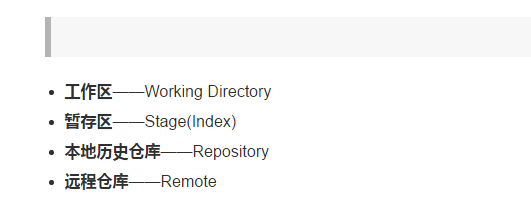
# git 的四个组成部分



工作区

暂存区

写完代码，提交，修改等

Git checkout 版本号（还原

之前的版本）

提交，仓库生成快照，暂存区清空

本地仓库（快照）

暂存区

Reset xxx ,把扣个快照覆盖到暂存区

工作区

本地仓库（快照）

Checkout xxx,直接把工作区还原

成某个快照

Push快照（可能冲突）

远程仓库

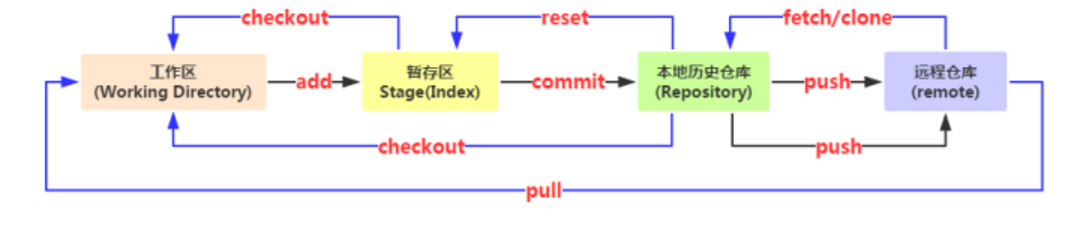
本地仓库

拉取资源

远程仓库

工作区

pull同步代码



# **Git中文件的几个状态**



# 3.Git的下载安装

好了，关键的概念也就是上面这两点，接下来就是Git的一个下载安装了！

**Windows系统：**  
到[Git For Windows](https://link.jianshu.com?t=https://git-scm.com/download/win" \t "_blank)或者[git-for-windows.github.io](https://link.jianshu.com?t=https://git-for-windows.github.io/" \t "_blank)上下载，然后是傻瓜式的下一步

**Linux系统**：  
到[Download for Linux and Unix](https://link.jianshu.com?t=http://git-scm.com/download/linux" \t "_blank)下载，假如你是和我一样的Ubuntu的话直接在Terminal键入：  
**sudo apt-get install git**

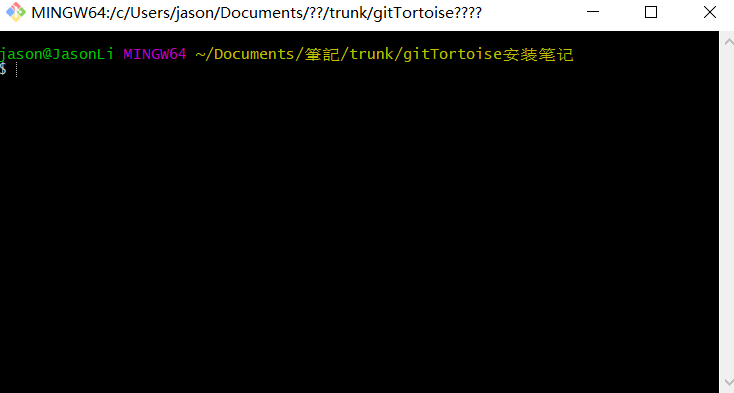
**Mac系统**：  
到[Installing on Mac](https://link.jianshu.com?t=http://git-scm.com/download/mac" \t "_blank)下载

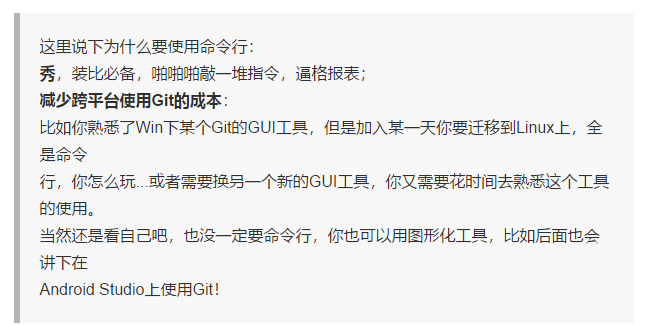
安装完Git后，我们就可以打开Git命令行：

**Windows**在任意位置右键，点击Git Bash打开Git命令行  
**Ubuntu**下直接打开Terminal就可以，Terminal的快捷键是:**ctrl + alt + t**

# ****4.打开git命令行****

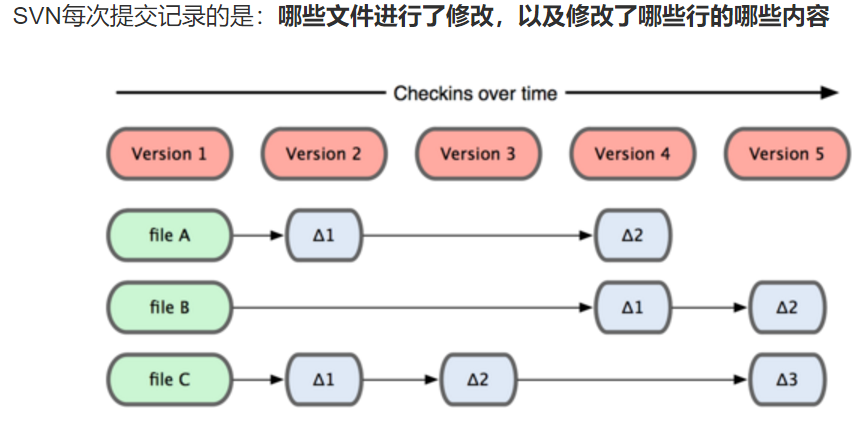






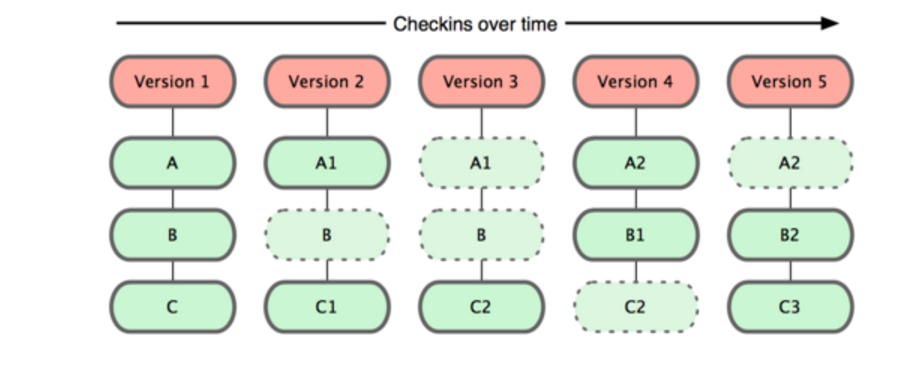
# Git 和 SVN的差异：

SVN:版本保存的是，发生变化的文件信息

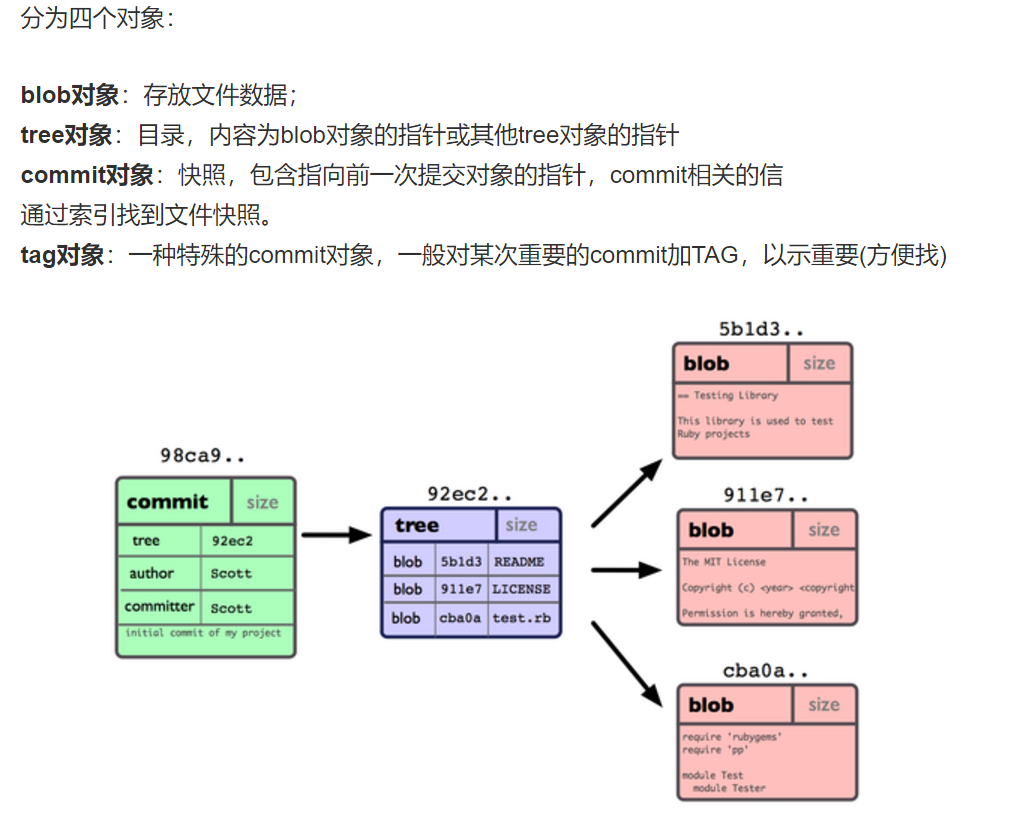
‘

如图：版本2中记录的是文件A和C的变化，而版本3中记录文件C的变化

GIT：版本保存的是 缓存区的所有文件--快照，以及上一次快照的连接



# 6.每次commit时仓库中的数据结构

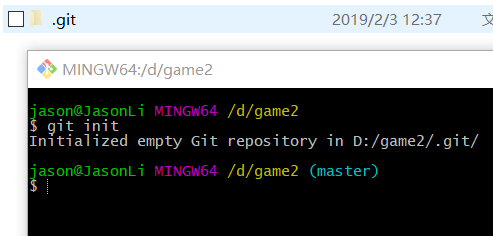


# 7.git获取帮助



# 8.git 创建仓库

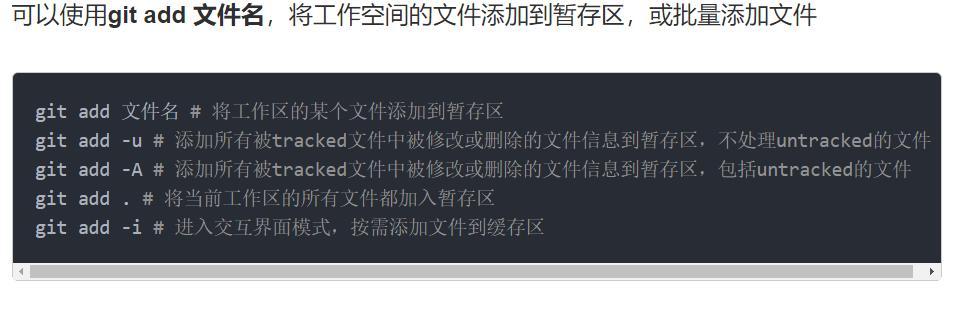
Git init （在所在的文件夹下创建git项目）或者 git init 文件夹名（在指定的文件夹名下创建git项目）



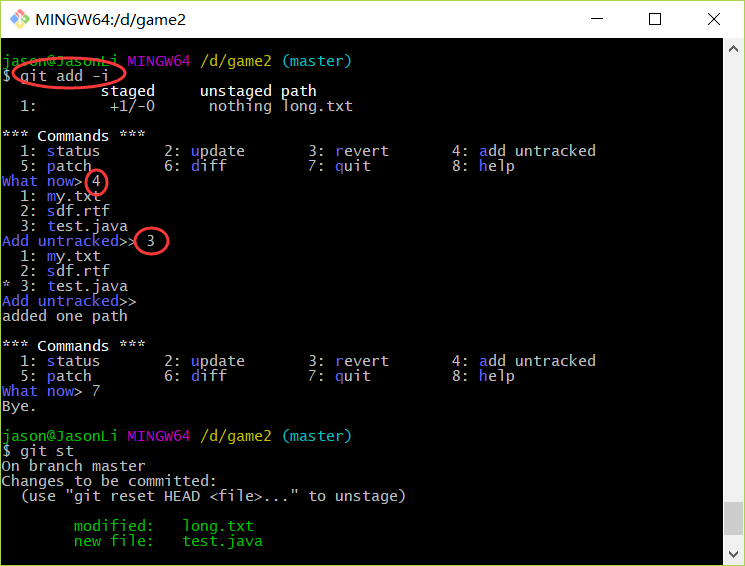
# git 的配置信息

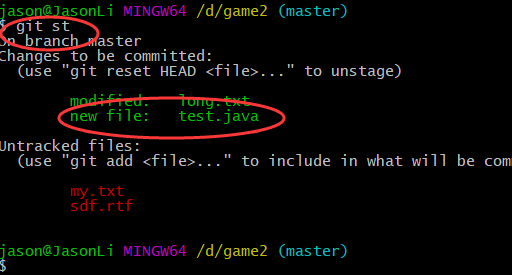


# **git add添加文件到暂存区/文件跟踪标记**

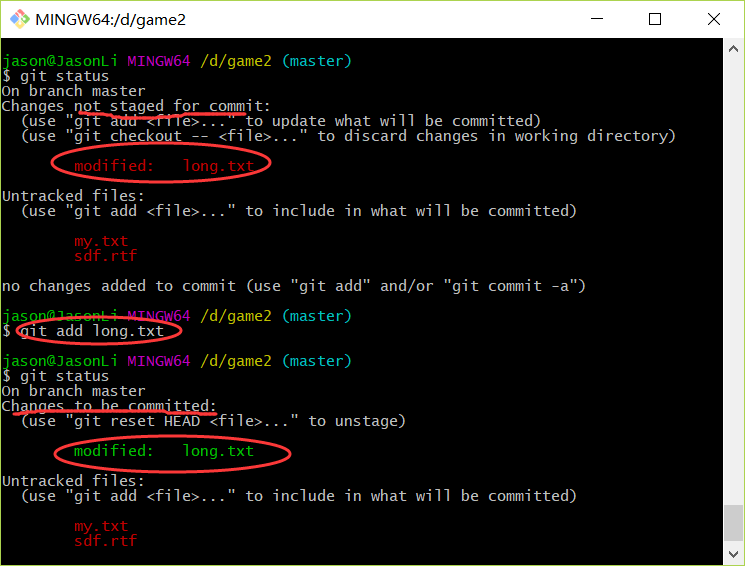


添加文件进入跟踪：

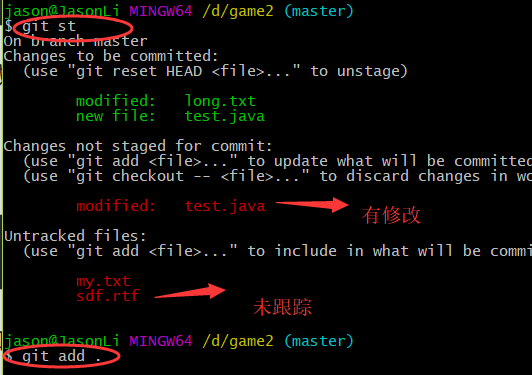


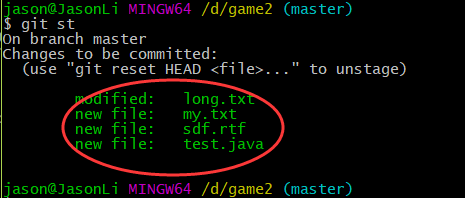


## Git add 已经加入跟踪的，修改过的文件

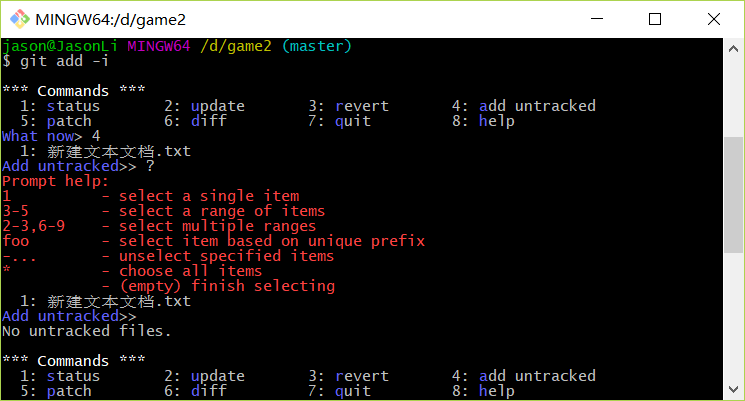


## 已跟踪，未commit的文件修改后，再放入暂存区：





## Git add -i进入交互界面演示：



输入 git add -i:

显示了几个选项：1.status 2.update 3.revert 4.add untracked 等等

此时，我们选择4，what now>4, 即，添加追踪

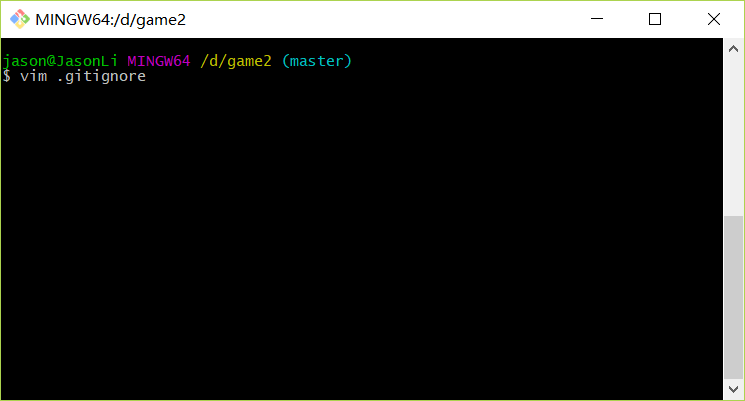
输入：Add untracked>> ? 将会列出帮助信息 ，根据信息选择指定的文件上传

# 忽略某些文件 -- .gitignore文件的配置

**注意: .gitignore与.git文件夹同目录**

## 1.用 GUI BASE创建.gitignore

用 Linux命令行进入.gitignore文件（如果没有则直接创建）



按Insert进入编辑模式，按Esc，:q :wq 等保存离开

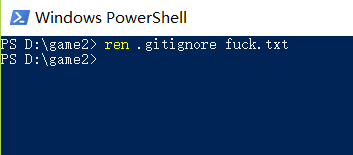
根据编写规则填写



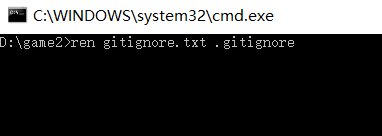
## 2.直接创建 gitignore.txt文件

用命令行修改文件扩展名

长按shift+右键：



或者用CMD命令：



## 3.gitignore的配置规则

# 忽略所有以 .c结尾的文件

\*.c

# 但是 stream.c 会被git追踪

!stream.c

# 只忽略当前文件夹下的TODO文件, 不包括其他文件夹下的TODO例如: subdir/TODO

/TODO

# 忽略所有在build文件夹下的文件

build/

# 忽略 doc/notes.txt, 但不包括多层下.txt例如: doc/server/arch.txt

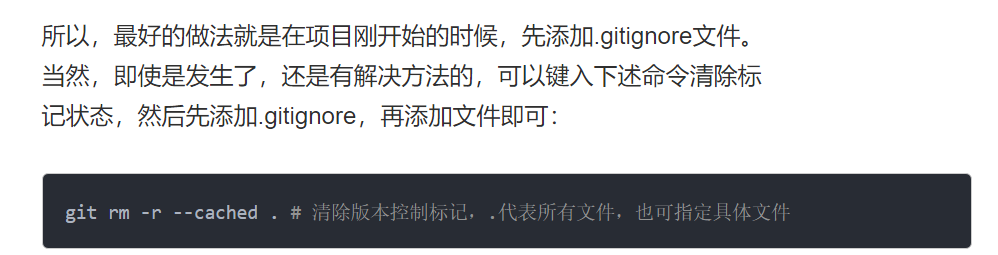
doc/\*.txt

# 忽略所有在doc目录下的.pdf文件

doc/\*\*/\*.pdf

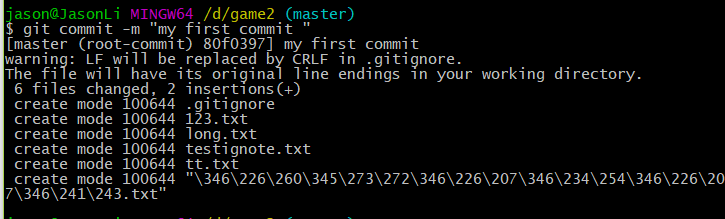
## 4.注意

配置.gitignore只对那些****没有添加到版本控制系统****的文件生效(****未Tracked的文件****)！



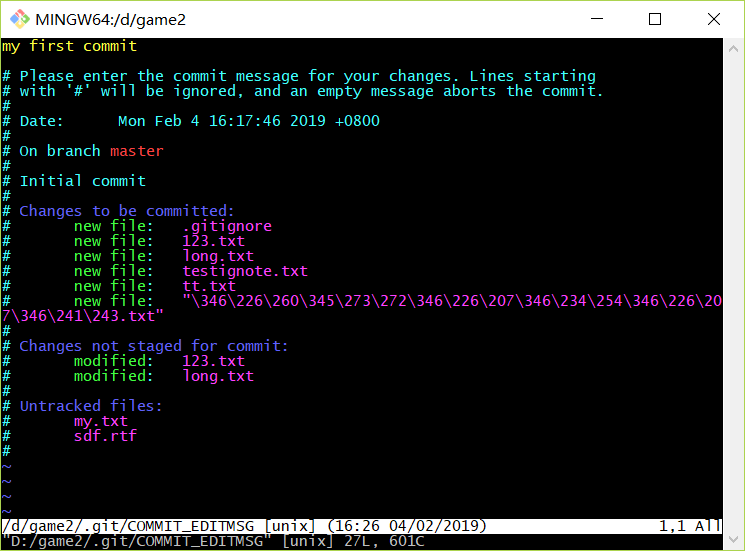
# 12.**暂存区内容提交到本地仓库【git commit】**



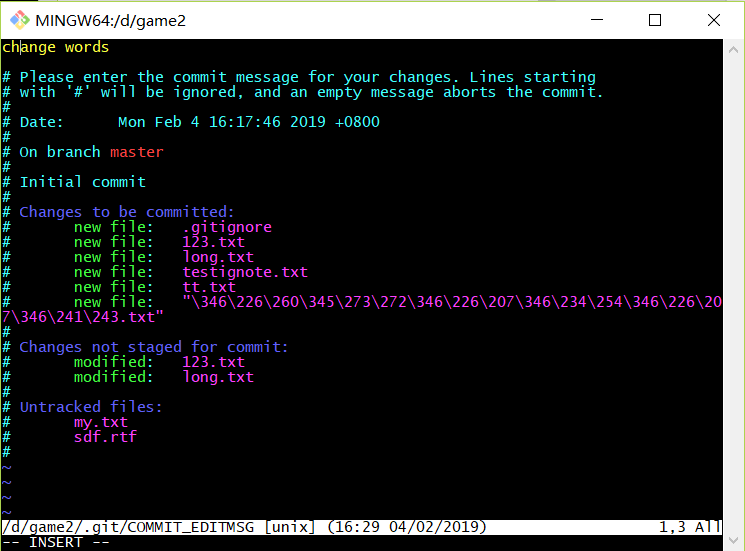


## 修改上次commit的备注：

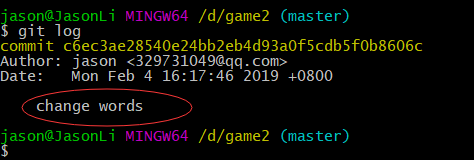




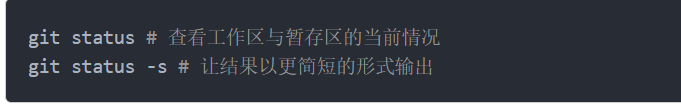
**修改：**

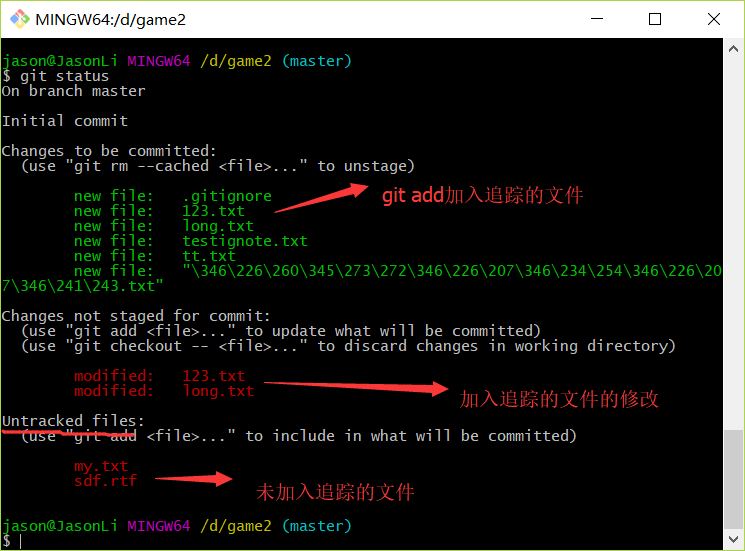




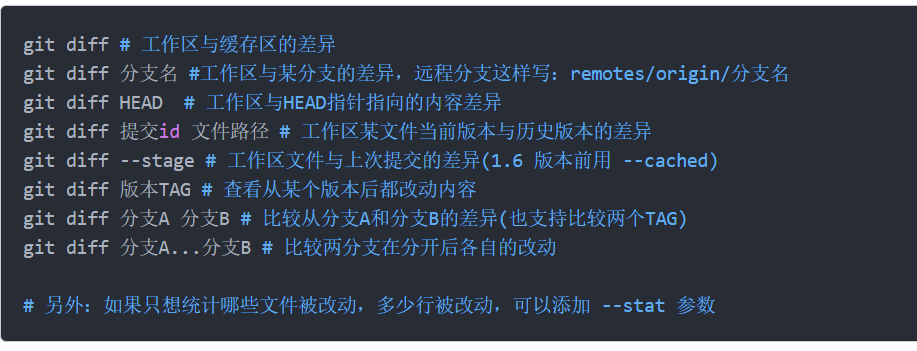


# 13.**查看工作区与缓存区的状态【git status】**

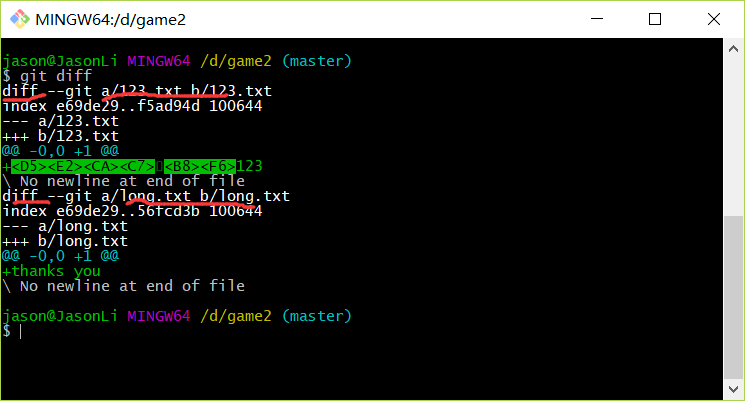




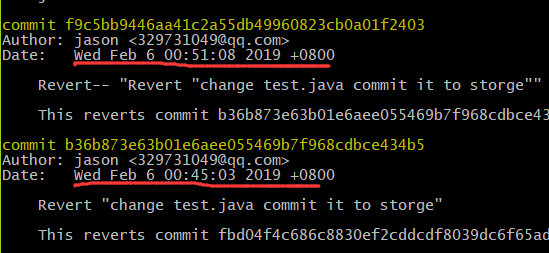
# 14.**差异对比(内容变化)【git diff】**



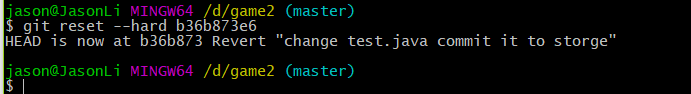
## 举例1：工作区和缓存区的差异



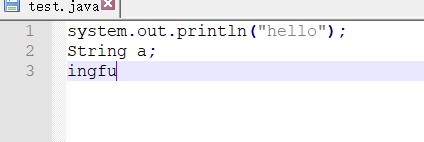
## 举例2：两个版本的内容对比：



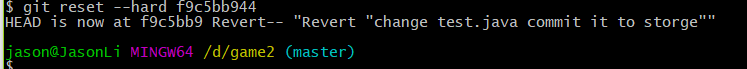
切换版本：b36b873e6

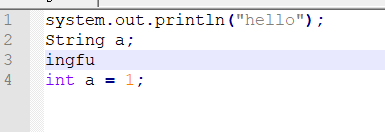


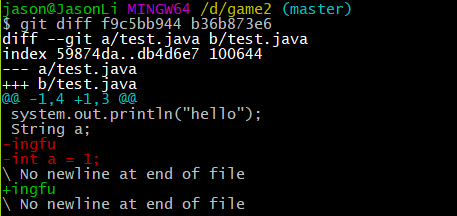
Test.java的内容：



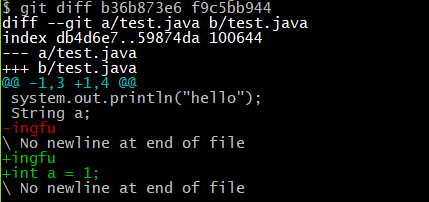
切换版本：f9c5bb944



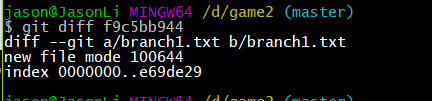


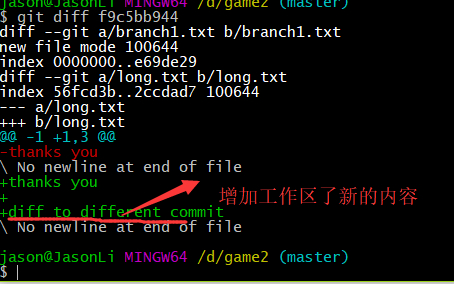


反过来对比



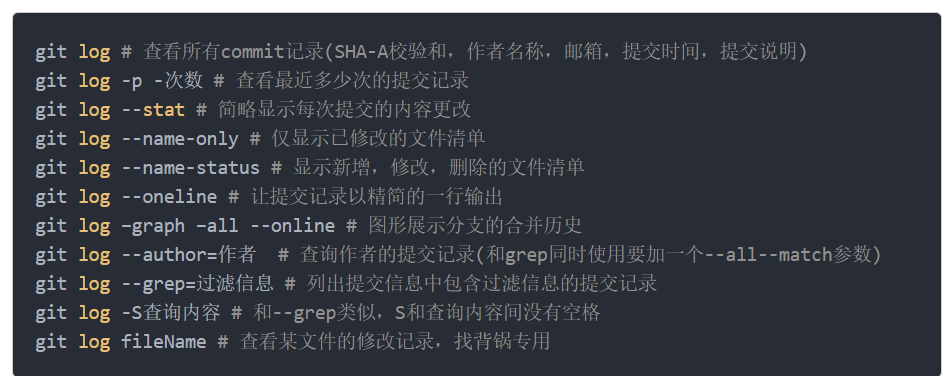
### 举例3：工作区域指定版本的比较：





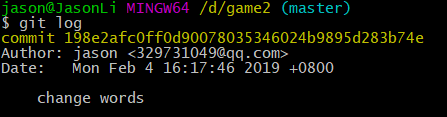
# **15.查看历史提交记录【git log】**

暂存区commit到本地仓库的历史查询

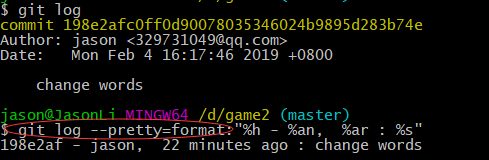




以上是由于还没把任意文件提交到本地仓库的缘故



## Log 的信息过滤



## Log 过滤的占位符说明：



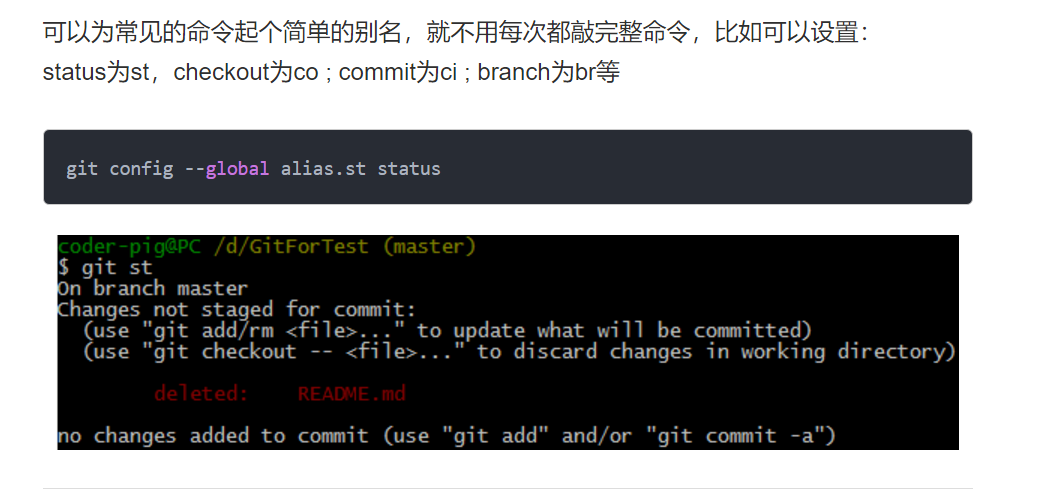




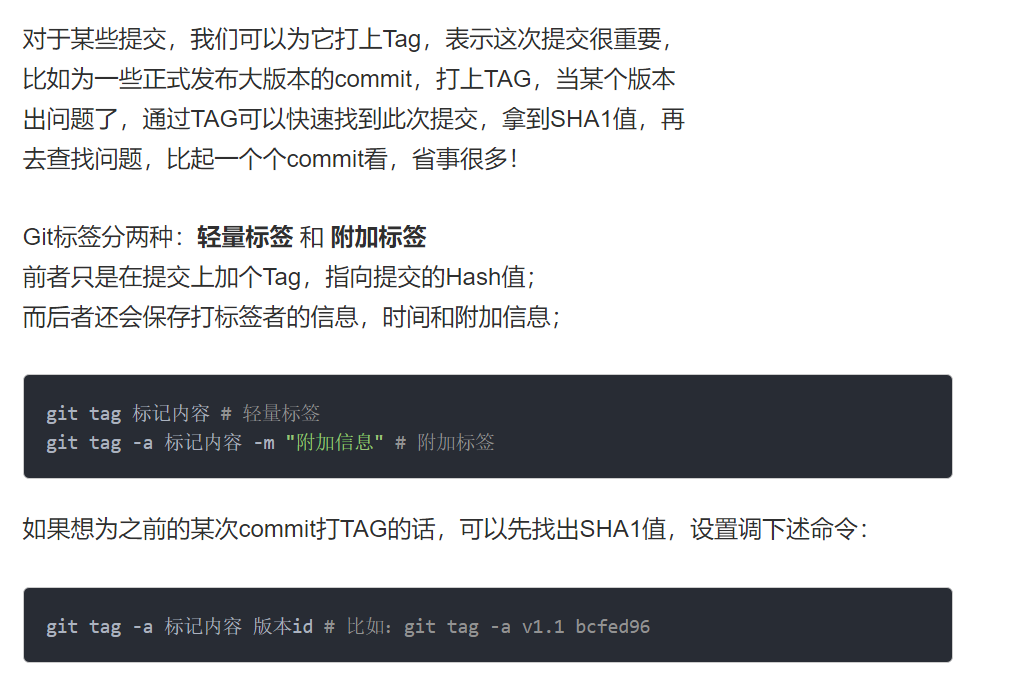
# 16.**查看某行代码是谁写的【git blame】**



# 17.**设置Git命令别名【git config --global alias】**



# 18.**为重要的commit打标签【git tag】**





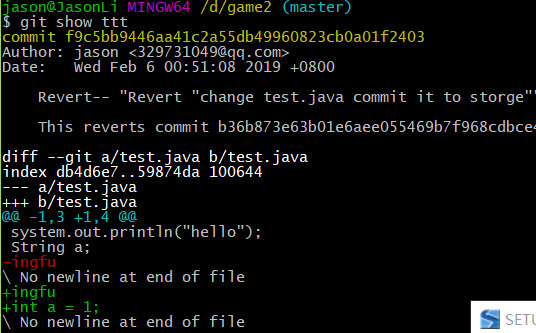


举例：

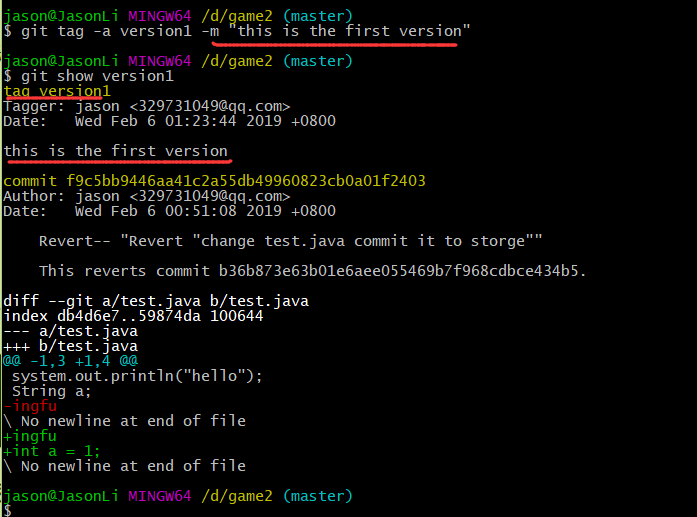
在当前的commit上加标签：



显示该标签：



举例2：

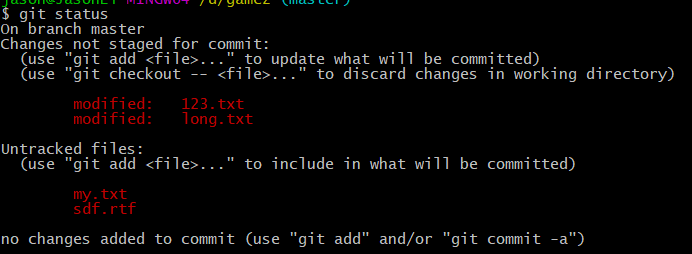


# **Git命令自动补全【输命令的时候按两次tab键】**

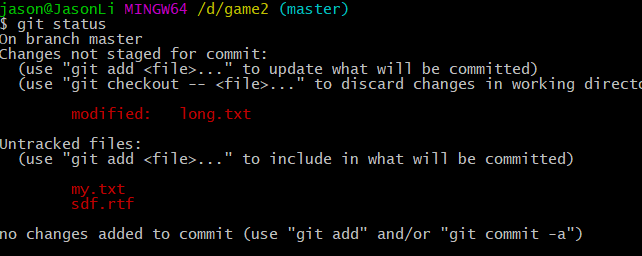
# **文件恢复(未commit)【git checkout】**



**举例：**

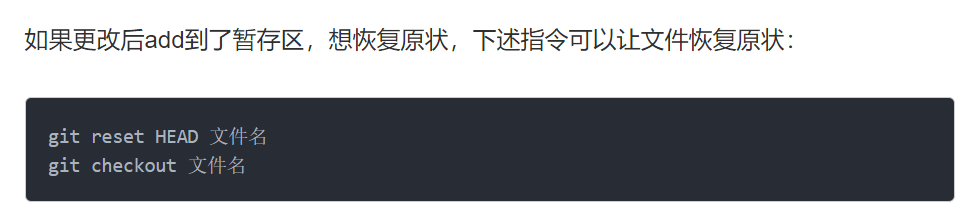






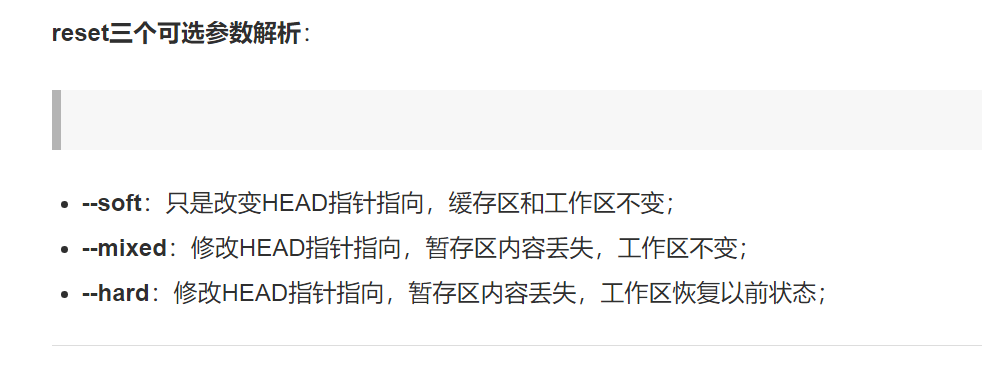
123.txt文件被从暂存区还原到工作区了，工作区的文件被覆盖了。。

# 21.

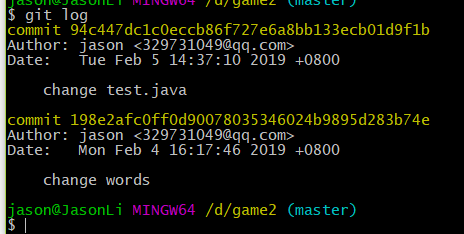


# 22.**版本回退(已commit)【git reset --hard】**

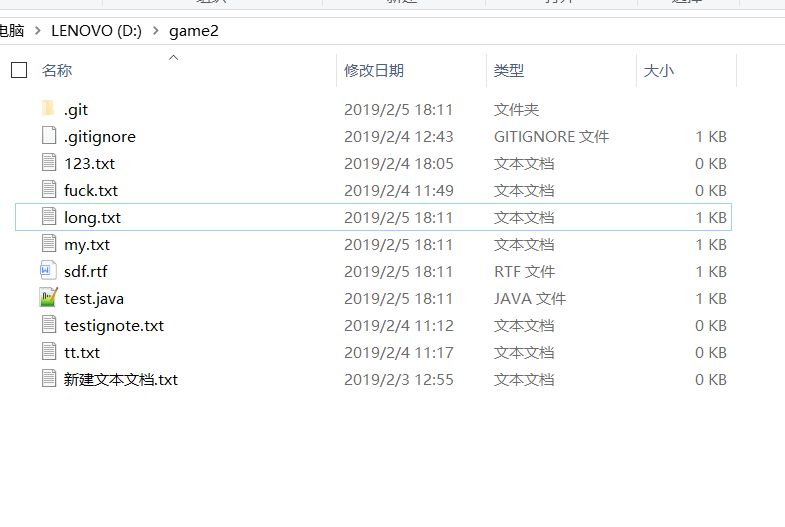




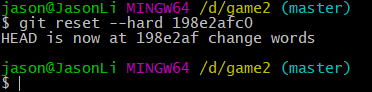
举例：

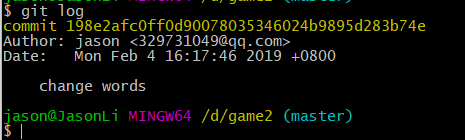


工作区间：

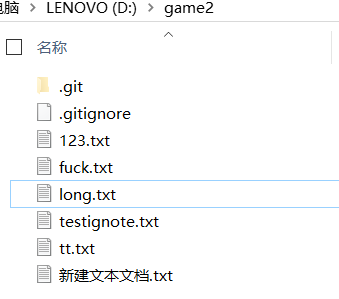


**回退到 198e版本**

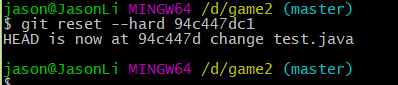


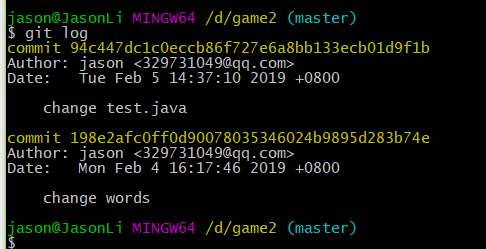


结果：

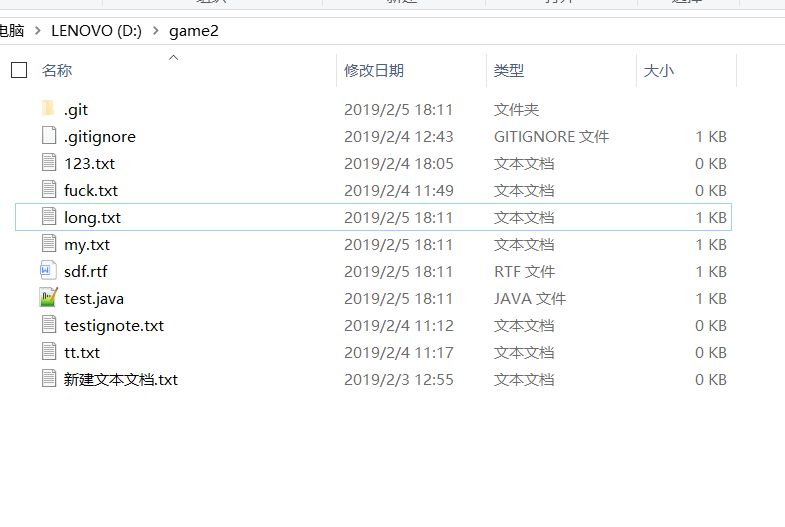


**又回到最新版本：**

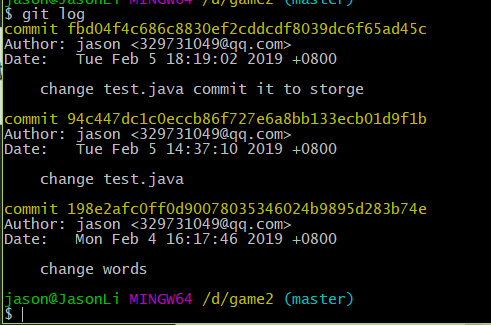


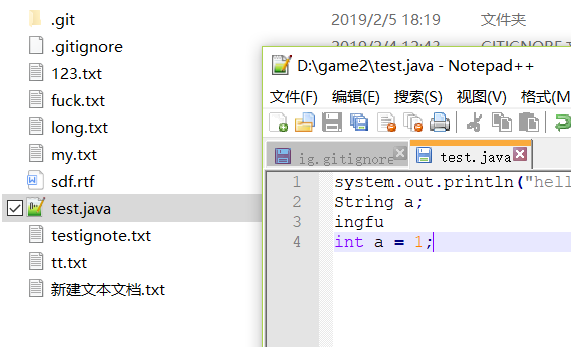


工作区间：

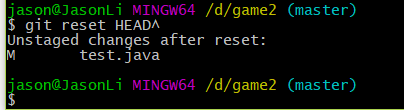


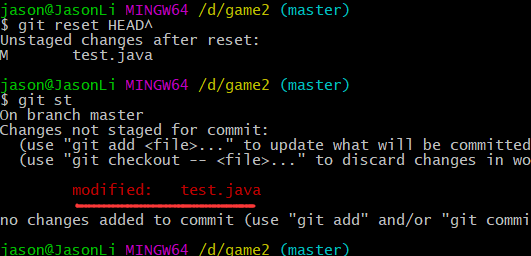
**举例2：**

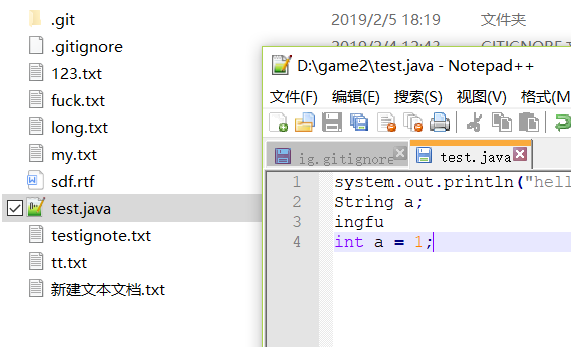




**回退到上一个版本：**



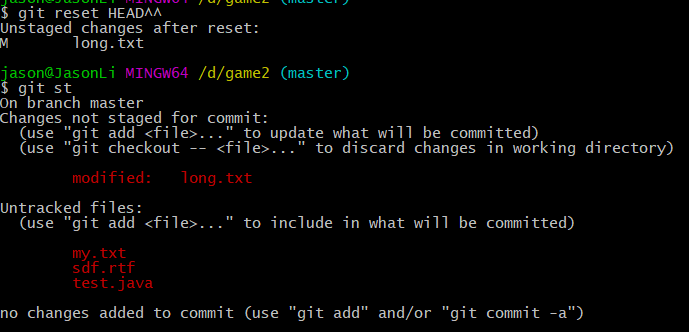




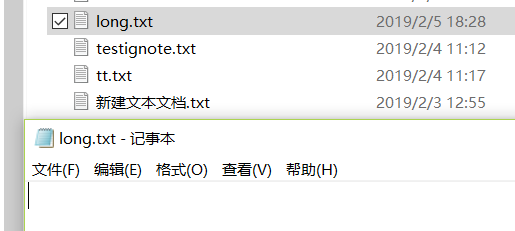
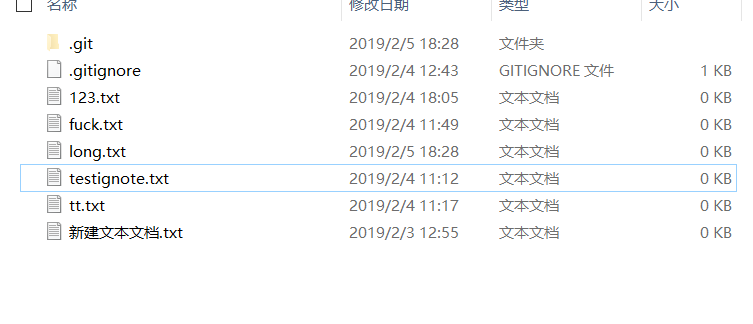
回退到上一个后，保留了modified的内容

**举例3：**

**回退到前面两个版本：**



两个版本前的版本：



## git reset HEAD^ 和git reset --hard xx的区别：

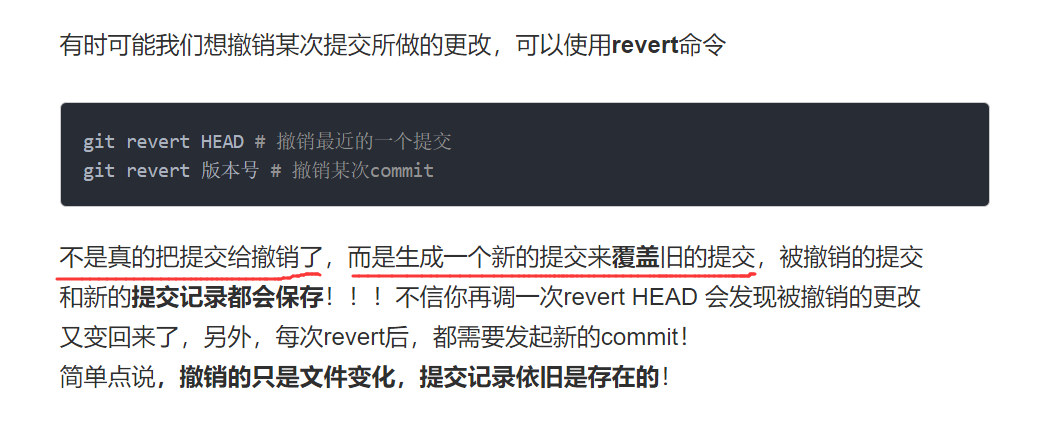
说名了一个问题：使用 git reset HEAD^的命令语句，会把版本回到前一个版本，但是工作区的内容不会删除，会与暂存区形成对比

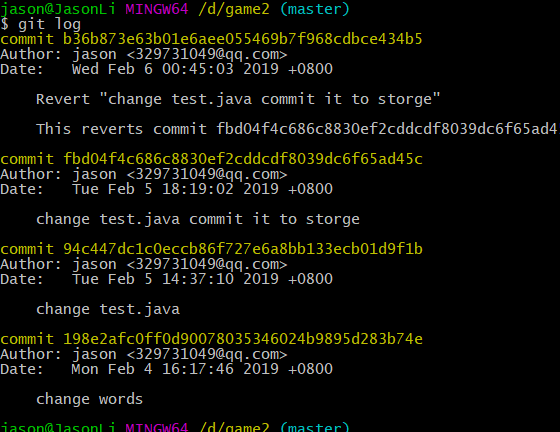
而是使用 git reset --hard xxx命令语句：则是在各个版本自由切换

# 23**查看输入指令记录【git reflog】**



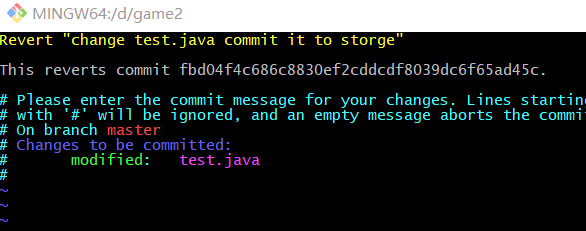
# 24**撤销某次提交【git revert】**



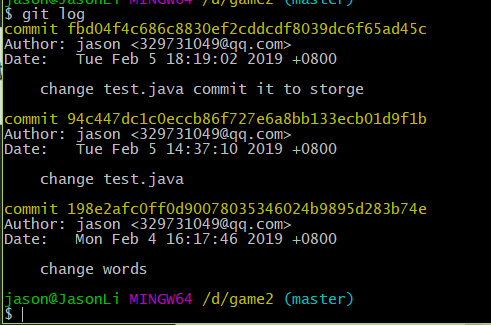


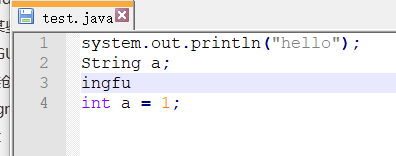
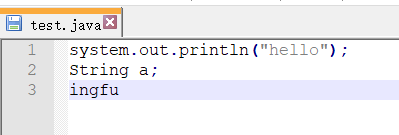
撤销：



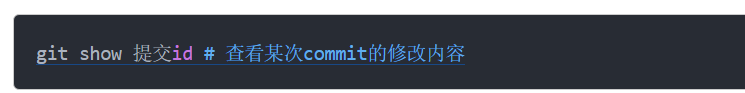


撤销后：



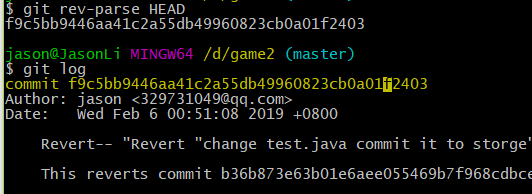
变成 

# 25.**查看某次提交修改内容【git show】**

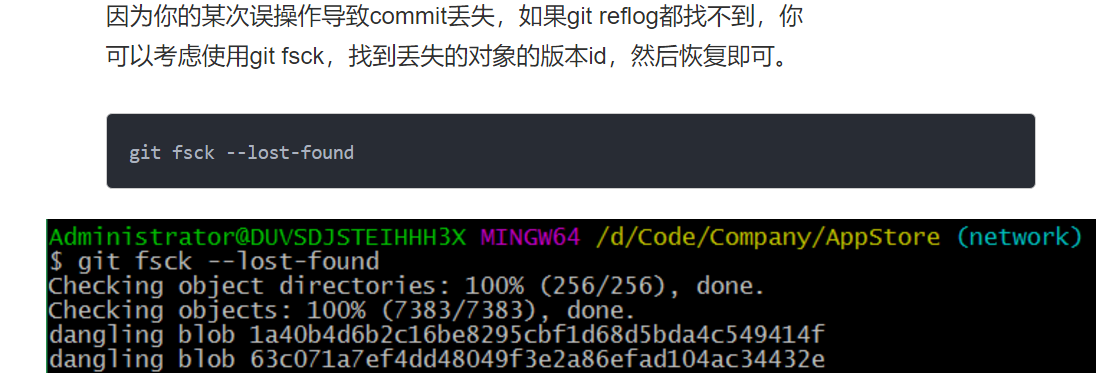


# 26.**查看某个分支的版本号【git rev-parse】**



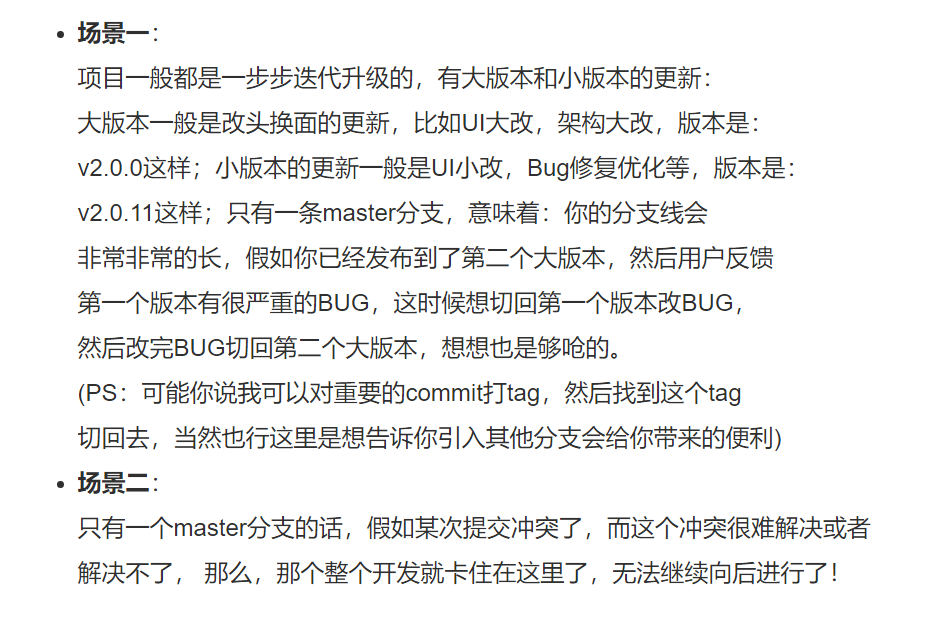


# **27.找回丢失对象的最后一点希望【git fsck】**

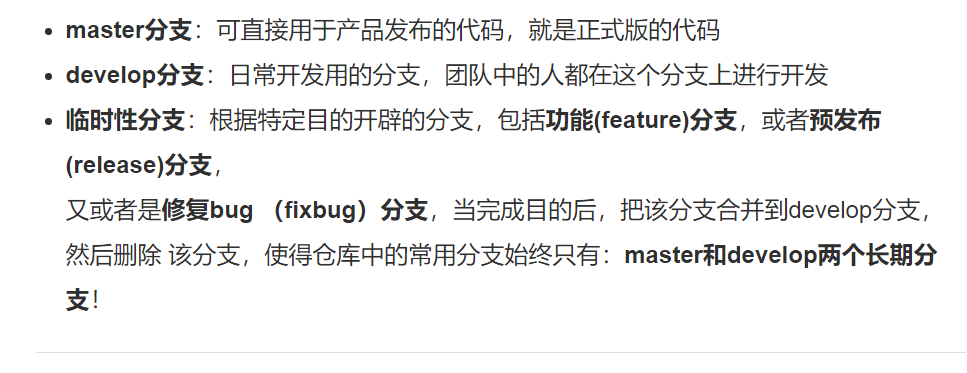


# 分支

## 1.Github的分支策略





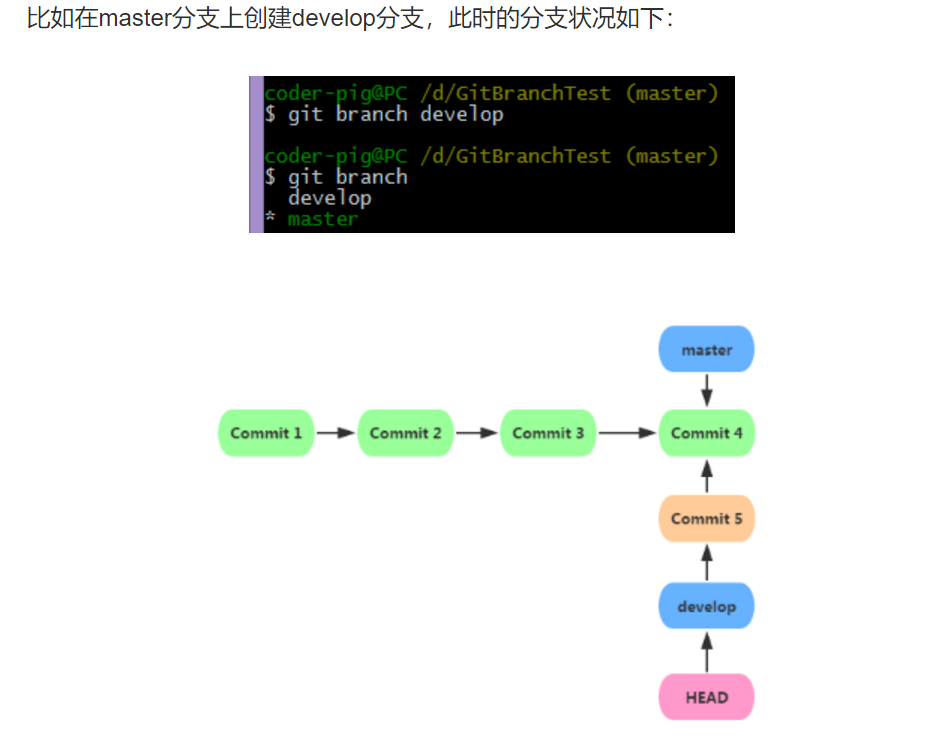


## 2.创建，查看，切换分支

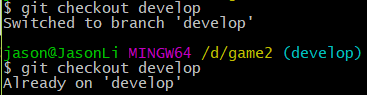




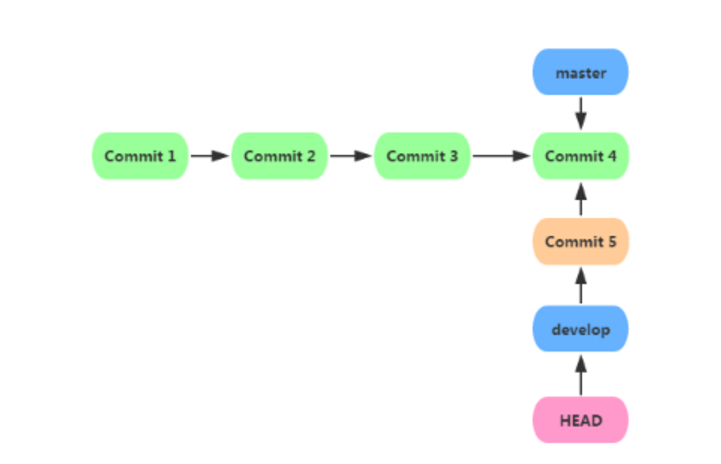
举例：

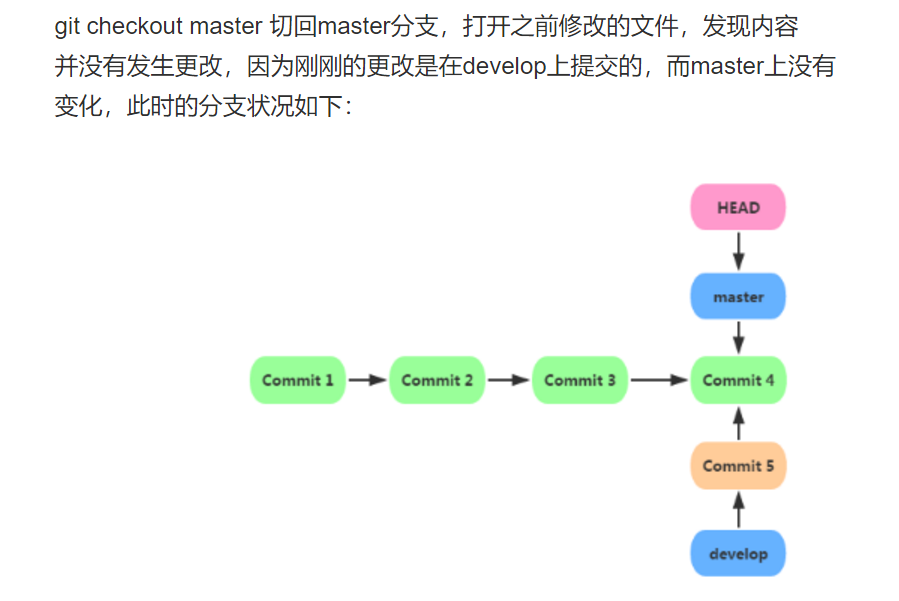


切换分支：



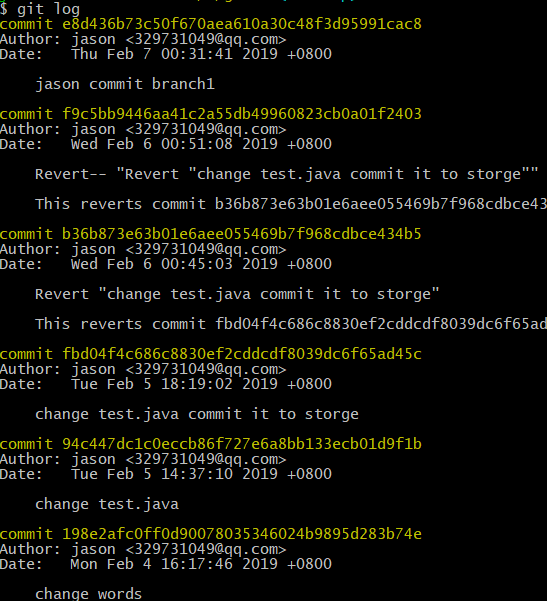




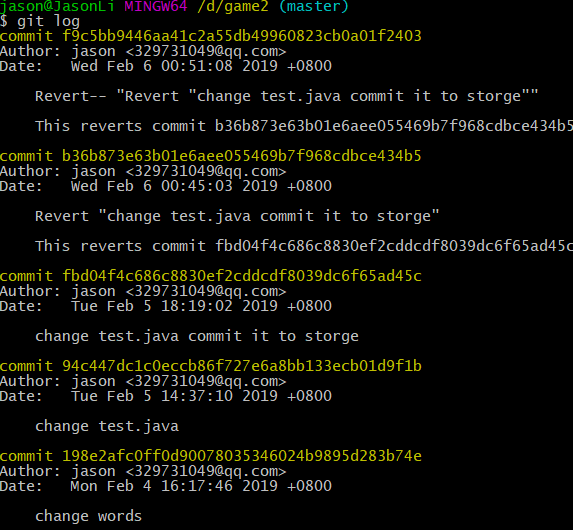


举例：在分支上建立了branch1文件

分支的log:



主干的log:

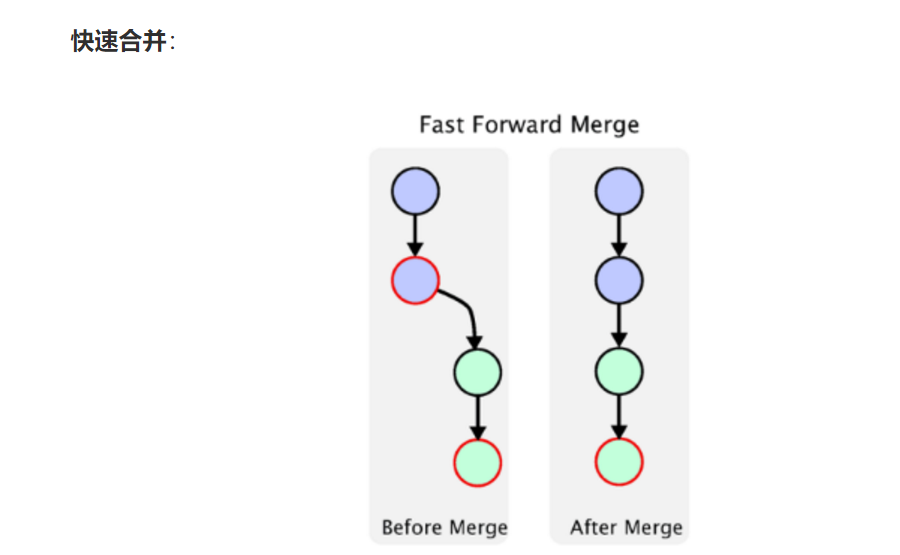


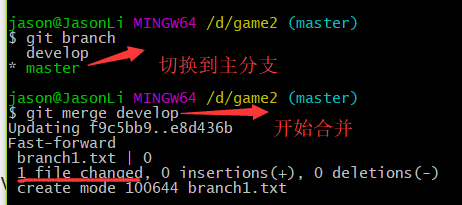
## 3.分支的合并,【git merge】 VS 【git rebase】

### git merge合并分支

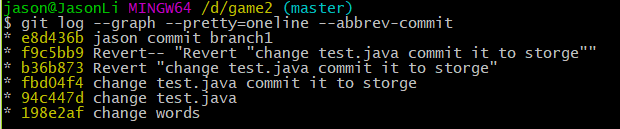
合并的方式分为两种：****快速合并**** 和 ****普通合并****，两者的区别在于：  
前者合并后看不出曾经做过合并，而后合并后的历史会有分支记录

#### 快速合并：git merge xxx分支

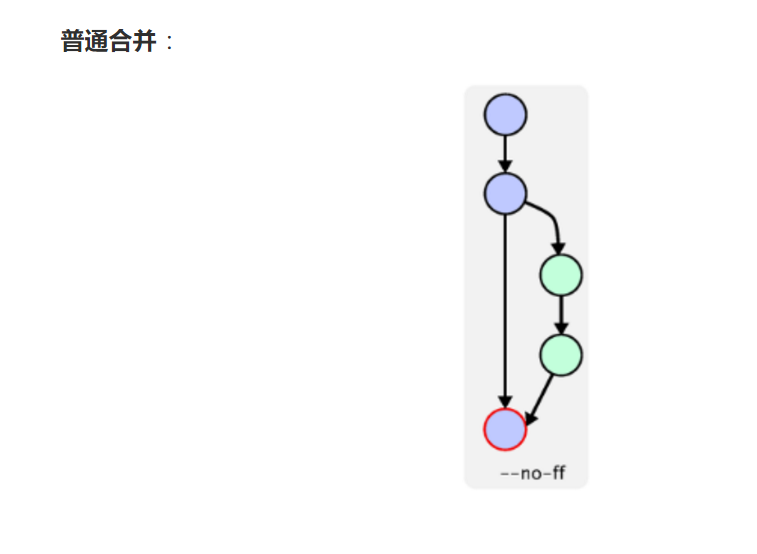


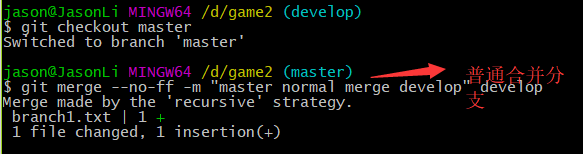


查看分支情况：

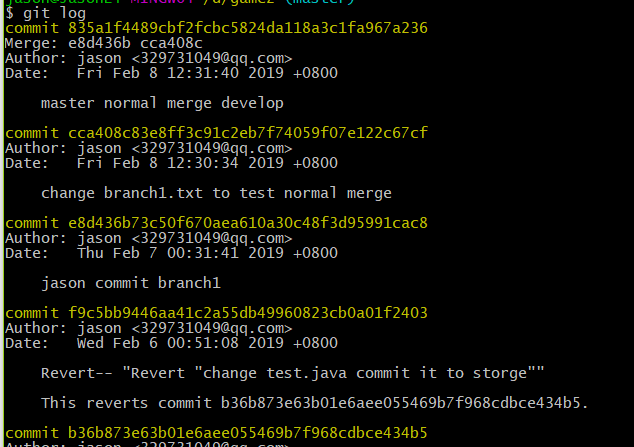


#### 普通合并：git merge --no-ff -m “message” xx分支

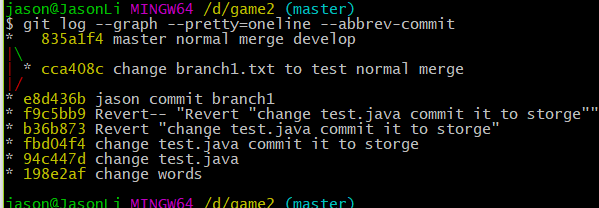




版本日志： git log



过滤的版本日志：



### git reabse合并分支