# 命令：

## 创建用git用户

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

## 创建版本库

* git init
* git add readme.txt
* git add .
* git commit -m "First commit."

## 查看仓库当前状态

* git status

## 查看修改的地方

* git diff
* git diff readme.txt //查看指定文件的修改情况
* git diff HEAD -- readme.txt //查看指定文件的工作区和版本库里当前分支最新版本的区别
* git diff HEAD^ -- readme.txt //查看指定文件的工作区和版本库里当前分支第二新版本的区别

## 版本回退

* git reset --hard HEAD^
* git reset --hard HEAD^^^
* git reset --hard HEAD~100
* git reset --hard 52fc18e

## 查看日志

* git log //用于查看提交的历史，只有一条时间线
* git reflog //用于查看命令历史，会包含所有时间
* git log --graph --pretty=oneline --abbrev-commit //画出图形

## 撤销修改

* git checkout **--** readme.txt //把文件在工作区的修改全部撤销
* 情况一：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
* 情况二：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 reset HEAD readme.txt //把暂存区的修改撤销掉，退回到版本库当前分支的最新状态

## 分支

* git checkout -b dev //创建dev分支，然后切换到dev分支，'-b'参数表示创建并切换，等价于：
* git branch dev //新建分支dev
* git checkout dev //切换到分支dev，工作区的内容也会随之改变
* git checkout -b dev origin/dev //创建本地分支名为"dev"，并切换到当前该新建的分支，然后将远程分支origin/dev添加到当前分支
* git branch //查看分支，带有\*表示当前分支
* git merge dev //将指定的dev分支合并到当前所处的分支上
* git branch -d dev //删除分支，只有合并后才能删除
* git branch -D deleteBeforeMerge //强行删除分支：commit之后没有合并，无法通过git branch -d deleteBeforeMerge删除
* 创建，合并，删除分支的过程非常快，所以Git鼓励使用分支完成某个任务，合并后在删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全

## 分支管理策略

* 通常，合并分支时，如果可能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
* 加上--no-ff参数就可以使用普通模式合并，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 merge --no-ff -m "merge with no-ff" dev //由于本次合并要创建新的commit，所以加上-m参数，并把commit描述写进去
* Bug分支：
* 场景：在自己所在的分支上干活，完成工作还需1天，现在无法提交，但是master上有一个bug需要修复
* git stash //将当前工作现场储藏起来，等以后恢复现场后继续工作。
* git stash list //查看工作现场
* git stash pop //恢复之前存储的工作现场，并删除stash内容，等价于
* git stash apply
* git stash drop

## 查看远程库的信息

* git remote
* git remote -v
* git remote add origin git@github.com:<账号名>/<项目名称>.git
* git remote remove origin

## 更新.gitignore

* .gitignore只能作用于Untracked Files，也就是那些没有被git记录过的文件(自创建以来，没有被git add 和git commit的文件)
* 更新之后，如何取消对原来文件的追踪：
* git rm --cached test.txt
* git rm -r --cached . **//-r代表recursively，指递归地取消追踪所有文件**
* 然后再执行git add . **//.gitignore中新增的项目就生效了**

## 查看git追踪的文件

* git ls-files
* ls -al //查看当前目录下所有文件

# 概念：

## 工作区：就是你电脑里能看到的目录

## 版本库：工作区里有一个隐藏目录.git，就是版本库

* Git的版本库里存了很多东西，其中最重要的就是称为stage(或index)的**暂存区**，还有Git为我们自动创建的第一个分支master以及指向master的一个指针叫HEAD
* 往Git版本库里添加是分两步执行的，第一步 git add 把文件添加进去，实际上就是把文件修改添加到暂存区；第二步 git commit 提交更改，实际上就是把暂存区的所有内容提交到当前分支

## 管理修改

* Git跟踪并管理的是修改，而非文件
* 删除也是一种修改
* 确实要从版本库中删除：git rm并且 git commit
* 错删后恢复：git checkout -- test.txt //利用版本库里的版本替换工作区里的版本(与7相同)

## 远程仓库

* 最好使用SSH协议(原生协议)
* 创建SSH Key：因为GitHub需要识别你推送的提交确实是你推送的，而不是别人，Git支持SSH协议，所以GitHub只要知道了你的公钥，就可以确认只有你自己才能推送
* GitHub允许添加多个Key
* git remote add origin git@github.com:liuyehcf/Hcf.git //与本地仓库建立关联
* git remote rm origin //删除origin所绑定的远程仓库的关联

## 把本地库的内容推送到远程库上

* git push -u origin master//第一次推送的时候加上-u参数，**本地的分支名称(当前分支)必须与远程的分支名称(这里所指定的master)相同**，**换句话说，当前所处的本地分支必须与指定推送的远程分支名称相同**
* git push origin master//之后的推送，**名称限制同上**
* git push origin dev:dev2 **//提交本地dev分支到远程的dev2分支(如果远程仓库不存在dev2分支，会先创建再推送)**

## 把远程库的内容拉到本地

* git pull //这个命令需要实现设置远程库与本地库的关联
* git branch --set-upstream dev1 origin/dev2 //将本地分支dev1与远程分支dev2建立联系，之后调用git pull命令，会选取该联系。
* git pull origin master //将远程库的master分支拉取到本地的当前分支
* git pull origin master:dev **//将远程库的master分支拉取到本地的dev分支(如果dev分支不存在，会先创建，再拉取)**
* git push origin :dev2 **//删除远程分支dev2**
* 从远程仓库克隆代码(克隆到当前目录下，会生成一个文件夹，其名字与远程仓库名字相同，进入到该文件夹中才会有.git文件夹)
* git clone git@github.com:liuyehcf/gitskills.git

## 分支管理

* 在版本回退里，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支
* **HEAD严格来说不是指向提交，而是指向分支，分支才是指向提交的，所以HEAD指向的就是当前分支**
* 每次提交，master分支都会向前移动一步
* Git创建一个分支(记为：dev)很快，因为除了增加一个指针，改改HEAD指针的指向，工作区的文件都没有任何变化
* Git的合并，就是直接把master指向dev的当前提交，工作区内容也不变
* 合并完后，可以删除分支，就是把新增的dev分支指针删除掉

## 解决冲突

* 手动解决冲突的代码
* 在本地git add以及git commit
* 推送到远程

## 多人协作

* 从一台新设备(之前没在这里开发过)继续项目的开发
* git clone git@github.com:liuyehcf/LTEV2X.git //只会在本地看到远程仓库的master分支
* 进入克隆下来项目的文件夹(该文件夹内已经含有.git文件夹了)
* git checkout -b dev origin/dev //创建本地分支名为"dev"，并切换到当前该新建的分支，然后将远程分支origin/dev添加到当前分支

## 标签管理

* 发布一个版本的时候，通常在版本库中打一个标签，这样就唯一确定了打标签时刻的版本，将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来，所以标签也是版本库的一个快照
* 标签是版本库的快照，实际它就是一个指向某个commit的指针，因此创建和删除标签都是瞬时完成的
* 打标签步命令介绍
* 首先，切换到需要打标签的分支上
* git tag v1.0 **//默认打在最新提交的commit**上
* git tag **//查看所有标签，注意标签不是按时间顺序列出，而是按字母排序的**
* git show v0.9 **//查看标签对应的commit的详细信息**
* git tag v0.9 abcd123 **//abcd123代表了commit的id**
* git tag -a v0.1 -m "version 0.1 released" abcd1234 **//-a 指定标签名，-m指定说明文字**
* git tag -d v0.1 //删除标签
* 创建的标签只存储在本地，不会自动推送到远程。如果要推送某个标签到远程，使用命令
* git push origin v1.0
* git push origin --tags //一次性推送尚未推送到远程的本地标签
* 删除已经推送到远程的标签
* git tag -d v0.9 **//先从本地删除**
* git push origin :refs/tags/v0.9 **//然后从远程删除**

## 配置别名

* git config --global alias.co checkout //用co替代checkout
* git config --global alias.ustage 'reset HEAD'
* git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
* --global 是针对当前用户起作用的，如果不加，那只针对当前仓库起作用
* 每个仓库的配置文件放在了.git/config文件中
* 当前用户的配置文件放在了：
* /Users/HCF //MAC
* 删除别名只需要删除配置文件中的对应行即可

# 常见问题

## 本地创建git仓库后没有任何分支

* 解决方法：git commit 一次就行

## 同一台设备的ssh切换

* 问题描述：想要更换远程仓库，即将本地代码推送到一个新建的远程仓库中去。
* 已经完成的步骤：
* 新建ssh秘钥：ssh-keygen -t rsa -C "描述性的话" **//该描述性的话会添加到ssh公钥的最后，随意写即可(注意这里存的秘钥文件的名字需要自己定义一下(假定为"extra\_rsa"，下面会用到)，否则会覆盖之前的秘钥)**
* 在github上添加了该秘钥
* 添加了关联：git remote add origin1 git@**github.com**:x/x.git
* 出现的问题：
* 如果此时用git push origin1 dev:dev，会出现推送失败，被另一个账号所决绝(denied)
* 原因分析：
* 上面添加的origin并没有指定秘钥
* 解决方法：
* 在.ssh目录中添加config文件，添加以下内容

Host **github.com //别名**

HostName github.com

PreferredAuthentications publickey

IdentityFile ~/.ssh/**id\_rsa**

Host **github1.com //别名**

HostName github.com

PreferredAuthentications publickey

IdentityFile ~/.ssh/**extra\_rsa**

* Host：使用这个别名进行关联。其规则就是：从上至下读取config的内容，在每个Host下寻找对应的私钥。
* git remote add origin1 git@**github1.com**:x/x.git