# Git笔记

## Git介绍

- Git是分布式版本控制系统
- 集中式VS分布式，SVN VS Git
  i. SVN和Git主要的区别在于历史版本维护的位置
  ii. Git本地仓库包含代码库还有历史库，在本地的环境开发就可以记录历史而SVN的历史库存在于中央仓库，每次对比与提交代码都必须连接到中央仓库才能进行。
  iii. 这样的好处在于：
  - 自己可以在脱机环境查看开发的版本历史。
  - 多人开发时如果充当中央仓库的Git仓库挂了，可以随时创建一个新的中央仓库然后同步就立刻恢复了中央库。

## Git命令

### Git配置

```
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
```

`git config`命令的`--global`参数，表明这台机器上的所有Git仓库都会使用这个配置，也可以对某个仓库指定不同的用户名和邮箱地址。

### 创建版本库

#### 初始化一个Git仓库

```
$ git init
```

#### 添加文件到Git仓库

包括两步：

```
$ git add <file>
$ git commit -m "description"
```

`git add`可以反复多次使用，添加多个文件，`git commit`可以一次提交很多文件，`-m`后面输入的是本次提交的说明，可以输入任意内容。

### 查看工作区状态

```
$ git status
```

### 查看修改内容

```
$ git diff
```

```
$ git diff --cached
```

```
$ git diff HEAD -- <file>
```

- `git diff` 可以查看工作区(w ork dict)和暂存区(stage)的区别
- `git diff --cached` 可以查看暂存区(stage)和分支(master)的区别
- `git diff HEAD -- <file>` 可以查看工作区和版本库里面最新版本的区别

## 查看提交日志

```
$ git log
```

简化日志输出信息

```
$ git log --pretty=oneline
```

## 查看命令历史

```
$ git reflog
```

## 版本回退

```
$ git reset --hard HEAD^
```

以上命令是返回上一个版本，在Git中，用 `HEAD` 表示当前版本，上一个版本就是 `HEAD^` ，上上一个版本是 `HEAD^^` ，往上100个版本写成 `HEAD~100` 。

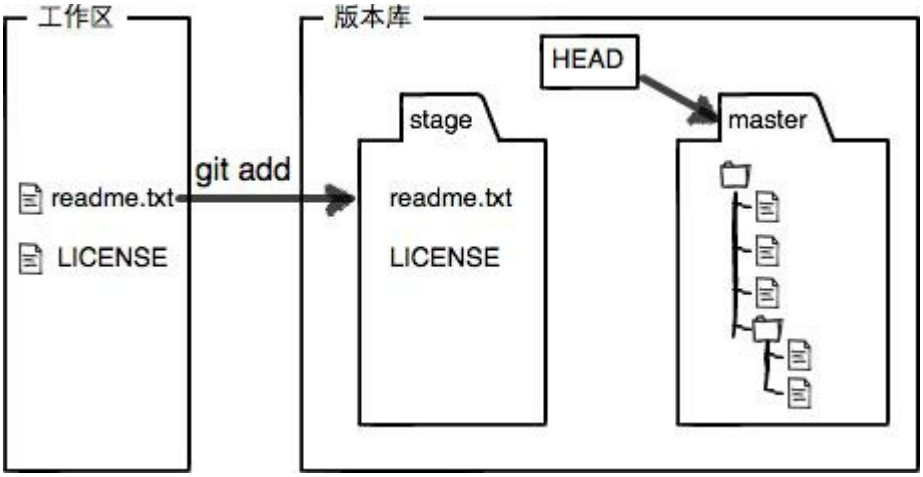
## 回退指定版本号

```
$ git reset --hard commit_id
```

commit_id是版本号，是一个用SHA1计算出的序列

## 工作区、暂存区和版本库

工作区：在电脑里能看到的目录；
版本库：在工作区有一个隐藏目录 `.git` ，是Git的版本库。
Git的版本库中存了很多东西，其中最重要的就是称为stage（或者称为index）的暂存区，还有Git自动创建的 `master` ，以及指向 `master` 的指针 `HEAD` 。




进一步解释一些命令：

- `git add` 实际上是把文件添加到暂存区
- `git commit` 实际上是把暂存区的所有内容提交到当前分支

# 撤销修改

## 丢弃工作区的修改

```
$ git checkout -- <file>
```

该命令是指将文件在工作区的修改全部撤销，这里有两种情况：

1. 一种是file自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；
2. 一种是file已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

## 丢弃暂存区的修改

分两步：
第一步，把暂存区的修改撤销掉(unstage)，重新放回工作区：

```
$ git reset HEAD <file>
```

第二步，撤销工作区的修改

```
$ git checkout -- <file>
```

小结：

1. 当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令`git checkout -- <file>`。

2. 当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令`git reset HEAD <file>`，就回到了第一步，第二步按第一步操作。

3. 已经提交了不合适的修改到版本库时，想要撤销本次提交，进行版本回退，前提是没有推送到远程库。

# 删除文件

```
$ git rm <file>
```

`git rm <file>`相当于执行

```
$ rm <file>
$ git add <file>
```

## 进一步的解释

Q：比如执行了`rm text.txt`误删了怎么恢复？
A：执行`git checkout -- text.txt`把版本库的东西重新写回工作区就行了
Q：如果执行了`git rm text.txt`我们会发现工作区的text.txt也删除了，怎么恢复？
A：先撤销暂存区修改，重新放回工作区，然后再从版本库写回到工作区

```
$ git reset head text.txt
$ git checkout -- text.txt
```

Q：如果真的想从版本库里面删除文件怎么做？
A：执行`git commit -m "delete text.txt"`，提交后最新的版本库将不包含这个文件

# 远程仓库

## 创建SSH Key

```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

## 关联远程仓库

```
$ git remote add origin https://github.com/username/repositoryname.git
```

## 推送到远程仓库

```
$ git push -u origin master
```

`-u`表示第一次推送master分支的所有内容，此后，每次本地提交后，只要有必要，就可以使用命令`git push origin master`推送最新修改。

## 从远程克隆

```
$ git clone https://github.com/usern/repositoryname.git
```

# 分支

### 创建分支

```
$ git branch <branchname>
```

### 查看分支

```
$ git branch
```

`git branch`命令会列出所有分支，当前分支前面会标一个*号。

### 切换分支

```
$ git checkout <branchname>
```

### 创建+切换分支

```
$ git checkout -b <branchname>
```

### 合并某分支到当前分支

```
$ git merge <branchname>
```

### 删除分支

```
$ git branch -d <branchname>
```

### 查看分支合并图

```
$ git log --graph
```

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。用`git log --graph`命令可以看到分支合并图。

### 普通模式合并分支

```
$ git merge --no-ff -m "description" <branchname>
```

因为本次合并要创建一个新的commit，所以加上`-m`参数，把commit描述写进去。合并分支时，加上`--no-ff`参数就可以用普通模式合并，能看出来曾经做过合并，包含作者和时间戳等信息，而fast forward合并就看不出来曾经做过合并。

### 保存工作现场

```
$ git stash
```

### 查看工作现场

```
$ git stash list
```

### 恢复工作现场

```
$ git stash pop
```

### 丢弃一个没有合并过的分支

```
$ git branch -D <branchname>
```

### 查看远程库信息

```
$ git remote -v
```

### 在本地创建和远程分支对应的分支

```
$ git checkout -b branch-name origin/branch-name，
```

本地和远程分支的名称最好一致；

### 建立本地分支和远程分支的关联

```
$ git branch --set-upstream branch-name origin/branch-name；
```

### 从本地推送分支

```
$ git push origin branch-name
```

如果推送失败，先用git pull抓取远程的新提交；

### 从远程抓取分支

```
$ git pull
```

如果有冲突，要先处理冲突。

## 标签

tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。

### 新建一个标签

```
$ git tag <tagname>
```

命令`git tag <tagname>`用于新建一个标签，默认为HEAD，也可以指定一个commit id。

### 指定标签信息

```
$ git tag -a <tagname> -m <description> <branchname> or commit_id
```

`git tag -a <tagname> -m "blablabla..."`可以指定标签信息。

### PGP签名标签

```
$ git tag -s <tagname> -m <description> <branchname> or commit_id
```

`git tag -s <tagname> -m "blablabla..."`可以用PGP签名标签。

### 查看所有标签

```
$ git tag
```

### 推送一个本地标签

```
$ git push origin <tagname>
```

### 推送全部未推送过的本地标签

```
$ git push origin --tags
```

### 删除一个本地标签

```
$ git tag -d <tagname>
```

### 删除一个远程标签

```
$ git push origin :refs/tags/<tagname>
```