本地库初始化

Git init  
¬ 形式  
用户名： tom  
Email 地址： goodMorning@atguigu.com  
¬ 作用： 区分不同开发人员的身份  
¬ 辨析： 这里设置的签名和登录远程库(代码托管中心)的账号、 密码没有任何关系。

命令  
ν 项目级别/仓库级别： 仅在当前本地库范围内有效  
υ git **config** user.name tom\_pro  
υ git **config** user.email goodMorning\_pro@atguigu.com  
υ 信息保存位置： ./.git/config 文件

ν 系统用户级别： 登录当前操作系统的用户范围  
υ git config **--global** user.name tom\_glb  
υ git config **--global** goodMorning\_pro@atguigu.com  
υ 信息保存位置： ~/.gitconfig 文件  
ν 级别优先级  
υ 就近原则： 项目级别优先于系统用户级别， 二者都有时采用项目级别  
的签名  
υ 如果只有系统用户级别的签名， 就以系统用户级别的签名为准  
υ 二者都没有不允许

Master主干 ;unstage

Git status 查看状态

Git add [fileName]添加

Git commit –m “\*\*” fileName

工作区（写代码）-暂存区（临时存储，先添加到暂存区后悔还能）-本地库（历史版本）

查看历史版本：

Git log/git log –pretty=oneline/git log –oneline/git reflog

**4.3.4** 查看历史记录  
git log  
多屏显示控制方式：  
空格向下翻页  
b 向上翻页  
q 退出

git log --pretty=oneline  
git log --oneline  
git reflog  
HEAD@{移动到当前版本需要多少步}

基于索引值操作

Git reset –hard [局部索引值]

Reset命令3个参数

--soft

--mixed

--hard

--hard 参数  
ν 在本地库移动 HEAD 指针  
ν 重置暂存区  
ν 重置工作区

删除文件并找回  
¬ 前提： 删除前， 文件存在时的状态提交到了本地库。  
¬ 操作： git reset --hard [指针位置] 2种情况

ν 删除操作已经提交到本地库： 指针位置指向历史记录  
ν 删除操作尚未提交到本地库： 指针位置使用 HEAD

Git reset –hard HEAD

比较文件差异

比较文件差异  
¬ git diff [文件名]  
 将工作区中的文件和暂存区进行比较  
¬ git diff [本地库中历史版本] [文件名]

将工作区中的文件和本地库历史记录比较

不带文件名比较多个文件

分支：

在版本控制过程中， 使用多条线同时推进多个任务

分支的好处？  
同时并行推进多个功能开发， 提高开发效率  
各个分支在开发过程中， 如果某一个分支开发失败， 不会对其他分支有任  
何影响。 失败的分支删除重新开始即可。

创建分支

Git branch [分支名]

查看分支

Git branch –v

切换分支

Git checkout [分支名]

合并分支  
 第一步： 切换到接受修改的分支（被合并， 增加新内容） 上  
git checkout [被合并分支名]

[Git chechout master]  
  
第二步： 执行 merge 命令  
git merge [有新内容分支名]

[Git merge hot\_fix]

Git在合并分支的时候产生冲突，解决冲突

解决冲突

冲突的解决  
υ 第一步： 编辑文件， 删除特殊符号  
υ 第二步： 把文件修改到满意的程度， 保存退出  
υ 第三步： git add [文件名]  
υ 第四步： git commit -m "日志信息"  
λ 注意： 此时 commit 一定不能带具体文件名

Git 原理

哈希 一种加密算法

明文---（加密算法）---密文

哈希是一个系列的加密算法， 各个不同的哈希算法虽然加密强度不同， 但是有以下  
几个共同点：  
①不管输入数据的数据量有多大，输入同一个哈希算法，得到的加密结果长度固定。  
②哈希算法确定， 输入数据确定， 输出数据能够保证不变  
③哈希算法确定， 输入数据有变化， 输出数据一定有变化， 而且通常变化很大  
④哈希算法不可逆  
Git 底层采用的是 SHA-1 算法。  
哈希算法可以被用来验证文件。 原理如下图所示：

Github代码托管中心

Git remote add origin(别名) <https://github.com/ctgitlgq/testgit.git>

Git remote –v

本地库推送到远程库

Git push origin master(分支)

克隆

Git clone <https://github.com/ctgitlgq/testgit.git>

3个效果：

效果  
完整的把远程库下载到本地  
创建 origin 远程地址别名  
初始化本地库

邀请成为collaborator才能push,才能有写的权限。

Fetch(拉取)merge 合并 collaborator 合作者 decline 谢绝

Pull包括

Fethc：

Git fetch origin master

抓取过后本地文件并没有变,只是把远程的内容下载到本地，并没有改本地工作区的文件。

想看下载下来的文件在哪

Git checkout origin/master

Cat [fileName]

Git checkout master

Git merge origin master 远程的master合并到本地

Merge

冲突：

要点如果不是基于 GitHub 远程库的最新版所做的修改， 不能推送， 必须先拉  
取。  
拉取下来后如果进入冲突状态， 则按照“分支冲突解决” 操作解决即可。

跨团队协作

Fork /clone/push

Pull request

SSH登录

进入当前用户的家目录  
$ cd ~  
删除.ssh 目录  
$ rm -rvf .ssh  
运行命令生成.ssh 密钥目录  
$ ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com  
[注意： 这里**-C** 这个参数是大写的 **C**]  
进入.ssh 目录查看文件列表  
$ cd .ssh  
$ ls -lF  
¬ 查看 id\_rsa.pub 文件内容  
$ cat id\_rsa.pub  
¬ 复制 id\_rsa.pub 文件内容， 登录 GitHub，点击用户头像→Settings→SSH and GPG  
keys  
¬ New SSH Key  
输入复制的密钥信息  
回到 Git bash 创建远程地址别名  
git remote add origin\_ssh git@github.com:atguigu2018ybuq/huashan.git  
推送文件进行测试