GIT版本管理

git目录下.git/config 为该git的配置信息，里面有各种信息

公司git网址：116.62.118.242

账号：liuzhaochen

密码：123qqq...A

# Git 工作区、暂存区和版本库

## 基本概念

我们先来理解下Git 工作区、暂存区和版本库概念

* **工作区：**就是你在电脑里能看到的目录。
* **暂存区：**英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
* **版本库：**工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

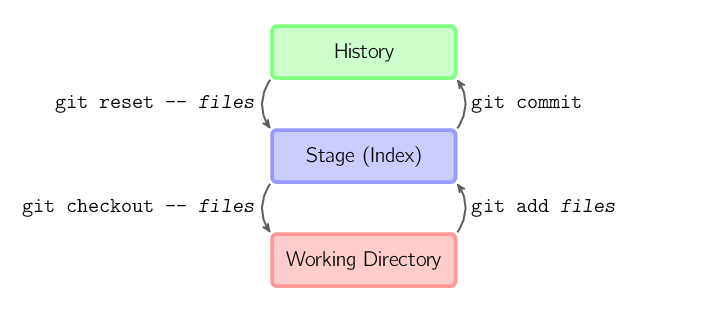
当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。即回到git add . 之前的状态

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。也会回到git add . 之前的状态，但工作区的还是有修改的，只是未加进暂存区。

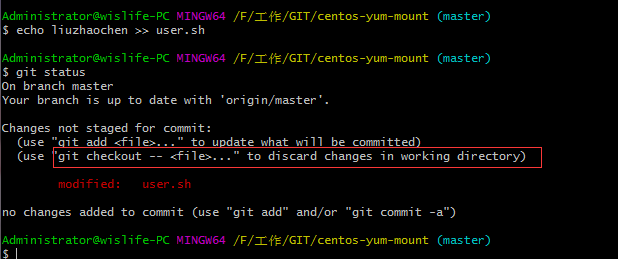
当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动，即还原工作区

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。



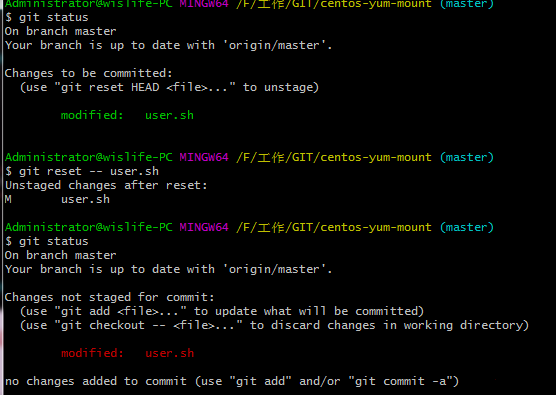
上面的四条命令在工作目录、暂存目录(也叫做索引)和仓库之间复制文件。

* git add files  #把当前文件放入暂存区域。
* git commit  #给暂存区域生成快照并提交至仓库。
* git reset -- files  #用来撤销最后一次git add files，你也可以用”git reset .”撤销所有暂存区域文件。回到git add 之前的状态
* git checkout -- files  或者git checkout . #把文件从暂存区域复制到工作目录，用来丢弃本地修改。还未执行”git add .” 命令之前可运行该命令进行还原，且git status会有提示，若是“git checkout .”则是还原全部文件



Git reset HEAD <file> 或者 git reset -- filename  或者 git reset . #若已执行了git add，可用来撤销最后一次git add files，你也可以用”git reset HEAD .”撤销所有暂存区域文件，用git status查看，已经回到git add前的状态，然后可用“git checkout .”还原工作区。

#若执行了git add . ，则需要用"git checkout HEAD .



git clone 时，可以所用不同的协议，包括 ssh, git, https 等，其中最常用的是 ssh，因为速度较快，还可以配置公钥免输入密码。各种写法如下：

git clone git@github.com:fsliurujie/test.git --SSH协议

git clone git://github.com/fsliurujie/test.git --GIT协议

git clone https://github.com/fsliurujie/test.git --HTTPS协议

克隆远程服务器test分支下来，使用特定的用户和密码：git clone -b test http://liuzhaochen:123qqq...A@116.62.118.242/prj/gx\_baobiao.git

克隆远程github官网的一个git版本，其中账号密码加入后，push时不需要再输入账号密码：

git clone <https://super-season:2012551208liu@github.com/super-season/centos-yum-mount.git>

191 git checkout -b produce origin/produce

192 git log

193 git branch

194 git checkout test

195 git log

196 git checkout produce

197 git log

198 git branch

199 git merge test

200 git log

201 git reflog

202 git tag -a V20190817 -m " 修改数据处理bug" b7ef29a #增加一个标签，最后面的为版本号，给版本号绑定上标签

203 git tag #查看所有标签

204 git show V20190817 #查看标签信息

205 git branch --set-upstream-to=origin/produce produce

206 git remote -v

207 git remote

208 git branch -a

209 git remote show origin

210 git branch -v

211 git branch -vv

212 git push

213 git push origin --tags （把本地所有标签上传至远程）

git push -u origin newproduct (把本地分支与远程分支绑定）

git push --set-upstream origin newproduct

或者在.git/config文件中更改配置也行

git本地新建一个分支后，必须要做远程分支关联。如果没有关联， git 会在下面的操作中提示你显示的添加关联。关联目的是如果在本地分支下操作： git pull, git push ，不需要指定在命令行指定远程的分支． 推送到远程分支后， 你只要没有显示指定，git pull 的时候，就会提示你。

解决办法是使用命令git branch --set-upstream ;实例如下，其中debug为创建的分支

git branch --set-upstream debug origin/debug

配置本地分支与远程分支的三种方法：

1.检出时建立关联关系：git checkout -b dev origin/dev

当我们检查时，git会自动为我们检出的分支和远程分支建立关联关系；

2.提交时配置关联关系：

git push -u origin <remote\_branch>或git push --set-upstream origin <remote\_branch>

3.更改git/config文件：git branch --set-upstream-to=<remote\_branch>

git push -u origin newproduct

git push --set-upstream origin newproduct

那么如何查看已经配置分支关联信息呢，通过下述三条命令均可：

git branch -vv

git remote show origin

cat .git/config

关联分支：在git中表现为upstream,无论是使用push -u 或是 git branch --set-upstream-to方法，均会将这种对应关系写入.git/config配置文件，如果一个本地分支没有关联分支，则无法执行 git push 或 git pull指令；

全局配置中，有一个push.default属性，其决定了git push操作的默认行为。

push.default 有以下几个可选值： nothing, current, upstream, simple, matching

nothing: 直接push会出错，需要显式的指出推送的远程分支，例如:git push origin <remote\_branch>；

current: 推送时只会推送当前所在的分支到远程同名分支，如果远程分支不存在相应的同名分支，则创建该分支；

upstream: 推送当前分支到它的upstream分支上，这个模式只适用于推送到与拉取数据相同的仓库(比如central workflow)；

simple(默认): simple和upstream是相似的，只有一点不同，simple必须保证本地分支和它的远程 upstream分支同名，否则会拒绝push操作。

matching:推送本地和远程都存在的同名分支。

git目录下.git/config 为该git的配置信息，里面有各种信息

git clone -b test http://liuzhaochen:123qqq...A@116.62.118.242/prj/gx\_baobiao.git //克隆test分支，并附上用户名和密码，以后就不需要输入密码

Git命令很多都需要在git仓库中运行

常用命令：

Git config --list #查看配置信息

Git init [目录] #把某目录初始化为一个git仓库，若省略，则表示当前目录，初始化后，该目录下会生产一个.git文件

git branch //查看分支情况

git branch name1 //创建name1分支

git branch -d name1 //删除name1分支

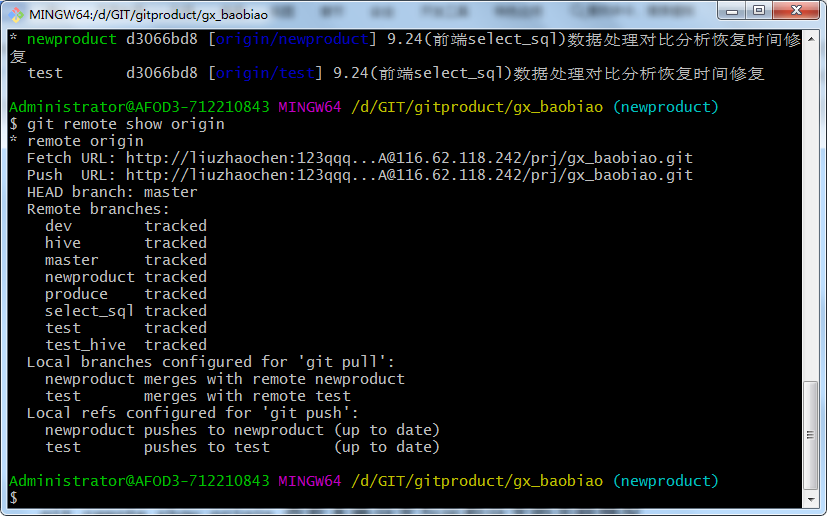
Git branch -m newname #把当前分支更改为新名字

git checkout name1 //切换到name1分支上

git branch -a //查看所有分支，包括远程分支

git branch -vv //查看本地分支与远程分支的关联情况和版本信息

**git remote show origin** //查看本地分支与远程分支的关联情况



git pull //从远程分支中拉取更新最新版本到本地分支,前提是建立了关联联系

git pull origin name2 //指定远程分支name2同步至当前分支

git pull origin name2:name2(远程分支：本地分支) //同步远程name2分支和本地name2分支，自身可在其它分支如name1上

git push //把本地git分支上传到远程分支，前提是与远程分支做好关联

git push origin HEAD:master //本地分支同步到远程master分支，两分支名不一样

git branch 查看分支情况

git branch name1 查看name1分支

git branch -m oldname newname //把分支名从oldname改为newname

git branch -d name1 删除name1分支

git branch -a 查看所有分支，包括远程分支

git branch -vv 产看本地分支与远程分支的关联情况和版本信息

git remote show origin //查看本地分支与远程分支的关联情况

git checkout name1 切换到name1分支上

git checkout -b name2 //创建并切换至分支name2

git checkout -b name2 origin/name2 //创建分支name2，并与远程分支name2关联

git push -u origin name2 //把当前分支与远程分支name2做关联，并提交新版本至远程分支，以后直接用git push就会直接上传到name2分支，关联信息会写入.git/config配置文件中

git push --set-upstream origin name2 //同上

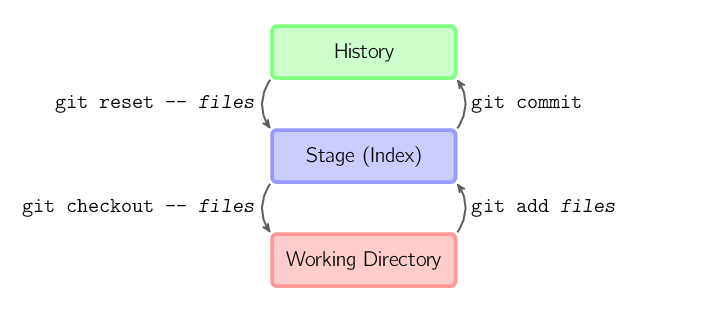
git branch --set-upstream-to=origin/name2(远程分支) name(本地分支) //把远程分支name2与本地分支name关联，关联信息会写入.git/config配置文件中

git remote add origin name2 //把远程分支name2与本地分支做关联

git status //查看当前分区工作区那些文件修改了，有没有上传暂存区（add）的活着仓库的（commit）

git add .(点代表所有修改，或者特定文件名) //把修改放进暂存区

git commit -m "描述信息" //上传版本，会自动生成新版本号



git merge dev //把dev分支合并到当前分支，Git合并分支也很快！就改改指针，工作区内容也不变

git merge --no-ff -m "merge with no-ff" dev //准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward

git commit、git push、git pull、 git fetch、git merge 的含义与区别

* git commit：是将本地修改过的文件提交到本地库中；
* git push：是将本地库中的最新信息发送给远程库；
* git pull：是从远程获取最新版本到本地，并自动merge；
* git fetch：是从远程获取最新版本到本地，不会自动merge；
* git merge：是用于从指定的commit(s)合并到当前分支，用来合并两个分支；

$ git merge -b // 指将 b 分支合并到当前分支

**git pull** 相当于 **git fetch + git merge**。

git log --graph //查看日志，并附带分支之间的合成图

Git log --pretty=oneline //查看commit日志

git log --pretty=oneline --abbrev-commit //查看commit日志,commit id是简写

标签管理

**发布一个版本时，我们通常先在版本库中打一个标签（tag），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。**

**Git的标签虽然是版本库的快照，但其实它就是指向某个commit id的指针，这样就不用记负载的commit id号，只记录版本号即可，而且创建和删除标签都是瞬间完成的。**

**Git tag #查看所有标签**

**Git tag V1321(标签名) f53c6223(分支指针id号，git reflog 第一列的字符串） #给特定的分支指针id号打标签，以后还原可根据标签来还原**

**Git tag -a V13131(标签名，通常是版本号) -m “版本说明，更新了什么” f53c6223（分支id号） #创建带有说明信息的标签**

**Git show V1321（标签名） #查看标签的详细信息**

**Git tag -d V1321(标签名) #删除标签**

**Git push origin V1321(标签名) #把本地某标签上传至远程git**

**Git push origin --tags #把本地所有标签上传至远程git，可以重复上传**

### **git rm**

如果只是简单地从工作目录中手工删除文件，运行 **git status** 时就会在 **Changes not staged for commit** 的提示。

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除，然后提交。可以用以下命令完成此项工作

git rm <file>

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 **-f**

git rm -f <file>

如果把文件从暂存区域移除，但仍然希望保留在当前工作目录中，换句话说，仅是从跟踪清单中删除，使用 **--cached** 选项即可，回到git add之前的状态，但工作区有修改的话，git status还能看到没添加进暂存区

git rm --cached <file>

如我们删除 hello.php文件：

$ git rm hello.php

rm 'hello.php'

$ ls

README

不从工作区中删除文件：

$ git rm --cached README

rm 'README'

$ ls

README

可以递归删除，即如果后面跟的是一个目录做为参数，则会递归删除整个目录中的所有子目录和文件：

git rm –r \*

进入某个目录中，执行此语句，会删除该目录下的所有文件和子目录。

202 git tag -a V20190817 -m " 修改数据处理bug" b7ef29a

203 git tag

204 git show V20190817

205 git branch --set-upstream-to=origin/produce produce

206 git remote -v

207 git remote

208 git branch -a

209 git remote show origin

210 git branch -v

211 git branch -vv

212 git push

213 git push origin --tags （把本地所有标签上传至远程）

在 Linux 系统中，可以通过查看 /proc/sys/vm/swappiness 内容的值来确定系统对 SWAP 分区的使用原则。当swappiness 内容的值为 0 时，表示最大限度地使用物理内存，物理内存使用完毕后，才会使用 SWAP 分区。当swappiness 内容的值为 100 时，表示积极地使用 SWAP 分区，并且把内存中的数据及时地置换到 SWAP 分区。 默认值为 0，表示需要在物理内存使用完毕后才会使用 SWAP 分区

版本回退：（修改指针指向的版本号）

git re**set** --hard HEAD^ #回退上一个版本

git re**set** --hard 1094a（版本id号） #回退到特定版本号