git

[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)

# hello git

代码版本控制器

用于保存各个阶段, 代码的状态. 便于后期的查找和恢复.

像svn, keepbit, ….cvs 都是版本控制器.

git, 开源项目, php项目, web项目 主要使用的版本控制器, 包括php代码也是托管在git上(github)

linux内核, git的作者是 linus同一个人.

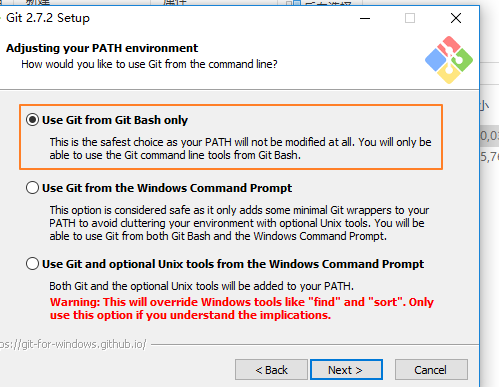
基于git搭建的github, 免费的代码托管网站.

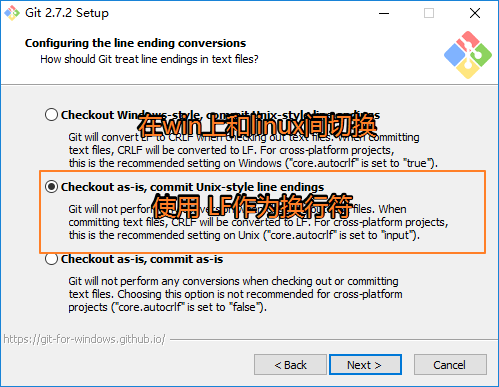


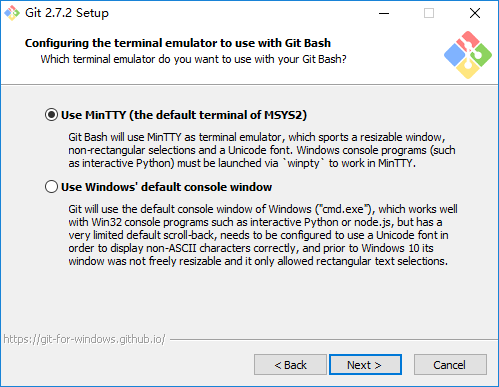
# 安装

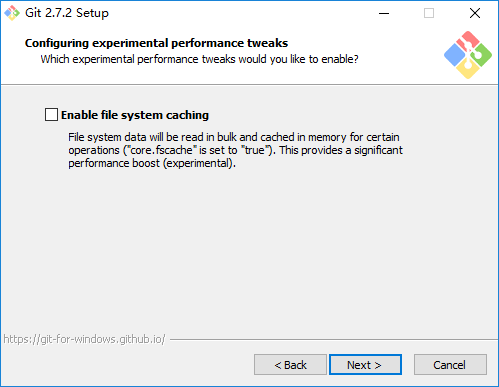


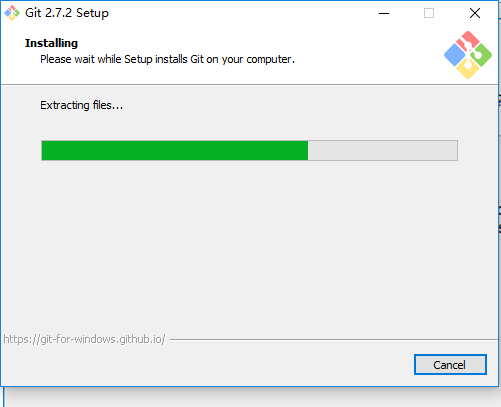
## windows

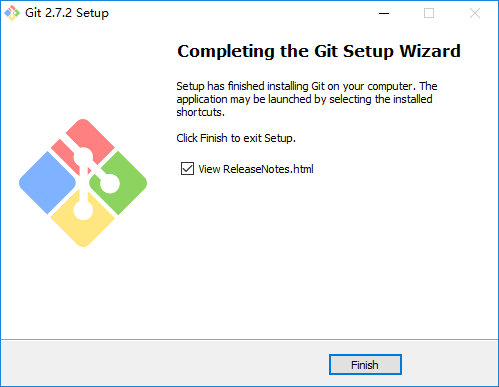


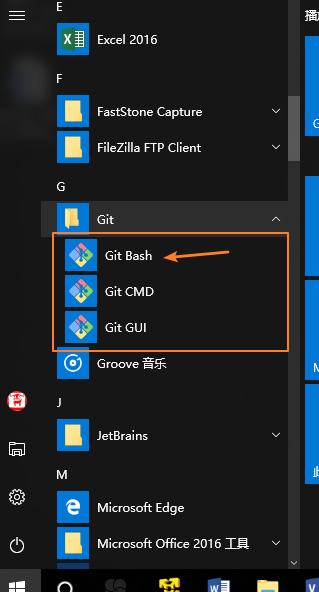




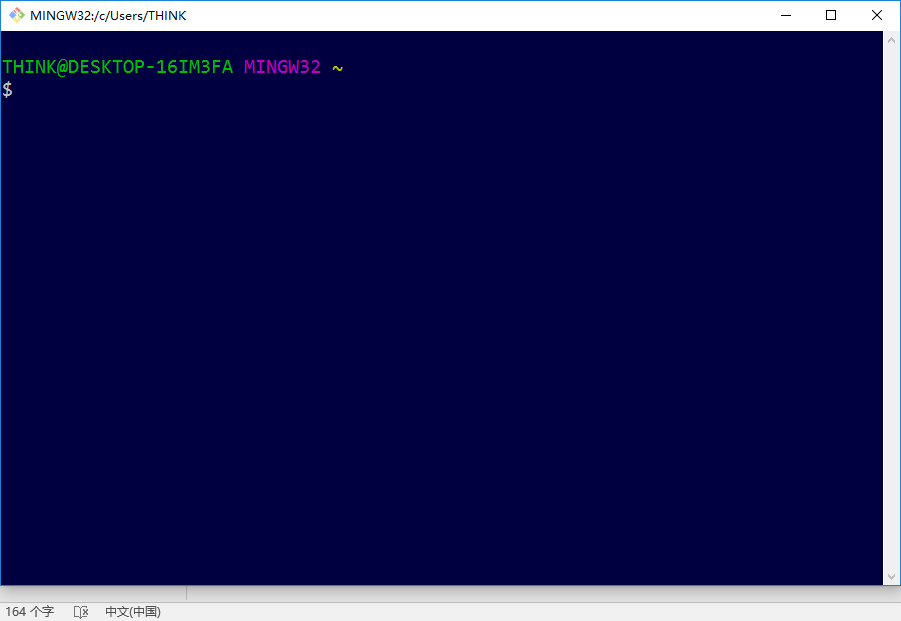




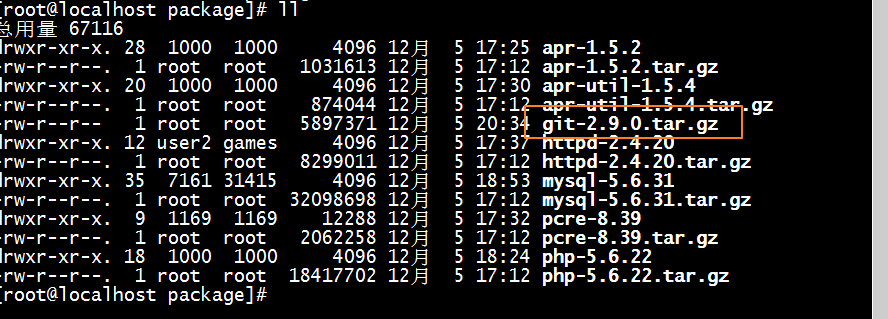




运行 git bash

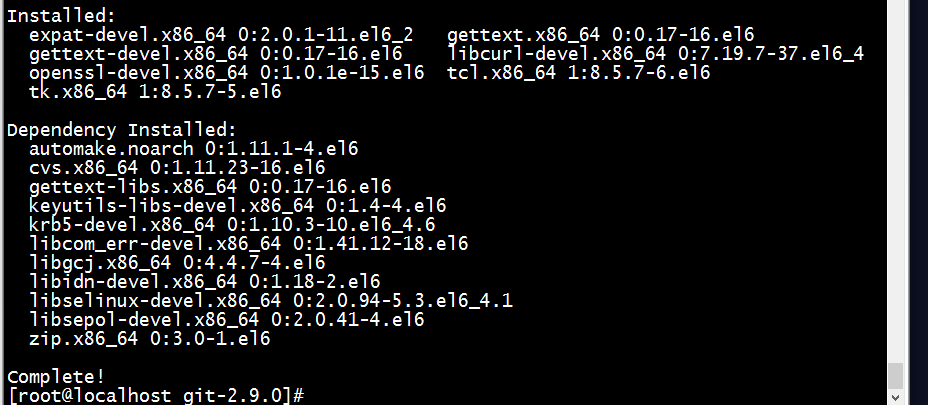


## linux



### 基础依赖

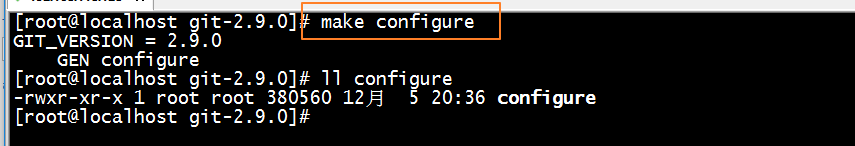
yum --disablerepo=\\* --enablerepo=c6-media install gcc gcc-c++ perl perl-devel curl-devel expat-devel gettext-devel openssl-devel perl-devel zlib-devel tcl build-essential tk gettext



### 解压, 进入

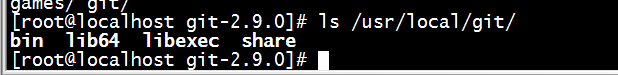


### 编译生成configure, make configure

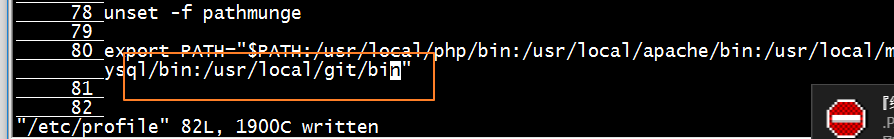


### 配置 编译 安装

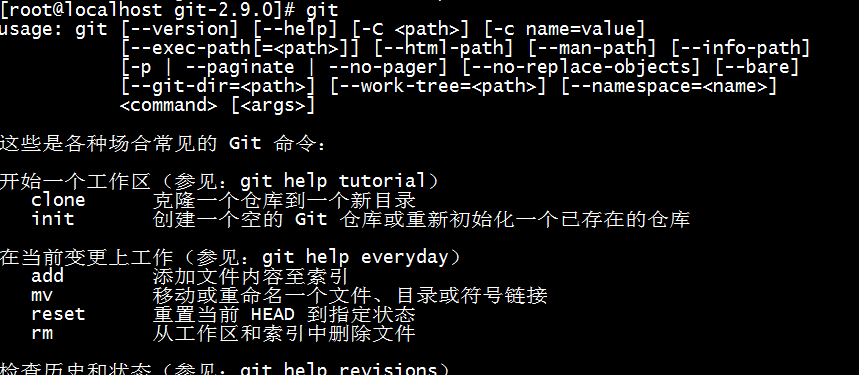
./configure --prefix=/usr/local/git && make && make install



### 配置环境变量



执行

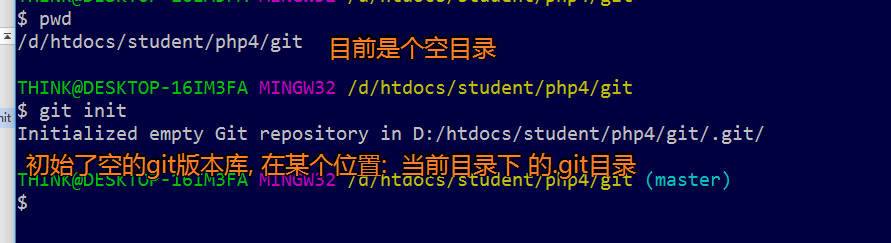


# 快速使用

## 初始化, git init

在某个项目目录执行git init, 将其初始成一个git管理的项目目录

初始化空目录:



初始非空目录



初始化后, 在当前目录下, 创建一个版本库目录, .git, 初始化后该版本库目录是空的:

强调:

项目目录不是版本库目录, 而项目目录中.git才是版本库目录.

## 正常编写代码

不用考虑 git 的问题, 常规敲代码即可

任何编辑器, 任何方式,(拷贝,粘贴, 剪切, 删除)都可以.



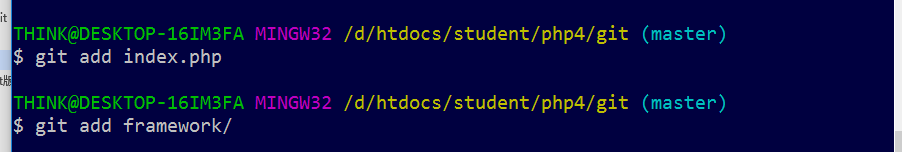
当代码告一段落.

## 添加, git add

git add .

将当前目录 . 中, 所有的修改都添加.

(PS, 如果有不需要添加到版本库中的代码, 需要一个个文件的处理.



)

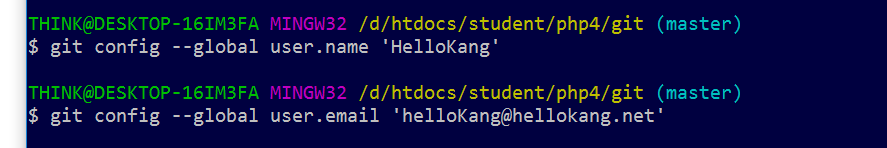
## 设置作者信息

第一次提交时, 要设置自己的用户信息.

设置作者信息, 包含, 姓名和邮箱信息.

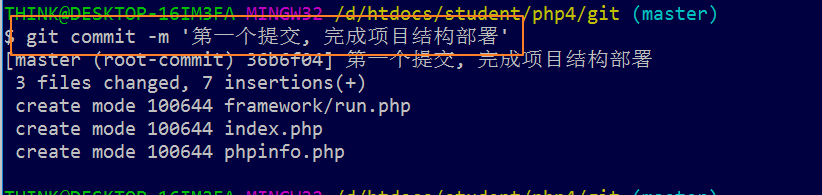
git config –global 配置项 值

进行设置,演示如下:



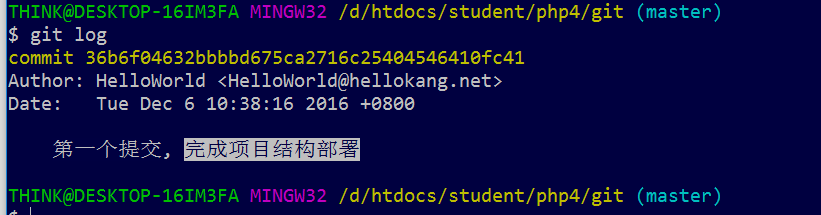
## 提交版本到git版本控制器, git commit

git commit -m “版本说明描述信息”



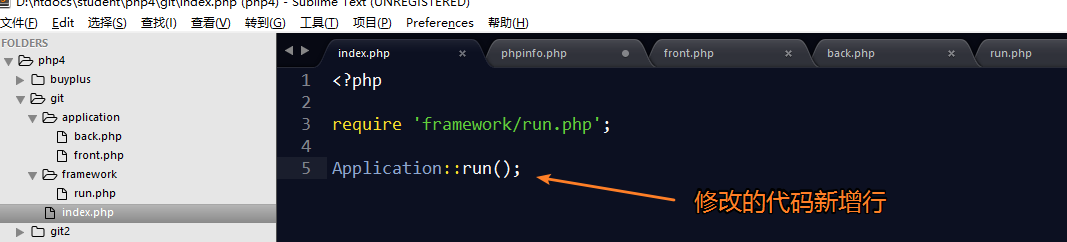
提交成功的代码, 永远也不会丢失!

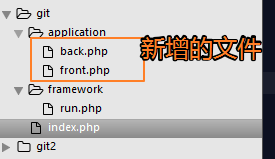
## git log, 查看提交日志



## 再次编辑, 提交新版本

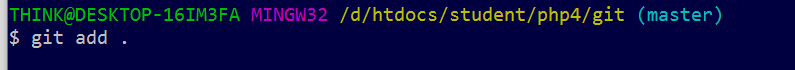
添加新文件, 删除旧文件, 修改文件



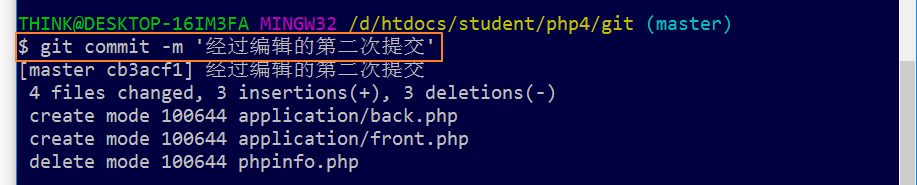


删除了phpinfo.php

添加

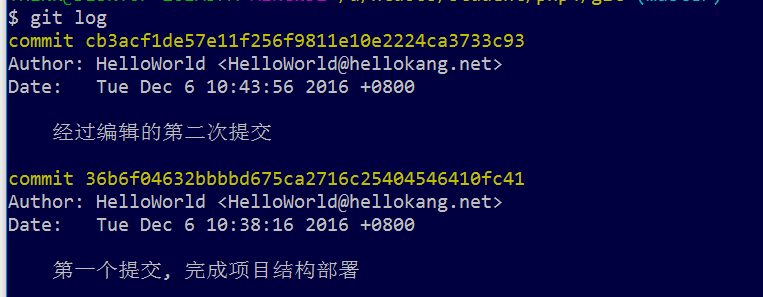


提交



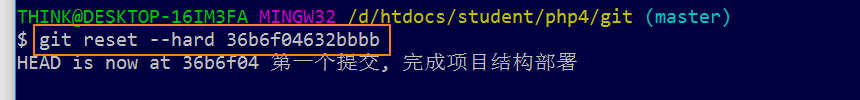
(一旦提交, 所有的修改都会被记录)

目前 版本库中存在2个版本



## 版本间的切换

git reset –hard 版本提交号



目前的项目文件, 回归到, 版本1的状态:

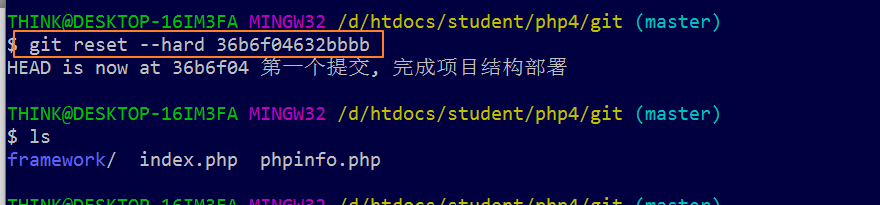
删除的文件被找回

添加的文件被删除

修改的内容被撤销

甚至再次回到之后的版本(将来的版本)

提供版本号即可



# 基本概念

如图所示:



## 工作区

项目目录就是工作区.

coder在工作区, 完成代码的编写.

## 暂存区

准备提交的修改, 临时存储的位置, 称之为暂存区.

如果修改要被提交到版本库, 需要现在暂存区统一设置管理等待.

## 版本库区

已经提交的修改, 存储在git的版本库区.

## 相应文件状态

未修改

已修改

未暂存

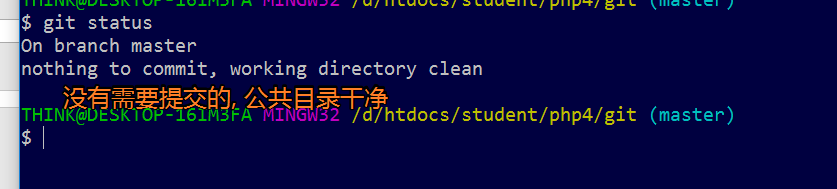
已暂存

未提交

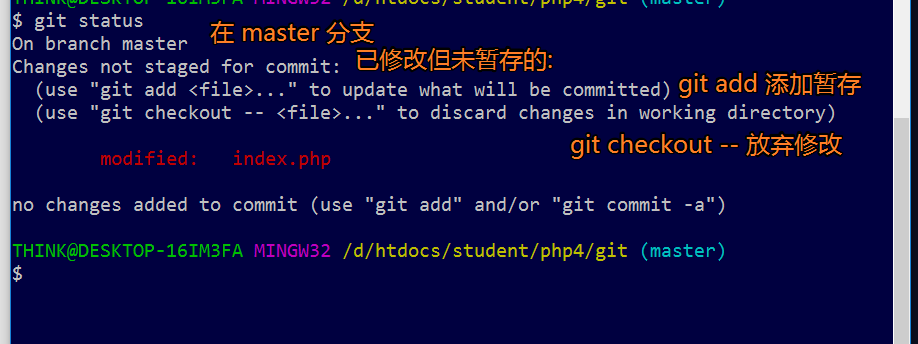
已提交

## git status 查看状态

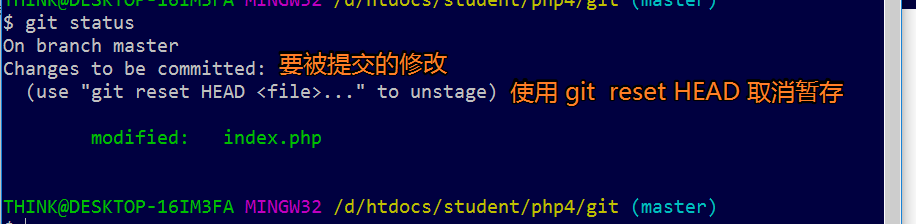
在没有任何的修改情况下



修改了但是未添加到暂存:



添加到暂存区

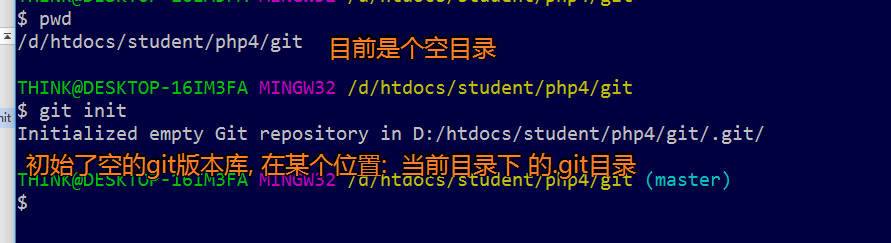


# 初始化 git init

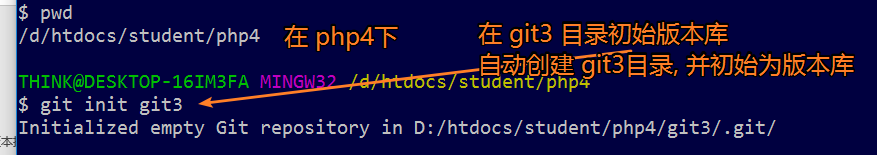
在某个目录初始为一个git管理的版本库

支持两种语法:

之一: 在工作目录下执行



之二: 指定目录去创建



# 查看状态, git status

查看文件, 提交, 暂存的状态

在做操作时, 保证操作的正确性. 而频繁执行 git status!

显示, 状态和操作提示, 快速完成处理



# 添加到暂存, git add

将修改或者新增的文件, 添加到暂存区

为了可以提交到版本库

典型的是 git add .

将当前目录全部内容 都一次性提交.

如果需要逐一提交,请使用 git add 文件|目录

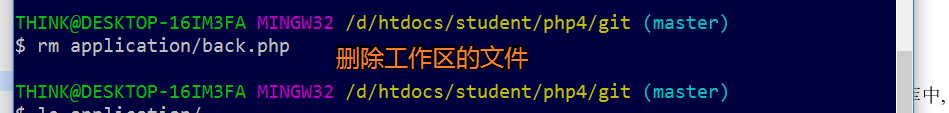
分文件和目录进行提交

# 删除 git rm

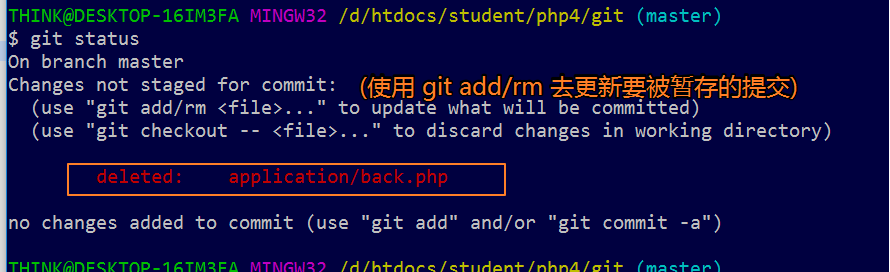
当删除了一个工作区的文件时, 需要使用git rm指令暂存此次删除. 否则, 版本库中, 已经存储的该文件是不会被删除的.

演示

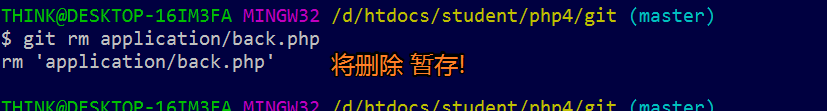
删除工作区的文件:



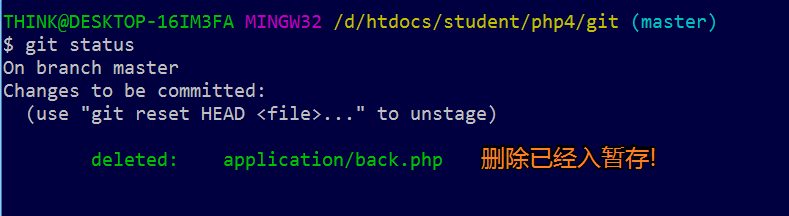
查看状态



git rm 将删除暂存



查看状态



本地删除后一定要git rm 将删除也暂存, 后续会提交!

(git add . 可以自动暂存删除的)

# 提交到版本库 git commit

将暂存的修改提交到版本库, 形成一个新的版本.

提交时, 要求设置版本的描述信息.

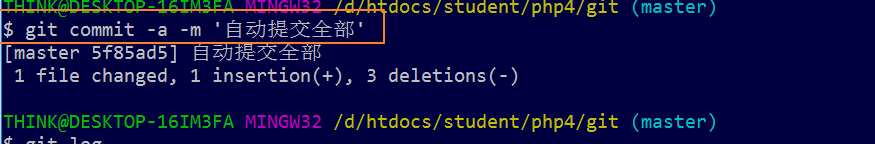
设置的方式有两种:

之一: -m 选项, 提供描述信息即可

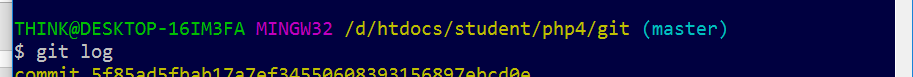
之二: 不携带-m选项, 会默认触发 vi编辑器, 编辑描述, 保存退出即可提交!~

操作时, 经常的使用 git add ., git commit –m使用,

可以使用 git commit -a选项简化



# 日志, git log



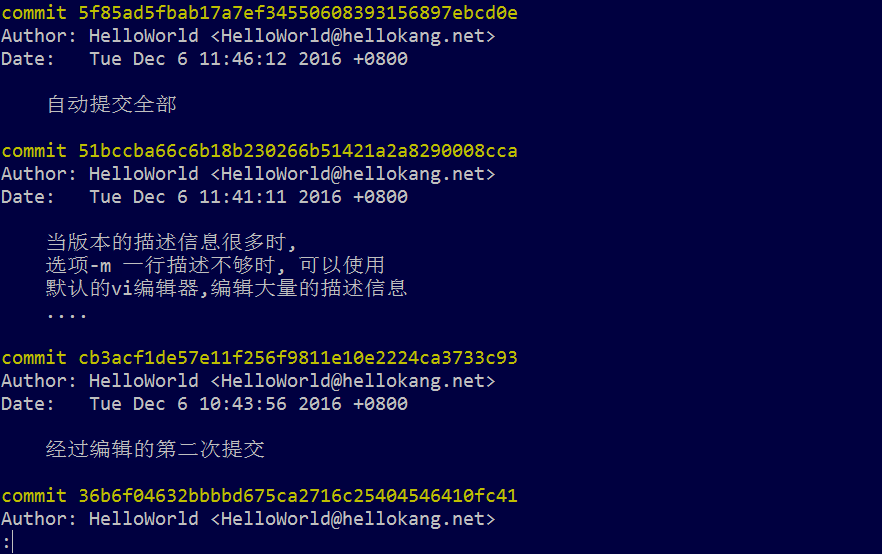
## 翻页查看

查看时, 由于日志会很多, 默认以分页方式查看. less进行分页查看.

less的使用方式与more类似

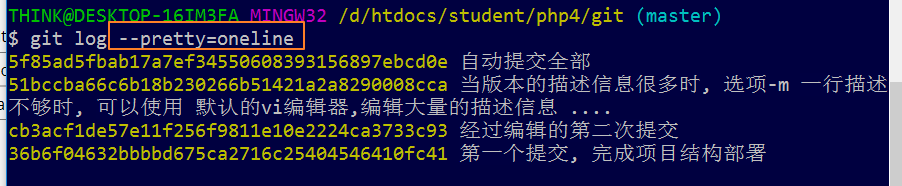
空格, ctrl+b 下上翻页

q退出

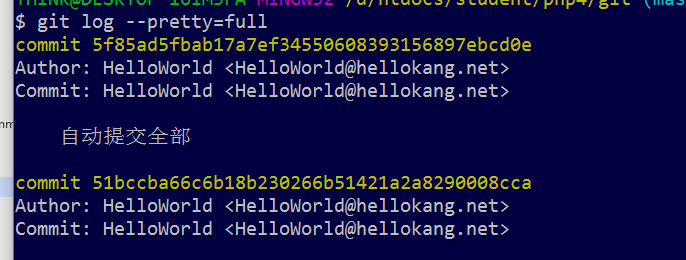


## 格式化

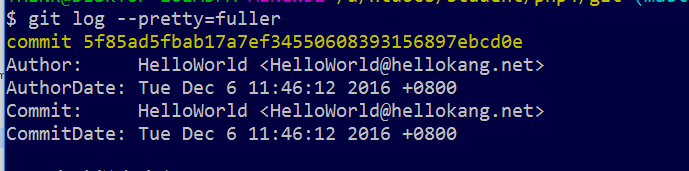
一行简要信息:



full, 全部基本信息(默认的形式)

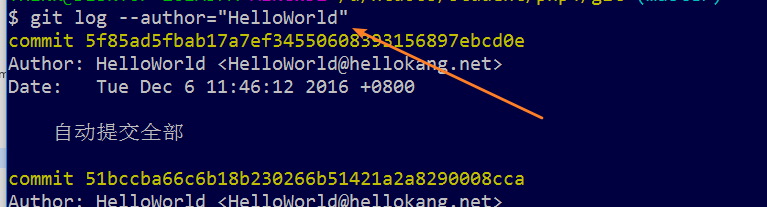


fuller, 更详细的信息



## 支持检索

支持作者 –author



支持时间 –after –before

after, 在某个时间点后

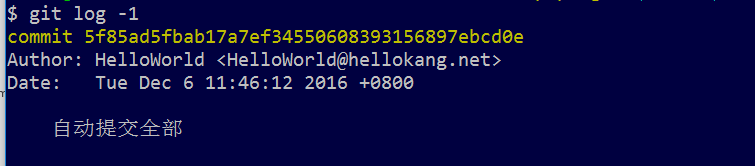
before 在某个时间点前





## 显示最近提交

快捷显示最近日志 –N表示最近N个提交日志



# 重置版本, git reset

## git reset --hard, 硬迁移(重置)

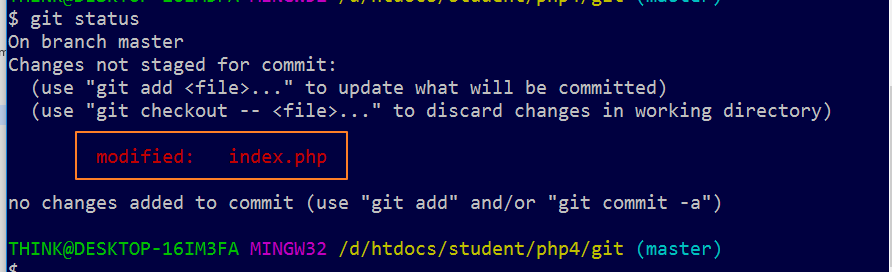
将版本, 强行迁移到, 固定的版本

需要知道目标的版本, 通常使用日志获取

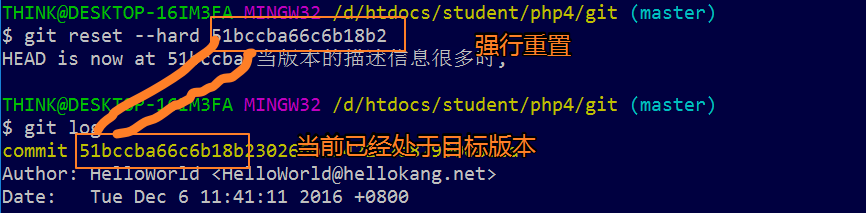
执行强制迁移, 工作区与暂存区, 都被重置, 当前的的修改是不会被记录下来. 无论是已暂存还是未暂存.

案例一:

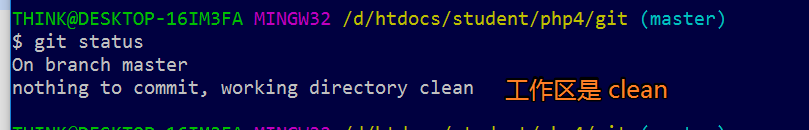
当前的状态, 工作区有修改的内容, 还未暂存



强行重置到某个版本 –hard



注意重置后的状态



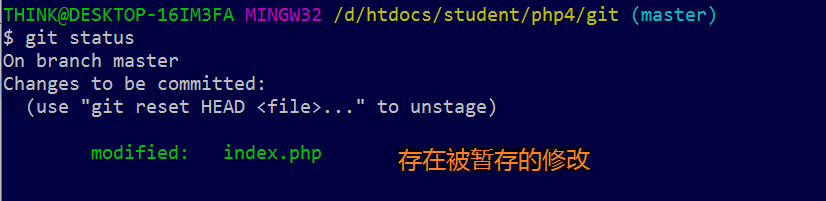
修改的index.php内容不在了.



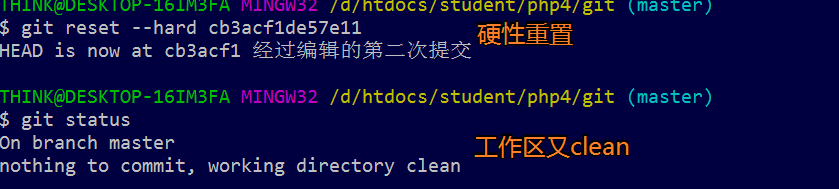
案例二:

再来举例子:

当前状态, 存在被暂存的修改:



执行--hard重置



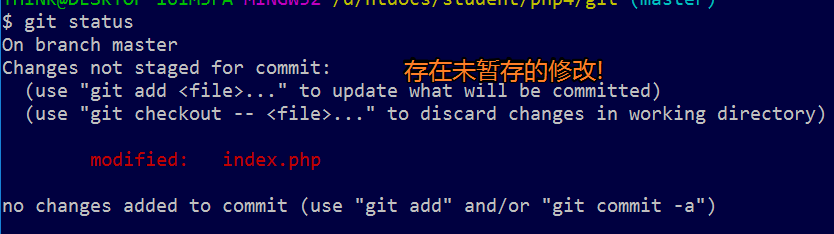
## git reset –soft, 软迁移(重置)

--soft修改, 会保留当前的工作区

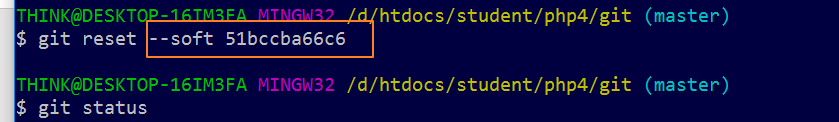
同时, 将版本间的差异, 记录下来, 保存到暂存区:

你做的当前版本的任何修改, 都会背保留. 你选择性的继续保留还是舍弃:

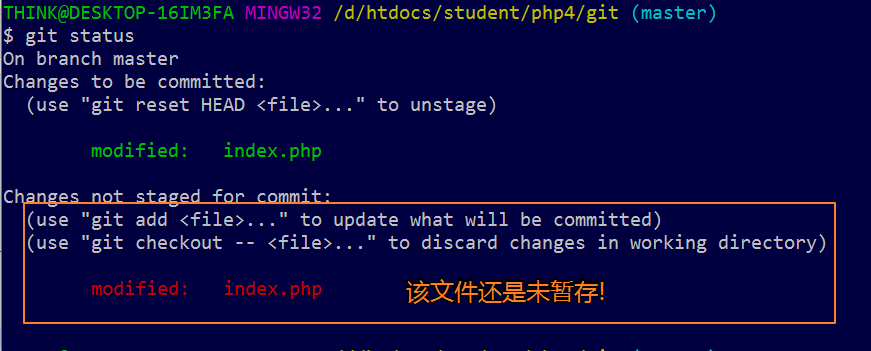
案例一: 存在未暂存的修改



做版本的迁移(重置)

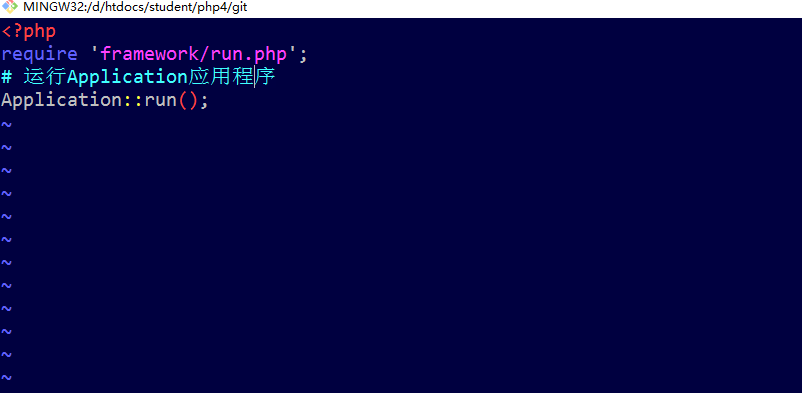


此时, 通过状态查看, 之前未暂存的修改还处在未暂存中



案例二: 两个版本间文件有修改, 重置到前面的版本

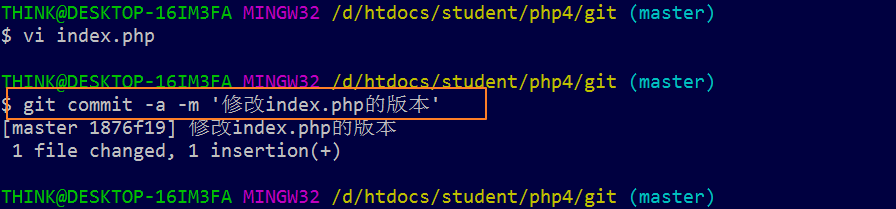
版本A的index.php



修改index.php

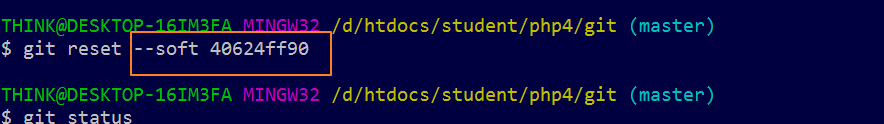


提交带有修改的新版本

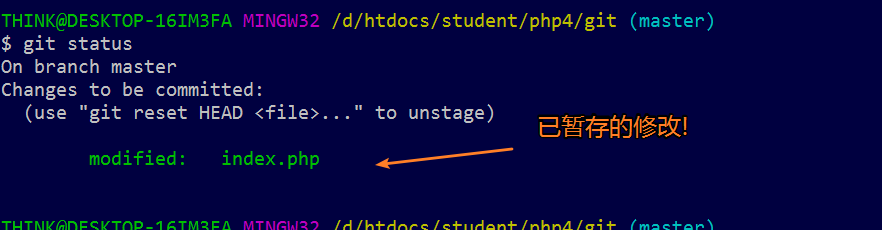


此时, index.php 在最近的2个版本被修改了.

重置版本 –soft重置

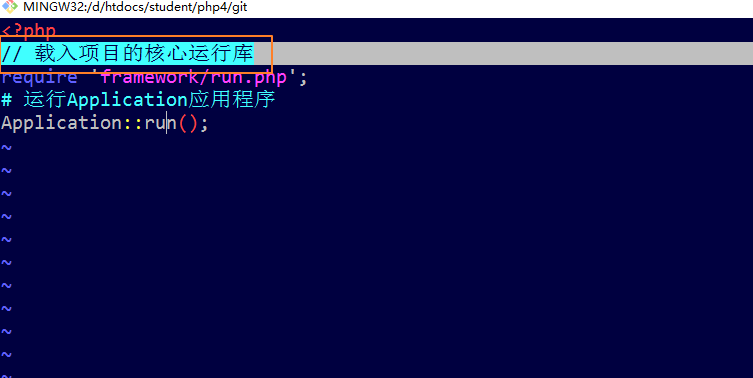


查看状态:



查看内容: 发现, index.php的修改, 被保留. 但是出现在暂存区.

记录下来, 存储在暂存区.



提示, 如果你希望某个文件例如index.php, 应该重置到目标版本中:

1, 撤销暂存, 2撤销修改

git reset HEAD index.php, 撤销暂存

git checkout – index.php, 撤销修改

文件就回归到目标版本一致的内容了.

## 版本号的获取

### git log 可以获取版本号

一旦重置到之前的版本中, git log是看不到将来的版本号的.

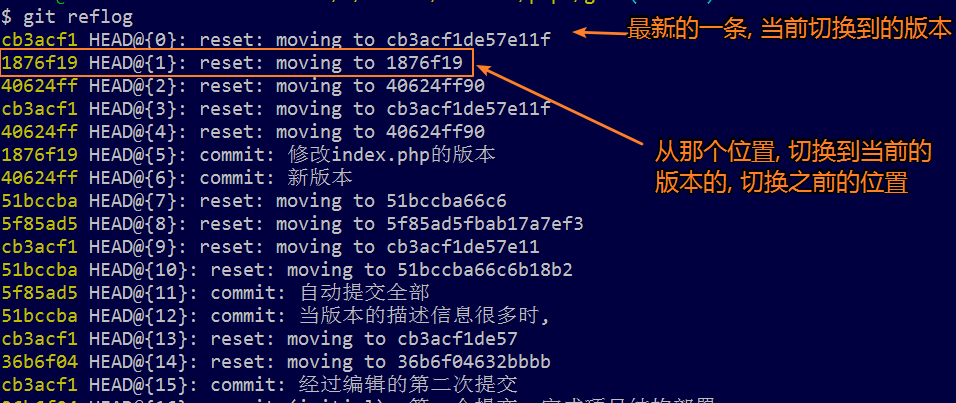
已经是版本10, 重置到了版本7, 再执行git log, 看不到 8, 9, 10三个版本log, 版本号.

需要使用: git reflog

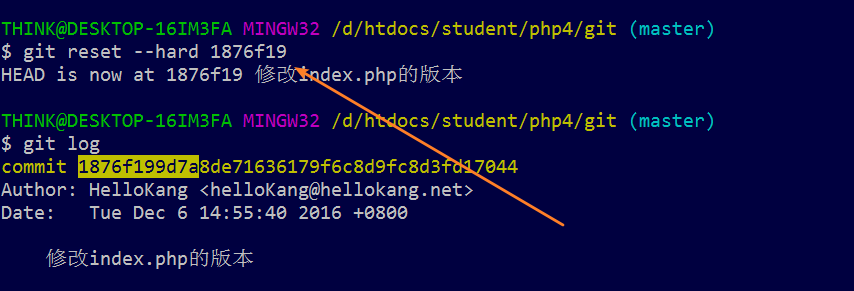
### git reflog, 重置操作 日志

获取版本重置操作日志.

重置之前, 是处在那个版本中



回到, 当前版本切换之前的某个版本上:



### HEAD指针, 相对

当需要切换到上一个(上N个)版本时, 需要一个相对的定位.

可以使用:

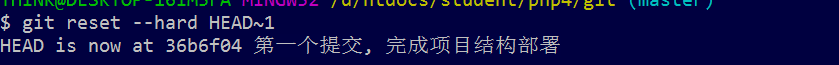
HEAD^, 上一个

HEAD^^, 上2个

HEAD^^^^^^^^^^^^^^^^^, 上N版本

HEAD~23, 上23个版本





HEAD永远指向正在编辑的版本:



重置, 仅仅是修改了HEAD指针指向的版本, 而不是将代码回退! 版本还是被记录了.

# 撤销修改

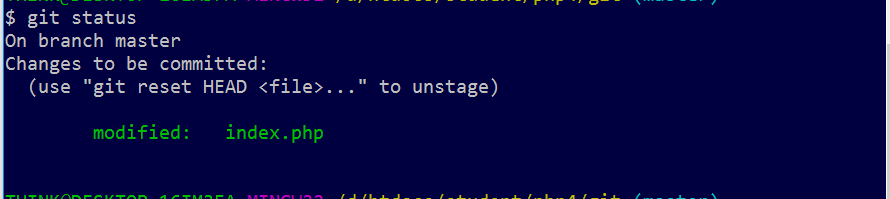
## 撤销暂存, git reset HEAD file

如果, 一个文件修改并被暂存了, 但是本次不希望提交.

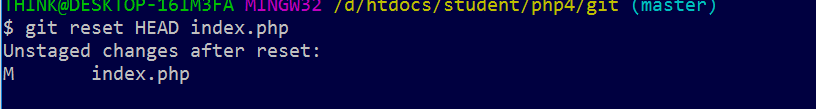
可以撤销暂存.

修改还在, 处于为暂存状态!

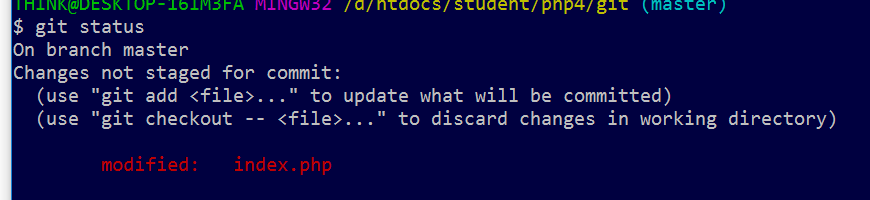
当前的状态, 文件处于暂存:



撤销暂存

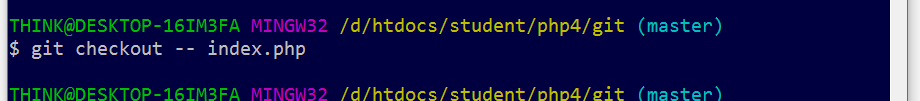


文件修改还在, 仅仅是没有被暂存



## 撤销修改, git checkout -- file

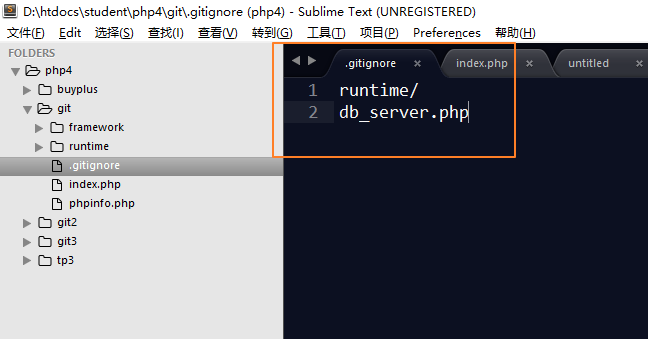
当前修改的内容, 被撤销 ,文件回到版本的初始位置



# 忽略提交 .gitignore

项目中 往往会出现, 不需要提交到版本库的代码. 例如

本地配置文件, 缓存文件, 运行时生成的文件



git 就不去监视 符合规则的文件的修改:

建议第一个次提交前, 就将忽略设置好.

# 多人协作

多个用户, 同时维护一个核心的版本库

## 典型的模式



要点:

之一: 本地也是版本库. 大多数的操作, 在本地执行即可

之二: 本地处理完毕, 才需要与远程的公共版本库通讯, 推送和拉取

之三: 远程公共版本库, 不需要工作区的概念, 不需要在其上编辑代码. 所有的操作都是本地完成

总结: 多人协作环境, 仅仅需要.

1, 搭建公共版本库(不需要工作区, 裸版本库)

2, 本地版本库, 克隆自公共版本库. clone

3, 本地编辑, 提交版本

4, 拉取

5, 推送

## 搭建公共裸版本库

裸版本库, 公共版本库, 不需要工作区!

就是在远程公共服务器上, 初始化(git init)一个版本库即可.

--bare 选项, 就是裸库!

假设, linux就是公共的服务器

设计版本库的位置

/srv/gitRepo



## 本地克隆自公共版本库

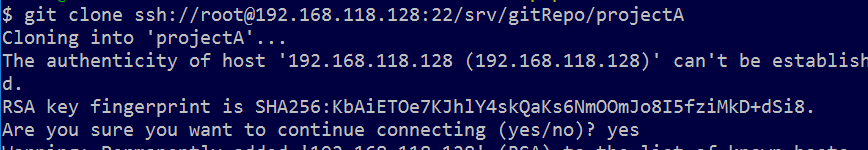
git clone 版本库地址

版本库地址, 支持ssh协议.

ssh://[user@]host.xz[:port]/path/to/repo.git/

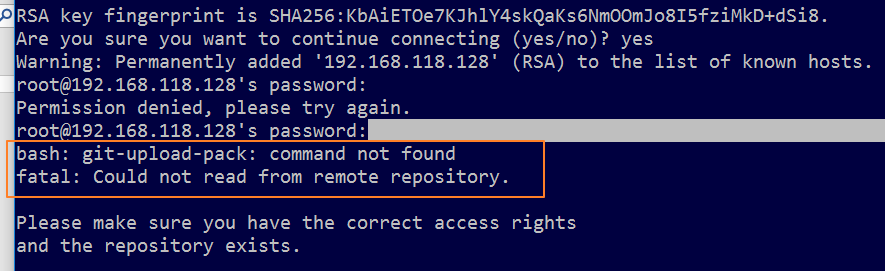
win就是本地

执行的命令如下:

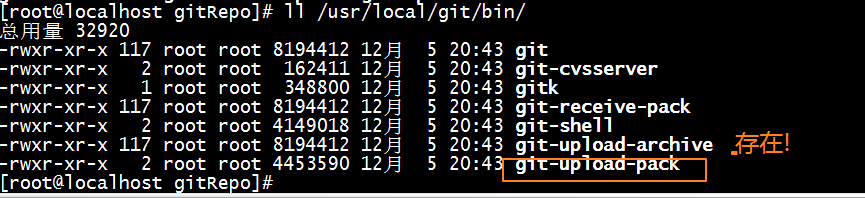


出现一个错误:

git-upload-pack



git-upload-pack, 是一个git的上传工具:



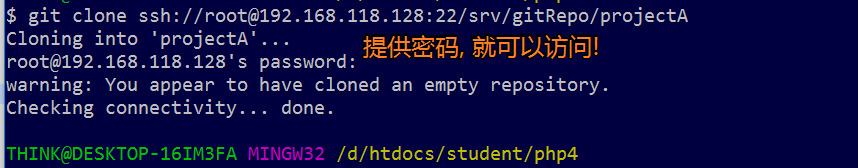


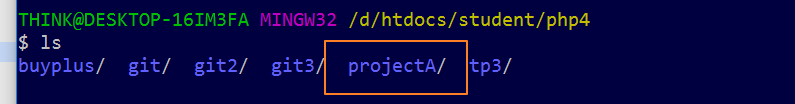
clone时, 没有识别 git/bin为PATH环境变量内容.

将命令, 加入到系统的bin目录即可:



继续 clone:





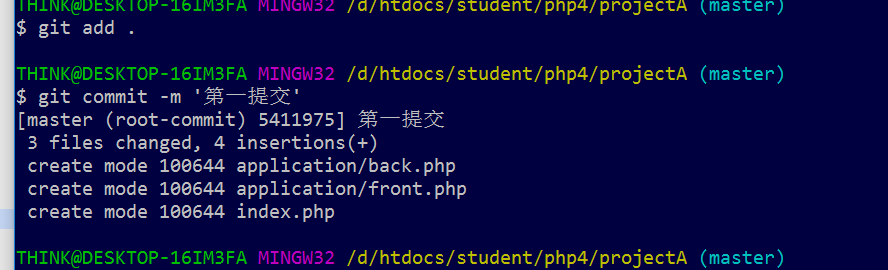
此时, win就是本地版本库. 而linux上, 就是公共, 对于本地来说,就是远程版本库

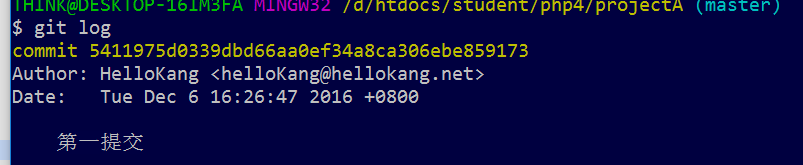
## 本地编辑

本地版本库, 常规编辑, 提交等操作即可

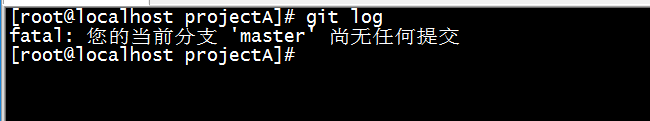
与远程毫无关系.

本地:





远程查看

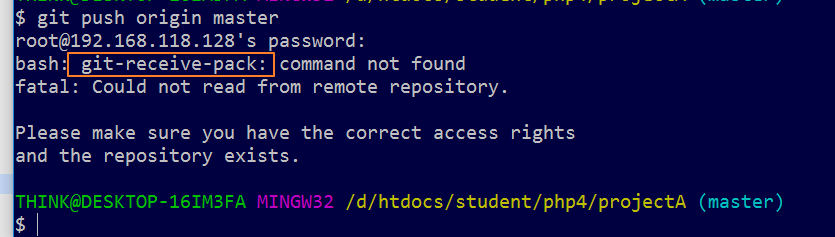


## 推送到远程

git push origin master

将master分支推送到来源origin

推送时 出现问题:



bash: git-receive-pack: command not found

类似的问题:

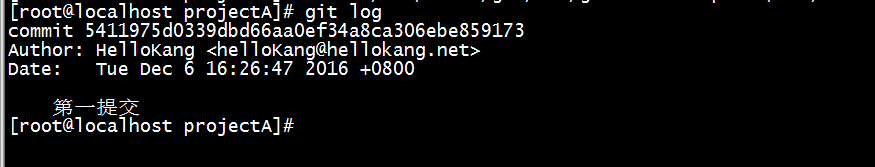
linux:



重新推送:

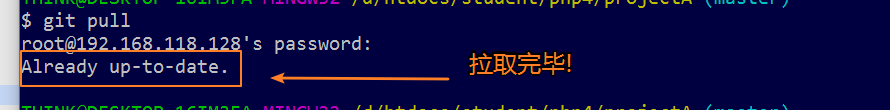


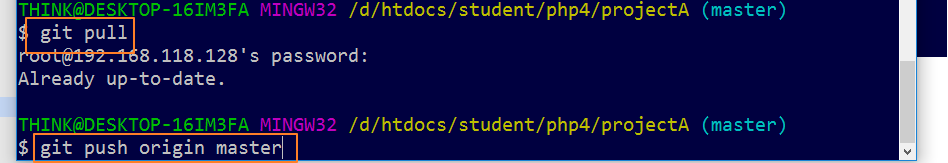
查看远程服务器状态



推送前, 通常需要执行先拉取. 可能出现的情况, 在你本地编辑的同时, 其他用户将修改推送到了, 远程公共版本库.

强调: push前, 一定要先pull拉取





## 多人协作 冲突解决

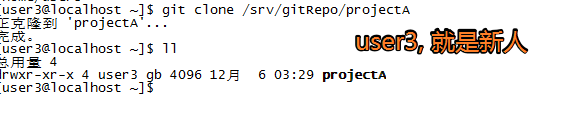
项目加入了新人,

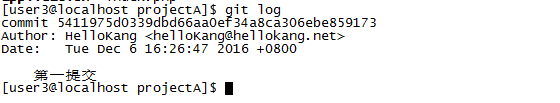
新人, 重新克隆自远程版本库.

假设新人在linux下工作:

克隆当前的版本库, 自远程版本库

在同一台电脑上, 直接使用 地址即可:



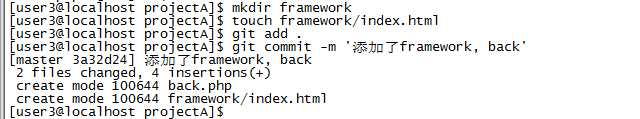


设计user3的信息



user3 开发项目, 本地开发, 本地提交

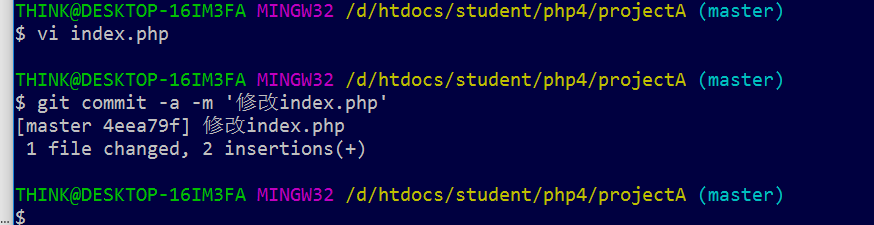
### user3开发的文件与hellokang不冲突



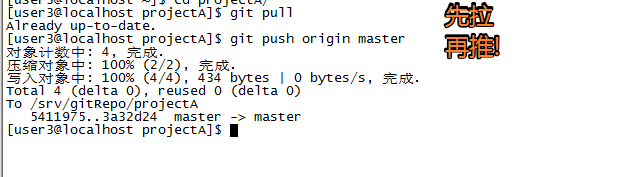


### 同时hellokang继续开发

本地开发



### user1, 推送到远程



推送时, 要保证用户对版本库目录,拥有写权限:

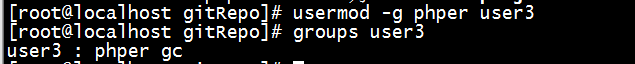
将版本库的所属组改为phper



增加版本库目录的所属组用户的写权限



将user3的主组, 设置为phper



### HelloKang, 推送修改到远程

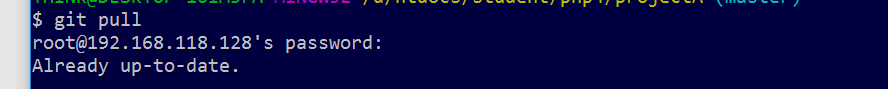
先拉取, 拉取到内容, 并自动合并成功:



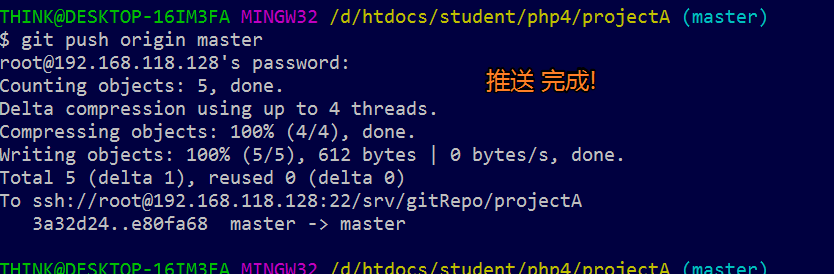
查看多了一个版本



没有在需要拉取的了



提交本地到远程

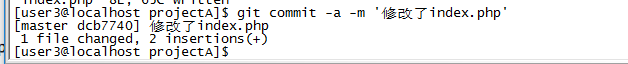


### user3 与 hellokang 开发的内容冲突

user3, 对index.php的修改



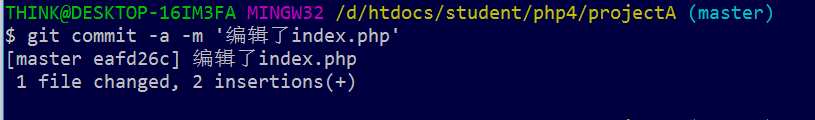
user3, 自己提交本地



### hellokang也修改index.php

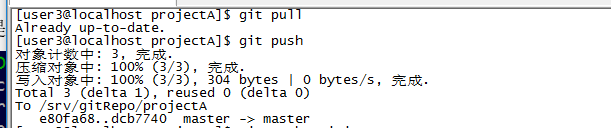


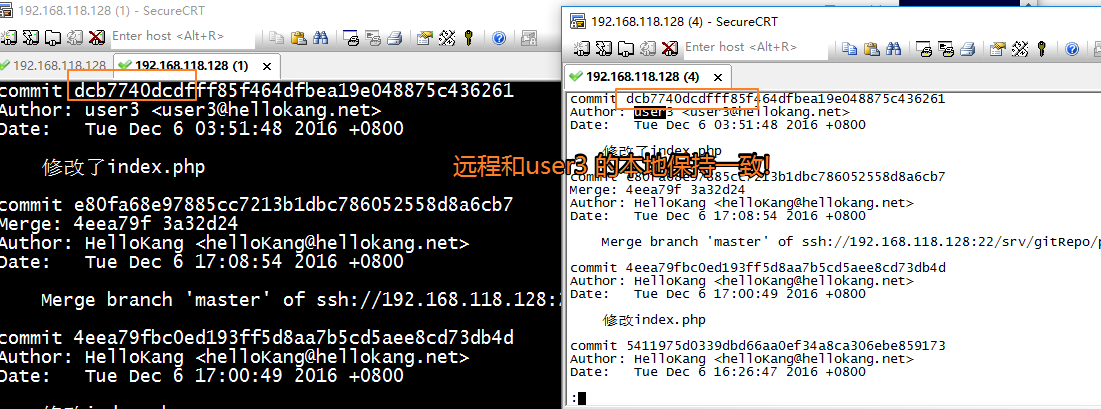
自己去提交到本地



### user3先将修改推送

git push origin master (默认值)





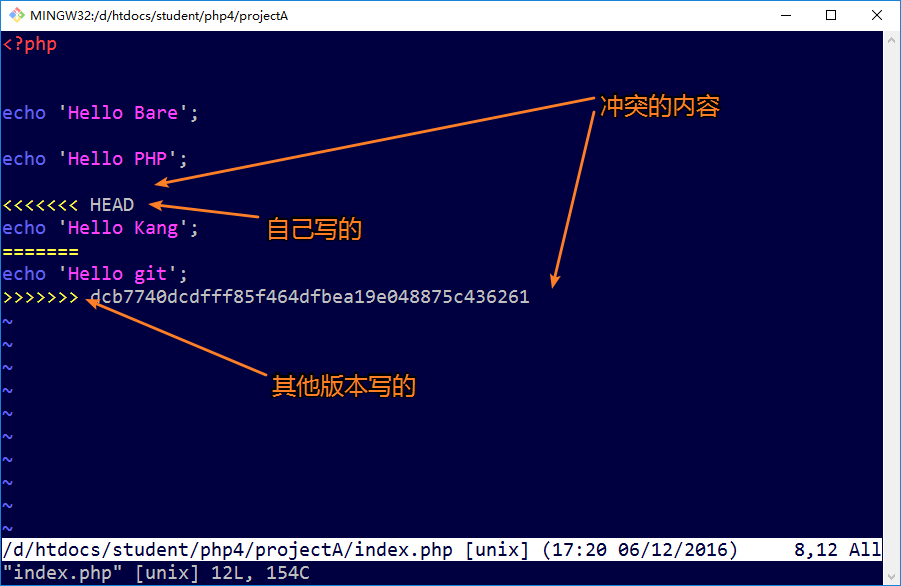
### hellokang, 将修改推送

在拉取时, git pull, 需要合并远程与本地代码, 由于编辑了一个文件, 到时自动合并失败:

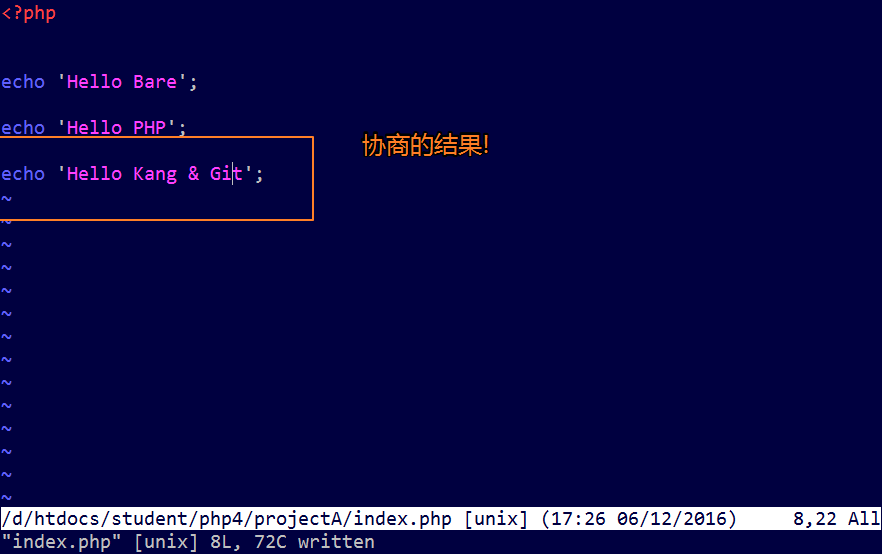


### 手动解决冲突

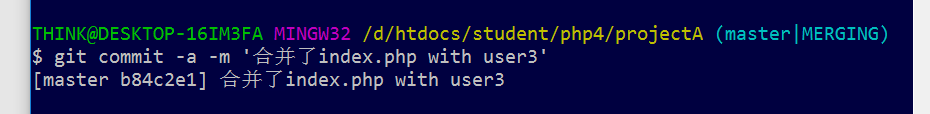
手动编辑冲突的文件



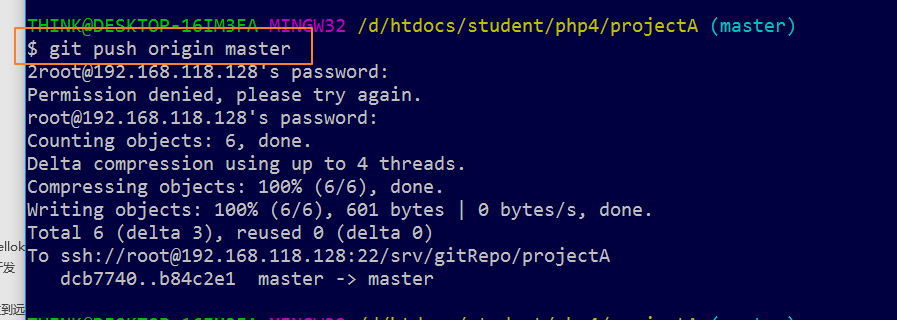
两个作者要协商解决:



相当于又做了修改, 需要重写提交



推送到远程:



# 分支, branch

master, 就是默认的分支, 称之为主分支!

当版本库创建好之后, 默认处在master分支上!

允许增加额外的分支的!

项目至少有2到三个分支:

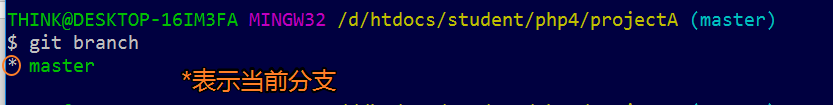
1: master, 主分支, 用于发布的代码.

2: dev, 开发分支, 当前处在开发阶段的分支. 应该主要的工作分支.

3: bug, 解决临时出现的问题.



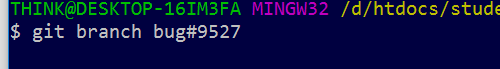
## 查看分支git branch

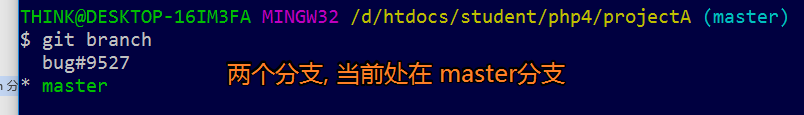


分支列表, \* 表示当前的分支!

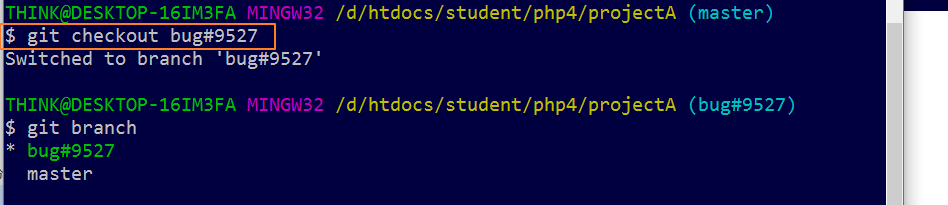
## 添加分支, git branch 分支名

bug#9527 分支名





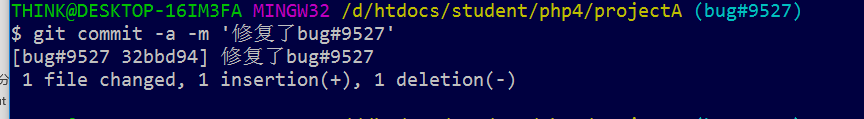
## 切换分支, git checkout 分支名



## 继续在分支上开发



## 分支上提交



## 合并分支

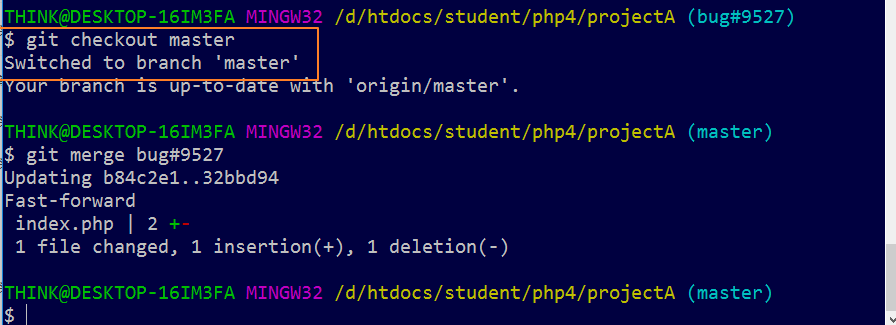
切换会, 合并的目标分支

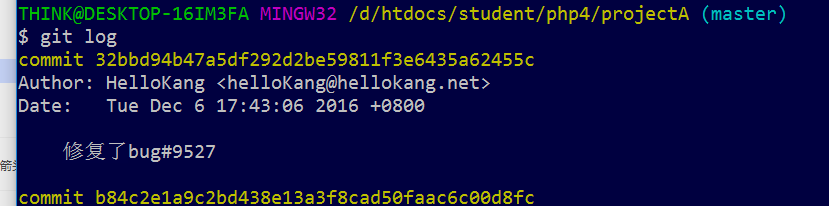
将某个分支, 合并到当前分支.

例如, 合并 bug#分支 到 master

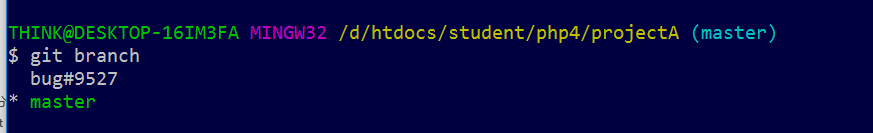
切换到master

使用 git merge 分支名





合并完毕后, 分支没有消失的



## 删除分支 git branch –d 分支

