代码托管

最佳实践

文档版本	01
发布日期	2023-03-31

华为技术有限公司

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。

目 录

1 Git on CodeArts Repo

1.1 概述

文档目的

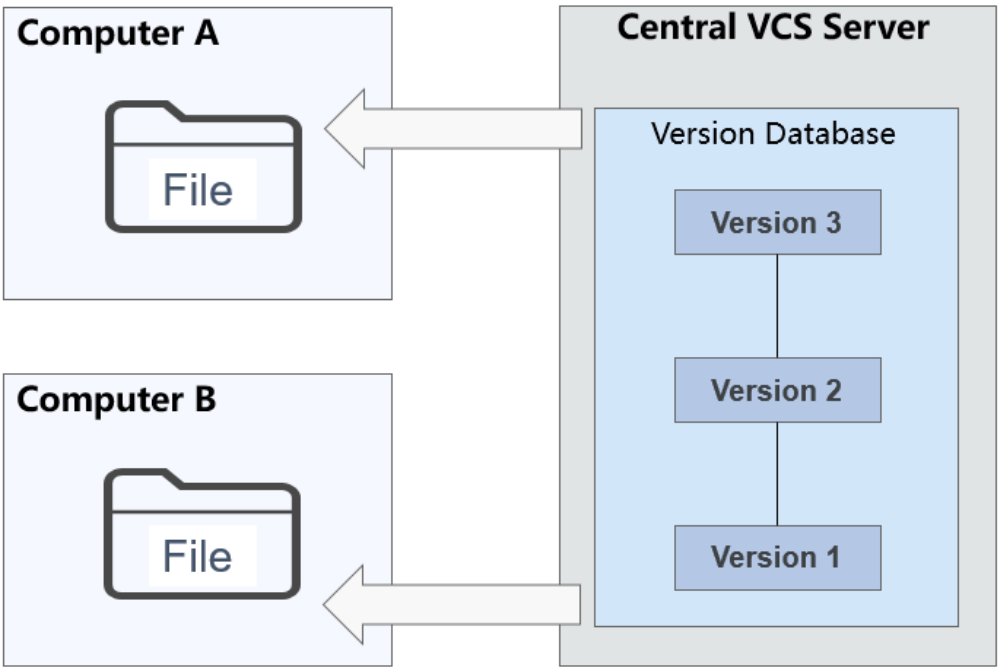
基于CodeArts实践所编写，用于帮助已经掌握或想要掌握Git的开发者，更好的应用Git，以及更好的将Git与CodeArts结合应用。

Git 概述

从狭义上来说，版本控制系统是软件项目开发过程中管理代码所有修订版本的软件，能够存储、追踪文件的修改历史，记录多个版本的开发和维护，事实上我们可以将任何对项目有帮助的文档交付版本控制系统进行管理。版本控制系统（Version Control Systems）主要分为两类，集中式和分布式。

集中式版本控制系统

集中式版本控制系统的特点是只有一台中央服务器，存放着所有研发数据，而其它客户端机器上保存的是中央服务器最新版本的文件快照，不包括项目文件的变更历史。所以，每个相关人员工作开始前，都需要从这台中央服务器同步最新版本，才能开始工作，如下图所示。



常见的集中式版本控制系统为CVS、VSS、SVN、ClearCase。

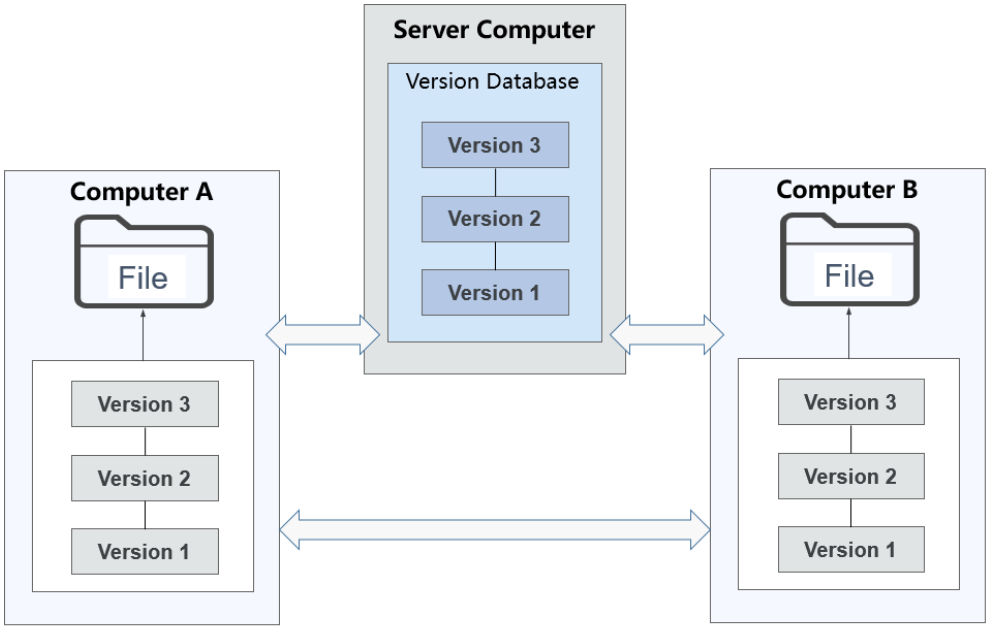
集中式版本控制系统的优点与缺点如下表所示。

表 1-1 集中式版本控制系统描述

优点	缺点
• 操作简单，使用没有难度，可轻松上手。 • 文件夹级权限控制，权限控制粒度小。 • 对客户端配置要求不高，无需存储全套代码。	• 网络环境要求高，相关人员必须联网才能工作。 • 中央服务器的单点故障影响全局，如果服务器宕机，所有人都无法工作。 • 中央服务器在没有备份的情况下，磁盘一旦被损坏，将丢失所有数据。

分布式版本控制系统

分布式版本控制系统的特点是每个客户端都是代码仓库的完整镜像，包括项目文件的变更历史。所有数据分布的存储在每个客户端，不存在中央服务器。可能有人会问，我们公司使用Git分布式存储工具，也有“中央服务器”啊？其实，这个所谓的“中央服务器”仅仅是用来方便管理多人协作，任何一台客户端都可以胜任它的工作，它和所有客户端没有本质区别，如下图所示。



常见的分布式版本控制系统为Git、Mercurial、Bazaar、Bitkeeper。

分布式版本控制系统的优点与缺点如下表所示。

表 1-2 分布式版本控制系统描述

优点	缺点
• 版本库本地化，版本库的完整克隆，包括标签、分支、版本记录等。 • 支持离线提交，适合跨地域协同开发。 • 分支切换快速高效，创建和销毁分支廉价。	• 学习成本高，不容易上手。 • 只能针对整个仓库创建分支，无法根据目录建立层次性的分支。

1.2 CodeArts Repo 云端操作

准备工作

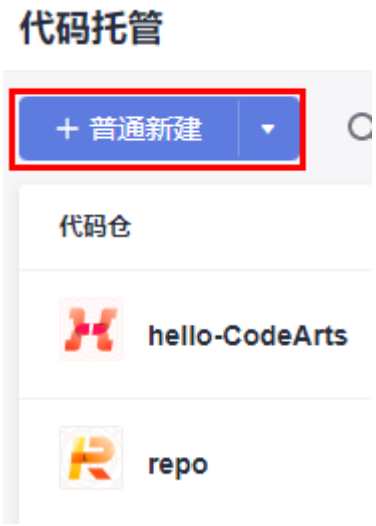
- 成为代码托管（CodeArts Repo）用户
- **已有Git客户端**
- **已创建好的项目**

云端仓库功能

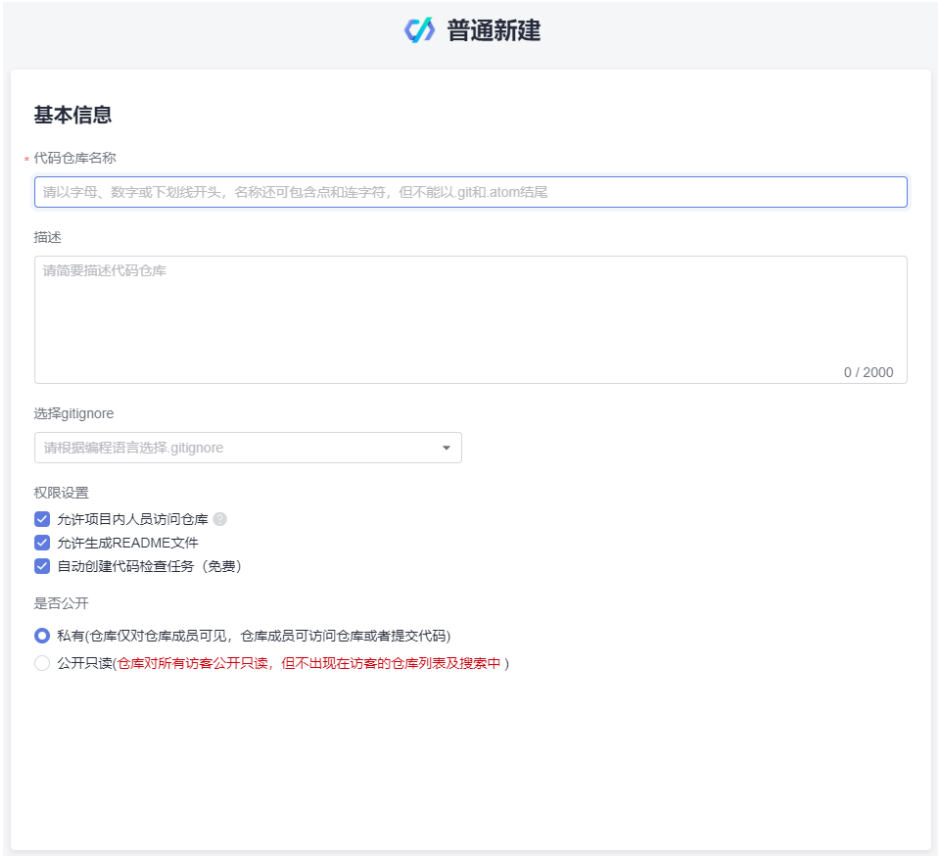
云端仓库功能支持新建仓库、仓库克隆、分支管理、标签管理、提交代码、拉取代码、推送代码、代码阅读、在线修改、仓库成员管理、密钥管理等，更多仓库功能介绍请参见**产品概述**。

新建空仓库

1. 在目标项目下的代码托管服务中，单击“普通新建”按钮，如下图所示。



2. 填写仓库的基本信息，下图所示。



3. 单击“确定”按钮，完成仓库新建，跳转到仓库列表。

设置 SSH 密钥/HTTPS 密码

后续需要在本地客户端进行代码仓库的克隆/推送，SSH密钥和HTTPS密码是客户端和服务端交互的凭证，需要先对它们进行设置。

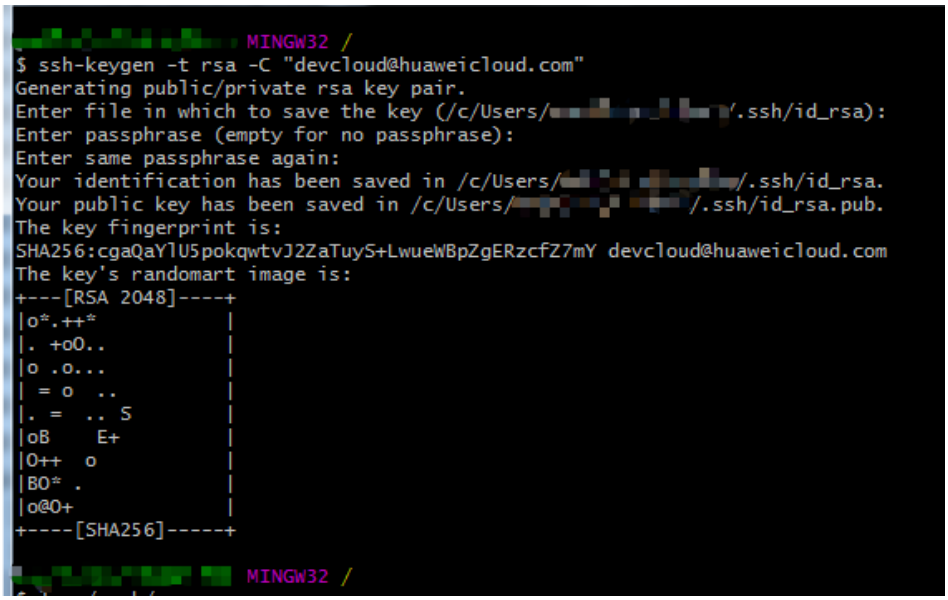
设置SSH密钥

SSH密钥是使用SSH协议和代码托管服务端交互的凭证，如果您使用windows下的Git Bash客户端并在其中已经生成，此步骤可以略过。

步骤1 打开Git客户端（Git Bash或linux的命令行窗口），输入以下命令行：

```
ssh-keygen -t rsa -C "<您的邮箱>"
```

然后输入3个回车（Enter键）即可，生成的SSH秘钥对默认在“~/.ssh/id_rsa、~/.ssh/id_rsa.pub”位置，如下图所示。

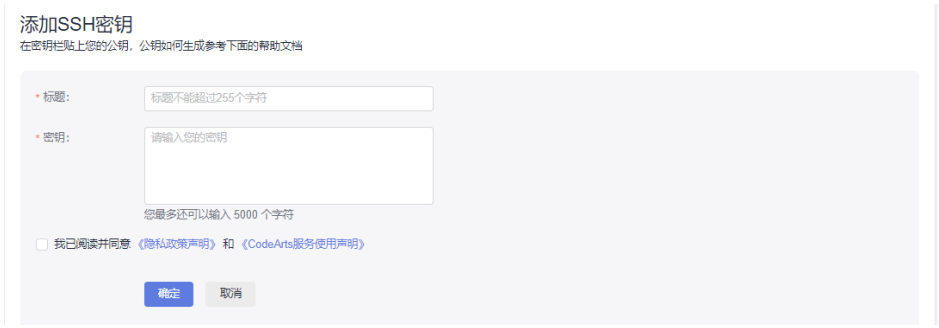


步骤2 添加SSH密钥到代码托管服务端：

打开Git客户端（Git Bash或linux的命令行窗口），将SSH密钥“~/.ssh/id_rsa.pub”的内容打印出来。

步骤3 复制上述的SSH密钥内容，登录您的代码托管服务仓库列表页，单击右上角昵称，单击“个人设置 > SSH密钥管理”，进入页面。

步骤4 在“SSH密钥管理”页面，单击“添加SSH密钥”，弹出“添加SSH密钥”页面，填写下图中信息，单击**“确定”**，页面会提示您操作成功。



您已经设置好了SSH密钥，您可以继续设置HTTPS密码。

----结束

设置HTTPS密码

HTTPS密码是使用HTTPS协议和代码托管服务端交互的凭证，设置步骤如下：

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

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

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。

----结束

1.3 Git 本地研发场景

背景介绍

在CodeArts云端创建一个README文件的空仓库，然后架构师或者项目负责人需要把本地框架代码推送到这个空仓库，最后，其他开发人员将云端架构代码克隆到本地，进行增量应用开发。

说明

- Git代码传输支持SSH和HTTPS两种传输协议，本节基于SSH传输协议进行的操作。
- 如果想使用HTTPS方式，直接下载HTTPS密码，当克隆、推送代码时直接输入HTTPS用户名密码即可。
- 同一仓库SSH和HTTPS的地址不同。

推送架构代码

1. 打开本地框架代码，确保根目录名与云端创建的代码仓库名一致，在根目录下右键打开**Git bash**终端。
2. 推送本地代码到云端。

   在当前**Git Bash**终端依次输入如下命令：

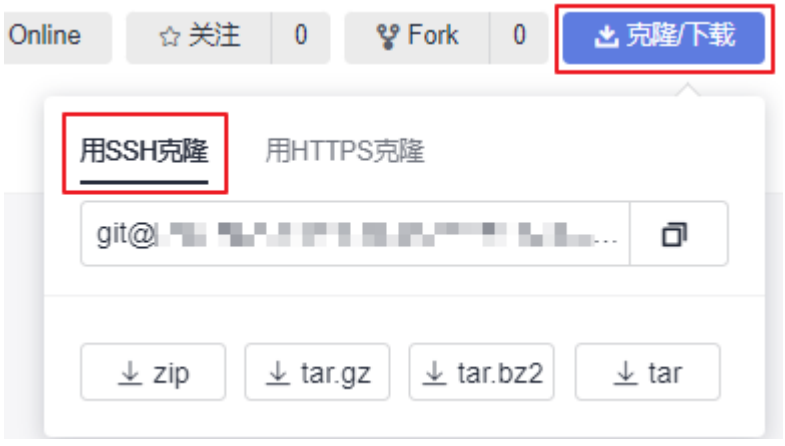
   a. 初始化本地代码仓库，执行该命令后，在“D:/code/repo1/”下多了一个“.git”文件夹。

   ```
   $ git init
   ```

   b. 关联云端代码仓库。

   ```
   $ git remote add origin repoUrl
   ```

   仓库地址如下图进入仓库详情页，单击”**克隆/下载**“，所示单击红色方框处获取。

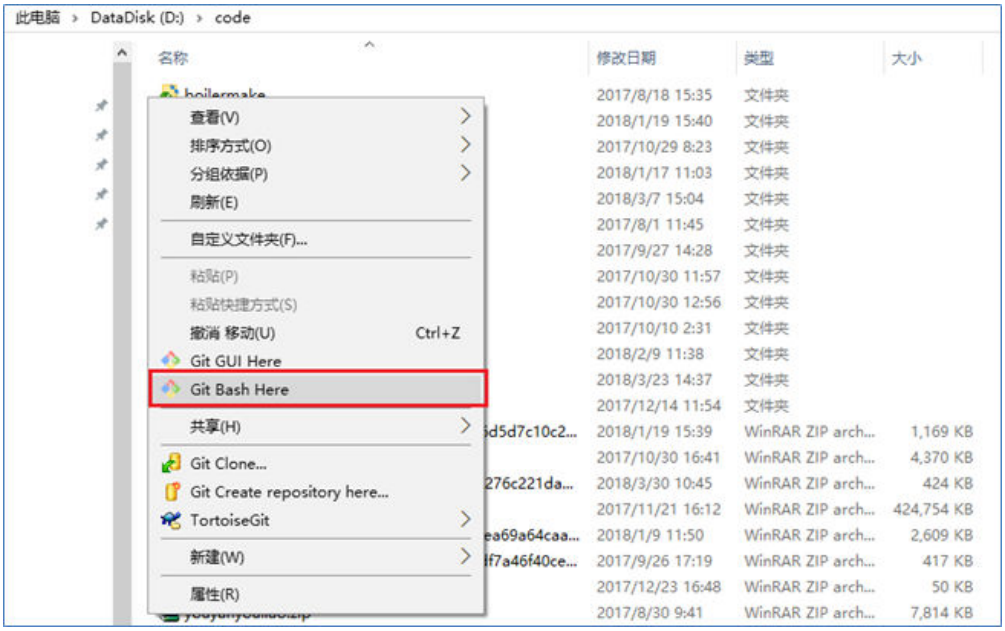
   c. 推送代码到云仓库。

   ```
   $ git  add .
   $ git  commit  -m "init project"
   $ git  branch --set-upstream-to=origin/master master
   $ git  pull  --rebase
   $ git  push
   ```

克隆代码

开发人员在本地准备克隆云端架构代码。

1. 在准备把代码克隆到的目标文件夹下，右键打开**Git bash**终端，如下图所示。



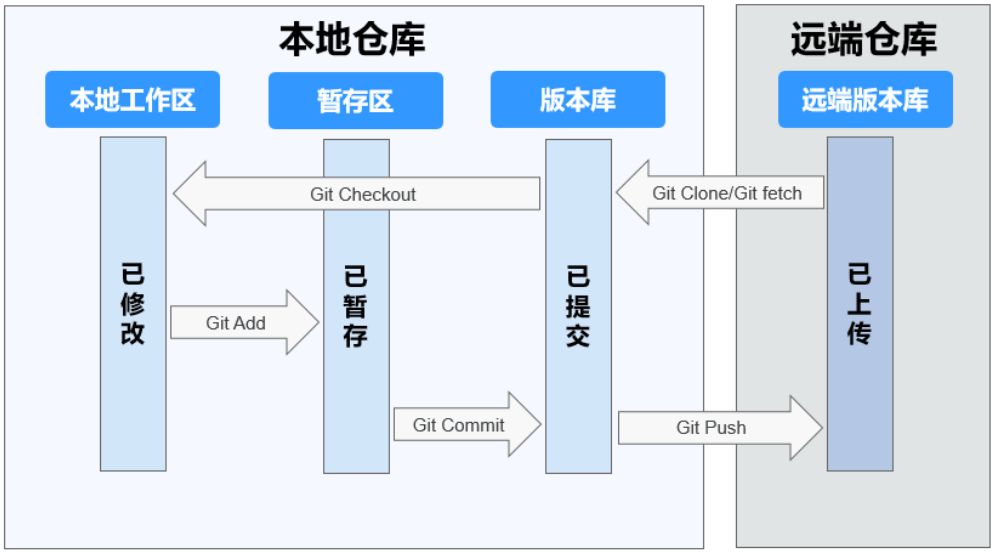
2. 克隆仓库，URL地址如下图所示单击红色方框处获取。

```
$ git clone repoUrl  //将代码从远端仓库clone到本地
```

代码提交

一次修改被成功提交到远端仓库会历经四个阶段：“1本地工作区 > 2缓存区 > 3版本库 > 4远端版本库”。

通过执行相应的Git命令，文件在这四个区域跳转，并呈现不同的状态，如下图所示。



主要涉及如下三步操作：

1. **#git add/rm filename** *//将新增、修改或者删除的文件增加到暂存区*
2. **#git commit –m “commit message”** *//将已暂存的文件提交到本地仓库*
3. **#git push** *//将本地代码仓库修改推送到远端仓库*

分支操作

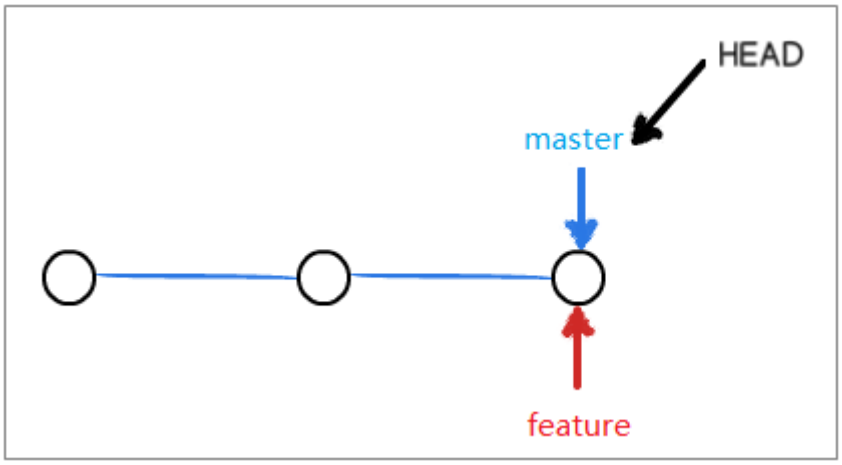
- 新建分支

  Git新建分支的本质就是创建一个指向最后一次提交的可变指针，所以，Git分支的创建不是复制版本库的内容，仅仅是新建了一个指针，它以40个字符长度SHA-1字串形式保存在文件中。

  ```
  #git branch branchName commitID
  ```

  基于commitID即某一个全球版本号拉出新分支，如果没有commitID则基于当前分支的HEAD拉出新分支。

  例如，新建feature分支，执行的命令为**git branch feature**，如下图所示。

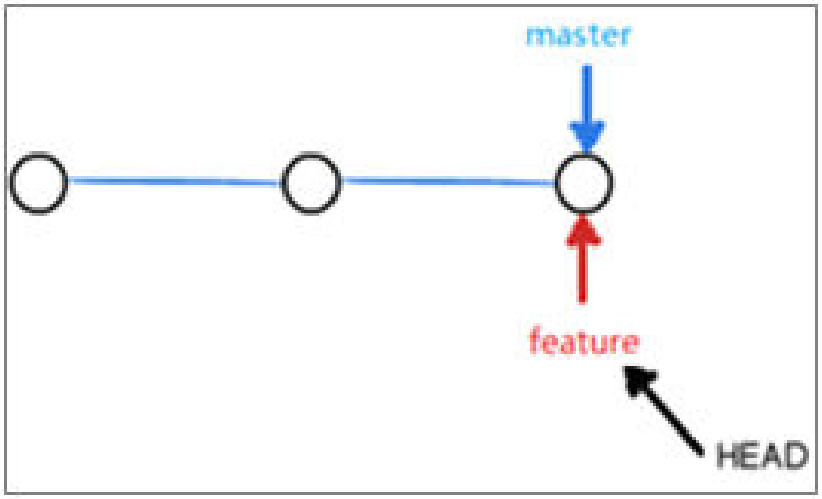


- 切换分支

  命令如下。

  ```
  #git checkout branchName
  ```

  例如，切换到feature分支，执行的命令为**git checkout feature**，如下图所示。

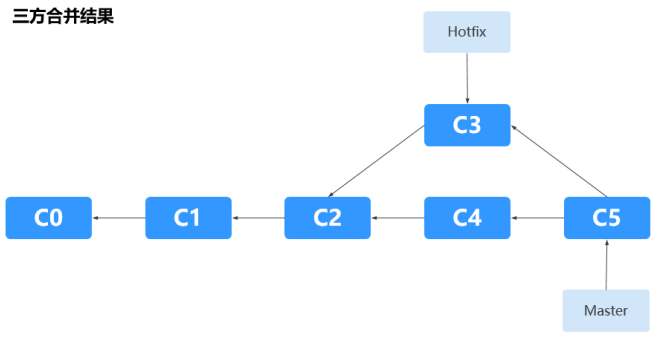
- 分支合并

  无论哪种工作流都会涉及到分支合并（把一个分支中的修改整合到当前分支），主要有两种方法：三方合并（merge）和衍合（rebase）。通过对同一种场景进行不同操作体会两种合并方法的区别。

  场景：master分支新增了C4节点， hotfix分支新增了C3节点，现将hotfix分支合并到master分支：

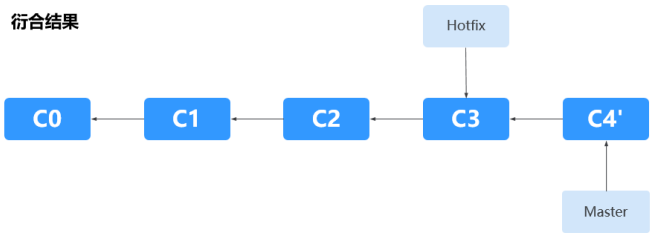
  a. 三方包括hotfix新增节点C3，master新增节点C4，以及两者的共同祖先节点C2。这种合并操作简单，但新增合并节点C5，形成了环形，版本记录可读性差，如下图所示。

  ```
  #git checkout master
  #git merge hotfix
  ```

  b. 衍合先将master分支新增节点C4以补丁形式保存在.git/rebase目录中，然后同步hotfix分支最新代码，再应用补丁C4’，如下图所示。

  ```
  #git checkout master
  #git rebase hotfix
  ```

- 冲突解决
  a. 场景一：两个合并分支修改了同一行代码

```
$ cat doc/README.txt
User1 hacked.
<<<<<<< HEAD
Hello, user2. #当前分支修改方法
=======
Hello, user1. #源分支修改方法
>>>>>>> a123390b8936882bd53033a582ab540850b6b5fb
User2 hacked.
User2 hacked again.
```

  解决方法：
  i. 分析哪种修改方法正确，手动合并。
  ii. 提交修改。
  b. 场景二：文件被重命名为不同的名字
  解决方法：
  i. 确认哪个名字是正确的，删除错误的。
  ii. 提交修改。

