**Git常用命令**

**Git跟踪管理的是修改而并非文件**

**git log --oneline(查看历史消息简介版)**

**git log --oneline --graph(查看历史消息简介版，用图形显示)**

**git log --reverse --oneline(逆序显示)**

**git log --author=cdl --oneline -5(显示作者，最近的五条)**

**git log --oneline --before={3.weeks.ago} --after={2018-1-1} --no-merges(显示三周前，一月一号之后提交的，no merge来隐藏合并)**

**git log --oneline --decorate --graph(decorate可以看到标签)**

**git rm \* -r 批量删除**

**git add -A批量添加（new modified deleted）**

**git add . 批量添加（new modified deleted）**

git add –ignore-removal(new modified)

git add –u(modified deleted)

**删除本地版本库：rm -rf .git(先删除.git) rm -rf learngit(再删除版本库)**

**1 git init 把某个文件夹作为版本库**

**2 git add a.txt 把文件加入到版本库（文件需先放在版本库）提交修改**

**3 git commit –m “commit 3 files”提交**

**4 git log 查看历史提交文档**

**5 git log –pretty=oneline查看一条主线的日志**

**6 git reset --hard HEAD^ hard后面指定版本（HEAD是当前版本 ^是当前版本的上一个版本 ^^是上上个版本）**

**7 git reset --hard a28b6df 后面跟上版本id就能回到那个版本（就是commit后面那个id）回退到上一个版本再想回到当前版本就必须指定commit id**

**8 git reflog 记录每一条git命令（包括commit id）**

 

git回退只是把当前head指针指向前一个版本，顺便更新工作区文件

**git status 比较工作区和版本库的差别**

9 .git称之为版本库，git add是把要提交的修改放在暂存区(stage)，提交的文件修改都放在暂存区，然后一次性提交暂存区的所有修改。







**10 git diff HEAD -- readme.txt比较工作区和版本库里面最新版本的区别（查看最后一次提交之后的所有改变 --指定文件）**

**diff --git a/a.txt b/a.txt**

**index 5575f6f..c1dbc5b 100644**

**--- a/a.txt**

**+++ b/a.txt**

@@ -2,3 +2,4 @@ git init

git is good

is yes ppp

uuu

+aaa//工作区增加的内容

git diff 查看修改还未载入的内容

git diff --staged 载入还未被提交（add not commit）

git diff v1.0 v1.1 两次版本的差异对比

git diff master dev 比较两个分支

git diff master...dev创建dev分支之后在这个分枝上的变换，用dev与master所交的那个分歧点和现在的dev分枝上最后一个快照进行比对

git diff ...dev如果在master分枝上，可以这样运行,查看将要合并的某个分枝会有什么样的变化

**11 git checkout -- a.txt**

**命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：**

**一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；**

**一种是readme.txt已经添加到暂存区后，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，就是让这个文件回到最近一次git commit或git add时的状态。**

**12 git rm test.txt/git commit -m “rm test.txt”删除文件并提交到版本库**

**13 git remote add origin** [**https://github.com/Chiron-C/gitTest.git**](https://github.com/Chiron-C/gitTest.git)**(连接到远程仓库，origin是仓库别名)**

**14 git push –u origin master 提交代码，第一次使用用u，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。**

**以后使用可以简化为：git push origin master**

**push出现问题：failed to push some refs to**

**原因是远程仓库的redme文件未同步到本地，首先git pull --rebase origin master（合并代码），然后再push** 注**pull=fetch+merge**

**15 git clone git@github.com:Chiron-C/gitSkil.git 克隆**

**16 创建与合并分枝：**

**每次提交，master都会向前移动一步，随着不断提交，master分枝线也越来越长**



**新建分枝dev，指针dev会指向当前指针master，同时head指针指向dev**



**这时对工作区修改就是针对dev了，每提交一次，dev指针向前移动一步**



**在dev分枝的工作完成后就可以合并到master**



**然后就可以把dev删除**



**git checkout –b cdl 创建分枝cdl///或者git branch cdl /git checkout cdl(创建分枝再转到分枝)**

**git branch 查看当前分枝**

**那么当前再对工作区的修改就是在分枝cdl上进行的了**

**git push origin cdl 推送到远程仓库**

**git checkout master切换到master分枝**

**git merge cdl 把cdl分枝合并到master**

**git branch –d cdl 删除cdl分枝**

**17 分枝管理策略**

**git merge --no-ff –m “merge with no-ff” dev 禁用Fast forward ，合并并且提交（commit）**

**git log --graph --pretty=oneline --abbrev=commit查看分枝历史**

\* d1be0be2a020c0f5e710fab0336c681297f74eda (**HEAD -> master**) merge with no-ff

|\

| \* abc292a9486cc4d4f9e15dd2826924ccd8941d4a (**dev**) dev commit

|/

\* 3b9c8828dc2cf924cdbb1bb83e44855312ef3062 (**origin/master**, **origin/dev**) dev

\* 83815c59e2fdb39a03f8b10762eecd6548811083 (**origin/cdl**) add 1234 2234

\* 21bb64675ec50d2f32bb0cb7b6b6ad07baa12fa5 commit

\* a28b6dfa2a35868e3c231cbcdc9b5c15d7bc5b5b add something to a.txt

\* 3bc6ec2ef4a82ef4f64cb86a68a8fa416d5592b3 a file commit



**不使用fast forward后，merge变成这样（其中master和dev都做过修改并且提交）**

**18 快照功能：某人在一个分枝的正在工作，需要新建一个分枝去修改bug，这时可以创建快照保存当前现场，然后等bug修复完再恢复现场**

**git stash 创建快照来保存现场，等到bug修复完成，再回到当前分枝，使用git stash pop来恢复现场（或者git stash apply恢复再用git stash drop删除快照）**

**git stash list 可以查看快照列表**

**当列表较多的时候可以用 git stash apply stash@{0}来恢复（指定快照名称）**

**19 future分枝，当需要开发一个功能的时候，创建一个future分枝，当这个功能用不到的时候就可以强制删除这个分枝 git branch –D feature**

**20 git remote查看远程库信息，git remote –v 查看更加详细信息**

并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

**21 多人协作：假设你的小伙伴在另外一台电脑上clone了远程仓库（默认只有master分枝）**

**git branch --track cdl origin/cdl 抓取远程分枝（建立与远程分枝的链接）**

**若是多人协作，先push如果说push出现错误，那就先git pull，先把远程仓库的啦取下来在本地进行合并再push**

**22 打标签 git tag -a v1.0(标签默认打在最新提交的commit上)，如果想要打在以前的commit上那就需要找到commit id，使用git log --pretty=online --abbrev-commit,找到commit id，然后git tag v0.9 commitid, git show v1.0显示标签信息，git tag –a v1.0 –m “hahha” commitid(-a指定标签名，-m指定说明文字) git tag –s v1.0 –m “aaa” commitid(使用私钥签名一个标签，使用PGP签名，先要安装gpg（GnuPG）)，git tag(查看所有的标签) git tag -d v1.0(删除标签) git push --tags(推送所有标签) ，git push origin v1.0(推送某个标签) ,git push origin --delete tag v1.0(删除远程标签git 1.7之后)／git push origin :refs/tags/v1.0(1.7之前)**

**23 使用git参加开源项目的时候，先fork到自己的gitHUb，在修改提交（直接克隆到电脑本地的话，提交会没有权限），修改完成后可以在gitHUb上发起一个pull request**

**24 忽略特殊文件：如果我们不想把一些特殊文件（如配置文件传到版本库），我们可以创建一个.gitignore，例子实例：**[https://github.com/github/gitignore](https://github.com/github/gitignore" \t "_blank)

校验.gitignore生效的标准是git status，是否已经忽略在列表中的文件。

当然也可以强行添加到git：用git add -f App.class

查看提交的文件是否符合gitignore规则：git check-ignore -v App.class

25 配置文件：配置别名：git config --global alias.st status

加上--global是针对当前用户的配置，配置文件放置在当前用户主目录下.gitconfig

如果说不加--global则配置文件放置在.git/config中可以修改文件的对映行来删除别名，如果修改错了，可以删除配置文件，再通过命令重新生成。

别名放置在.git/config文件中

26 搭建git服务器：https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/00137583770360579bc4b458f044ce7afed3df579123eca000

**GIt遇到的错误，Please make sure you have the correct access rights and the repository exists.**

**原因是ssh key有问题。**

**git config --global user.name "yourname"／／首先配置用户名和邮箱**

**git config --global user.email“your@email.com"**

**删除.ssh文件下的known\_hosts**

**ssh-keygen -t rsa -C "your@email.com" 生成秘钥**

**进入.ssh，打开id\_rsa.pub,拷贝全部内容，登陆github，进入ssh设置，将里面内容复制到ssh keys**