初始化

Git Cheat Sheet by GeekHour

初始化设置用户名和邮箱

```
git config --global user.name "Your Name"
git config --global user.email  email@mail.com
git config --global credential.helper store
```

创建仓库

创建一个新的本地仓库（省略project-name则在当前目录创建）

```
git init <project-name>
```

下载一个远程仓库

```
git clone <url>
```

Git的四个区域

- `工作区（Working Directory）`：就是你在电脑里能看到的目录
- `暂存区（Stage/Index）`：一般存放在`.git`目录下的`index`文件，所以我们把暂存区有时也叫作索引（index）
- `本地仓库（Repository）`：工作区有一个隐藏目录`.git`，这个不算工作区，而是Git的版本库
- `远程仓库（Remote）`：托管在远程服务器上的仓库

Git的三种状态

- `已修改（Modified）`：修改了文件，但没保存到暂存区
- `已暂存（Staged）`：把修改后的文件放到暂存区
- `已提交（Committed）`：把暂存区的文件提交到本地仓库

基本概念

- `main`：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~4`：上4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`：指定文件的属性，比如换行符
- `.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

添加一个文件到仓库

```
git add <file>
```

添加所有文件到仓库

```
git add .
```

提交所有暂存区的文件到仓库

```
git commit -m "message"
```

提交所有已修改的文件到仓库

```
git commit -am "message"
```

分支

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

```
git branch
```

创建一个新分支

```
git branch <branch-name>
```

切换到指定分支，并更新工作区

```
git checkout <branch-name>
```

创建一个新分支，并切换到该分支

```
git checkout -b <branch-name>
```

删除一个已经合并的分支

```
git branch -d <branch-name>
```

删除一个分支，不管是否合并

```
git branch -D <branch-name>
```

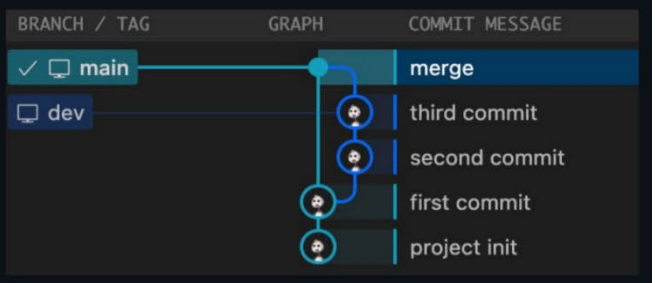
给当前的提交打上标签，通常用于版本发布

```
git tag <tag-name>
```

合并分支

合并分支a到分支b，`-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 merge --no-ff -m "message" <branch-name>
```




```
git merge --ff -m "message" <branch-name>
```

合并&squash所有提交到一个提交

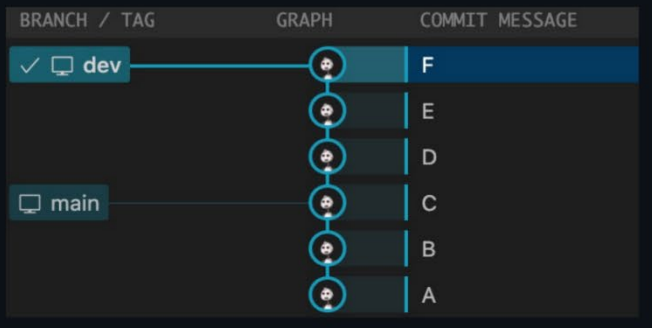
```
git merge --squash <branch-name>
```

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。

Rebase

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。

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



撤销

移动一个文件到新的位置

```
git mv <file> <new-file>
```

从工作区和暂存区中删除一个文件，然后暂存删除操作

```
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>
```

创建一个新的提交，用来撤销指定的提交，后者的所有变化都将被前者抵消，并且应用到当前分支

```
git revert <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 reset --mixed <commit-id>
```

撤销暂存区的文件，重新放回工作区（git add的反向操作）

```
git restore --staged <file>
```

查看

列出还未提交的新的或修改的文件

```
git status
```

查看提交历史，`--oneline`可省略

```
git log --oneline
```

查看未暂存的文件更新了哪些部分

```
git diff
```

查看两个提交之间的差异

```
git diff <commit-id> <commit-id>
```

Stash

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 save "message"
```

查看所有stash

```
git stash list
```

恢复最近一次stash

```
git stash pop
```

恢复指定的stash，stash@{2}表示第三个stash，stash@{0}表示最近的stash

```
git stash pop stash@{2}
```

重新接受最近一次stash

```
git stash apply
```

`pop`和`apply`的区别是，`pop`会把`stash`内容删除，而`apply`不会。可以用`git stash drop`来删除`stash`。

```
git stash drop stash@{2}
```

删除所有stash

```
git stash clear
```

远程仓库

添加远程仓库

```
git remote add <remote-name> <remote-url>
```

查看远程仓库

```
git remote -v
```

删除远程仓库

```
git remote rm <remote-name>
```

重命名远程仓库

```
git remote rename <old-name> <new-name>
```

从远程仓库拉取代码

```
git pull <remote-name> <branch-name>
```

fetch默认远程仓库（origin）当前分支的代码，然后合并到本地分支

```
git pull
```

将本地改动的代码Rebase到远程仓库最新的代码上（为了有一个干净的、线性的提交历史）

```
git pull --rebase
```

推送代码到远程仓库（然后再发起pull request）

```
git push <remote-name> <branch-name>
```

获取所有远程分支

```
git fetch <remote-name>
```

查看远程分支

```
git branch -r
```

fetch某一个特定的远程分支

```
git fetch <remote-name> <branch-name>
```

Git Flow

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）`：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。所有功能分支、发布分支和修补分支都应该从 develop 分支派生。
- `功能分支（feature）`：用于开发单独的功能或特性。每个功能分支应该从 develop 分支派生，并在开发完成后合并回 develop 分支。
- `发布分支（release）`：用于准备项目发布。发布分支应该从 develop 分支派生，并在准备好发布版本后合并回 master 和 develop 分支。
- `修补分支（hotfix）`：用于修复主分支上的紧急问题。修补分支应该从 master 分支派生，并在修复完成后合并回 master 和 develop 分支。