1. **起步**

//设置用户名和邮箱

$ git config --global user.name "Your Name"

$ git config --global user.email ["email@example.com"](mailto:%22email@example.com%22)

$ cat readme.txt //查看readme.txt的内容

$ Ls -al //查看仓库中的文件

$ pwd //显示当前目录

$ vim readme.txt //按“i”编辑readme.txt文件

Esc退出vim编辑 :wq回车 保存并退出

$ rm -rf .git //删除本地仓库

1. **创建版本库**

$ mkdir learngit //创建名为learngit的仓库

$ cd learngit //进入仓库

//将这个目录变成git可管理的仓库

$ git init

Initialized empty Git repository in /Users/michael/learngit/.git/

$ git add readme.txt //把文件添加到仓库（暂存区）

$ git add –all //一次添加所有

$ git commit -m "这里写提交的内容" //把文件提交到仓库 一次可以提交多个文件

1. **版本控制**

$ git status //查看当前仓库状态

$ git diff readme.txt //查看修改reademe.txt的内容

1. **版本回退**

$ git log //查看每个版本（最近到最远）commit后的位版本号

$ git reset --hard HEAD^ //回退到上个版本 HEAD^^上上个 依次类推 HEAD~100往上100个

$ git reset --hard 版本号 //回退到这个版本 （版本号写前几位即可）

$ git reflog //查看所有的版本号

1. **版本库**



1. **撤销修改**

$ git checkout -- readme.txt //撤销readme.txt的修改

有两种情况：

1. readme.txt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. readme.txt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。

\*git checkout -- readme.txt 命令如果没有“--”就变成了“切换到另一个分支”的命令

git reset HEAD <file>可以把暂存区的修改撤销掉，重新放回工作区

再用git checkout -- readme.txt丢弃工作区的修改

1. **删除文件**

$ rm test.txt //工作区删除test.txt

$ git commit -m "remove test.txt" //从版本库也删除

$ git checkout -- test.txt //恢复删除的文件（版本库的最新版本）

1. **远程仓库**

$ ssh-keygen -t rsa -C ["youremail@example.com"](mailto:%22youremail@example.com%22) //创建SSH key

1. **添加远程库**

//将GitHub的仓库（origin）与本地仓库关联

$ git remote add origin（GitHub仓库名） [git@github.com:gitname/learngit.git](mailto:git@github.com:michaelliao/learngit.git)

$ git remote origin set-url [url] //修改远程仓库地址:修改url 或删除远程仓库重新添加

$ git push -u origin master //将本地仓库的所有内容推送到远程库上

$ git push origin master //只要本地仓库做了修改 就能用这个命令将最新内容推送到远程仓库

$ git remote rm origin //删除远程仓库

$ git remote -v //仓看远程仓库

1. **从远程库克隆**

$ git clone [git@github.com:michaelliao/gitskills.git](mailto:git@github.com:michaelliao/gitskills.git) //克隆一个本地库

1. **创建&合并分支**

$ git branch //查看分支

$ git branch <name> //创建分支

$ git checkout <name> //切换到分支

$ git checkout -b <name> //创建并切换到分支

$ git merge <name> //合并某分支到当前分支

$ git branch -d <name> //删除分支 删除一个未合并过的分支用 -D

通常，合并分支时Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。如果强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。（用--no-ff的方式合并，如下）

$ git merge --no-ff -m "message" dev

1. **解决冲突**

当一个分支和主分支都做了修改再合并时就会产生同冲突，必须先解决冲突

解决冲突就是把git合并失败的文件手动编辑为我们希望的内容，重新提交

$ git log --graph //查看分支合并图

1. **隐藏工作区**

（1）**git stash** save "save message"  : 执行存储时，添加备注，方便查找，只有git stash 也要可以的，但查找时不方便识别。

（2）**git stash list** ：查看stash了哪些存储

（3）**git stash show** ：显示做了哪些改动，默认show第一个存储,如果要显示其他存贮，后面加stash@{$num}，比如第二个 git stash show stash@{1}

（4）**git stash show -p** : 显示第一个存储的改动，如果想显示其他存存储，命令：git stash show  stash@{$num}  -p ，比如第二个：git stash show  stash@{1}  -p

（5）**git stash apply** :应用某个存储,但不会把存储从存储列表中删除，默认使用第一个存储,即stash@{0}，如果要使用其他个，git stash apply stash@{$num} ， 比如第二个：git stash apply stash@{1}

（6）**git stash pop** ：命令恢复之前缓存的工作目录，将缓存堆栈中的对应stash删除，并将对应修改应用到当前的工作目录下,默认为第一个stash,即stash@{0}，如果要应用并删除其他stash，命令：git stash pop stash@{$num} ，比如应用并删除第二个：git stash pop stash@{1}

（7）**git stash drop** stash@{$num} ：丢弃stash@{$num}存储，从列表中删除这个存储

（8）**git stash clear** ：删除所有缓存的stash

1. **多人协作**

$ git remote //查看远程库的信息

$ git remote -v //显示更详细的信息

$ git push origin master //把该分支的提交推送的远程库的主分支也可以推到其他分支

**多人协作的工作模式通常是这样：**

首先，可以试图用git push origin <branch-name>推送自己的修改；

如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>。

1. **创建标签**

首先，切换到需要打标签的分支上

$ git tag <name> //在最新的commit上创建一个标签

$ git tag //查看所有标签

$ git tag <name> xxxxx //为id为xxxxx的一次commit创建一个标签

$ git show <tagname> //查看指定的标签

$ git tag -a v0.1 -m "xxx" xxx /创建带有说明的标签 /-a标签名 -m说明文字

标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。