

## 初始化设置

```
git config --global user.name "Your Name"    配置用户名
git config --global user.email "mail@example.com" 配置邮箱
git config --global credential store         存储配置
```

## 创建仓库

```
git init <project-name>
创建一个新的本地仓库（省略 project-name 则在当前目录创建。
```

```
git clone <url>
克隆一个远程仓库。
```

## 四个区域

**工作区 (Working Directory)**  
就是你在电脑里能实际看到的目录。

**暂存区 (Stage/Index)**  
暂存区也叫索引，用来临时存放未提交的内容，一般在 .git 目录下的 index 中。

**本地仓库 (Repository)**  
Git 在本地的版本库，仓库信息存储在 .git 这个隐藏目录中。

**远程仓库 (Remote Repository)**  
托管在远程服务器上的仓库。常用的有 GitHub、GitLab、Gitee。

## 文件状态

**已修改 (Modified)**  
修改了但是没有保存到暂存区的文件。

**已暂存 (Staged)**  
修改后已经保存到暂存区的文件。

**已提交 (Committed)**  
把暂存区的文件提交到本地仓库后的状态。

## 文件状态

main/master	默认主分支
Origin	默认远程仓库
HEAD	指向当前分支的指针
HEAD^	上一个版本
HEAD~	上四个版本

## 特殊文件

.git	Git 仓库的元数据和对象数据库
.gitignore	忽略文件，不需要提交到仓库的文件
.gitattributes	指向当前分支的指针
.gitkeep	使空目录被提交到仓库
.gitmodules	记录子模块的信息
.gitconfig	记录仓库的配置信息

## 添加和提交

```
git add <file>
添加一个文件到暂存区，比如 git add . 就表示添加所有文件到暂存区。
```

```
git commit -m "message"
提交所有暂存区的文件到本地仓库。
```

```
git commit -am "message"
提交所有已修改的文件到本地仓库。
```

## 分支

```
git branch
查看所有本地分支，当前分支前面会有一个星号*，-r 查看远程分支，-a 查看所有分支。
```

```
git branch <branch-name>
创建一个新的分支。
```

```
git checkout -b <branch-name>
切换到指定分支，并更新工作区。
```

```
git branch -d <branch-name>
删除一个已经合并的分支。
```

```
git checkout -D <branch-name>
删除一个分支，不管是否合并。
```

```
git tag <tag-name>
给当前的提交打上标签，通常用于版本发布。
```

```
git merge --no-ff -m message <branch-name>
合并分支，--no-ff 参数表示禁用 Fast Forward 模式，合并后的历史有分支，能看出曾经做过合并，而 -ff 参数表示使用 Fast Forward 模式，合并后的历史会变成一条直线。
```



```
git squash <branch-name>
合并&挤压 (squash) 所有提交到一个提交。
```

```
git checkout <dev>
git rebase <main>
```

**rebase** 操作可以把本地未 **push** 的分叉提交历史整理成直线，看起来更加直观。但是，如果多人协作时，不要对已经推送到远程的分支执行 **rebase** 操作。  
**rebase** 不会产生新的提交，而是把当前分支的每一个提交都“复制”到目标分支上，然后再把当前分支指向目标分支，而 **merge** 会产生一个新的提交，这个提交有两个分支的所有修改。



## 撤销和恢复

```
git mv <file> <new-file>
移动一个文件到新的位置。
```

```
git rm <file>
只从暂存区中删除一个文件，工作区中的文件没有变化。
```

```
git rm --cached <file>
恢复一个文件到之前的版本。
```

```
git checkout <file> <commit-id>
恢复一个文件到之前的版本。
```

```
git revert <commit-id>
创建一个新的提交，用来撤销指定的提交，后者的所有变化将被前者抵消，并且应用到当前分支。
```

```
git reset --mixed <commit-id>
重置当前分支的 HEAD 为之前的某个提交，并且删除所有之后的提交。
--hard 参数表示重置工作区和暂存区，
--soft 参数表示重置暂存区，
--mixed 参数表示重置工作区。
```

```
git restore --staged <file>
撤销暂存区的文件，重新放回工作区 (git add 的反向操作)。
```

## 查看状态或差异

```
git status
查看仓库状态，列出还未提交的新的或修改的文件。
```

```
git log --oneline
查看提交历史，--oneline 表示简介模式。
```

```
git diff
查看未暂存的文件更新了哪些部分。
```

```
git diff <commit-id> <commit-id>
查看两个提交之间的差异。
```

## Stash

```
git stash save "message"
Stash 操作可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作。
-u 参数表示把所有未跟踪的文件也一并存储；
-a 参数表示把所有未跟踪的文件和忽略的文件也一并存储；
save 参数表示存储的信息，可以不写。
```

```
git stash list
查看所有 stash。
git stash pop
恢复最近一次 stash。
git stash pop stash@{2}
恢复指定的 stash，stash@{2} 表示第三个 stash，stash@{0} 表示最近的 stash。
```

```
git stash apply
重新接受最近一次 stash。
```

```
git stash drop stash@{2}
pop 和 apply 的区别是，pop 会把 stash 内容删除，而 apply 不会。
可以使用 git stash drop 来删除 stash。
```

```
git stash clear
删除所有 stash。
```

## 远程仓库

```
git remote add <remote-name> <remote-url>
添加远程仓库。
```

```
git remote -v
查看远程仓库。
```

```
git remote rm <remote-name>
删除远程仓库。
```

```
git remote rename <old-name> <new-name>
重命名远程仓库。
```

```
git pull <remote-name> <branch-name>
从远程仓库拉取代码。默认拉取远程仓库名 origin 的 master 或者 main 分支。
```

```
git pull --rebase
将本地改动的代码 rebase 到远程仓库的最新代码上 (为了有一个干净、线性的提交历史)。
```

```
git push <remote-name> <branch-name>
推送代码到远程仓库 (然后再发起 pull request)。
```

```
git fetch <remote-name>
获取所有远程分支。
```

```
git branch -r
查看远程分支。
```

```
git fetch <remote-name> <branch-name>
Fetch 某一个特定的远程分支。
```

## GitFlow

**GitFlow** 是一种流程模型，用于在 Git 上管理软件开发项目。

**主分支 (master/main)**：代表了项目的稳定版本，每个提交到主分支的代码都应该是经过测试和审核的。

**开发分支 (develop)**：用于日常开发。所有的功能分支、发布分支和热修复分支都应该从开发分支派生出来。

**功能分支 (feature)**：用于开发单独的功能或者特性。每个功能分支都应该从开发分支派生，并在开发完成后合并回开发分支。

**发布分支 (release)**：用于准备项目发布。发布分支应该从开发分支派生，并在准备好发布版本后合并回主分支和开发分支。

**热修复分支 (hotfix)**：用于修复主分支上的紧急问题。热修复分支应该从主分支派生，并在修复完成后，合并回主分支和开发分支。