Git学习笔记

所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

千万不要使用Windows自带的**记事本**编辑任何文本文件，因为Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符。

git log --pretty=oneline,在一行显示改动。

Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

git reflog查看命令历史，以便确定要回到未来的哪个版本。

Git跟踪并管理的是修改，而非文件。

每次修改，如果不add到暂存区，那就不会加入到commit中。

直接丢弃工作区的修改时，用命令git checkout -- file。让这个文件回到最近一次git commit或git add时的状态。让这个文件回到最近一次git commit或git add时的状态。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。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，前提是没有推送到远程库。

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令。

删除文件：

要从版本库中删除该文件，那就用命令git rm删掉，并且git commit。

删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：$ git checkout -- test.txt。git checkout其实是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”。

设置SSH钥匙：

生成密匙(git bash)：ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

关联远程库：

git remote add origin [git@github.com:michaelliao/learngit.git](mailto:git@github.com:michaelliao/learngit.git) ： 需要在对应的learngit库下面使用Git Bash使用。

远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

git push -u origin master

git push命令，实际上是把当前分支master推送到远程。

由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git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 push origin master就可以把本地master分支的最新修改推送至GitHub。

Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。

创建分支：

git checkout -b dev相当于 git branch dev; git checkout dev

在master上合并dev， git merge dev

删除dev分支：git branch -d dev

用git log --graph命令可以看到分支合并图。

分支策略：

master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

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。

Git还提供了一个stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作，git stash list命令查看stash，

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；另一种方式是用git stash pop，恢复的同时把stash内容也删了。git stash apply stash@{0}指定恢复的stash。

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

git remote -v : 显示详细的远程库信息；显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址。

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上，如git push origin master或git push origin dev.

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 branch-name origin/branch-name.

git tag <name>用于新建一个标签，默认为HEAD，也可以指定一个commit id

git tag -a <tagname> -m "blablabla..."可以指定标签信息

git tag -s <tagname> -m "blablabla..."可以用PGP签名标签

git tag可以查看所有标签

git show <tagname>可以看到说明文字和签名信息

git push origin <tagname>可以推送一个本地标签

git push origin --tags可以推送全部未推送过的本地标签

git tag -d <tagname>可以删除一个本地标签

git push origin :refs/tags/<tagname>可以删除一个远程标签

删除github远程连接：git remote rm origin。

忽略某些文件时，需要编写.gitignore

.gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理

.gitignore写得有问题，需要找出来到底哪个规则写错了，可以用git check-ignore命令检查，如git check-ignore -v App.class

设置别名：将status设置别名为st

git config --global alias.st status

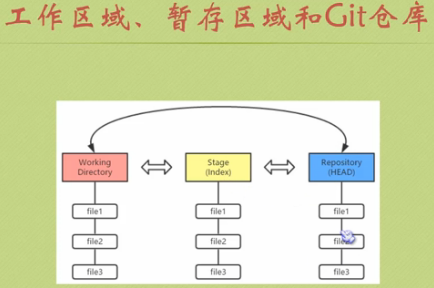
--global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用

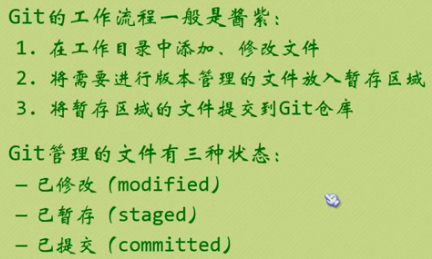
配置Git的时候，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用；每个仓库的Git配置文件都放在.git/config文件中。

当前用户的Git配置文件放在用户主目录下的一个隐藏文件.gitconfig中。

设置Git:

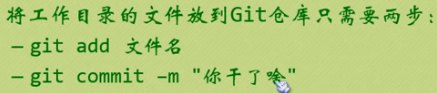






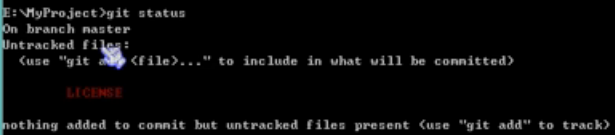
在dos命令中，输入git init初始化git仓库。

Git通过.git文件来管理项目文件。

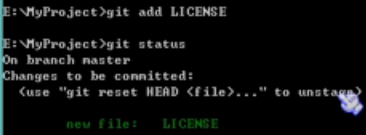




查看当前git文件夹中的文件状态，此信息表示在git的master分支中，没有需要提交的文件。

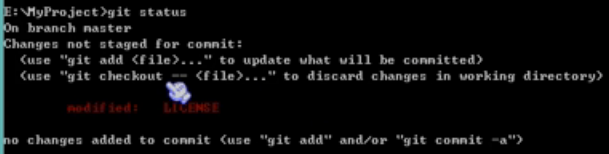


在git目录中含有未被追踪的文件，git建议将其添加为可追踪文件并可提交。



添加后可使用git reset文件重新恢复为原暂存区域。可通过指定file名称指定恢复的文件，也可只用HEAD不指定名称，将所有文件恢复问之前的状态。

有文件修改后：

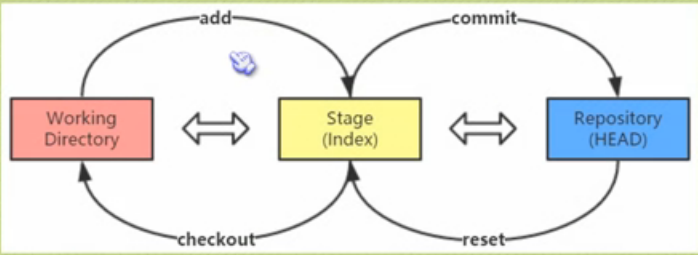


提示有修改的文件，git给出两个建议：

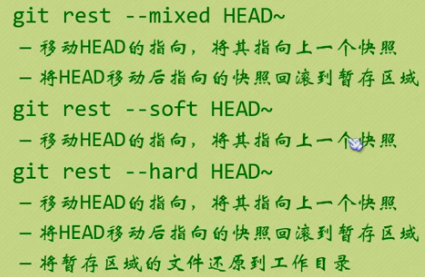
一是添加提交，二是使用checkout返回修改前的文件。

使用checkout是文件名前有空格。

git log：查看历史提交记录。

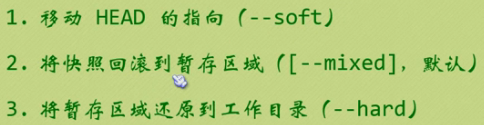


git reset HEAD~:返回git的上一个快照，可使用多个~表示上一个；也可使用数字表示向上指向的数量。



使用soft时不会修改再存区域的内容，撤销上一次的commit命令。

使用hard时会将工作目录最新的文件覆盖掉。



可使用指定的id号回滚到指定文件：如git reset id号

回滚个别文件，这是head的指针位置不变。

比较暂存区域和工作目录：git diff



比较两个历史快照

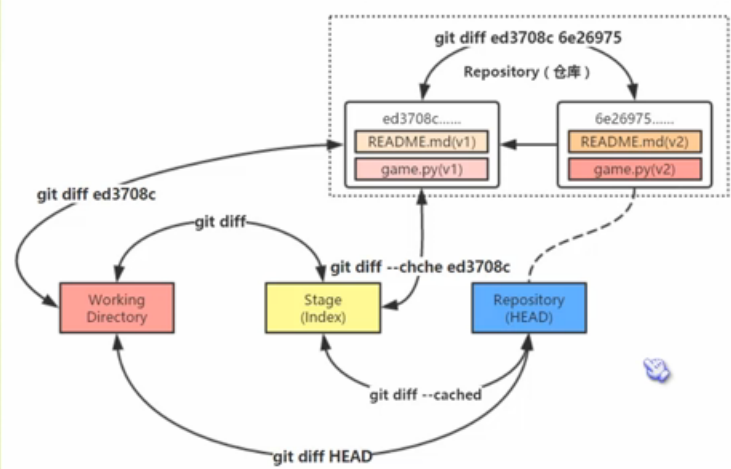


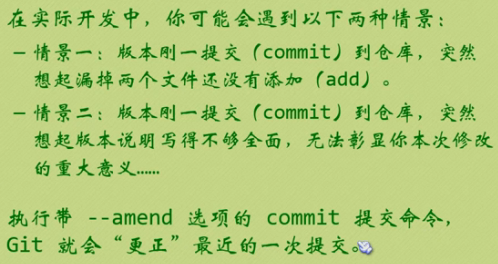
比较工作目录和仓库中的快照



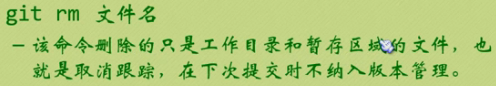
比较暂存区域和仓库的快照



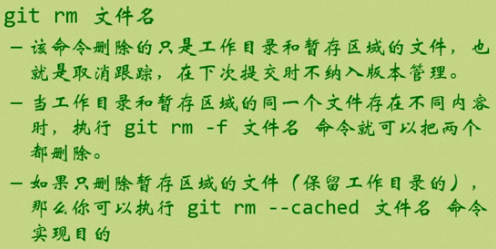




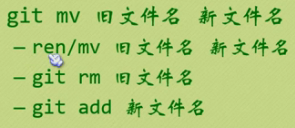
删除文件



此时仓库中的快照没有删除可使用git reset –soft HEAD~改变head指针位置。



重命名文件

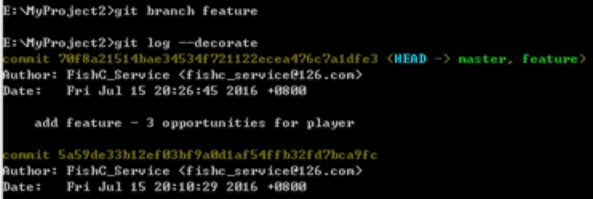


Git分支：



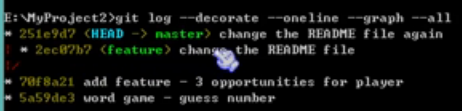
创建分支：添加一个指向快照的指针。





默认分支为master。

切换分支：git checkout 分支名



以图形的形式显示分支。--oneline是以精简都得方式显示分支信息。

Git是按照时间的先后创建的。

实际开发中的分支



合并分支：



删除分支：

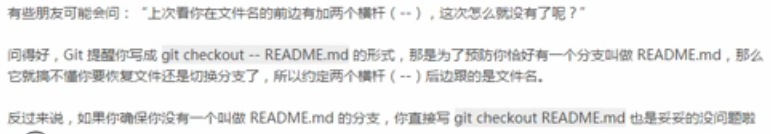


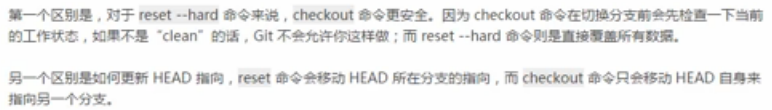
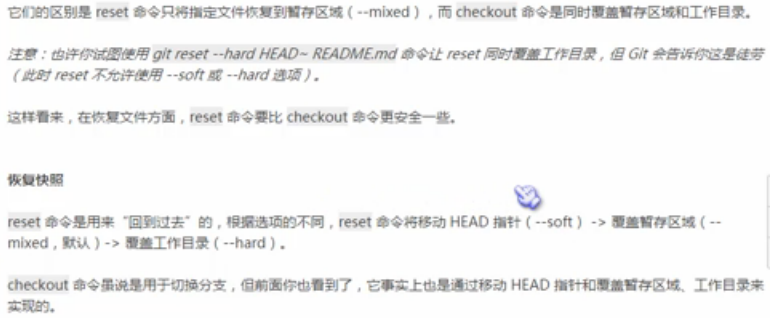
使用匿名分支：git checkout 不添加分支名，可以做一些是实验性的操作而不会产生不好的影响。

命令checkout的作用：从历史快照或缓存区域拷贝文件到工作区；或切换分支。

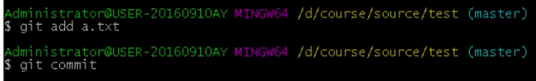


如果命令没有指定具体的快照ID，则将从暂存区域恢复指定文件到工作区。





使用git clone(使用http的地址)是从远程库中项目拉取到本地



使用git add将文件添加在暂存区内，

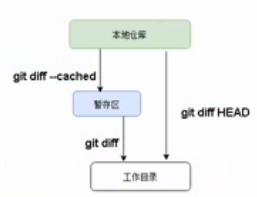
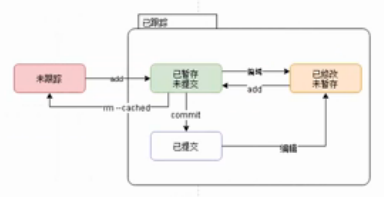
git commit提交到本地库中(即git文件夹中)。

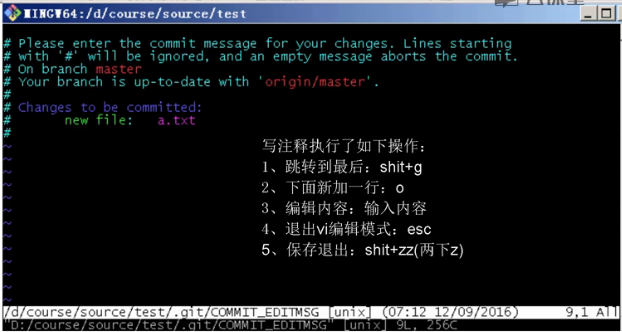
git diff 比较的是工作目录和暂存区的区别

git diff HEAD 比较的是工作目录和本地仓库的区别。

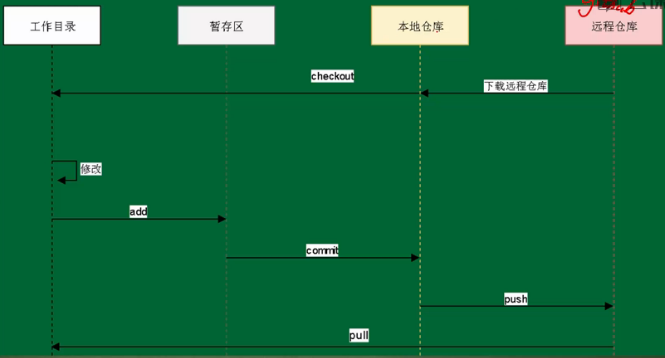
git diff –cached 比较暂存区和本地仓库的区别。

git rm –cached 文件名 从暂存区移出

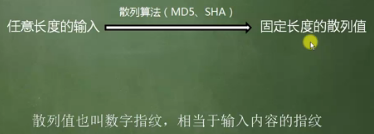




使用git push命令时才会把文件推到远程库中，使用git pull更新本地库。



深入Git的存储：



散列值不可逆，不可反向推出。

git ls-files –s:列出暂存区中的内容。

暂存区之存储文件的名称和哈希值。

暂存区就是.git文件中的index文件



Git分支：

git log:查看提交记录

git log –oneline

git branch : 查看分支

git branch dev：创建dev分支

git branch -b 分支名：新建并切换当前分支

cat .git/HEAD: 查看当前所在分支

git checkout master:将当前分支切换到master上

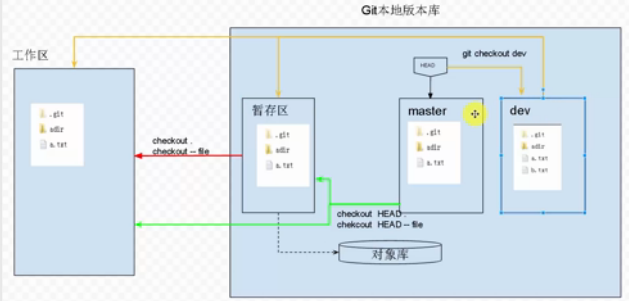
git log --oneline –decorate:head指向当前分支

可添加—allow-empty参数进行空的提交：git commit –allow-empty -m ‘’

git branch 分支名 master：给予master创建分支

git log --oneline --all --decorate --graph

git config alias.logg “log --oneline --all --decorate --graph”:给命令起别名



git ls-tree

git cat-file p :b 查看暂存区中b文件内容

git cat-file p b:b 查看仓库中b文件内容

git checkout .用暂存区的内容覆盖所有工作区文件

git checkout --fileName：暂存区文件覆盖特定文件内容

Stash：

git stash：可将工作区和暂存区未commit的内容存放在一个位置。

git stash list:查看stash列表

git stash show -p stash@[0]:查看标识未0的stash内容

git stash pop：将stash列表中的第一个数据放入文件。(stash列表的内容会被删除)

git stash apply stash@[0]: 将stash列表中的第一个数据放入文件。(stash列表的内容不会被删除)

git stash drop stash@[0]:删除stash列表内容

分之合并：

先切换到相应的分支，如master

再使用git merge b

分支合并冲突：

git mergetool

配置mergetool：git config merge.tool bc

设置程序路径： git config merge.tool bc.path “”

充值分支：git reset –hard 哈希值

rebase的使用：git rebase –-onto newrebase start branch

git rebase –onto master start dev

远程分支：git branch -a查看所有分支