git-代码版控制器

[1 标题1 错误!未定义书签。](#_Toc449540698)

[1.1 标题2 **错误!未定义书签。**](#_Toc449540699)

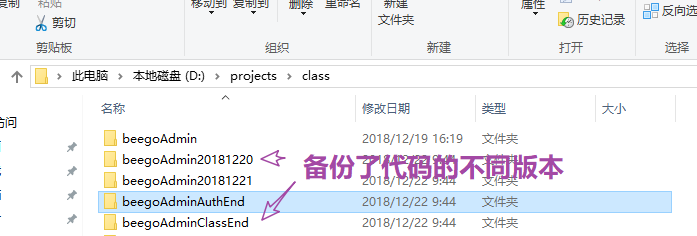
[1.1.1 标题3 **错误!未定义书签。**](#_Toc449540700)

# 概述，什么是代码的版本？

beegoAdmin项目，由很多功能模块，由很多天完成。代码会出现，增增减减，来回修改情况。如何才能快速，更加完整的得到每个阶段的代码呢？

典型的两个办法：

* 根据时间来进行备份，例如下班前，或者午饭前。凡是不准备继续敲代码了，就将已有代码进行一次备份，记录备份时的摘要信息。
* 根据功能进度来备份，例如班级添加完成时，班级管理模块完成时，每当一个功能结束的时候，做一次备份，记录摘要信息。



以上就是代码的版本管理，是我们手动完成的。以上的代码版本，也成版本快照。

实操时，需要一个更方便的代码的管理工具来使用，就是git。也有其他的很多版本库管理器（svn，cvs，bitkeep .etc）

# 快速使用git

## 安装git

windows下安装：

官网：<https://git-scm.com/>

下载页面：<https://git-scm.com/downloads>得到最新的git安装

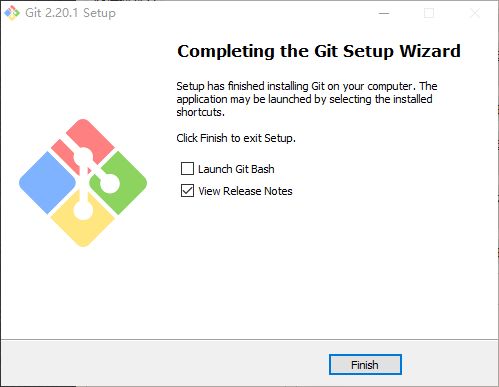
目前是：2.20



运行安装。

未截图的就保持默认即可。

安装位置，如果需要，自定定义，不要安装在特殊字符目录中。（包括中文，特殊符号，空白）



windows上安装的软件：



## 快速体验

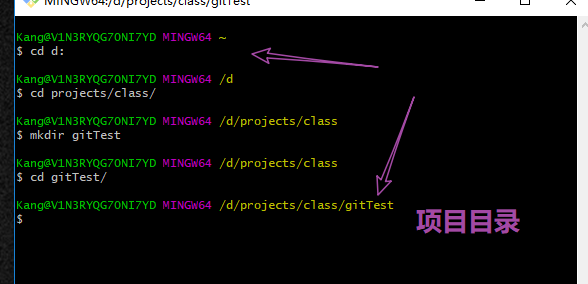
### 初始化版本库

创建项目的版本库，让git来管理代码版本。

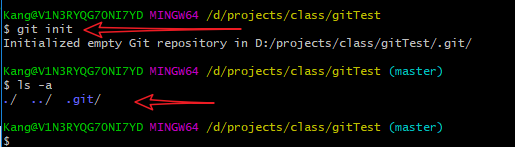
项目目录

gitTest

(建议使用git-bash操作)



执行git初始化操作，git init



成功的标志，就是目录中，存在一个 .git 子目录

### 写代码

写代码不受git影响。

常规开发即可。

假设创建了几个目录，同时创建了几个文件。

用任何编辑器都可以

与git无关。任意操作。

### 提交版本

此时使用git管理。与项目功能无关。

代码在任何节点，时间节点，功能节点。都可以形成一个版本。

一旦形成版本，代码就一定不会丢失了，一定可以找回来，即使后面覆盖了，删除了，也可以找回来。

使用：

git add .

git commit -m ‘信息’

来实现



### 反复编写代码，反复提交版本

时间节点，功能节点，都应该提交一次。



### 版本库重置

要一份刚刚main包初始化好的代码。

由于已经记录了 初始化的版本（此步骤没有，任何工具也没有办法。），使用

git reset 操作将其重置

查看版本号（commit-id）



执行重置：



还可以重置到将来（之前版本的将来版本）的版本。

需要再得到重置时重置掉的某些版本。

本例中，需要再回到，修改了main.go的版本中。

（知道，commit-id即可）

通过 git reflog (关联日志)确定commit-id，再git reset –hard 即可



需要学习的

* 本地版本库管理
* github的使用
* 多人协作

## git help 命令

获取命令的帮助信息

# 配置 config

git config

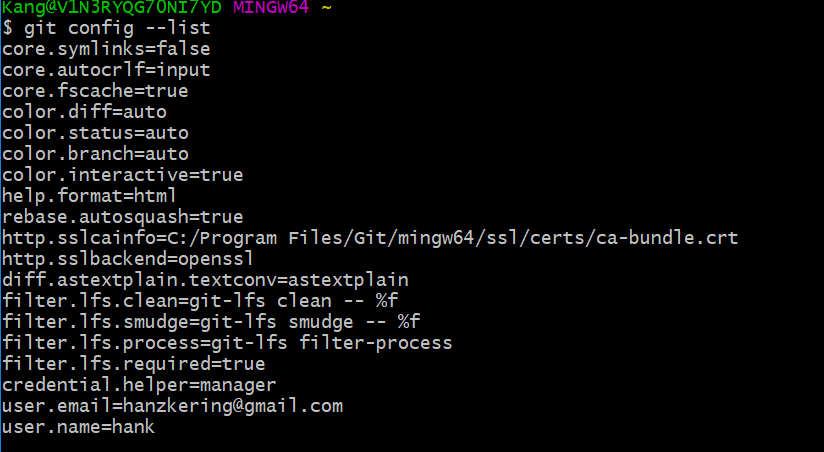
安装完毕git之后，第一件需要完成的事，配置git。

最需要提前做的配置：用户的邮箱的配置。

## git config --list

-l 也可以

列出可用的配置项



user.email

user.name

就是初始化后，第一个需要配置的项。

## --system --global --local

配置项的层级，由大到小（有外到内），顺序如下：

system，当前系统，最外层配置，配置存储在git工具的目录。只要使用git就会启用该配置。

global，当前系统登录用户，配置存储在登录用户的家目录中：~/.gitconfig。针对于当前登录用户生效。

local，当前项目，需要进入到某个git管理的项目目录。默认级别，建议修改配置的级别。

采用 内层会重写外层的配置机制。

建议，大多数的配置，还是应该以项目为配置单元。配置仅仅影响所管理的项目，不要影响其他项目。

关于用户的配置，可以（建议）配置在 global层，因为global层也是用户的概念。

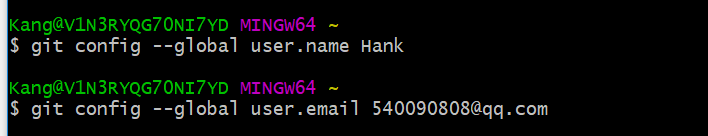
## 配置, git config

git config --level user.email [xxxx@yyy.zz](mailto:xxxx@yyy.zz)

用户级别，使用--global,不写表示项目级别，在项目目录使用。

$ git config --global user.name Hank

$ git config --global user.email 540090808@qq.com



任意的name和email。（后边与github交互时，要设置为github关联的邮箱）

## 其他配置

在具体功能时，再议！

# 初始化 init

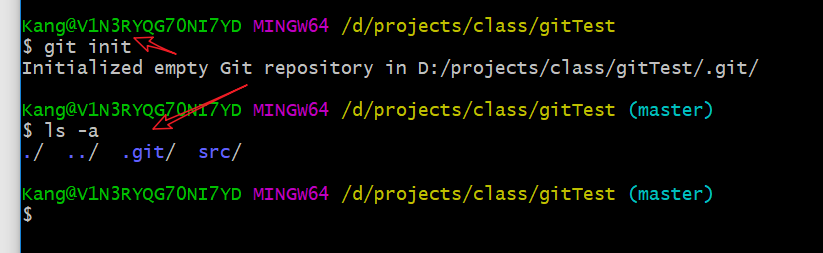
将当前项目，作为git管理项目。

在项目目录形成 .git /目录, 用于管理.git的全部对象。例如，版本信息，文件改动信息等待。

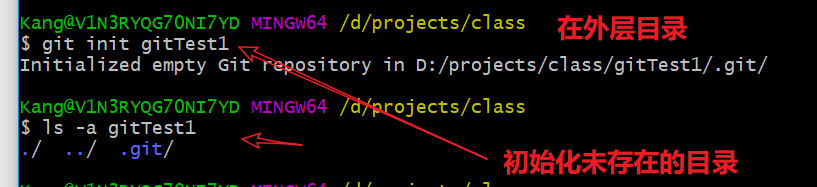
使用 git init 来实现。

使用方式：

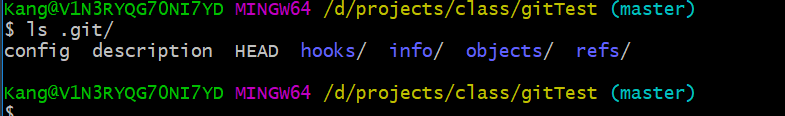
初始化已有项目不论项目是否有代码，进入目录，执行git init



直接使用 git init 创建目录，执行 git init 目录名



.git/目录内容



如果需要，将配置在该项目独立配置，则进入项目目录配置即可！

# 提交版本，add，commit

命令：

针对于存在新文件（未被追踪的文件）

git add .

git commit -m ‘message’

针对不存在新文件的情况：

git commit -a -m ‘message’

理解提交，需要知晓以下的概念：

工作区（workSpace），暂存区（stageSpace），版本区（versionSpace）

涉及的命令：

git add

git status，查看当前文件状态。文件状态，未被追踪，已被追踪，已被修改，未被暂存，已经暂存，未被提交，已被提交。



git add 文件列表

需要被暂存的文件列表，通常使用 . 表当前目录下的全部修改。

git commit 需要使用 -m 选项，设置一个提交的版本信息。不适用-m 会弹出一个编辑器，要求键入版本信息。编辑器默认是vi。



弹出vi编辑器：



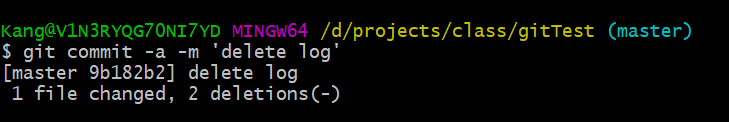
先输入 i

输入一些message信息

使用 ESC，在输入 :wq 即可

随时使用 git status 查看当前状态，得到提示信息。更好的操作。

如果没有新文件（新目录）。可以直接使用 git commit -a -m ‘message’ 选项完成 一次性 添加到暂存和提交操作。（省了git add 的使用）

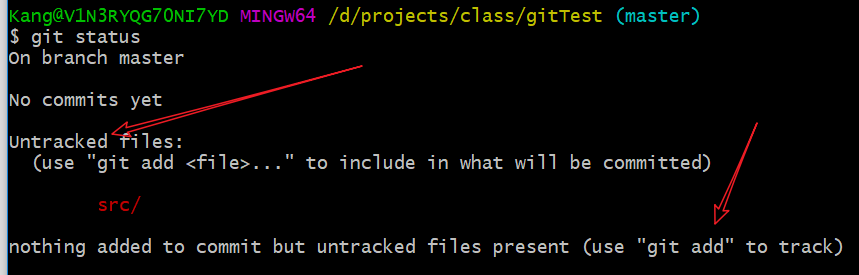


# git status，文件状态

## 未被追踪

当文件第一次出现在工作目录时（或者版本版块刚刚初始化后）。版本库（git），还不知道有这个文件存在时。文件处于未被追踪：untracked：

（刚刚初始化的项目目录）



## 被暂存

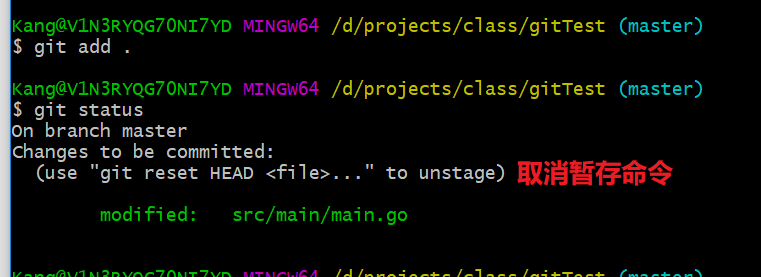


## 修改未被暂存

文件已经被追踪。文件被修改之后。处于该状态。

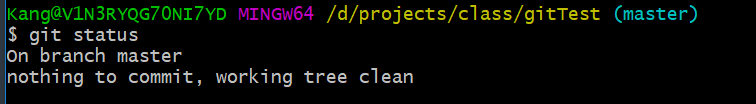


此时提交后，文件处于暂存区：



## 干净的工作区

当全部的修改被提交后，此时工作区状态：clean

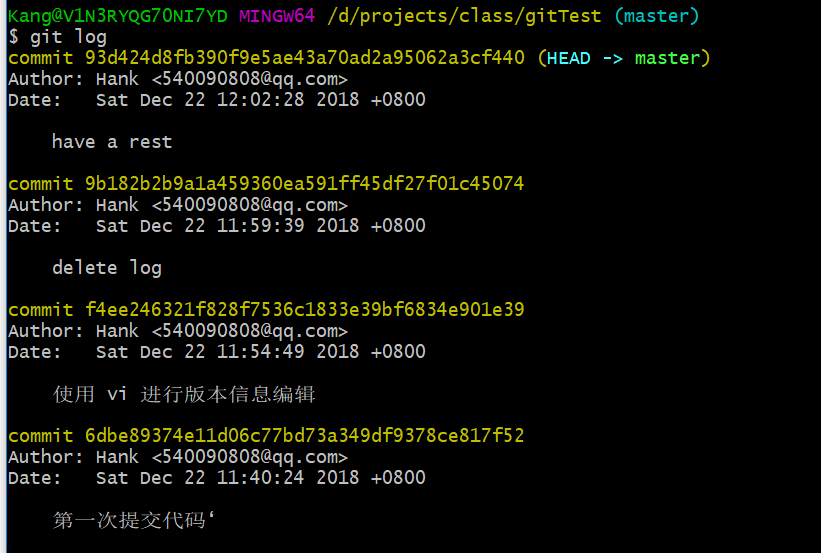


# 日志，log，reflog

## git log

通过 log 命令，来获取当前版本库的版本提交日志。

git log 常规日志



展示了 commit-id, Author, Date版本ID，作者信息，提交时间

当日志很多时，需要翻页才可以获取全部日志信息

使用 回车（下一条），空格（下一页），来翻页。

使用 q 退出查看。



## 支持日志过滤

当需要获取部分的日志时。例如，某个时间段的提交，或者某个特殊的作者的提交。

使用选项：

--author=<pattern>

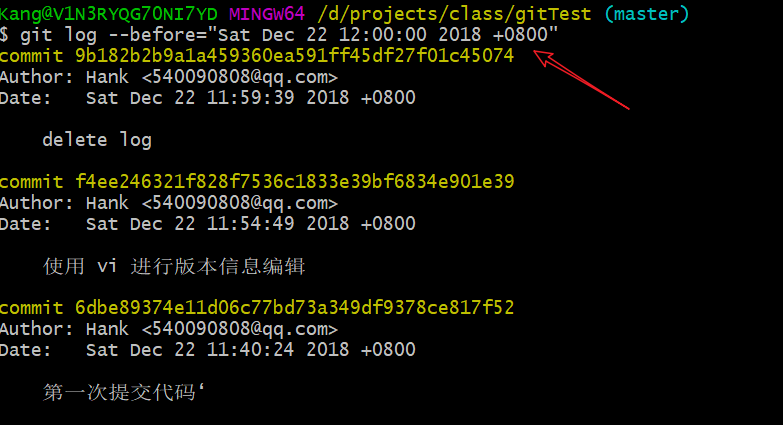
--until=<date>，直到某个时间点

--before=<date>，在某个时间点之前，与上同

--since=<date>，从某个时间点开始

--after=<date>，在某个时间点后，与上同

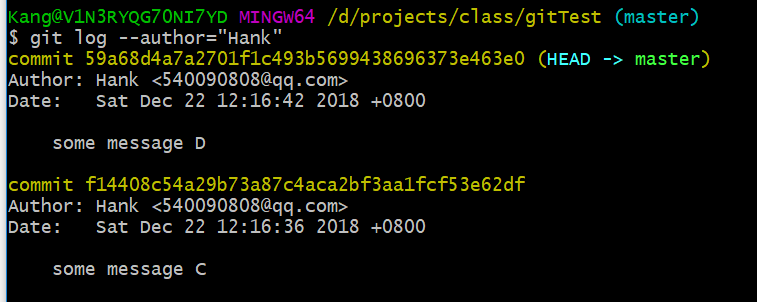
演示，获取



可以混合使用：



作者过滤：

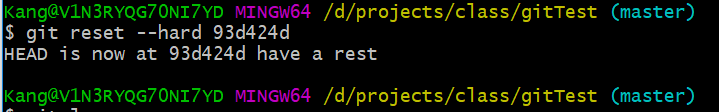


## git reflog

记录版本间的关联的状态。

主要用于记录，每个版本是如何产生的。即使是回退（重置）后，版本也会在关系日志中被记录。

reset：



reflog：



# 重置，reset

将版本重置到固定的其他版本。只要是在版本库中，出现的版本即可。无论是是否被回退的版本。

## git reset 来实现

就是 git reset --mixed

不适用任何的 reset 选项，使用默认行为。

行为是：

版本被回退到指定的目标版本，但是工作区文件内容并未被实质性的回退到目标版本，版本库同时记录了目标版本与当前工作区文件的修改差异。

或者说：版本库回退了，之后将文件修改成之前的样子，但没有提交（暂存）。

演示：

reset



但文件内容未变：



此时工作区状态，git status:



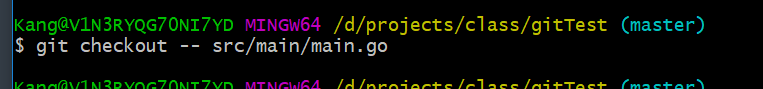
查看文件差异，git diff

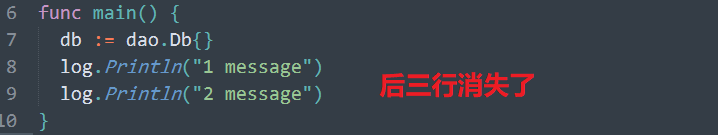


可见：git reset 重置了版本，同时记录了版本间的修改。便于我们后期去编辑修改！

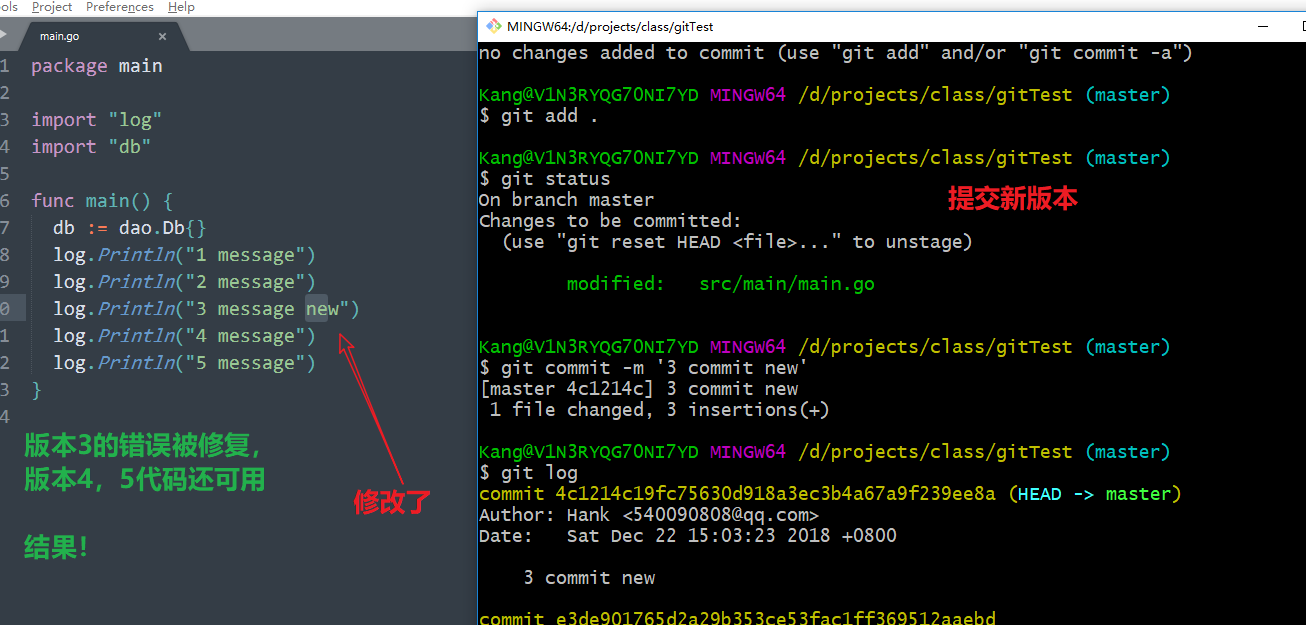
完毕后，可以选择：

1. 放弃版本间的修改，使用git checkout -- 文件





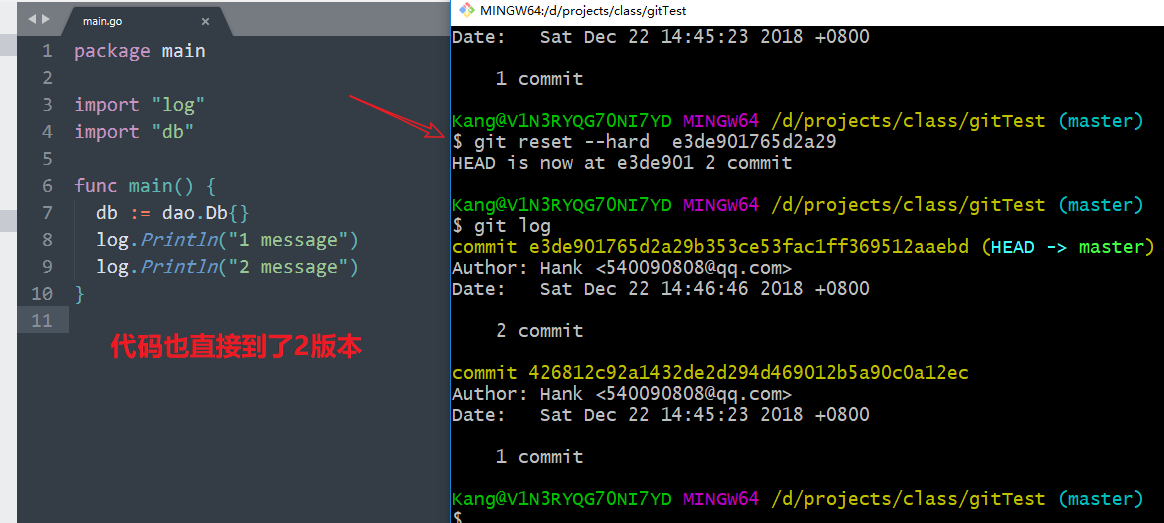
1. 再修改后重新提交



## reset --hard，硬性重置

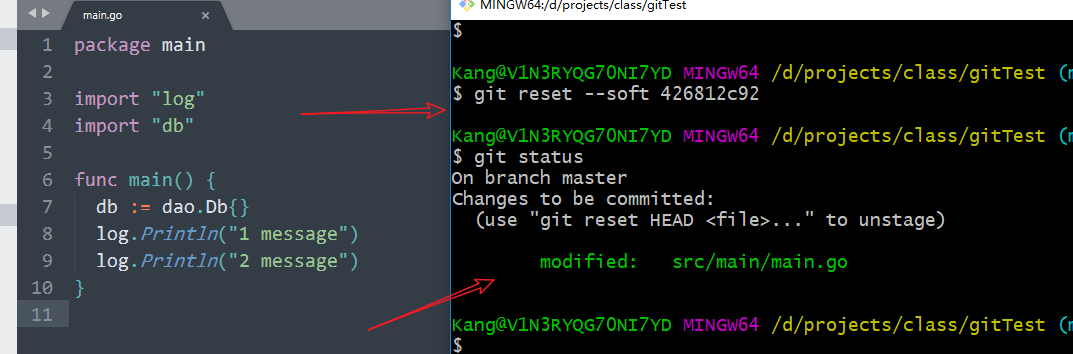
与mixed重置相比，不但重置了版本库信息，也同时重置了工作区。

代码直接重置到了某个版本上：



## reset --soft，软重置

软重置：将版本库重置，但代码（工作区）并没有重置，同时记录版本间的修改，同时将修改添加到了暂存区（相对的mixed重置没有将修改添加到暂存区）。



## 类比：mixed hard soft

hard = mixed + checkout --

soft = mixed + add

## git checkout --,放弃工作区的修改

是 checkout --, 而不是checkout！

如果需要将工作区中的修改回到当前版本的初始状态。

需要使用 git checkout – 完成

此命令针对 工作区的，有修改但为暂存的。（已暂存的不能使用该命令）



## git reset HEAD，取消暂存

例如，已暂存的文件状态：



执行放弃暂存：



此时，若也不想保留修改，则使用 git checkout -- 命令完成

## HEAD，头指针

用于指向当前的版本。

reset操作，其实就是修改HEAD的操作。



你当前所处的版本，就是HEAD来指向的。

使用HEAD，完成版本的重置。

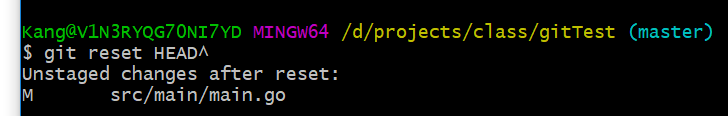
HEAD使用用来相对的重置版本的。

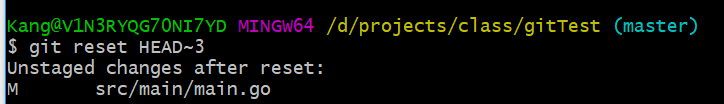
HEAD^，前1个版本，HEAD~1

HEAD^^^^^^^^^^，前10个版本, HEAD~10

HEAD~99，前99个版本。

演示：





# tag，标签

版本号（）别名，使版本更具有语义化。典型的别名，就是版本号：

v1.0

v0.9

v3

v5-meta(内测)

v4-rc(公测)

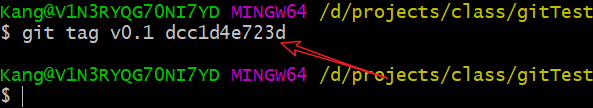
v6-release(发布)

语法上使用 git tag 来完成管理

## 添加标签

git tag 标签 版本号

可以在某个版本上增加标签



不适用版本号，标识当前版本号：



## 查看标签

git tag

列出所有标签



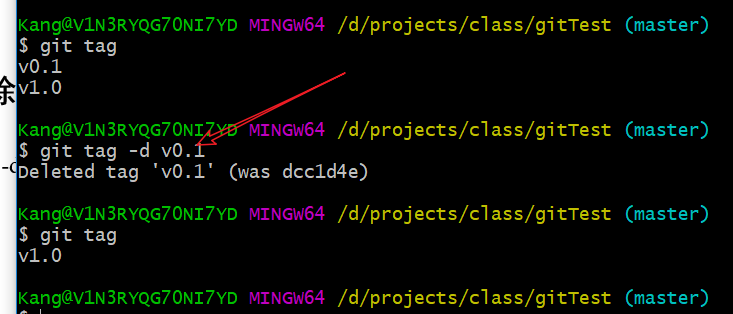
git show 标签

查看标签对应的版本信息



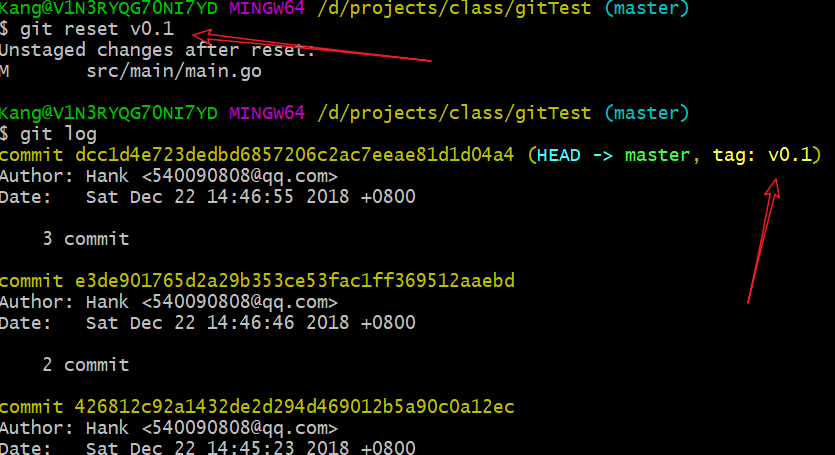
## 删除标签

git tag -d 标签



## 利用标签进行重置

git reset 标签



## 远程标签的管理

未完待续！

# 远程版本仓库，协同开发

git，分布式的代码版本库控制器。

分布式，版本库可以分散到网络上！

当需要多人协作开发时，同时会使用多个本地版本库，与同一个远程版本库相互的模式。



远程版本库：remote

本地版本库：local

本地推送到远程：push

远程拉去到本地：pull

将远程克隆到本地：clone，用于建立远程和本地间的联系

## 创建远程版本库

要求是一个没有工作区的版本库（不应该在远程版本库上操作）只有版本库的版本库，称之为裸版本库。

使用 git init -- bare 来实现



统一的命名方式为.git 结尾的文件夹名

--bare 导致目录中没有.git目录了。

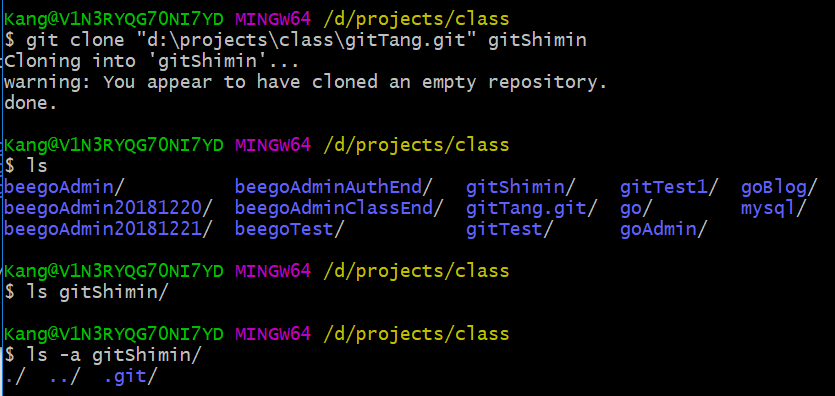
## 通过克隆得到与远程版本库相关的本地版本库

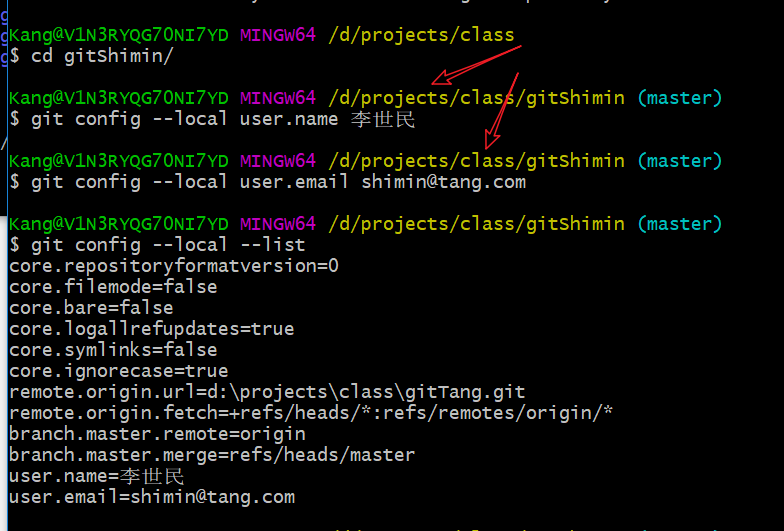
以 李世民的身份，克隆，并将用户设置为李世民

git clone 版本库地址 本地项目目录

其中 版本库地址，由于本机的远程版本库，使用文件路径即可。若是远程的，需要使用远程地址，例如域名什么的。

clone，标识本地仓库和远程仓库是一致的。





以李建成的身份，克隆，并将用户设置为李建成



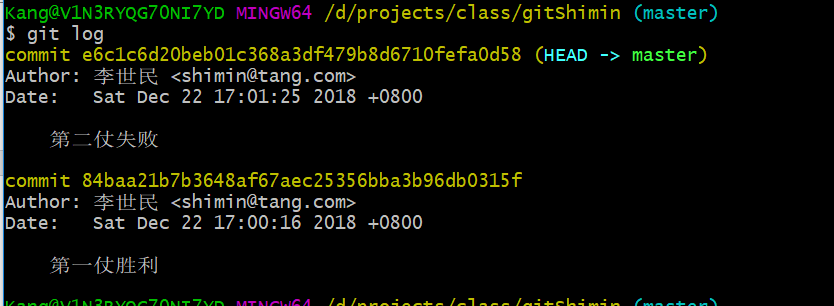
## 推送，push

本地版本维护完毕后。将其推到远程。

李世民，维护了本地版本。

推送到远程

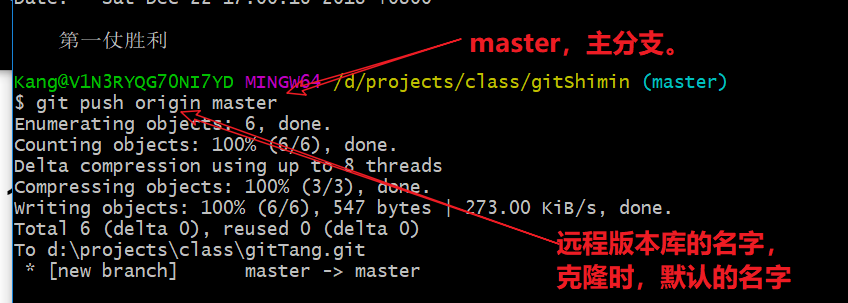
李世民：提交本地版本：



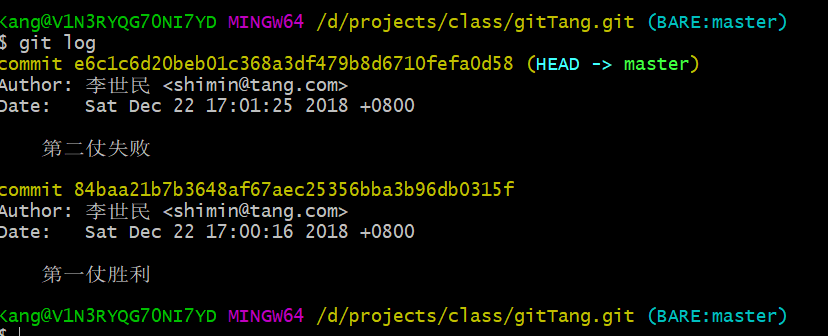
此时，远程和其他本地都没有该版本

李世民，推送到远程：

clone的本地会采用下面的方式。origin master



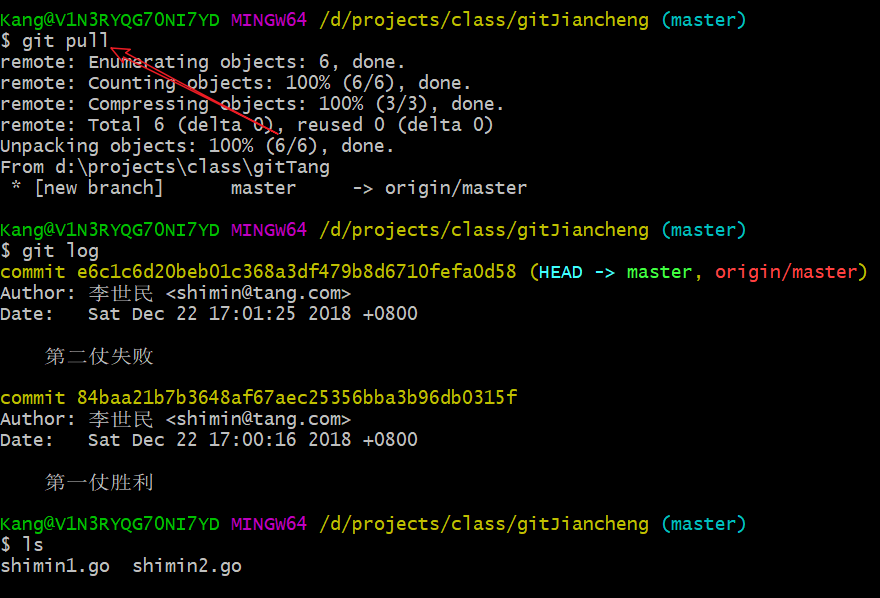
此时，远程版本库上，已经存在被推送的版本了：



注意：实际操作时，push推送之前，一定要先pull拉取。因为在你推送前，其他人也可能会将代码推送到远程版本库了。

## 拉取，pull

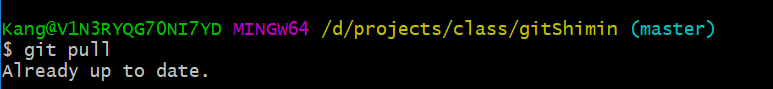
演示，李建成，从远程版本库，拉取其他人（李世民）提交的版本：



注意：实际操作时，push推送之前，一定要先pull拉取。因为在你推送前，其他人也可能会将代码推送到远程版本库了。

当本地与远程版本库一致时，执行拉取，提示 up to date.

什么都不会做！



## 多用户推送时非冲突情况（什么都不需要做）

摘要：

推送前，先拉取。后拉取的用户，需要将远程和本地合并。由于没有冲突，直接保存合并版本的commit信息即可。

非冲突指的是，用户没有修改相同的代码。

李世民：新建文件 进行推送





李建成，新文件，在推送



在执行git pull 拉取时，由于远程版本库与本地不是同步了（李世民已经推送了），拉取时，现将远程的内容与本地的进行合并，在没有冲突的情况下，git自动合并就可以直接成功。需要去编辑一个合并后的版本信息。默认信息已经有了，只需要保存即可：

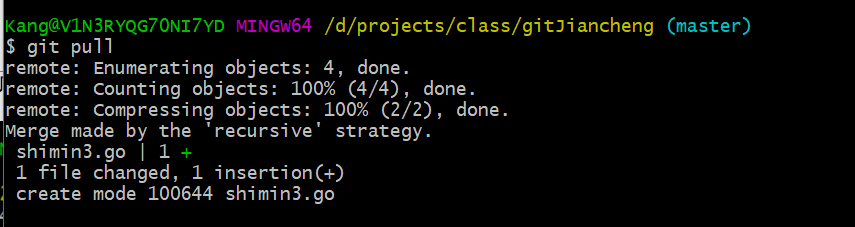
键入 git pull 会弹出vi。

vi的保存(英文下)：ESC :wq 键入回车

即可



pull 拉取：



git log 发现合并了一个新版本：

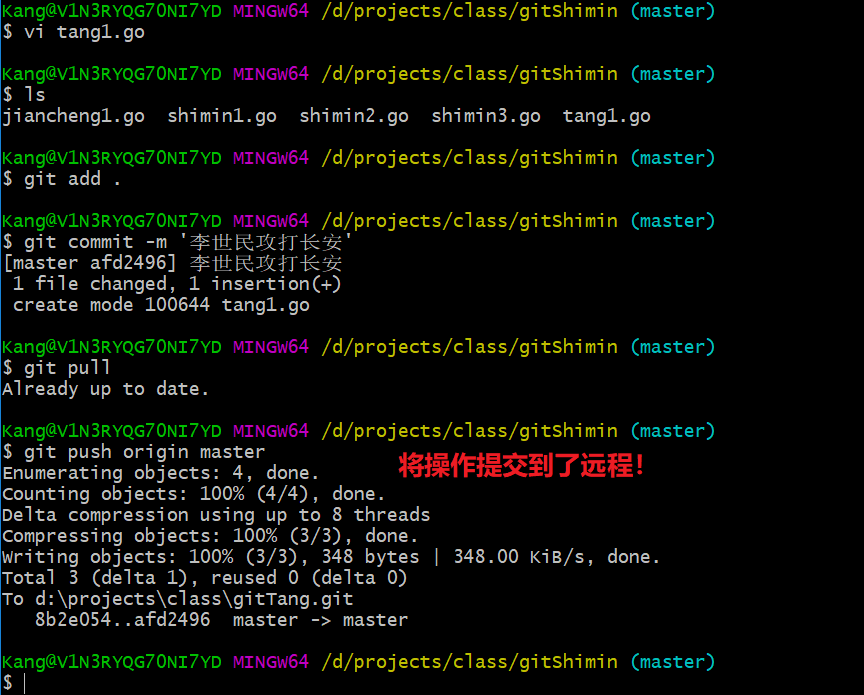


再推送到远程即可

## 多用户推送时存在冲突的情况（需要手动解决冲突）

李世民和李建成维护了同一个文件，产生冲突。

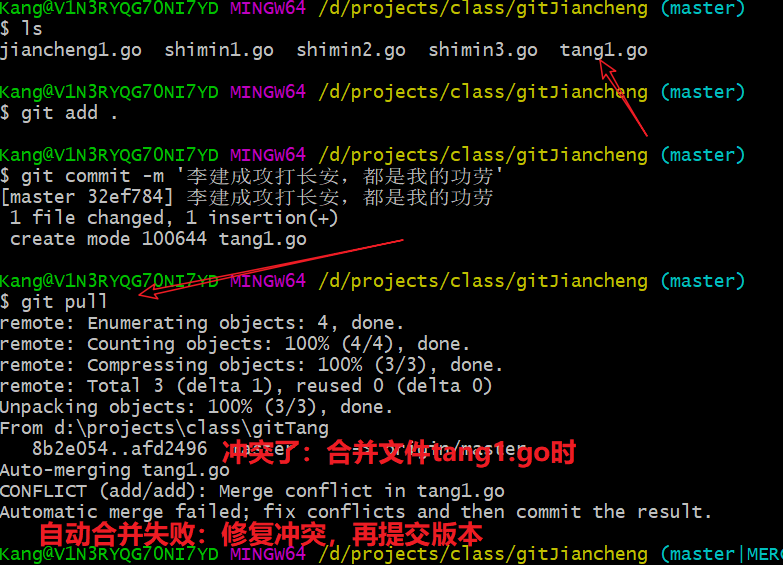
李世民：



李建成：编辑了同样的文件tang1.go

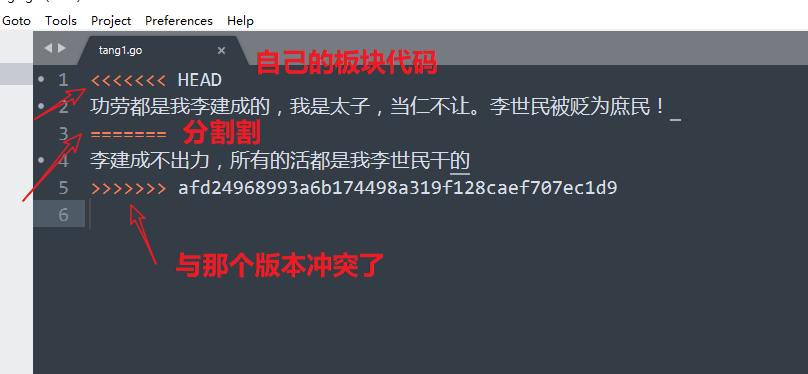
本地提交完毕，准备推送时，先拉取，此时问题出现。

有代码冲突未被解决：



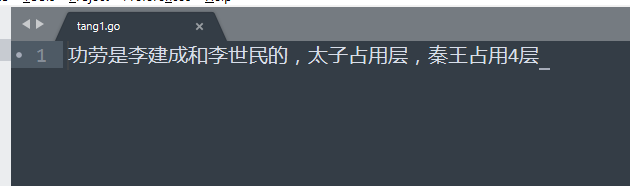
此时自动合并失败（在没有冲突时，自动合并会成功）。

需要李建成，修复这个冲突，根据提示，修改冲突的文件：tang1.go

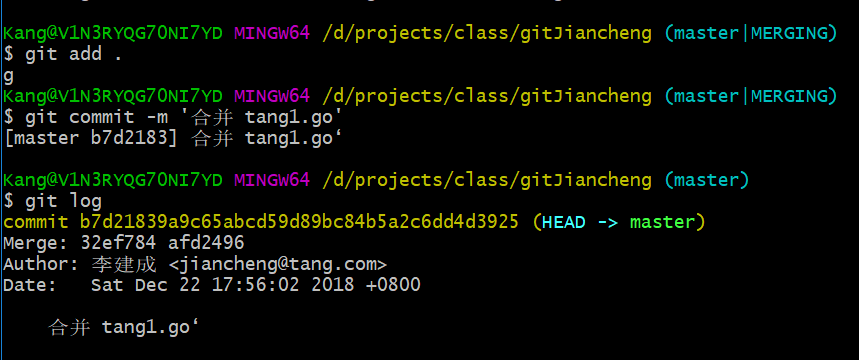


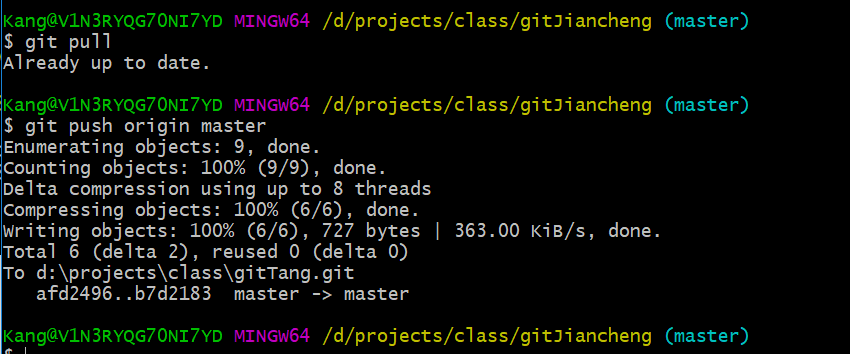
以行为单位检测是否一致的。

解决冲突，两个作者协商解决。通过commit-id确定其他作者（在公共远程版本库查看到对方信息）。



修改全部的冲突后，提交新合并版本：





## 对比 冲突与非冲突

过程是一致的：

git pull 时候，git先尝试自动合并。若成功，自动提交一个新版本。若失败，表示有冲突，手动解决冲突，手动提交合并版本。

# github