代码托管

用户指南

文档版本 01

发布日期 2024-09-25

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

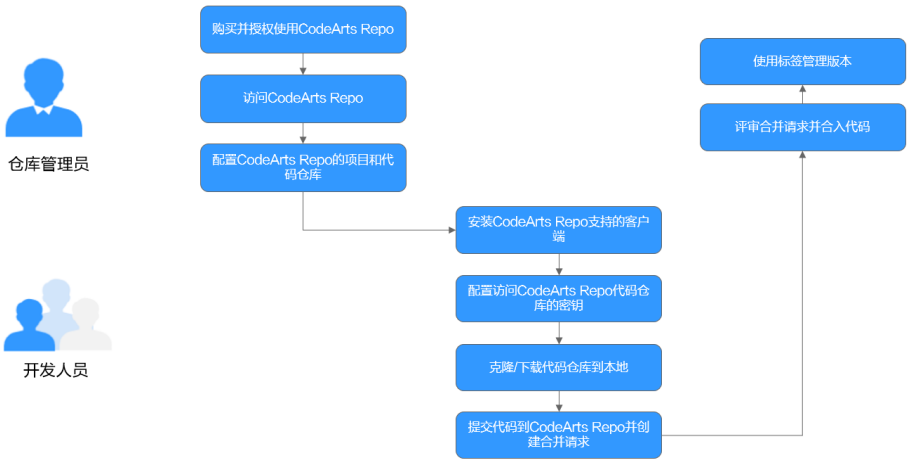
地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 代码托管(CodeArts Repo)使用流程

代码托管(CodeArts Repo)的使用流程如下图所示。



2 购买并授权使用 Repo

前提条件

在购买CodeArts Repo前，您需要已拥有租户账号或者Tenant Administrator权限的IAM用户账号，配置权限的策略请参考**创建用户组并授权**。

购买 CodeArts Repo 套餐

步骤1 使用IAM账号**登录CodeArts Repo购买页面**。

步骤2 在购买CodeArts Repo套餐页面，参考**下表**填写购买参数。

表 2-1 购买 CodeArts Repo 套餐参数表格

参数	说明
计费模式	该参数不可修改，默认为包年/包月。CodeArts Repo套餐的计费模式为包月或者包年。
区域	该参数必填。当前CodeArts Repo中国站支持如下局点：华北-北京一、华北-北京四、华东-上海一、华东-上海二、华南-广州、西南-贵阳一。 **说明** • 不同区域购买的资源不能跨区域使用，请谨慎选择。 • 中国站支持购买国际站区域。
产品	该参数不可修改，默认为CodeArts Repo套餐。

参数	说明
规格	该参数必填，根据您的需要，选择基础版或者专业版套餐。 • 基础版，该套餐提供以下功能：分支权限管理、代码评审、仓库配置和工作项关联。您可以使用总容量不超过50GB的代码仓库，每个仓库的容量最大为10GB，每次推送的文件大小不超过200MB，并且您可以创建任意数量的代码仓库。 • 专业版，该套餐提供以下功能：包含基础版所有功能，并且提供合并请求模板、检视意见分类及模板和星级评价。您可以使用总容量不超过500GB的代码仓库，每个仓库的容量最大为20GB，每次推送的文件大小不超过300MB，并且您可以创建任意数量的代码仓库。 **说明** 基础版更适用于个人开发者和小微型企业，专业版适用于中大型企业。
购买人数	该参数必填，根据您的需要，选择购买人数，至少1人，最多9999人。
购买时长	该参数必填，根据您的需要，选择购买时长，支持购买1~9个月、1~3年。您还可以根据需要，选择是否勾选自动续费，请参考**自动续费**规则，关于续费时长，如果您是按月购买，每次续费1个月，次数不限；如果您是按年购买：每次续费1年，次数不限。
协议	该参数必填。

步骤3 填完购买参数后，确认订单内容无误后，单击“去支付”，付款后，进入**Repo控制台页**，左上角切换到您购买的区域，可查看到购买的套餐信息。

----结束

说明

- 如果您当前正处于套餐期，无法进行购买操作。
- 如果您想要体验一站式体验一站式、全流程、安全可信的软件开发生产线(CodeArts)，您可以使用IAM账号登录**开通/购买软件开发生产线服务组合套餐**，购买CodeArts套餐，CodeArts套餐包含的Repo套餐。

购买资源扩展

代码托管服务支持对存储容量扩展，详情介绍请参见资源扩展。

步骤1 进入购买资源扩展页面。

步骤2 根据需要选择区域、产品、购买时长、是否自动续费相关配置项，勾选同意声明后单击“下一步：确认订单”。

步骤3 确认订单内容：如果需要修改，单击“上一步”；如果确认无误，单击“下一步”。

步骤4 根据页面提示完成支付。

----结束

3 访问 Repo 首页

从帮助中心首页进入 Repo 首页

步骤1 登录华为云控制台页面。单击页面左上角 ☰ ，在服务列表中选择“开发与运维 > 代码托管CodeArts Repo”。

步骤2 CodeArts Repo页面有两种访问方式：首页入口和项目入口。

- **首页入口**

  单击“立即使用”，进入CodeArts Repo首页。该页面展示的是与当前用户下的仓库列表。

- **项目入口**

  a. 单击“立即使用”，进入CodeArts Repo首页。

  b. 单击导航栏“首页”。

  c. 单击需要查看的项目名称。

  d. 选择“代码 > 代码托管”，进入指定项目下代码仓库列表页。

----结束

4 环境和个人配置

4.1 Git 客户端安装与配置

Repo当前支持的客户端及安装指导链接请参见表4-1。

表 4-1 Repo 支持的 Git 客户端

客户端名称	操作系统	官方的安装指导链接
Git客户端	Windows系统	Windows Git客户端安装指导
	Linux系统	Linux Git客户端安装指导
	Mac系统	Mac Git客户端安装指导
TortoiseGit客户端	Windows系统	Windows TortoiseGit客户端安装指导

安装好Windows Git客户端后，需要配置用户名和邮箱，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
```

说明

- CodeArts Repo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。
- CodeArts Repo当前支持TLS1.2和TLS1.3版本协议，在Git为最新版本的前提下，您可以执行如下命令指定TLS协议版本。其中，test.com为在CodeArts Repo下Git上传/下载的域名，tls1_2表示指定TLS协议版本为TLS1.2。不同Git客户端的解决方案您可以参考适配CodeArts Repo支持的TLS协议版本。

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

4.2 密钥概述

Repo的代码仓库支持SSH和HTTPS两种访问协议，您可以选择以下两种方式之一进行配置。

- SSH密钥是一种安全的连接方式，用于在本地计算机与您账号下的Repo之间建立安全连接。不同的用户通常使用不同的计算机，因此在使用SSH方式连接Repo代码仓库前，需要在自己的电脑上生成自己的SSH密钥，并将公钥添加到Repo中。一旦在本地计算机上配置了SSH密钥，并添加公钥到Repo中，此账号下的所有代码仓库与这台计算机之间都可以使用该密钥进行连接。
- HTTPS密码是一种用于HTTPS协议方式下载、上传时使用的用户凭证。

须知

- 在Repo中，HTTPS协议所支持的单文件推送大小不超过200M。如果需要传输大于200M的文件，请使用SSH方式。
- 可以绑定邮箱的账号才能使用HTTPS协议。

- GPG(GNU Privacy Guard)是一种用于数字签名和认证的手段。当您需要将本地代码推送到代码托管仓库时，GPG公钥在Git中用于对代码的提交和Tag进行签名和验证，以确保提交的来源可信以及代码的完整性。

4.3 配置 SSH 密钥

步骤1 运行Git Bash，先检查本地是否已生成过SSH密钥。请在Git Bash中执行如下命令：

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

- 如果提示“No such file or directory”，说明您这台计算机没生成过SSH密钥，请继续执行**步骤2**。
- 如果返回以ssh-rsa开头的字符串，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3**，如果想重新生成密钥，请从**步骤2**向下执行。

步骤2 生成SSH密钥。在Git Bash中生成密钥的命令如下：

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

其中，-t rsa表示生成的是RSA类型密钥，-b 4096是密钥长度（该长度的RSA密钥更具安全性），-C your_email@example.com表示在生成的公钥文件中添加注释，方便识别这个密钥对的用途。

如果选择ED25519算法，在Git Bash中生成密钥的命令如下：

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

其中，-t ed25519表示生成的是ED25519类型密钥，-b 521是密钥长度（该长度的ED25519密钥更具安全性），-C your_email@example.com表示在生成的公钥文件中添加注释，方便识别这个密钥对的用途。

输入生成密钥的命令后，直接回车，密钥会默认存储到~/.ssh/id_rsa路径下，对应的公钥文件为~/.ssh/id_rsa.pub。

步骤3 复制SSH公钥到剪切板。请根据您的操作系统，选择相应的执行命令，将SSH公钥复制到您的剪切板。

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

步骤4 登录并进入Repo的代码仓库列表页，单击右上角昵称，选择“个人设置” > “代码托管” > “SSH密钥”，进入配置SSH密钥页面。

也可以在Repo的代码仓库列表页，单击右上角“设置我的SSH密钥”，进入配置SSH密钥页面。

步骤5 在**“标题”**中为您的新密钥起一个名称，将您在**步骤3**中复制的SSH公钥粘贴进**“密钥”**中，单击确定后，弹出页面“密钥已设置成功，单击 立即返回，无操作3S后自动跳转”，表示密钥设置成功。

----结束

说明

- 在一台电脑上配置了SSH密钥并添加公钥到CodeArts Repo中后，所有该账号下的代码仓库与这台电脑之间都可以使用该SSH密钥进行连接；而不同的用户通常使用不同的电脑。因此，在使用SSH方式连接CodeArts Repo之前，每个用户都需要在自己的电脑上配置各自的SSH密钥。
- 在配置SSH密钥时，提示：“此密钥已存在，请重新生成密钥”，表示该密钥在该账号或者其它账户下被添加过。解决办法：可参考如上操作步骤，在本地重新生成一次SSH密钥，再把生成的密钥配置到CodeArts Repo。

4.4 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密码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**

  由租户名和IAM用户名组成，请完整输入，格式为“租户名/IAM用户名”。
- **HTTPS密码**

  密码长度为8到32位字符，至少包含数字、大小写字母及特殊字符至少三种字符类型，且不能与HTTPS密码的“用户名”或者倒序的“用户名”相同。

说明

- 每个用户只需要设置一次HTTPS密码。
- 为保证代码仓安全，需要妥善保存您的HTTPS密码，建议定期更换。

设置 HTTPS 密码

HTTPS密码默认使用您的登录密码，支持密码实时同步，您也可以根据如下步骤操作，通过“自行设置密码”来修改密码。

步骤1 进入代码托管服务仓库列表页，单击右上角昵称，选择“个人设置 > 代码托管 > HTTPS密码”。

您也可以进入代码托管的仓库列表页，单击右上角“设置我的HTTPS密码”，进入“HTTPS密码”页面。

步骤2 单击“自行设置密码”，进入重设密码页面。

步骤3 填写新密码与邮箱验证码，勾选“我已阅读并同意《隐私政策声明》和《CodeArts服务使用声明》”，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 > 用户帐户 > 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和密码。

----结束

说明

- 如果用户账号升级为华为账号，租户级的**“使用华为云登录密码”**功能将不再获得支持（子账号级仍然有效）。
- 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。
- 因为**联邦账号**无法绑定邮箱，所以无法使用HTTPS协议。
- 在执行步骤4时，如果界面提示“SSL certificate problem”，请在Git客户端执行**git config --global http.sslVerify false**，关闭Git的SSL验证功能。

验证 HTTPS 密码是否生效

当设置好HTTPS密码后，您可以在Git Bash执行**git clone https://username:password@example.com/repo_path.git**克隆您有权限访问的代码仓。其中，“username”为您配置的HTTPS用户名，“password”为您配置的HTTPS密码，“example.com/repo_path.git”为您要克隆的代码仓的https地址。如果根据此命令，成功克隆代码，说明HTTPS密码设置成功。

4.5 配置访问令牌

登录您的代码托管服务仓库列表页，单击右上角昵称，选择“个人设置 > 代码托管 > 访问令牌”，单击“新建Token”，参考下列表格填写参数。

表 4-2 参数说明

参数	说明
Token名称	必填参数。自定义名称，字符上限为200。
描述	非必填参数。此处描述为空，列表会显示“--”。字符上限为200。
权限	默认勾选且不可修改。读/写仓库：授予使用Https方式，读写访问仓库权限。

参数	说明
失效时间	必填参数。设置Token失效的时间。 **说明** 默认为当前日期的30天之后，包含当天共30天。例如7月3号新建的Token，失效日期默认为8月2号23点59分59秒。其中，失效日期最长可设置为1年，且不可为空。

填写完上述参数后，Token成功生成，请复制此Token，并在应用或脚本中使用。

须知

- 为保证仓库权限，关闭此弹窗后Token将不再展示，请妥善保管，如遗失或忘记可重新生成。
- CodeArts Repo生成Token数量上限为20个。

4.6 配置 GPG 公钥

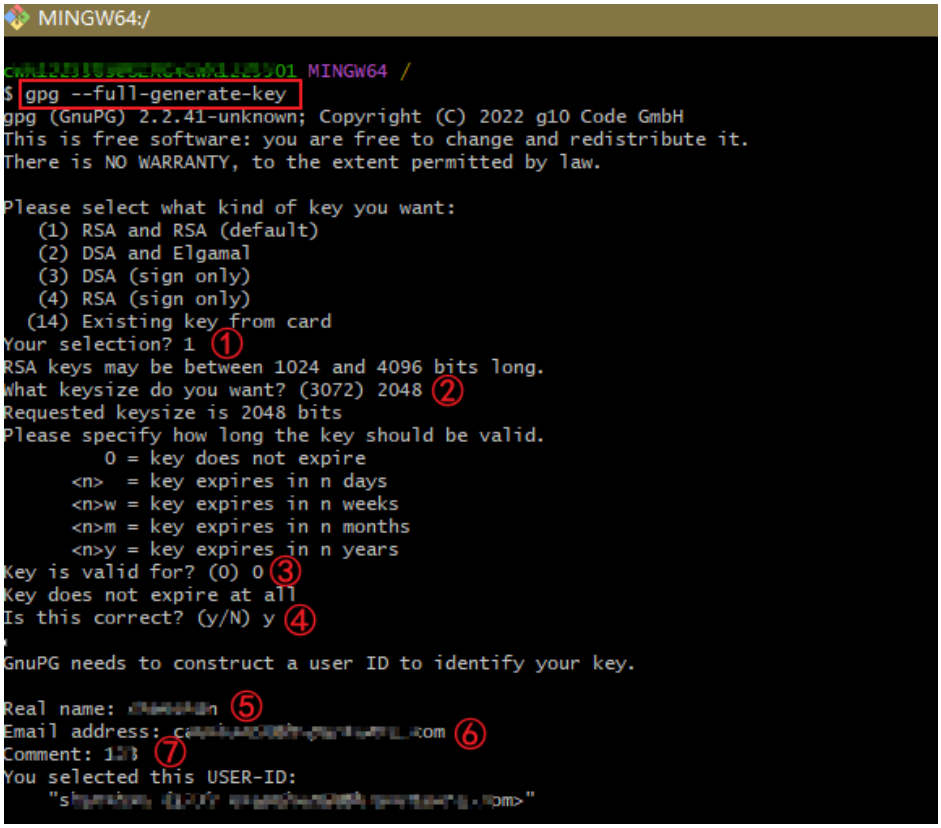
您可以根据如下步骤，在CodeArts Repo生成和配置GPG公钥。

- **在gpg4win官网下载GPG密钥生成工具。**
- **生成GPG密钥对。**
- **验证GPG密钥生成是否成功。**
- **复制GPG密钥对到剪切板。**
- **进入GPG密钥配置页面。**
- **填写“新建GPG”公钥参数。**
- **检查GPG公钥是否配置成功。**

步骤1 在**gpg4win官网**下载GPG密钥生成工具。

步骤2 在本地Git客户端执行**gpg --full-generate-key**命令，按照提示，依次选择加密算法、密钥长度、过期时间、正确性后，输入用户名、邮箱、注释，如**图4-1**所示。

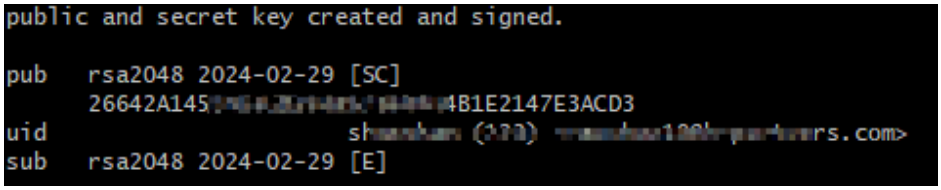
图 4-1 生成 GPG 密钥对



步骤3 确认信息无误后，按照提示，输入**O**并按回车键，弹出输入密码窗口和确认密码窗口，正确输入密码后生成密钥。

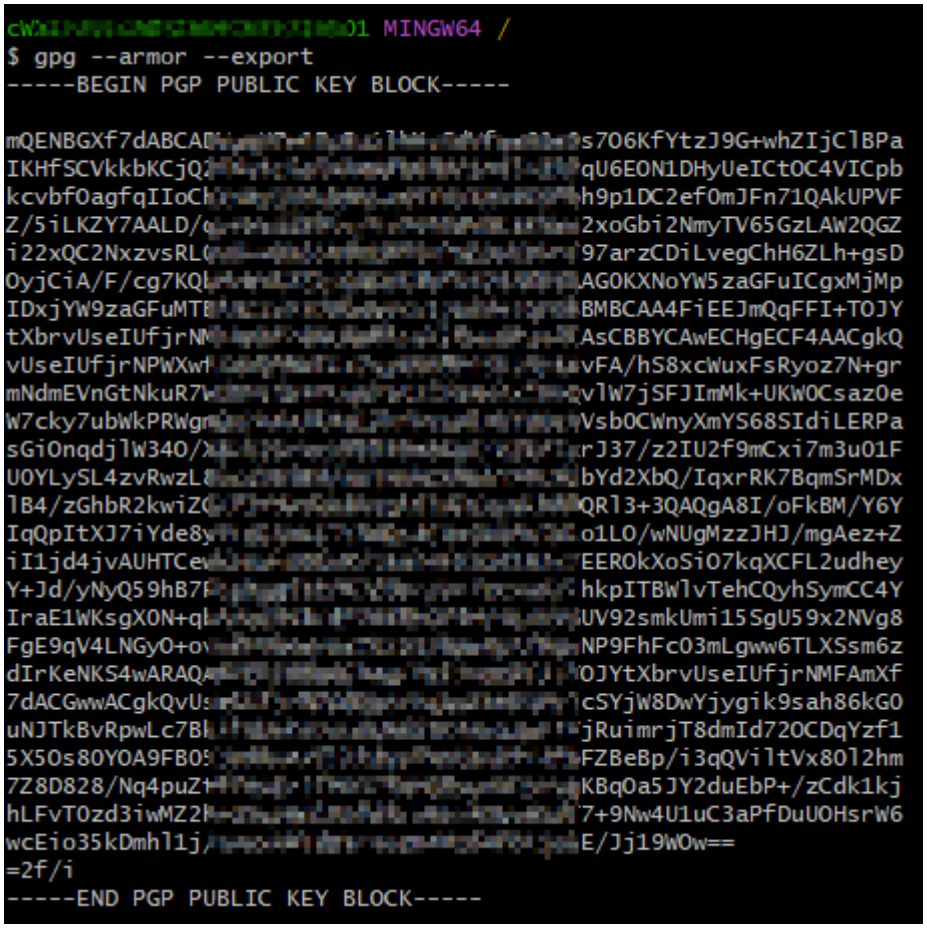
出现图4-2所示的信息，说明GPG密钥生成成功。

图 4-2 GPG 密钥生成成功



步骤4 执行**gpg --armor --export**命令导出公钥，如图4 导出GPG公钥所示，并复制公钥到剪切板，包括“-----BEGIN PGP PUBLIC KEY BLOCK-----”和“-----END PGP PUBLIC KEY BLOCK-----”。

图 4-3 导出 GPG 公钥



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，选择“个人设置 > 代码托管 > GPG公钥”，进入GPG设置页面。

步骤6 单击“新建GPG公钥”按钮，进入新建GPG公钥页面，填写如下参数：

表 4-3 新建 GPG 公钥参数列表

参数	说明
GPG标题	必填参数。自定义GPG公钥名称，限制在200字符以内。
GPG公钥	必填参数。将**步骤4**中复制的GPG公钥粘贴到此输入框。
描述	非必填参数。当描述为空，列表显示--，字符上限为200。

步骤7 单击“确定”，成功新建GPG公钥，返回至GPG公钥列表页面。

说明

- 同一个GPG公钥不能重复使用，如果添加失败，请检查您是否已经添加过此公钥、粘贴时前后是否有多余的空格。
- 添加成功后，可以在“GPG公钥”列表页面查看到您添加的公钥，当您确认不再使用时，可以将其删除。

----结束

5 迁移代码与同步仓库

5.1 迁移代码仓库概述

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

- **迁移第三方Git仓**
- **导入本地Git仓**
- **迁移SNV代码仓**

5.2 迁移第三方 Git 仓到 Repo

5.2.1 使用 URL 导 Git 仓入 Repo

步骤1 进入CodeArts Repo首页后，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“导入仓库”，导入方式选择“Git Url”，参数填写请参考**表5-1**。

表 5-1 “获取授权”参数填写

字段名称	说明
源仓库路径	该参数必填，该参数表示要导入的仓库路径。源仓库路径需要以（http://）或（https://）开头，以（.git）结尾。 **说明** • 如果仓库过大或者网络较差时，仓库导入时间可能会超过30min。如果出现导入超时，建议使用客户端clone/push来处理，具体可参考**导入外部仓库提示超时**。 • 该功能需要保证被导入的仓库域名和服务节点网络连通。

字段名称	说明
源仓库访问权限	必填。分两种情况填写： • 如果您导入的源仓可见范围是对所有访客公开，勾选“不需要校验权限”。 • 如果您导入的源仓可见范围是私仓，请勾选“需要校验权限”。当前支持两种鉴权方式，“通过服务扩展点”和“通过用户名密码授权”，参数填写请参考**校验导仓权限**。

步骤3 单击“下一步”，进入“填写基本信息”页，请参考**表格**填写参数。

步骤4 请参考**表1 同步仓库设置的参数表格**，填写“同步仓库”设置参数。

----结束

说明

- 填写完参数后，会自动跳转到“我创建的”代码仓列表页面，如果新建代码仓库名称颜色为灰色，且仓库名称旁有红色感叹号，表示该仓库导入失败，可能原因：用户名或者密码/Access Token错误。可以将该代码仓删除，按照如上步骤操作，重新导入外部仓库。
- 当前Git支持的外部导入源包括：bitbucket.org、code.aliyun.com、coding.net、git.qcloud.com、gitee.com、github.com、gitlab.com、visualstudio.com、xiaolvyun.baidu.com。
- 在新建代码仓库后，仅有创建者能够访问该仓库。其他项目成员需要手动添加到仓库中，并分配相应的权限。因此，您需要根据需求，手动为代码仓库添加成员并为新增成员配置访问权限。

5.2.2 导入 GitHub 仓

步骤1 进入CodeArts Repo首页后，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“导入仓库”，导入方式选择“Github”。

步骤3 选择授权方式。您可以通过“服务扩展点”授权，参考**服务扩展点授权**，也可以“通过个人访问令牌授权”，参考**获取Access Token**。

步骤4 单击“下一步”，自动跳转到“选择导入仓库”页面，勾选您需要导入的仓库，单击“下一步”，进入“填写基本信息”页面，请参考表格**填写仓库基本信息**为每个需要导入的仓库填写仓库的基本信息，继续参考**表1 同步仓库设置的参数表格**，填写“同步仓库”设置参数。

----结束

服务扩展点授权

表 5-2 服务扩展点授权参数

参数	说明
连接名称	必填，根据自定义填写名称，连接名称最大长度不超过256个字符。
验证方式	必填，根据需要选择： • 如果选择“OAuth认证”，单击“授权并确定”，将自动跳转到GitHub登录页面，输入GitHub登录的账号和密码后，单击“Authorize huaweidevcloud”即可完成授权。授权成功后，新建扩展点页面将出现“授权成功”，页面右上角会弹出“新建接入点成功!”，这时您可以下拉框选择刚才创建成功的扩展点。 • 如果选择“AccessToken认证”，需要使用有仓库管理员权限的账号在GitHub创建的Access Token，可参考**获取Access Token**。

5.3 导入本地 Git 仓

如果您的代码仓还没有纳入过任何的版本系统，如Git或者SVN，在源代码的根目录，执行如下操作，把本地自建的代码仓导入到CodeArts Repo。

步骤1 进入CodeArts Repo首页，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“普通仓库”，填写对应参数信息并去勾选“允许生成README文件”和“选择gitignore”，完成新的代码仓库创建，并自动跳转到该代码仓库首页。

步骤3 执行命令**git init**，在本地新建一个空的Git代码仓库目录。

步骤4 执行命令**git add ***，将文件加入版本库。

步骤5 执行命令**git commit -m "init commit"**，创建初始提交。

----结束

须知

在新建代码仓库后，仅有创建者能够访问该仓库。其他项目成员需要手动添加到仓库中，并分配相应的权限。因此，您需要根据需求，手动为代码仓库添加成员并为新增成员配置访问权限。

说明

如果CodeArts Repo的仓库容量快满的时候，您可以进入代码仓库详情页，使用如下的方法清理代码仓库资源：

- 选择“代码 > 分支”，选择不需要的分支，单击 ，删除不需要的分支。
- 选择“代码 > 标签”，选择不需要的标签，单击 ，删除不需要的标签。
- 选择“设置 > 仓库管理 > 仓库加速”，清除缓存数据。
- 选择“设置 > 仓库管理 > 子模块设置”，删除不需要的子模块。

5.4 迁移 SVN 代码仓

在线导入 SVN 平台的代码仓库到 CodeArts Repo

步骤1 进入CodeArts Repo首页后，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“导入仓库”，导入方式选择“SVN”，参数填写请参考**表5-3**。

表 5-3 导入 SVN 平台代码仓库的参数表格

字段名称	说明
源仓库路径	该参数必填，该参数表示要导入的仓库路径。源仓库路径需要以（ http:// ）开头。 **说明** • 如果仓库过大或者网络较差时，仓库导入时间可能会超过30min。如果出现导入超时，建议使用客户端clone/push来处理，具体可参考**通过Git Bash导入SVN平台的代码仓库到CodeArts Repo**。 • 在线导入的操作方式简单，且将SVN中的分支、Tags进行平移，如果后续想在此代码仓的基础上继续开发，请利用Git Bash客户端导入，具体可参考**通过Git Bash导入SVN平台的代码仓库到CodeArts Repo**。 • 该功能需要保证被导入的仓库域名和服务节点网络连通。
源仓库访问权限	必填。分两种情况填写： • 如果您导入的源仓可见范围是对所有访客公开，勾选“不需要校验权限”。 • 如果您导入的源仓可见范围是私仓，请勾选“需要校验权限”。当前支持两种鉴权方式，“通过服务扩展点”和“通过用户名密码授权”，参数填写请参考**校验导仓权限**。

步骤3 单击“下一步”，进入“填写基本信息”页，请参考表格填写参数。

步骤4 请参考**表1 同步仓库设置的参数表格**，填写“同步仓库”设置参数。

----结束

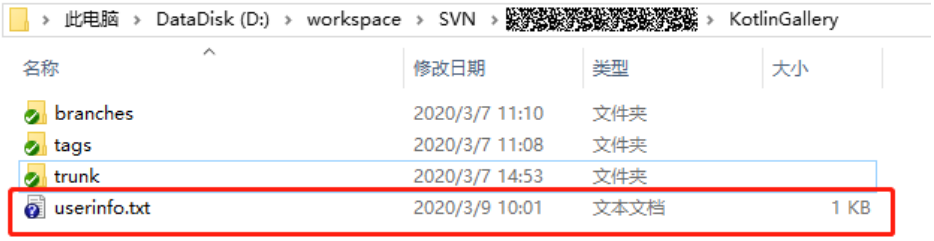
通过 Git Bash 导入 SVN 平台的代码仓库到 CodeArts Repo

步骤1 获取SVN代码库提交者信息。

1. 通过TortoiseSVN将待迁移的代码仓库下载到本地。
2. 进入本地SVN代码仓库（本文为KotlinGallery），在Git Bash客户端执行如下命令：

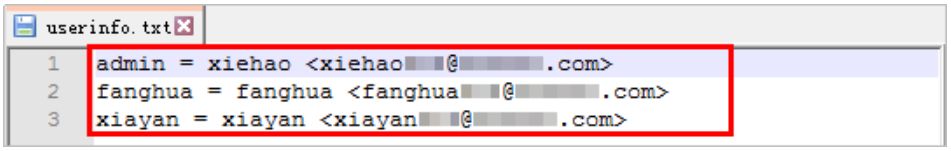
```
svn log --xml | grep "^<author" | sort -u | \awk -F '<author>' '{print $2}' | awk -F '</author>' '{print $1}' > userinfo.txt
```

执行完毕后，“KotlinGallery”目录下将生成文件“userinfo.txt”，如下图所示。



3. 打开文件“userinfo.txt”，可看到文件中显示所有对该仓库有提交操作的提交者信息。
4. 因为Git是用邮箱来标识一个提交者的，为了更好的将SVN已有的信息映射到Git仓库里，需要从SVN用户名到Git作一个映射关系。

   修改“userinfo.txt”，使每一行中，svn作者 = Git作者昵称 <邮箱地址>，映射关系的格式如下图所示。



步骤2 建立本地Git仓库。

1. 执行命令**git init**，在本地新建一个空的Git代码仓库目录。
2. 将**步骤1**中的“userinfo.txt”文件拷贝到该目录下，并执行如下命令切换到该目录下。

```
cd 目标目录地址
```

3. 在该目录下启动Git Bash客户端，并执行如下命令克隆一个Git版本库。

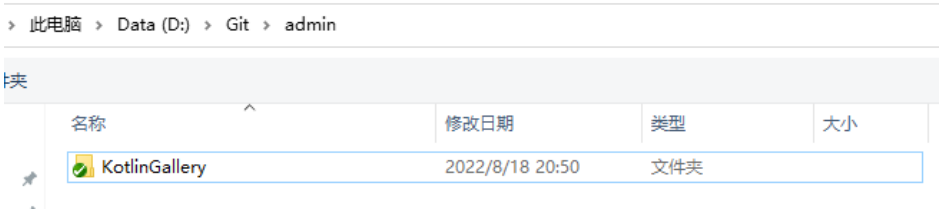
```
git svn clone SVN仓库地址 --no-metadata --authors-file=userinfo.txt --trunk=trunk --tags=tags --branches=branches
```

命令行中的参数说明如下，请根据实际情况选择相应参数:

参数	说明
--no-metadata	表示不将 SVN 的元数据导入到Git仓库中，这样可以减小Git代码仓库的大小，但是可能会丢失一些SVN的历史信息。

参数	说明
--authors-file=userinfo.txt	表示使用指定的用户信息文件来进行作者信息的映射。
--trunk=trunk	表示将SVN仓库中的“trunk”分支作为Git代码仓库的主分支。
--tags=tags	表示将SVN代码仓库中的tags目录作为Git代码仓库的标签。
--branches=branches	表示将SVN代码仓库中的branches目录作为Git代码仓库的分支。

执行成功后，本地将生成一个名为KotlinGallery的Git代码仓库。



4. 执行以下命令，进入“KotlinGallery”文件夹，并验证当前Git仓库分支结构。

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

如上图所示，所有SVN中的目录结构均以Git分支的形式迁移成功。

步骤3 本地分支修正。

因此在上传到代码托管仓库前，需要先对本地分支进行调整，使之符合Git使用规范。

1. 进入本地Git代码仓库目录下，在Git Bash客户端执行如下命令，把Tags分支变成合适的Git标签。

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



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

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

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

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

2. 执行以下命令，把“refs/remotes”下面剩下的索引变成本地分支。

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

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

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

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

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

3. 执行以下命令，将trunk分支合入master分支，并删除trunk分支。

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

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

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

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

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

步骤4 上传本地代码仓到CocdeArts Repo。

1. 参考**配置SSH密钥**，设置代码仓库的SSH密钥。
2. 进入CodeArts Repo首页，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

3. 仓库类型选择“普通仓库”，填写对应参数信息并去勾选“允许生成README文件”和“选择gitignore”，完成新的代码仓库创建，并自动跳转到该代码仓库首页。
4. 选择右上角的“克隆/下载” > “用HTTPS克隆”，复制HTTPS地址。
5. 执行以下命令，将本地代码仓库与CodeArts Repo进行关联，并推送master分支到CodeArts Repo的代码仓库。在执行命令时，需要您输入CoeArts Repo的HTTPS账号和密码。
   ```
   git remote add origin 新建的代码仓库的HTTPS地址
   git push --set-upstream origin master
   ```
   推送成功后，进入该代码仓库首页，选择“代码” > “分支”，查看到当前代码仓库下的master分支。
6. 继续执行以下命令，把本地其余分支推送到CodeArts Repo。
   ```
   git push origin --all
   ```
   推送成功后，进入该代码仓库首页，选择“代码” > “分支”，代码仓库下新增了r1.1_hotfix分支。
7. 执行以下命令，从本地推送tags到CodeArts Repo。
   ```
   git push origin --tags
   ```
   推送成功后，进入该代码仓库首页，选择“代码” > “Tags”，代码仓库下已有标签“r1.0”与“r1.1”。

----结束

5.5 同步仓库设置

同步仓库设置

表 5-4 同步仓库设置的参数表格

字段名称	说明
分支设置	该参数必填。包括两个选项： ● 默认分支。指新建代码仓库时自动创建的主分支，例如master分支。 ● 全部分支。指代码仓库中的所有分支，包括默认分支及其他自定义分支。
增加定时同步	非必填。 **须知** 如果您勾选了此参数，导入的仓库将为镜像仓，仓库将无法提交代码，只能从源仓定时同步，并且是每24小时自动刷新一次代码，刷新内容为源仓库24小时前的内容。

5.6 校验导仓权限

Repo当前支持两种方式的校验权限，可通过服务扩展点校验权限，也可以通过用户名密码授权。

- 通过服务扩展点校验权限

步骤1 连接名称。必填，根据自定义填写名称，连接名称最大长度不超过256个字符。

步骤2 Git仓库Url。必填，输入导入源仓的URL地址。

步骤3 用户名。当源代码仓库为私有时，该参数必填。该参数表示HTTPS克隆代码时的用户名，例如为GitHub的登录名称。

步骤4 密码或Access Token。当源代码仓库为私有时，该参数必填。该参数表示HTTPS克隆代码时的密码或者Access Token，例如为GitHub的登录密码或者在GitHub创建的Access Token。该参数的获取方式请参考**在GitHub获取Access Token**。

----结束

- 通过用户名密码授权

您也可以勾选“通过用户名密码授权”，“用户名”填写请参考**用户名参数填写**，“密码或Access Token”填写请参考**密码或Access Token参数填写**。

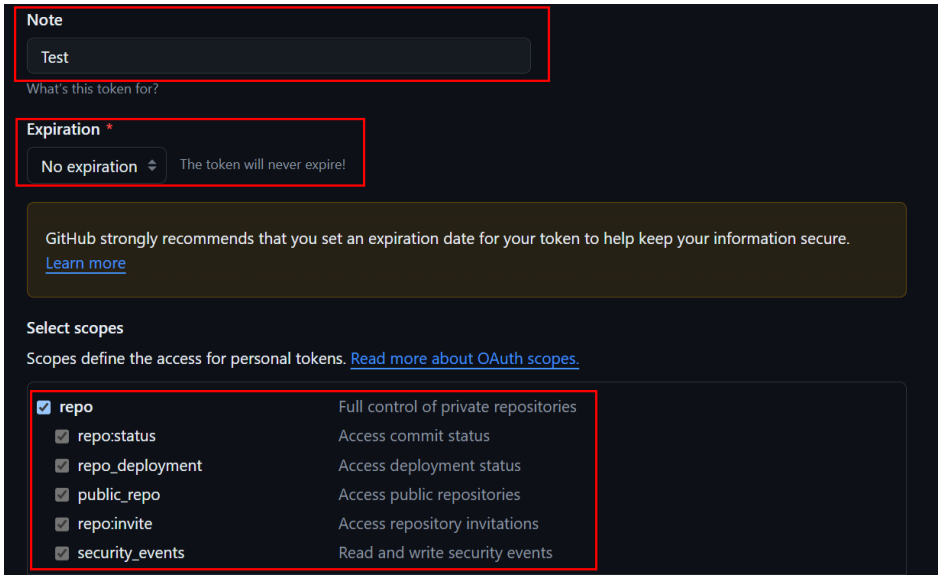
5.7 获取 Access Token

在 GitHub 获取 Access Token

步骤1 **登录GitHub**，单击右上角头像，选择“Settings” > “Developer settings”。

步骤2 选择“Personal access tokens ” > “Personal access tokens (classic)” > “Generate new token (classic)”，填写关键信息，如**下图**所示。

图 5-1 填写“新建 Token”的关键信息



步骤3 填写好关键信息，完成Token的新建，并自动跳转到新建的Token页面，由于该Token是临时生成的，请复制并保存该Token。

----结束

5.8 填写仓库基本信息

表 5-5 填写仓库基本信息

参数	说明
代码组路径	非必填。默认为“/”，表示不归属于任何代码组路径。您也可以下拉框选择已有的代码组路径。
代码仓库名称	必填。请您为导入的仓库命名。需要以大小写字母、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。
描述	非必填。为仓库添加描述信息，最多不能超过2000个字符。
初始化设置	非必选。如果您已开通代码检查(CodeArts Check)服务，推荐您勾选该选项，代码仓库创建完成后，在代码检查(CodeArts Check)任务列表中，可看到对应仓库的检查任务。
可见范围	该参数为非必填。该参数表示源仓库的可见范围，包括两个选项： ● 公开。包含三个选项：“项目内成员只读”、“租户内成员只读”和“所有访客只读”。 ● 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）。

6 新建 Repo 代码仓库

6.1 不同场景下新建代码仓库的区别

CodeArts Repo当前支持五种新建代码仓库的方式，您可以根据自己的使用习惯创建代码仓库。

6.2 新建自定义代码仓库

步骤1 进入CodeArts Repo首页后，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“普通仓库”，根据**表格**填写参数。

表 6-1 新建仓库的参数说明

字段名称	说明
代码仓库名称	该参数为必填。填写该参数时，需要以大小写字母、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。
描述	该参数为非必填。该参数限制2000个字符。
选择gitignore	该参数为非必填。推荐您填写该参数，下拉框选择代码仓库要开发的编程语言，可以避免后续不必要的文件被跟踪，以此保持代码库的整洁和可维护性。

字段名称	说明
初始化设置	该参数为非必填。包括两个选项： • 允许生成README文件。推荐您勾选该选项，生成该文件后，您可以通过编辑README文件，记录项目的架构、编写目的等信息，帮助其他人更快了解该代码仓。 • 自动创建代码检查任务（免费）。推荐您勾选该选项，代码仓库创建完成后，在代码检查(CodeArts Check)任务列表中，可看到对应仓库的检查任务。
可见范围	该参数为非必填。您可根据自己的需求进行选择，包括两个选项： • 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。包含三个选项：“项目内成员只读”、“租户内成员只读”和“所有访客只读”。 **说明** 代码仓库可以相互转换“私有”或者“公开”，进入要设置的代码仓库详情页面，选择“设置 > 基本设置 > 仓库信息”，修改代码仓库的可见范围。
添加开源许可证	该参数在“可见范围”选择为“公开只读”时必填。下拉框选择已有的许可证。

----结束

说明

在新建代码仓库后，仅有创建者能够访问该仓库。其他项目成员需要手动添加到仓库中，并分配相应的权限。因此，您需要根据需求，手动为代码仓库添加成员并为新增成员配置访问权限。

6.3 按模板新建代码仓库

步骤1 进入CodeArts Repo首页后，单击“新建仓库”，在“归属项目”下拉框中选择已有的项目或者“新建项目”。

步骤2 仓库类型选择“模板仓库”，这里可选择“官方模板”或“个人模板”。其中，个人模板需要进入仓库设置，将官方模板设置为个人模板。选择模板后，根据表格填写参数。

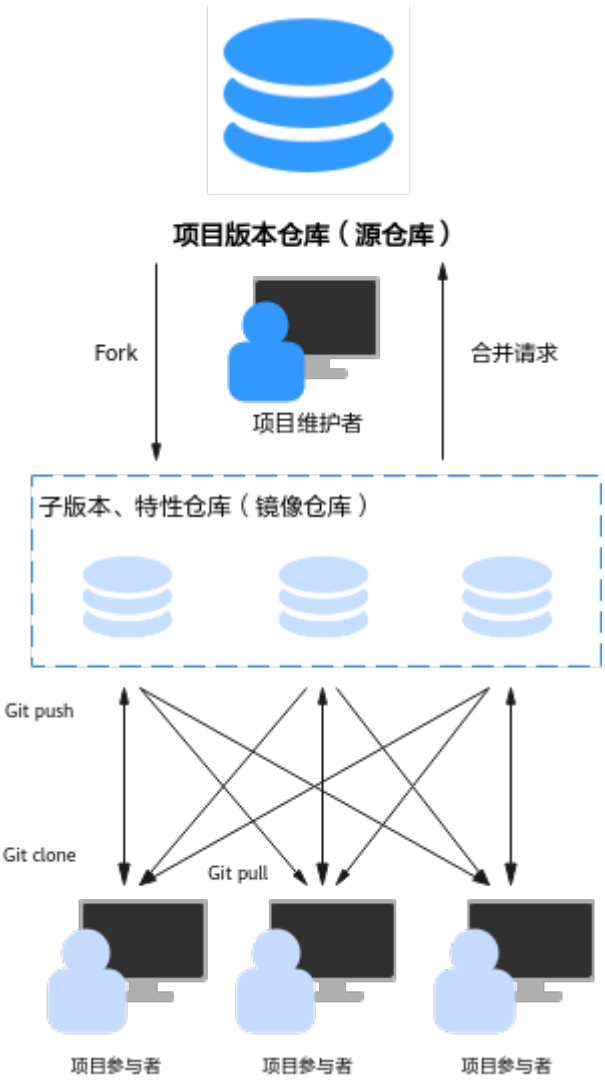
----结束

6.4 Fork 仓库

Fork 仓库的应用场景

Fork仓适用于大型项目，其中包含众多子项目时的开发场景。Fork基于某个仓库，可以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。在合并未发生时，镜像仓和源仓库的修改都不会对彼此产生影响。

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。因此，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都是在复制仓库，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**

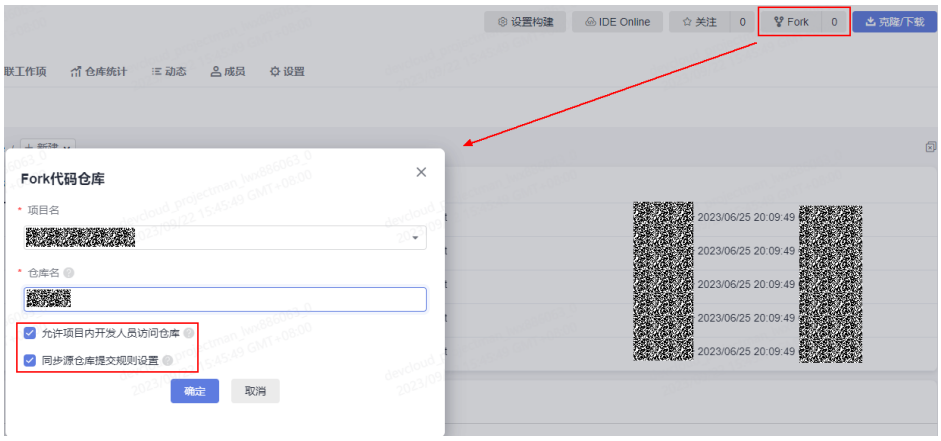
  - Fork仅应用于代码托管平台内的仓库间复制。
  - Fork仓库时，会基于源仓库的当前版本复制出一个内容相同的副本仓库，您在副本仓库的修改，可以申请合并（可以理解为一种跨仓库的分支合并）回源仓库，但副本仓库不能再获取源仓库的更新。
- **导入外部仓库**
  - 导入外部仓库不仅可以将其它版本管理平台的仓库进行导入（主要针对基于Git、SVN存储的托管平台），也可以导入代码托管服务自己的仓库。
  - 导入外部仓库时，也会基于源仓库的当前版本复制出一个内容相同的副本仓库，所不同的是，副本仓库不能向源仓库提交合并申请，但是副本仓库可以随时拉取源仓库的默认分支，以起到获取最新版本的作用。

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。

7 配置 CodeArts Repo 代码仓库设置

配置代码仓库级的仓库设置

如果在项目级“仓库设置”勾选了“开启强制继承”，代码仓库下不支持“仓库设置”。

如果不继承项目级配置，可参考下表设置参数。

表 7-1 代码仓库级的仓库设置参数填写表格

参数	说明
默认分支管理	此参数非必填。默认将“master”分支设置为默认分支，即创建代码仓库时的主分支。
开启开发人员创建分支权限白名单	此参数非必填。默认不勾选，勾选后，开启开发人员创建分支权限白名单，只有开发人员角色的仓库成员才能进入此白名单。非开发人员将不会被显示，并且即使配置后也不会生效。
禁止Fork仓	非必填参数。勾选此选项，表示任何人不可以Fork该项目下的代码仓库。
MR预合并	非必填参数。勾选此选项，表示启用MR预合，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。

参数	说明
分支名规则	非必填参数。所有分支名都必须匹配正则表达式，分支名规则不能超过500个字符。如果此字段不填写，则允许任何分支名。规则需要满足基本的Tag命名规则： ● 不能超过500个字符。 ● 不支持以 - . refs/heads/ refs/ remotes/ 开头，不支持空格 [\ < ~ ^ : ? * ! ( ) ' " | 等特殊字符，不支持以. / .lock结尾。
Tag名规则	非必填参数。所有Tag名都必须匹配正则表达式。如果此字段不填写，则允许任何Tag名。 需满足基本的Tag命名规则： ● 不能超过500个字符。 ● 不支持以 - . refs/heads/ refs/ remotes/ 开头，不支持空格 [\ < ~ ^ : ? * ! ( ) ' " | 等特殊字符，不支持以. / .lock结尾。

配置代码仓库级的保护分支规则

如果勾选“继承项目设置”，代码仓库下不支持再次“新建保护分支”，代码仓库下的成员均可执行该操作。

如果不继承项目级配置，可参考下表设置参数。

表 7-2 新建保护分支的参数表格

参数名称	参数解释
选择需要添加的保护分支	根据自己的需要输入完整的分支名或者带通配符的分支名。 仅支持单个添加，不支持批量添加。要求以“refs/heads/”开头，结尾可以有“*”，其它位置不可以出现特殊字符。
添加权限	该参数非必填。支持对管理员/项目经理、Committer和开发人员添加如下权限： ● 推送。拥有该权限，可推送Commit到保护分支。 ● 合并。拥有该权限，可以对该保护分支合入合并请求。 **说明** 如果打开推送权限，默认同步打开合并权限，且不可单独关闭合并权限。

配置代码仓库级的保护 Tag 规则

进入要设置设置的代码仓库首页，，选择“设置 > 策略设置 > 保护Tags”，单击“新建保护Tag”，参考下表填写配置参数。

表 7-3 新建保护 Tag 参数说明

参数	说明
选择需要保护的Tag	该参数必填。根据自己的需要输入完整的Tag或者带通配符的Tag。 仅支持单个添加，不支持批量添加。要求以“refs/heads/”开头，结尾可以有“*”，其它位置不可以出现特殊字符。
允许创建	该参数必填。表示添加“允许创建保护Tag”的角色。您可以在下拉框选择允许创建的角色。

配置代码仓库级的提交规则

CodeArts Repo支持为代码的提交建立校验、限制规则，以确保代码质量，您可以勾选“继承项目设置”，自动继承并使用项目下设置且不支持更改。您也可以进入要配置的代码仓库首页，选择“设置” > “策略设置” > “提交规则”，单击“新建提交规则”，参数填写请参见表格表14-2。

配置代码仓库的子模块设置

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
```

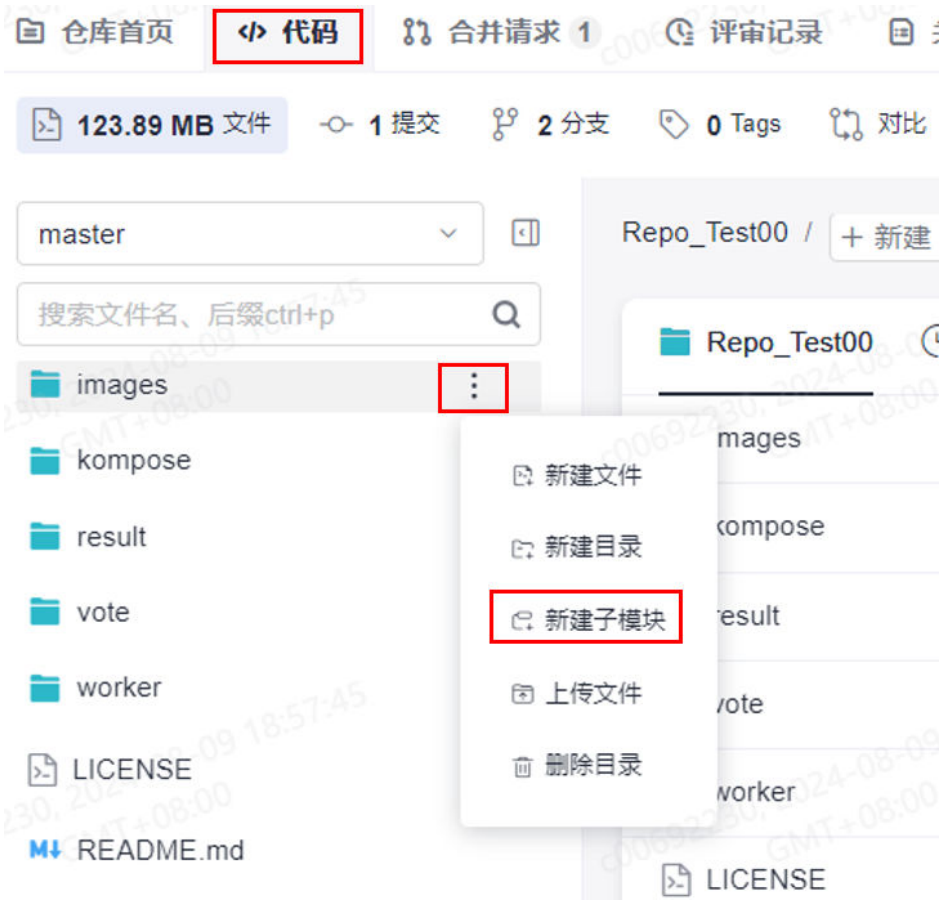
这时，位于“file_path”目录下的源代码，将会来自“repo_url”。

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。

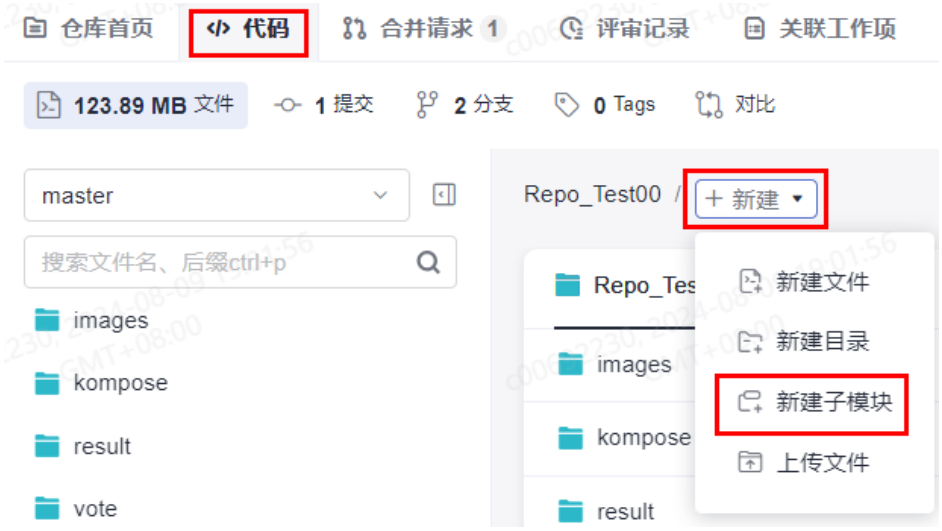
    单击扩展按钮 ，选择“新建子模块”，如下图所示。



- **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 7-4 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

更多详情请参见官方文档**Git 工具 - 子模块**。

8 分层管理代码仓

8.1 新建代码组

代码组概述

代码组是由一个或多个仓库组成的群体。您可以为代码组下的仓库或子代码组进行统一的仓库规则配置管理操作，包含提交规则、成员权限配置等。

说明

最多可新建5层代码组。

新建代码组

进入**项目或父组织中**，单击 

图标下拉框选择“**新建代码组**”，进入新建代码组页面，根据下表填写基本信息，单击“**确定**”，完成代码组的新建，代码组最多支持三层目录。

表 8-1 新建代码组参数说明

字段说明	是否必填	备注说明
归属项目	是	• 代码组必须存在项目下。 • 如果账号下没有项目请在项目选择框中选择“新建项目”会先弹出“新建项目”页面，这时建立的项目是Scrum。 **说明** 只有通过“代码托管”首页入口新建代码组时，才能新建项目。
代码组路径	否	代码组路径对应建仓接口的参数groupId。如果groupId为空，则表示创建项目下的仓库，没有对应的代码组。您可根据自己实际需求选择代码组路径。代码组路径范围是所有首层代码组、子代码组的根组织路径。

字段说明	是否必填	备注说明
代码组名称	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。代码组和仓库总长度限制为256字符。
描述	否	为您的代码组填写描述，限制2000字符。
是否公开	是	可选择私有和公开只读，默认选择私有。 • 私有 仅对代码组成员可见。私有代码组下的子代码组和仓库的公开性只能为“私有”。 • 公开只读 代码组对所有访客公开可读，但不出现在访客的代码组列表及搜索中。公开代码组下的子代码组和仓库的公开性支持私有和公开只读两种。

8.2 使用代码组

8.2.1 查看代码组列表

您可以通过以下方式进入代码托管服务代码组列表页。

进入软件开发生产线首页，单击“**服务**”图标下的“**代码托管**”，默认展示我参与的仓库列表页，单击“代码组”下的任一菜单，即可进入代码托管服务代码组列表页。

在这里您可以完成新建代码组、配置代码组等操作。

- ＋ 新建代码组：单击该图标，可进入新建代码组页面。
- ☆：单击该图标，关注代码组。可在我关注的代码组中查看该代码组。
- ：单击代码组所在行右侧的该图标，可进入子代码组首页。
- ：单击父代码组后的该图标，可展示“仓库”、“成员”、“设置”和“新建子代码组”图标。
  - ：单击该图标，可直接进入代码仓（组）列表页面。
  - ：单击该图标，可直接进入代码组成员列表页面。
  - ：单击该图标，可直接进入代码组“设置”页签下 的代码组信息页面。
  - ：单击该图标，可直接进入新建子代码组页面。

个人首页：支持查看“**我关注的**”、“**我参与的**”及“**我创建的**”等分类的代码组。右上角支持查看“**最近创建**”和“**最近更新**”的代码组。

8.2.2 查看代码组详情

在代码组列表中单击代码组名称可进入该代码组的详情页面，代码托管服务提供了丰富的控制台操作，详情如下。

表 8-2 页签说明

功能说明	页签说明
代码仓（组）	用于展示代码组的数量、仓库数量、开启中的MR的数量和成员数量等信息。同时您也可以新建仓库和查看未锁定的仓库。
成员	代码组成员管理页面，支持添加成员，调整代码组成员角色。
设置	此代码组的设置入口，代码组所有成员均可查看，但是仅支持项目管理员或代码组所有者修改。

另外代码组详情页框架上还提供以下功能的快捷入口：

获取仓库地址：可通过单击“SSH”或“HTTPS”后的图标，获取仓库地址。

☆：单击该图标，可关注该代码组。

···：在代码组下单击该图标，可查看仓库数目，并进入每个仓库，查看代码组成员并进行代码组成员设置，创建新的子代码组或仓库，按照模板创建新的仓库，以及导入外部仓库。在仓库下单击该图标，可进行关联工作项、成员管理和删除仓库操作。

8.2.3 查看代码组首页

代码组首页用于展示代码组的基础情况。

表 8-3 字段说明

字段	说明
子代码组	统计代码组数量。
仓库	统计仓库数量。
开启中的MR	统计开启中MR数量。
成员	统计代码组中成员数量，单击图标支持跳转至“成员”页签，进行成员管理。
新建仓库	单击图标支持进入“新建仓库”页面，新建仓库。
所有仓库	所有仓库，支持统计锁定仓库和未锁定仓库。

8.2.4 代码组成员管理

代码托管服务支持在代码组中**添加成员**或**成员组**。

- **“成员列表”**、**“成员组列表”****“待审核”**和**“添加成员”**位于代码组详情的“**成员**”页签下。
  - **“成员列表”**用于展示代码组中所有成员“**用户名**”、**“用户来源”**、“**项目角色**”、“**代码组角色**”和“操作”。
  - “**成员组列表**”用于展示代码组中所有成员组的**“成员组名称”**、**“成员组数量”**、**“描述”**和**“操作”**。
  - “**待审核**”用于展示即将加入代码组中**待审核**成员，包括**“用户名”**、**“项目角色”**、**“代码组角色”**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**”用于代码组**添加成员**或**添加成员组**。

说明

父代码组下的成员无条件继承到子代码组或子仓库，且不允许被删除。

当项目角色发生变化时，如果项目角色和仓库角色一致，则仓库角色同步更新。关于代码组继承成员或成员组添加的成员角色优先级，以最近一次更新为准。

仓库所有者在本仓库中作为管理员角色，享有仓库所有权限，且不可被移除和编辑。

项目管理员为项目下最高权限成员，将同步加入仓库并作为管理员角色，享有仓库所有权限，且不可被移除或编辑。

代码组创建者享有本代码组以及子代码组/仓库的最高权限，且不可被移除和编辑。

该成员通过成员组添加如需删除该成员，请前往所在成员组进行删除操作。

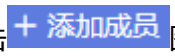
该成员继承于上层代码组，如需删除该成员，请在上层代码组删除即可。

在代码组中添加成员或成员组

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 找到代码组父组织，进入代码组首页。

步骤4 单击菜单“**成员**”，单击  图标，弹出“**添加成员**”弹框。

步骤5 在“**添加成员**”弹框，单击菜单“**成员**”，您可搜索需添加的成员，选择成员后，单击“**确定**”按钮，添加当前页面的成员进入仓库。

步骤6 在“**添加成员**”弹框，单击菜单“**成员组**”，在下拉框中选择需添加的成员组，单击“**确定**”按钮，添加当前页面的成员组进入仓库。

----结束

8.3 配置代码组

8.3.1 代码组信息

代码组信息可在代码组详情的“**设置 > 基本设置 > 代码组信息**”查看和修改。

此设置只针对被设置的代码组生效。

代码组下所有成员都能查看这个页面，项目管理员和代码组创建者能看到这个页面且有设置权限。

代码组名称默认不可修改。

代码组描述用于描述代码组相关信息。

8.3.2 仓库设置

仓库设置位于代码组详情中的“**设置 > 仓库管理 > 仓库设置**”。

默认分支会作为进入本代码组时，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。代码组新建时，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代码组生效。

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。设置完成后单击**“提交”**即可生效。

表 8-4 参数说明

参数项	说明
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。
分支名规则	所有分支名都必须匹配正则表达式。如果此字段为空，则允许任何分支名。需满足基本的分支命名规则，限制500个字符。示例：^feature-[0-9a-zA-Z]+ • 至多500个字符。 • 创建分支不支持以“-”、“**refs/heads/**”、“**refs/remotes/**”开头，不支持空格[\<~^：？*! （ ）' ‘ ’ | 等特殊字符，不支持以“**./**”或“**.lock**”结尾。 • 新建的分支不可以和原有的分支/tag名重复。

参数项	说明
Tag名规则	所有Tag名都必须匹配正则表达式。如果此字段为空，则允许任何Tag名。需满足基本的Tag命名规则，限制500个字符。示例：^TAG*$ ● 不能超过500个字符。 ● 创建tag不支持以“-”、“**refs/heads/**”、“**refs/remotes/**”开头，不支持空格[\<~^：？*！（）'‘’|等特殊字符，不支持以“**./**”或“**.lock**”结尾。 ● 新建的tag不可以和原有的分支/tag名重复。

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。

8.3.3 风险操作

风险操作位于代码组详情中的“**设置 > 风险操作**”。

代码组所有成员均可查看，但是仅支持项目管理员或代码组所有者修改。

目前有如下操作：

- **删除代码组**：删除代码组将导致所有子代码组和资源被删除。删除的代码组无法复原。您只能删除一次，并且无法恢复，请再三确认！
- **更改代码组名称**：将同步修改代码组路径和仓库路径，修改后原路径不可用，请谨慎操作！更改代码组名称可能会引起超出预期的情况。

  说明

  - 更改代码组名称会影响仓库克隆地址，需检查和更新相关配置，否则影响使用，请谨慎操作。
  - 如果代码组下仓库配置了相关流水线，修改代码组名称后，流水线将无法触发，需同步更新流水线相关配置（执行计划和流水线源），具体可参考的“配置流水线”章节。
  - 更改代码组名称后，代码构建、代码检查、部署、CodeArts IDE等服务也需要排查和修改相关配置。

8.3.4 权限管理

代码组的**权限管理**位于代码组详情中“**设置**”页签下。

您可根据下表给各角色配置权限。

说明

代码组权限矩阵仅支持项目管理员及各层代码组的所有者修改。

如果该仓库成员是从代码组下继承的，那么其角色默认为代码组角色，在仓库中修改该仓库成员的角色后，单击“成员列表”页签下仓库成员所在行对应操作列的 按钮时，则该角色权限会改为之前代码组角色。

表 8-5 代码组角色权限

角色/功能	操作权限	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、运维经理和产品经理	浏览者	自定义角色
代码组	新建	B	B	B	B	C	D	C
	删除	B	D	D	D	D	D	C
	设置	B	D	D	D	D	D	C

角色/功能	操作权限	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、运维经理和产品经理	浏览者	自定义角色
仓库	新建	B	B	B	B	C	D	C
	Fork	B	B	B	B	C	D	C
	删除	B	D	D	D	D	D	C
	设置	B	D	D	D	D	D	C
代码	提交	B	A	A	A	C	D	C
	下载	B	A	A	A	C	D	C
成员	添加	B	D	D	D	D	D	C
	修改	B	D	D	D	D	D	C
	删除	B	D	D	D	D	D	C
分支	新建	B	B	B	B	C	D	C
	删除	B	B	B	B	C	D	C
Tag	新建	B	B	B	B	C	D	C
	删除	B	C	C	C	C	D	C
MR	新建	B	B	B	B	C	D	C
	编辑	B	B	C	C	D	D	C
	评论	B	B	B	B	C	C	C
	检视	B	B	B	B	D	C	C
	审核	B	B	C	C	D	D	C
	合并	B	B	C	C	D	D	C
	关闭	B	B	C	C	D	D	C
	重开	B	B	C	C	D	D	C

说明

- A：表示该角色默认拥有该权限且不可被移除。
- B：表示该角色默认拥有该权限且可被移除。
- C：表示该角色可分配到该权限。
- D：表示该角色不可分配到该权限。

仓库级权限管理位于仓库详情中“设置”页签下。

您可根据下表给各角色配置权限。

表 8-6 仓库级角色权限

角色/功能	操作权限	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、运维经理和产品经理	浏览者	自定义角色
仓库	Fork	B	B	B	B	C	D	C
	删除	B	D	D	D	D	D	C
	设置	B	D	D	D	D	D	C
代码	提交	B	A	A	A	C	D	C
	下载	B	A	A	A	C	D	C
成员	添加	B	D	D	D	D	D	C
	修改	B	D	D	D	D	D	C
	删除	B	D	D	D	D	D	C
分支	新建	B	B	B	B	C	D	C
	删除	B	B	B	B	C	D	C
Tag	新建	B	B	B	B	C	D	C
	删除	B	C	C	C	C	D	C
MR	新建	B	B	B	B	C	D	C
	编辑	B	B	C	C	D	D	C
	评论	B	B	B	B	C	C	C
	检视	B	B	B	B	D	C	C
	审核	B	B	C	C	D	D	C
	合并	B	B	C	C	D	D	C

角色/功能	操作权限	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、运维经理和产品经理	浏览者	自定义角色
	关闭	B	B	C	C	D	D	C
	重开	B	B	C	C	D	D	C

说明

- A：表示该角色默认拥有该权限且不可被移除。
- B：表示该角色默认拥有该权限且可被移除。
- C：表示该角色可分配到该权限。
- D：表示该角色不可分配到该权限。

9 设置仓库

配置代码仓库间的同步设置

CodeArts Repo支持将当前仓库设置自定义同步至其他仓库，当前功能仅支持跨项目同步，暂时不支持跨区域同步。

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。

如果您已开启继承设置后，无法使用同步设置。

仅有仓库的“设置”权限成员可以执行此操作，仓库内的仓库成员可以查看该页面。

进入要设置的代码仓库首页，选择 “设置 > 仓库管理 > 同步设置”。单击“添加仓库”，在弹框中选择目标仓库。

说明

- 同步仓库需保证网络连通。
  - 对于公开平台，CodeArts Repo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。
- 常见的同步失败原因：
  1. **“提交规则”**同步失败：一般是因为源仓库没有设置提交规则。
  2. **“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。

配置 Webhook 设置

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。

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。

表 9-1 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**需符合正则表达式。 – 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

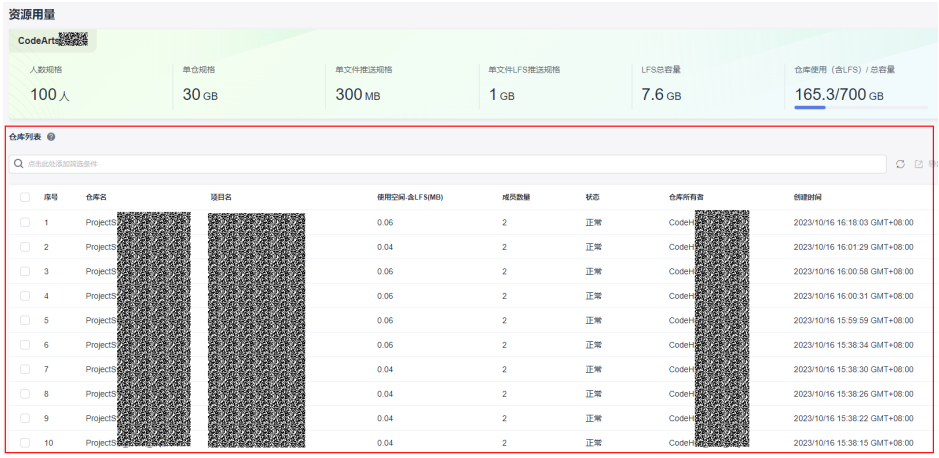
- 每个仓库最多只能设置20个Webhook。
- 您在配置Webhook的时候，还可以选择设置您的Token，该Token会与您的Webhook URL关联，系统会将该Token放在请求头的X-Repo-Token字段发送给您。

9.1 查看仓库列表

CodeArts Repo支持从“我关注的”、“我参与的”和“我创建的”三个维度去查看仓库列表。您可以通过以下三种方式进入代码托管服务的仓库列表。

- 进入软件开发生产线首页，单击 服务 图标下的“代码托管”，进入代码托管服务的仓库列表页，此时界面展示为该租户下所有的代码仓库。
- 进入软件开发生产线首页，进入要查看的项目，选择“代码 > 代码托管”，进入代码托管服务的仓库列表页，此时界面展示为该项目下所有的代码仓库。
- 如果您已开通新版本CodeArts套餐或者已购买CodeArts Repo任一套餐，进入软件开发生产线首页，单击“头像”，选择“租户设置 > 代码托管 > 资源用量”，进入资源用量页面，在“仓库列表”下，单击要查看的项目，跳转到项目对应的仓库列表页，如下图所示。您也可以在CodeArts Repo首页，单击左上角的“租户存储空间”和“租户仓库数”，直接跳转到“资源用量”页面。

图 9-1 资源用量



选择“仓库 > 我参与的”，可查看您参与的所有代码仓库，如果您在某个仓库所在行勾选 ☆，表示您将关注此仓库，可以在“仓库 > 我关注的”，查看您关注的仓库。如果您要查看您创建的仓库，请选择“仓库 > 我创建的”。

9.2 查看仓库详情

在仓库列表中单击要查看的仓库名称，即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 9-2 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。

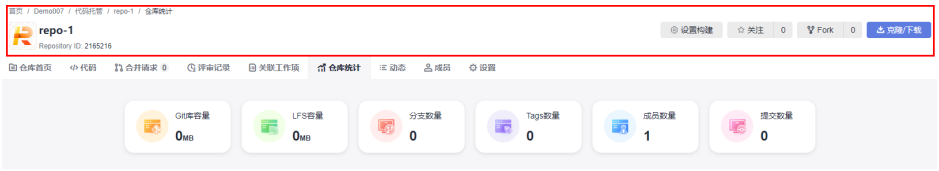
功能页签	功能说明
代码	• 文件列表：支持新建文件、新建目录、新建子模块、上传文件、在线修改文件、修改追溯和查看提交历史等操作。 • 提交：支持查看提交记录及仓库网络图。 • 分支：支持在控制台管理分支。 • Tags：支持在控制台管理标签。 • 对比：支持通过对比查看分支之间或标签版本之间发生的代码变化。
合并请求	支持在控制台管理分支的合并请求。
评审记录	支持查看合并请求的评审记录与Commit的评审记录。
关联工作项	所关联工作项的列表，其可设置与需求管理中工作项的联动，提升效率。
仓库统计	仓库提交记录的可视化图表，主要展现了代码贡献度等信息。
动态	支持查看仓库动态信息。
成员	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。
- IDE Online：可使用IDE Online打开代码（目前免费体验，仅支持北京一、北京四及大连局点）。
- 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

    仓库页面上方只显示“设置构建”按钮。
  - 当用户有构建权限

    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”。

9.3 查看仓库首页

“仓库首页”页签用于展示仓库的基本信息，如下图所示，可根据表格。

表 9-3 字段说明

字段	说明
仓库容量	截止当前仓库的容量。图中表示当前仓库已使用0.04M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	统计仓库默认分支的提交数量，单击数字或者图标，可跳转到“代码”页签下的“提交”，查看提交详情。此示例表示有两次提交信息。

字段	说明
仓库分支	统计仓库的分支数量，单击数字或者图标，可跳转到“**代码**”页签下的“**分支**”，进行分支管理。
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。

字段	说明
Readme	显示仓库的Readme文件，单击文件名称可跳转到**“代码”**页签下查看文件内容。
语言	显示仓库各语言的占比，统计单位为文件大小。

9.4 备份仓库

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：**将仓库备份到华为云的其它区域。
  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。

10 管理 Repo 成员权限

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 10-1 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）
产品经理	产品经理（默认）
测试经理	测试经理（默认）
运维经理	运维经理（默认）
系统工程师	系统工程师（默认）
Committer	Committer（默认）
开发人员	开发人员（默认）

项目中的角色	仓库中的角色
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）
自定义角色	自定义角色（默认）

其中，项目产生的费用将计算在项目经理所属的租户下。

项目创建者在项目中默认为“项目管理员”，在Repo对应为“管理员（仓库所有者）”。

10.2 配置项目级的 Repo 权限

步骤1 登录CodeArts Repo首页，并在左侧导航栏，选择“设置” > “通用设置” > “权限管理”，进入设置权限的页面。

步骤2 选择对应的“角色” > “代码托管”， 单击“编辑”，可设置角色的权限。

说明

1. 项目经理和其他具有管理权限的用户，可以在该页面修改不同角色在项目下的默认操作权限。
2. 可在“角色”列单击 + 创建角色，新增的角色名称不能与系统角色名称重复，但新增角色可复制已有角色的权限。新增角色如果没有复制已有角色的权限，没有任何权限，但是可根据需要添加自定义角色的权限，如表1所示。

----结束

表 10-2 设置项目级角色权限

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
分支	新建	B	C	C	C	B	B	B	C	C	D	C
	删除	B	C	C	C	B	B	B	C	C	D	C
代码	提交	B	C	C	C	A	A	A	C	C	D	C
	下载	B	C	C	C	A	A	A	C	C	D	C

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
代码组	新建	B	C	C	C	B	B	B	C	C	D	C
	删除	B	D	D	D	D	D	D	D	D	D	C
	设置	B	D	D	D	D	D	D	D	D	D	C
成员	添加	B	D	D	D	D	D	D	D	D	D	C
	修改	B	D	D	D	D	D	D	D	D	D	C
	删除	B	D	D	D	D	D	D	D	D	D	C
MR	新建	B	C	C	C	B	B	B	C	C	D	C
	编辑	B	D	D	D	C	B	C	D	D	D	C
	评论	B	C	C	C	B	B	B	C	C	C	C
	检视	B	D	D	D	B	B	B	D	D	C	C
	审核	B	D	D	D	C	B	C	D	D	D	C
	合并	B	D	D	D	C	B	C	D	D	D	C
	关闭	B	D	D	D	C	B	C	D	D	D	C
	重开	B	D	D	D	C	B	C	D	D	D	C
仓库	新建	B	C	C	C	B	B	B	C	C	D	C
	fork(MR)	B	C	C	C	B	B	B	C	C	D	C

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
	删除	B	D	D	D	D	D	D	D	D	D	C
	设置	B	D	D	D	D	D	D	D	D	D	C
Tag	新建	B	C	C	C	B	B	B	C	C	D	C
	删除	B	C	C	C	C	C	C	C	C	D	C

说明

- A：表示该角色默认拥有该权限且不可被移除。
- B：表示该角色默认拥有该权限且可被移除。
- C：表示该角色可分配到该权限。
- D：表示该角色不可分配到该权限。

10.3 配置代码仓库级的权限

仓库权限矩阵仅支持管理员修改，项目管理员及各层父级代码组和仓库所有者可作为管理员。在确认您是管理员的前提下，进入代码托管首页，单击要设置的代码仓名称，进入代码仓的详情页，单击导航栏的“成员”，可为代码仓添加成员。完成代码仓的成员配置，单击导航栏的“设置”，进入仓库设置页面，选择“安全管理” > “权限管理”，若开启“使用项目级权限配置”，当前角色列表成员的权限将与项目权限保持一致，且会覆盖当前的权限配置。

单击右侧的  ，可同步项目自定义角色，自定义角色默认没有仓库的操作的权限，同步后，可根据需要添加**表10-3**所示的权限。

表 10-3 配置代码仓角色权限

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
仓库	fork	B	C	B	C	B	B	B	C	C	D	C
	删除	B	D	D	D	D	D	D	D	D	D	C

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
	设置	B	D	D	D	D	D	D	D	D	D	C
代码	提交	B	C	C	C	A	A	A	C	C	D	C
	下载	B	C	C	C	A	A	A	C	C	D	C
成员	添加	B	D	D	D	D	D	D	D	D	D	C
	修改	B	D	D	D	D	D	D	D	D	D	C
	删除	B	D	D	D	D	D	D	D	D	D	C
分支	新建	B	C	C	C	B	B	B	C	C	D	C
	删除	B	C	C	C	B	B	B	C	C	D	C
Tag	新建	B	C	C	C	B	B	B	C	C	D	C
	删除	B	C	C	C	C	C	C	C	C	D	C
MR	新建	B	C	C	C	B	B	B	C	C	D	C
	编辑	B	D	D	D	C	B	C	D	D	D	C
	评论	B	C	C	C	B	B	B	C	C	C	C
	检视	B	D	D	D	B	B	B	D	D	C	C
	审核	B	D	D	D	C	B	C	D	D	D	C
	合并	B	D	D	D	C	B	C	D	D	D	C
	关闭	B	D	D	D	C	B	C	D	D	D	C

角色/权限	操作权限	项目经理	产品经理	测试经理	运维经理	系统工程师	Committer	开发人员	测试人员	参与者	浏览者	自定义角色
	重开	B	D	D	D	C	B	C	D	D	D	C

说明

- A：表示该角色默认拥有该权限且不可被移除。B：表示该角色默认拥有该权限且可被移除。C：表示该角色可分配到该权限。D：表示该角色不可分配到该权限。
- 关于公开仓库的权限矩阵，默认添加下载权限与评论权限，且不可编辑，其他权限与私仓默认权限一致。

10.4 同步项目成员到代码托管

Repo支持将项目成员同步到代码组和代码仓，帮助您更好管理项目和代码托管，支持自动同步和手动同步，选择其中一种方式即可。

添加Repo代码组及仓库成员，需要确保该成员已加入项目，项目成员管理请参考**项目级成员管理**。

仓库所有者，仓库管理员，以及有成员权限的自定义角色能对仓库人员进行变更，其他人员只能浏览仓库成员列表。

自动同步项目成员到代码组或仓库

Repo支持一键同步项目成员，开启后可自动同步所选角色项目成员至本项目下所有代码组及仓库。

进入要设置的项目首页，左侧导航栏选择“设置 > 代码托管设置”，进入“仓库设置”页面，选择“安全设置 > 成员同步”，勾选需要同步的角色，打开“同步项目成员”后，将自动同步所选角色项目成员至代码组及仓库，项目经理不依赖开关始终同步，可单击刷新按钮触发一次全量同步。

须知

打开“同步项目成员”开关，更新项目成员时才会触发自动同步。

手动添加项目成员到代码组或仓库

进入要设置的代码组或者代码仓库首页，选择“成员”，单击“添加成员”，弹出添加成员页面，两种方式：

- 在“成员”页签，输入关键字，按Enter键可以搜索成员。
- 在“成员组”页签，下拉框选择成员组。

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，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。

11 克隆/下载代码仓库到本地

11.1 克隆代码仓和下载代码仓的区别

克隆代码仓和下载代码仓都是获取代码仓库的方式，但是它们的具体操作和效果有所不同。

1. 克隆代码仓库到本地

   使用SSH密钥或者HTTPS协议克隆代码仓，是将整个代码仓库的内容复制到本地计算机上，并创建一个本地仓库，这个本地仓库包含了完整的代码提交历史记录、分支（Branches）、标签（Tags），可以进行版本控制和修改。当前Repo支持使用Git Bash 和TortoiseGit客户端克隆代码仓，在通过SSH密钥克隆Repo的代码仓库之前，需要**配置访问Repo的SSH密钥**。

2. 下载代码仓库

   下载代码仓则是将代码仓库中的某个或某些文件或文件夹下载到本地计算机上，并不包含完整的代码提交历史记录、分支（Branches）、标签（Tags），不能进行版本控制和修改。当前Repo支持通过浏览器下载代码。

因此，如果需要对代码仓库进行版本控制和修改，您需要选择使用SSH密钥或者HTTPS协议克隆代码仓库；如果您只需要获取代码仓库的某个或者某些文件，可以选择使用浏览器下载代码仓库。

须知

- 如果要克隆代码仓库并在本地进行代码开发，请先在CodeArts Repo进入要克隆的代码仓库主页，选择“分支” > “新建分支”，基于主分支创建一条属于您的开发分支。
- CodeArts Repo当前仅支持一次克隆一个代码仓库，如果想要一次克隆多个代码仓库到本地，您可以通过Shell或者批处理命令实现多个仓库下载。

11.2 使用 SSH 密钥克隆代码仓库到本地

使用 Git Bash 克隆代码仓库到本地

SSH密钥是一种安全的身份验证方式，用于访问远程服务器。使用SSH密钥克隆代码仓可以避免每次都需要输入用户名和密码，提高克隆代码仓的效率。

步骤1 登录**CodeArts Repo首页**。

步骤2 进入要克隆的代码仓库主页，您可以先创建个人分支，再单击“克隆/下载”按钮，并复制SSH地址。

步骤3 在本地Git Bash客户端，执行如下命令，进入您要克隆代码仓的地址，该命令表示克隆的代码仓将克隆到D盘的Repo文件夹下，您可以根据需要修改地址。

```
cd D:/Repo
```

步骤4 执行如下命令，克隆代码仓到该目录下。

```
git clone 代码仓库的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。

如果出现**下图**，说明克隆仓库成功。

图 11-1 使用 SSH 密钥克隆代码仓成功示意图

如果在执行步骤3时，Git Bash报错“git@test.com: Permission denied.fatal: Could not read from remote repository.Please make sure you have the correct access rights and the repository exists.”，表示您还未配置访问Repo的SSH密钥，请先配置SSH密钥，具体请参考**配置SSH密钥**。

----结束

使用 TortoiseGit 克隆代码仓库到本地

步骤1 登录**CodeArts Repo首页**。

步骤2 进入要克隆的代码仓库主页，单击“克隆/下载”按钮，并复制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**步骤2**复制的SSH地址粘贴到URL输入框中，勾选“加载Putty密钥”并选择安装TortoiseGit客户端时生成的私钥文件。

步骤5 单击“确定”。如果您是第一次在TortoiseGit客户端克隆代码仓，系统会询问您是否信任远程仓库，单击“是”即可。

----结束

11.3 使用 HTTPS 协议克隆代码仓库到本地

使用 Git Bash 克隆代码仓库到本地

步骤1 登录**CodeArts Repo首页**。

步骤2 进入要克隆的代码仓库主页，单击“克隆/下载”按钮，并复制HTTPS链接。

步骤3 在本地Git Bash客户端，执行命令**cd D:/Repo**，进入您要克隆代码仓的地址。如下命令表示克隆的代码仓将克隆到D盘的Repo文件夹下。

步骤4 执行如下命令，克隆代码仓到该目录下。

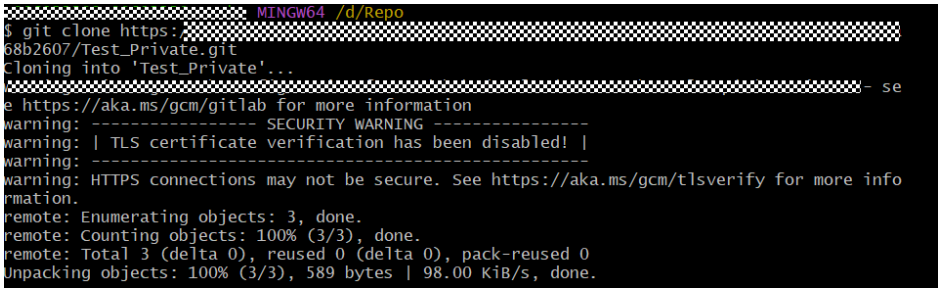
```
git clone 代码仓库的HTTPS链接
```

如果您是第一次克隆代码仓库，您需要填写用户名和密码，有两种类型的用户名和密码，根据您的配置情况，选择如下的一种方式即可：

- 如果需要查看用户名和密码，请登录并进入Repo的代码仓库列表页，单击右上角昵称，选择“个人设置” > “代码托管” > “HTTPS密码”，获取您的用户名和密码，如果忘记密码，可以重新设置HTTPS密码。
- Token用户名和密码。其中，Token的用户名为“private-token”，Token密码为您配置的Token，如果遗失或忘记，可参考**配置访问令牌**重新生成Token。

如果出现**下图**，说明克隆仓库成功。如果克隆代码仓库失败，请根据**说明**去排查解决问题。

图 11-2 使用 HTTPS 协议克隆代码仓成功示意图



----结束

说明

- 在执行**步骤3**时，Git Bash报错“fatal: unable to access 'https:test.com/Test_Private.git/': SSL certificate problem: unable to get local issuer certificate”，请在执行git clone 命令前，先执行如下命令，使Git在使用HTTPS协议克隆代码仓时不进行SSL证书验证：
  ```
  git config --global http.sslVerify false
  ```
- 在执行**步骤3**时，Git Bash报错“fatal: unable to access 'https://test.com/Remote_Test.git/': Failed to connect to test.com port 443 after 21161 ms: Couldn't connect to server”，表示网络不通，请联系您本地所属网络管理员。
- 在执行**步骤3**时，Git Bash报错“fatal: unable to access 'https://xxx.git/'': Recy failure: Connection was reset”，表示域名解析错误，解决办法请参考常见问题。
- 在执行**步骤3**时，Git Bash报错“fatal: destination path 'Test_Private' already exists and is not an empty directory.”，表示Test_Private代码仓已克隆到该路径下且代码仓不为空，解决办法：切换一个新的空目录，重新执行**步骤3**。
- 在执行**步骤3**时，Git Bash报错“fetal: Authentication failed for 'https:/xxx.git/'”，表示您的密码有误，可以登录并进入Repo的代码仓库列表页，单击右上角昵称，选择“个人设置”>“代码托管”>“HTTPS密码”，获取您的用户名和密码，如果忘记密码，可以重新设置HTTPS密码。
- 在CentOS系统下使用HTTPS协议克隆代码时，报错“The requested URL returned error: 401”。这是由于Git版本不匹配。
- 如果您想要通过将访问令牌嵌入HTTPS下载链接，您可以在**步骤3**执行如下命令。其中，password为通过您配置的Token，如果遗失或忘记，可参考**配置访问令牌**重新生成，{project_name}为项目名称，{repository_name}为要克隆的代码仓库名称。
  ```
  git clone https://private-token:password@codehub.test.com/{project_name}/{repository_name}.git
  ```

11.4 使用浏览器下载代码包到本地

Repo不仅支持克隆代码仓，同时支持将仓库代码打包下载到本地。

步骤1 登录**CodeArts Repo首页**。

步骤2 进入要克隆的代码仓库主页，切换到需要下载的分支，单击“克隆/下载”按钮。

步骤3 在弹出的窗口中单击需要的代码包类型即可直接下载。

----结束

说明

- 切换分支后，下载的压缩包为指定分支的内容。
- 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。

12 上传代码文件到 Repo

12.1 在 Repo 编辑并创建合并请求

进入要编辑的代码仓库首页，单击“代码”进入代码首页，基于要合入的代码分支新建一个分支。选择要基于修改的分支，根据您的选择进行编辑代码和新建合并请求：

- 如果要新增某个代码文件，单击“新建”，可以新建代码文件，也可以从本地上传单个代码文件，基于某个分支修改后，在“代码”页面右侧，单击“新建合并请求”。
- 如果要在线修改某个代码文件，在“代码”页面，单击要修改的文件名，进入要修改的文件页面，单击 进入文件的编辑模式，编辑并保存后，单击“新建合并请求”，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。

须知

分支名不支持以 - . refs/heads/ refs/remotes/ 开头，不支持空格 [\ < ~ ^ : ? * ! ( ) ' " | 等特殊字符，不支持以. / .lock结尾。

12.2 在 Git Bash 创建分支并开发代码

步骤1 进入本地仓库目录，打开Git Bash。执行如下命令，基于master分支新建一条分支feature1，并切换到feature1分支。

```
git checkout -b feature1
```

步骤2 以下步骤模拟将字符串“hello CR”写入到名为hello_cr.txt的文件中。

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

步骤3 将当前目录下所有修改过的文件添加到Git的暂存区中，准备提交到版本库。

```
git add .
```

步骤4 将当前修改的代码提交到本地代码仓库中，并添加一条提交信息。

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


步骤5 查看最近一次提交的详细信息。

```
git log -1
```

步骤6 执行如下命令，将本地分支feature1推送到您远程仓库的origin分支，并将本地分支与远程分支建立追踪关系。

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

说明

- 执行步骤6时，如果提示“connect to host ********.com port 22: Connection timed out”，表示您的网络被限制，无法访问代码托管服务，请求助您本地所属网络管理员。
- 如果您在创建commit后，把本地路径带到CodeArts Repo的代码仓库里，您不能更改提交代码的路径，只能本地删除该文件或者回退commit强制提交，然后重新提交。
- 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7 进入要新建合并请求的代码仓库首页，选择“合并请求” > “新建”，选择要发起合并请求的源分支和目标分支。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信息。

----结束

须知

- 提交本地代码到CodeArts Repo，需要使用git push命令，git commit只是将本地仓库中的修改保存到本地仓库中，每次commit都会生成一个新的commit记录，记录了修改的内容、作者、时间等信息。commit操作只会将代码更改保存到本地仓库中，并不会将修改同步到远程仓库。
- 向CodeArts Repo推送代码时，提示“You are not allowed to push code to protected branches on this project”。原因是该分支为受保护分支，您没有权限推送代码到这个分支。解决方案：仓库所有者或者项目管理员进入代码仓库详情页，选择“设置 > 策略设置 > 保护分支”，单击  ，解除对该分支的保护。
- 向CodeArts Repo推送代码时，提示“src refspec master does not match any”。原因是您没有使用git add、git commit命令依次将文件从工作区加入暂存区。解决方案：在执行git push命令之前，请先使用git add、git commit将修改后的文件提交至暂存区中，再使用push命令推送至云端代码仓库中。
- 向CodeArts Repo推送代码时，提示“error: failed to push some refs to 'https://codehub”。原因是CodeArts Repo的该仓库与本地仓库代码不一致，所以从本地提交代码的操作被拒绝。解决方案：先使用git pull命令拉取CodeArts Repo远端仓的代码，与本地代码仓库合并，再使用git push命令推送代码到CodeArts Repo。
- 使用git pull命令拉取代码失败，提示“Merge branch "master’ of https://codehub/testMaven Please enter a commit message to explain why this merge is necessary”。原因是CodeArts Repo的代码仓库与您本地仓库内容不一致，拉取代码时会跟本地代码进行合并（merge），弹框是提示是否确认本次merge操作，并提交备注信息。解决方案请参考**使用git pull拉取代码失败，报错"Merge branch 'master' of https://xx.com Please Enter a commit"**。
- 执行步骤4时，如果报错“unable to auto-detect email address”，原因是未设置用户名、邮箱。您可以执行如下命令配置您的个人信息。
  ```
  git config --global user.name {你的名字}
  git config --global user.email {你的邮箱}
  ```
- 执行步骤6时，如果报错“'origion' does not appear to be a git repository...”，原因是远程不存在origion这个仓库名称，具体解决方案请参考**执行git push 命令时，报错'origion' does not appear to be a git repository...**。
- 执行步骤3时，如果报错“Not a git respository”，原因是您不在当前代码仓目录下，需要使用cd命令进入代码仓库目录下。
- 提交合并请求时，如果报错"failed to push some refs to '....git'"，请参考**解决合并请求冲突**。
- 执行步骤6时，提示“Connection reset by test port 22 fatal: Could not read from remote repository.”，原因是网络不稳定，该请求被重置，如果该问题是偶现，可能为网络原因。

12.3 在 Eclipse 提交代码并创建合并请求

如果您本地的Eclipse安装了EGit，可以把本地Git代码仓库代码提交到远程CodeArts Repo，CodeArts Repo当前仅支持Ecplise 4.4及以上版本。

说明

- 如果是首次提交：
  1. 在本地计算机建立一个仓库，称本地仓库。
  2. 在本地进行**Commit**，将更新提交到本地仓库。
  3. 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**Push**到服务器端，即进行一次远程提交。
- 如果非首次提交：
  1. 将修改的代码**Commit**更新到本地仓库。
  2. 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**Push**到服务器端。

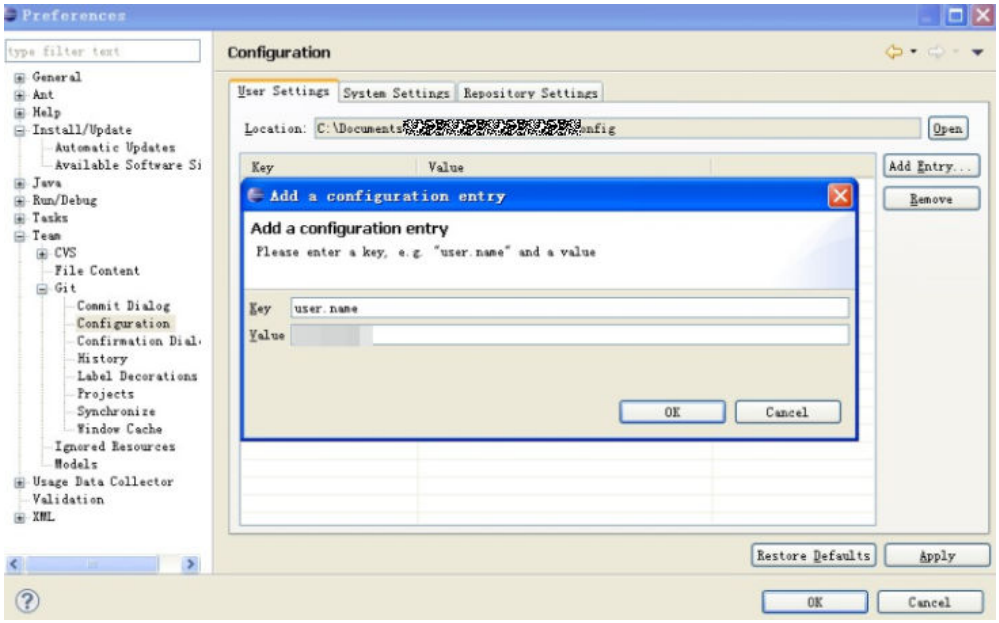
步骤一：在 Eclipse 上安装 EGit 插件

执行如下步骤，安装Eclipse的4.4版本：

在Eclipse上方工具栏选择“Help > Install New Software...”，弹出的“Install”窗口中，单击“Add...”，“Name”栏填写“EGit”，“Location”栏填写**EGit插件地址**，单击“OK”按钮，随后连续单击“Next >”默认安装，安装完成后重启Eclipse。

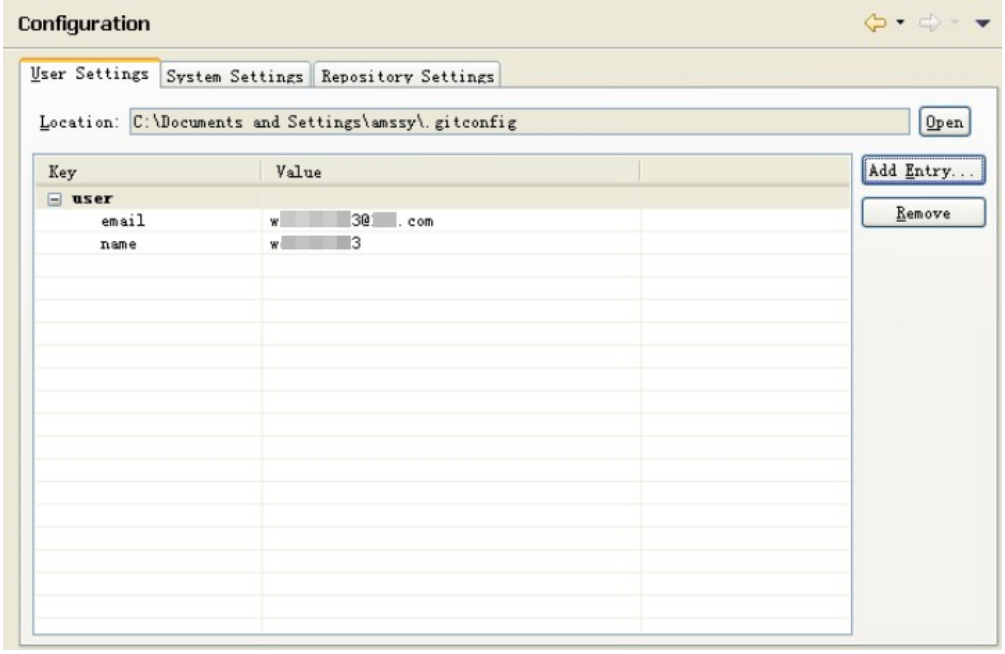
步骤二：在 Eclipse 中配置 EGit

1. 在Eclipse上方工具栏选择“Window > Preferences > Team > Git > Configuration”，填写“User Settings”信息。



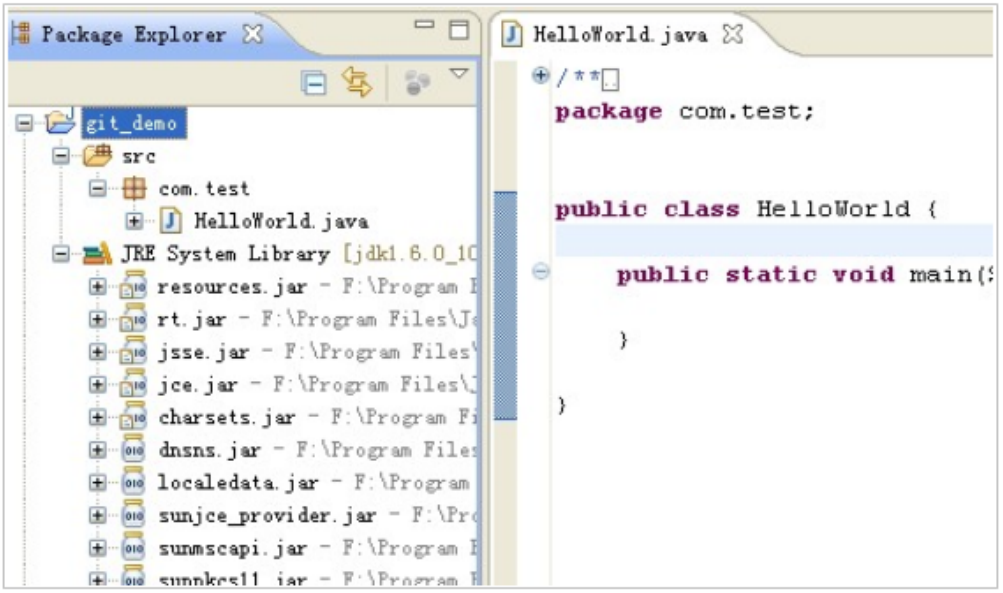
2. 单击“OK”，如下图所示。

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

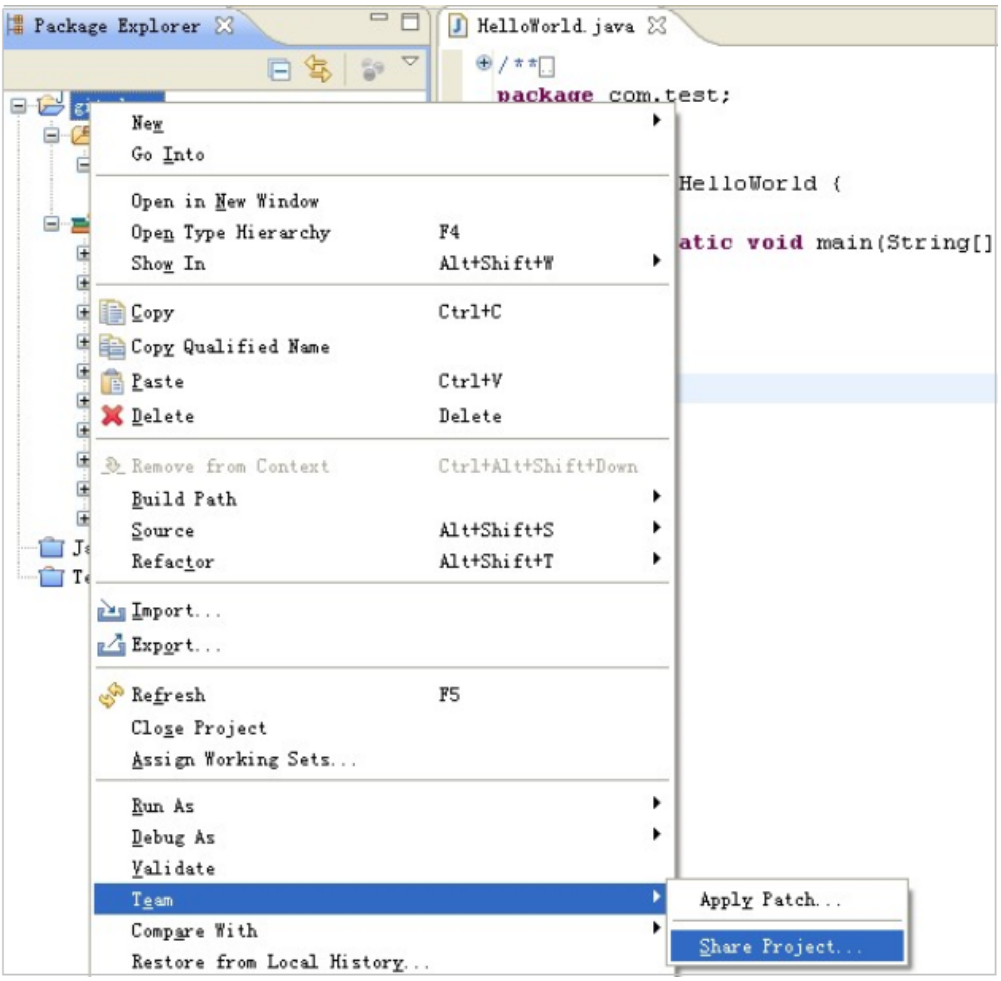


步骤三：新建项目，并将代码提交到本地的 Git 仓库中

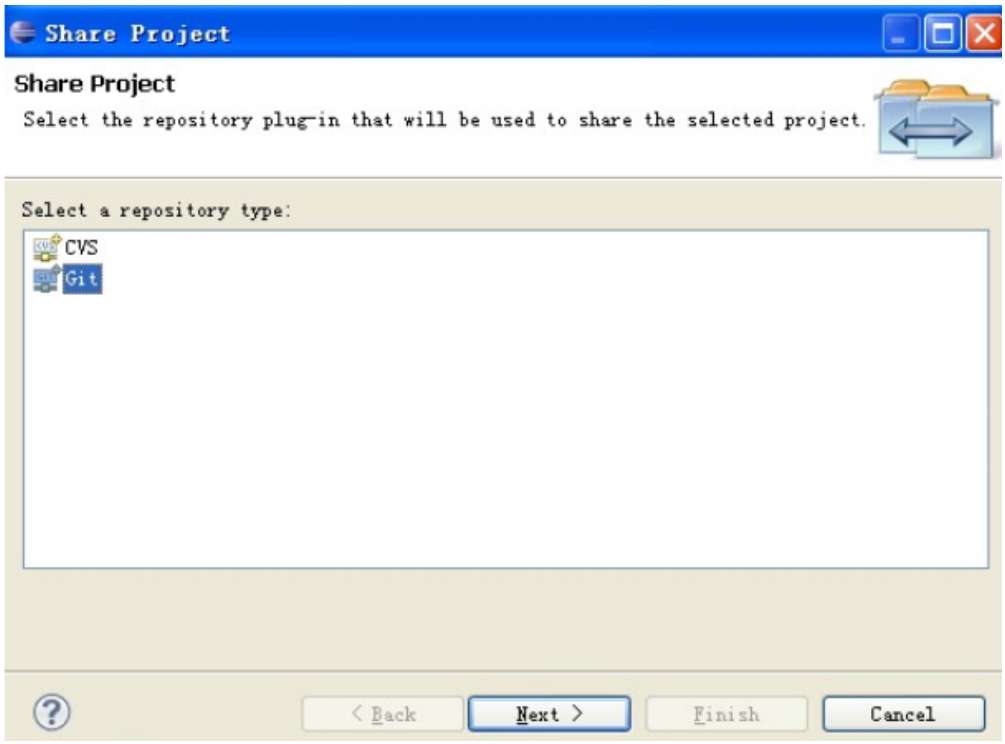
1. 新建项目“git_demo”，并新建“HelloWorld.java”类，如下图所示。



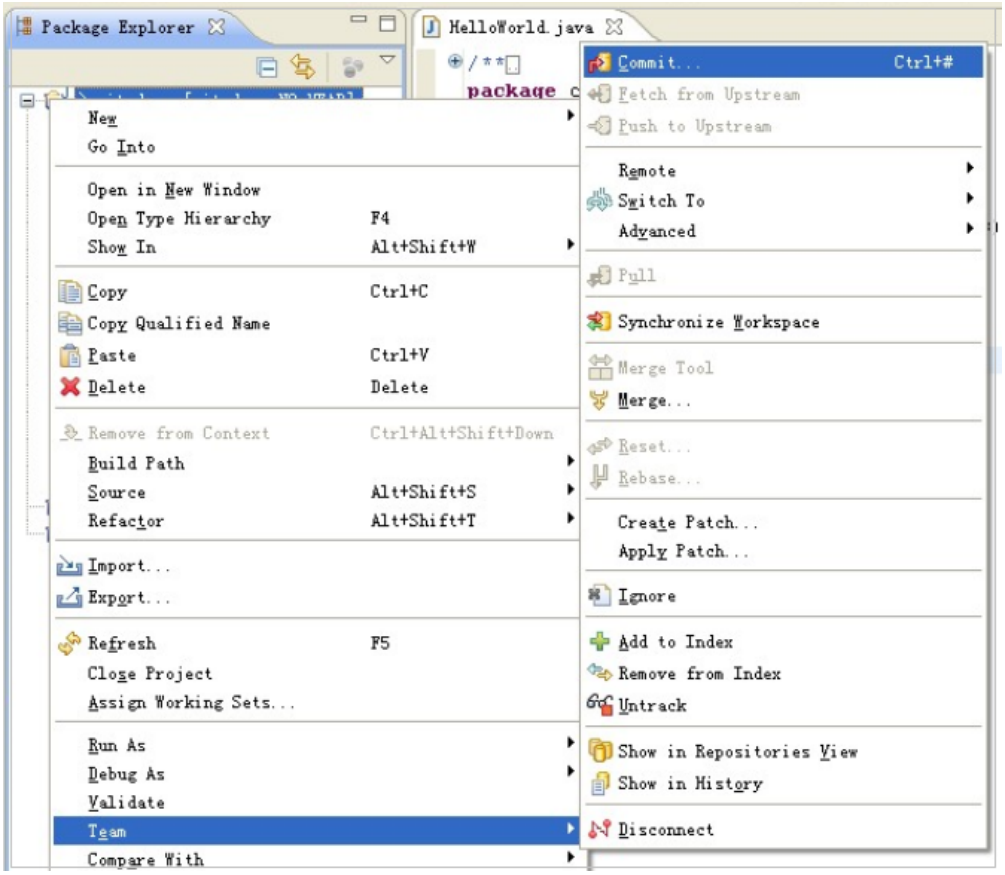
2. 将“git_demo”项目提交到本地仓库，如下图所示。



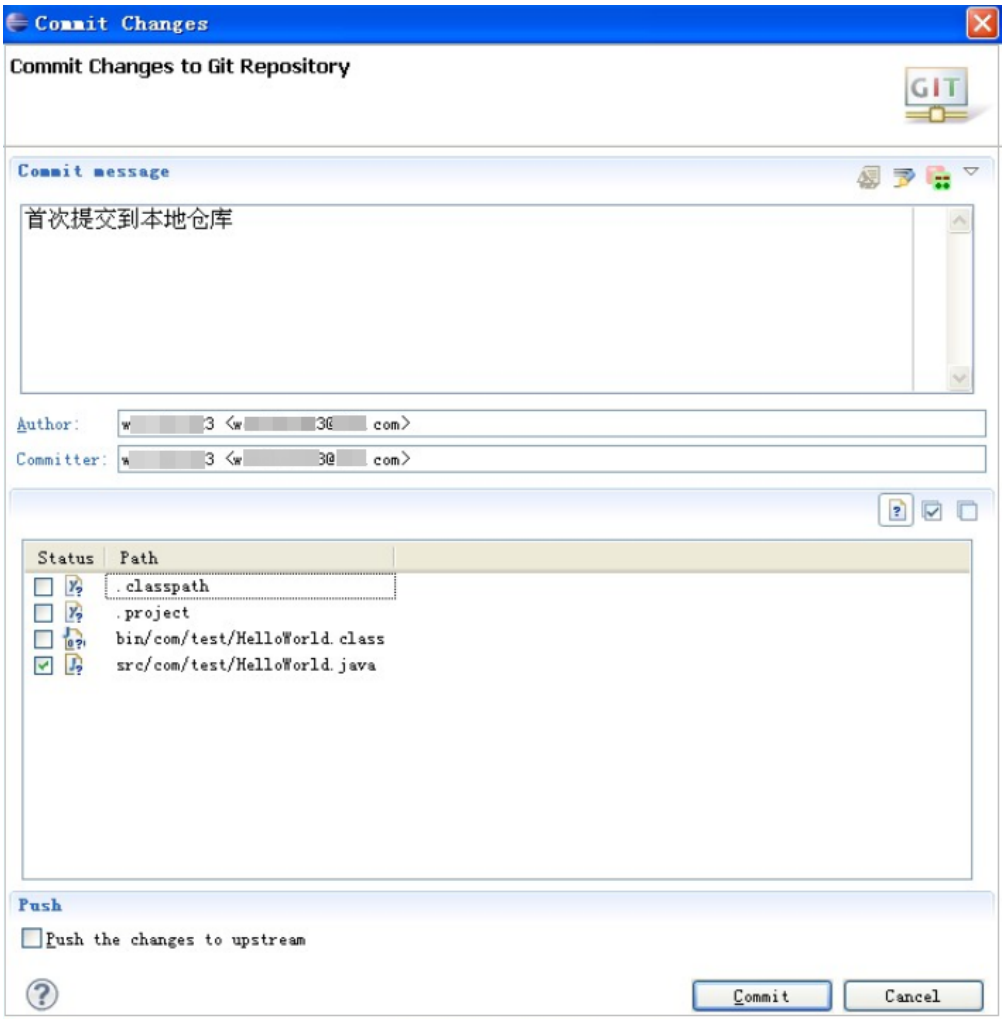
3. 在弹出的“Share Project”窗口中，选中“Git”，如下图所示。



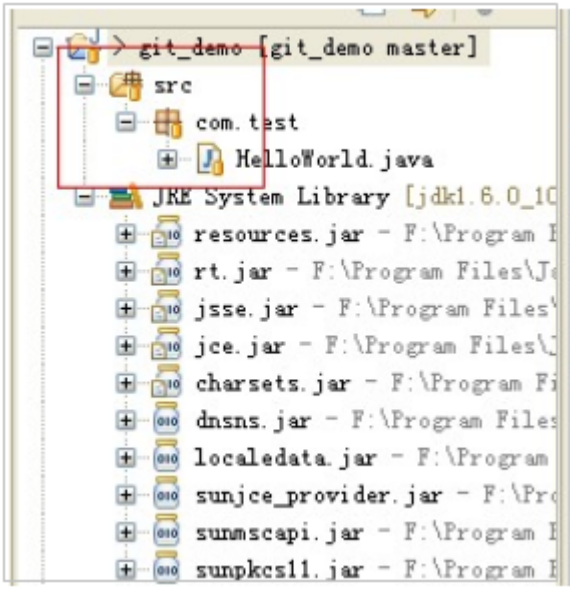
4. 单击“Next >”，弹出“Configure Git Repository”，勾选“Use or create repository in parent folder of project”，单击“Create Repository”。
5. 单击“Create Repository”，成功创建Git仓库。
   文件夹此时处于“untracked”状态（文件夹中以符号“?”表示）。
   此时需要提交代码到本地仓库，如下图所示开始提交。



6. 弹出“Commit Changes”窗口，设置提交信息，如下图所示。

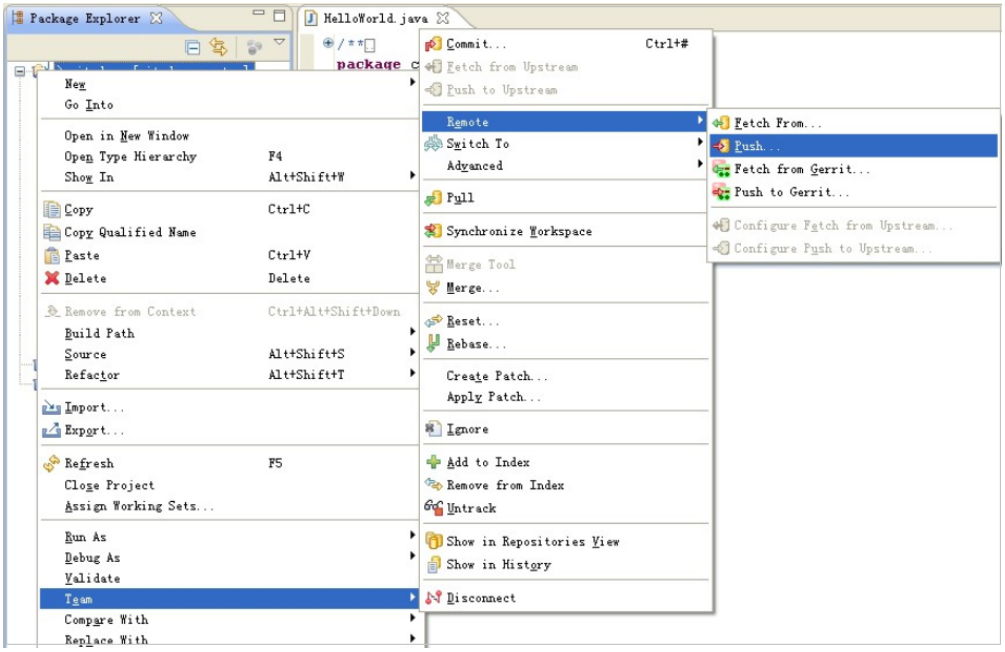


7. 单击“Commit”，代码提交到本地仓库，如下图所示。

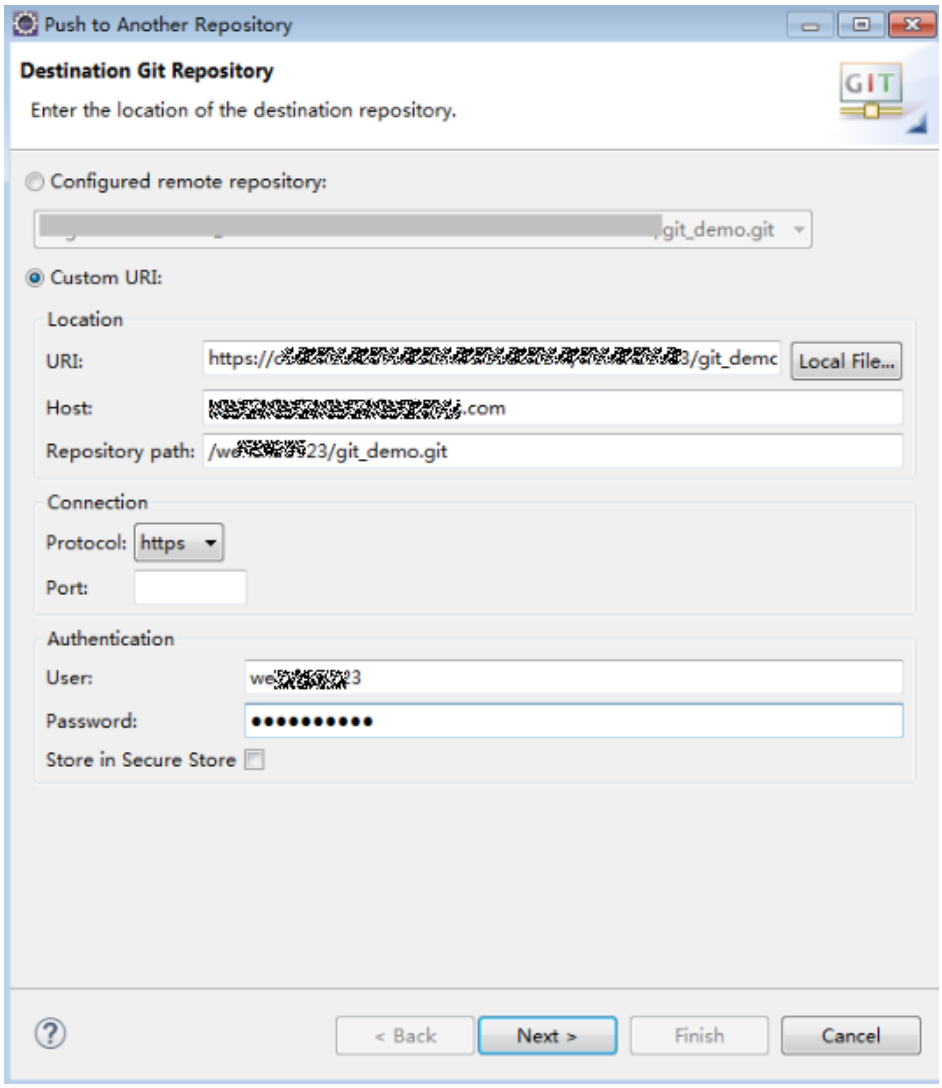


步骤四：将本地仓库代码提交到远程的 Git 仓库中

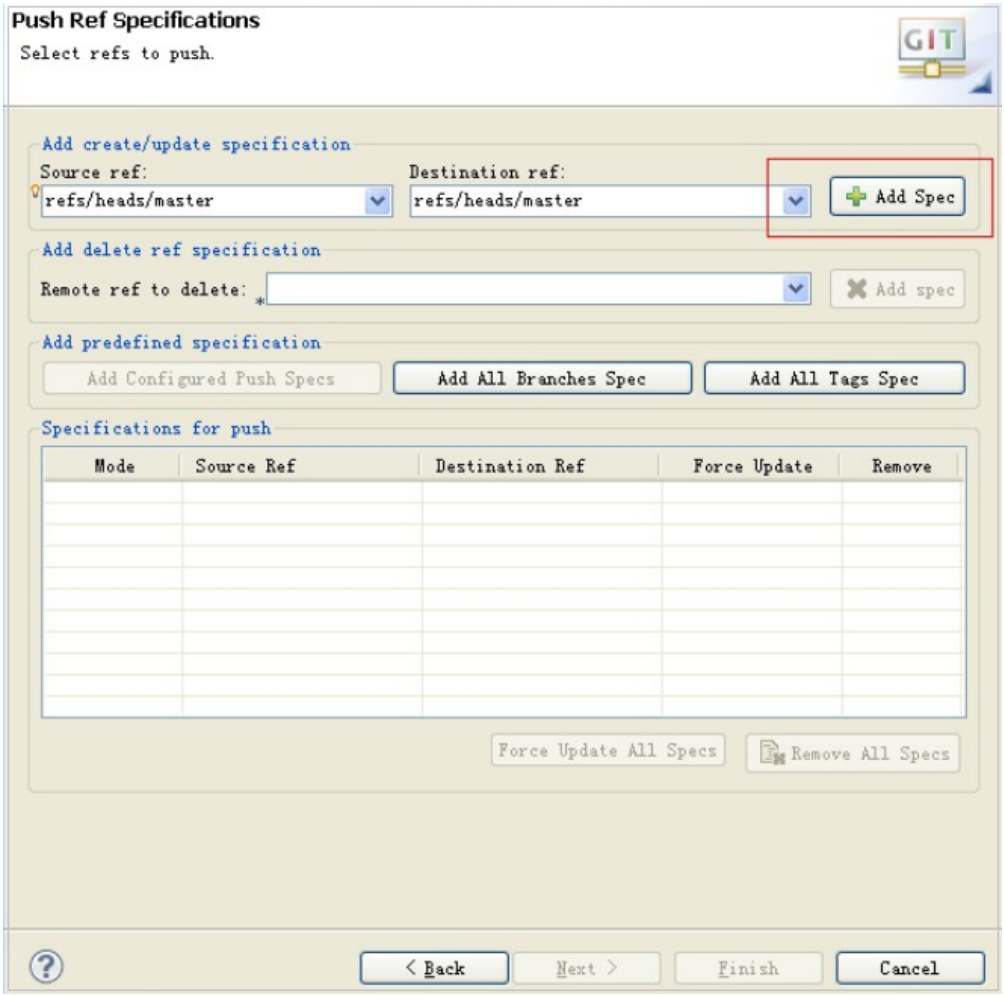
1. 在代码托管服务中**创建仓库**。
   创建好远程仓库后，进入远程代码仓库详情页面，可以复制远程仓库地址。
2. 选择Push菜单，开始将代码提交到远程仓库，如下图所示。



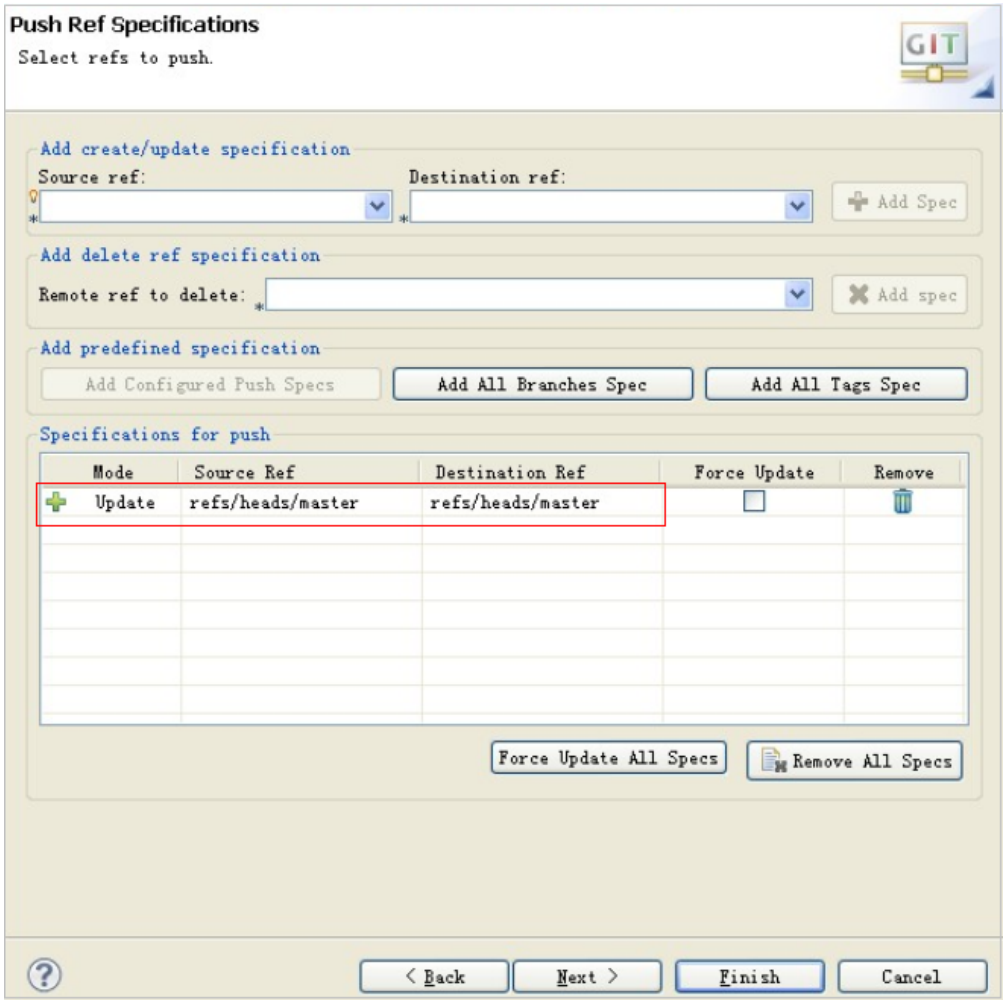
3. 在弹出的“Push to Another Repository”窗口中，设置相应参数，如下图所示。



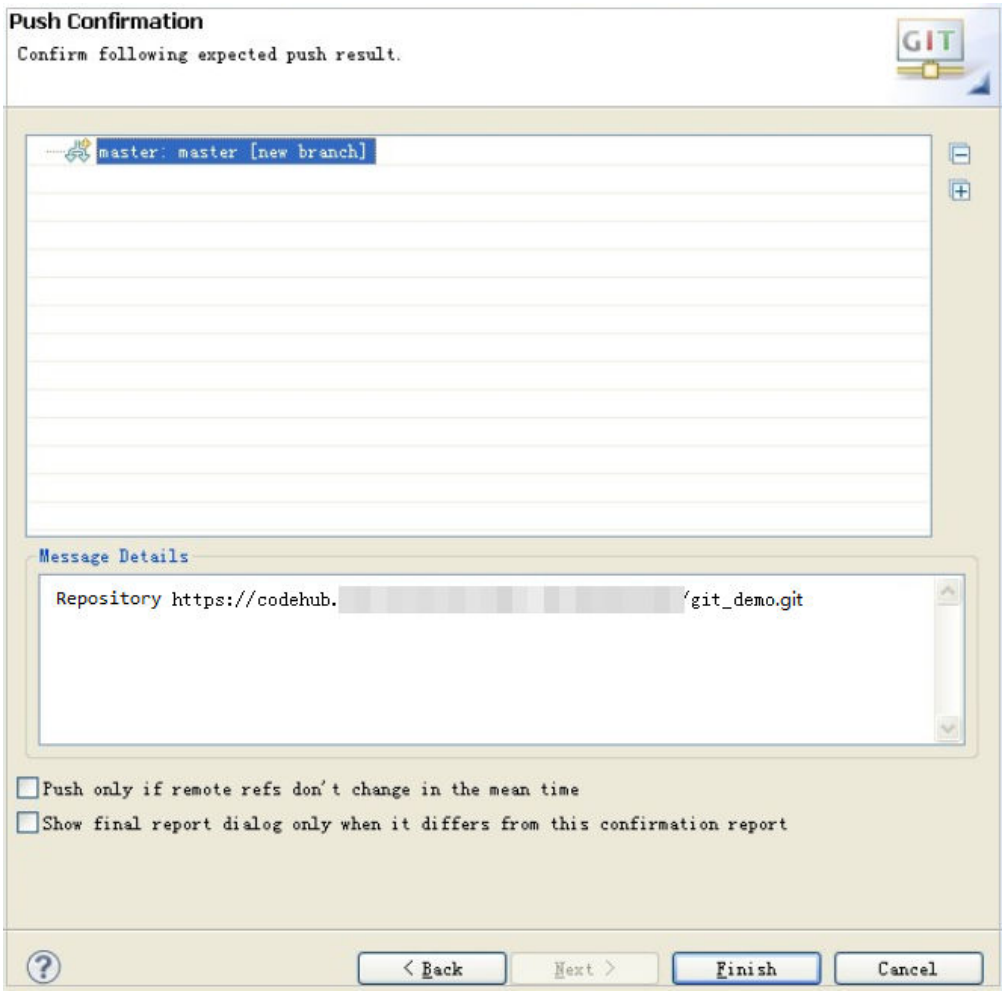
4. 单击“Next”，弹出“Push Ref Specifications”，如下图所示。



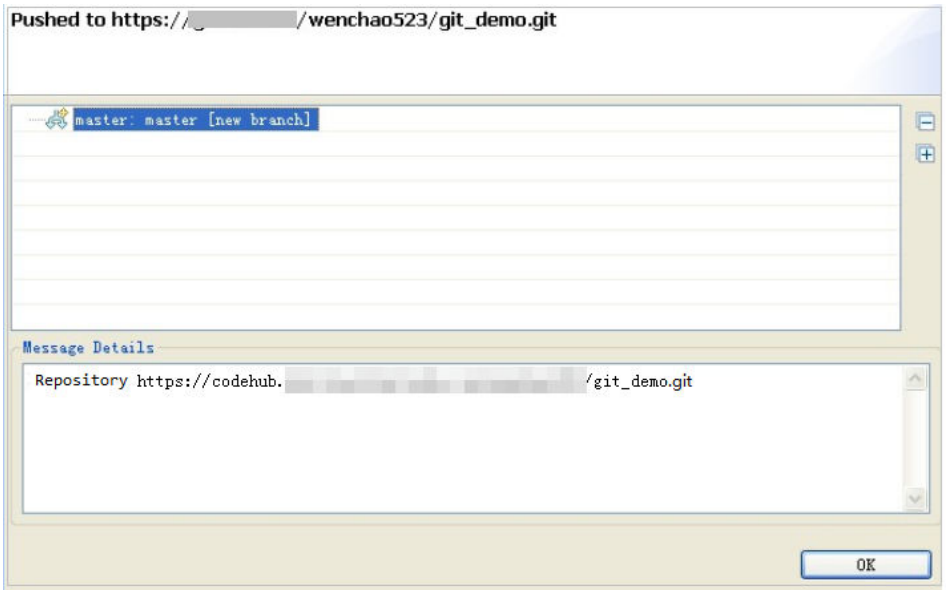
5. 单击“Add Spec”，成功添加，如下图所示。



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。

说明

在Eclipse使用HTTPS方式连接CodeArts Repo的代码仓库时，提示“Transport Error: cannot get remote repository refs. XXX.git: cannot open git-upload-pack”，这是由于Eclipse中Egit插件的配置问题。解决方案：在Eclipse中，右键选择“Windows > Preferences > Team > Git > Configuration > User Settings”。单击“Add Entry”，“Key”填写内容“http.sslVerify”，“Value”填写内容“false”。

步骤五：在 CodeArts Repo 新建合并请求

进入要新建合并请求的代码仓库首页，选择“合并请求” > “新建”，选择要发起合并请求的源分支和目标分支。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信息。

12.4 在 Git 客户端使用 git-crypt 传输敏感数据

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。git-crypt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，并且该仓库可以同普通仓库一样被拉取和推送，只有持对应文件密钥的人才能查看到加密文件的内容，但不会限制参与者对非加密文件读写。

在 Windows 中使用密钥对方式进行加密、解密

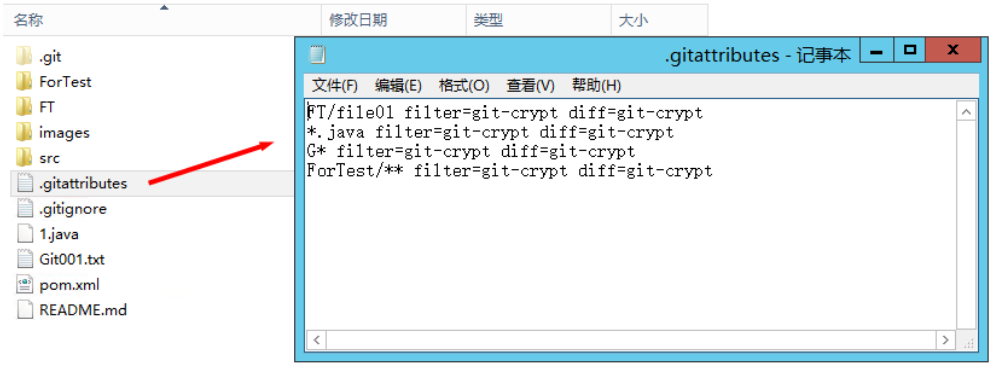
步骤1 下载并安装最新的Windows Git客户端，下载最新基于Windows的git-crypt，把下载到的exe文件放到Git安装目录下的“cmd”文件夹中。

步骤2 执行如下命令，在本地生成密钥对。

1. 打开“Git Bash”，并进入本地代码仓库。
2. 执行如下命令，在Git代码仓库中创建“.git-crypt”文件夹，文件夹包含加密文件所需的密钥和配置文件。
   ```
   git-crypt init
   ```
3. 执行如下命令，把密钥文件导出到C:/test目录并命名为KeyFile。
   ```
   git-crypt export-key /c/test/keyfile
   ```
4. 执行完上述步骤，您可以到密钥导出的文件路径进行验证，确认是否已成功生成密钥。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3 执行如下命令，为代码仓库配置加密范围。

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
   ```

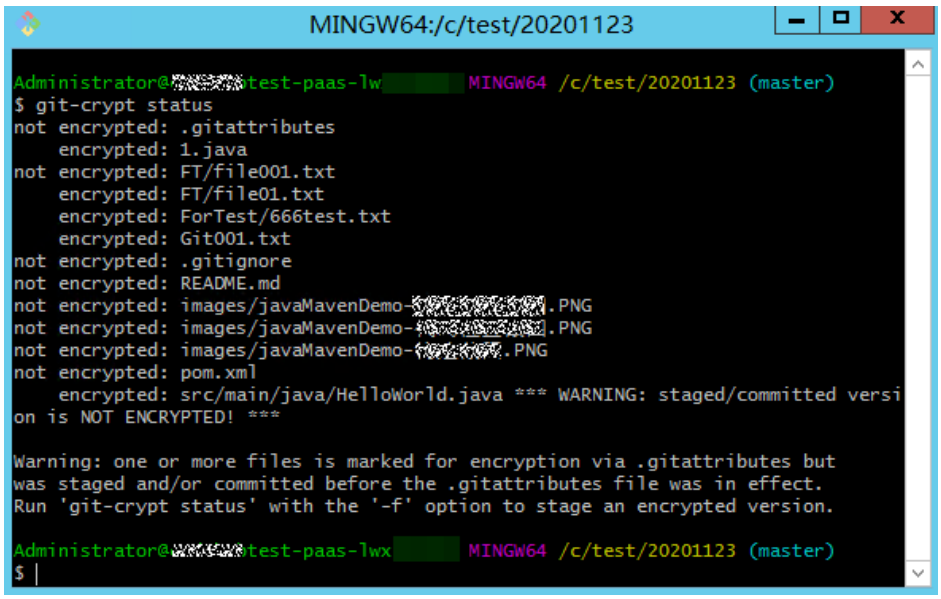
说明

- 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，否则配置会无效。

步骤4 进行文件加密。

在仓库根目录打开Git bash，执行如下指令即可完成加密，加密后可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。

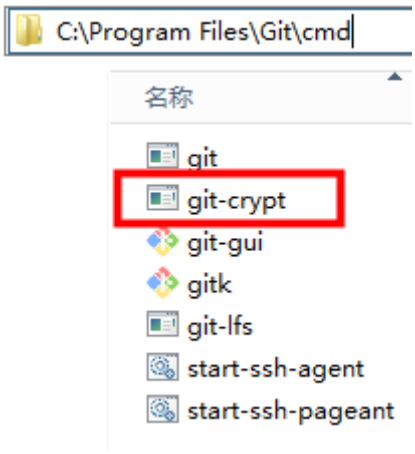
说明

“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”。

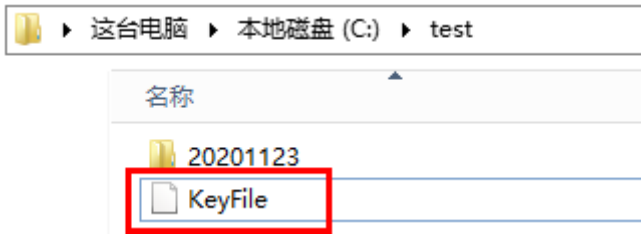
在让团队合作中 -f （强制执行）具有一定的风险，请谨慎使用。

步骤5 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。

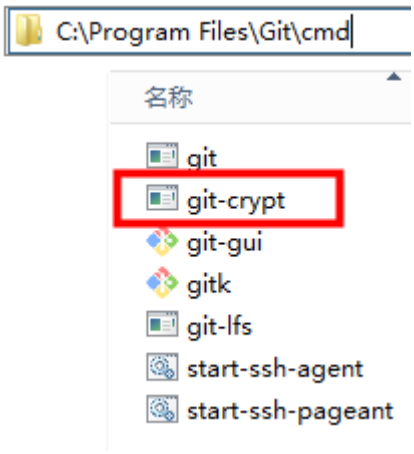
```
git-crypt unlock /C/test/KeyFile     #请将 /C/test/KeyFile 更换为您实际的密钥存储路径
```

----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。



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

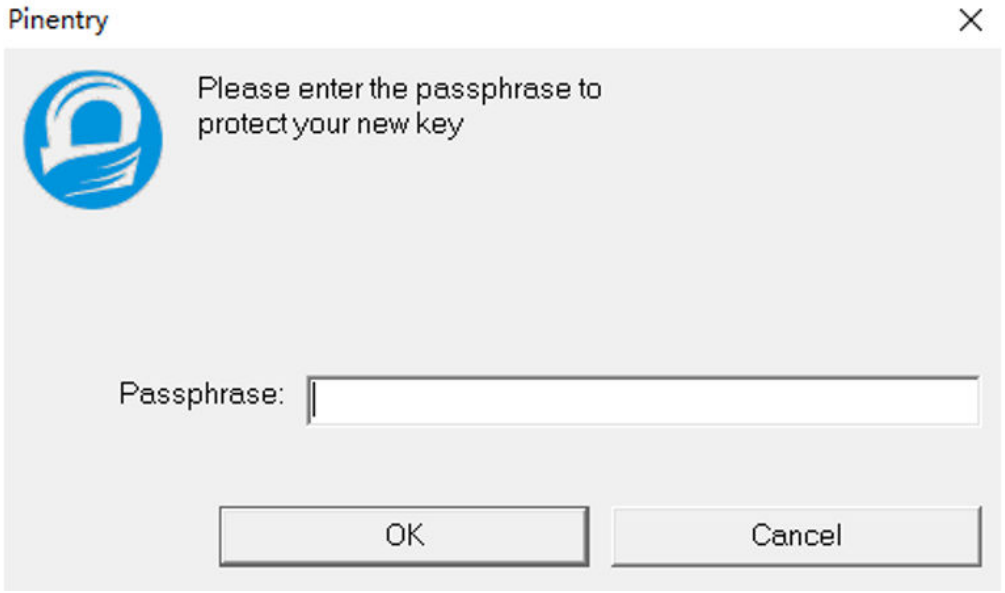
双击进行安装，单击“下一步”，即可完成安装。

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-[redacted] MINGW64 /c/dev/test
$ gpg --gen-key
gpg (GnuPG) 2.2.23-unknown; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

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

GnuPG needs to construct a user ID to identify your key.

Real name: gpgTest
Email address: gpgTest@huahua.com
You selected this USER-ID:
    "gpgTest <gpgTest@huahua.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? |
```

3. 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，建议设置新的密码。

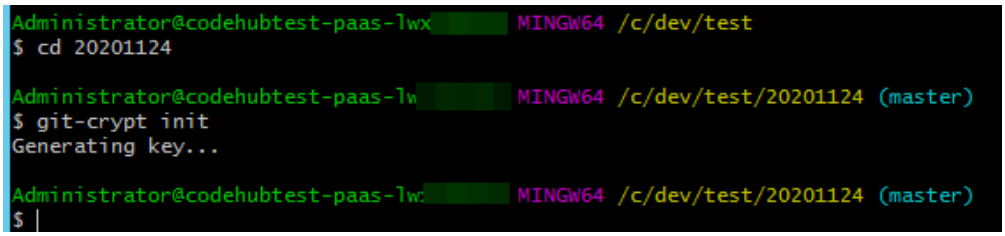
4. 显示如下图返回内容，则为GPG密钥对生成成功。

```
public and secret key created and signed.

pub   rsa3072 2020-11-24 [SC] [expires: 2022-11-24]
      0D[redacted]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。

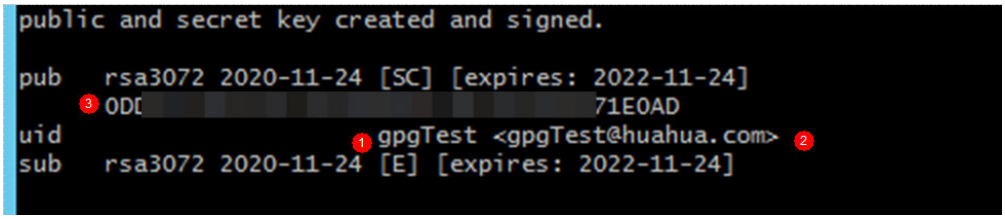
```
git-crypt init
```

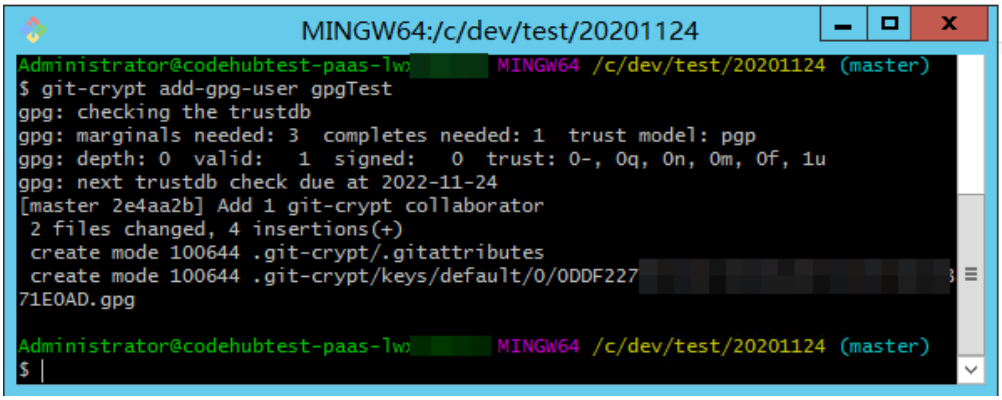
2. 将密钥副本添加到您的仓库，该副本已使用您的公共GPG密钥加密，指令如下。

```
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）这三种是可以唯一标识此密钥的凭证。



执行后会提示您创建了.git-crypt文件夹以及其中的两个文件。



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。

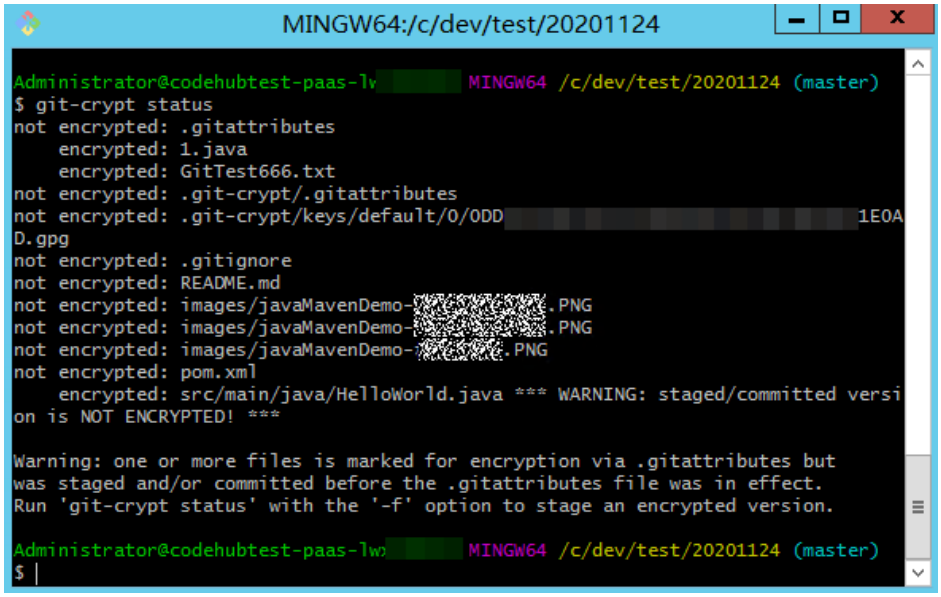
```
FT/file01.txt filter=git-crypt diff=git-crypt   #将 特定文件加密，这里加密的是FT文件夹下的file01.txt
*.java filter=git-crypt diff=git-crypt   #将 .java类型文件加密
G* filter=git-crypt diff=git-crypt       #将 文件名为 G 开头的文件加密
ForTest/** filter=git-crypt diff=git-crypt #将 ForTest 文件夹下的文件加密
```

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

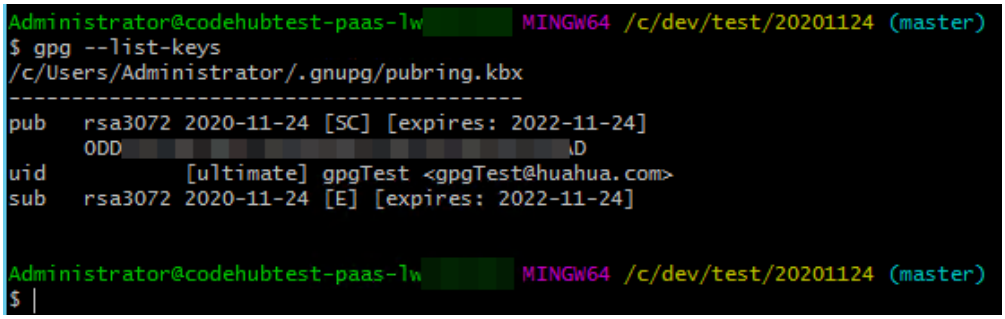
“git-crypt status”只会加密本次待提交的文件，对本次未发生修改的历史文件不会产生加密作用，Git会对此设定涉及的未加密文件做出提示（见上图中的Warning），如果想将仓库中的对应类型文件全部加密，请使用“git-crypt status -f”。

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
```



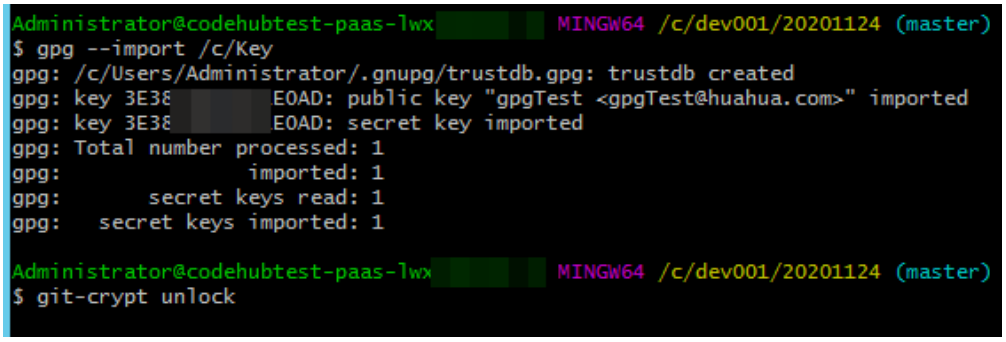
2. 使用GPG --export-secret-key指令导出密钥，本示例中将名称为“GPGTest”的密钥导出到“C盘”，并命名为“Key”。
```
GPG --export-secret-key -a GPGTest > /c/key
```
执行时会提示您输入此次密码，正确输入即可。

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。
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**步骤8**，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。导入时会提示您输入密钥的密码。
```
GPG --import  /c/key
```
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。



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

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。

  下载源码

  ```
  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。

  下载源码

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

- 使用Debian包安装git-crypt。

  下载源码

  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
  ```


13 开发协作工作流

13.1 工作流概述

Git工作流不仅可以进行版本控制，还可以管理项目流程和团队协同开发，有效提高项目管理水平和团队协同开发能力。在实际应用中，有必要根据团队的需求和工作流程，选择合适的Git工作流，实现持续集成、持续交付和快速迭代的目标。

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

13.2 集中式工作流

集中式工作流适合刚从SVN工具转型为Git的小型团队。集中式工作流的开发都在一个中心仓库进行，开发者从中心仓库克隆代码仓库，开发完成后，将代码推送到中心仓库。

集中式工作流优点

- 中央管理。所有的代码仓都存储在一个中心仓库，方便管理和维护代码。
- 高效协作。团队成员可以通过中央仓库进行代码的共享和协作。
- 安全可靠：中央仓库可以备份和恢复，保证代码的安全可靠。

集中式工作流缺点

- 依赖中央仓库：所有的代码都依赖于中央仓库，如果中央仓库出现问题，整个团队的开发工作都将受到影响。
- 代码冲突：由于所有的代码都在中央仓库中进行管理，团队成员在进行代码修改时容易发生冲突，需要通过手动解决冲突来保证代码的正确性。
- 需要权限管理：由于所有的代码都在中央仓库中进行管理，需要对团队成员的权限进行管理，以保证代码的安全性和可靠性。

- 不适合大型项目团队：对于大型项目团队而言，集中式工作流可能会导致中央仓库的管理和维护变得困难，影响开发效率和代码质量。

集中式工作流流程

步骤1 创建代码仓库。Repo目前支持**新建自定义代码仓库**、**按模板新建代码仓库**、**Fork已有的代码仓库**，也支持**从本地导入已有的代码仓库**、**导入Git平台的代码仓**、**导入SVN平台的代码仓**。

步骤2 开发者克隆代码仓。Repo目前支持**使用SSH密钥克隆代码仓到本地**、**使用HHPS协议克隆代码仓到本地**。

步骤3 开发者**在本地创建分支并开发代码**或者**在线创建分支分支并开发代码**。

步骤4 开发者提交更改的代码文件到缓存区。Repo目前支持**使用Git Bash提交代码**、**在Eclipse提交代码**。

步骤5 开发者**新建合并请求**。

步骤6 开发者**解决检视意见**。

步骤7 Committer**合入合并请求**。

----结束

13.3 Git Flow 工作流

14 新建并配置 CodeArts 项目设置

14.1 配置项目级的代码仓库设置

在代码托管首页，进入项目首页，选择“设置” > “策略设置” > “仓库设置”。参数填写请参见表格表14-1。

表 14-1 项目级仓库设置参数填写表格

参数	说明
开启强制继承	非必填参数。如果勾选此参数，本项目下的所有代码组和代码仓库均使用以下参数的设置，且代码组和仓库下设置不可更改，请谨慎选择。
禁止Fork仓	非必填参数。勾选此选项，表示任何人不可以Fork该项目下的代码仓库。
MR预合并	非必填参数。勾选此选项，表示启用MR预合，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。
分支名规则	非必填参数。所有分支名都必须匹配正则表达式，分支名规则不能超过500个字符。如果此字段不填写，则允许任何分支名。规则需要满足基本的Tag命名规则： • 不能超过500个字符。 • 不支持以 - . refs/heads/ refs/ remotes/ 开头，不支持空格 [\ < ~ ^ : ? * ! ( ) ' " | 等特殊字符，不支持以. / .lock结尾。

参数	说明
Tag名规则	非必填参数。所有Tag名都必须匹配正则表达式。如果此字段不填写，则允许任何Tag名。 需满足基本的Tag命名规则： • 不能超过500个字符。 • 不支持以 - . refs/heads/ refs/ remotes/ 开头，不支持空格 [\ < ~ ^ : ? * ! ( ) ' " | 等特殊字符，不支持以. / .lock结尾。

配置项目级的保护分支规则

CodeArts Repo可以保护代码分支的安全性，阻止管理者外的人推送代码、阻止任何人强行推送代码或者阻止任何人删除这个分支，您可以将这个分支设置保护分支。具体具体操作过程如下：在代码托管首页，进入项目首页，选择“设置” > “策略设置” > “保护分支”，单击“新建保护分支”，请根据如下步骤填写参数：

步骤1 填写分支名称。该参数必填，请您根据自己的需要输入完整的分支名或者带通配符的分支名。如果分支中包含单斜杠（/），由于fnmatch语法规则，该分支无法用通配符“*”匹配。

步骤2 可以为管理员/项目经理、committer和开发人员设置推送或者合并的权限，两种权限不能同时拥有，原因是保护分支不能强行被推送代码或者合入代码，支持批量新建、编辑、删除保护分支。

----结束

如果您想让本项目下所有代码组和仓库均使用以上设置，勾选“开启强制继承”即可。

配置项目级的提交规则

CodeArts Repo支持为代码的提交建立校验、限制规则，以确保代码质量。在代码托管首页，进入项目首页，选择“设置” > “策略设置” > “提交规则”，单击“新建提交规则”，参数填写请参见表格**表14-2**。

表 14-2 项目级提交规则的参数填写

参数名	参数说明
规则名称	必填参数，自定义规则名称。
分支规则	必填参数，需要输入完整规则名或创建一个正则表达式。需要对输入进行校验，包括分支名的校验和正则表达式校验。

参数名	参数说明
提交规则	非必填。 • **提交信息匹配规则**：提交信息默认为空，不会对提交信息校验，任何提交信息都可以提交。若符合正则匹配，则允许提交。您也可以设置在提交信息中必须包含工作项单号，实现代码的E2E追溯，限制500个字符。 • **提交信息负面匹配规则：**提交信息负面匹配规则默认为空，不会对提交信息校验，任何提交信息都可以提交。若符合正则匹配，则不允许提交。容限制500个字符。 • **提交人**：提交人默认为空，不会对提交人校验，任何人都可以提交，限制200个字符。 提交人可通过“git config -l”查看user.name的值，并通过“git config --global user.name”设置user.name的值。 例如： 设置提交人规则：([a-z][A-Z]{3})([0-9]{1,9}) • **提交人邮箱地址**：提交人邮箱地址默认为空，不会对提交人邮箱地址校验，任何邮箱地址都可以提交，限制200个字符。 提交人可通过“git config -l”查看user.email的值，并通过“git config --global user.email”设置邮箱。 例如： 设置提交人邮箱规则：@my-company.com$
文件基本属性规则	非必填。 • **禁止提交的文件名称**：禁止提交的文件名称规则默认为空，不会对文件名校验，任何文件都可以提交，建议正则编写时使用规范的正则语句进行匹配，文件名禁用规则处默认会根据规则校验文件所属路径，限制2000个字符。 例如： 设置禁止提交的文件名称规则：(\.jar|\.exe)$ • **单文件大小限制(MB)**：文件大小上限默认显示为50，表示添加或更新文件大小超过50MB，推送将被拒绝。 **说明** 创建仓库时默认的提交规则（default）中的单文件大小限制为50MB，新建提交规则时单文件大小限制默认推荐50MB，最大不可以超过200MB。

参数名	参数说明
二进制规则	非必填。 二进制规则默认不勾选，默认勾选“禁止禁止新增二进制文件（对特权用户无效）”。**“允许修改二进制文件”**勾选后，提交文件为modify状态的二进制文件不会拦截，可直接上传。二进制文件可以直接删除，不会进行二进制检查。 ● 禁止新增二进制文件（对特权用户无效）。 ● 允许修改二进制文件（对特权用户无效）。 ● 二进制文件白名单（可直接入库的文件，限制2000个字符）。 ● 特权用户（特权用户上限为50人）。 **说明** 如果特权用户已经不是仓库成员，单击保存会提示“特权用户校验失败”，需要将非仓库成员的特权用户移除才能保存成功。
规则生效时间	非必填。 在生效日期之后创建的所有提交都必须与hook设置相匹配才能被推送。如果此字段为空，则无论提交日期如何，都将检查所有提交。

表 14-3 常见正则表达式示例

规则	示例
单个a或b或c字符	[abc]
非a或b或c的字符	[^abc]
在a到z范围内的小写英语字母字符	[a-z]
在a到z范围外的字符	[^a-z]
在a到z或A到Z范围内的大小写英语字母字符	[a-zA-Z]
任意单个字符	.
选择 - 匹配 a 或 b	a|b
任意空白字符	\s
非空白字符	\S
阿拉伯数字字符	\d

规则	示例
非阿拉伯数字字符	\D
字母、数字或下划线的字符	\w
非字母、数字或下划线的字符	\W
匹配括号中的内容（不捕获）	(?:...)
匹配并捕获括号中的内容	(...)
零个或一个a	a?
零个或更多a	a*
一个及以上a	a+
三个a	a{3}
三个a以上	a{3,}
3到6个a	a{3,6}
文本开头	^
文本末尾	$
单词边界	\b
非单词边界	\B
换行符	\n
回车符	\r
制表符	\t
空字符	\0

配置项目级的合并请求规则

合并请求规则包含三个部分：合入机制、合入条件、MR设置和合并模式。

表 14-4 合入机制的参数说明

参数	说明
合入机制	必填参数。包含两个选项： ● **打分机制：**包含代码检视，以打分为基础，可设置最低合入分值，分值范围为0~5分。只有分数和必选评审达到门禁条件时，代码才可以合入，勾选打分机制时需设置最低分值。 ● **审核机制：**包含代码检视和合并审核两个步骤，以通过人数为基础，只有审核通过的人数达到门禁条件时，代码才可以合入。 **说明** ● 合并请求默认为“审核机制”，可手动切换为“打分机制”。 ● 修改合入机制后，会改变合并请求的工作流，但之前创建的合并请求仍保留之前的合入机制。

表 14-5 合入条件参数说明

参数	说明
合入条件	非必填参数。包括两个选项： ● 勾选“评审问题全部解决才能合入”，如果评审意见被勾选为“这是一个需要被解决的问题”，则合入条件会提示“存在未解决的评审意见”且“合入”按钮置灰；如果只是一个普通的评审意见，则不存在“已解决”开关，也不会被合入条件拦截。 ● 如果勾选“必须与CodeArts Req关联”，被关联的所有E2E单号校验必须通过；一个MR只能关联一个单号；可添加多个分支配置合并请求策略，支持手动输入通配符匹配，按回车确认，如：*-stable或production/*。

表 14-6 MR 设置参数说明

参数	说明
禁止合入自己创建的合并请求	勾选后，您在查看自己创建的MR时，“合入”按钮置灰，表示自己无法合入代码，需要找其他有合入权限的人合入。
禁止审核自己创建的合并请求	勾选后，您在查看自己创建的MR时，**“审核”**按钮置灰，自己无法审核，需要找其他有审核权限的人审核。
禁止检视自己创建的合并请求	勾选后，您在查看自己创建的MR时，**“检视”**按钮置灰，自己无法检视，需要找其他有检视权限的人检视。
允许仓库管理员强制合入	项目创建者和管理员有强制合入的权限，当合入条件不满足，也可通过**“强行合并”**按钮合入MR。
允许合并请求合并后继续做代码检视和评论	勾选后，已合入MR可继续做代码检视、评论。
是否将自动合并的MR状态标记为关闭状态（如果B MR中的所有commits都包含在A MR中，那么当A MR合并后，则B MR会自动合并。默认B MR会标记为merged状态，可以通过该选项控制将B MR标记为Closed状态）	● 未勾选时，自动合并的MR被标记为已合并。 ● 勾选后，自动合并的MR的状态将会标记为关闭状态。
不能重新打开一个已经关闭的合并请求	勾选后，当分支合并请求已经关闭后，不能将其重新置回“开启”状态，右上方的**“重开”**按钮将隐藏。 此设置一般用于流程管控，使历史评审不会被篡改。
合并请求合入后，默认删除源分支	合并成功后，源分支将被删除。 ● 已经设置成保护分支的源分支不会被删除。 ● 此设置对历史合入请求，不会生效，不必担心启用此设置会丢失分支。
禁止Squash合并	勾选后，**“Squash合并”**按钮被禁止，且合并请求中无该功能使用入口。

参数	说明
新建合并请求，默认开启Squash合并	Squash合并是指Git在做两个分支间的合并时，会把被合并分支上的所有变更“压缩（squash）”成一个提交，追加到当前分支的后面作为“合并提交”（merge commit），可以使分支变得简洁。Squash合并和普通Merge合并唯一的区别体现在提交历史上：对于普通Merge而言，在当前分支上的合并提交通常会有两个提交信息；而Squash Merge只有一个提交信息。

配置项目级的成员同步

进入项目首页，选择左侧导航栏“代码”>“代码托管”，选择“设置” > “安全设置” > “成员同步”。

开启“同步项目成员”后，勾选要同步的角色，开启后，自动同步所选角色项目成员至代码组及仓库，项目经理不依赖开关始终同步，可单击刷新按钮触发一次全量同步。

当项目成员有新增、删除或者修改，实现自动同步。

历史项目的“同步项目成员”默认关闭，新建的项目默认开启此功能，且默认勾选“项目经理”、“Committer”和“开发人员”。

15 提交代码到 Repo 并创建合并请求

15.1 设置代码仓库级的合并请求规则

合并请求配置是指代码合入条件、合入模式的配置，且项目级的合入请求规则可继承到代码仓库、代码组。

您可以勾选“继承项目设置”，自动继承并使用项目下设置且不支持更改。您也可以进入要配置的代码仓库首页，选择“设置”>“策略设置”>“合并请求”。合并请求有两种机制，打分机制和审核机制，两种模式区别如下：

- 打分机制，该模式仅包含代码检视，以打分为基础，只有分数达到门禁条件时，代码才可以合入。
- 审核机制，该模式包含代码检视和合并审核两个步骤，以通过人数为基础，只有检视和审核通过的人数达到门禁条件时，代码才可以合入。

  当您选择合入机制后，参考下表填写其余参数，且该配置对整个代码仓库生效。

表 15-1 设置合入条件、合入模式的参数表

参数名称	参数解释
合入条件	此参数非必填，共2个选项： ● 评审问题全部解决才能合入。勾选后，如果评审意见被勾选为“这是一个需要被解决的问题”，则合入条件会提示“存在未解决的评审意见”且“合入”按钮置灰；如果只是一个普通的评审意见，则不存在“已解决”开关，也不会被合入条件拦截。 ● 必须与CodeArts Req关联。包含如下3个子选项： 1. 只能关联一个单号。勾选后，一个MR只能关联一个单号。 2. 所有E2E单号校验必须通过。勾选后，被关联的所有E2E单号校验必须通过。 3. 选择分支配置合并请求策略。可添加多个分支配置合并请求策略，支持手动输入通配符匹配，按回车确认，如：*-stable或production/*。

参数名称	参数解释
MR设置	该参数非必填。包含如下选项： ● 禁止合入自己创建的合并请求。勾选后，您在查看自己创建的MR时，“合入”按钮置灰，自己无法合入，需要找其他有合入权限的人合入。 ● 禁止审核自己创建的合并请求。 ● 禁止检视自己创建的合并请求。 ● 允许仓库管理员及项目经理强制合入。 ● 允许合并请求合并或关闭后继续做代码检视和评论。 ● 是否将自动合并的MR状态标记为关闭状态。如果A MR中的包含在A MR中所有Commits，那么当A合并后，则B MR会自动合并。默认B MR会标记为合并状态，可以通过该选项控制将B MR标记为关闭状态。 ● 不能重新打开一个已经关闭的合并请求。默认打开，您可以根据自己的需要打开或关闭。 ● 新建合并请求，默认开启合并后删除源分支。 ● 禁止Squash合并（合入MR时禁止Squash合并）。 ● 新建合并请求，默认开启Squash合并。

参数名称	参数解释
合并模式	此参数必填，共3个选项： • 通过Merge Commit合并。勾选后，每次合并操作都会产生一个merge commit点，只要没有检测到冲突就能够执行合并操作。即不管基线点是不是最新的点，无冲突就可以合并。 – Squash合并不产生Merge节点：勾选后，squash合并不会产生merge节点。 – 使用MR合入者生成Merge Commit ： 勾选后，可用于记录Commit信息。 使用MR创建者生成Merge Commit： 勾选后，可用于记录Commit信息。 • 通过Merge commit 合并(记录半线性历史)。勾选后，每次合并操作会记录一个merge commit提交，但是与“通过Merge commit合并”不同，必须基于目标分支最新的commit提交点进行提交，否则会提示开发者进行rebase操作。这种合并模式下可以非常确定一点，如果merge request能够正确构建，合并完成后目标分支也能够正确构建。 • Fast-forward 合并。勾选后，每次合并操作不会记录一个merge commit提交，且必须基于目标分支最新的commit提交点进行提交，否则会提示开发者进行rebase操作。

设置分支策略

如果您在上述选择“合入机制”为“审核机制”，并且想为某个分支配置合并策略，您可以进入要配置的代码仓库首页，选择“设置”>“策略设置”>“合并请求”，单击“新建分支策略”，参考下表填写参数。

单击“新建分支策略”按钮，可以为指定分支或该仓库下的全部分支设置合入策略。

表 15-2 新建分支策略参数说明

参数	说明
分支	该参数必填。下拉框选择您想要设置的分支，支持选择全部分支。

参数	说明
最小检视人数	该参数必填。默认为0，表示无需检视人检视通过，也可通过检视门禁。
最小审核人数	该参数必填。默认为0，表示无需审核人审核通过，也可通过审核门禁。
重置审核门禁	该参数非必填。默认勾选，表示当重新推送代码到MR的源分支时，将MR审核门禁重置。
重置检视门禁	该参数非必填。默认勾选，表示当重新推送代码到MR的源分支时，将MR检视门禁重置。
仅能从以下审核人/检视人中追加审核人/检视人	该参数非必填。勾选后，可配置**“追加审核人”**名单与**“追加检视人”**名单，当您想在**“审核人”**与**“检视人”**的必选名单外追加成员时，只允许从**“追加审核人”**名单与**“追加检视人”**名单中追加成员。
开启流水线门禁	该参数非必填。勾选后，合并前需要满足流水线门禁都通过的条件，将CI融入代码开发流程。
合并人	该参数非必填。可配置**必选合并人名单**，在新建合并请求时，该名单将自动同步至合并请求中。
审核人	该参数非必填。可配置**必选审核人名单**，在新建合并请求时，该名单将自动同步至合并请求中。
检视人	该参数非必填。可配置**必选检视人名单**，在新建合并请求时，该名单将自动同步至合并请求中。

说明

分支策略优先级示例如下：

- 假设在仓库下有A策略与B策略，它们配置的分支相同，则系统默认使用最新创建的分支策略。
- 假设在仓库下有A策略与B策略，A策略配置的分支为a分支与b分支，B策略配置的分支为a分支，在发起目标分支为a分支的合并请求时，系统默认使用B策略。

在审核机制下未设置分支策略，则在发起合并请求时使用默认分支策略，该分支策略支持编辑、查看但不可删除，策略配置如下：

- **分支**：*，默认全部分支且不可修改。
- **最小检视人数**：默认为 0。
- **最小审核人数**：默认为 0。
- **重置审核门禁**：默认勾选。
- **重置检视门禁**：默认勾选。
- **仅能从以下审核/检视人中追加审核人/检视人**：默认不勾选。
- **开启流水线门禁**：默认不勾选。
- **合并人**：默认为空。
- **审核人**：默认为空。
- **检视人**：默认为空。

  必选名单举例：

  - **最小审核人数**为2人，如果**必选审核人名单**为空，**追加审核人名单**2人均审核通过，审核门禁通过。
  - **最小审核人数**为2人，如果**必选审核人名单**非空，则该名单内至少一人审核通过，审核门禁才可通过。

15.2 配置 CodeArts Repo 的合并请求通知设置

CodeArts Repo支持通过邮件或者企业微信的方式推送关于代码仓库和合并请求通知，您可以根据需要打开其中一种通知或者同时打开两种通知，代码仓库内的成员可以查看该页面，仅拥有代码仓库“设置”权限的角色可以配置代码仓库的通知设置。

- 配置邮件通知设置请参考**配置代码仓库的邮件通知设置**。
- 配置代码仓库的企业微信通知设置请参考**表15-4**。

配置代码仓库的邮件通知设置

表 15-3 邮件通知设置的参数说明

参数	说明
仓库	该参数非必填。根据您想收到的邮件通知设置即可，包含四个选项，默认勾选“冻结仓库”和“关闭仓库”，且不可更改，如果仓库出现冻结或者关闭，邮件将通知仓库所有者和项目管理员。另外两个选项如下，并且您可以选择想要邮件通知的对象： ● 如果勾选“删除仓库”，表示有成员删除仓库时，系统将通过邮件通知的方式告知您。 ● 如果勾选“容量预警”，表示超过设置的容量阈值，系统将通过邮件通知的方式告知您，并且您可以下拉选择阈值：60%、80%和90%。
合并请求	该参数非必填，根据您的需要勾选对应选项即可，包含如下选项： ● 开启合并请求。表示有合并请求开启时(包括新建和重开合并请求)，会邮件通知到您勾选的角色，默认勾选的角色：评审人、审核人、检视人和合并人。 ● 更新合并请求。表示更新合并请求关联分支的代码时，会推送更新邮件，默认勾选的角色：评审人、审核人和检视人。 ● 合并合并请求。表示合并请求时，会推送邮件，默认勾选的角色：MR创建人。还可以勾选“合并人”。 ● 检视合并请求。表示会推送邮件通知检视合并请求，默认勾选角色：MR创建人。 ● 审核合并请求。表示会推送邮件通知审核合并请求，默认勾选角色：MR创建人。 ● 新建评审意见。表示会将新建的评审意见推送给选中角色，默认勾选角色：MR创建人。 ● 解决评审意见。表示会推送邮件给选中角色，让其解决评审意见，默认勾选角色：MR创建人。

说明

如果在CodeArts Repo已打开邮件通知设置，但仍未收到相关邮件通知，请前往**CodeArts的消息设置**，检查邮箱配置、邮件通知是否开启。

配置代码仓库的企业微信通知设置

表 15-4 企业微信通知设置的参数说明

参数	说明
Webhook地址	该参数必填。用于识别CodeArts Repo成员组所添加机器人的Webhook地址，长度上限为500字符。
仓库	该参数非必填。根据您想收到的微信通知设置即可，包含两个个选项，默认勾选如下两个选项，并且您可以选择想要邮件通知的对象： • 如果勾选“删除仓库”，表示有成员删除仓库时，系统将通过邮件通知的方式告知您。 • 如果勾选“容量预警”，表示超过设置的容量阈值，系统将通过邮件通知的方式告知您，并且您可以下拉选择阈值：60%、80%和90%。
合并请求	该参数非必填，根据您的需要勾选对应选项即可，包含如下选项： • 合并请求状态变更。表示开启、更新或者合并请求状态时，会通过微信机器人的方式推送通知。默认勾选的状态：开启、合并。 • 合并请求检视审核。包括“检视”和“审核”两种状态。 • 合并请求评审意见。默认勾选“新建”状态，还可以根据需要，勾选是否要通知“解决”状态。

15.3 解决评审意见并合入代码

通过评审意见门禁

如果为目标仓库开启了合并请求门禁，即勾选“评审问题全部解决才能合入”。合并请求的检视人或审核人可在合并请求的“文件变更”中，将鼠标置于要提检视意见的代码行，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。

当您已解决评审意见后，在合并请求的“详情 > 评审意见”中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”，表示发起合并请求的人将所有评审意见解决，可合入该合并请求。

通过流水线门禁

如果为目标仓库开启了流水线门禁，即勾选“开启流水线门禁”。执行如下步骤通过流水线门禁：

步骤1 进入目标仓库首页。选择左侧导航栏“持续交付 > 流水线”，进入流水线服务。

步骤2 单击“新建流水线”，填写以下信息后，单击“下一步”，根据您的需求，选择目标模板。

- 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 任务创建成功后会自动跳转任务详情中的“任务编排”页签，切换到“执行计划”页签。

步骤4 开启“合并请求时触发”，根据实际情况勾选以下的一种触发事件。

- 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5 完成流水线任务其他信息配置，单击“保存”。

步骤6 返回CodeArts Repo，触发“执行计划”中已勾选的事件让代码仓库执行流水线任务即可。

----结束

进入合并请求详情页，门禁显示为“合并请求流水线门禁已通过”，表示最新commit/预合并commit成功拉起流水线。

通过 E2E 单号关联门禁

如果为目标仓库开启了E2E单号关联，即勾选“必须与CodeArts Req关联”。执行如下步骤，完成E2E单号关联：

步骤1 进入目标仓库，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2 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3 单击**“确定”**，完成E2E单号关联。

----结束

当合并请求成功关联工作项时，门禁显示为“E2E单号关联通过”。

说明

- 如果您在合并代码的时候提示单个仓库容量超过2GB，不允许合并申请，请检查是否有Git提交的缓存文件所致。
- MR关联工作项最大数量为100个。

16 管理合并请求

16.1 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。

- **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。

门禁的通过

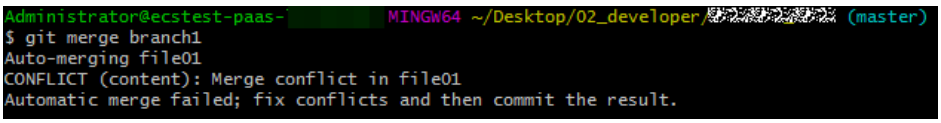
当您已解决评审意见中提出的问题后，可在合并请求的“**详情 > 评审意见**”中需要将评审意见的状态由“**未解决**”切换成“**已解决**”，此时门禁将显示为“**评审意见门禁已通过**”。

16.2 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。

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）。

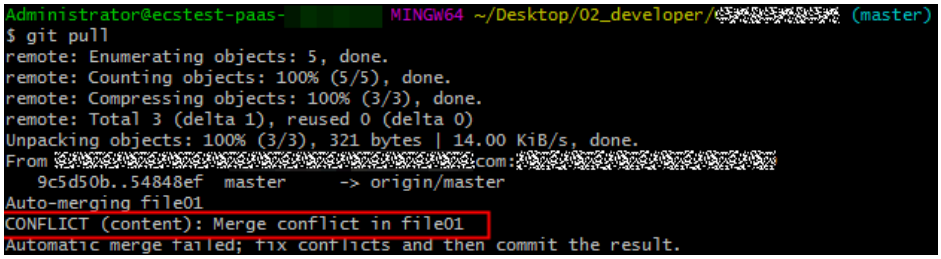
  

- 可解决合并请求的文件冲突数量不得超过50个，且单文件冲突内容的大小不得超过200KB。

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。



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  ##file01AAAAAAAAAAA
2  ##file02BBBBBBBBBBB
3  ##file03CCCCCCCCCCC
4  ##file04DDDDDDDDDDD
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  ##file01AAAAAAAAAAA
2  ##modify by 01_dev
3  ##file03CCCCCCCCCCC
4  ##file04DDDDDDDDDDD
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  ##file04DDDDDDDDDDD
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：

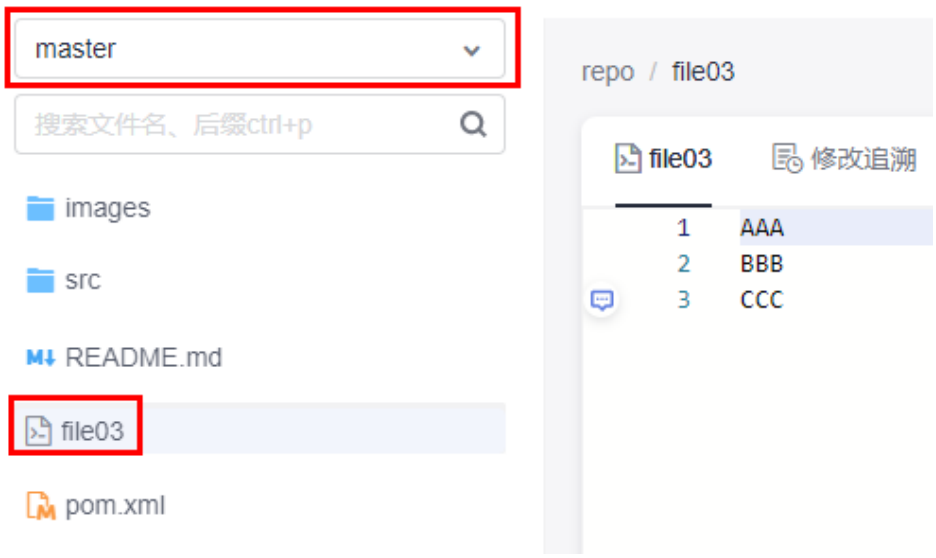
如果开发者每天都要对本地仓库进行修改，则建议在本地新建一条开发分支进行代码修改，在要提交远程仓库时，切换到master分支并将远程仓库的最新master分支内容拉取到本地，在本地进行分支合并，对产生的冲突进行修复，成功将内容合并到master分支后，再提交到远程仓库。

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。

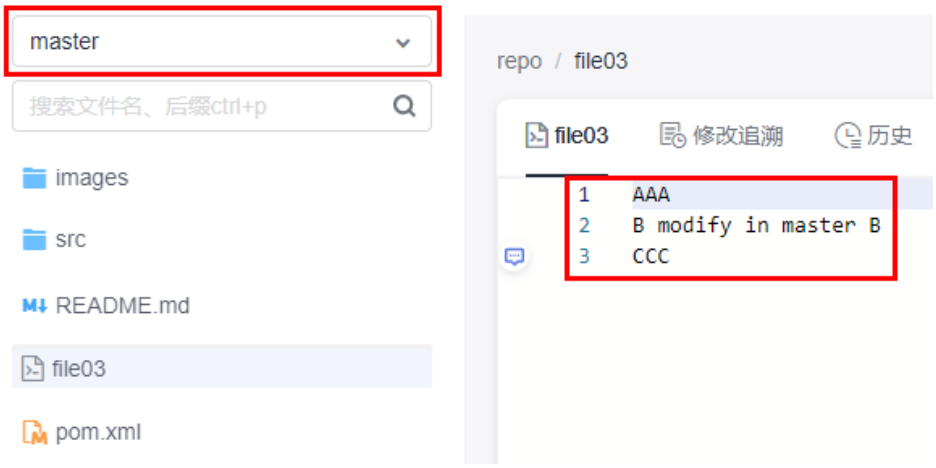
步骤2 在仓库中**新建一个文件**，在本案例中，在master分支上新建一个名为file03的文件，其内容初始编写如下。



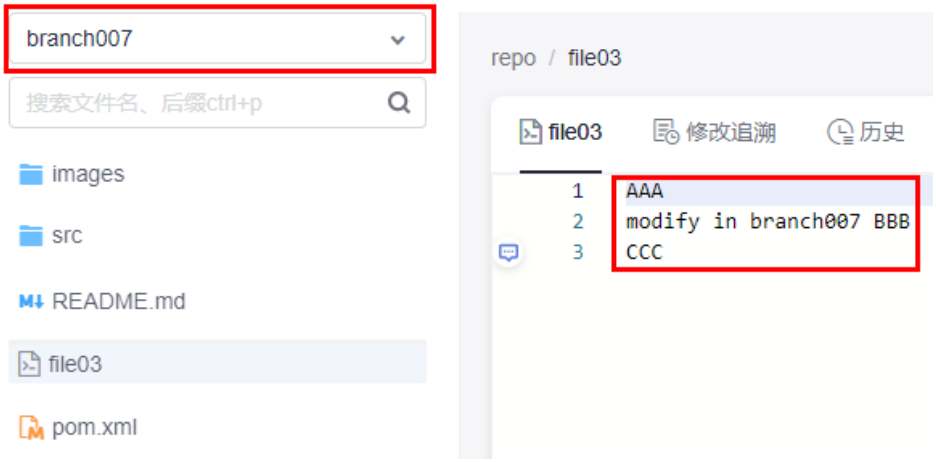
步骤3 基于master分支**新建一个分支**，在本案例中，将其命名为“branch007”。

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”按钮执行分支合并的操作，系统会提示您合并成功。

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

16.3 创建 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，以此简洁提交信息。当您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 **新建仓库并创建分支。**

仓库名称命名为“repo”，分支名称命名为“Dev”。

步骤2 Dev分支：新建两个文件并分别命名为“功能一”和“功能二”。

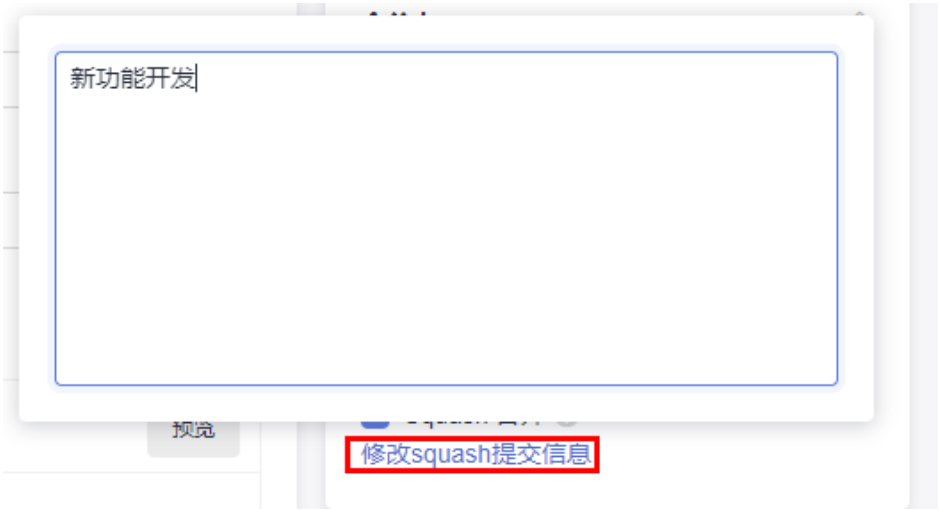
步骤3 查看开启“**Squash 合并**”前的效果。

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4 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. 完成合并请求的检视、审核后，即可合入请求。

步骤5 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

17 管理代码文件

17.1 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。

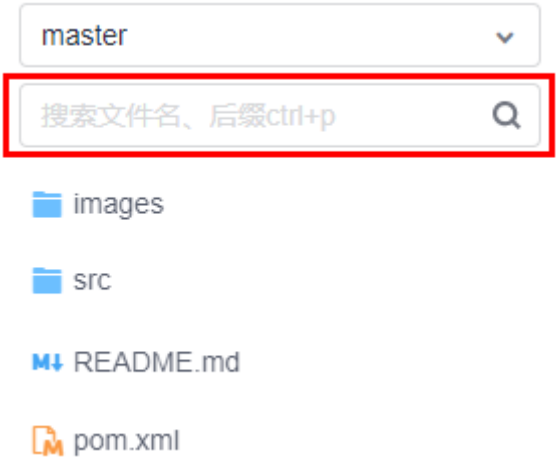
当您进入仓库详情控制台，系统将为您定位到“代码”页签下的**“文件”**子页签，在这里您可以切换到不同的分支、标签，查看对应版本中文件的情况，如下图，左侧为主分支下的文件列表，右侧为可切换的页签：仓库名称（分支或标签版本的文件详情）、历史（分支或标签版本）。

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 + 新建 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11.7-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

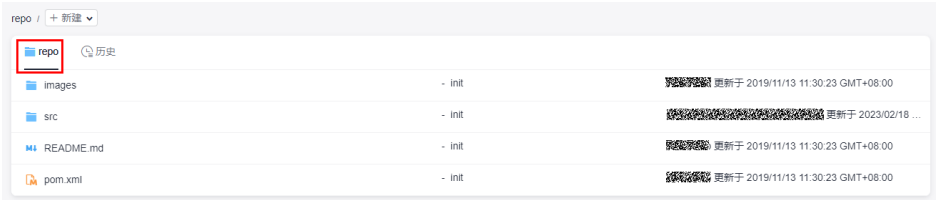
鼠标停留在文件夹名称处，单击显示的 图标，可该文件夹下进行以上操作。

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：

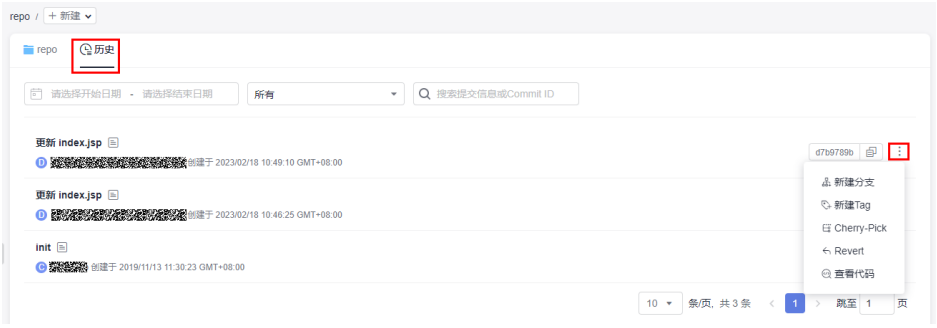
- 文件：文件或文件夹的名称。
- 提交信息：此文件或文件夹的上次提交信息（commit的-m），单击可定位到此次提交记录。
- 创建者：此文件或文件夹的上次提交创建者。
- 更新时间：此文件或文件夹的上次更新时间。

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11.7-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 17-1 界面说明

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。

界面功能	功能说明
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。

17.2 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。

17.3 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。

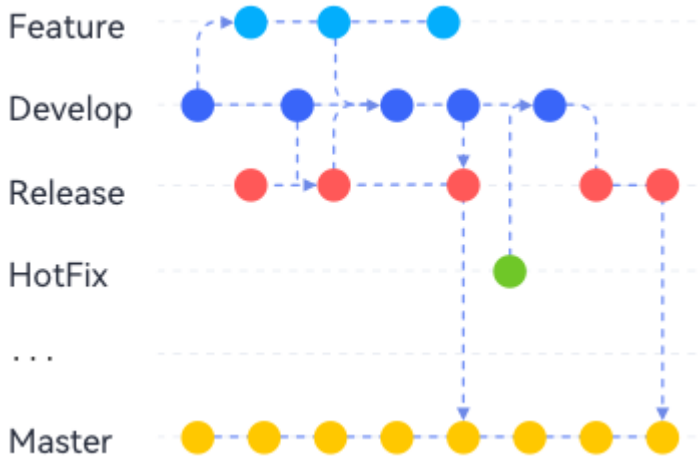


表 17-2 Git-Flow 工作模式中分支的使用建议

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，基于对应的父feature分支创建	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...
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
何时结束生命周期	-	-	其对应的特性已经验收(发布、稳定)后	-	其对应的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：

- 合并请求设置
- 默认分支管理
- 分支保护设置

关于分支的常用 Git 命令

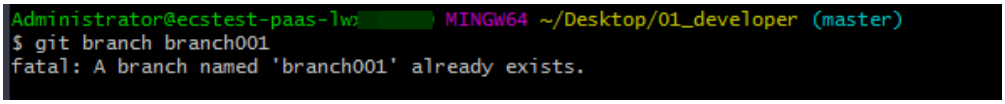
- **新建分支**

```
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。



- **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：

- **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：

- **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。

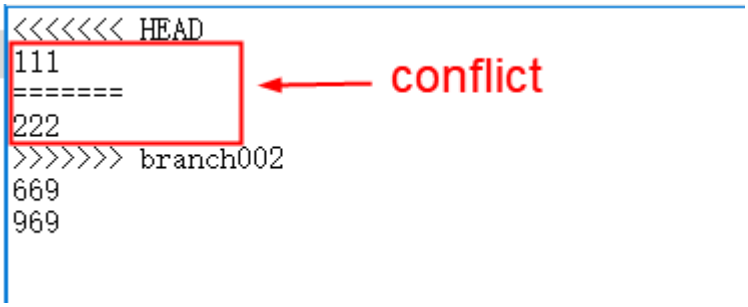
```
Administrator@ecstest-paas-lw      MINGW64 ~/Desktop/01_developer (branch001)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'HTTPSOrigin/master'.

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。




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

- **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://[redacted].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://[redacted]d.com/[redacted]639472/newmerge
remote:
To https://[redacted].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。

17.4 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。

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

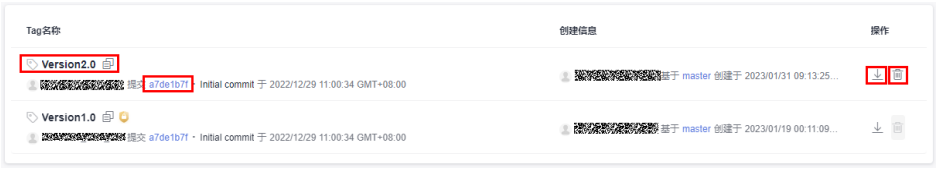
如果在描述中输入信息会生成附注标签（描述相当于 -m 后的内容），不输入则生成轻量标签。（什么是附注标签？）

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。

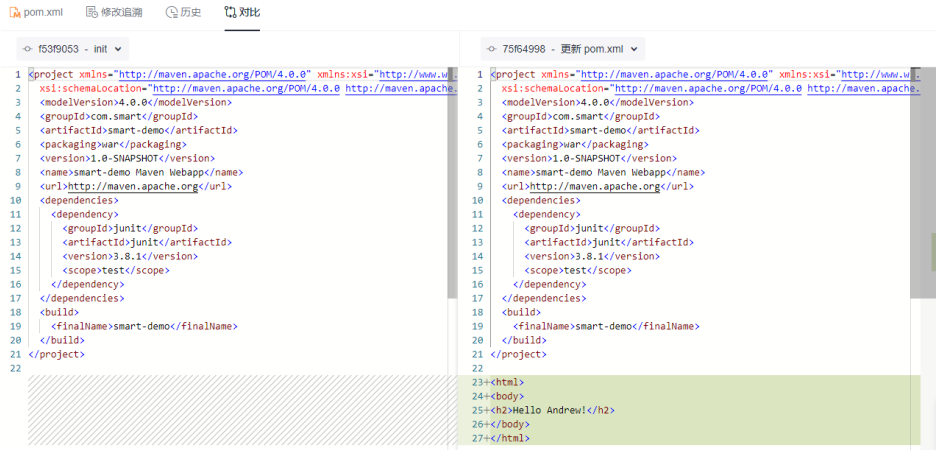


  - 单击“标签名”，跳转到该标签对应版本的文件列表。
  - 单击**“提交号”**，跳转到该次提交（commit）的详情页面。
  - 单击 ，可下载tar.gz或zip格式的被标签版本的文件包。
  - 单击  ，可以将此标签从代码托管仓库删除（想从本地删除请clone、pull或本地手动-d删除）。

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。



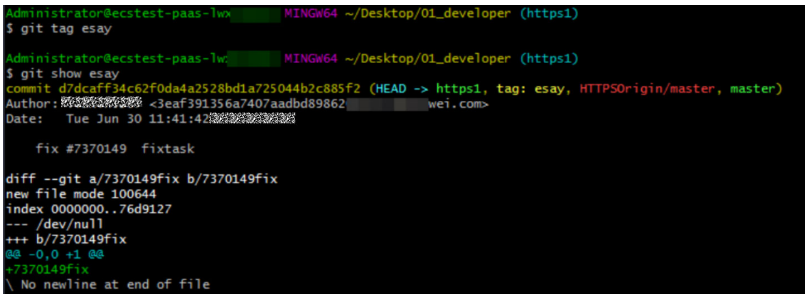
标签的分类

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>
```

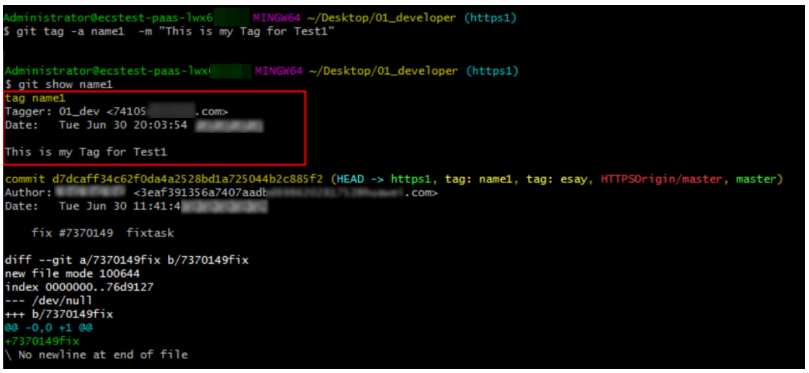
如下图是一个轻量标签被查看详情时的显示内容，可以看到它其实就是一次commit的别名。



- **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
  ```

- **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。

- **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。

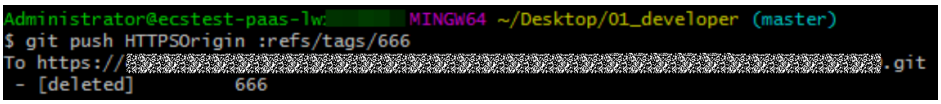
- **删除远程仓库标签**

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
```



如何使用标签找回历史版本

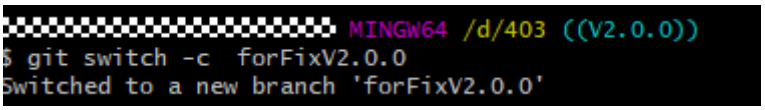
当您要查看某个标签指向版本的代码时，可以将其检出到工作区。由于被检出的版本仅隶属于标签，而不属于任何分支，因此该代码可以编辑，但是不能add、commit。您可以基于工作区新建一条分支，在此分支上修改代码，并将此分支合入主干。具体的操作步骤如下所示。

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
```

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
```

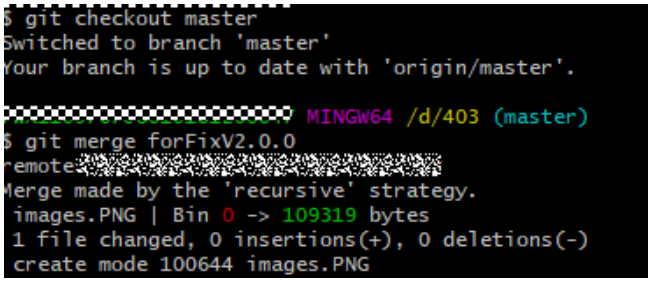


3. （可选）如果修改了新建的分支的内容，需要将修改内容提交到该分支的版本库中。

```
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
```

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
```

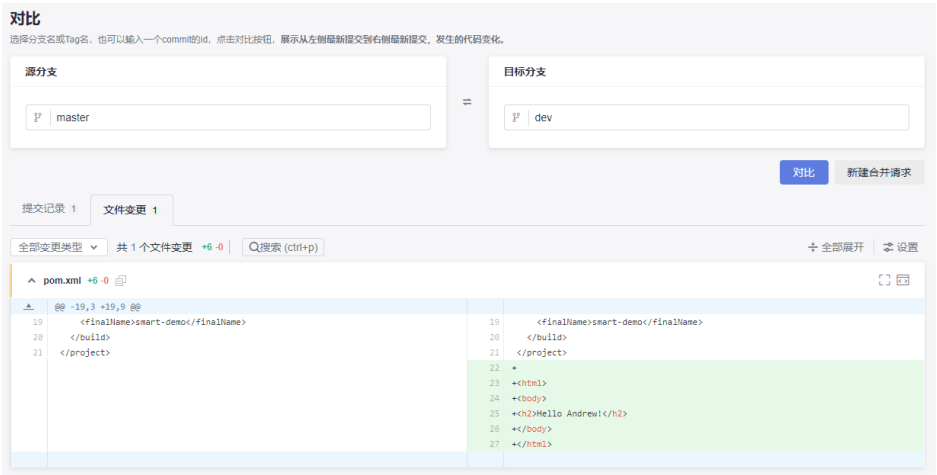


说明

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。

17.5 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。

18 CodeArts Repo 的安全管理

CodeArts Repo为保证代码仓库的安全性，支持添加IP白名单、支持更改代码仓库所有者、删除代码仓库、更改代码仓库名称、增加水印设置、锁定仓库、记录审计日志，具体可参考如下章节。且这些操作只有具有代码组或者代码仓库“设置”权限的人员可执行，代码仓库的“设置”权限可参考**配置代码仓库级的权限**。

为代码仓库配置部署密钥

为保证代码仓的安全性，有些代码仓库只支持克隆/下载，不支持合入代码等其它变更代码仓库操作，代码仓库处于只读模式，此时需要为该代码仓库配置部署密钥。配置部署密钥位于代码仓库详情中的“设置 > 安全管理 > 部署密钥”，进入部署密钥页面，单击“添加部署密钥”，本地生成SSH密钥可参考**配置SSH密钥**的步骤1~步骤3。

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有区别：前者与用户/计算机关联，后者与代码仓库关联；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代码库生效。

CodeArts Repo 的风险操作

CodeArts Repo支持更改代码仓库所有者、删除代码仓库和更改代码仓库名称，但该操作存在风险，请谨慎操作。

风险操作位于代码仓库详情中的 “设置 > 安全管理 > 风险操作” 。支持如下三个操作：

- 移交仓库所有者：可以将当前代码仓库移交给仓库内的其他人（ 不能移交给浏览者 ）。
- 删除仓库：一旦您删除该代码仓，代码仓库内所有内容都将会被永久删除。这是一个不可恢复的操作，请谨慎操作。
- 更改仓库名：更改仓库名称将导致仓库的访问和克隆地址改变，在此之前的地址将失效，请谨慎操作。

为 CodeArts Repo 的代码仓库增加水印设置

CodeArts Repo支持为代码仓库增加水印，以此保护代码仓库的知识产权。

水印设置位于代码仓库详情中的 “设置 > 安全管理 > 水印设置” 。

打开水印设置按钮，该代码仓库将展示如下的水印内容：账户+时间。

锁定 CodeArts Repo 的代码仓库

CodeArts Repo支持锁定代码仓库，以此防止任何人破坏即将发布版本的代码仓库。

锁定仓库设置位于代码仓库详情中的 “设置 > 安全管理 > 锁定仓库” 。拥有修改“设置”权限的仓库成员可以执行此操作。

打开水印设置按钮，表示锁定该代码仓库，锁定后将完全只读，任何人无法向任何分支提交代码，也不能创建评论和其他相关新增的操作。

为 CodeArts Repo 代码仓库设置 IP 白名单

CodeArts支持通过设置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，增强代码仓库的安全性。IP白名单仅对可见范围为“私有”、“项目内成员只读”和“租户内成员只读”的仓库生效。

IP白名单设置位于代码仓库详情中的 “设置 > 安全管理 > IP白名单” 。IP白名单支持IPv4和IPv6，有3种格式，如下表所示。

单击“新建IP白名单”，参考下表填写参数，如果您需要修改，单击IP白名单所在行的 即可。

表 18-1 新建 IP 白名单参数说明

参数	说明
IPv4	如果选择该参数，您指定IP、设置IP范围或者设置CIDR格式的路由，区别如下： • 指定IP，表示该IP将被添加到白名单中，例如将您的个人家庭电脑的IP添加到白名单中。 • 指定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，您可以添加一个IP白名单范围。示例：100.*.*.0 - 100.*.*.255。 • 设置CIDR格式的路由，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。
IPv6	如果选择该参数，您指定IP、设置IP范围，可参考指定IP和设置IP范围。
备注	非必填参数。

参数	说明
访问控制	非必填参数，根据您的需要，勾选对应选项即可： • 允许访问仓库：勾选该选项后，白名单内的IP才可以访问该仓库，仓库所有者不受限制。 • 允许下载代码：勾选该选项后，白名单内的IP允许在线下载、本地克隆代码。 • 允许提交代码：勾选该选项后，白名单内的IP才可以在线修改、在线上传和本地提交代码；构建工程代码化编排，yaml文件同步功能不受访问控制。

说明

如果您需要对租户下的所有代码仓库统一设置IP白名单，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，其配置规则与如上配置相同。

CodeArts Repo 记录审计日志

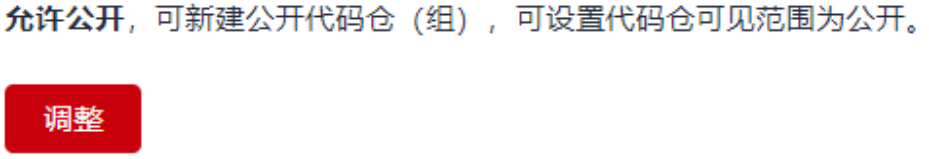
CodeArts Repo支持更改代码仓库属性，因此CodeArts Repo会将关于该代码仓的代码提交、合并请求等信息进行记录，每一条审计日志包含操作者、操作类型和操作内容，您可以根据时间段进行筛选查看。

调整仓库公开性

CodeArts Repo支持调整仓库的公开性。

在CodeArts首页，单击个人头像，选择“租户设置”，在左侧导航栏选择“代码托管 > 调整仓库公开性”，单击“调整”，可调整租户下代码仓库的公开性。

- 如果界面图如下图所示，表示可新建公开代码仓（组），可设置代码仓可见范围为公开。



- 如果界面图如下图所示，表示禁止新建公开代码仓（组），不可设置代码仓可见范围为公开。

禁止公开，禁止新建公开代码仓（组），不可设置代码仓可见范围为公开。

调整

