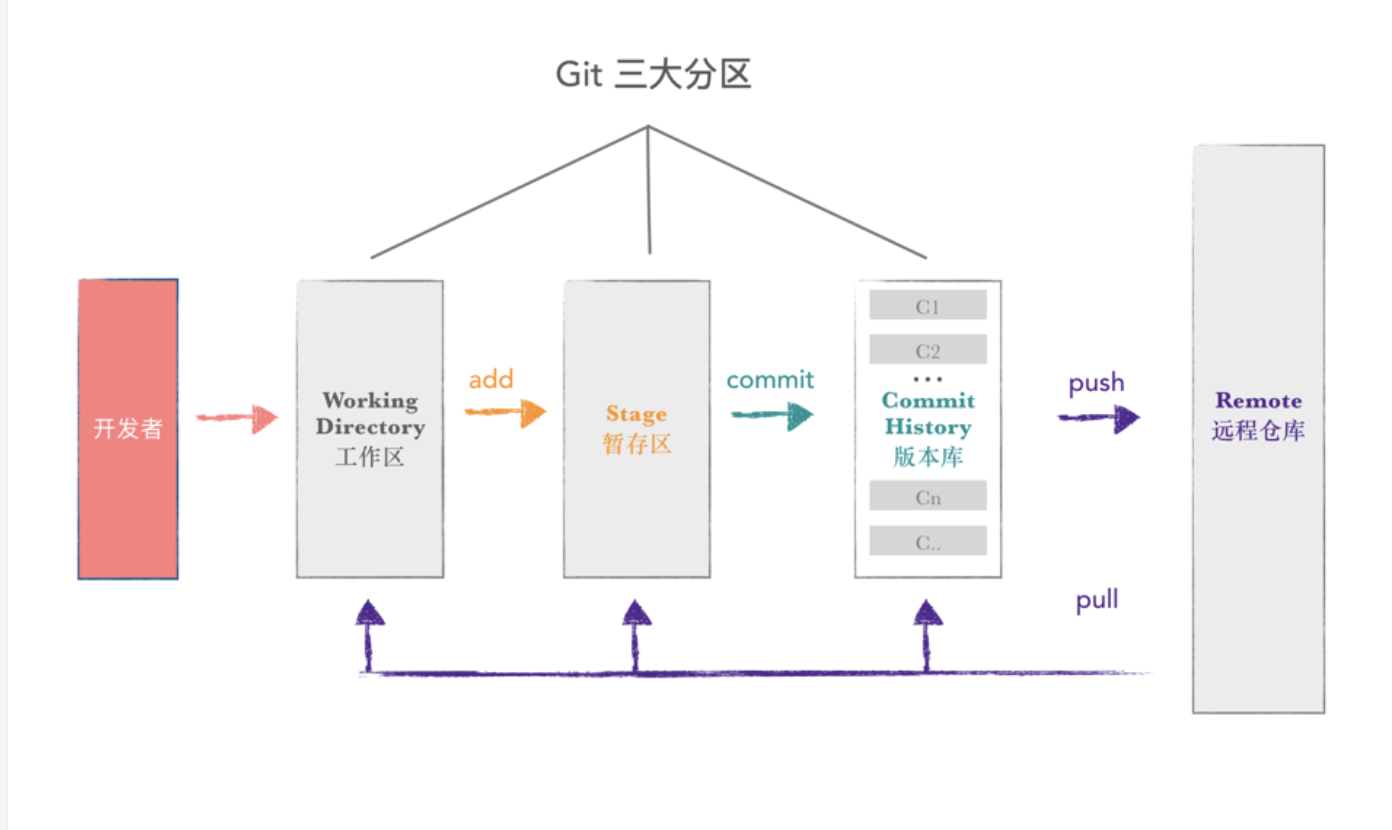
## Git技术分享

Git是目前世界上最先进的分布式版本控制系统（同一个Git仓库，可以分布到不同的机器上）。他可以自动的帮助记录每次文档的修改，还可以让他人协作编辑。极大的方便的文件的管理。分布式版本控制系统的安全性比集中式版本控制系统要高很多，因为每个人电脑里都有完整的版本库。Git跟踪并管理的是修改，而非文件。

**Git的三大分区**



工作区(working diretory) 在电脑中能看到的目录，用于修改文件

暂存区(stage) 是用来暂时存放工作区中修改的内容（重要概念，划重点）

版本库(repository)工作区中有一个隐藏目录.git不算工作区，而是git的版本库

在Git的常见命令中

git add # 将工作区的修改提交到暂存区；

git commit # 将暂存区的修改提交到当前分支；

git stash # 保存某次修改；

git pull # 从远程更新代码；

git push # 将本地代码更新到远程分支上；

git status # 查看当前仓库的状态；

git diff # 查看修改；

git log # 查看提交历史；

git revert # 回退某个修改。

**创建版本库**

**版本库又名仓库，这个仓库中的所有文件都可以被Git所管理，每个文件的修改删除Git都可以跟踪，以便任何时刻都可以追踪历史，而且还有可被还原。**

**Mkdir *xxx* 该命令用来创建一个名为xxx的目录；**

**cd xxx 该命令用于进入xxx文件夹；**

**pwd 该命令用于显示当前目录；**

**git into 通过该命令把xxx目录变成Git可以管理的仓库。**

***把文件添加到版本库*（注：所有的版本控制系统，只能跟踪文本文件的改动，但Word格式是二进制，版本控制系统无法跟踪，，所以要真正使用版本控制系统，要使用纯文本方式编写文件。）**

**编写文件（放在创建的目录或者子目录下）可以创建一个txt文件直接编辑该文件，也可以使用vi指令进入编辑状态。**

**Vi指令的使用：**

**使用 vim 加文件路径（或文件名）的模式打开文件，如果文件存在则打开现有文件，如果文件不存在则新建文件。**

**键盘输入字母i进入插入编辑模式。**

**在编辑模式下编辑文件**

**按下ESC键，退出编辑模式，切换到命令模式。**

**在命令模式下键入"ZZ"或者":wq"保存修改并且退出 vim。**

**如果只想保存文件，则键入":w"，回车后底行会提示写入操作结果，并保持停留在命令模式**

**放弃所有文件修改：按下ESC键进入命令模式，键入":q!"回车后放弃修改并退出vim。**

**放弃所有文件修改，但不退出 vi，即回退到文件打开后最后一次保存操作的状态，继续进行文件操作：按下ESC键进入命令模式，键入":e!"，回车后回到命令模式。**

**把文件放在git仓库中：**

**git add 把文件添加到仓库（该命令下没有显示就是对的）把修改等放入暂存区**

**git commit -m”本次提交的描述”** **该命令只会将git add .存入暂存区的修改内容（划重点）提交至本地仓库中，若文件未添加至暂存区（未使用git add），则提交时不会提交任何修改。**

**时光穿梭机**

**git status 用于时刻掌握仓库当前状态**

**git diff 查看difference 可以看到我们增加了什么，删去了什么。如果git status告诉你文件被修改过，你便可以具体使用git diff查看具体修改的内容。**

**版本退回**

**git log 该命令用于显示提交日志**

**git log –pretty=oneline**

**注：在git中用HEAD表示当前版本，上100个可以表示为HEAD~100**

**git reset –hard HEAD^ 该命令用于退回到上一个版本，还可以用于把暂存区修改回退到工作区。**

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是把HEAD从指向append GPL：改为指向add distributed：然后顺便把工作区的文件更新了。所以你让HEAD指向哪个版本号，你就把当前版本定位在哪。

**git reflog 该命令用来记录查看每一次命令历史，以便确定回到未来的某个版本**

**修改撤销**

**git checkout -- 文件名 （--很重要） 该命令用于把文件在工作区的修改全部撤销：**

1. **文件自修改后还没有被放到暂存区，现在撤销修改就是回到和版本库一模一样的状态；**
2. **文件已经添加到暂存区之后，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，就是让文件回到最新一次 git commit或者git add时的状态。**

**场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。**

**场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。**

**场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。**

**删除文件**

**一般情况下可以直接在文件管理器中把没用的文件删了，或者使用rm命令**

**此时，工作区和版本库就不一致了。**

**现在有两种情况：**

1. **确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit；**
2. **删错了，因为版本库里还有，所以可以很轻松地把误删的文件恢复到最新版本，使用git checkout 命令，该命令用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。**

**远程仓库**

1. **自行注册GitHub账号**
2. **由于本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以需要设置**
3. **使用Git Bush创建SSH Keys**

**cd ~/.ssh/(如果没有相应的文件夹，则执行 mkdir ./.ssh)**

**ssh-keygen -t rsa -C”** [youremail@example.com](mailto:youremail@example.com)**”**

1. **登陆GitHub，打开“Account settings”，“SSH Keys”页面，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容**

**添加远程库**

**git push 该命令用于将本地分支的更新，推送到远程主机。**

**$ git push <远程主机名> <本地分支名>:<远程分支名>**

**git push origin master**

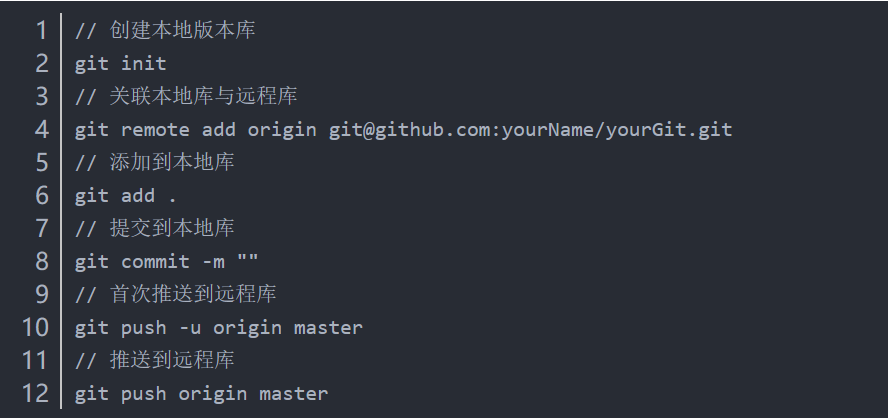
**上面命令表示，将本地的master分支推送到origin主机的master分支。如果master不存在，则会被新建。**

**如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支。**

**即git push origin :master 等同于 git push origin --delete master**

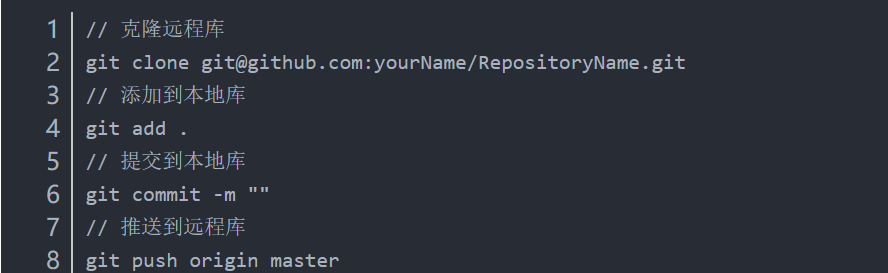
**由于远程库是空的，我们第一次推送master分支时，如果加上了-u参数，Git就不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。**

**git remote add origin** [git@server-name:path/repo-name.git](mailto:git@server-name:path/repo-name.git) **该命令用于关联一个远程库。**



**从远程库克隆**

**git clone 可以使用该命令克隆，但是首先必须要知道仓库的地址。**

****

**ls 指令**

**ls -1：每行列出一个文件，即以单列形式列出**

**ls -a：列出所有文件，包括隐藏文件**

**ls -la：所有文件的长格式列表（含权限、所有权、大小和修改日期）**

**ls -lh：使用人可读单位（KB，MB，GB）显示大小的长格式列表**

**ls -lS：按大小排序的长格式列表（降序）**

**创建与合并分支**

**一开始的时候，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每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长。**

**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，不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变。**



**假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并**

**git checkout -b <name> 该命令加上-b参数表示创建并切换**

**相当于git branch dev 和git checkout dev 两条指令**

**git branch 该指令用于显示当前分支**

**git checkout 该命令用于切换分支**

**git merge 该命令用于合并指定分支到当前分支**

**git branch -d <name> 该指令用于删除分支**

**git branch -D <name> 该命令可用于强行删除没有合并过的分支**

**切换分支还可用switch**

**git switch -c <name> 该分支用于创建并切换到新的分支**

**git switch 该命令用于切换到已有分支**

**分支管理策略**

**git merge --no-ff -m "merge with no-ff" dev**

**注意--no-ff参数，表示禁用Fast forward**

**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。**

**Bug分支**

**git stash 该命令用于把当前工作现场“储藏”起来，等以后恢复现场后继续工作**

**git stash list 该命令用于查看之前的工作现场去哪里了**

**工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：**

**git stash apply 该命令用于恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除**

**git stash pop 该命令在恢复的同时把stash内容也删了**

**git cherry-pick 该命令用于复制一个特定的提交到当前分支**

**多人协作**

**git remote 该命令用于查看远程库信息**

**git remote -v 该命令用于显示更详细的信息**

**git push <远程主机名> <本地分支名>:<远程分支名> 该命令用于把把该分支上的所有本地提交推送到远程库**

**如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支。**

**如果当前分支与远程分支之间存在追踪关系，则本地分支和远程分支都可以省略**

**如果当前分支只有一个追踪分支，那么主机名都可以省略，可直接push**

**如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机，这样后面就可以不加任何参数使用git push**

**git pull <远程主机名> <远程分支名>:<本地分支名> 该命令用于取回远程主机某个分支的更新，再与本地的指定分支合并（如果远程分支要与当前分支合并，则冒号后面的部分可以省略）**

**git pull = git fetch + git merge git pull使用给定的参数运行git fetch，并调用git merge将检索到的分支头合并到当前分支中**

**多人协作的工作模式通常是这样：**

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

**2.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；**

**3.如果合并有冲突，则解决冲突，并在本地提交；**

**4.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**

**Rebase**

**git rebase 该命令用于将分叉的提交历史整理成一条直线看上去更直观，使的我们在查看历史提交的变化时更容易（但是缺点是本地的分叉提交已被修改过）**

**标签管理**

**标签其实是指向某个commit的指针，tag就是一个让人容易记住的有意义的名字，它跟某个commit绑在一起。**

**创建标签**

**git tag <name> 该命令用于创建一个新的标签**

**git tag 该命令用于查看标签**

**git log --pretty=oneline --abbrev-commit 该命令用于找到历史提交的commit id**

**git show <tagname> 该命令用于显示标签信息**

**git tag -a v0.1 -m "version 0.1 released" 该命令用于创建带有说明的标签，用-a指定标签名，-m指定说明文字**

**注：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。**

**操作标签**

**git tag -d v0.1 该命令用于删除标签**

**git push origin <tagname> 使用该命令将标签推到远程仓库**

**git push origin –tags 该命令用于一次性推送全部尚未推送到远程的本地标签**

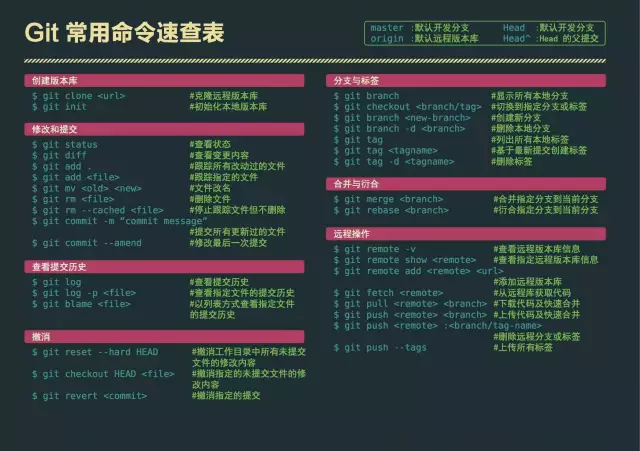
**注：如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除使用git tag -d v0.9 然后，从远程删除。删除命令也是push git push origin :refs/tags/v0.9**

**自定义git**

**git config --global color.ui true 使用该命令Git会适当地显示不同的颜色，如给文件名标上颜色**

**git config --global alias.st status 使用该命令后敲git st就表示git status**

**--global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用**

****