CodeArts Req 关联”**，单击**“提交”**保存设置，门禁开启。
- 取消勾选合入条件下的**“必须与 CodeArts Req 关联”**，单击**“提交”**保存设置，门禁关闭。

----结束

门禁触发的效果

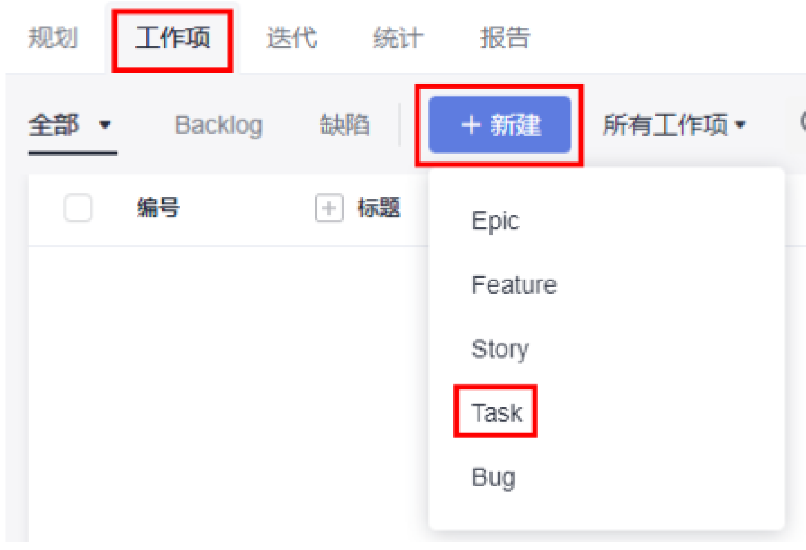
- **E2E 单号关联通过：**当合并请求成功关联工作项时显示。

- **E2E 单号未关联：**当合并请求无关联工作项时显示。

门禁的通过

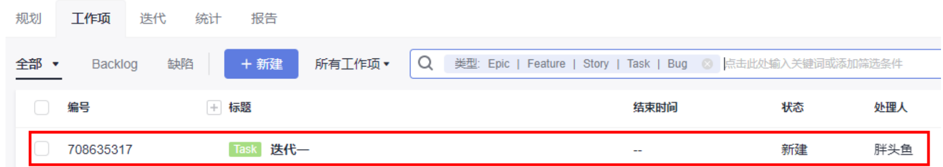
步骤 1 单击目标项目名称，进入项目。

步骤 2 在当前**“工作项”**页面，单击**“新建”**，在弹出的下拉框中选择“Task”，进入新建工作项页面。



步骤 3 填写标题，例如：迭代一。

其他参数默认即可，单击“保存”按钮进行保存。



步骤 4 单击菜单**“代码 > 代码托管”**，进入代码托管服务。

步骤 5 单击目标仓库名称，进入目标仓库。

步骤 6 切换到**“合并请求”**页签，单击目标合并请求名称，进入目标合并请求。

步骤 7 单击**“详情”**页中**“关联工作项”**旁的图标，搜索并选择目标工作项。

步骤 8 单击**“确定”**，完成 E2E 单号关联。

----结束

8.5.6 星级评价门禁详解

门禁的开启/关闭

步骤 1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤 2 配置门禁。

- 勾选合入条件下的**“是否将星级评价作为合入门禁”**，单击**“提交”**保存设置，门禁开启。
- 取消勾选合入条件下的**“是否将星级评价作为合入门禁”**，单击**“提交”**保存设置，门禁开启。

----结束

门禁触发的效果

- **星级评价已通过**：当指定人员已对合并请求进行星级评价时显示。

- **星级评价未通过**：当指定人员未对合并请求进行星级评价时显示。

门禁的通过

指定仓库角色的人员至少需要一个 Committer 以上权限在合并请求**“详情 > MR 评价”**中主星级进行星级评价，完成评价视为门禁通过。

说明

进入目标仓库，单击**“设置 > MR 评价”**。

- 勾选“启用 MR 自定义评价维度分类”，可添加评价维度（输入维度名称，按 Enter 键保存，最多新建 20 个，最多 200 个字符），单击**“提交”**保存设置。合并请求详情页面为多维度 MR 评价，包括主星级评价和其它维度评价。
- 取消勾选“启用 MR 自定义评价维度分类”，单击**“提交”**保存设置。合并请求详情页面只有主星级评价。

8.5.7 检视门禁详解

📖 **说明**

检视门禁仅支持合入机制为**“审核机制”**的合并请求。

门禁的开启/关闭

步骤 1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤 2 单击**“新建”**，为目标分支设置分支策略。

步骤 3 配置门禁。

- 配置策略下的**“最小检视人数”**不为 0，单击**“确定”**保存设置，门禁开启。
- 配置策略下的**“最小检视人数”**为 0，单击**“确定”**保存设置，门禁关闭。

----结束

门禁触发的效果

- **检视门禁已通过：**当检视通过人员数量达到**“最小检视人数”**时显示。

- **检视门禁未通过：**当检视通过人员数量未达到**“最小检视人数”**时显示。

门禁的通过

检视人员完成检视后，需在合并请求**“详情 > 检视门禁”**中，单击**“检视通过”**，视为检视通过。

8.5.8 审核门禁详解

说明

审核门禁仅支持合入机制为**“审核机制”**的合并请求。

门禁的开启/关闭

步骤 1 进入目标仓库，单击**“设置 > 策略设置 > 合并请求”**。

步骤 2 单击**“新建”**，为目标分支设置分支策略。

步骤 3 配置门禁。

- 配置策略下的**“最小审核人数”**不为0，单击**“确定”**保存设置，门禁开启。
- 配置策略下的**“最小审核人数”**为0，单击**“确定”**保存设置，门禁关闭。

----结束

门禁触发的效果

- **审核门禁已通过：**当审核通过人员数量达到**“最小审核人数”**时显示。

- **审核门禁未通过：**当审核通过人员数量未达到**“最小审核人数”**时显示。

门禁的通过

审核人员完成审核后，需在合并请求**“详情 > 审核门禁”**中，单击**“通过”**，视为审核通过。

8.6 查看仓库的评审记录

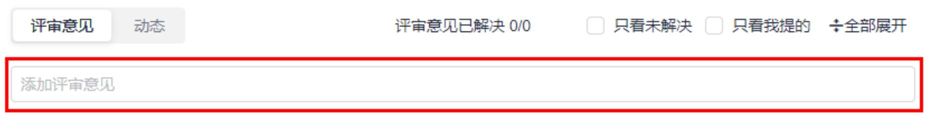
在仓库详情中的“评审记录”页签，可以查看仓库源自于合并请求与 Commit 的评审信息，可根据选择具体的筛选条件进行筛选记录。

表8-7 评审记录参数说明

参数项	参数说明
状态	评审记录分为“未解决”、“已解决”、“无需解决”三种状态。
评审意见	审核人提出的意见内容。
评审人	提出该评审意见的审核人。
评审日期	审核人提交评审意见的日期。
指派给	指派给系统默认人员或指定人员。

添加“源自合并请求的评审记录”方式

方式一：进入目标合并请求详情页，在页面最下方的位置即可添加评审意见。



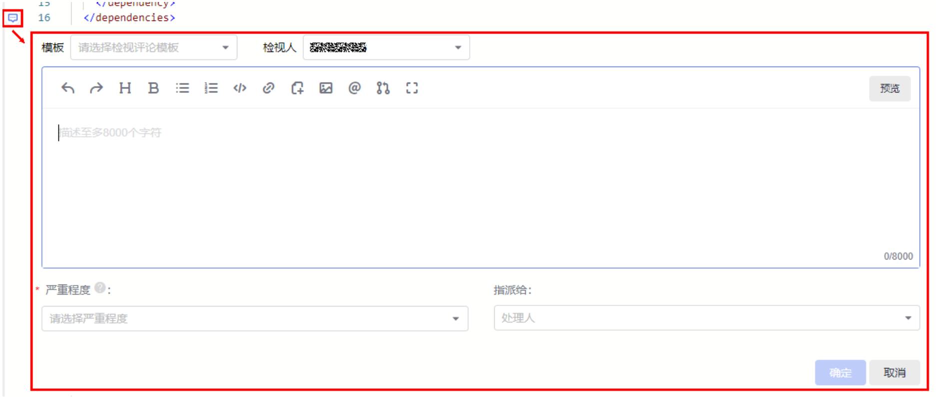
方式二：进入目标合并请求详情页，单击“**文件变更**”子菜单，在代码文件中，单击某行代码旁的 图标，即可添加检视意见。

说明

检视意见的输入框采用 markdown 格式，字符数限制在 8000 字符以内，即将超出上限时，使用顶部的操作按钮会按照 markdown 的语法替换内容。

添加“源自 Commit 的评审记录”方式

方式一：在代码文件中，单击某行代码旁的 图标，即可添加评审意见。

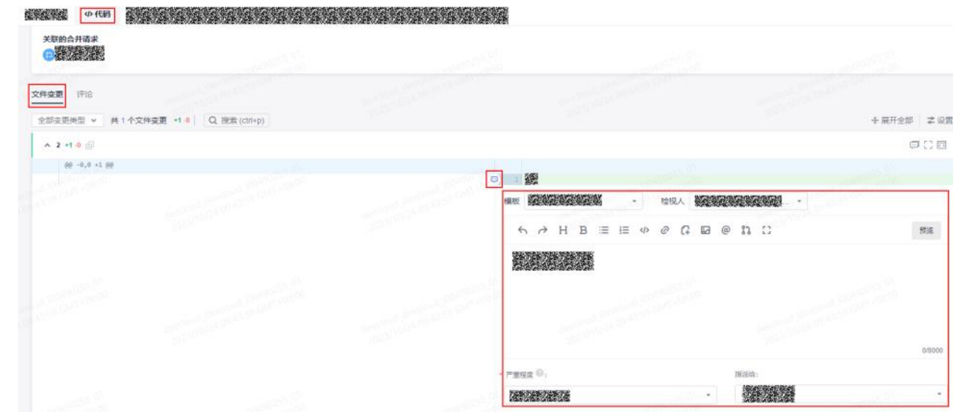


📖 **说明**

评审意见的输入框采用 markdown 格式，字符数限制在 8000 字符以内，即将超出上限时，使用顶部的操作按钮会按照 markdown 的语法替换内容。

方式二：在“**提交**”页面中，单击某个提交，切换“**评论**”界面，即可添加评审意见。

方式三：在“**提交**”页面中，单击“**文件变更**”子菜单，单击某行代码旁的 图标，即可添加检视意见。



8.7 查看关联工作项

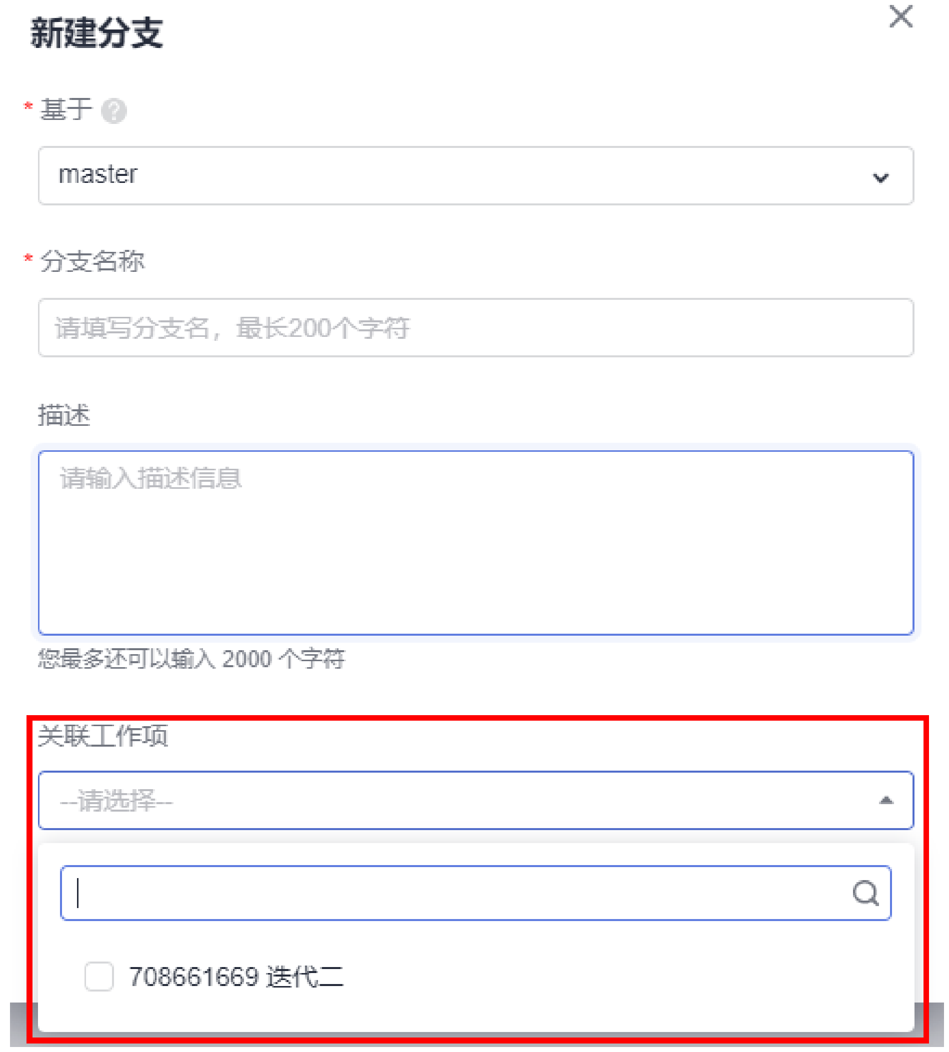
8.7.1 概述

工作项是需求管理中对工作内容的跟踪方法之一，通常有一个唯一的编号和描述信息，工作项可以是需求、缺陷和任务，在需求管理服务中，工作项是一种可以被可视化管理的工作内容清单。

代码托管服务支持以下三种关联。

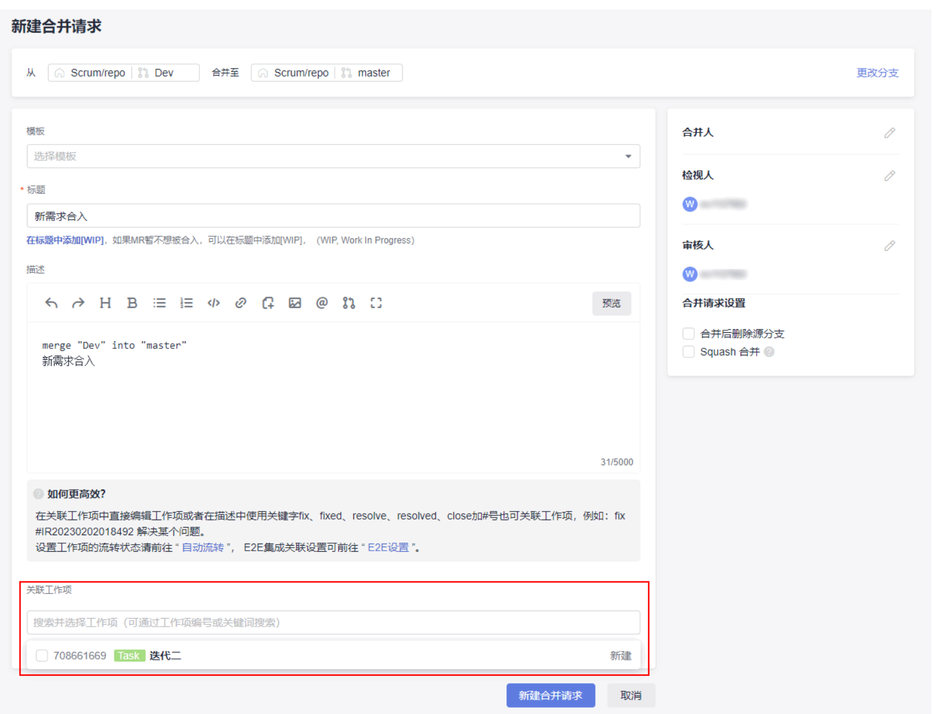
- Commit 关联。
- 新建分支关联。

  您可以在新建分支界面的**“关联工作项”**中选择目标工作项进行关联。



- 合并请求关联。

  您可以在新建合并请求界面的**“关联工作项”**中选择目标工作项进行关联。



说明

需求管理：同是软件开发生产线下的一个服务，其功能是为研发团队提供简单高效的团队协作服务，包含多项目管理、敏捷 Scrum、精益看板、需求管理、缺陷跟踪、Wiki 在线协作、文档托管、统计分析，工时管理等功能。

前置准备

步骤 1 （可选）配置代码提交流转状态。

说明

默认状态下，代码提交流转状态的默认设置如下：

- fix 关键字绑定在“已解决”目标状态（默认开启使用）。
- close 关键字绑定在“已关闭”目标状态（默认不开启使用）。
- resolve 关键字绑定在“已解决”目标状态（默认开启使用）。

在项目设置中，项目经理或其他拥有项目设置权限的角色，可以对不同的工作项类型（Epic、Feature、Story、Task、Bug）分别设置三个预设的提交信息关键字（fix、close、resolve），对于每个关键字可绑定目标状态（如：已解决、已关闭等，工作项状态支持自定义）。

下面以将 Task 工作项类型中的 close 关键字绑定在“已拒绝”为例，进行操作演示。

1. 单击目标项目名称，进入项目。
2. 参照下图，找到 Task 工作项类型所对应的代码提交流转状态。

3. 单击 close 的“目标状态”，将其设定为“已拒绝”，并将其的“是否使用”设置为 。此时将自动保存设置状态。

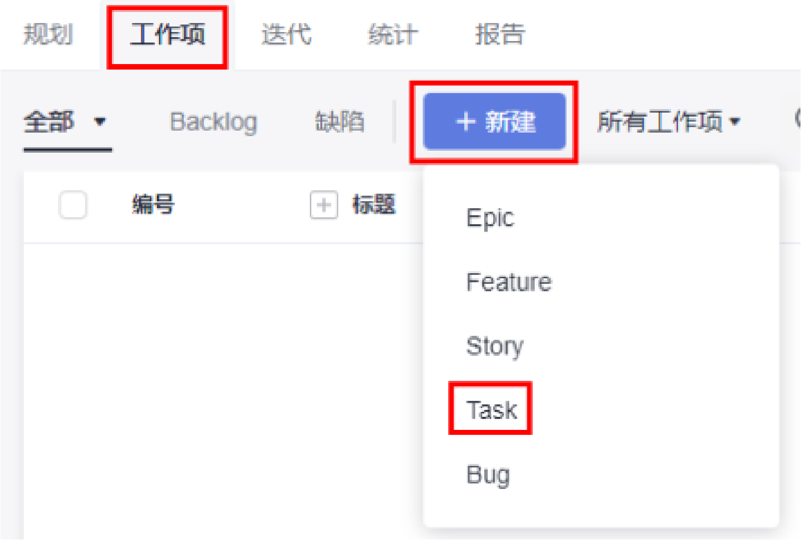
   设置完成后，当在本地提交代码时，可以在提交备注中用 close 关键字去改变 Task 类工作项的状态为“已拒绝”。

   示例如下：

```
git commit -m "close #Task的编号 本次提交的信息"
```

步骤 2 新建工作项。

1. 单击目标项目名称，进入项目。
2. 在当前**“工作项”**页面，单击**“新建”**，在弹出的下拉框中选择“Task”，进入新建工作项页面。



3. 填写标题，例如：迭代一。

   其他参数默认即可，单击“保存”按钮进行保存。

说明

此时系统会自动跳转回“工作项管理页面”，可以查看到工作项的 ID（编号），且状态都是“新建”。

本示例中：

- task01 的编号是 708206208。
- task02 的编号是 708206209。

获取工作项目编号的方式为项目“工作 > 工作项”。

----结束

8.7.2 Commit 关联

代码托管服务可以将每一次代码提交（commit）关联到需求管理的工作项中。

- 关联工作项可以帮助开发者精确记录每一次修复 bug、提交新特性时所对应工作任务。
- 关联工作项可以帮助项目管理者查看每一个需求、bug 修复时，所涉及修改内容的提交人、具体提交内容等信息。

说明

提交（commit）：可以将工作区的文件操作进行提交保存，有且不限于新建、编辑、删除的操作，其参数 -m 是必须的，后面跟的是**提交信息**。其应用格式如下：

```
git commit -m <本次的提交信息>
```

在代码托管控制台中，对文件的任何操作在保存时都会要求必须填写一个提交信息（或备注、每个页面用词不一样）才能保存，可以理解为控制台的每一次保存都是一次 commit 操作，其必填的提交信息对应了 commit 命令的-m 内容

代码托管服务以从-m（提交信息）中捕获关键字的方式来自动关联工作项，最常用的是“fix”关键字，这也是控制台操作提示中推荐的关键字，其使用时需要满足如下格式，才能被识别：

```
git commit -m "fix #工作项的编号 本次提交的信息"
```

在工作项关联成功时，系统会根据配置的代码提交流转状态来自动变更工作项的状态，默认情况下“fix”关键字会将工作项置于“已解决”状态。

例如下面的这次提交：

```
git commit -m "fix #123456 修复了这个 bug"
```

当其被推送到代码托管仓库时，会将编号为 123456 的工作项置于“已解决”状态。

代码托管服务同时支持您在本地、代码托管控制台的两种提交方式来关联工作项，下面分别对这两种方式进行操作说明。

说明

- 只有同是项目、仓库成员的账号才能进行工作项关联。
- 只有工作项的创建人、指定的修改人、或者在项目中有所有工作项修改权限的账号（比如“项目经理”）的关联操作才能改变工作项的状态（新建、已解决等）并生成评论记录，“关联结果”为“流转成功”。无权限账号进行操作时，只会产生关联记录，不会改变工作项的状态，也不会生成评论记录，“关联结果”为“关联成功”。

在本地提交代码并关联工作项

首先您需要在本地具备 Git 环境，详细请参考 Git 客户端安装与配置，在可以访问仓库时（已经关联到了对应的远程仓库），可以开始进行以下操作。

在本地的 master 分支上新建一个文件，将其推送到远程仓库，在推送时-m 里使用“fix”关键字去关联工作项 task01。

> **说明**
> - 本示例直接修改 master 分支，是为了缩短流程减少杂音让开发者更快的了解本地提交关联工作项的操作和原理。
> - 在实际代码开发中，尽量不要直接修改 master 分支，推荐新建一个分支进行文件操作，操作完后再合并到 master 分支并将 master 推送到远程仓库。（这是一种默认规则和良好习惯）

步骤 1 在本地仓库文件夹下单击鼠标右键，打开 Git Bash 客户端。

步骤 2 确认远程仓库地址绑定是否成功。

```
git remote -v          #该命令可以查看目前本地仓库所绑定的远程仓库地址。
```

如下图返回内容中，红线部分是本地仓所关联的远程仓库地址，地址之前是远程仓库在本地的别名。

```
Administrator@ecstest-paas MINGW64 ~/Desktop/02_developer/_0009
 (master)
$ git remote -v
origin  git@_000
9.git (fetch)
origin  git@_000
9.git (push)
```

如果发现绑定的仓库并非需要关联的仓库，或者没有绑定仓库，推荐直接将想绑定的仓库 Clone 到本地。

Clone 成功以后再次执行“git remote -v”查看确认绑定正确即可。

步骤 3 （上步骤已 Clone 的仓库可跨过此步）用 status 命令查看下目前仓库的状态，切换到 master 分支。

```
git status            #查看当前仓库状态，可以看到目前处于哪个分支、该分支有没有未暂存、未提交、未推送的修改
git checkout master    #切换到 master 分支，如果当前没有处于 master 分支时使用
```

步骤 4 在本地仓库文件夹下新建一个文件，本示例中将其命名为“fileFor708206208”。

步骤 5 在 Git Bash 中将新建的文件添加到暂存区。

```
git add fileFor708206208
```

步骤 6 在 Git Bash 中将本次操作提交。

```
git commit -m "fix #708206208 Task01"    #/本次提交用 fix 关键字关联了编号为 708206208 的 task01
```

> **说明**
> 708206208 是 task01 的编号。

步骤 7 在 Git Bash 将提交的内容推送到关联的代码托管仓库。

```
git push
```

如下图为推送成功，不同仓库结构返回会略有不同，只要看到所有步骤都 100%、done 就是推送成功了，如果推送失败一般是您的密钥问题。

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

步骤 8 验证关联结果。

上述操作完成后，进入工作项列表，找到编号为 708206208 的工作项，进入查看详情，如下图所示：

- 其状态已经置于“已解决”。
- 增加了一条关联的代码提交记录，单击提交编号可以前往查看提交详情。
- 增加了一条自动生成得评论以说明本次工作项关联。

----结束

在代码托管控制台提交代码并关联工作项

步骤 1 进入仓库详情页。

步骤 2 新建一个文件，如下图所示，在填写“提交信息”时以 fix #708206209 开头，其他信息任意即可。

说明

708206209 是 task02 的编号。

步骤 3 单击“确定”按钮，此时系统相当于在代码托管仓库上执行了以下操作：

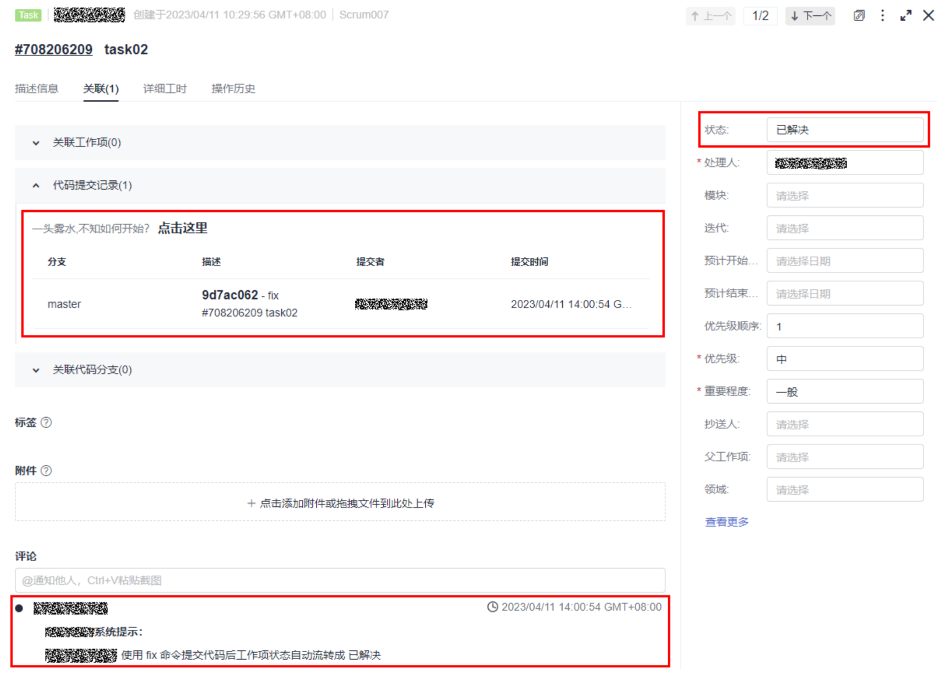
```
新建文件写入内容
git add .
git commit -m "fix #708206209 Task02"
```

也就是将一个新建的文件进行了一次 commit，并通过-m 参数中的“fix”关键字关联到了 task02 工作项。

步骤 4 验证。

此时您再去查看 task02 工作项时，如下图所示：

- 其状态已经置于“已解决”。
- 增加了一条关联得代码提交记录，单击提交编号可以前往查看提交详情。
- 增加了一条自动生成的评论以说明本次工作项关联。



----结束

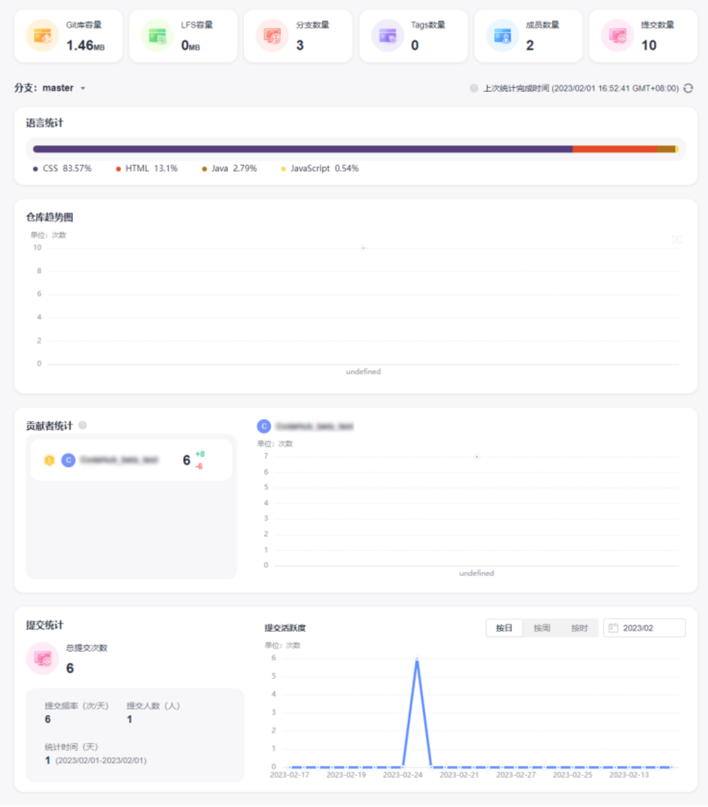
8.8 查看仓库的统计信息

在仓库详情中的“仓库统计”页签，可以查看仓库的相关统计信息，详情如下：

- 仓库信息概要：主要显示 Git 库容量、LFS 容量、分支数量、Tags 数量、成员数量、提交数量。可选择分支，对仓库趋势图、贡献者统计、提交统计的统计范围产生影响。（不会影响仓库信息概要）
- 语言统计：显示仓库当前分支的各语言分布情况。
- 仓库趋势图：显示仓库当前分支的提交分布情况。
- 贡献者统计：统计当前分支中代码提交者们的贡献度（提交次数、代码行数）。
- 提交统计：按不同维度（每周、每天、每小时）统计代码提交活跃度。

说明

- 仓库成员可以触发代码贡献度统计与语言比例统计。
- 因资源限制，每个仓库一天可以统计 10 次。
- 每个用户一天可以统计 1000 次。
- 统计完成，将显示每一位用户在截止时间之前的全部新增、删除的代码行数量（“+”表示新增，“-”表示删除）。
- merge（将两个或两个以上的开发历史合并在一起的操作）节点的提交均不被统计。



8.9 查看仓库的动态

在仓库详情中的“动态”页签，可以查看截止当前仓库的全部动态。

- **全部**：展示截止当前该仓库的所有操作记录。
- **推送**：展示截至当前该仓库所有的推送操作记录，例如推送代码、新建/删除分支等。
- **合并请求**：展示截至当前该仓库所有合并请求的操作记录，单击合并请求的序号可查看详情，例如新建/关闭/重开/合入合并请求等。
- **检视意见**：展示截至当前该仓库所有检视意见记录，单击提交号可查看详情，例如添加/删除检视意见等。
- **成员**：展示截至当前该仓库所有成员的管理记录，例如添加/移除成员、编辑成员权限等。

说明

- 展示内容为操作者、具体的操作内容及操作时间。
- 支持选择时间范围、操作人等条件进行筛选查询。

8.10 管理仓库成员

8.10.1 IAM 用户、项目成员与仓库成员的关系

仓库成员来源于其所属项目的项目成员，项目成员主要来源于租户的 IAM 用户，除项目创建者所在租户外，还可以邀请其它租户下的 IAM 账号加入项目。如下图为 IAM 用户、项目成员、仓库成员的包含关系示意图。

表8-8 项目角色与仓库角色对应关系

项目中的角色	仓库中的角色
项目经理	项目经理（默认）
开发人员	开发人员（默认）
系统工程师	系统工程师（默认）
Committer	Committer（默认）
产品经理	产品经理（默认）
测试经理	测试经理（默认）
测试人员	测试人员（默认）
参与者	参与者（默认）
浏览者	浏览者（默认）
运维经理	运维经理（默认）
自定义角色	自定义角色（默认）

8.10.2 配置成员管理

仓库成员管理功能位于仓库详情的“成员”页签。只有仓库所有者、仓库管理员才能对仓库人员进行变动，其他人员只能浏览仓库成员列表，以下是编辑成员管理的操作流程。

说明

代码托管目前仅支持将项目成员导入为仓库成员，添加项目成员或修改项目成员角色，请参考需求管理服务《用户指南》中“设置 > 成员管理”章节。

手动添加项目成员到仓库

单击“添加成员”，弹出添加成员页面，可以从仓库所在项目的成员列表中选择成员加入仓库中，会根据项目角色赋予其**默认仓库角色**，其对应关系如下表所示。

表8-9 项目下成员加入到仓库时的对应角色关系

项目中的角色	仓库中的角色
项目经理	项目经理（默认）
Committer	Committer（默认）
系统工程师	系统工程师（默认）
开发人员	开发人员（默认）
产品经理	产品经理（默认）
测试经理	测试经理（默认）
测试人员	测试人员（默认）
参与者	参与者（默认）
项目自定义角色	项目自定义角色（默认）
运维经理	运维经理（默认）
浏览者	浏览者（默认）
自定义角色	自定义角色（默认）

说明

- 在成员列表中，所有成员均可设置为项目角色中的任意一种角色，且均可被移出仓库。
- 如果仓库级“添加成员”列表为空，则说明此仓库下没有除仓库所有者之外的成员，请添加项目成员。

9 配置代码托管仓库

基本设置

仓库管理

策略设置

服务集成

模板管理

安全管理

9.1 基本设置

9.1.1 仓库信息

仓库信息可在仓库详情的“设置 > 基本设置 > 仓库信息”查看和修改。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

仓库描述是在模板开源（公开示例模板）时的备注字段，用途是便于查找。

可见范围

- **私有**：仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码。
- **公开只读**：仓库对所有访客公开只读，但不出现在访客的仓库列表及搜索中。
- **公开示例模板**：仓库公开只读，并成为全站都能使用的代码模板，需要填写模板的分享标题及作者。

仓库信息

仓库名称

task

仓库描述

task

可见范围：

私有 公开只读 公开示例模板

仓库仅对仓库成员可见，仓库成员可访问仓库或者提交代码

提交

9.1.2 通知设置

代码托管服务内的通知设置

通知设置位于仓库详情中的 “设置 > 基本设置 > 通知设置”。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

邮件通知

说明

如果已关闭通知设置中所有的通知类型，在进行以下操作时，系统会默认发送相应的邮件通知给创建者或管理员。

- 新建仓库时，默认给项目管理员发送邮件通知。
- 非仓库成员申请加入仓库，默认给仓库所有者发送邮件通知。
- 冻结/关闭仓库时，默认给仓库所有者/项目管理员发送邮件通知。

通知设置

仓库删除事件发生时，可选择设置邮件通知仓库不同类型的成员或选择不发送邮件。

邮件

冻结仓库: 所有者 管理员 注：系统设置，不可修改。

关闭仓库: 所有者 管理员 注：系统设置，不可修改。

合并请求: 合并人 评审人 注：系统设置，不可修改。

删除仓库: 所有者 管理员 Committer 开发者 浏览者 不发送

容量预警: 所有者 管理员 Committer 开发者 阈值: 90%

确定

- **冻结仓库：**默认给仓库所有者、项目管理员发送邮件通知，不可手动设置。

  当出现关闭服务或仓库欠费时，仓库会被冻结，冻结的仓库不能进行任何操作。

  在仓库冻结后的 30 天内，进行仓库续费或开启服务等操作，即可解除仓库冻结状态。
- **关闭仓库：**默认给仓库所有者、项目管理员发送邮件通知，不可手动设置。

  仓库关闭等同于被彻底删除，当仓库被冻结时间超过 30 天，仓库将会关闭。
- **删除仓库：**可手动设置给仓库所有者、项目管理员、项目经理、Committer、开发人员、浏览者等发送邮件通知。
- **容量预警：**默认阈值为 90%，您可根据需要手动设置容量预警阈值。当单仓容量超过所设置的阈值时，系统会给仓库所有者、项目管理员、项目经理、Committer 及开发人员等发送预警邮件通知。预警邮件只发送一次，除非您更新预警设置。
- **开启合并请求：**当合并请求开启时（包括新建和重开合并请求），会推送开启邮件。默认给评审人、审核人、检视人、合并人发送邮件通知，也可手动设置不发送。
- **更新合并请求：**当更新合并请求关联分支的代码时，会推送更新邮件。默认给评审人、审核人、检视人发送邮件通知，也可手动设置不发送。
- **合并合并请求：**默认给 MR 创建人发送邮件通知，可手动设置给合并人发送邮件通知或均不发送通知。
- **检视合并请求：**默认给 MR 创建人发送邮件通知，可手动设置不发送通知。
- **审核合并请求：**默认给 MR 创建人发送邮件通知，可手动设置不发送通知。
- **新建评审意见：**默认给 MR 创建人发送邮件通知，可手动设置不发送通知。
- **解决评审意见：**默认给 MR 创建人发送邮件通知，可手动设置不发送通知。

说明

如果没有收到相关的所以无法使用 HTTPS 协议邮件通知，请前往消息设置，查看邮件通知设置是否开启。

如果您想用发送邮件以外的方式知晓仓库的变化，可通过配置“**服务集成**”中的“**Webhook 设置**”并在自己系统（第三方系统）中自定义实现。

软件开发生产线的消息设置

软件开发生产线提供可配置的消息提醒功能，在软件开发生产线首页，单击右上角您的用户名，在弹出框中选择“个人设置”即可进行消息设置。

在**“通用设置 > 消息设置”**页面中，您可以开启/关闭来自于邮件的通知，并可以更改您接收消息邮件的邮箱。

您也可以设置勿扰时段，以在特定的时间段不接受消息提醒。

消息设置

勿扰时段设置:

开启后，在设定时间段内不会收到新的消息。

邮件通知

接收通知的邮箱 .com 更改设置

开启

关闭

9.2 仓库管理

9.2.1 仓库设置

仓库设置位于仓库详情中的 “设置 > 仓库管理 > 仓库设置”。

默认分支会作为进入本仓库时，默认选中的分支，也会作为创建合并请求时，默认的目标分支。仓库新建时，master 分支将被作为默认分支，可以随时手动调整。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

表9-1 参数说明

参数项	说明
禁止 Fork 仓库	默认不勾选，勾选后，该仓库会禁止所有人 Fork 仓库。
开启开发人员创建分支权限白名单	默认不勾选，勾选后，开启开发人员创建分支权限白名单。 **说明** 只能展示和配置仓库角色为开发人员的仓库成员进入此白名单，非开发人员不展示且配置后不生效。
MR 预合并	默认不勾选，勾选后，服务端会自动生成 MR 预合并的代码，相比客户端使用命令做预合并操作更高效简洁、构建结果更准确，适用于对构建实时性要求严格的场景。

说明

- 字节（byte）：指一小组相邻的二进制数码，是计算机重要的数据单位，通常用大写 B 表示，1B（byte）= 8bit（位）。
- 字符：表示数据和信息的字母、数字或其他符号。

配置“MR 预合并”

当 MR 创建后，您可自定义 WebHook、流水线等下载插件的脚本，即下载代码内容可以由您自己控制。

- 如果勾选“**MR 预合并**”，则服务端会帮助您生成一个隐藏分支，表示该 MR 代码已经合入，进而您可以直接下载已经存在在隐藏分支的代码。
- 如果未勾选“**MR 预合并**”，您需要在客户端本地做预合并，即分别下载 MR 源分支、MR 目标分支的代码，并在构建执行机自己做合并动作。

操作命令

服务端预合并命令如下：

```
git init
git remote add origin ${repo_url克隆/下载地址}
git fetch origin +refs/merge-requests/${repo_MR_iid}/merge:refs/${repo_MR_iid}merge
```

如果未勾选，则可以通过客户端做预合并操作，本地新建干净的工作目录，命令如下：

```
git init
git remote add origin ${repo_url克隆/下载地址}
git fetch origin
+refs/heads/${repoTargetBranch}:refs/remotes/origin/${repoTargetBranch}
git checkout ${repoTargetBranch}
git fetch origin +refs/merge-
requests/${repo_MR_iid}/head:refs/remotes/origin/${repo_MR_iid}/head
git merge refs/remotes/origin/${repo_MR_iid}/head --no-edit
```

功能优势

对于构建实时性要求高的场景，如：一个MR可能拉起几十或上百台服务器的构建，本地/客户端做预合并可能会与服务端产生的结果不一致，导致构建代码获取不够准确、构建结果不准确等问题。使用服务端预合并可以解决该实时性问题，并且构建脚本命令更简单，开发人员或CIE更好上手。

9.2.2 仓库加速

仓库加速位于仓库详情中的 “设置 > 仓库管理 > 仓库加速”。

仓库加速是针对当前仓库运行后台进行整理任务，压缩文件并移除不再使用的对象，等同于Git中的gc（garbage collect）功能，也就是垃圾回收功能，该功能可以降低您的仓库空间占用，提升读写仓库的效率。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

说明

频繁单击加速操作，并不能持续加速，建议1个月左右执行一次。

9.2.3 同步设置

同步设置位于仓库详情中的 “设置 > 仓库管理 > 同步设置”。

该功能是将当前仓库设置自定义同步至其他仓库。该功能支持跨项目同步，暂时不支持跨地域同步。

一般推荐用于基于该仓库Fork出的仓库，因为Fork仓库时虽然会复制其所有分支和文件内容，但并不会自动复制仓库设置。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考权限管理页面。

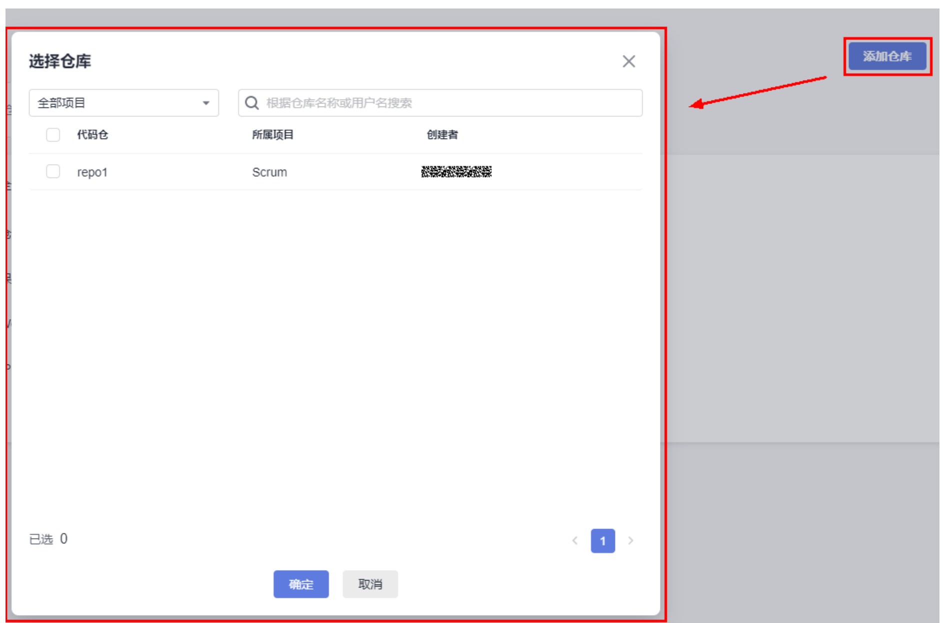
添加同步仓库

须知

同步仓库需保证网络连通。

- 对于公开平台，代码托管服务支持访问代码仓库。
- 对于连接内网私有仓库平台，用户需自行保证 CodeArts Repo 到用户仓库的网络畅通。

步骤 1 单击“添加仓库”，在弹框中选择目标仓库。



步骤 2 单击**“确定”**，完成仓库同步。

----结束

说明

常见的同步失败原因：

- **“提交规则”**同步失败：一般是因为源仓库没有设置提交规则。
- **“保护分支”**同步失败：一般是因为源仓库与目标仓库的分支命名不一样。

9.2.4 子模块设置

背景信息

子模块（submodule）是 Git 为管理仓库共用而衍生出的一个工具，通过子模块您可以将公共仓库作为子目录包含到您的仓库中，并能够双向同步该公共仓库的代码，借助子模块您能将公共仓库隔离、复用，能随时拉取最新代码以及对它提交修复，能大大提高您的团队效率。

有种情况经常会遇到：某个工作中的项目 A 需要包含并使用项目 B（第三方库，或者你独立开发的，用于多个父项目的库），如果想要把它们当做两个独立的项目，同时又想在项目 A 中使用项目 B，可以使用 Git 的子模块功能。 子模块允许您将一个 Git 仓库作为另一个 Git 仓库的子目录。 它能让你将另一个仓库克隆到自己的项目中，同时还保持提交的独立。

子模块将被记录在一个名叫“.gitmodules”的文件中，其中会记录子模块的信息：

```
[submodule "module_name"]      #子模块名称
path = file_path              #子模块在本仓库（父仓）中文件的存储路径。
url = repo_url               #子模块（子仓库）的远程仓地址
```

这时，位于“file_path”目录下的源代码，将会来自“repo_url”。

控制台操作

- **控制台添加子模块**
  - **入口一：**

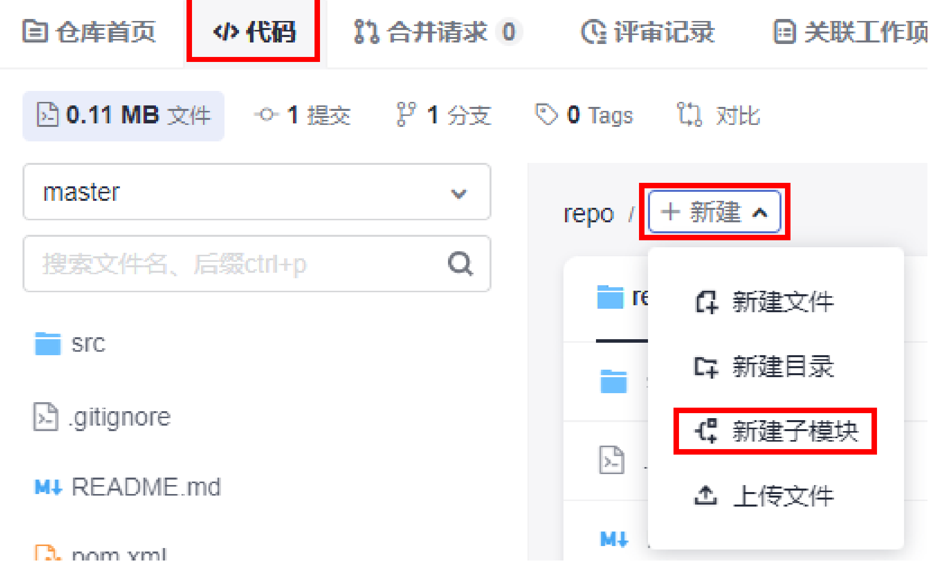
    可以在仓库文件列表中的某个文件夹下添加子模块。

    单击扩展按钮 ⋮，选择“新建子模块”，如下图所示。

- 入口二：

  可以在仓库的**"代码"**页签中，添加子模块。

  单击扩展按钮 + 新建 ∨ ，选择"新建子模块"，如下图所示。



- 入口三：

  可以在仓库设置中，为仓库创建子模块。

  其操作路径为"设置 > 仓库管理 > 子模块设置 > 新建子模块"。

- **填写说明：**

  使用以上三种方法均可进入"新建子模块"页面。

请参考下表填写，完成后单击“确定”按钮，即可完成新建子仓库操作。

表9-2 新建子模块—字段说明

字段	填写说明
子模块仓库路径	选择一个仓库作为子仓库。
子模块仓库分支	选择同步子仓库的目标分支到父仓库。
子模块文件路径	配置子模块文件在本仓库下的路径，注意用“/”分割层级。
提交信息	作为您新建子仓库的备注信息，可以在文件历史中查找到本次操作，限制 2000 个字符。

说明

子模块新建完成后，可以在仓库文件列表的对应目录内找到子模块（子仓库）内容，其对应的文件左侧图标为 。

- **控制台查询子模块状态、同步、删除子模块**

  管理员可以通过查看“设置”页面下的“子模块设置”页面，查看子模块状态，同步子模块，删除子模块。

- **控制台同步部署密钥**

  对于客户端提交的子模块，需要仓库管理员在“设置”页面下的“子模块设置”页面，将父仓库的部署密钥同步到子仓库中，从而保证在构建父仓库时，可以将对应提交的子仓库一同拉取下来。

Git 客户端操作

步骤 1 **添加 Submodule。**

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

示例：

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

步骤 2 **拉取包含 submodule 的仓库。**

```
git clone <repo> [<dir>] --recursive
```

示例：

```
git clone git@***.***.com:****/WEB-INF.git --recursive
```

步骤 3 **获取远端 Submodule 更新。**

```
git submodule update --remote
```

步骤 4 **推送更新到子库。**

```
git push --recurse-submodules=check
```

步骤 5 删除 Submodule。

1. 删除“.gitsubmodule”中对应 submodule 的条目。
2. 删除“.git/config”中对应 submodule 的条目。
3. 执行命令，删除子模块对应的文件夹。

```
git rm --cached {submodule_path}    #注意更换为您的子模块路径
```

说明

注意：路径不要加后面的“/”。

示例：你的 submodule 保存在“src/main/webapp/WEB-INF/”目录，则执行命令为：

```
git rm --cached src/main/webapp/WEB-INF
```

----结束

9.2.5 仓库备份

异地备份位于仓库详情中的 “设置 > 仓库管理 > 仓库备份”。

仓库的备份操作分为两种备份形式：

- **备份到异地：**将仓库备份到其它区域。

  其本质是一次导入外部仓库，将一个区域的仓库备份到另一个区域中。

- **备份到本地：**将仓库备份到您本地计算机。

  可使用 HTTPS、SSH 两种 clone 形式，如下图会生成 clone 命令，只要粘贴进本地 Git 客户端并执行即可。（需要保证仓库连通性）

  仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考权限管理页面。

9.2.6 同步仓库

只有通过导入外部仓库方式创建的仓库，才会出现**“同步仓库”**设置项。

同步仓库位于仓库详情中的 “设置 > 仓库管理 > 同步仓库”。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

单击“同步仓库”按钮可以重新同步源仓库的默认分支。如果在配置导入外部仓库界面时，勾选“定时同步导入仓库”选项，就会在**“同步仓库”**设置中出现**“定时同步镜像仓”**的开关，如下图所示。

- 打开**“定时同步镜像仓”**功能时，代码仓库为只读状态，无法提交/上传代码。同步仓库将每小时刷新一次，刷新内容为源仓库 24 小时前的内容，例如：您今天 10 时对源仓库的默认分支进行了修改，明天 10 时修改的内容才会被同步到镜像仓。
- 关闭**“定时同步镜像仓”**功能时，代码仓库为可编辑状态，该功能从页面移除，无法恢复。

同步仓库
源仓地址：https://[illegible].git
源分支：全部分支
同步分支：默认分支
同步仓库
点击开始手动同步代码仓库。警告：同步远端的仓库，可能会覆盖本地已经提交的代码，导致代码丢失！
定时同步镜像仓： 关闭
打开定时同步功能时，代码仓库为只读状态，无法提交/上传代码。关闭自动同步后，代码仓库可写，但是无法再次恢复自动同步。

须知

- 同步仓库只针对默认分支生效，如需更新其他分支代码，需要在仓库设置中手动更改默认分支后才能有效。
- 将源仓地址的内容，同步到本仓库，可能会覆盖本仓库已经提交的代码，导致代码丢失。

9.3 策略设置

9.3.1 检视意见

检视意见位于仓库详情中的 “设置 > 策略设置 > 检视意见”。检视意见设置用于规范检视评论及配置检视评论模板。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

设置检视意见

步骤 1 根据需要选择是否勾选**“启用检视意见分类与模块”**启用检视意见。

步骤 2 配置检视意见分类。

- 启用系统预置检视意见分类

  勾选**“启用系统预置检视意见分类”**，可直接使用系统预置检视意见分类。
- 自定义分类

  支持自定义检视意见分类，输入类型名称，按 Enter 键保存。

说明

请输入分类名称，按 Enter 键结束；名称中不可包含英文冒号，可用英文逗号分隔，上限为 200 字符；个数上限为 20 ,不可重复。

步骤 3 在**“检视意见模块设置”**下输入框输入**“类型名称”**。

说明

请输入模块名称，按 Enter 键结束；上限为 200 字符；可用英文逗号分隔，个数上限为 20，不可重复。

步骤 4 根据自己的需要勾选**“启用新建/编辑检视意见时必填字段校验”**。

步骤 5 单击**“提交”**，保存设置。

----结束

9.4 服务集成

9.4.1 E2E 设置

E2E 设置可以帮助您记录每次代码合入的原因，开发了一个需求，修复了一个问题单，或者完成了一个工作项，Repo 系统将记录关联信息方便日后追溯。Repo 系统已默认设置了可关联。

可集成系统

与 CodeArts Req 系统集成，使用 CodeArts Req 的工作项关联对应代码提交。

集成策略

可选枚举值，用于限定用户在关联工作项时的选择条件。

排除状态：标识哪些状态的工作项不能关联合并请求。

可关联类别：允许关联哪些类型的工作项单。

应用分支：限定分支遵循以上限制条件，其他分支无限制。

自动提取单号规则

自动提取单号规则（根据代码提交信息自动提取单号），配置规则具体如下：

- **单号前缀**：非必填项，支持多个前缀，最多 10 个，如 **“【问题单号 or 需求单号】”**。

  说明

  如果单号前缀、分隔符、单号后缀规则不为空，则默认开启自动单号提取功能

- **分隔符**：非必填项，默认为 **“;”**。
- **单号后缀**：非必填项，默认为换行符。

  说明

  - 前缀、分隔符、后缀不能相同。
  - 分隔符为空时，前缀和后缀不能为“;;”。
  - 后缀为空时，前缀和分隔符不能为\n。
  - 前缀、分隔符、后缀为全字符匹配，不支持正则表达式。

示例

步骤 1 配置 E2E 设置。

1. 进入目标仓库。
2. 单击 **“设置 > 服务集成 > E2E 设置”**，切换到 **“E2E 设置”** 页面。

3. 配置以下集成策略，单击 **“提交”** 。

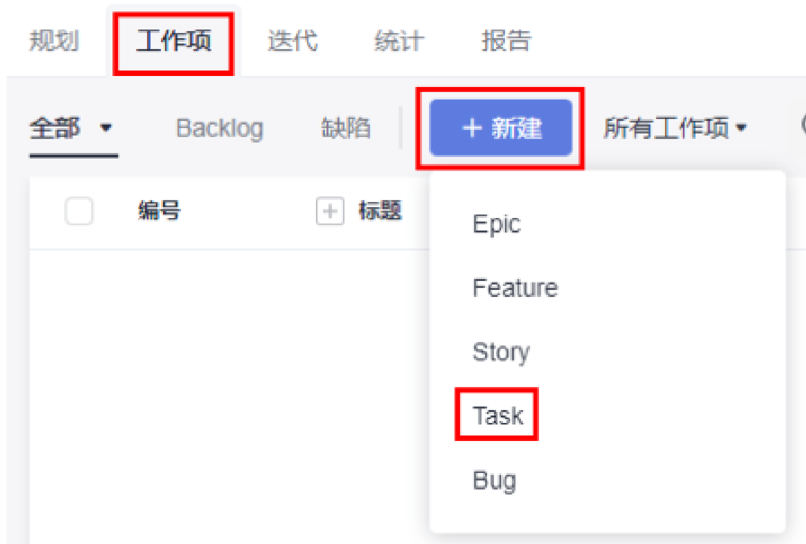
   应用分支：选择目标分支，例如：Dev。

单号前缀：自定义单号前缀，例如：“合入需求:”。

集成策略
排除状态 以下状态的工作项不可关联
点击此处添加状态
可关联类别 允许关联的工作项类型 e.g. Story/Task/Bug
点击此处添加工作类型
应用分支 限定分支遵循以上限制条件，其他分支无限制
Dev
自动提取单号规则
单号前缀
合入需求:
请输入前缀，支持多个...
分隔符
请输入分隔符，默认使用','
单号后缀
请输入后缀，默认使用换行
提交

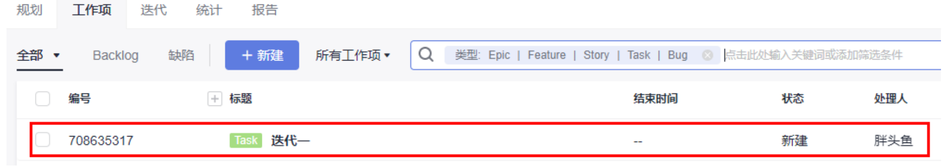
步骤 2 创建工作项。

1. 单击目标项目名称，进入项目。
2. 在当前“工作项”页面，单击 **“新建”**，在弹出的下拉框中选择“Task”，进入新建工作项页面。



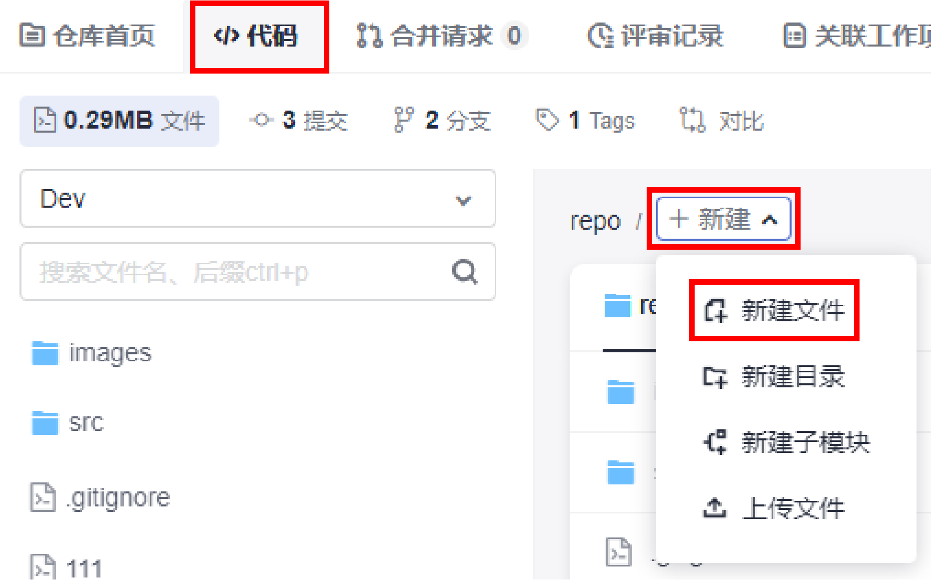
3. 填写标题，例如：迭代一。

   其他参数默认即可，单击“保存”按钮进行保存。



步骤 3 新建文件。

1. 进入代码托管仓库列表页，单击目标仓库名称，进入仓库。
2. 在 **“代码”** 页签下，单击 **“新建”**，在弹出的下拉框中选择 **“新建文件”**，进入新建文件页面。



3. 填写以下信息，其余参数默认即可，单击**“确定”**完成文件的新建。

   文件名：自定义文件名称，例如：示例代码。

   文件内容：自定义文件内容。

   提交信息：填写 E2E 设置中的前缀及工作项的单号，例如：合入需求：708635317。

新建文件

示例代码

```
1  /**
2   * Hello world
3   *
4   */
5
6  public class HelloWorld {
7
8    public static void main(String[] args) {
9      System.out.println("Hello World!");
10   }
11
12 }
13
```

提交信息

合入需求：708635317

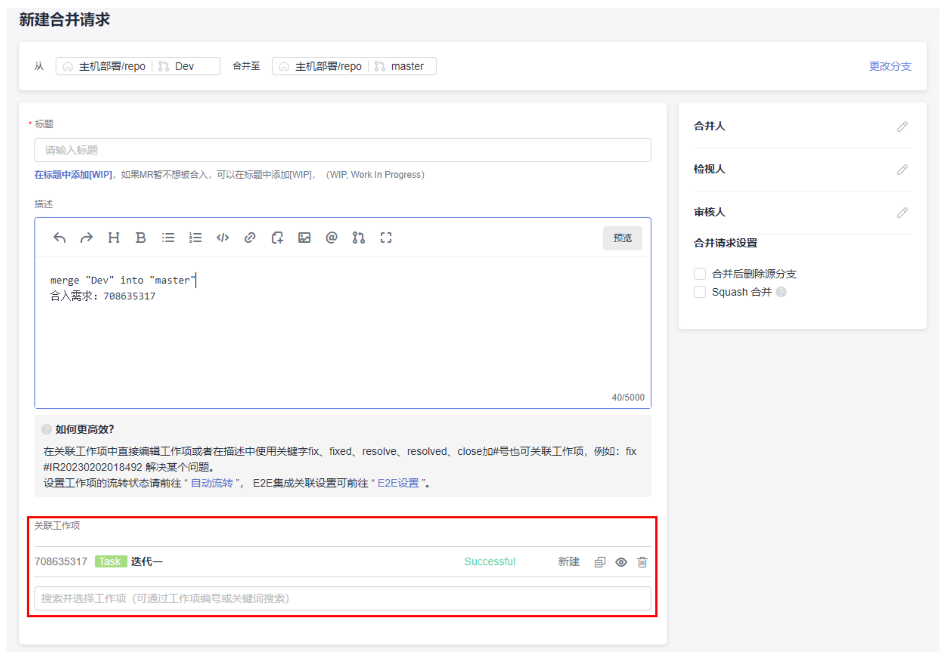
您最多还可以输入 1986 个字符

确定　取消

步骤 4 新建合并请求时提取单号。

1. 切换为**“合并请求”**页签，单击**“新建”**。
2. 选择源分支为**“Dev”**，目标分支为**“master”**，单击**“下一步”，进入新建合并请求界面**。

   此时，工作项被自动提取至该合并请求。



----结束

9.4.2 Webhook 设置

Webhook 简介

开发人员可在 Webhook 界面配置第三方系统的 URL，并根据项目需求订阅代码托管仓库的分支推送(push)、标签推送(tag push)等事件。当订阅事件发生时，可通过 Webhook 向第三方系统的 URL 发送 POST 请求，用以触发自己系统（第三方系统）的相关操作，例如：触发自己系统（第三方系统）界面的通知弹窗；或触发自己系统（第三方系统）的构建、更新镜像、部署等操作。

如果您需要用发送邮件作为仓库变化的通知方式，可通过配置**“基本设置”**中的**“通知设置”**实现。

Webhook 设置

Webhook 设置位于仓库详情中的 “设置 ＞ 服务集成 ＞Webhook 设置”。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考权限管理页面。

表9-3 新建 Webhook 字段说明

字段	说明
名称	可自定义名称。
描述	用于描述该 WebHook。
URL	必填项。WebHook URL 需第三方 CI/CD 系统提供。
Token 类型	用于第三方服务 WebHook 接口鉴权，分为以下三项： ● X-Repo-Token ● X-Gitlab-Token ● X-Auth-Token
Token	用于第三方 CI/CD 系统鉴权，鉴权信息放在 http 请求 header。
事件类型	系统可订阅以下事件： ● 推送事件 – 如果勾选推送事件，则出现**分支过滤正则规则**。 说明 **分支过滤正则规则**，默认为.*，代表全部分支，长度上限不超过 500 字符。 **分支过滤正则规则**需符合。 – 在代码托管仓库进行代码更新，如 LFS 文件代码更新、子模块中代码更新、在线或本地 Git 客户端中推送代码更新均会触发该事件。 ● Tag 推送事件 在代码托管仓库新建或删除 Tag 会触发该事件。 ● 合并请求事件 – 在代码托管仓库新建合并请求会触发该事件。 – 在代码托管仓库更新合并请求会触发该事件。如更新代码内容/更新合并请求状态（关闭、重开）/更新合并请求标题或描述/更新合并人/更新工作项/删除源分支/更新 Squash 合并。 – 在代码托管仓库合入合并请求会触发该事件。 ● 评论事件 – 在代码托管仓库添加检视意见会触发该事件。如在代码文件中添加检视意见、在提交详情文件变更下添加检视意见、在合并请求文件变更中添加检视意见。 – 在代码托管仓库提交详情和在合并请求详情中添加评论会触发事件。

说明

- 每个仓库最多只能设置 20 个 Webhook。
- 您在配置 Webhook 的时候，还可以选择设置您的 Token，该 Token 会与您的 Webhook URL 关联，系统会将该 Token 放在请求头的 X-Repo-Token 字段发送给您。

9.5 模板管理

9.5.1 合并请求模板

合并请求模板位于仓库详情中的 “设置 > 模板管理 > 合并请求模板”。当创建合并请求时，您可以选择一个合并请求模板，模板内容将会自动应用到合并请求上。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

新建合并请求模板

表9-4 字段说明

字段	说明
模板名称	必填项，新建模板的名称。
设置为默认模板	非必填项，勾选后，创建 MR 时默认应用此模板。
自动提取合并请求标题	非必填项，可选以下类型： • 不提取。 • 提取提交信息。 • 提取 e2e 单标题。
标题	非必填项，当**“自动提取合并请求标题”**选择**“不提取”**时，标题可自定义。
描述	非必填项，填写模板的描述信息，限制 5000 个字符。

9.5.2 检视评论模板

检视评论模板位于仓库详情中的 “设置 > 模板管理 > 检视评论模板”。您可新建、编辑和删除检视意见模板，并根据自身习惯定制检视意见模板信息，包括：严重程度、指派给、意见分类、意见模板和描述。当仓库成员进行检视评论时，您可选择一个检视评论模板，模板内容将会自动应用到合并请求上。

此设置只针对被设置的仓库生效。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

新建检视评论模板

表9-5 字段说明

字段	说明
模板名称	必填项，新建模板的名称。
设置为默认模板	非必填项，勾选后，进行检视评论时默认应用此模板。
严重程度	非必填项，根据问题的严重程度可分为“致命、严重、一般、建议”四种类型。

指派给	非必填项。
	• 指派给为**“空”**。
	  – MR 中添加检视意见时，默认指派给 MR 创建者。
	  – 文件或 Commit 中添加检视意见时，默认指派给为空。
	• 指派给为**“MR 创建者/Commit 作者”**。
	  – MR 中添加检视意见时，默认指派给 MR 创建者。
	  – 文件或 Commit 中添加检视意见时，默认指派给 Commit 作者。
	• 指派给为**“具体人员”**。
	  – MR 中添加检视意见时，默认指派给具体人员。
	  – 文件或 Commit 中添加检视意见时，默认指派给具体人员。

意见分类	非必填项，默认禁用，需**“启用检视意见分类与模块”**并配置检视意见分类才可设置，详情请参考检视意见。
意见模块	非必填项，默认禁用，需**“启用检视意见分类与模块”**后配置检视意见模块才可设置，详情请参考检视意见。
描述	非必填项，填写模板的描述信息，支持描述信息预览。

9.6 安全管理

9.6.1 部署密钥

部署密钥是您本地生成的SSH 密钥的公钥，但仓库的部署密钥和 SSH 密钥不能配置成一个。当配置了部署密钥，代码托管服务将允许您通过 SSH 协议以只读的方式克隆仓库，主要在仓库部署、持续集成等场景中使用。

说明

- 多个仓库之间可以使用同一个部署密钥，一个仓库最多可以添加 10 个不同的部署密钥。

- SSH 密钥与仓库部署密钥的区别为，前者与用户/计算机关联，后者与仓库关联；SSH 密钥对仓库有读写权限，部署密钥对仓库是只读权限。
- 此设置只针对被设置的仓库生效。
- 仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

部署密钥的设置位于仓库详情中的 “设置 > 安全管理 > 部署密钥”。您可以理解成在这里配置的密钥是一种对仓库只有只读权限的密钥。

单击“设置部署密钥”，以新建部署密钥。在本地生成密钥的方式请参考生成并设置您的 SSH 密钥。

9.6.2 IP 白名单

IP 白名单含义

- IP 白名单是通过设置 IP 白名单的 IP 范围和访问权限，限制用户的访问和上传下载权限，大大增强仓库的安全性。
- 配置 IP 白名单仅支持可见范围为**私有**的仓库，可见范围为**公开只读**或**公开示例模板**的仓库均不受该配置限制。

IP 白名单格式

IP 白名单支持 IPv4 和 IPv6，有 3 种格式，如下表所示。

表9-6 IP 白名单格式

格式	说明
单个 IP	这是最简单的一种 IP 白名单格式，如将您的个人家庭电脑的 IP 添加到白名单中，比如：100.*.*.123。
IP 范围	当您拥有不止一台服务器而且 IP 段是连续的，或者您的 IP 会在一个网段内动态变化，这时您可以添加一个 IP 白名单范围，比如：100.*.*.0 - 100.*.*.255。
CIDR 格式（无类别域间路由）	• 当您的服务器在一个局域网内并使用 CIDR 路由时，您可以指定局域网的 32 位出口 IP 以及一个指定网络前缀的位数。 • 从同一个 IP 发起的请求，只要网络前缀同您设定的前缀部分相同，即可视为来自同一授信范围从而被接受。

配置 IP 白名单

代码托管服务提供多种维度的 IP 白名单设置，仓库管理者可针对所需要的应用场景选择使用。

说明

- **代码托管（单仓库）IP 白名单设置**：请在具体代码仓库的“设置>安全管理>IP 白名单”中设置（支持 IPv4 和 IPv6，具体分类参见 IP 白名单格式），这种控制类型

仅限制名单外的 IP 对特定仓库的访问、操作限制，以下是对非白名单的访问限制的配置：

允许访问仓库：勾选该选项后，非名单内的 IP 不可访问该仓库，仓库所有者不受限制。

允许下载代码：勾选该选项后，非名单内的 IP 禁止代码的在线下载、本地克隆。

允许提交代码：勾选该选项后，非名单内的 IP 禁止代码的在线修改、在线上传、本地提交；构建工程代码化编排，yaml 文件同步功能不受访问控制。

说明

- **提交代码**：新建/编辑/删除/上传/重命名文件，新建/删除目录，新建/删除子模块，新建/删除分支，新建/删除 tag，解决代码冲突，创建/合入 MR，cherrypick，revert，LFS 存储，rebase。
- **下载代码**：单文件/分支/tag/仓库下载，仓库备份。
- **本地下载**：基于 SSH 协议的下载操作，基于 HTTPS 协议的下载操作，基于部署密钥的克隆仓库操作。
- **本地提交**：基于 SSH 协议的提交操作，基于 HTTPS 协议的提交操作。
- 仓库同步操作不受 IP 白名单影响。

- **租户级 IP 白名单**：当您需要对租户下的所有仓库都统一设置 IP 白名单时，可以登录您的代码托管服务仓库列表页，单击右上角昵称，单击“租户设置 > 代码托管 > 租户级 IP 白名单”，进入页面，如下图所示，其配置规则与仓库级 IP 白名单相同。

只有租户账号有配置**“租户级 IP 白名单”**的权限，单击**“新建租户级白名单”**旁的扩展按钮，可勾选“租户优先设置”，Git 客户端克隆/界面下载仓库源码逻辑可参考下表。

租户优先设置	是否配置租户级白名单	是否配置仓库级白名单	优先级
开启	×	×	不判断白名单，所有 IP 均可通过。
	×	√	以仓库级白名单为准。
	√	×	以租户级白名单为准。
	√	√	以租户级白名单与仓库级白名单的交集为准。
关闭	×	×	不判断白名单，所有 IP 均可通过。
	×	√	以仓库级白名单为准。
	√	×	以租户级白名单为准。
	√	√	以仓库级白名单为准。

说明

在新建或编辑租户级和仓库级 IP 白名单时，支持添加备注，字符限制在 200 以内，可为空。

9.6.3 风险操作

风险操作位于仓库详情中的 “设置 > 安全管理 > 风险操作”。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

目前有三个操作：

- 移交仓库所有者：可以移交给仓库内的其他人（不能移交给浏览者）。
- 删除仓库：删除后无法恢复。
- 更改仓库名：更改后请及时排查与仓库名相关的配置。

9.6.4 水印设置

水印设置位于仓库详情中的 “设置 > 安全管理 >水印设置”，水印内容组成为：账户+时间。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

可在代码仓库中显示代码的界面增加水印，降低代码资产泄露风险。

水印设置

为防止未经授权拍照、截图或其他手段随意传播公司核心资产，建议开启水印设置。

9.6.5 锁定仓库

为防止任何人破坏即将发布版本的代码仓库，管理员可以锁定仓库，在锁定仓库后，任何人都无法向任何分支提交代码（包括管理员本人）。

锁定仓库位于仓库详情中的 “设置 > 安全管理 > 锁定仓库”。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

当管理员锁定仓库后，任何人无法执行表 9-7 中的仓库功能。

表9-7 不可执行的功能列表

页签	功能
代码	当仓库已锁定，在“代码”页签，不可执行如下功能： • 新建/编辑/删除/重命名/上传文件 • 新建/删除目录 • 新建/删除子模块 • Cherry-Pick，revert 文件 • 新增/删除/编辑/回复/解决检视意见和评论
分支&tag	当仓库已锁定，在“代码”页签下“分支”或“tag”子页签下，不可执行如下功能： • 新建/编辑/删除分支、分支合并和设置保护分支 • 新建/删除 tag
合并请求	当仓库已锁定，在“合并请求”详情页面，不可执行如下功能： • 新建/编辑/关闭/重开/合并合并请求 • Cherry-Pick，revert 合并请求 • 解决代码冲突 • 新增/删除/编辑/回复/解决检视意见
仓库&成员	当仓库已锁定，不可执行如下功能： • Fork 仓库

页签	功能
	• 新增/删除/编辑/审核成员
设置	当仓库已锁定，在“设置”页签，不可执行如下功能： • 仓库设置 • 子模块设置 • 部署密钥同步功能 • 仓库加速 • 策略设置（全部） • 服务集成（全部）

说明

当仓库锁定后， 如果从项目级更改成员则会同步到仓库里面，从而影响仓库成员。

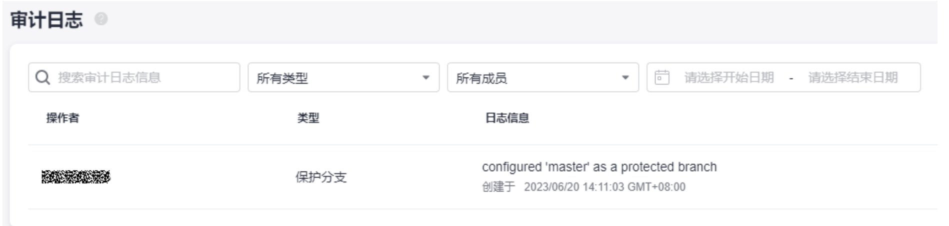
9.6.6 审计日志

审计日志位于仓库详情中的 “设置 > 安全管理 > 审计日志”。

仓库内的仓库成员可以查看该页面，仓库成员是否具有仓库设置权限，请参考“权限管理”页面。

审计日志用于记录对本仓库设置属性的更改，关于对代码的提交、检视以及人员变动等日常的开发活动记录，请通过仓库动态查看。

您可以根据选择具体的时间段、操作者、操作类型或日志信息进行筛选记录。其中操作类型包含以下类型：仓库信息、提交规则、合并请求、合并请求策略等。



9.6.7 权限管理

项目级权限是指服务支持在项目设置中统一配置仓库下各角色的默认权限。仓库级权限是指服务中支持在仓库设置中统一配置仓库下各角色的默认权限。

如果多个仓库需配置同一套角色权限，推荐使用项目级权限配置。

如果多个仓库中每个仓库的角色配置权限不一致，推荐使用仓库级权限配置。

项目级权限控制

配置方法：

步骤 1 登录软件开发生产线平台。

步骤 2 单击目标项目名称，进入项目。

步骤 3 单击“**设置 > 通用设置 > 权限管理**”，进入**权限**页面。

步骤 4 单击对应的“**角色 > 代码托管服务**”，单击“**编辑**”即可修改角色权限。

项目创建者和其他具有管理权限的用户可以在该页面修改不同角色对项目下资源的默认操作权限。

----结束

表9-8 项目级权限列表

类型	操作权限	项目管理员	仓库所有者	项目经理	Committer	开发人员	系统工程师	测试经理、测试人员、参与者、运维经理、产品经理	浏览者	自定义角色	备注
仓库	新建	1	1	2	2	2	2	3	4	3	-
	Fork	1	1	2	2	2	2	3	4	3	Fork 是对既有对象有无限复制的权限，至于能否复制成功，同时还取决于在目标项目下有无新建仓库的权限。 如：成员 A 在项目 A 下有 Fork 仓库权限。 • 如果在项目 B 下有新建仓库的权限，则 Fork 成功。 • 如果成员 A 在项目 B 下无新建仓库权限，则无法 Fork 成功。

类型	操作权限	项目管理员	仓库所有者	项目经理	Committer	开发人员	系统工程师	测试经理、测试人员、参与者、运维经理、产品经理	浏览者	自定义角色	备注
	删除	1	1	2	4	4	4	4	4	3	-
	设置	1	1	3	4	4	4	4	4	3	-
代码	提交	1	1	2	1	1	1	3	4	3	-
	下载	1	1	2	1	1	1	3	4	3	-
成员（组）	添加	1	1	2	4	4	4	4	4	3	-
	修改	1	1	2	4	4	4	4	4	3	-
	删除	1	1	2	4	4	4	4	4	3	-
分支	新建	1	1	2	2	2	2	3	4	3	-
	删除	1	1	2	2	2	2	3	4	3	-
Tag	新建	1	1	2	2	2	2	3	4	3	-
	删除	1	1	2	3	3	3	3	4	3	-
MR	新建	1	1	2	2	2	2	3	4	3	-
	编辑	1	1	2	2	3	3	4	4	3	• 已合并 MR 不支持编辑。 • 编辑/关闭/重

类型	操作权限	项目管理员	仓库所有者	项目经理	Committer	开发人员	系统工程师	测试经理、测试人员、参与者、运维经理、产品经理	浏览者	自定义角色	备注
											开 MR 依然保留 MR 创建者可以编辑自己的 MR。
	评论	1	1	2	2	2	2	3	3	3	-
	检视	1	1	2	2	2	2	4	3	3	-
	审核	1	1	2	2	3	3	4	4	3	-
	合并	1	1	2	2	3	3	4	4	3	-
	关闭	1	1	2	2	3	3	4	4	3	编辑 MR、关闭 MR、重开 MR 依然保留 MR 创建者可以编辑自己的 MR。
	重开	1	1	2	2	3	3	4	4	3	编辑 MR、关闭 MR、重开 MR 依然保留 MR 创建者可以编辑自己的 MR。

说明

- 只有仓库成员具备“仓库”、“代码”、“成员”、“分支”、“Tag”和“MR”的查看权限。
- 仓库所有者、项目经理、Committer 和评论者可以解决 MR 意见。评分模式下，Committer、项目经理可以评分±2 分，其余角色可评分±1 分，浏览者不得评分。
- 1：表示该角色默认拥有该权限且不可被移除。
- 2：表示该角色默认拥有该权限且可被移除。

- 3：表示该角色可分配到该权限。
- 4：表示该角色不可分配到该权限。

仓库级权限控制

配置方法：

步骤 1 进入软件开发生产线首页，单击目标项目名称，进入项目。

步骤 2 单击菜单**“服务 > 代码托管”**，进入代码托管服务。

步骤 3 单击“**设置 > 安全管理 > 权限管理**”，进入**权限管理**页面。

支持对“**仓库**”、“**代码**”、“**成员**”、“**分支**”、“**Tag**”和“**MR**”六种维度进行角色权限配置。

----结束

表9-9 仓库级权限列表

类型	操作权限	仓库所有者	项目经理	Committer	开发人员	系统工程师	测试经理、测试人员、参与者、运维经理、产品经理	浏览者	自定义角色	备注
仓库	Fork	1	2	2	2	2	3	4	3	-
	删除	1	2	4	4	4	4	4	3	-
	设置	1	2	4	4	4	4	4	3	-
代码	提交	1	2	1	1	1	3	4	3	-
	下载	1	2	1	1	1	3	4	3	-
成员	添加	1	2	4	4	4	4	4	3	-
	修改	1	2	4	4	4	4	4	3	-
	删除	1	2	4	4	4	4	4	3	-
分	新	1	2	2	2	2	3	4	3	-

类型	操作权限	仓库所有者	项目经理	Committer	开发人员	系统工程师	测试经理、测试人员、参与者、运维经理、产品经理	浏览者	自定义角色	备注
支	建									
	删除	1	2	2	2	2	3	4	3	-
Tag	新建	1	2	2	2	2	3	4	3	-
	删除	1	2	3	3	3	3	4	3	-
MR	新建	1	2	2	2	2	3	4	3	-
	编辑	1	2	2	3	3	4	4	3	-
	评论	1	2	2	2	2	3	3	3	-
	检视	1	2	2	2	2	4	3	3	-
	审核	1	2	2	3	3	4	4	3	-
	合并	1	2	2	3	3	4	4	3	-
	关闭	1	2	2	3	3	4	4	3	-
	重开	1	2	2	3	3	4	4	3	-

说明

- 您可以将已配置好的角色权限按照上表进行修改。
- 只有仓库成员具备“仓库”、“代码”、“成员”、“分支”、“Tag”和“MR”的查看权限。
- 1：表示该角色默认拥有该权限且不可被移除。
- 2：表示该角色默认拥有该权限且可被移除。
- 3：表示该角色可分配到该权限。
- 4：表示该角色不可分配到该权限。

10 提交代码到代码托管仓库

创建提交
加密传输与存储
查看提交历史
在 Eclipse 提交代码到代码托管

10.1 创建提交

背景说明

在日常代码开发中，开发者更多的时候是将代码托管仓库克隆到本地，在本地进行代码开发，完成了阶段性开发任务后，再提交回代码托管仓库，本文将介绍使用 Git 客户端提交修改代码的方法。

前提条件

1. Git 客户端安装配置。
2. 在代码托管服务中创建仓库。
3. 设置 HTTPS 密码。
4. 将代码托管仓库克隆到本地。

操作步骤

一般情况下，开发者不会直接在 master 分支中进行开发，而是基于 master 或者 dev 分支创建一条 feature 分支，在 feature 中进行开发，然后将其推送到代码托管仓库，最后在代码托管仓库中将其合并到 master 或 dev 分支中，下面将模拟以上操作。

步骤 1 进入本地仓库目录，打开 Git 客户端，本案例以 Git Bash 为例，其它使用 Git 进行管理的工具的原理和命令使用基本是一致的。

步骤 2 基于 master 分支新建一条分支 feature1001，并切换到其中，在 master 分支中执行以下命令。

```
git checkout -b feature1001    #如下图 1
```

这个命令相当于先新建分支，然后直接切换到此分支。

执行成功如下图中 2 所示，此时可用 ls 命令查看其中包含的文件(如下图中 3)，此时他与 master 分支中内容是一样的。

```
Administrator@ecstest-paas-lwx      MINGW64 ~/Desktop/liu'Code/          (master)
$ git checkout -b feature1001 ①
Switched to a new branch 'feature1001' ②

Administrator@ecstest-paas-lwx      MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ ls ③
111           7370149       0008/           file02              fileFor7370151         FromFork  phoenix-sample/    xin/
1111/         7370149fix    0009b/          file03              fileOnBranch002.txt    hooks/    README.md
20200714.txt  after996.txt  devOnBranch009  file10              forShowMergeRequest    lfs/      testMergeBranch012
666           .xlsx         file01          fileFor7370149.txt  forTestFor             liu/      testMergeBranch013
```

步骤 3 在 feature 分支中进行修改（代码开发）。

Git 支持 Linux 命令，本案例用 touch 命令新建一个 newFeature1001.html 文件，代表开发者已经在本地完成了新特性的开发，其对本地代码库的影响是新增了文件。

```
touch newFeature1001.html
```

创建后再次使用 ls 命令可以看到多出了这个文件。

步骤 4 使用 add、commit 命令依次将文件从工作区加入暂存区，再提交到本地版本库。（这是什么原理？）

期间可以穿插使用 status 命令，观察文件状态。

1. 使用 status 命令看到，目前工作区有一个文件未纳入版本管理，如图中 1。
2. 使用 add 命令将文件加入暂存区，如图中 2。

```
git add .    #使用“.”代表所有文件，包括隐藏的，也可以直接指定某个文件
```

3. 使用 status 命令看到，文件已经加入到暂存区，正在等待提交，如图中 3。
4. 使用 commit 命令将文件提交到本地版本库，如图中 4。

```
git commit -m “您的提交备注”
```

5. 再次查看状态，没有可处置文件，说明提交成功了，如图中⑤。

```
Administrator@ecstest-paas-lwx      MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ git status ①
On branch feature1001
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        newFeature1001.html

nothing added to commit but untracked files present (use "git add" to track)

Administrator@ecstest-paas-lwx      MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ git add . ②

Administrator@ecstest-paas-lwx69    MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ git status ③
On branch feature1001
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newFeature1001.html

Administrator@ecstest-paas-lwx6     0 MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ git commit -m "This is a commit for feature1001~!" ④
[feature1001 4c8db12] This is a commit for feature1001~!
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newFeature1001.html

Administrator@ecstest-paas-lw       MINGW64 ~/Desktop/liu'Code/          (feature1001)
$ git status ⑤
On branch feature1001
nothing to commit, working tree clean
```

步骤 5 将本地的分支推送到代码托管仓库。

```
git push --set-upstream origin feature1001
```

本命令会在代码托管仓库新建一条与您本地 feature1001 一样的分支，并将其进行关联、同步。

其中 origin 是您的代码托管仓库名称，一般直接可控的仓库默认别名为 origin，您也可以直接用仓库地址代替。

```
Administrator@ecstest-paas-lwx...... MINGW64 ~/Desktop/liu'Code/ (feature1001)
$ git push --set-upstream origin feature1001
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 287.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for feature1001, visit:
remote:   https://
remote:
To .git
 * [new branch]      feature1001 -> feature1001
Branch 'feature1001' set up to track remote branch 'feature1001' from 'origin'.
```

说明

如果推送失败请检查连通性：

- 确保您的网络可以访问代码托管服务。

  请在 git 客户端使用如下测试命令验证网络连通性。

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

  如果返回内容含有“connect to host ********.com port 22: Connection timed out”，则您的网络被限制，无法访问代码托管服务，请求助您本地所属网络管理员。
- 请检查建立的密钥配对关系，必要时重新生成密钥并到代码托管控制台进行配置，请确认SSH 密钥或确认HTTPS 密码配置正确。
- 检查 IP 白名单。注意，在未配置白名单时，全部 IP 均会放行，如果配置了则只允许名单内的 IP 访问。

步骤 6 查看代码托管仓库分支。

登录代码托管服务，进入您的仓库，在文件列中可以看到此时已经可以在代码托管仓库切换到您的分支。

说明

如果没有看到您刚推送上来的分支，很可能是您的 origin 绑定到了另外的仓库，请使用仓库地址再次推送。

步骤 7 此时您可以使用代码托管服务提供的合并请求管理功能，发起分支合并，并通知审核人进行评审，最终将新特性合入到 master 或 dev 分支中。

----结束

10.2 加密传输与存储

代码托管服务通过使用 git-crypt 来满足开发者对机密、敏感文件的加密存储与传输需要。

git-crypt 简介

git-crypt 是一款第三方开源软件，可以用于对 Git 仓库中的文件进行透明化的加密和解密。其可对指定文件、指定文件类型等进行加密存储，开发者可以将加密文件（如机密信息或敏感数据）与可共享的代码存储在同一个仓库中，并如同普通仓库一样被拉取和推送，只有持有对应文件密钥的人才能查看到加密文件的内容，但并不会限制参与者对非加密文件读写。

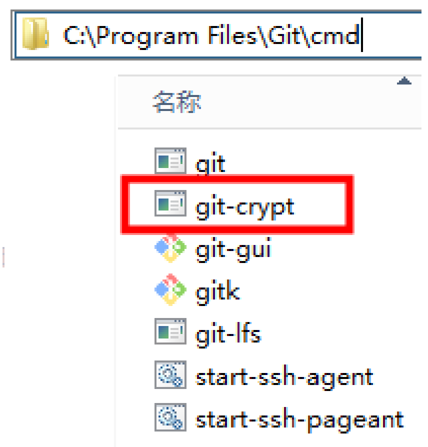
也就是说，使用 git-crypt 可以让您仅加密特定文件，而不需要锁定整个仓库，这既方便了团队合作，又可兼顾信息安全。

在 Windows 中使用密钥对方式进行加密、解密

步骤 1 安装并初始化 Git。

步骤 2 下载最新基于 Windows 的 git-crypt，将下载到的 exe 文件放到 Git 安装目录下的“cmd”文件夹中，下图以“Windows Server 2012 R2 标准版 64”的默认 Git Bash 安路径为例。

放进文件夹即可，不需要运行此 exe。



步骤 3 **生成密钥对。**

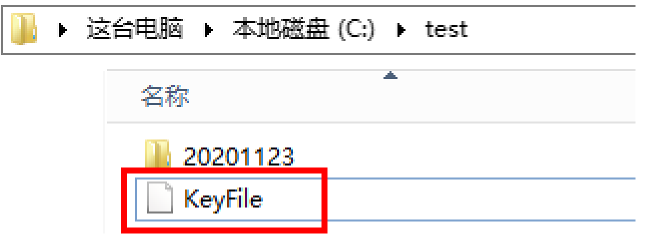
1. 打开“Git Bash”，并进入本地仓库（如下图 1）。
2. 生成密钥对，输入指令如下（如下图 2）：

```
git-crypt init
```

3. 将密钥文件导出，本示例中将密钥文件导出到 C:\test 目录并名为 KeyFile，输入指令如下（如下图 3）：

```
git-crypt export-key /c/test/keyfile
```

4. 执行完以上步骤，可以到密钥导出的文件路径，验证下是否生成了密钥，在本示例中到 C:\test 路径下验证是否有 KeyFile 文件，如下图所示。



持有这个密钥文件的计算机，可以解密对应的加密文件。

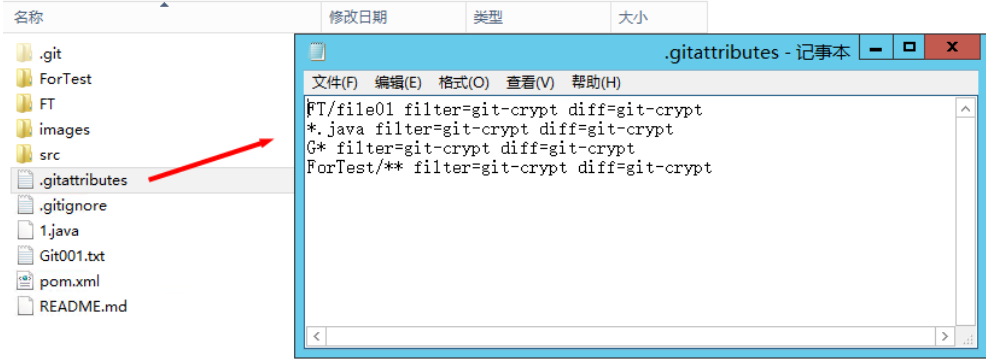
步骤 4 **为仓库配置加密范围**。

1. 在仓库的根目录下新建一个名为.gitattributes 的文件。
2. 打开.gitattributes 文件，设置加密范围，语法如下。

```
文件名或文件范围 filter=git-crypt diff=git-crypt
```

下面给出四个示例。

```
FT/file01.txt filter=git-crypt diff=git-crypt   #将 特定文件加密，这里加密的是 FT 文件夹下的 file01.txt
*.java filter=git-crypt diff=git-crypt   #将 .java 类型文件加密
G* filter=git-crypt diff=git-crypt      #将 文件名为 G 开头的文件加密
ForTest/** filter=git-crypt diff=git-crypt #将 ForTest 文件夹下的文件加密
```



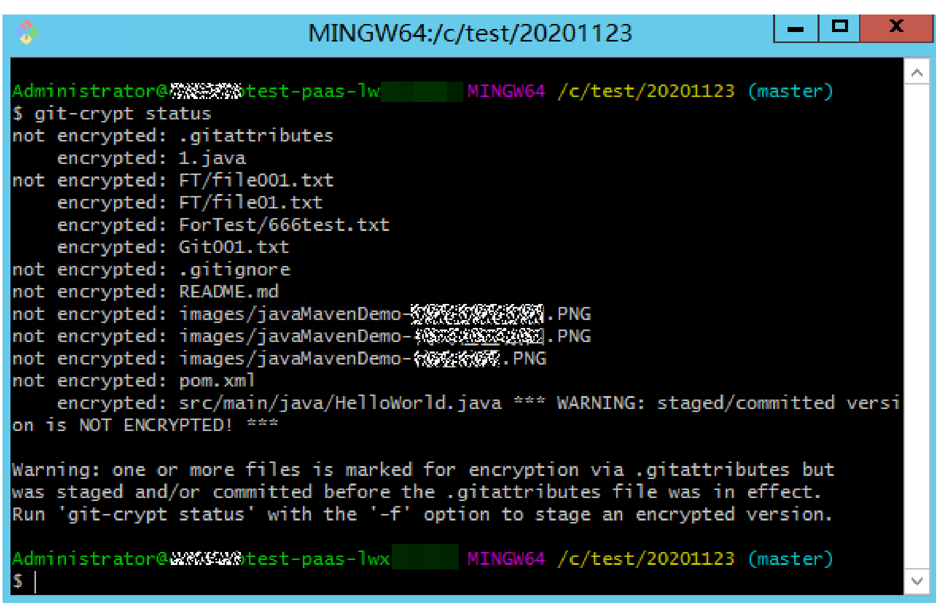
说明

- 如果创建.gitattributes 文件时提示“必须键入文件名”，可以将文件名填写成“.gitattributes.”即可创建成功，如果使用 Linux 指令创建文件，则不会出现此问题。
- 注意不要将.gitattributes 保存成 txt 文件，这样配置会无效。

步骤 5 **进行文件加密**。

仓库根目录打开 Git bash，执行如下指令即可完成加密，并会看到目前文件的加密状态。

```
git-crypt status
```



加密执行后，在您的本地仓库仍能明文方式打开和编辑这些加密文件，这是因为您本地仓库有密钥存在。

这时你可以使用 add 、commit、push 组合将仓库推送到代码托管仓库，此时加密文件将一同被推送。

加密文件在代码托管仓库中将以加密二进制方式存储，无法直接查看。如果没有密钥，就算将其下载到本地，也无法解密。

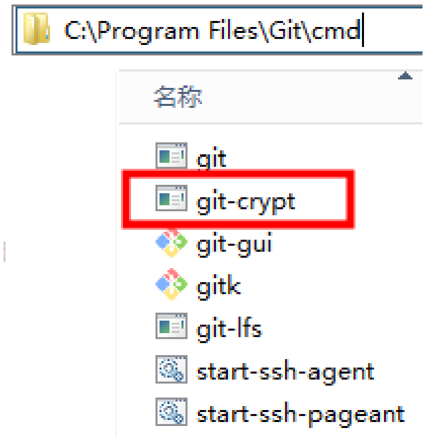
说明

"git-crypt status"只会加密本次待提交的文件，对本次未发生修改的历史文件不会产生加密作用，Git 会对此设定涉及的未加密文件做出提示（见上图中的 Warning），如果想将仓库中的对应类型文件全部加密，请使用"git-crypt status -f"。

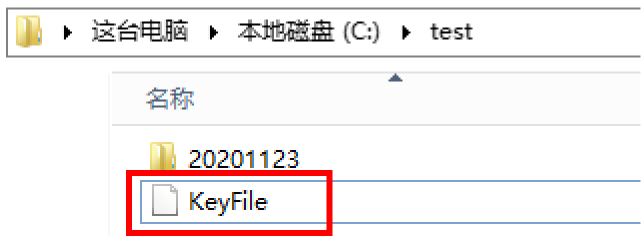
在让团队合作中 -f （强制执行）具有一定的风险，可能会对合作伙伴的工作产生不变，请谨慎使用。

步骤 6 进行文件解密。

1. 确认本机器 Git 安装路径下存在 git-crypt 文件。



2. 将仓库从代码托管克隆到本地。
3. 获取加密此仓库的密钥文件，并存储于本地计算机。



4. 进入仓库目录，右键打开 Git bash。
5. 执行解密指令，执行后无回显，则为执行成功。

```
git-crypt unlock /C/test/KeyFile     #请将 /C/test/KeyFile 更换为您实际的密钥存储路径
```

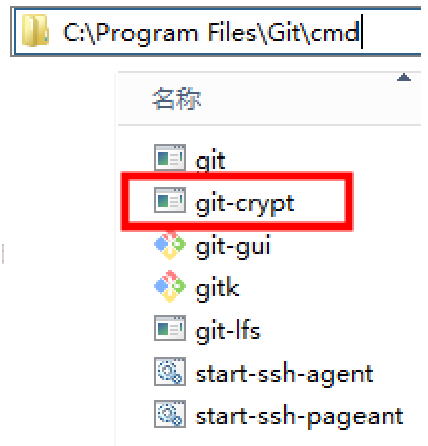
----结束

在 Windows 中使用 GPG 方式进行加密、解密

步骤 1 安装并初始化 Git。

步骤 2 下载最新基于 Windows 的 git-crypt，将下载到的 exe 文件放到 Git 安装目录下的“cmd”文件夹中，下图以“Windows Server 2012 R2 标准版 64”的默认 Git Bash 安路径为例。

放进文件夹即可，不需要运行此 exe。



步骤 3 下载 GPG 最新版本，当提示您捐赠此开源软件时，选“0”,即可跳过捐赠环节。

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

双击进行安装，仅使用“下一步”，即可完成安装。

步骤 4 **使用 GPG 方式生成密钥对。**

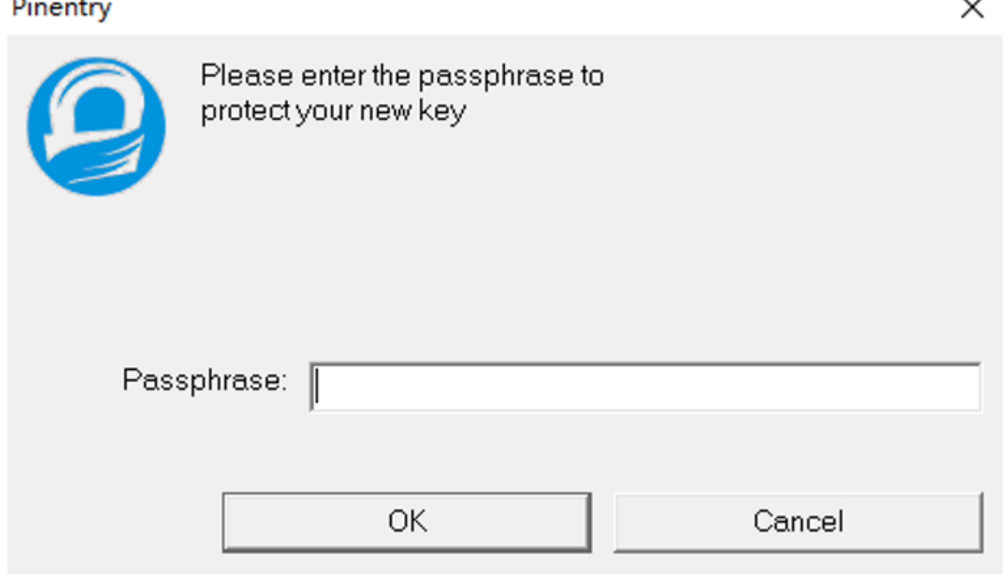
1. 任意位置打开 Git Bash，执行如下指令。

```
gpg --gen-key
```

2. 根据提示，输入名称、邮箱。

```
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. 确认无误后，按提示输入 “o”,并回车，此时会弹出输入密码窗口和确认密码窗口。



密码可以为空，出于信息安全考虑，建议输入符合标准的密码（解密时需要这个密码）。

4. 出现如下图返回内容，则为 GPG 密钥对生成成功。

```
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]
```

步骤 5　进行仓库加密初始化设置。

1. 仓库根目录打开 Git bash，执行如下指令进行初始化。

```
git-crypt init
```

```
Administrator@codehubtest-paas-lwx MINGW64 /c/dev/test
$ cd 20201124

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

Administrator@codehubtest-paas-lw MINGW64 /c/dev/test/20201124 (master)
$ |
```

2. 将密钥副本添加到您的仓库，该副本已使用您的公共 GPG 密钥加密，指令如下。

```
git-crypt add-gpg-user USER_ID
```

此处的“USER_ID”可以是生成此密钥时的名称（1）、邮箱（2）或指纹（3）这三种可以唯一标识此密钥的东西。

```
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]
```

执行后会提示您创建了.git-crypt 文件夹以及其中的两个文件。

```
MINGW64:/c/dev/test/20201124
Administrator@codehubtest-paas-lw MINGW64 /c/dev/test/20201124 (master)
$ git-crypt add-gpg-user gpgTest
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2022-11-24
[master 2e4aa2b] Add 1 git-crypt collaborator
 2 files changed, 4 insertions(+)
 create mode 100644 .git-crypt/.gitattributes
 create mode 100644 .git-crypt/keys/default/0/0DDF227
71E0AD.gpg

Administrator@codehubtest-paas-lw MINGW64 /c/dev/test/20201124 (master)
$ |
```

步骤 6 为仓库配置加密范围。

1. 进入仓库下的.git-crypt 文件夹。
2. 打开.gitattributes 文件，设置加密范围，语法如下。

```
文件名或文件范围 filter=git-crypt diff=git-crypt
```

下面给出四个示例。

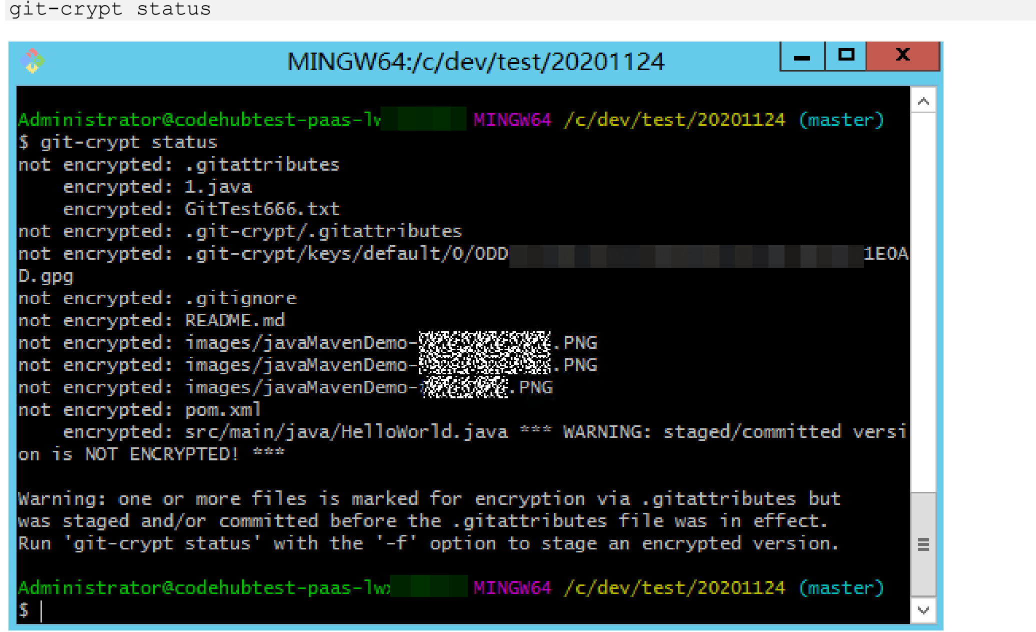
```
FT/file01.txt filter=git-crypt diff=git-crypt   #将 特定文件加密，这里加密的是 FT 文件夹下的 file01.txt
*.java filter=git-crypt diff=git-crypt   #将 .java 类型文件加密
G* filter=git-crypt diff=git-crypt      #将 文件名为 G 开头的文件加密
ForTest/** filter=git-crypt diff=git-crypt #将 ForTest 文件夹下的文件加密
```

3. 将.gitattributes 文件复制到仓库的根目录下。

步骤 7 进行文件加密。

仓库根目录打开 Git bash，执行如下指令即可完成加密，并会看到目前文件的加密状态。

```
git-crypt status
```



加密执行后，在您的本地仓库仍能明文方式打开和编辑这些加密文件，这是因为您本地仓库有密钥存在。

这时你可以使用 add 、commit、push 组合将仓库推送到代码托管仓库，此时加密文件将一同被推送。

加密文件在代码托管仓库中将以加密二进制方式存储，无法直接查看。如果没有密钥，就算将其下载到本地，也无法解密。

说明

"git-crypt status"只会加密本次待提交的文件，对本次未发生修改的历史文件不会产生加密作用，Git 会对此设定涉及的未加密文件做出提示（见上图中的 Warning)，如果想将仓库中的对应类型文件全部加密，请使用"git-crypt status -f"。

在让团队合作中 -f （强制执行）具有一定的风险，可能会对合作伙伴的工作产生不变，请谨慎使用。

步骤 8 将密钥导出。

1. 列出当前可见密钥，可以看到每个密钥的名称、邮箱、指纹信息。

```
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 D
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. 使用 gpg --export-secret-key 指令导出密钥，本示例中将名称为“gpgTest”的密钥导出到“C 盘”，并命名为“Key”。

```
gpg --export-secret-key -a gpgTest > /c/key   # -a 代表以 文本形式显示输出
```

执行时会提示您输出此密钥的密码，正确输入即可。

执行后无回显，到对应目录（本示例中为 C 盘）可看到密钥文件。

3. 将生成的密钥发送给团队内需要共享秘密文件的伙伴。

步骤 9 将密钥导入并解密文件。

1. 想要在另一台机器解密文件，首先也需要基于 Git，下载安装 git-crypt、GPG，可以参考本文前几步骤的操作。
2. 将对应仓库 Clone 到本地。
3. 取得对应加密文件的密钥，密钥的导出请参见上一步骤，本示例中将取得的密钥放在 C 盘。
4. 进入仓库，打开 Git Bash 使用 import 指令导入密钥。

```
gpg --import  /c/key
#其中 /c/Key 是本示例的密钥路径和密钥自定义名称，实际使用时，请根据实际情况替换
```

导入时会弹出按提示您输入此密钥的密码，导入成功会的效果如下图。

5. 使用 unlock 指令，解密文件。

```
git-crypt unlock
```

解锁时会弹窗提示您输入此密钥的密码，正确输入后无回显，则为解锁成功。

```
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 E0AD: public key "gpgTest <gpgTest@huahua.com>" imported
gpg: key 3E38 E0AD: 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
```

步骤 10 解密完成后，查看文件可以看到文件内容已经不是加密状态。

----结束

git-crypt 加密在团队合作中的应用

在很多时候，团队需要在代码仓库中存储**限制公开**的文件，这时可以优先考虑使用“CodeArts Repo” + “Git” + “git-crypt”的组合，来实现部分文件在仓库分布式开源中的加密。

通常，直接使用密钥对方式的加密就能满足限制部分文件访问的需要。

当团队需要将加密文件设置不同的秘密级别时，可以使用 GPG 方式加密，这种方式支持您对同一个仓库的不同文件使用不同的密钥加密，将不同密级的密钥分别随仓库共享给组织内的伙伴，即可实现文件的定向分级限制访问。

Linux、Mac 平台的 git-crypt、gpg 安装

Linux 平台安装 git-crypt、gpg

- Linux 安装依赖环境。

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

- Linux 环境下，使用源码编译方式安装 git-crypt。

```
make
make install
```

安装到指定目录。

```
make install PREFIX=/usr/local
```

- Linux 环境下，使用源码编译方式安装 GPG。

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

- 使用 Debian 包安装 git-crypt。

Debian 打包可以在项目 Git 仓库的“debian”分支中找到。

软件包是用“git-buildpackage”构建的，如下所示。

```
git checkout debian
git-buildpackage -uc -us
```

- Debian 环境下使用构建包安装 GPG。

```
sudo apt-get install gnupg
```

MAC 平台安装 git-crypt、gpg

- macOS 上安装 git-crypt。

使用 brew 软件包管理器，只需运行如下命令。

```
brew install git-crypt
```

- macOS 上安装 GPG。

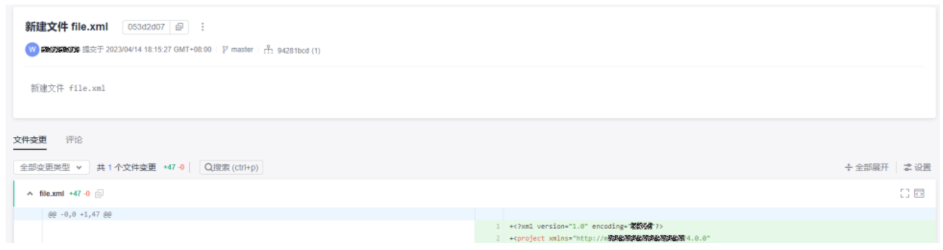
  使用 brew 软件包管理器，只需运行如下命令。

  ```
  brew install gpg
  ```

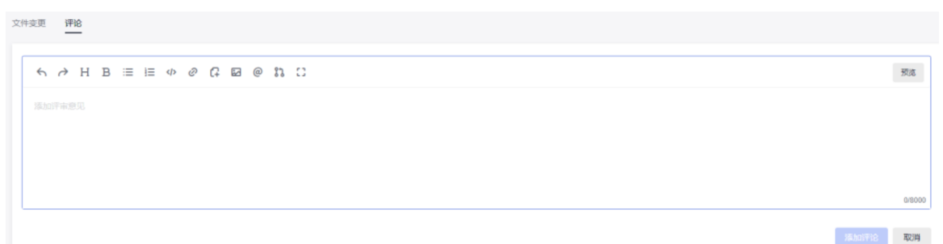
10.3 查看提交历史

代码托管服务支持查看提交历史的详细信息以及其涉及的文件变更。

您可以在仓库的动态、仓库文件列表的历史页签中，查看提交历史的清单，单击某次提交历史可以进入查看此次提交提交人、提交号、父节点、此条提交下评论的数量、代码变更对比等。



您可以对提交内容进行评论，也可对评论内容进行跟帖。



单击下图中的图标可以切换代码变更对比的横版或纵版显示，单击**“全部展开”**可以查看此次提交中涉及的文件的全文。

10.4 在 Eclipse 提交代码到代码托管

背景信息

Eclipse 安装 Git 插件 EGit 后，可以完全对接代码托管，可以将本地 Git 仓库代码完整提交到远程 Git 仓库中。

说明

只支持 Ecplise 4.4 以上版本（在 Eclipse3.3 版本没有自带 EGit 插件，无法安装）。

- 如果是首次提交：
- 首先在本地计算机建立一个仓库，称本地仓库。

1. 然后在本地进行 **Commit**，将更新提交到本地仓库。
2. 最后将服务器端的更新 **Pull** 到本地仓库进行合并，最后将合并好的本地仓库推送到服务器端，即进行一次远程提交。

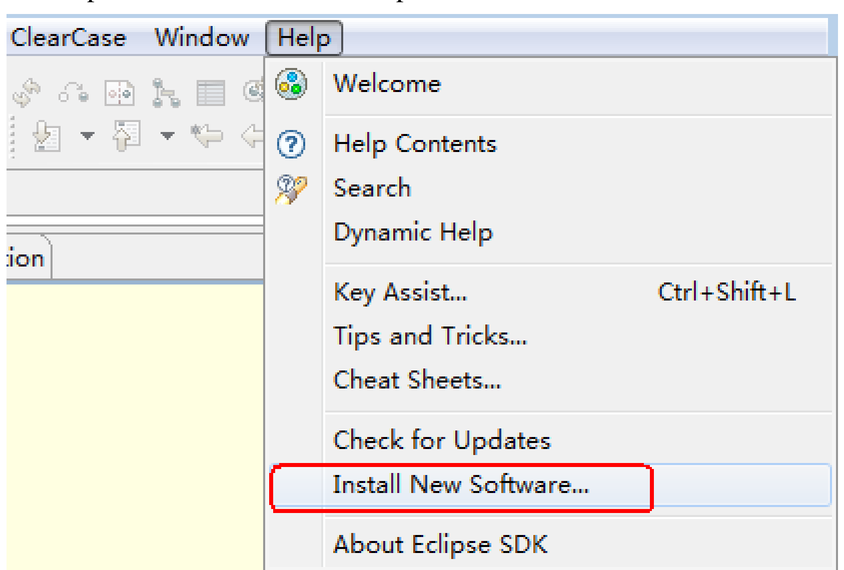
- 如果非首次提交：
- 首先将修改的代码 **Commit** 更新到本地仓库。

1. 然后将服务器端的更新 **Pull** 到本地仓库进行合并，最后将合并好的本地仓库**推送到**服务器端。

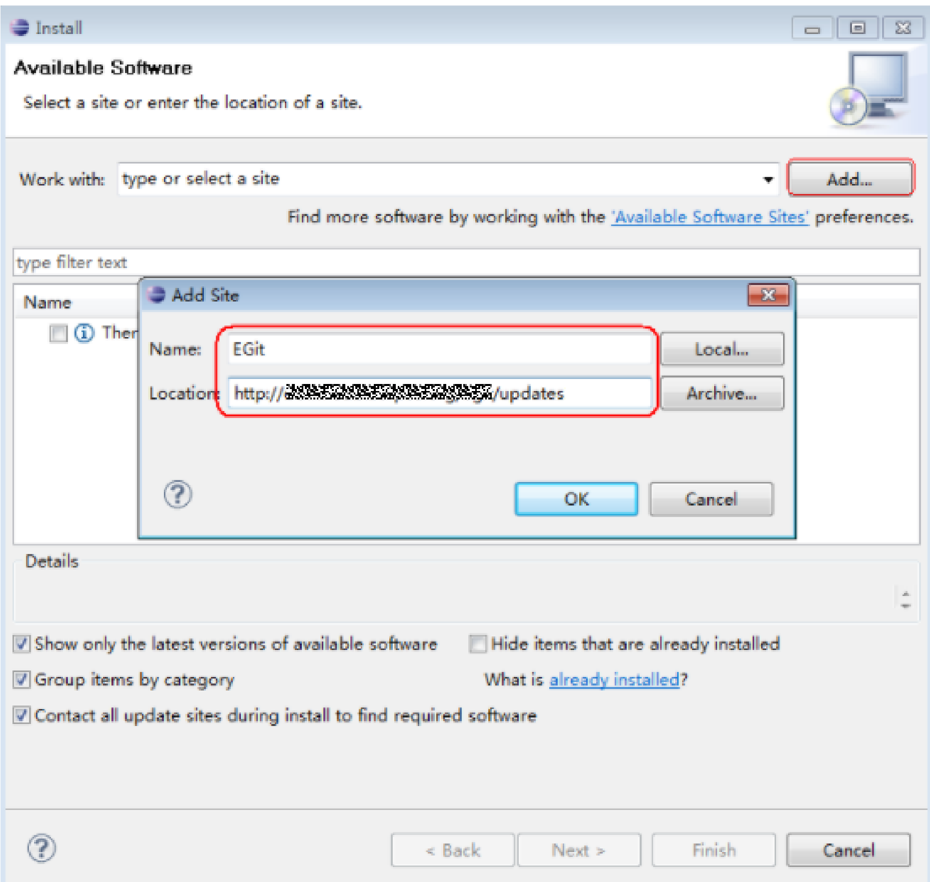
步骤一：在 Eclipse 上安装 Git 插件 EGit

以 Eclipse 的版本 4.4 为例，具体操作如下：

1. 在 Eclipse 上方工具栏选择“Help > Install New Software...”菜单，如下图所示。



2. 在弹出的“Install”窗口中，单击“Add...”按钮，如下图所示。



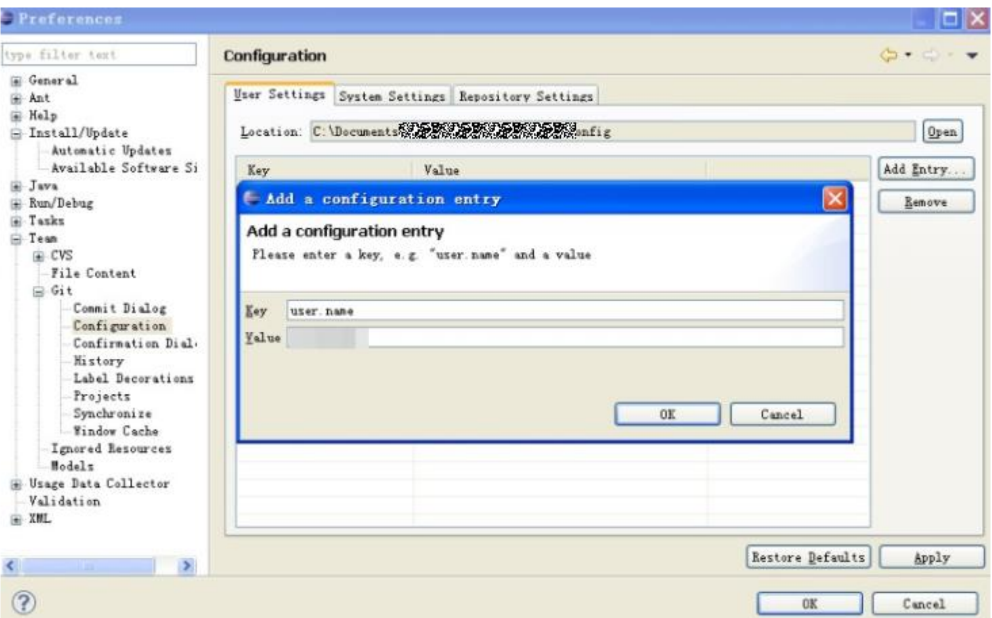
3. 单击“OK”按钮，随后连续下一步默认安装即可。

   安装完成后重启 Eclipse。

步骤二：在 Eclipse 中配置 EGit

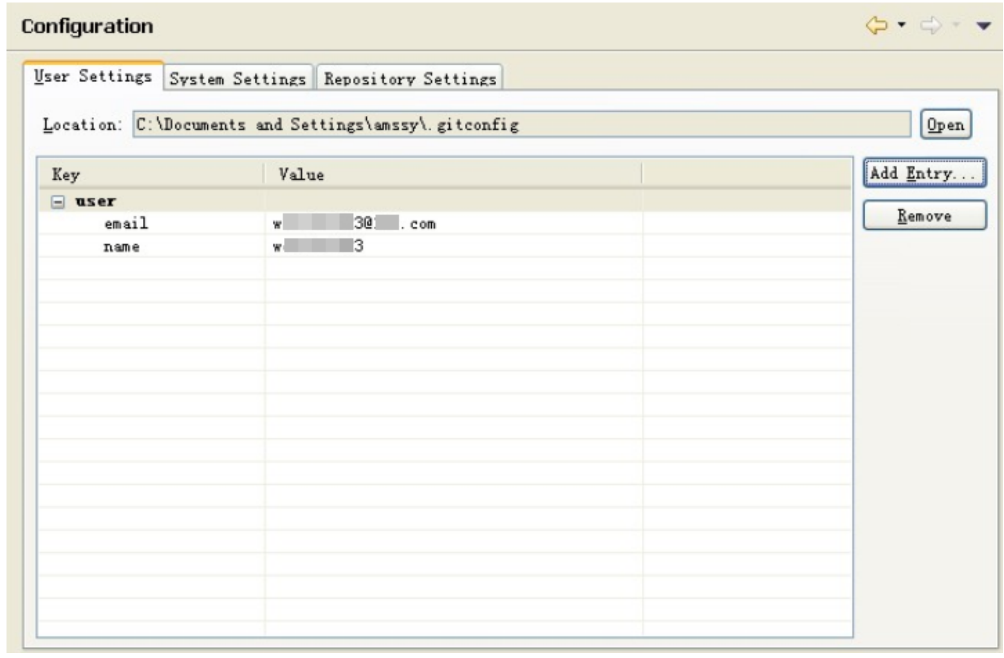
1. 在 Eclipse 上方工具栏选择“Window > Preferences > Team > Git > Configuration”， 如下图所示。

   “user.name”为已注册的用户名。



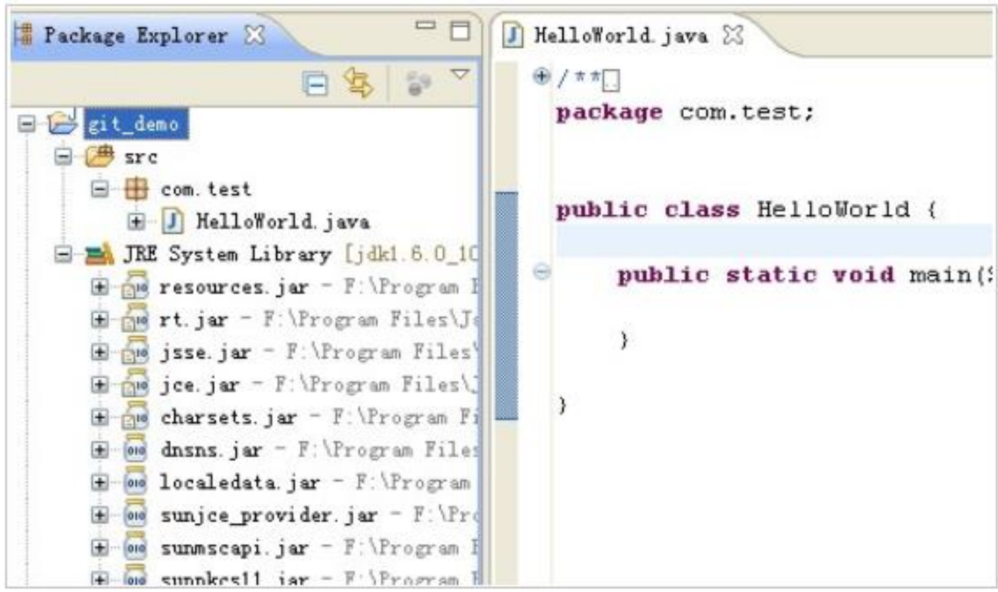
2. 单击“OK”，如下图所示。

   “user.email”为已绑定的邮箱。在这里配置“user.name”即可。

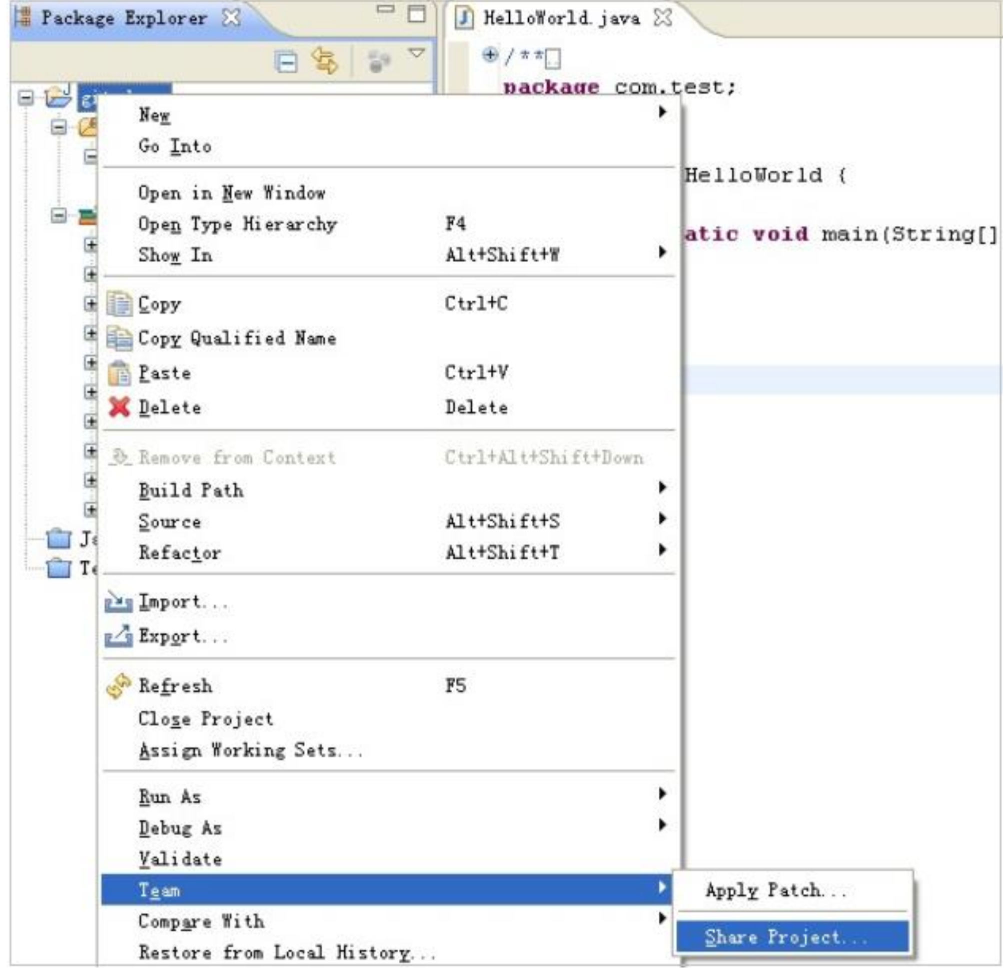


步骤三：新建项目，并将代码提交到本地的 Git 仓库中

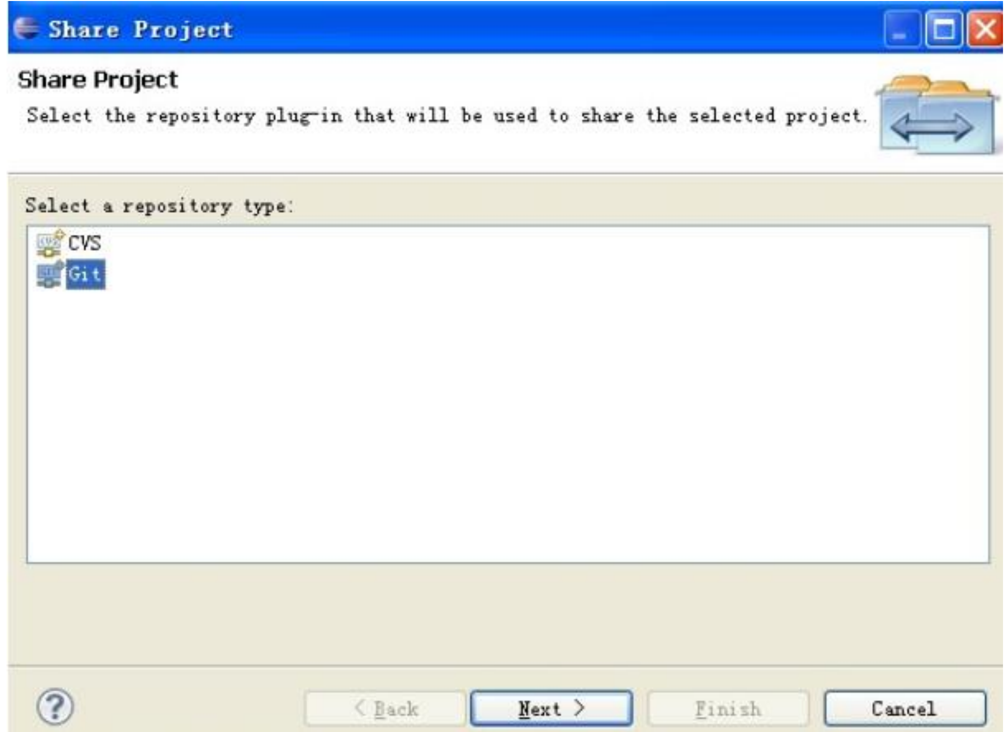
1. 新建项目“git_demo”，并新建“HelloWorld.java”类，如下图所示。



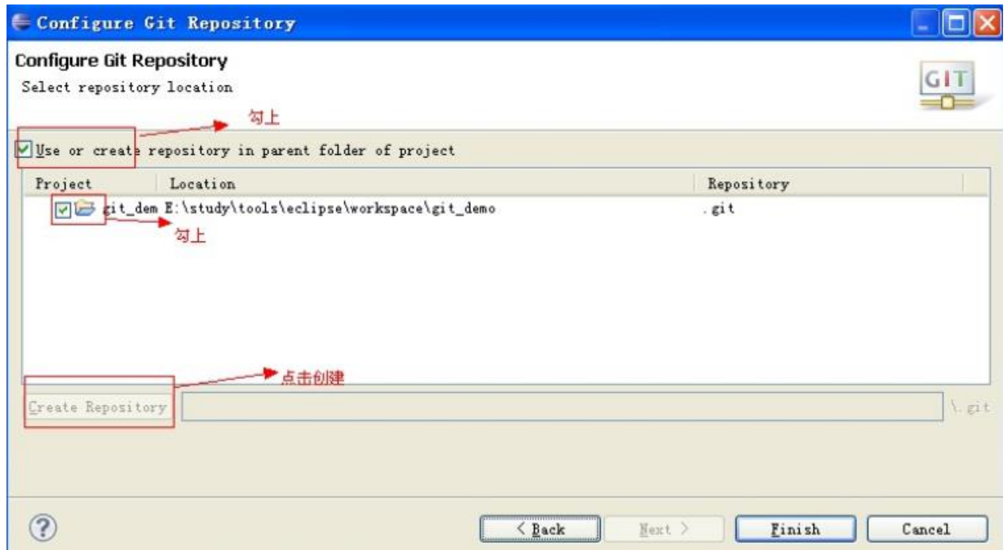
2. 将“git_demo”项目提交到本地仓库，如下图所示。



3. 在弹出的“Share Project”窗口中，选中“Git”，如下图所示。



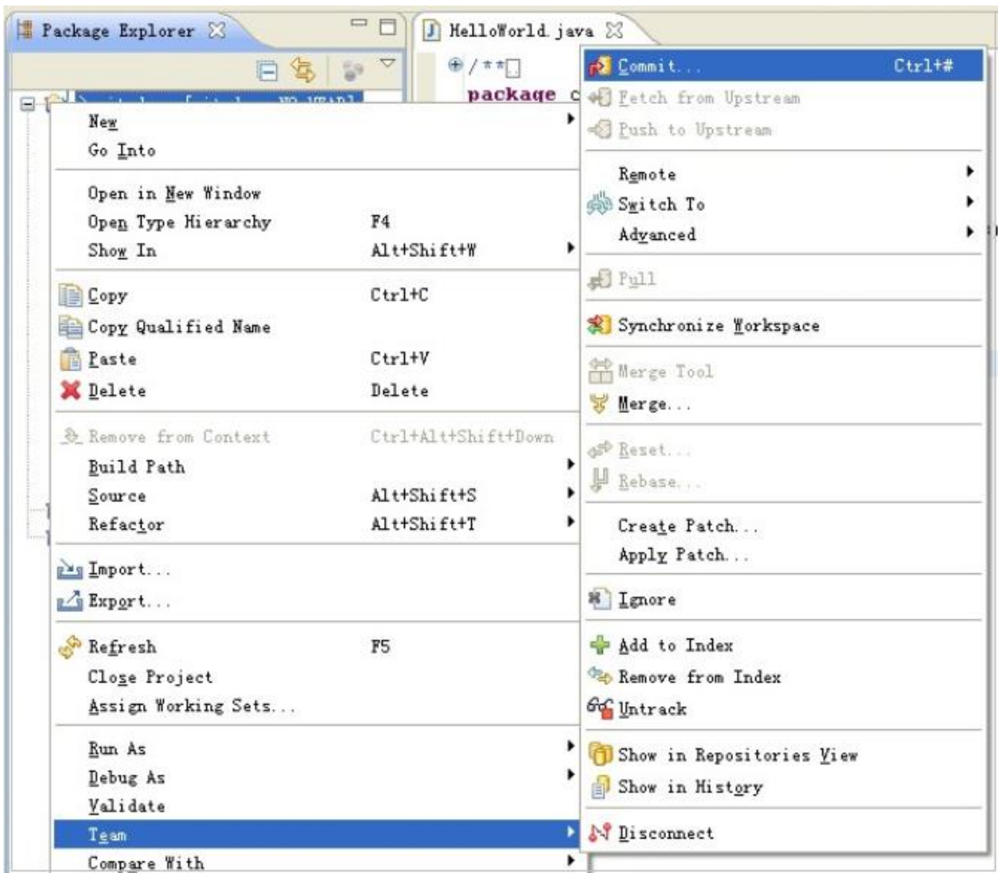
4. 单击“Next”，弹出“Configure Git Repository”，如下图所示。



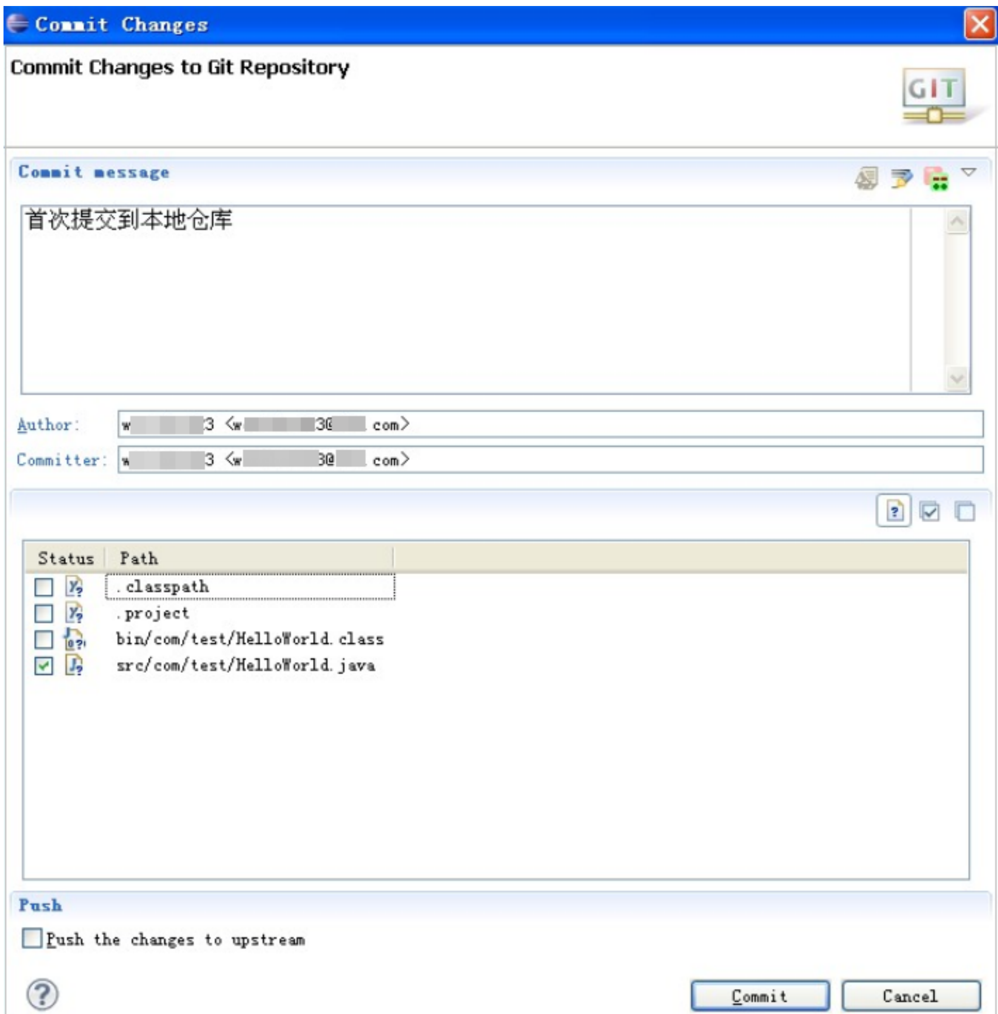
5. 单击“Create Repository”，成功创建 Git 仓库。

文件夹此时处于“untracked”状态（文件夹中以符号“?”表示）。

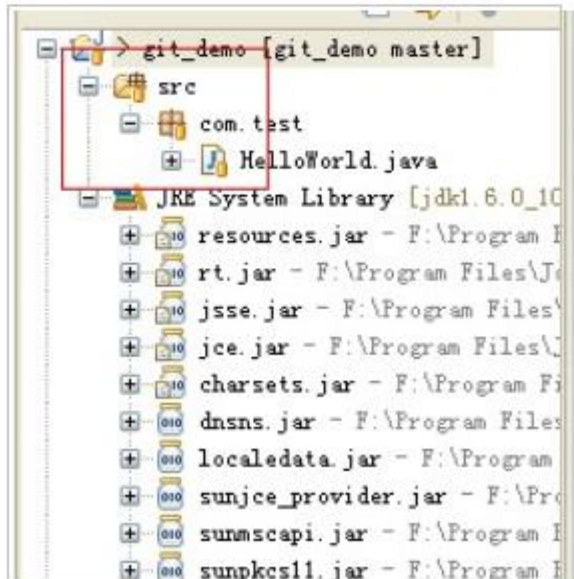
此时需要提交代码到本地仓库，如下图所示开始提交。



6. 弹出“Commit Changes”窗口，设置提交信息，如下图所示。



7. 单击“Commit”，代码提交到本地仓库，如下图所示。

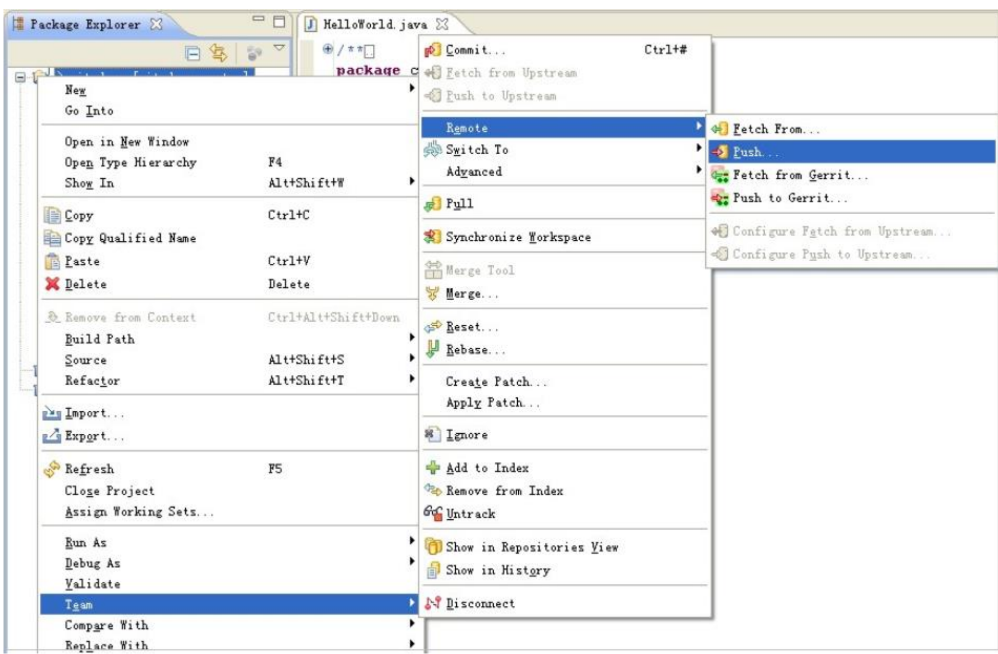


步骤四：将本地仓库代码提交到远程的 Git 仓库中

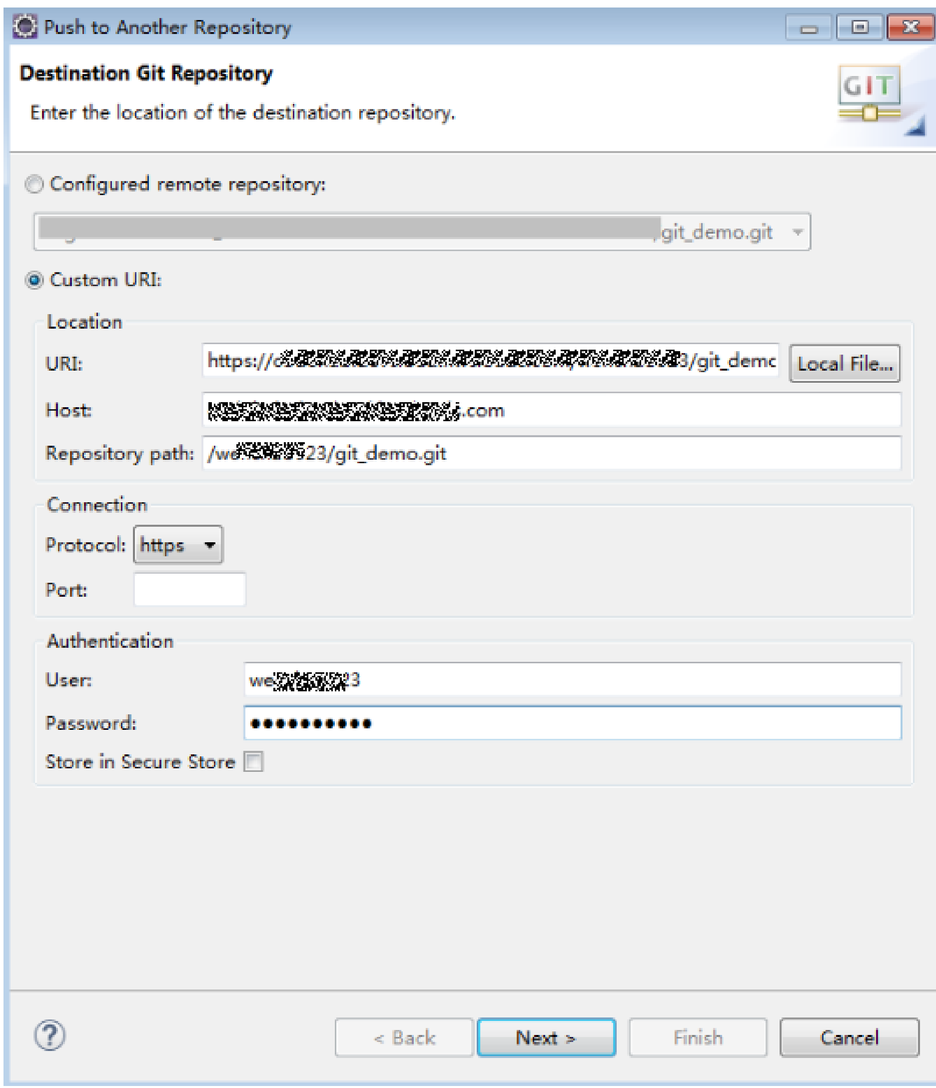
1. 在代码托管服务中创建仓库。

   创建好远程仓库后，进入远程代码仓库详情页面，可以复制远程仓库地址。

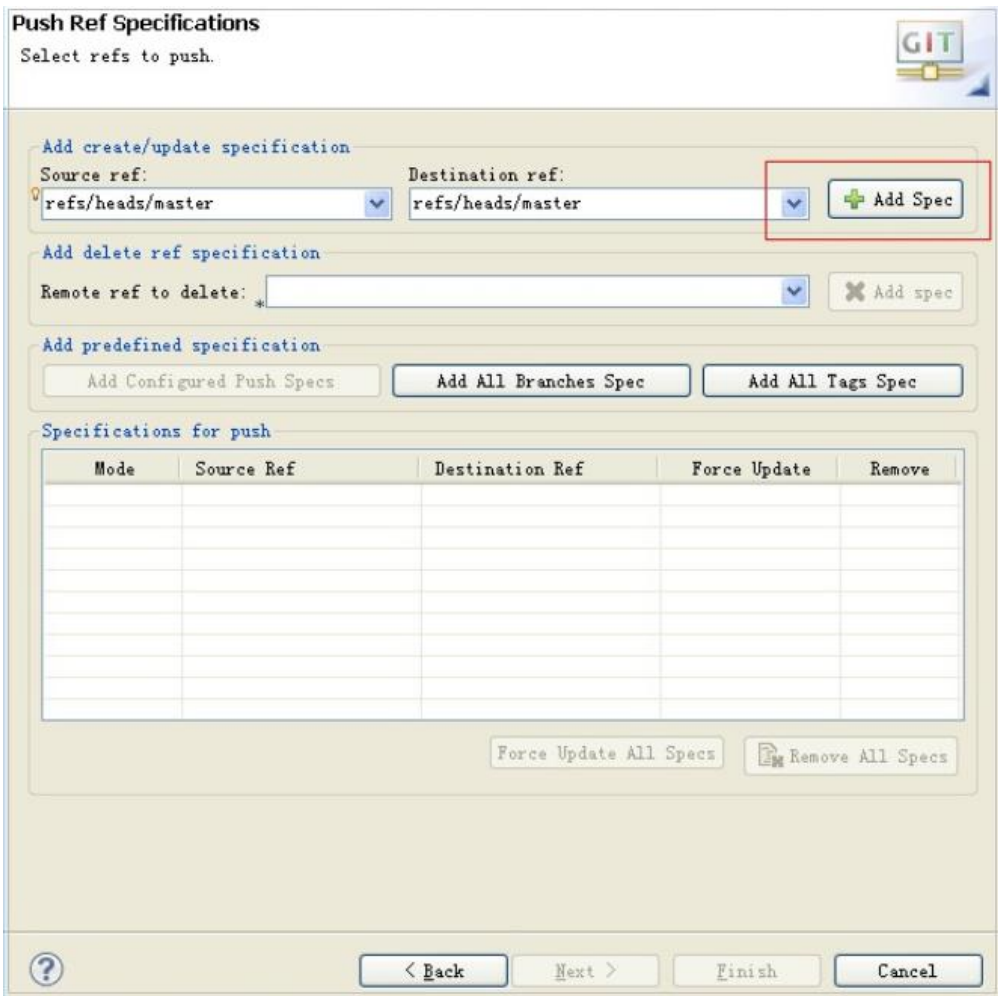
2. 选择 Push 菜单，开始将代码提交到远程仓库，如下图所示。



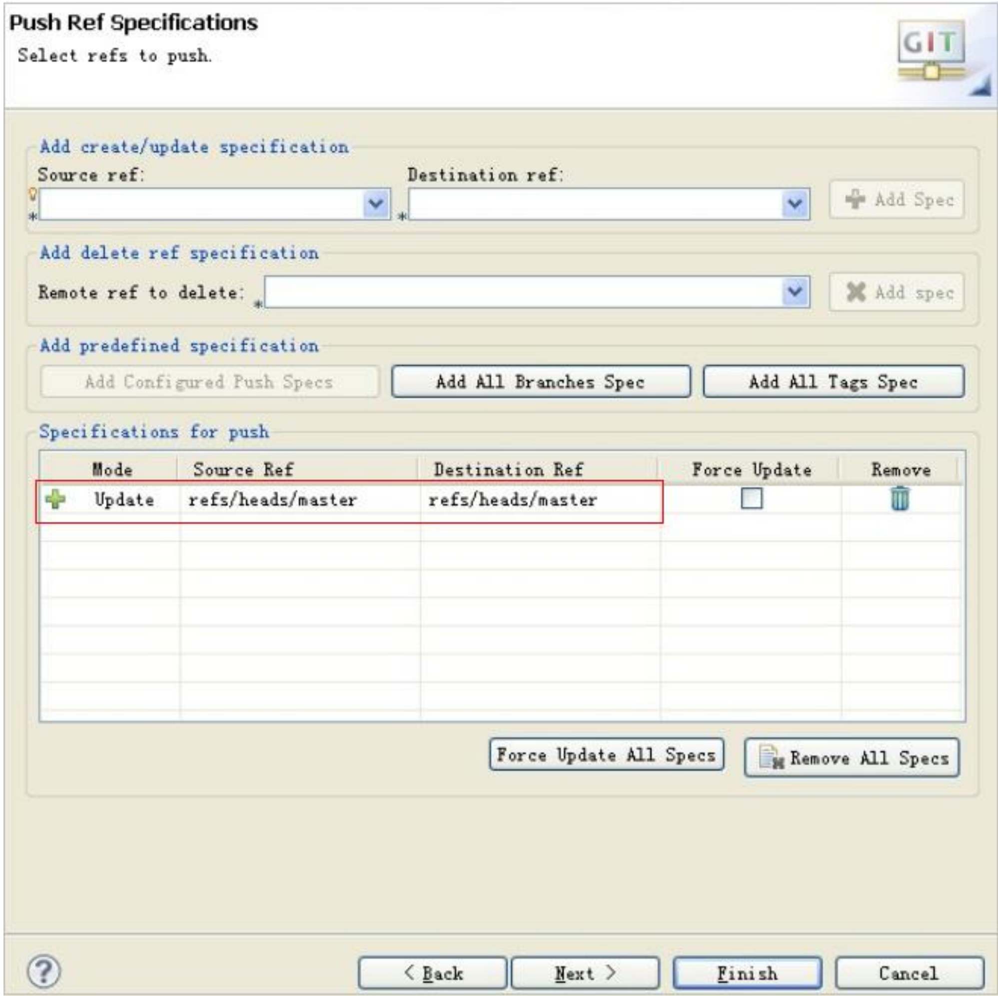
3. 在弹出的“Push to Another Repository”窗口中，设置相应参数，如下图所示。



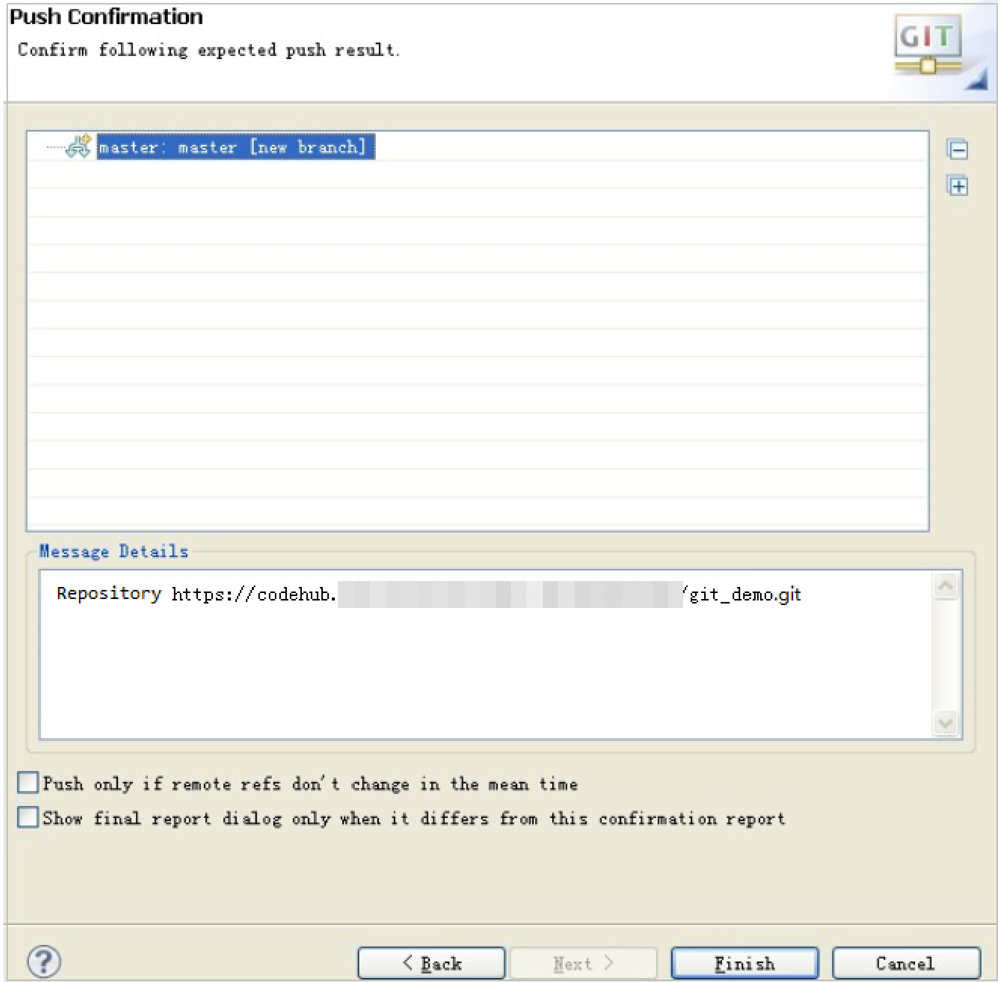
4. 单击“Next”，弹出“Push Ref Specifications”，如下图所示。



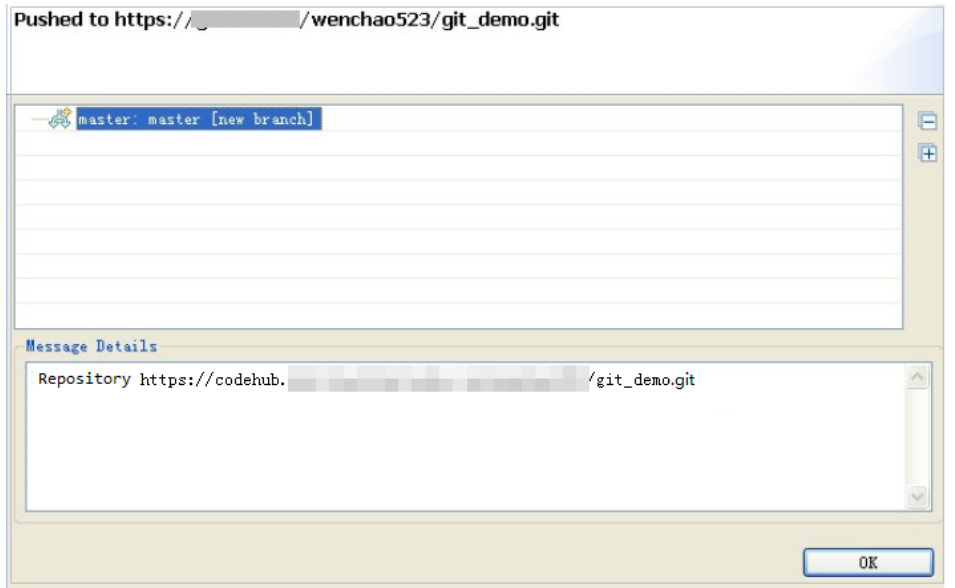
5. 单击“Add Spec”，成功添加，如下图所示。



6. 单击“Next”，弹出“Push Confirmation”窗口，如下图所示。



7. 单击“Finish”提交本地代码，如下图所示。



8. 单击“OK”，完成代码提交远程仓库。

   登录远程仓库地址，核对提交的代码。

11 更多 Git 知识

Git 客户端使用
使用 HTTPS 协议设置免密码提交代码
TortoiseGit 客户端使用
Git 客户端示例
Git 常用命令
Git LFS 使用
Git 工作流

11.1 Git 客户端使用

背景信息

使用 Git 客户端前，需要了解 Git 安装、创建新仓库、检出仓库、工作流、添加与提交、推送改动、创建分支、更新与合并分支、创建标签、替换本地改动等基本操作。

前提条件

已安装 Git 客户端。

使用流程

Git 客户端基本使用流程如下图所示。

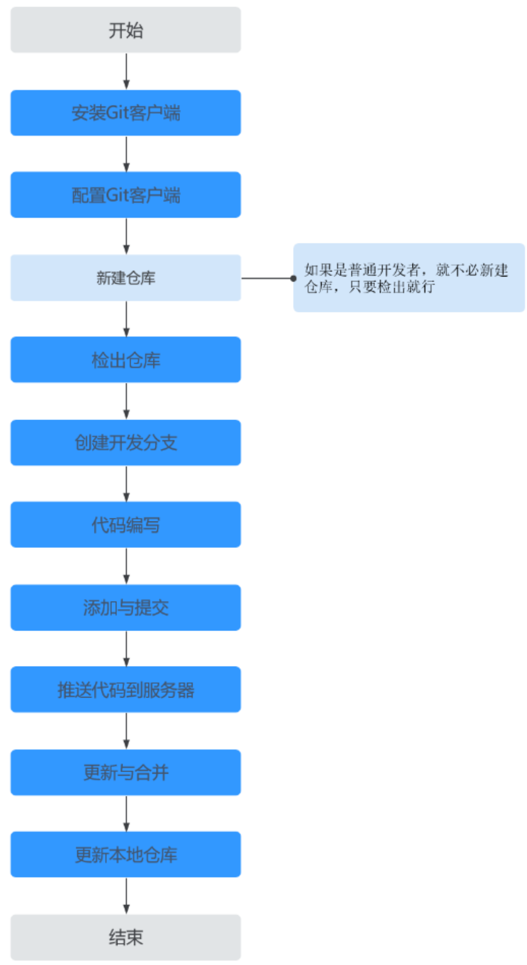


表11-1 使用流程说明

操作步骤	说明
安装	根据需要安装对应版本： • Git Windows 版 • Git OSX 版 • Git Linux 版
新建仓库	创建并打开新文件夹，然后执行如下命令： `git init` 以创建新的 Git 仓库。

检出仓库	创建一个本地仓库的克隆版本，执行如下命令：
	`git clone /path/to/repository`
	如果是远端服务器上的仓库，命令为：
	`git clone username@host:/path/to/repository`

工作流	本地仓库由 Git 维护的三棵“树”组成：

操作步骤	说明
	• 工作目录：持有实际文件。 • 缓存区（Index）：像个缓存区域，临时保存做的改动。 • HEAD：指向最近一次提交后的结果。

添加与提交	将改动内容添加到缓存区，使用如下命令：
	`git add <filename>`
	`git add *`
	这是 Git 基本工作流程的第一步；使用如下命令提交实际改动：
	`git commit -m "代码提交信息"`
	此时改动已经提交到了 HEAD，但是还没到远端仓库。

推送改动	改动内容目前已经在本地仓库的 HEAD 中。执行如下命令以将这些改动提交到远端仓库：
	`git push origin master`
	可以将 master 换成需要推送的任何分支。
	如果没有克隆现有仓库，且需要将仓库连接到某个远程服务器，可以使用如下命令添加：
	`git remote add origin <server>`
	将改动推送到所添加的服务器上。

创建分支	分支是用来将特性开发绝缘开来的。在创建仓库时，master 是“默认的”主分支。在其他分支上进行开发，完成后再将它们合并到主分支上。
	1. 创建一个名为“feature_x”的分支，并切换过去：
	`git checkout -b feature_x`
	2. 切换回主分支：
	`git checkout master`
	3. 将分支推送到远端仓库（不推送该分支，仅自己所见）：
	`git push origin <branch>`
	4. 再把新建的分支删掉：
	`git branch -d feature_x`

更新与合并（分支）	1. 更新本地仓库至最新改动，执行：
	`git pull`
	以在你的工作目录中获取（fetch）并合并（merge）远端的改动。
	2. 合并其他分支到当前分支（如 master），执行：
	`git merge <branch>`
	说明
	两种情况下，Git 都会尝试去自动合并改动。但自动合并并非次次都能成功，并可能导致冲突（conflicts）。这时候就需要修改这些文件，手动合并这些冲突（conflicts）。
	3. 改完之后，需要执行如下命令将它们标记为合并成功：

<table>
<tr><th>操作步骤</th><th>说明</th></tr>
<tr><td></td><td><pre><code>git add <filename></code></pre>4. 在合并改动之前，也可以使用如下命令查看：<pre><code>git diff <source_branch> <target_branch></code></pre></td></tr>
<tr><td>创建标签</td><td>在软件发布时创建标签，是被推荐的。可以执行如下命令以创建一个名为“1.0.0”的标签：<pre><code>git tag 1.0.0 1b2e1d63ff</code></pre>“1b2e1d63ff”是要标记的提交 ID 的前 10 位字符。使用如下命令获取提交 ID：<pre><code>git log</code></pre>也可以用该提交 ID 的少一些的前几位，保持 ID 唯一即可。</td></tr>
<tr><td>替换本地改动</td><td>如果误操作，可以使用如下命令替换掉本地改动：<pre><code>git checkout -- <filename></code></pre>此命令会使用 HEAD 中的最新内容替换掉工作目录中的文件。已添加到缓存区的改动，以及新文件，均不受影响。
如果需要丢弃所有的本地改动与提交，可以到服务器上获取最新的版本并将本地主分支指向到它：<pre><code>git fetch origin
git reset --hard origin/master</code></pre></td></tr>
</table>

11.2 使用 HTTPS 协议设置免密码提交代码

背景信息

为避免每次访问都输入用户名和密码，可以使用 Git 的凭证存储功能实现免密码访问（为保证该功能正常使用，建议安装 **Git2.5** 以上版本），不同操作系统的设置方法如下：

- Windows 系统如何使用 HTTPS 协议设置免密码提交代码。
- MAC 系统如何使用 HTTPS 协议设置免密码提交代码。
- Linux 系统如何使用 HTTPS 协议设置免密码提交代码。

前提条件

- 使用 HTTPS 协议访问代码仓库，需要设置 HTTPS 密码。
- 使用 HTTPS 协议方式在进行 Git clone、Git fetch、Git pull 以及 Git push 等操作时，需要输入代码托管的用户名和密码。

Windows 系统如何使用 HTTPS 协议设置免密码提交代码

Windows 系统设置免密码提交代码方法，如下表所示。

表11-2 Windows 系统设置免密码提交代码

方法	操作说明
本地配置 HTTPS 密码	1. 设置 Git 验证方式。 打开 Git 客户端：**git config --global credential.helper store** 2. 使用 Git 命令进行首次 Clone 或 Push 代码，根据提示填写用户名和密码。 3. 打开“.git-credentials”文件，如果已在本地存储了用户名和密码，会出现如下记录： `https://username:password@***.***.***.com`

MAC 系统如何使用 HTTPS 协议设置免密码提交代码

安装“osxkeychain”工具来实现免密码访问：

1. 查找当前系统是否已经安装该工具。

```
git credential -osxkeychain
# Test for the cred helper
Usage: git credential -osxkeychain < get|store|erase >
```

   如果回显如下，则为未安装。

```
git: 'credential -osxkeychain' is not a git command. See 'git --help'.
```

2. 如果该工具未安装，先获取安装包：

```
git credential -osxkeychain
# Test for the cred helper
git: 'credential -osxkeychain' is not a git command. See 'git --help'.
curl -s -o \
https://github-media-downloads.s3.amazonaws.com/osx/git-credential-osxkeychain
# Download the helper
chmod u+x git-credential-osxkeychain
# Fix the permissions on the file so it can be run
```

3. 将“osxkeychain”安装在 Git 的同一个目录下：

```
sudo mv git-credential-osxkeychain\
"$(dirname $(which git))/git-credential-osxkeychain"
# Move the helper to the path where git is installed
Password:[enter your password]
```

4. 使用“osxkeychain”工具将 Git 配置成免密码模式：

```
git config --global credential.helper osxkeychain
#Set git to use the osxkeychain credential helper
```

说明

第一次执行操作时会提示输入密码，输入后将由“osxkeychain”管理用户名和密码，后续再执行 Git 操作时将不再需要密码校验。

Linux 系统如何使用 HTTPS 协议设置免密码提交代码

Linux 系统提供两种可选的免密码访问模式：

- **cache** 模式：
  - 将凭证存放在内存中一段时间，密码永远不会被存储在磁盘中，并且在 15 分钟后从内存中清除：

```
git config --global credential.helper cache
#Set git to use the credential memory cache
```

  - 通过 **timeout** 选项设置过期时间，单位为秒：

```
git config --global credential.helper 'cache --timeout=3600'
# Set the cache to timeout after 1 hour (setting is in seconds)
```

- **store** 模式：

  将凭证用明文的形式存放在磁盘“home”目录下（默认是“~/.git-credentials”），永不过期，除非手动修改在 Git 服务器上的密码，否则永远不需要再次输入凭证信息。“git-credentials”文件内容如下：

  https://username:password@***********.com

  保存退出后，执行如下命令即可完成：

```
git config --global credential.helper store
```

报错处理

如果在使用 HTTPS 协议下载时提示“SSL certificate problem: self signed certificate”错误信息，请在客户端进行如下设置：

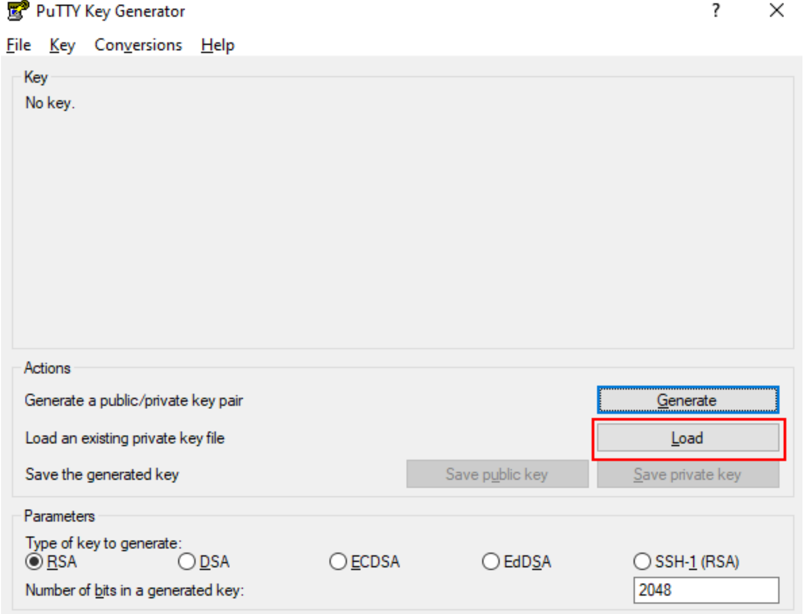
```
git config --global http.sslVerify false
```

11.3 TortoiseGit 客户端使用

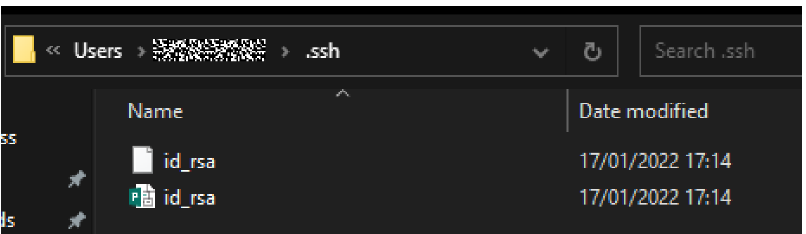
生成 PPK 文件

使用 TortoiseGit 作为客户端时，下载和提交代码需要 PPK 文件，主要有如下两种场景：

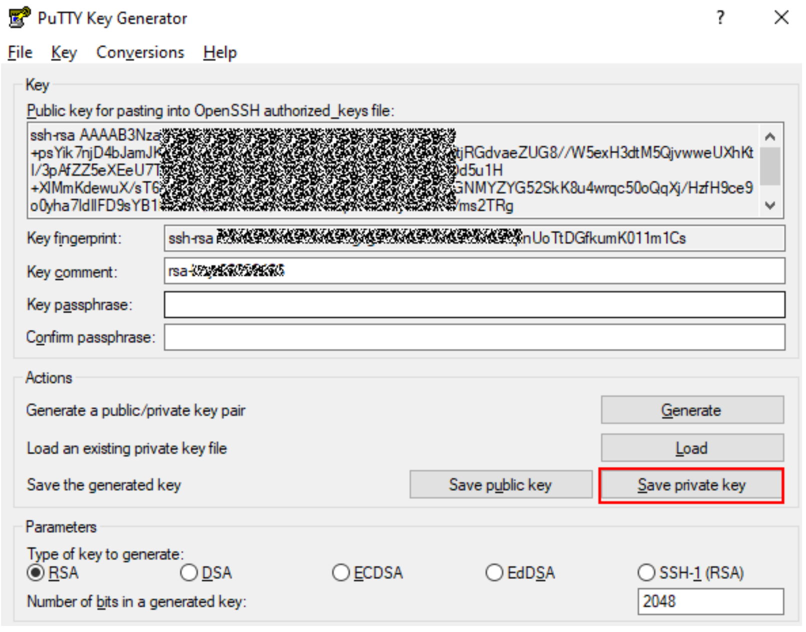
- **Git 客户端生成了公私钥对，并将该公钥添加到代码仓库的 ssh-key 配置中**
  a. 在“开始”菜单，搜索并选择“PuttyGen”。
  b. 单击“Load”按钮，如下图所示。



c. 选择您的公私钥目录下的“id_rsa”文件，然后单击“打开”按钮。

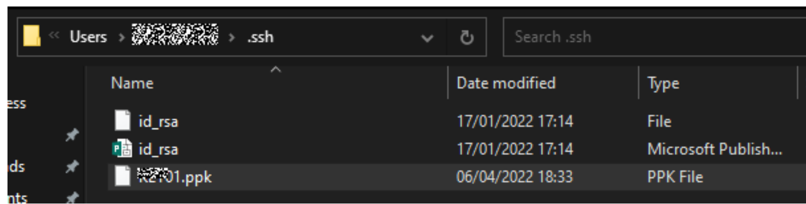


d. 单击“确定”，再选择“Save private key”，如下图所示。

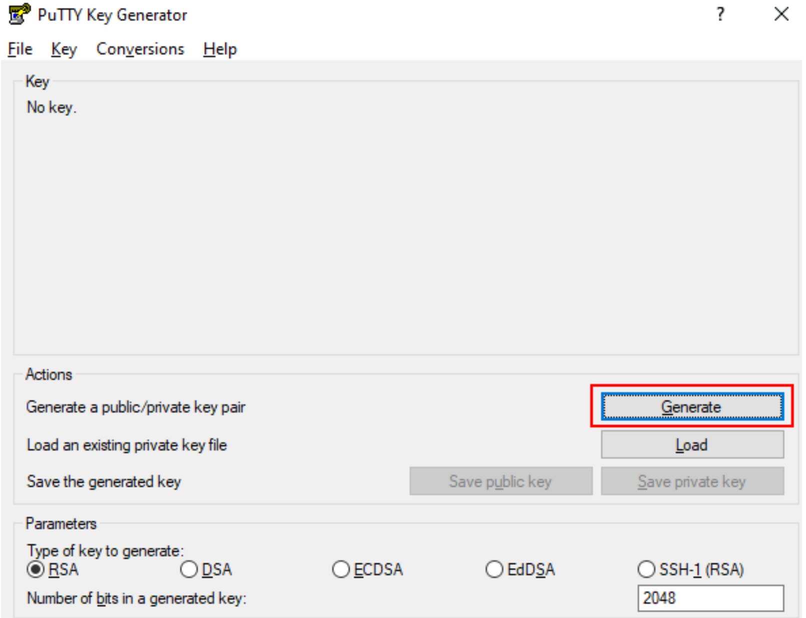


e. 根据提示单击“是(Y)”确定生成。

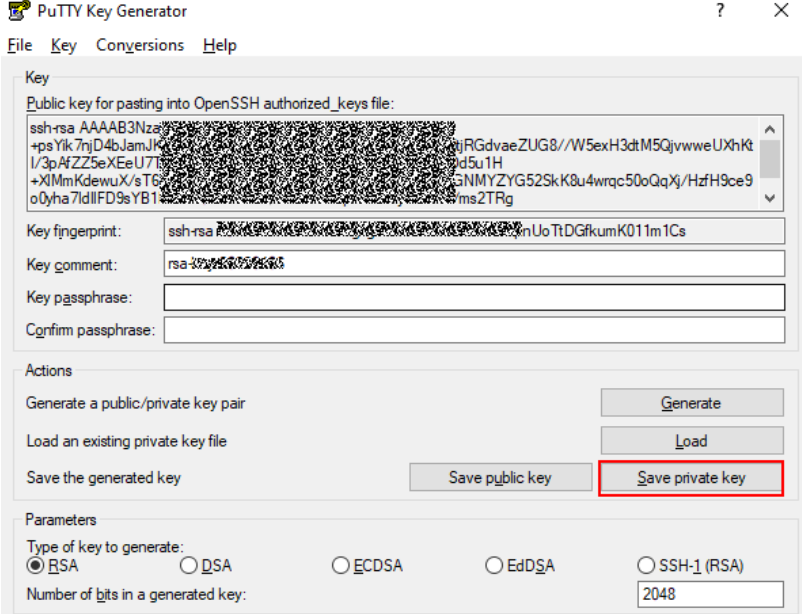
f. 保存到您的个人账户公私钥目录下，如下图所示。



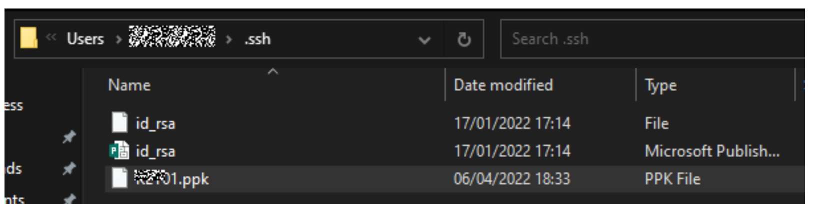
- **此前未添加互信操作，即没有添加公钥到代码仓库中**
  a. 在“开始”菜单，搜索并选择“PuttyGen”。
  b. 单击“Generate”按钮，即可生成密钥，如下图所示。



  c. 单击“Save private key”按钮，把生成的密钥保存为 PPK 文件，如下图所示。



  d. 根据提示单击“是(Y)”确定生成。
  e. 保存到您的个人账户公私钥目录下，如下图所示。

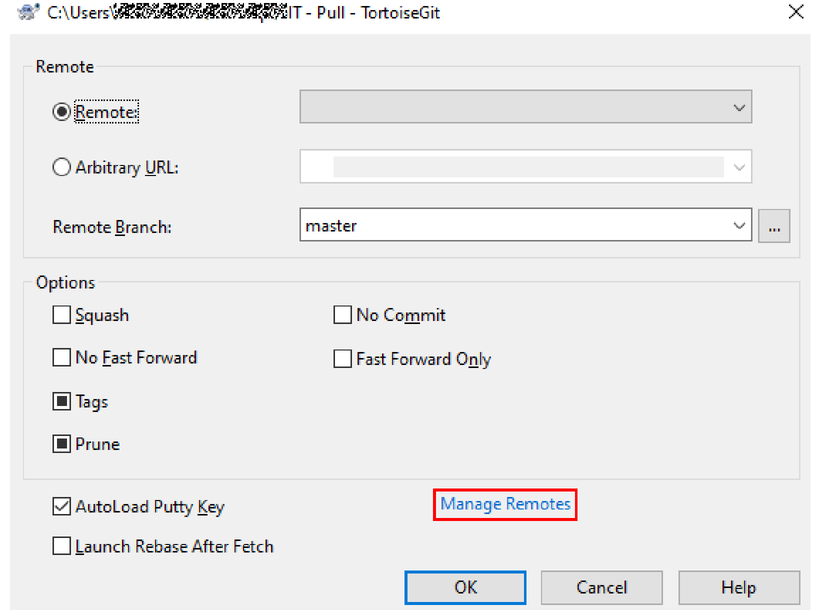


创建 Git 版本库

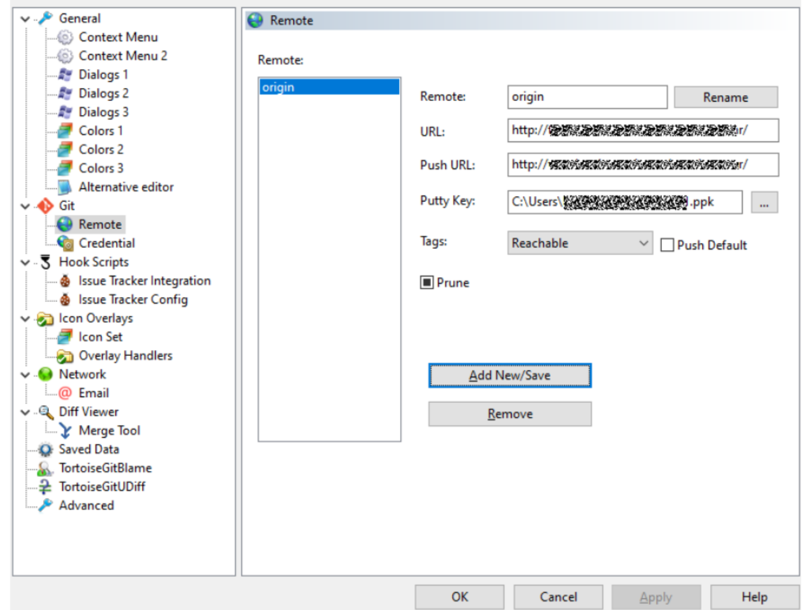
第一次建立版本库时，在本地任意空的文件夹下右键，选择“Git 在这里创建版本库”。

Clone 版本库

1. 选中本地 Git 库文件夹（即创建版本库的文件夹），右键选择“拉取（Pull）”。
2. 管理远端，如下图所示。

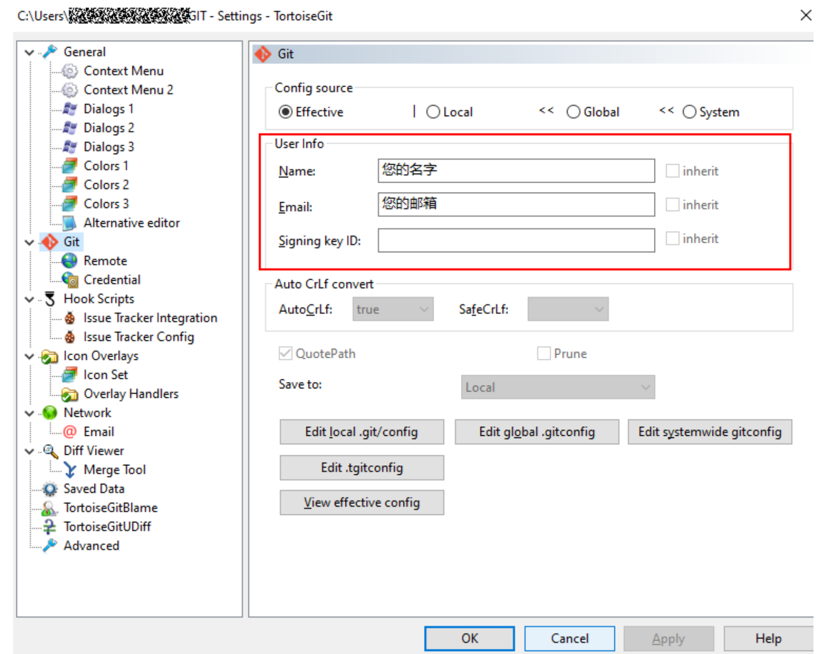


3. 配置相应的 **URL** 和 PPK 文件，单击“确定”，如下图所示。



Push 版本库

1. 配置用户名、邮箱和签名密钥（PPK 文件）。
2. 在空白处右键，选择“TortoiseGit > setting”。
3. 选中“Git”节点，如下图所示，填写用户名与邮箱地址。



说明

如果不能 Push，请运行如下脚本进行排查，并将生成的“git.log”发给售后支持：

```
#!/bin/bash
# this script will collect some logs for Coding.net
### how to use ###
# first enter your git reposiztory
# then execute this bash, please make sure you have correct rights
echo "## git version  ##################" >> git.log
git version  >> git.log
echo "## ping #########################" >> git.log
ping code*************.com  >> git.log
echo "## curl *************.com ###########" >> git.log
curl -v https://code*************.com >> git.log 2>&1
echo "## ssh -vT git@*************.com ##############" >> git.log
ssh -vT git@*************.com >> git.log 2>&1
echo "## git pull  ##############" >> git.log
GIT_CURL_VERBOSE=1 GIT_TRACE=1 GIT_TRACE_PACKET=1 git pull  >> git.log  2>&1
```

11.4 Git 客户端示例

11.4.1 Git 客户端上传下载代码

1. 检查网络。

   在客户端输入：**telnet *************.com 22**

   如果显示 **commant not found**，则说明网络无法访问代码托管服务；

2. 检查客户端和代码托管服务器互信。

   如果在 **pull**、**push** 时提示要输入密码，请确定是否已经添加了公钥文件。

   添加完成后，判断互信是否添加成功：**$ ssh -vT git@*************.com**

如果出现如下图所示信息则表示互信正常。

```
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
Welcome to GitLab, 100314597!
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: client_input_channel_req: channel 0 rtype eow@openssh.com reply 0
debug1: channel 0: free: client-session, nchannels 1
Transferred: sent 3536, received 3488 bytes, in 0.3 seconds
Bytes per second: sent 11491.6, received 11335.6
debug1: Exit status 0

MINGW64 /d/Gitlab
$
```

3. 如果已添加代码仓库的互信，双方指纹信息有变化，在提交代码时报公钥认证出错，需要做如下操作：
   a. 删除“~/.ssh/known_hosts”中***************.com** 相关的行。
   b. 重试 **push**、**pull** 或 **ssh -T git@*************.com**。
   c. 遇到询问是否信任服务器公钥，输入“yes”即可。
4. 代码正常下载，如果页面分支列表做了分支保护，提交的分支会显示在已受保护分支列表中，则向某个分支 Push 代码会不成功。
5. 联系仓库管理员，解除保护后，开发人员可以正常提交代码。

11.4.2 Git 客户端修改文件名大小写后，如何提交到远端

背景信息

Git 修改了文件名大小写后，远端将不识别对应文件。

例如：远端服务器“AppTest.java”，本地重命名后“apptest.java”，提交后远端还是“AppTest.java”。

操作步骤

请按顺序执行如下命令：

```
git mv --force AppTest.java apptest.java
git add apptest.java
git commit -m "rename"
git push origin XXX(分支)
```

11.4.3 Git 客户端设置系统的换行符转换

背景信息

由于操作系统不兼容的缘故，在跨平台上查看文本文件会因为换行符不同而造成障碍，使用版本控制系统也同样存在换行符的问题。

操作步骤

1. （可选）默认 Git 不对 **core.autocrlf** 进行配置，请设置如下值来辨别并对文本文件执行换行符转换：

– **Windows** 系统

设置配置变量“core.autocrlf”为“true”，相当于在版本库中所有的文本文件都使用“LF”作为换行符，而检出到工作区无论是什么操作系统都使用 **CRLF** 作为换行符。

– **Linux** 系统

设置配置变量“core.autocrlf”为“input”，相当于在新增入库的文本文件的换行符自动转换为“LF”，如果将文件从版本库检出到工作区则不进行文件转换。

2. 通过配置变量 **core.autocrlf** 来开启文本文件换行符转换的功能：

```
git config --global core.autocrlf true
```

11.4.4 Git 客户端提交隐藏文件

请使用 **git add .**。

说明

- 不能使用 **git add ***，使用*会忽略隐藏文件。
- 文件和文件夹名称中不能包含特殊字符。

11.4.5 Git 客户端提交已被更改的文件

背景信息

Git 提交代码时，如果文件被修改了，会出现如下图所示信息。

```
git.exe push --progress "origin" master:master

To git@
1.        .com:fdae56335080433a8298a5c72aed2fe6/          .git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@
1.        .com:fdae56335080433a8298a5c72aed2fe6/          .git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

操作步骤

1. 拉取远端最新代码。

```
git pull origin XXX(分支)
```

2. 正确修改并提交代码。

```
git push origin XXX(分支)
```

11.5 Git 常用命令

背景信息

- Git 是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或大或小的项目，能有效高速地处理从很小到非常大的项目版本管理。
- 通过 Git 可以从服务器克隆完整的 Git 仓库（包括代码和版本信息）到单机上，然后根据不同开发目的灵活创建分支、修改代码、提交代码、合并分支等。

常用命令

Git 常用命令的功能、格式、参数说明以及使用示例如下所示。

表11-3 Git 常用命令

命令	功能	格式	参数说明	使用实例
ssh－keygen －t rsa	生成密钥	ssh－keygen －t rsa －C [email]	email：邮箱地址	在 C 盘.ssh 文件夹下获取密钥文件“id_rsa.pub” ssh－keygen －t rsa －C "devcloud_key01@XXX.com"
git branch	新建分支。	git branch [new branchname]	new branchname：新的分支名	新建分支： git branch newbranch
git branch －D	删除分支	git branch －D [new branchname]	new branchname：新的分支名	删除本地分支： git branch －D newbranch 删除服务器仓库分支 git branch －rd origin/newbranch 同步远端已删除的分支 git remote prune origin
git add	添加文件到暂存区。	git add [filename]	filename：文件名	添加一个文件到暂缓区： git add filename 添加所有修改的和新增的文件到暂缓区： git add .
git rm	删除本地目录或文件。	git rm [filename]	filename：文件名或目录名	删除文件： git rm filename
git clone	克隆远程仓	git clone [VersionAddress]	VersionAddress：版本库的网	克隆 **jQuery** 的版本库： git clone

命令	功能	格式	参数说明	使用实例
	库。		址。	https://github.com/jquery/jquery.git 该命令会在本地主机生成一个目录，与远程主机的版本库同名。
git pull	把远程仓库的分支pull 到本地，再与本地的指定分支合并。	git pull [RemoteHostname] [RemoteBranchname]:[LocalBranchname]	-	取回“origin”主机的“next”分支，与本地的“master”分支合并： git pull origin next:master
git diff	文件、分支、目录或版本的比较。	git diff	-	当前与“master”分支的比较： git diff master
git commit	文件提交。	git commit	-	添加提交信息： git commit －m "commit message"
git push	推送文件到远程仓库。	git push [RemoteHostname] [LocalBranchname] [RemoteBranchname]	-	如果省略远程分支名，则表示将本地分支推送与之存在“追踪关系”的远程分支（通常两者同名），如果该远程分支不存在，则会被新建： git push origin master 上面命令表示，将本地的 master 分支推送到 origin 主机的 master 分支。如果后者不存在，则会被新建。
git merge	合并分支。	git merge [branch]	branch：分支名	假设当前分支为“develop”，将 master 主分支之后的最新提交 merge 到当前的 develop 分支上： git merge master

命令	功能	格式	参数说明	使用实例
git checkout	切换分支。	git checkout [branchname]	branchname：分支名	切换到 master 分支： git checkout master
git log	列出日志信息。	git log	-	列出所有的 log： git log －-all
git status	查看状态输出。	git status	-	git status
git grep	查找字符串。	git grep	-	查找是否有“hello”字符串： git grep "hello"
git show	显示内容或修改的内容。	git show	-	• git show v1 显示“tag v1”的修改内容 • git show HEAD 显示当前版本的修改文件 • git show HEAD^ 显示前一版本所有的修改文件 • git show HEAD~4 显示前 4 版本的修改文件
git stash	暂存区。	git stash	-	• git stash 用于保存和恢复工作进度 • git stash list 列出暂存区的文件 • git stash pop 取出最新的一笔，并移除 • git stash apply 取出但不移除 • git stash clear 清除暂存区
git ls-files	查看文件。	git ls-files	-	• git ls-files －d 查看已经删除的文件 • git ls-files －d |xargs git checkout 将已删除的文件还原
git	操作	git remote	-	• git push origin master:newbranch

命令	功能	格式	参数说明	使用实例
remote	远程。			增加远程仓库的分支 • git remote add newbranch 增加远程仓库的分支 • git remote show 列出现在远程有多少版本库 • git remote rm newbranch 删除远程仓库的新分支 • git remote update 更新远程所有版本的分支

11.6 Git LFS 使用

背景信息

- 代码托管支持 Git LFS（Large File Storage，大文件存储）协议，可以把音乐、图片、视频等指定的任意大文件资源存储在 Git 仓库之外，对于使用者而言，类似在操作一个完整的 Git 仓库，非常方便。通过将大文件存储在 Git 原有的数据结构之中，可以减小 Git 仓库本身的体积，使克隆 Git 仓库的速度加快，也使得 Git 不会因为仓库中充满大文件而损失性能。
- 当您要上传的文件单个超过 200MB 时，需要使用 Git LFS。
- 使用操作包含以下内容：
  - 安装 Git LFS
  - 配置追踪文件
  - 提交大文件
  - 克隆包含 Git LFS 文件的远程仓库

安装 Git LFS

不同操作系统的安装方法如下表所示。

表11-4 Git LFS 安装方法

操作系统	安装方法
Windows	安装不低于 Git 1.8.5 版的 Git 客户端，然后在命令行中执行： `git lfs install`
Linux	根据自己的操作系统和 cpu 架构在 PackageCloud 网站下载对应的安装包。 先解压安装包，再执行 install.sh 脚本进行安装，然后执行如下命令检查

<table>
<tr><th>操作系统</th><th>安装方法</th></tr>
<tr><td></td><td>是否安装成功：<pre><code>git lfs version</code></pre></td></tr>
<tr><td>macOS</td><td>首先安装 Homebrew 软件包管理工具，然后在命令行中执行：<pre><code>$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ brew install git-lfs
$ git lfs install</code></pre></td></tr>
</table>

配置追踪文件

配置追踪文件方法如下所示。

表11-5 追踪文件配置方法

<table>
<tr><th>场景</th><th>方法</th></tr>
<tr><td>追踪所有后缀名为“.psd”的文件</td><td>追踪所有后缀名为“.psd”的文件：<pre><code>git lfs track "*.psd"</code></pre></td></tr>
<tr><td>追踪单个文件</td><td>追踪单个文件：<pre><code>git lfs track "logo.png"</code></pre></td></tr>
<tr><td>查看已追踪的文件</td><td>查看已追踪的文件，可以通过 git lfs track，或通过查看“.gitattributes”文件，获取详情：<pre><code>$ git lfs track
Listing tracked patterns
    *.png (.gitattributes)
    *.pptx (.gitattributes)
$ cat .gitattributes
*.png filter=lfs diff=lfs merge=lfs -text
*.pptx filter=lfs diff=lfs merge=lfs -text</code></pre></td></tr>
</table>

提交大文件

提交代码时需要将“.gitattributes”文件也提交到仓库，提交完成后，执行 **git lfs ls-files** 命令可以查看 LFS 跟踪的文件列表。

```
$ git push origin master
Git LFS: (2 of 2 files) 12.58 MB / 12.58 MB
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 548 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
```

```
To <URL>
<SHA_ID1>..<SHA_ID2>  master -> master
$ git lfs ls-files
61758d79c4 * <FILE_NAME_1>
a227019fde * <FILE_NAME_2>
```

克隆包含 Git LFS 文件的远程仓库

使用 **git lfs clone** 命令克隆包含“Git LFS”文件的远程仓库到本地：

```
$ git lfs clone <URL>
Cloning into '<dirname>'
remote: Counting objects: 16,done.
remote: Compressing objects: 100% (12/12),done.
remote: Total 16 (delta 3), reused 9 (delta 1)
Receiving objects: 100% (16/16),done.
Resolving deltas: 100% (3/3),done.
Checking connectively...done.
Git LFS: (4 of 4 files) 0 B / 100 B
```

11.7 Git 工作流

11.7.1 Git 工作流概述

什么是 Git 工作流？你可以理解为代码管理的分支策略，它不仅仅是版本管理范畴，更服务于项目流程管理和团队协同开发。所以，有必要制定适合自己研发场景的工作流。

下面介绍四种工作流的工作方式、优缺点，以及使用中的一些注意事项。

- 集中式工作流
- 功能分支工作流
- Git flow 工作流（推荐）
- Forking 工作流

研发团队可以根据实际研发场景制定合理的工作流，能有效提高项目管理水平和团队协同开发能力， 并通过 CodeArts Repo 平台，高效、安全的管理代码资产，将更多的精力集中在业务开发上，实现持续集成、持续交付和快速迭代的目标。

11.7.2 集中式工作流

集中式工作流适合 5 人左右小开发团队，它只有一个默认的 maste 分支，所有人的修改都是在 master 分支上进行的。但是，这种工作流无法充分发挥 git 优势和多人协同，不推荐使用。

工作方式

开发人员将 master 分支从中央仓库克隆到本地，修改完成后再推送回中央仓库 master 分支。

优点

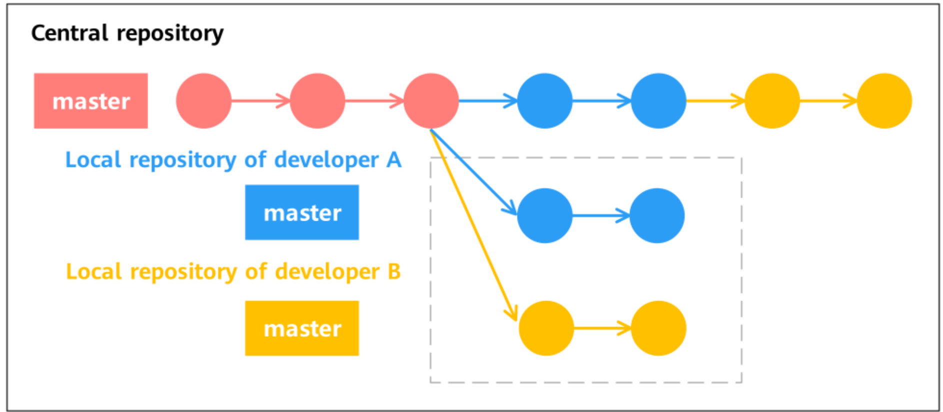
不涉及分支交互操作。

缺点

- 不适合人员较多的团队，当人员 10+时，解决开发人员之间的代码冲突会耗费很多时间。
- master 分支因提交频繁会导致分支不稳定，不利于集成测试。

Tips：如何尽量避免产生冲突和不合理的提交历史？

开发人员在开发一个新功能之前，一定要在本地同步中央仓库最新代码，使自己的工作基于最新的代码之上；开发完成后，在提交新功能到中央仓库前，需要先 fetch 中央库的新增提交，并 rebase 自己的提交。这样做的目的是，把自己的修改加到中央仓别人已经提交的修改之上，使最终的提交记录是一个完整的线性历史，而不是环形，工作流举例如下图所示。



1. 开发人员 A 和开发人员 B 同时在某个时间拉取了中央仓库的代码。
2. 开发人员 A 先完成了自己的工作，并提交到中央仓库。
3. 开发人员 B 需要在本地执行 **git pull －rebase** 中央仓库的新提交，这时开发人员 B 的本地仓库就包含了开发人员 A 修改的内容，并在 A 的基础上增加了自己的修改。
4. 开发人员 B 将代码推送到中央仓库。

11.7.3 分支开发工作流

通过新建几个功能分支，增加开发者的交流和协作，它的理念是所有的功能开发都应该在 master 分支外的一个独立分支进行，这种方式隔离了开发者的工作空间不被互相干扰，保证了 master 分支的稳定性。

工作方式

开发人员每次在开始新功能开发前，需要在 master 分支上拉取一个新分支，并起个有描述性的名字，比如 **video-output** 或 **issue-#1061**，这样可以让分支用途明确。功能分支不但存在开发人员本地仓库，也应该推送到中央仓库，这样就可以在代码不合入 master 分支的情况下与其他开发人员分享代码。

优点

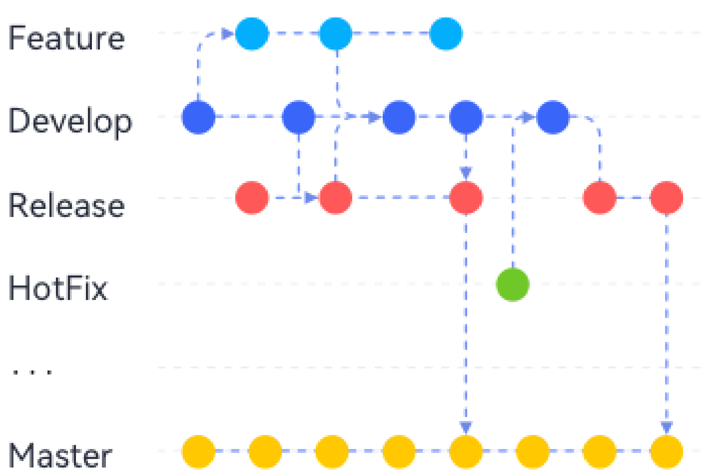
- 分支合并前可以使用 **pull request** 进行 **code review**。
- 降低了 master 分支的提交频率。

缺点

只有一个 master 分支作为集成，仍然不是很稳定，不适合大型开发。

11.7.4 Git flow 工作流

Gitflow 一般用于管理大型项目，它为不同的分支分配一个很明确的工作角色，并定义分支之间什么时候进行交互，如 Gitflow 工作流如下图所示。



工作方式

- **master 分支：**

  生产分支，最稳定的版本，一直是 ready to deploy 状态。不接受开发人员直接 commit，只接受从其他分支 merge 操作。在很多企业中，这个分支被默认开启分支保护，只有维护者可以操作。
- **hotfix 分支：**

  从 master 分支拉取的临时修复分支，用于解决一线紧急 bug。bug 解决后需要合入 master 分支并打上新的版本号，这个修改也需要同时合入 develop 分支。
- **develop 分支：**

  从 master 分支拉取的开发分支，用于功能集成。包含所有要发布到下一个 Release 的代码用于开发集成、系统测试。
- **release 分支：**

  临近既定的发布日，就从 develop 分支上拉取一个 release 分支，任何不在当前分支中的新功能都推到下个发布中。release 分支用于发布，所以从当前时间点之后新的功能不能再加到这个分支上，这个分支只做 Bug 修复、文档生成和其它面向发布的任务。当对外发布的工作都完成了，release 分支合并到 master 分支并分配一个版本号打好 Tag；另外，这些从 release 分支新做的修改要反向合并回 develop 分支。

- **feature 分支：**

  开发者使用的特性分支，父分支是 develop 分支，当新功能完成时，合入 develop 分支。新功能提交从不直接与 master 分支交互。

优点

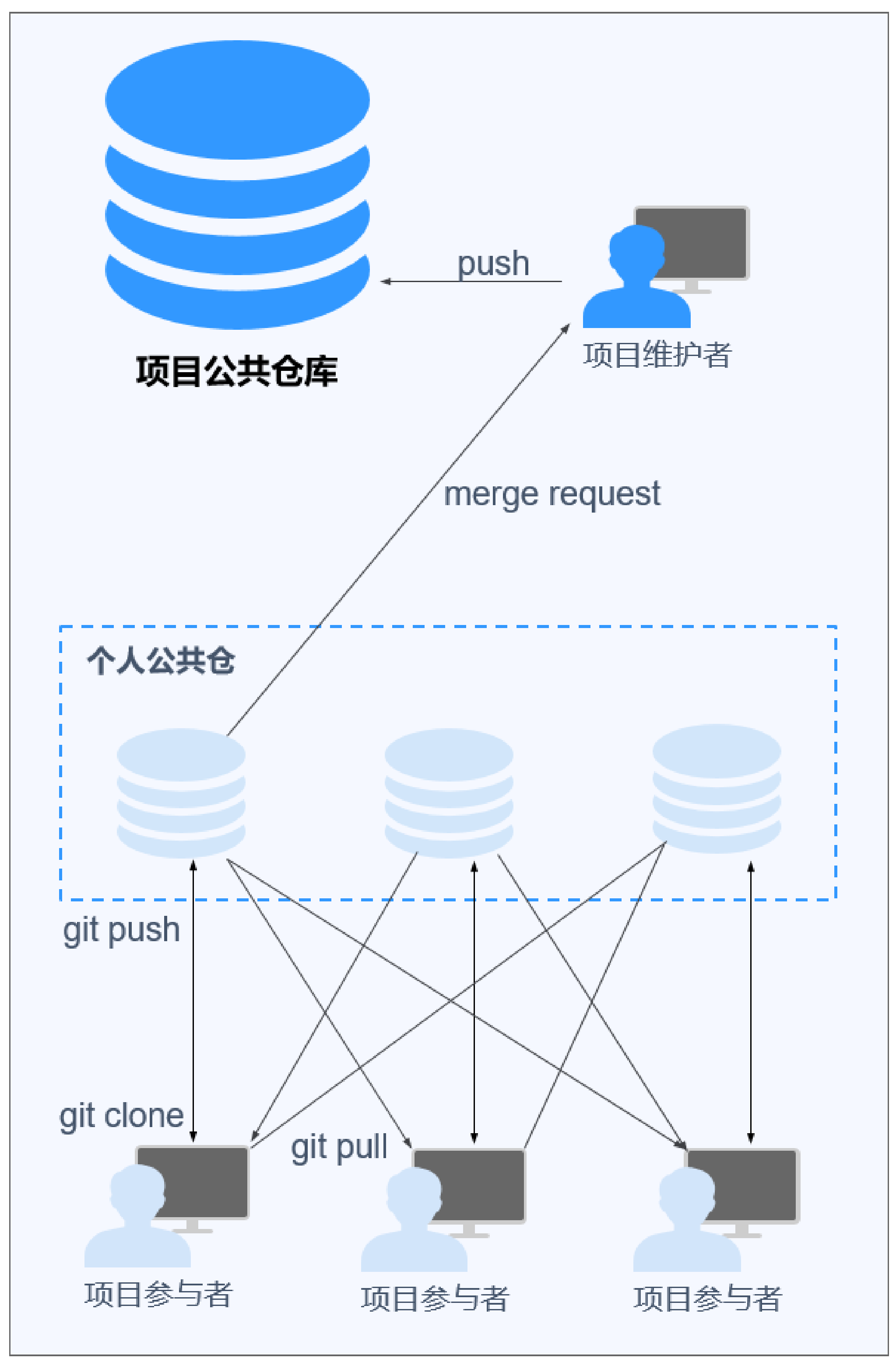
- 使用一个用于发布准备的专门分支（release 分支），使得一个团队可以在完善当前的发布版本的同时，可以在 develop 分支并行继续开发下个版本的功能。这也打造了可视化的发布阶段，团队成员都可以在仓库网状结构中可以看到发布状态。
- 使用紧急修复分支（hotfix 分支）让团队可以处理紧急问题的同时而不打断其它工作或是等待下一个发布再合入 hotfix 修改。您可以把 hotfix 分支想成是一个直接在 master 分支上处理的临时发布。
- 大型项目人员协作频繁，流程较多，合理的多角色分支帮助研发有条不紊进行。
- 更符合 devops 理念。

缺点

- 学习成本较高。
- 如果团队不遵守使用约定，带来的影响更大。

11.7.5 Forking 工作流

Forking 工作流区别于前三种工作流的最大特点是每个开发人员都有一个从公共仓库 fork 出来的属于自己的公共仓。Forking 工作流适合外包、众包以及众创和开源场景。接包方的开发人员从项目公共仓 fork 自己的公共仓库进行操作，并不需要被项目公共仓直接授权，Forking 工作流如下图所示。



工作方式

1. 将“项目公共仓”fork 出一个“个人公共仓”。
2. 将“个人公共仓”clone 到“本地仓库”。

3. 操作“本地仓库”，修改完成后提交到“个人公共仓”。
4. 为“个人公共仓”提交一个 pull request 给项目维护者，申请代码合入“项目公共仓”。
5. 项目维护者在本地 review、验证本地提交，审核通过后 push 进入“项目公共仓”。

说明

如果开发人员 A 的代码未被审核通过合入“公共仓库”，而此代码对开发人员 B 有借鉴作用，开发人员 B 可以直接从开发人员 A 的“个人公共仓”拉取代码。

优点

- 开发人员之间如果需要代码协作，可以直接从其他人的“个人公共仓”拉取，无需等到代码提交到项目公共仓。
- “项目公共仓”无需为每个代码贡献者授权。
- 项目维护者通过审核 pull request 成为代码安全的重要防线。
- 仓库分支的选择可以根据项目实际情况综合使用前三种工作流。

缺点

提交开发人员代码到最终版本库的周期较长，步骤繁琐。