# http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000

# 全局配置

git init

git config --global user.name "your user name"

git config --global user.email "your email"

--global为改变全局配置，将对所有的仓库进行配置

不加--global为对个别仓库进行配置

git只能控制文本文件的改动，word属于二进制文件无法跟踪改动。

windows下需要使用专用编辑器，避免boom头

git add README.md

git commit -m "some test file"   //不使用-m参数，需要修改文件，来确定提交

-m为本次提交的说明

# 版本回退

git status

     目前的状态，显示哪个文件被修改，是否提交等

git diff README.md

     查看修改内容

git log --pretty=oneline

     查看提交日志，每次提交信息在一行内显示

     3628164fb26d48395383f8f31179f24e0882e1e0  some test file

     前面是SHA1的版本号

git reset --hard HEAD^

     回退到上个版本(^符号是上个版本，^^是上上个版本）

git reset --hard HEAD~100

     回退到前100个版本

git reset --hard 3628164

     使用版本号的前几位进行回退

回退的速度很快，只是修改了指针指向

git reset *commit-sha1* 回退到某个版本，不更改工作区，只保留源码，退回commit和stage信息，此时提交需要重新add到缓存区

git reset --soft *commit-sha1* 回退到某个版本，只回退commit信息，不回退stage信息，可以直接提交

git reset --hard *commit-sha1* 完全的回退到某个版本，包括源码、stage信息和commit版本号

git reflog

     查看最近几次执行的git历史命令和版本信息

# 工作区和暂存区

仓库文件夹内就是工作区

## 版本库是仓库目录下的.git文件夹

index是暂存区，HEAD是目前master的指针指向的版本号

git add的时候，其实就是把文件添加到了暂存区

git commit是把暂存区的内容提交到当前分支，也就是提交到了版本库.git内

使用git status可以查看当前工作区的内容

## 管理修改

git add后的修改，都不会提交，除非修改后再进行git add，才能合并提交

## 撤销修改

### 1、在还没有提交到暂存区时：

git checkout -- README.md   #注意要加"--"

撤销目前在工作区的全部修改

### 2、已经提交到暂存区(stage)（已经执行git add，git status可以查看到改动）

先使用 git reset HEAD README.md

HEAD表示最新的版本，该命令只回退到上次的版本，工作区文件不会修改，会回退到上次的stage信息和commit信息

再使用 git checkout -- README.md

已经提交到版本库（git commit）

使用版本回退

git checkout 其实是用版本库里的版本来替换工作区的版本

# 克隆版本库

git clone git@服务器名:用户名/仓库名称.git

例如：

    git clone git@192.168.1.165:testuser/redis.git

     这里使用的是ssh协议，克隆前需要配置密钥

第一次推送更新

     git push -u origin master

也可以使用http协议

     git clone http://192.168.1.165/testuser/redis.git

然后输入用户名和密码就可以了

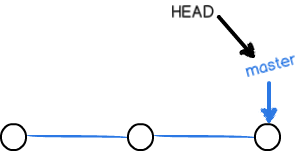
git clone -b dev http://192.168.1.165/testuser/redis.git

仅克隆dev分支到本地

# 分支管理

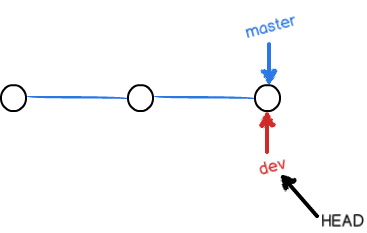
在当前在的稳定代码提交后，又想在这个版本上开发其他功能，就可以使用分支功能创建一个分支，来进行开发新功能，主分支代码还可以供别的人来开发和提交，不影响别人开发的进度

master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点

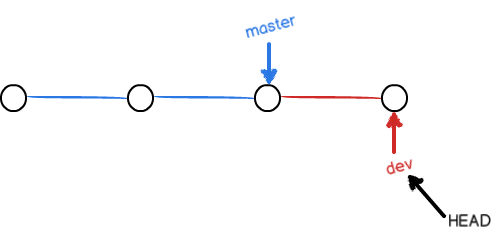


当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上

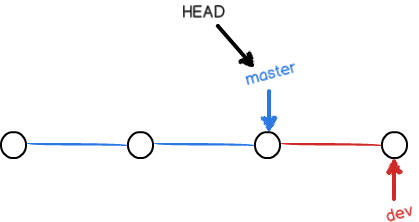
其实就是增加了一个指针，更改了HEAD的指向



当对dev提交后，这时就可以分别对dev和master进行提交



切换回master时



## git checkout -b dev

创建分支，此时已经创建分支dev，并且已经切换到dev分支

此时使用git commit来提交的话，就会全部都提交到dev分支上，master分支不受影响

## git branch

查看当前处于哪个分支

## git checkout master

切换分支，此时我们又切换回master了

## git merge dev

合并分支

执行合并分支时，默认同时会执行提交动作，使用参数"--no-commit"来避免提交

## git branch -d dev

当合并分支之后，就可以安全的删除dev分支了

当不需要当前分支，并且还没进行合并的时候，是不能删除的

## git branch -D dev

强制删除分支

当遇master修改提交后，dev也修改并提交，此时进行合并时，就会出现冲突

例如README文件

git会提示：both modified：README

此时再打开文件，会在文件内标识出有冲突的行：

|  |
| --- |
| <<<<<<< HEAD  adfasdfasdfasdf  =======  lkjl;joijilkj  >>>>>>> dev |

（有些工具可以处理识别出这些标记，来帮助你修改，如vimdiff，具体使用git mergetool --tool-help会列出相关工具）

此时需要把冲突文件手动更改，才能进行提交

将以上文件内容手动更改为：

|  |
| --- |
| adfasdfasdfasdf  lkjl;joijilkj |

才可以提交

## git log --graph

     查看分支合并图

合并后，一般是看不到合并历史的，添加--no-ff参数，就会显示出合并历史

## git merge --no-ff -m "merge with no-ff"

     本次合并会增加一个提交，所以添加上-m参数

# 临时存储分支

## git stash

临时存储当前的工作状态，用于当下需要立即创建一个新分支时，而当前分支还没有提交的时候

## git stash list

查看当前存储的工作状态

## git stash apply

恢复之前保存的工作状态

## git stash drop

删除之前保存的工作状态

## git stash pop

恢复并删除之前保存的工作状态

该功能在如有bug需要解决时，当前dev分支还不需要提交，就可以使用该命令进行存储，等处理完bug后，再恢复

将分支推送到远程仓库

## git remote -v

查看远程仓库有多少个分支

## git push origin dev

推送到dev分支

# 抓取分支

先git clone版本库到本地，然后再切换到分支

## git checkout -b dev origin/dev

切换到远程的dev分支

当在推送到远程分支时，发生了冲突，也就是说，别人在你提交前，已经对该分支进行了修改并且已经提交到了远程仓库，需要先使用 git pull来获取当前最新的版本

先执行：git branch --set-upstream dev origin/dev

再执行：git pull

git pull 其实是执行了git fetch（下载版本到本地） 和git merge（合并该版本），建议分开操作

git会提示有冲突，让你修复冲突，并且会在文件内标识出来，解决冲突后，就可以进行提交了

# 标签管理

git有标签功能，来标识一个版本

## git tag v1.0

标签定义为v1.0

## git tag v1.0 883928

为历史上的一次提交打tag

## git show v1.0

查看该tag上的提交信息

## git tag -a v0.1 -m "version 0.1 released" 3628164

使用-m添加说明文字

## git tag -d v1.0

删除该标签

## git push origin v1.0

推送标签到远程

## git push origin --tags

只推送本地添加的标签到远程

删除远程标签：

## git tag -d v1.0

先删除本地的标签

再推送到远程

## git push origin :refs/tags/<tagname>

例：git push origin :refs/tags/v1.0