1、Git是什么：Git是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。

2、Git特征：

- git每台电脑都有一个版本库，可以在本地做版本管理。
- 速度快，git的速度远超大部分版本管理系统，包括svn。
- 强大的分支管理功能。
- Git不需要联网在本地就可以进行版本提交。
- 活跃的开源社区，如最著名的GitHub。

3、Git的去中心化和分布式体现在每台电脑都有一个版本库，可以在本地做版本管理。不像svn不能本地提交，每次提交都是直接提交到SVN服务器，即它只有一个中心为SVN 服务器。

4、【git init】 初始化一个本地仓库，让Git开始管理这个文件夹，在同级目录下会出现一个隐藏的.git文件。

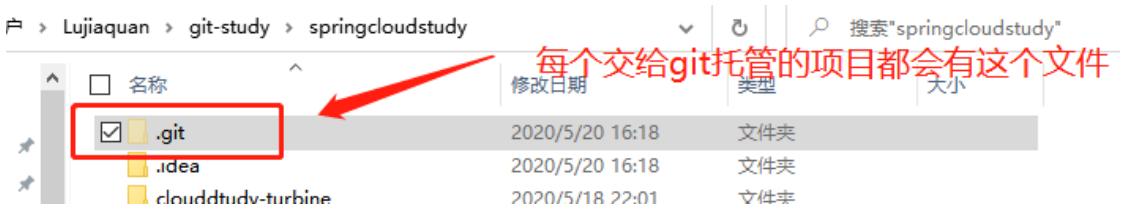
5、【git config -l 】查看所有配置信息，【git config xxx】查看具体某项配置。

6、【git config --global user.name 'name' ； git config --global user.email 'email'】命令只能用于初次配置user.name/email，如果不小心配置错误，或者重复配置，不可以通过重复执行以上命令来修改user.name/email。修改user.name/user.email有以下两种方式：

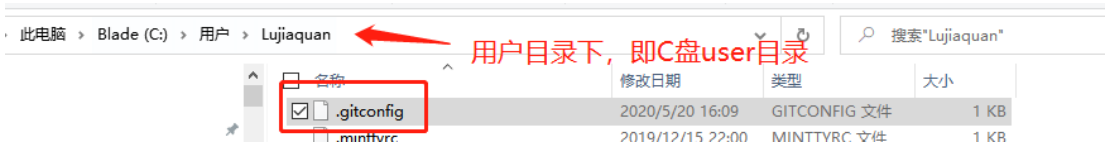
- git bash用命令修改：
  - 修改user.name：git config --global --replace-all user.name "your user name"
  - 修改user.email：git config --global --replace-all user.email "your user email"
- 修改.gitconfig文件：该文件是隐藏文件，位于C:\Users\{user}\.gitconfig，直接修改里边的name或者email，如果有重复的name或email，可以将其删掉，只剩下一个就好。

## 7、git的三级文件配置（当三个配置文件中有相同配置的时候生效优先级为 1>2>3）：

### 1. .git/config：次配置文件在每个项目的.git文件夹下：




### 2. 当前用户目录C:\Users\Lujiaquan\.gitconfig：



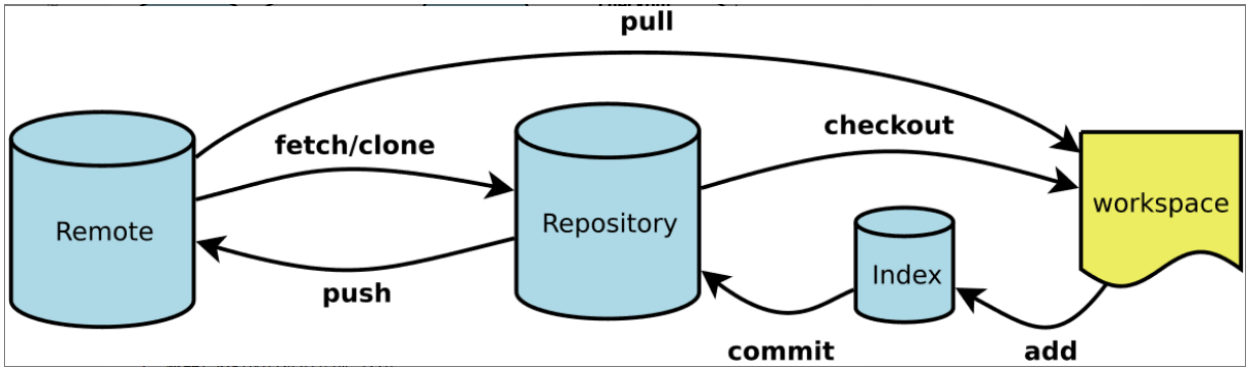

### 3. git安装目录/etc/gitconfig：我本机git安装到了D盘：

## 8、Git工作区、暂存区、版本库、远程仓库：

- 工作区(Working Directory)：在git管理下的正常目录都算是工作区。就是你平时存放项目代码的地方。我们通过idea等进行操作的java文件等都属于工作区。
- 暂存区(Stage/Index)：用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息，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(Repository或Git Directory)：就是安全存放数据的位置，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，这里面有你提交到所有版本的数据。其中HEAD指向最新放入仓库的版本。

- git仓库(Remote Directory)：远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换。

## 9、文件在四个区域之间转换关系如下：




**10、提交本地文件根据文件是否被追踪可以分两种情况（https://www.cnblogs.com/smile-fanyin/p/10827438.html）：**

- **文件已被追踪：加入缓存区并提交，即执行【git commit -am "版本描述"】命令即可。**
- **文件未被追踪：先加入到缓存区，然后提交版本。即先执行【git add -A】命令，在执行【git commit -m "版本描述"】；**

**11、git add . 和git add -A都是将文件加入缓存区（.或者-A表示全部，也可以是单独文件，单独文件的话把.或者-A替换成文件名），它俩的区别如下：**

- **git add . ： .代表将所有新增、修改的文件加入缓冲区。**
- **git add -A：-A代表将新增、修改、删除的文件加入缓存区。**

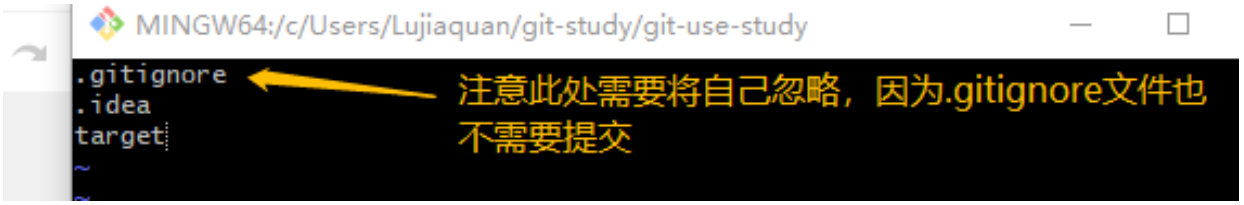
**12、git中的三类文件（https://blog.csdn.net/ShunXiangL/article/details/51465389）：**

正如上文所说，Git在未进行commit操作之前，存在三种状态：Untracked files，Changes not staged for commit及Changes to be committed，每种状态之间可以随意进行互相转换。了解这三种状态各自所对应的不同情况，能够帮助你方便有效的使用Git来管理项目。

- **被追踪的文件(tracked)：已经纳入暂存区或者版本库，即已经执行过【git add】命令的文件也就是已经被加入缓存区的文件。已被追踪的文件修改之后状态还是被追踪，只需要进行commit就ok。**
- **不被追踪的文件(untracked)：没有被纳入暂存区或者版本库的文件。还没执行过【git add】命令的文件。新加的文件状态是untracked，所以需要先git add，在执行git commit。**

- 忽略的文件(ignored)：忽略那些不需要管理的文件夹或者文件，例如.idea文件夹、target文件夹等。需要注意的是gitignore还可以指定要将哪些文件需要添加到版本管理中即不能被忽略，为什么要有两种规则呢？想象一个场景：我们只需要管理/mtk/目录中的one.txt文件，这个目录中的其他文件都不需要管理。那么我们就需要使用图片中的配置方式，假设我们只有过滤规则没有添加规则，那么我们就需要把/mtk/目录下除了one.txt以外的所有文件都写出来：

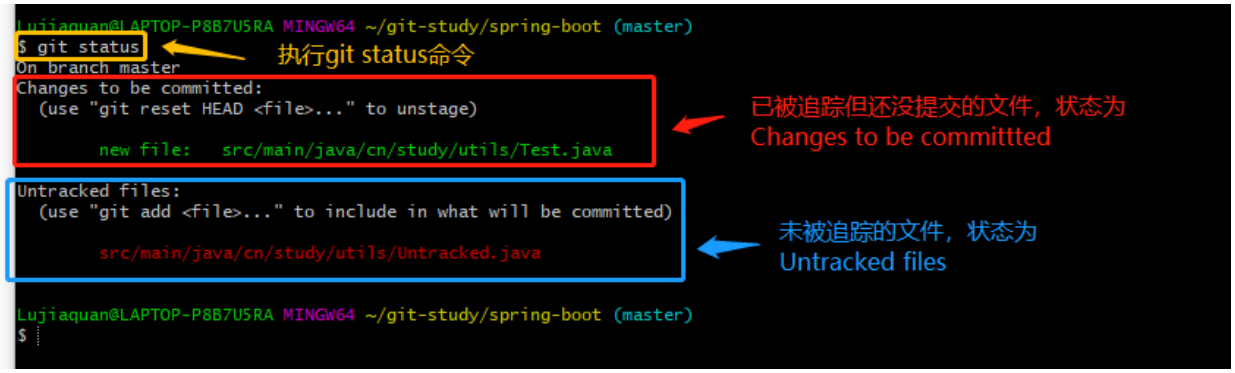

13、如何忽略和排除文件：在目录下使用【touch .gitignore】新建一个.gitignore文件，此文件的影响范围为当前文件夹以及子文件夹（即加入我们在hivegl目录下新建了一个.gitignore文件，并且文件里边排除target文件夹，则hivegl-facade目录下的target文件夹也会被排除，所以对于多模块的maven工程，我们只需要在父工程目录下操作即可，子模块都会生效）：



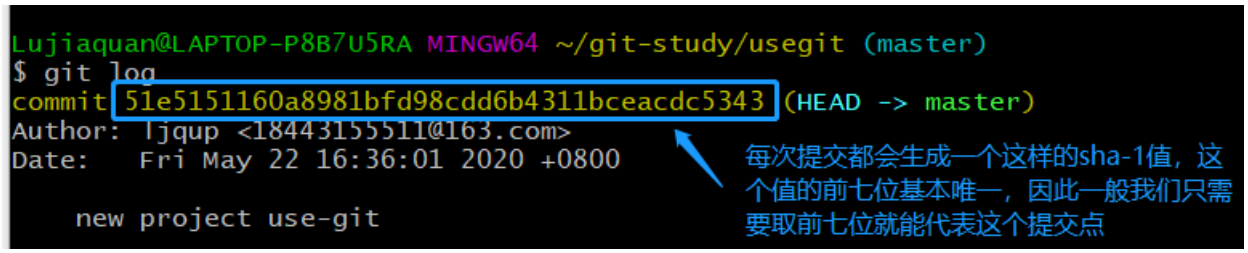

14、忽略和排除文件的两种情况：

- 当文件未被提交且未被追踪，此时只需要在目下新建.gitignore文件来配置那些需要忽略，那些不被忽略。
- 当文件已被追踪或者是已被提交时，此时需要先把本地缓存删除（将文件状态改为未被追踪），命令【git rm -r --cached .】（.代表删除所有文件缓存，此处也可以确切指定某个文件，），然后重新提交。【git rm --f readme1.txt】删除readme1.txt的跟踪，并且删除本地文件。

15、git status命令用来检查版本状态，要养成一个好习惯，每次提交代码前执行此命令查看一下状态，避免出错：




**16、git提交点：每一次commit都是一个提交点，git会通过对文件的内容或目录的结构计算出一个唯一的 SHA-1 哈希值，ID可使用前4至7个数字表示，基本前七个数字就是唯一了：**




**17、查看提交内容：**

- **查看某次commit的内容：git show commit_id，如果提交点有打标签则也可以用git show tagname命令查看提交内容。**
- **查看近n次提交的修改内容：git log -p -n，指定n为1则可以查看最近一次修改的内容；**
- **查看某次commit中具体某个文件的修改：git show commit_id fileName；**

**18、Git Tag（标签）作用：标签其实跟commit 生成的sha1值作用相似，就是给当前的版本做个标记，以便回退到此版本。如果大家都记不住那条冗长的sha1码，所以用tag标签来做记录。一般开发分支不会打标签，因为提交次数比较多，打标签反而会很混乱，主要是在master分支对每次发布投产的版本打标签进行记录，以便于版本回退：**

- **给当前commit打标签（即head所指提交点）：git tag <tagname>；**
- **查看所有标签：git tag；**
- **在历史提交上边打tag：git tag <tagname> commit_id；**
- **带说明的标签，用-a指定标签名，-m指定说明文字：git tag -a <tagname> -m "xxx"；**

- 删除标签：git tag -d <tagname>；
- 推送某个标签到远程：git push origin <tagname>；
- 推送所有标签到远程：git push origin --tags；（此处的origin是远程仓库地址）。

19、git日志：

- git log、git log --oneline、git log --oneline --graph;
- git reflog：git reflog可以查看所有的git操作日志，比如我们提交了1、2、3三个版本，并且当前head指向3版本，当reset到2版本后，log只显示1、2两个版本的提交点。此时如果在想回退到3版本的话就需要通过reflog查看3的提交点进行reset操作。rebase之后我们也可通过reflog查看之前分支的提交日志。

20、HEAD：HEAD节点代表最新的commit， 显示信息【git show HEAD】。

21、git reset命令可以将当前的HEAD重置到特定的状态，即将版本回退到某个提交点，有三种常用模式（--soft、--mixed、--hard）
https://www.jianshu.com/p/c6927e80a01d:

- 【git reset --soft commit_id】：使用--soft参数将会仅仅重置HEAD到指定的版本，不会修改index（暂存区）和working tree（工作区），例如当前分支现在有A、B、C三个提交点，当从C提交点回退到B提交点时C提交点的代码还在暂存区和工作区。即此时C提交点的代码状态从commited状态变成了staged。如果想把暂存区的代码删除使用【git rm <文件名> --cached】，如果想重新提交这些代码无需再一次执行git add 。
- 【git reset --mixed commit_id】：使用--mixed参数与--soft的不同之处在于，--mixed修改了index（暂存区），即相当于比--soft多做了一个删除缓存的操作，例如当前分支现在有A、B、C三个提交点，当从C提交点回退到B提交点时C提交点的代码现在只存在于工作区。即此时C提交点的代码状态从commited状态变成了untracked。如果现在执行git commit 将不会发生任何事，因为暂存区中没有修改，在提交之前需要再次执行git add。
- 【git reset --hard commit_id】：使用--hard不但会会修改index（暂存区），同时也会修改working tree（工作区），即本地文件的修改都会被清除，彻底还原到上一次提交的状态。所以在执行reset --hard之前一定要小心。例如当前

分支有A、B、C三个提交点，当从C提交点回退到B提交点时C提交点的代码会从缓存区、工作区清除（即从idea中也看不到了）。不过也不用慌，如果回退完之后又发现C提交点的代码有用，此时可以通过git reflog找到C提交点的sha-1值，然后在reset到C提交点就ok。

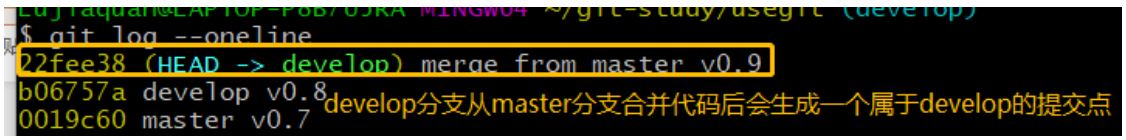
22、【git revert <commit_id>】是撤销某一个已提交的版本，例如我们提交了 6 个版本，其中 3、4 包含了错误的代码需要被回滚掉。 同时希望不影响到后续的 5-6。此时可以使用revert将3、4两个版本的代码给剔除。要撤销一串提交可以用 <commit1>..<commit2> 语法。revert会新建一个提交点，例如A、B、C三个提交点，通过revert将C提交点撤销以后会生成一个新的提交点D，D提交点内容和B提交点完全一样。

22、git分支——分支意味着你可以从开发主线(master)上分离开，在不影响主线的同时继续工作

- 【git branch】：列出当前开发所有分支，并标示当前所在分支；
- 【git branch develop】：创建一个名为develop的分支（从最新提交点开始）；
- 【git branch develop commit_id】：从指定提交点创建一个新分支；
- 【git branch -d develop】：删除develop分支；
- 【git checkout develop】：切换到develop分支；

22、合并分支（http://pinkyjie.com/2014/08/10/git-notes-part-3/）：

- 【git cherry-pick <commit_id1> <commit_id2>】：git cherry-pick后边需要跟提交点，即将另一个分支上某一次提交或者N次提交的代码合并到当前分支。
- 【git merge <分支名>】：git merge是将另一个分支上的所有不同代码都合并到当前分支，合并后会生成一个新的提交点，这个提交点属于当前分支：




- 【git rebase <分支名>】：https://www.jianshu.com/p/6960811ac89c。如果你想要你的分支树呈现简洁，不罗嗦，线性的commit记录，那就采用rebase。

**23、代码合并时冲突以及解决方案：如果在不同分支修改了同一块代码，则在合并时会出现冲突，出现冲突以后我们通过git status可以查看哪些文件有冲突，解决冲突（手工修改内容）以后再执行一次commit命令就ok了。注意如果出现了冲突，就必须先解决冲突才能切换分支。避免冲突需要开发规范来限制，开发过程中最好没有冲突。**

**24、【gitk】命令会显示简单的图形化界面。**

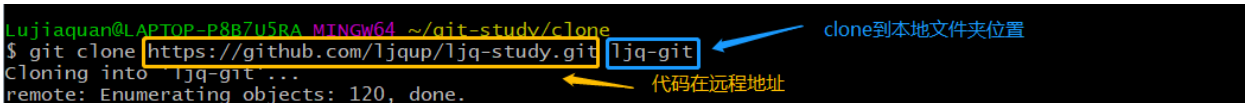
**25、git diff 命令显示区别**

**26、gitHub是一个面向开源及私有软件项目的托管平台，因为只支持git 作为唯一的版本库格式进行托管，故名gitHub。与github连接需要通过加密通道，有两种方式， 一种ssh，一种https。ssh方式需要配置秘钥，生成秘钥的命令【ssh-keygen -t rsa -C <注册邮箱>】，详情看word。**

**27、【git remote add ljq-remote <远程地址>】：本地新建一个远程连接ljq-remote，ljq-remote相当于远程地址的一个别名，建立以后进行clone、pull、push操作的时候可以使用这个别名来指定地址：**


**27、【git remote】查看远程连接；【git remote -v】查看远程连接详情。**

**28、【git clone <远程地址> <本地文件夹>】：clone一个远程项目到本地：**




**29、上传本地项目至gitHub：使用【git push --set-upstream ljq-remote master】，此命令可简化成【git push -u ljq-remote master】，命令后需要输入github用户名和密码，且本地目前在那个分支就会推送到那个分支，假如本地有A、B两个分支，现在checkout到B分支，此时你只能push到B分支，不能push到A分支。**

**30、往远程仓库push代码前一定要先pull将仓库中最新的代码拉到本地，有冲突的话将冲突解决，如果中间有其他人push过代码，但你没拉取，此时不先pull直接push的话会push失败。往远程仓库push代码忽略不需要的文件是必要的，例如target下边打的jar包等等，每次jar包变了但是代码没有一点更改，这种push是没有意义的。**

**31、当远程仓库代码有修改以后不会自动同步到本地，需要主动获取，有两种方式：**

- **【git fetch ljq-remote master】：fetch是将远程主机的最新内容拉到本地，不进行合并，需要我们在手动执行一次【git merge】命令。**
- **【git pull ljq-remote master】：pull 则是将远程主机的最新内容拉下来后直接合并 fetch+merge。一般使用这个命令就ok。**

**32、规范流程是先提交本地代码，然后再pull，最后在push。当使用命令行pull代码时如果本地有未提交的代码则git会提示你先提交。而使用idea直接pull的时候idea会显示冲突并让你解决。**

**33、master分支为主分支，保持稳定性，不允许直接往这个分支提交代码，这个分支只能从其他分支合并代码，而且所有在master分支上的commit应该打Tag。**

**34、git管理规范参考word和ppt。**