# Git

## 时光机穿梭

**git init 将目录变成Git可管理的仓库**

初始化一个Git仓库，使用git init命令。

**git add 添加文件，可反复使用，可添加多个**

**git commit 提交文件**

**git commit -m “本次提交说明” 说明是必须的**

使用命令git add <file>，注意，可反复多次使用，添加多个文件；

使用命令git commit -m <message>，完成。

**git status 查看工作区状态**

**git diff 查看修改的内容**

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

**git log 查看提交历史**

**git reset 版本回退**

**git reflog 查看命令历史**

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

**git reset --hard HEAD^ 回退到上个版本**

在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

HEAD指向的版本就是当前版本，因此，**Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。**

**git diff HEAD -- 文件名 命令可以查看工作区和版本库里面最新版本的区别**

**git checkout -- 文件名 撤回修改的东西**

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到git checkout命令。

撤回修改分为三种情况

当文件还没有git add提交的时候用 git checkout -- 文件名

当文件已经git add 提交 但是没有 git commit 的时候 使用 git reset HEAD 文件名 退回到工作区 然后 用git checkout --文件名放弃工作区的修改

当文件已经git add 提交 并且还git commit 后的时候，可使用git reset --hard HEAD^ 回退到上个版本 （条件是还未推送到远程）

**git rm 文件 删除提交的文件**

**删除过后 需要重新 git commit 提交下**

删除文件分两种

一种是 本地rm 直接删除后，确定没有删除错误，然后

git rm 文件 删除版本库中的文件（当这个删除后即无法用 git checkout -- 文件）恢复

一种是觉得删除错误， 可以用

git checkout -- 文件 从版本库中恢复

## 远程仓库--远程连接篇

创建SSH KEY

**ssh-keygen -t rsa -C “邮箱地址”**

创建时会出现用户主目录

根据用户主目录找到.ssh 下的

**id\_rsa 和id\_rsa.pub文件**

**id\_rsa 私钥**

**id\_rsa.pub 公钥**

1.在Github上面建一个新的空仓库，注意不要勾选用readme初始化。

2.在github 网站添加完成后运行下面命令，让本地库与远程库关联起

**git remote add origin [git@github.com:github用户名/仓库.git](mailto:git@github.com:github用户名/仓库.git) \*\*此关联在windows 下不管用**

**git remote add origin [https://github.com/用户名/仓库.git](https://github.com/daoke0818/testGit2.git) 这个是正确的 记得将用户名和对应的库名修改**

3.关联成功后用如下命令把本地内容推送到远程库中，其中 -u 是指定后面的仓库名和分支名为默认，以后直接用 git push 即可

**Git push -u origin master**

**修改远程库**

除了上面删除再添加的办法，还可以使用如下命令：

**git remote set-url origin [git@github.com:git用户名](mailto:git@gitee.com:git用户名) /新的仓库地址.git**

**从远程仓库克隆**

经测试，windows上

**git clone git@github.com:git用户名/仓库名.git**

报错：Please make sure you have the correct access rights and the repository exists.

大概还是网络原因，用https协议就可以：

**git clone <https://github.com/用户名/仓库.git>**

尽管ssh支持的原生git协议要比https协议快。

## 分支管理

创建分支

**Git branch 分支名称 创建分支**

**Git checkout 分支名称 切换分支**

**Git checkout -b 分支名称 创建分支并切换**

-b 参数表示创建并切换

**Git branch 查看当前分支名称**

**Git merge 合并到当前分支的分支名称 合并分支**

**Git branch -d 分支名称 删除分支**

-d 代表删除分支

**Switch 创建与切换**

**Git switch -c 分支名称 创建分支并切换**

**Git switch 分支名称 切换分支**

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>或者git switch <name>

创建+切换分支：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

**git log --graph --pretty=oneline --abbrev-commit 查看图形化流程**

其中， --graph 是图形化， --pretty=oneline 是一行显示， --abbrev-commit 是只显示每次提交id的前几位

**默认情况下，如果情况允许，Git会自动用快进模式合并分支，但这样合并后不会留下分支存在过的痕迹。删除分支后就会丢失相应信息。如果不想这样做，则在合并时加上参数 --no-ff ，Git则会生成一个提交，所以同时再加上一个提交信息（如果不加则会进入vim模式让编辑提交信息），代码如下：**

**Git merge --no-ff -m "说明" 分支名称**

**分支说明：**

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

**Git stash 存储分支**

--即如果在当前分支正在工作时突然间有其他工作需要修改先存储当前分支进度，然后切换回master 分支后创建新的分支修改工作完成后在切换会原来分支即可

**Bug 修改完成后可用：**

**Git stash list 查看存储的分支**

**恢复分支有两种方法**

**Git stash apply 恢复分支**

**Git stash drop 删除存储分支**

Git stash apply方法恢复后，原来stash 存储的内容并不删除，需要git stash drop 删除存储分支

**Git stash pop 恢复分支**

Git stash pop方法恢复分支的同时也将stash的存储的内容也删除了，推荐使用此种方法

存储分支可以多次 stash 存储 恢复的时候 先使用git stash list 查看

然后恢复指定stash

**Git stash apply stash@{0} 恢复指定的存储分支 0代表存储的哪次**

使用多次存储时恢复 推荐使用 git stash apply 方法来恢复

**Git cherry-pick 提交commit ID 将对应的提交复制至当前分支**

**Git branch -D 分支名称 强行删除分支**

## 其它

**Git 的一些相关信息**

### **设置或者修改用户名及邮箱**

git config --global user.name “你的用户名”

git config –-global user.email “你的邮箱”

### **查看配置信息**

Git config --list

当

$ git add readme.txt  
warning: LF will be replaced by CRLF in readme.txt.  
The file will have its original line endings in your working directory.

时

解决办法：

$ rm -rf .git // 删除.git

$ git config --global core.autocrlf false //禁用自动转换

然后重新执行：

$ git init

$ git add 文件名

其实只需要 $ git config --global core.autocrlf false 就可以了