git pull origin dev (dev是分支名称) 下拉指定分支文件

小技巧：

清除屏幕： clear

粘贴用: Shift + Insert

返回操作区域 q

查看文件： ll 或者 ls 查看

cd命令进入目录 比如:c/github5（空文件夹）

第一种： cd c: cd github5

第二种： cd c:\github5\2016-11-16

小技巧：可以通过 tab 键自动补全

cd .. 返回上一层目录

在工作区创建一个文件去查看状态（git status） 显示红色

常用命令：

git clone 后面写地址 下拉远程文件

git clone 远程地址。

cd .. 返回上一层目录

git status 查看有哪些文件

git add 文件名 放到暂存区

git add . 把所有文件添加到暂存区

git commit -m "名字" 放到版本区

git commit -a -m "注释" 快速版本控制：

git push origin master 同步到远程仓库啊 最新版本 （文件夹的）

对比：

查看版本（如果出现：号 按q键返回） git log

查看暂存区与工作区的差异： git diff

查看暂存区与版本区的差异： git diff -–cached

查看本地与版本区的差异： git diff master

撤销:

git reset HEAD 文件名 暂存区撤销到本地区

git checkout –-（空格）文件名 工作区代码还原成 暂存区 或版本区

删除:

git rm 文件名 本地已经删除 现在是删除暂存区的

git rm –f 文件名 删除本地和暂存区的 一次性删除

git rm –cached 文件名 只删除暂存区不删除本地

恢复(回滚)：操作版本区:

回滚指定文件： git checkout 文件名

回滚指定版本文件： git checkout 版本号（5位） 文件名

回滚所有整个版本的文件： git reset --hard 版本号

查看已经操作过的历史记录 git reflog

回滚当前历史记录的上一个记录 git reset --hard HEAD^

回滚当前历史记录的上几个记录 git reset --hard HEAD~2

撤销：

暂存区的控制撤销到工作区 git reset HEAD 文件名

工作区代码还原成暂存区或版本区 git checkout -- 文件名

如果有2个以上文件，一个提交到版本库了，

另一个忘记提交，可以先将没提交的文件拉到缓存区，

然后通过

git commit -m "注释" --amend 撤销回来，

最后自动一次性提交暂存区中的所有文件，

git log查看是否提交成功

同步远程仓库：

git push origin master 同步到远程仓库

origin : 远程仓库的名字（默认的）

git remote : 查看远程仓库的名字

git remote -v 可以查看拉取下来的地址和推送的地址是否保持一致。

git远程创建文件夹：

在github上，有个加号，第一个就是新建项目

填写项目名称、项目的描述、（勾选）README.md -> 点击创建按钮

设置多人开发权限:

1.先进入github项目

2.点击 加号 New collaborator

3.加开发伙伴

4.等待伙伴确认

5.有权限的伙伴,克隆地址

6.进行开发

如果有冲突的情况：

1.git fetch 查看冲突先将远程仓库的代码拉取下来

2.git diff master origin/master 查看版本库与远程的差异（冲突）

3.git merge origin/master 版本库与远程的代码进行合并

4.手动修改冲突

5.修改完之后，版本控制 ->再 push

如果没有权限还要参与别人的项目：

1.fork下别的人项目

2.自己仓库会多一份fork到的项目

3.git clone 项目

4.该修改就修改，该push就push

5.创建一个new pull request

6.等待别人回复

当收到别人发来的修改请求：

1.先查看别人修改了什么？

2.如果合适就合并

点击 merge pull request按钮进行合并

分支： 默认为matser分支

查看分支： git branch (当前分支会有\*号)

创建分支： git branch 分支名

切换分支: git checkout 分支名

快速创建并且切换 git checkout -b 新的分知名

git branch -d 分支名 （合并完分支后删除）

git branch -D 分支名 （强制删除没有合并的分支）

git merge 分支名 （合并分支）

git branch --no-merged （查看没有合并的分支）

查看当前主干上有多少个分支已经合并: git branch --merged

mast分支的操作：

git push origin master

git branch 分支名 创建分支

git branch 查看分支当前分支会有\*号

git checkout 分知名 切换分支

git checkout –b 新分支名 并切换到新分支（快速创建切换）

git branch –d 分支名 删除分支

git branch –D 强制删除 没有合并时

合并分支 先进入主分支

git merge 分知名 合并分支

Git常用命令

查看、添加、提交、删除、找回，重置修改文件

git help <command> # 显示command的help

git show # 显示某次提交的内容 git show $id

git co -- <file> # 抛弃工作区修改

git co . # 抛弃工作区修改

git add <file> # 将工作文件修改提交到本地暂存区

git add . # 将所有修改过的工作文件提交暂存区

git rm <file> # 从版本库中删除文件

git rm <file> --cached # 从版本库中删除文件，但不删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset -- . # 从暂存区恢复到工作文件

git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git ci <file> git ci . git ci -a # 将git add, git rm和git ci等操作都合并在一起做 git ci -am "some comments"

git ci --amend # 修改最后一次提交记录

git revert <$id> # 恢复某次提交的状态，恢复动作本身也创建次提交对象

git revert HEAD # 恢复最后一次提交的状态

查看文件diff

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <id1><id1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

查看提交记录

git log git log <file> # 查看该文件每次提交记录

git log -p <file> # 查看每次详细修改内容的diff

git log -p -2 # 查看最近两次详细修改内容的diff

git log --stat #查看提交统计信息

tig

Mac上可以使用tig代替diff和log，brew install tig

Git 本地分支管理

查看、切换、创建和删除分支

git br -r # 查看远程分支

git br <new\_branch> # 创建新的分支

git br -v # 查看各个分支最后提交信息

git br --merged # 查看已经被合并到当前分支的分支

git br --no-merged # 查看尚未被合并到当前分支的分支

git co <branch> # 切换到某个分支

git co -b <new\_branch> # 创建新的分支，并且切换过去

git co -b <new\_branch> <branch> # 基于branch创建新的new\_branch

git co $id # 把某次历史提交记录checkout出来，但无分支信息，切换到其他分支会自动删除

git co $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支

git br -d <branch> # 删除某个分支

git br -D <branch> # 强制删除某个分支 (未被合并的分支被删除的时候需要强制)

分支合并和rebase

git merge <branch> # 将branch分支合并到当前分支

git merge origin/master --no-ff # 不要Fast-Foward合并，这样可以生成merge提交

git rebase master <branch> # 将master rebase到branch，相当于： git co <branch> && git rebase master && git co master && git merge <branch>

Git补丁管理(方便在多台机器上开发同步时用)

git diff > ../sync.patch # 生成补丁

git apply ../sync.patch # 打补丁

git apply --check ../sync.patch #测试补丁能否成功

Git暂存管理

git stash # 暂存

git stash list # 列所有stash

git stash apply # 恢复暂存的内容

git stash drop # 删除暂存区

Git远程分支管理

git pull # 抓取远程仓库所有分支更新并合并到本地

git pull --no-ff # 抓取远程仓库所有分支更新并合并到本地，不要快进合并

git fetch origin # 抓取远程仓库更新

git merge origin/master # 将远程主分支合并到本地当前分支

git co --track origin/branch # 跟踪某个远程分支创建相应的本地分支

git co -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上

git push # push所有分支

git push origin master # 将本地主分支推到远程主分支

git push -u origin master # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

Git远程仓库管理

GitHub

git remote -v # 查看远程服务器地址和仓库名称

git remote show origin # 查看远程服务器仓库状态

git remote add origin git@ github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址(用于修改远程仓库地址) git remote rm <repository> # 删除远程仓库

创建远程仓库

git clone --bare robbin\_site robbin\_site.git # 用带版本的项目创建纯版本仓库

scp -r my\_project.git git@ git.csdn.net:~ # 将纯仓库上传到服务器上

mkdir robbin\_site.git && cd robbin\_site.git && git --bare init # 在服务器创建纯仓库

git remote add origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址

git push -u origin master # 客户端首次提交

git push -u origin develop # 首次将本地develop分支提交到远程develop分支，并且track

git remote set-head origin master # 设置远程仓库的HEAD指向master分支

也可以命令设置跟踪远程库和本地库

git branch --set-upstream master origin/master

git branch --set-upstream develop origin/develop