**一 shell命令回顾：**

cd 更改目录

pwd 显示当前目录

mkdir 创建一个空的文件夹

touch 创建一个空的文件

cat 读取文件内容，翻页显示

**二 git安装**

在windows下安装git需要Cygwin这样的模拟环境。msysgit已经将此模拟环境和Git打包好了。安装这个exe程序即可。

在开始菜单看到git-git bash就证明安装上了。

因Git是分布式版本控制系统，所以每个机器必须自报家门：你的名字和Email地址。

故需要配置：

$ git config --global user.name "YOUR NAME"

$ git config --global user.email "email@example.com"

注意 git config命令的--global参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

**三 创建仓库，将文件添加到版本库**

$ git init #这个是将当前目录变成git可以管理的仓库

$ git add file(或者文件夹名) #这个是将file给添加到仓库中。在commit之前可以多次add不同的文件

$ git commit -m "本次的提交说明" #将添加到仓库中的提交到仓库。

只有git add和git commit两步都操作了，才算将文件放到仓库里了。

首次提交后，master分支就会被自动创建

**四 掌握工作区状态**

$ git status # 掌握当前工作区状态，如文件已被修改过但还未储存，文件准备被提交，没有需要提交的修改，等等

$ git diff file # 查看file这个文件做了哪些修改。diff是difference的缩写

**五 版本管理**

$ git log #当前版本提交历史，显示每次commit的ID号/时间/提交说明（用于回退很好用）

$ git log --pretty='oneline' #显示当前版本的每次commit，每行显示一个提交，有ID号/提交说明

$ git reflog #命令历史，显示所有每次操作的ID号/提交说明（用于回到未来的哪个版本很好用）

git用HEAD表示当前版本，HEAD^表示上一个版本，HEAD^^表示上上一个版本，多的话，HEAD~100表示往上第100个版本

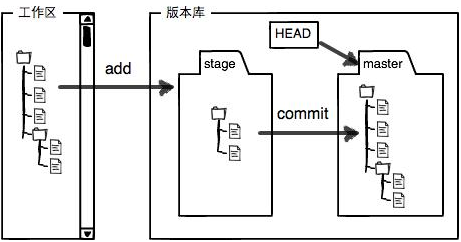
其实HEAD相当于一个指针，当回溯或更改版本时，改的是HEAD的指向

$ git reset --hard HEAD^ #回溯到上一个版本

$ git reset --hard commit\_ID号 #回溯到特定版本，若想回溯的commit ID 是d4d1940，则$ git reset --hard d4d1940

**六 工作区与暂存库**

工作区（working directory）就是电脑里能看到的目录



.git这个文件就相当于版本库，里面有：

stage（或者index）叫暂存区，

Git自动为我们创建的第一个分支master，

指向master的HEAD指针

**七 只要修改而不add，就不会commit到版本库。**

如一个文件，先修改再add，然后再修改，然后commit。结果发现第二次修改的没有被commit进去。这就是因为只要修改了，git都会记录，但修改后一定也要add并commit

Git内部存储的不只是difference，是整个文件。

**八 撤回工作区的修改**

当在工作区修改后，想撤回。用：

$ git checkout -- file #回到最近的一个add或commit命令的版本，其实是版本库里的版本替换工作区里的版本

如，在工作区修改后，还未add，使用上述命令可以将修改删掉

若在工作区修改后，已经add，则此命令不起什么作用，但想unstage掉，这时要用reset，

使用$ git reset HEAD file即可

reset可以回退版本，也可以把暂存区里的修改回退到工作区。使用HEAD时表示最新的版本

然后再使用$ git checkout -- file即可

**九 删除文件**

在工作区添加了一个文件，而且add并commit到了git库。

在工作区删除了这个文件，然后git status发现提示这个文件在工作区被删除了：

1.也想把git库的删除，则可以

$ git rm file # 记得也要commit

$ git commit -m '删除文件file' # 这时此文件才从git库删除

2.发现只是误删，想恢复回来，则

$ git checkout -- file #这样工作区的file文件就回来了，实际上就是git库里的那个版本

**十 远程仓库**

以上都是讲如何在git库里对文件操作，分布式git的魅力之一就是远程仓库。

git分布式版本管理系统，同一个git库，可以分布在不同的机器上，而且没有主次之分。

可以找一台电脑充当服务器的角色，其他人都可以推送或者提取。

服务器可以自己搭建，也可以使用GitHub，它提供git仓库托管服务。

本地git仓库和github仓库之间的传输是通过SSH加密的，故

1.在本机用户主目录下查看是否有.ssh目录，里面是否有id\_rsa和id\_rsa.pub这两个文件，没有的就：

$ ssh-keygen -t rsa -C "your@email.com"

然后一路回车即可。key可以不设置密码。

之后用户主目录可看到id\_rsa和id\_rsa.pub文件，id\_rsa是私钥，不要告诉别人；id\_rsa.pub是公钥，可以随便告诉别人。

2.登陆GitHub，点击setting->ssh->ssh key->add key，将id\_rsa.pub里的公钥粘到内容里即可

可以把你每台电脑的key都添加到GitHub中去，这样你所有电脑都可以将内容推送到GitHub了。

Git库是公开的，要么交钱私有化要么自己大件git服务器。

**十一 添加远程仓库**

在GitHub上添加一个repository，然后按照上面的提示将本地的推送过去

$ git remote add origin https://github.com/queenaLou/GitLearning.git

关联一个远程库

$git push -u origin master

这样就将本地库添加到远程库了。

注意因首次添加仓库，用了-u参数，有两个作用，1.git把本地的master分支内容推送到远程的新的master分支内容；2.把本地master分支和远程master分支关联起来，在以后的推送或拉取时简化命令。

之后只要本地做了修改，就可以用$ git push origin master同步到GitHub了。

若每次都让输入账号密码，极有可能是开始添加origin时用的HTTPS协议，需要改变登录的协议。 创建ssh的key之后用如下命令更改为ssh

$git remote rm origin

$git remote add origin "Git仓库的ssh格式地址"

$git push origin master

SSH协议比HTTPS协议要快。

**十二 从远程库克隆**

$ git clone git地址(如git@github.com:queenaLou/GitLearning.git)

**十三 创建/切换/删除分支**

git中的HEAD指针，表示当前分支的当前版本。

创建分支：$ git branch 分支名字

切换分支：$ git checkout 分支名字

创建 + 切换分支：$ git checkout -b 分支名字 #表示新创建分支，并切换到这里

查看分支：$ git branch #当前所在分支前会有\*标注

合并分支：$ git merge 分支名字 #将指定分支合并到当前分支上

删除分支：$ git branch -d 分支名字

强制删除分支:$ git branch -D 分支名字

fast forward表示快速合并，即直接将当前分支指针指向指定分支的当前提交，所以合并速度非常快

**十四 解决冲突**

当在分支future修改并add并commit后，同一文件在master也修改并add并commit了。

那么当在master上想将future的合并上去，$ git merge future时出错，提示需要手动解决冲突

这时$ git status就能查看这个文件两个分支上内容有什么不一样，

解决方法是打开文件保留要保留的删掉该删的，然后再次add 再次commit，就解决了冲突。

而之后只要future里的内容不修改，无论master再怎么commit新内容，只要$ git merge future，git就会提示：up-to-date

使用$ git log --graph命令可以查看分支合并图

**十五 使用--no-ff参数禁止快速合并模式**

在master上快速合并是将master指针指向指定分支的当前提交，不会留下合并痕迹。

而不快速合并时，会在合并时有一个commit提交，能看出合并痕迹。

当合并时不想使用fast forward模式，可以加上--no-ff参数，即不使用快速合并模式。同时也要带上-m 提交说明

如$ git merge --no-ff -m 提交说明 分支名称

**十六 保存临时修改及恢复**

$ git stash #可以将当前工作现场‘储藏’起来，等以后恢复现场后继续工作；之后若使用$ git status可以看到工作区是clean的

$ git stash list #显示所有的stash，包括其他分支的

$ git stash apply stash名 #将某个stash恢复，如$ git stash apply stash@{0}

$ git stash drop stash名 #将某个stash删除

$ git stash pop #将最近的一个stash恢复并删除

一般情况下在master分支上新建bug分支，修复后合并到master，删除bug分支；在dev分支上新建feature分支，完成后合并到dev分支，删除feature分支；当一个版本的dev分支确认稳定后，合并到master分支，继续开发。

**十七 多人协作**

$ git remote add origin 远程库地址 #添加一个远程库

$ git remote #查看远程库信息

$ git remote -v #显示更详细的信息，包括fetch和push

$ git remote rm 远程库名 #删除一个远程库

$ git push origin 分支名 #将分支上的内容推送到远程库相应分支，当远程库不存在此分支时，会自动创建

$ git checkout -b 分支名 origin/分支名 #当从远程库克隆过来时是看不到分支的，需要先添加分支(拉取最新)，最好本地和远程的分支名一样

$ git pull <origin> <branch-name> #将最新的提交从远程分支抓下来，在本地合并。然后处理冲突后再提交

$ git branch --set-upstream 分支名 origin/分支名 #将本地某分支与远程某分支建立连接

$ git branch -a #查看所有分支，包括远程库里的

**十八 标签**

其实分支和标签就是指向某个commit的指针，但分支可以移动，而标签不能移动。

标签主要用来版本管理。而带gpg签名的标签，其他开发者将不能修改你所签名的这个tag

$ git tag <tag-name> #在当前分支最近的commit处打一个标签

$ git tag <tag-name> <commit-id> #在当前分支某commit处打一个标签

$ git tag -a <tag-name> -m <tag-description> <commit-id> #在当前分支某commit处打一个有说明的标签；-a指定标签名，-m指定说明文字

$ git tag #显示所有的tag，按字母排序

$ git show <tag-name> #显示某tag的详细信息

$ git tag -d <tag-name> #删除某标签

$ git push origin <tag-name> #将某标签推送到远程库

$ git push origin --tags #将所有标签推送到远程库

当已经将标签推送到远程库，但又后悔了，想删掉远程库的那个标签，

1.从本地删除此标签：$ git tag -d <tag-name>

2.从远程库删除：$ git push origin :refs/tags/<tag-name> #注意origin后有空格

带有gpg签名的标签，需要用到gpg。

$ gpg --gen-key #生成gpg key，根据提示输入相关信息如real name，email等。我写的是queena lou,louxiaojie2009@126.com

#可以输入一个passphrase，我输入了Iccpassword1

$ gpg --list-keys #显示相关的gpg信息，如公钥信息

$ gpg --list-secrect-keys #显示相关的gpg信息，如私钥信息

然后就可以使用-s参数来签名了。

$ git tag -s <tag-name> -m <tag-description> <commit-id> #带有gpg的签名

如果还失败，说明用户不对，这时需要使用-u参数。

$ git tag -u 'real name' -s <tag-name> -m <tag-description> <commit-id> #-u参数表示可切换local的另一个用户来签名