

# Git

---

xbZhong

2024-08-29

[本页PDF](#)

## Git

---

==分布式版本控制工具==

## 版本控制

---

版本控制是一种管理和跟踪文件更改的系统，主要用于协作开发中跟踪代码或文档的修改历史。它允许你随时**回溯到任何一个历史版本**，并在多人合作时有效地处理多个修改。

## 分布式版本控制

- 不依赖于中央服务器，可以在本地进行版本控制，**每个开发者都有项目的完整副本**
- 每个开发者都可以互相查看代码，只需要对别人的仓库进行同步

## 工作流程

---

- 工作区：写代码
- 暂存区：临时存储
- 本地库：历史版本
- 远程库：即代码托管中心，可以把本地库中的代码传到远程库
- 工作区和暂存区的代码可以删除，一旦把代码上传到本地库，代码就无法删除，只能覆盖

## 远程库(代码托管中心)

---

- 局域网：gitlab，非开源
- 互联网：github

## git常用命令

---

- git config --global user.name 用户名：设置用户签名
- git config --global user.email 邮箱：设置用户签名
- git init：初始化本地库
- git status：查看本地库状态
- git add：后面跟文件名，将文件增加到暂存区
- git rm --cached:后面跟文件名，从暂存区删除对应文件
- git commit -m “日志信息” “文件名”:日志信息是版本信息
- git reflog:查看版本号
- git log:查看详细版本信息
- vim 文件名:修改文件
- cat 文件名:查看文件
- git reset --hard 版本号:查看历史版本，版本穿梭

| 命令                                | 说明                               |
|-----------------------------------|----------------------------------|
| git config --global user.name 用户名 | 设置用户签名                           |
| git config --global user.email 邮箱 | 设置用户邮箱签名                         |
| git init                          | 初始化本地库（需要在被管理的文件夹的路径下初始化）        |
| git status                        | 查看本地库状态                          |
| git add 文件名                       | 后面跟文件名，将文件添加到暂存区                 |
| git rm --cached 文件名               | 后面跟文件名，在暂存区删除对应文件                |
| git config list                   | 查看配置文件中的所有配置项                    |
| git commit -m “日志信息” “文件名”        | 日志信息是版本信息，便于查看是哪个版本              |
| git reflog                        | 查看版本信息，可以看到对应的版本号                |
| git reset -hard 版本号               | 版本穿梭                             |
| git fetch                         | 用于从远程库拉取代码，可以在执行命令之后决定要不要合并到当前分支 |
|                                   |                                  |
|                                   |                                  |
|                                   |                                  |

## Gitlab

### Git分支

- 可以同时推进多个功能开发，提升效率，不影响用户的使用。一般来说只有master一个分支
- git branch 分支名:创建分支
- git branch -v：查看分支、
- git checkout 分支名：切换分支
- git merge 分支名：把指定的分支合并到当前分支上
  - 冲突合并：同一个文件在不同分支都做了修改，需要手动合并，保留需要的代码，把不需要的代码和特殊符号删除并保存到暂存区

### 跟踪远程分支

可以让==本地分支跟踪远程分支==，在进行push和pull时不用指定远程仓库

### 工作流介绍

#### 功能开发工作流

- 在master分支上进行上线
- ==每个功能对应一个分支==，在该分支上设计代码，开发功能
- 开发完功能后将代码合并到master分支进行上线

#### GitFlow工作流

- 在master分支上进行上线

- 在developer分支进行开发
  - 要从master分支上先拉取到developer分支
  - ==每个功能对应一个分支==，在该分支上开发功能
- 在测试分支上对开发的功能进行测试
- 在hotfix分支上进行修复代码，通常不需要改核心内容，只是==对外部配置文件进行修改==

## 标记

对==议题==和==合并请求==进行标记，通常用于标记某个重要的代码版本

## 议题

- 用于跟踪和管理任务、错误、功能请求以及讨论的内容
- 议题提供了一个永久的记录，能够让团队回顾以前讨论的内容、决策、问题以及解决方案
- 可以在议题中创建分支，方便进行开发

## 冲突

- ==往远程仓库进行推送时才会产生冲突==
- 原因是因为**远程库和本地库的版本不同**，即远程库领先了多个commit

## 不同的人修改不同的文件

- 别人commit代码后，远程库版本更新，在你push时会产生冲突
- 可以直接进行merge合并，把最新的远程库版本拉取到本地，保证本地库和远程库的版本是一致的，然后再进行push

## 不同的人修改同文件的不同位置

直接merge即可

- 在不同位置修改文件后会使得远程库的版本领先于本地库
- 在push时会产生冲突，直接merge即可

## 不同的人修改同文件的相同位置

- 产生冲突的原因同上，但需要在push时进行手动合并
- 需要判断这个位置的代码要使用你自己的还是别人的，这时候，需要你进行手动合并，然后再进行push

## 不同的人修改同一文件的文件名

- Git无法判断最终文件名的名字
- 需要在push前先进行pull或者fetch，==手动==决定使用什么文件名，然后再进行push

## ==注意事项==

1. 远程库分支受保护时，不可以直接==push==，需要在本地新建一个和远程库不同名字的分支，然后push，**接着再gitlab上发送合并请求**
2. `git pull -rebase` 可以把你的本地提交“移到”远程更新之后，从而避免产生额外的合并提交。==合并提交会将远程和本地的提交进行合并，然后在本地库进行提交==