git学习笔记

Git

版本控制

1.版本迭代

2.记录所有版本，版本控制的工具，方便查看更改历史记录。

3.多人开发使用的版本控制工具。

4.Git可以直接看到更新了哪些代码和文件.

5.Git 分类

Git Bash:Unix与Linux风格的命令行,使用最多,推荐最多

Git CMD :Windows风格命令行

Git GUI:图形界面的git

6.基本的Linux命令

Cd 切换目录

Pwd :显示当前所在的目录路径

clear:清屏

ls:列出当前文件夹所有文件

rm -r 名称:删除一个文件

touch:新建一个文件

mkdir:创建一个文件夹

Mv 名字 文件夹:移动文件到文件夹

reset:重新初始化终端/清屏\

history:历史使用过的命令

exit:退出

#:注释

Git的相关配置文件

Git config --global user.name 姓名 配置git用户名

Git config --global user.email 邮箱 配置邮箱

Git config --global credential.helper store 保存用户名和密码

Git config -l:查看所有git配置

Git config --system --list:查看系统的config

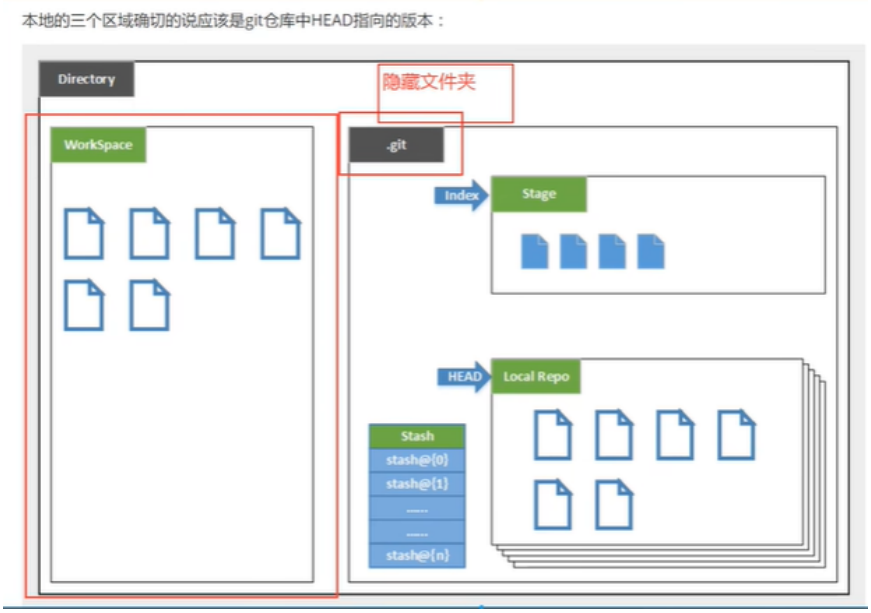
Git config --global --list:查看当前用户的配置

相关说明:

省略(local):本地配置,只对本地仓库有效

--global:全局配置,所有仓库生效

--system:系统配置,对所有用户生效



Git的基本理论:

工作流程:

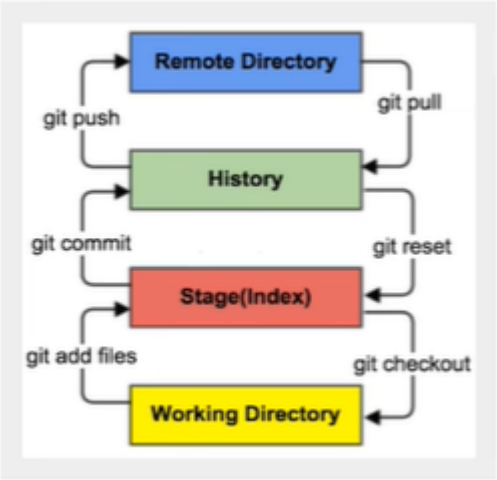
1.在工作目录中添加,修改文件;

2,将需要进行的版本管理的文件放入暂存区域. Git add .

3.将暂存区域的文件提交到git仓库。 git commit

4提交到云端 git push

因此git管理的文件有三种状态：已修改(modified), 已暂存(staged)，已提交(committed)



git项目的搭建：

1.git init 在当前目录新建一个Git代码库。

2.克隆远程目录，是将远程服务器上的仓库完全镜像一份至本地。

Git clone [url]

git相关区域划分:

Working Directory(工作区)

.git所在的目录

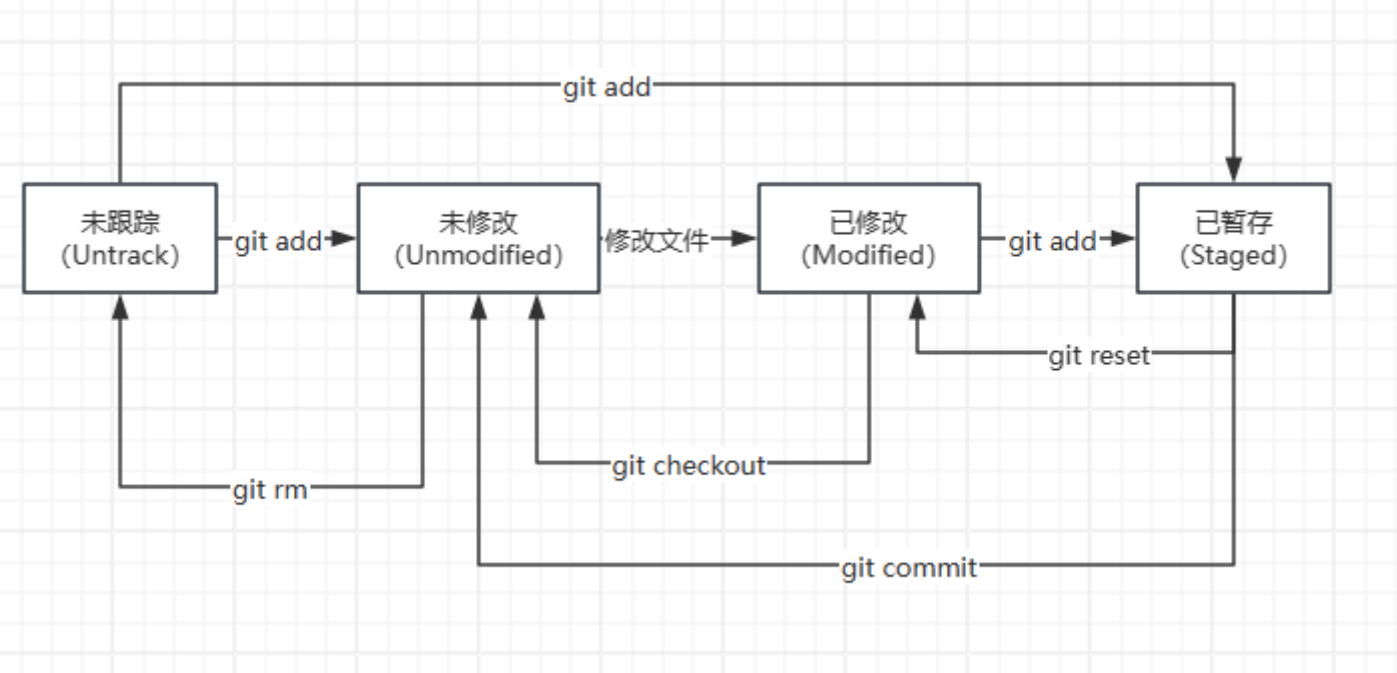
Staging Area/Index(暂存区)

.git/index

Local Repository(本地仓库)

.git/objects

git文件状态描述:



git文件操作

Git status [filename] 查看指定文件状态

Git ls-files 查看暂存区中的内容

git status 查看所有文件的状态

Git add 将文件添加到暂存区

Git rm --cached<file> 将存到暂存区中的内容取消

Git commit -m “提交的信息(相当于命名)” 提交暂存区中的内容到本地仓库

Git log 查看提交记录

Git log --online 查看简洁的提交记录

Git reflog 查看操作的历史记录

Git pull 将远程仓库内容拉取到本地仓库

Git push 将本地仓库传输到远程仓库

Git commit -a -m "信息" 可以完成暂存和提交两个功能

Git reset三种模式

Git reset --soft

回退到某一个版本,并保留工作区和暂存区的所有修改内容

Git reset --hard

回退到某一个版本,并丢弃工作区和暂存区的所有修改内容

Git reset --mixed (reset命令的默认参数)

回退到某一个版本,只保留工作区的内容,丢弃暂存区的修改内容

git diff查看差异

查看工作区,暂存区,本地仓库之间的差异

Git diff 默认比较工作区和暂存区之间的差异内容

Git diff HEAD 比较工作区和版本库之间的差异

Git diff --cached/git diff --staged 比较暂存区和版本库之间的差异

查看不同版本之间的差异

Git diff +所比较的两个版本的版本号id

Git diff HEAD 版本号id 比较与当前最新版本的差异

HEAD:分支的最新提交节点

HEAD~或者HEAD^:上一个版本

HEAD~+数字:表示HEAD之前的几个版本

比较后面加文件名表示只比较当前的文件差异

查看不同分支之间的差异

Git diff <brabch_name><branch_name>

git中删除文件

使用linux命令 rm 文件名称

删除时工作区内容删除掉后要使用git add通知暂存区也删除掉该内容

Git rm 文件名

Git rm --cached 文件名 只删除版本库中的文件

在工作区与暂存区中都删除该文件

但要记得git commit 提交,否则版本库中仍然存在该文件

忽略文件

应该忽略哪些文件?

1. 系统或软件自动生成的文件
2. 编译产生的中间文件和结果文件
3. 运行时生成的日志文件,缓存文件,临时文件
4. 涉及身份,密码,口令.密钥等敏感信息文件

设置忽略文件配置 .gitignore

忽略的文件不包括已经被添加到仓库中的

*.log 忽略所有以log结尾的文件

文件夹/ 忽略该文件夹下的内容

详情见下图

.gitignore文件的匹配规则

1. 从上到下逐行匹配,每一行表示一个忽略模式
2. 空行或者以#开头的行会被git忽略,一般空行用于可读性的分隔,#一般用作注释
3. 使用标准的Blob模式匹配

   星号*通配任意个字符

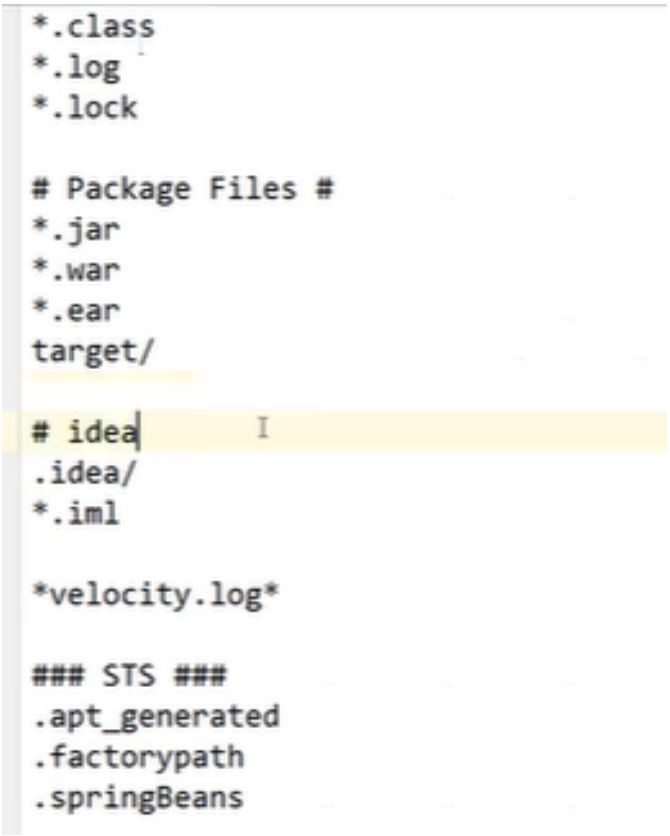
   问号?匹配单个字符

   中括号[]表示匹配列表中的单个字符,比如:[abc]表示a/b/c
4. 两个星号**表示匹配任意的中间目录
5. 中括号可以使用短中线连接 [0-9]表示任意一位数字,[a-z]表示任意一位小写字母
6. 感叹号! 表示取反

使用idea时

在主目录下建立“.gitignore”文件，此文件有如下规则

1. 忽略文件中的空行或以井号（#）开始的行将会被忽略。
2. 可以使用Linux通配符。例如：星号（*）代表任意多个字符，问号（？）代表一个字符，方括号（[abc]）代表可选字符范围，大括号（{string1,string2,...}）代表可选的字符串等。
3. 如果名称的最前面有一个感叹号（!），表示例外规则，将不被忽略。
4. 如果名称的最前面是一个路径分隔符（/），表示要忽略的文件在此目录下，而子目录中的文件不忽略。
5. 如果名称的最后面是一个路径分隔符（/），表示要忽略的是此目录下该名称的子目录，而非文件（默认文件或目录都忽略）。



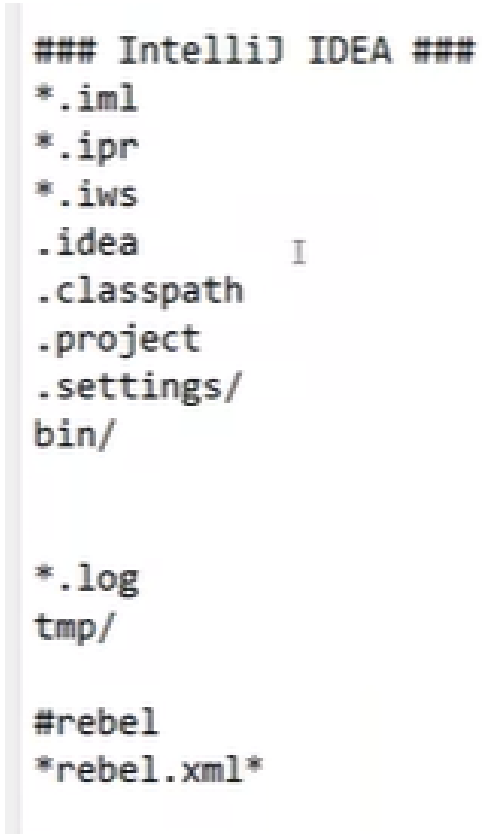
```
*.class
*.log
*.lock

# Package Files #
*.jar
*.war
*.ear
target/

# idea
.idea/
*.iml

*velocity.log*

### STS ###
.apt_generated
.factorypath
.springBeans
```



```
### IntelliJ IDEA ###
*.iml
*.ipr
*.iws
.idea
.classpath
.project
.settings/
bin/


*.log
tmp/

#rebel
*rebel.xml*
```

设置本机绑定SSH公钥，实现免密码登录！

进入C：\Users\Administrator\.ssh 目录

cd

Cd .ssh

ssh-keygen -t rsa -b 4096 生成密钥信息 第一个输入生成密钥的文件名称

上传.pub的文件内容

在.ssh目录下创建config写入

#github

Host github.com

HostName github.com

PreferredAuthentications publickey

IdentityFile ~/.ssh/密钥文件

指定使用什么密钥

本地仓库连接远程仓库

1. 创建仓库
2. 复制url的地址
3. Git remote add<shortname><url>

4. Git remote -v 查看当前仓库所对应的远程仓库的别名和地址
5. Git push -u origin master将两个仓库的master关联起来 Git push -u<远程仓库名><分支名>
6. Git pull <远程仓库名><远程分支名>:<本地分支名>
7. Git fetch 获取远程仓库的修改并不会自动合并到本地仓库

查看分支

Git branch

远程的分支

Git branch -r

创建分支

Git branch 分支名字

新建一个分支，并切换到该分支

Git checkout -b 分支名字

Git switch 分支名 2.23更新,语义更加明确

合并指定分支到当前分支

Git merge 分支名字

git log --graph --oneline --decorate -all 查看分支图

删除分支

Git branch -d 分支名字 只能用于在分支合并之后

Git branch -D branch-name 强制删除分支

删除远程分支

Git push origin --delete 分支名字

Git branch -dr [remote/branch]

文件合并时的情况

同一个文件合并分支时都被修改会引起冲突,解决方法就是可以修改冲突文件后重新提交.是否替换掉.

Git merge --abort 终止合并

主分支

master分支应该十分稳定,用来发布新版本,一般情况下不允许在上面工作,工作一般在新建的分支上工作,工作完成后可以合并到主分支上.

rebase

在dev上执行就是将dev合并到main的分支中,

在main上执行就是将main合并到dev的分支中,

rebase的机制

分支都会有一个指针,而rebase会先找到当前分支与目标分支的共同祖先,再把当前分支上从共同祖先到最新提交记录的所有提交都移动到目标分支的最新提交后面.

alias 别名="命令" 将命令改为一个别名可以直接使用

Rebase和Merge有什么区别该如何区分使用

Merge

优点:不会破坏原分支的提交记录,方便回溯和查看

缺点:会产生额外的提交节点,分支图比较复杂.

Rebase

优点:不会新增额外的提交记录,形成线性历史,比较直观干净;

缺点:会改变提交历史,改变了当前分支branch out的节点.避免在共享分支使用.