

# 版本控制

---

版本控制是指对软件开发过程中各种程序代码、配置文件及说明文档等文件变更的管理，是软件配置管理的核心思想之一。

## 什么是版本控制

版本控制 (Revision control) 是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

- 实现跨区域多人协同开发
- 追踪和记载一个或者多个文件的历史记录
- 组织和保护你的源代码和文档
- 统计工作量
- 并行开发、提高开发效率
- 跟踪记录整个软件的开发过程
- 减轻开发人员的负担，节省时间，同时降低人为错误

简单说就是用于管理多人协同开发项目的技术。

没有进行版本控制或者版本控制本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并发性、软件源代码的安全性，以及软件的整合等问题。

## 聊聊Git的历史

同生活中的许多伟大事物一样，Git 诞生于一个极富纷争大举创新的年代。

Linux 内核开源项目有着为数众多的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上(1991 - 2002年间)。到 2002 年，整个项目组开始启用一个专有的分布式版本控制系统 BitKeeper 来管理和维护代码。

Linux社区中存在很多的大佬！破解研究 BitKeeper ！

到了 2005 年，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了 Linux 内核社区免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区(特别是 Linux 的缔造者 Linus Torvalds)基于使用 BitKeeper 时的经验教训，开发出自己的版本系统。（2 周左右！）也就是后来的 Git！

**Git是目前世界上最先进的分布式版本控制系统。**

Git是免费、开源的，最初Git是为辅助 Linux 内核开发的，来替代 BitKeeper！

## Git的安装

---

官网：<https://git-scm.com/>

About

Documentation

Downloads


GUI Clients

Logos

Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

# Downloading Git



**Your download is starting...**

You are downloading the latest (**2.28.0**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **about 2 months ago**, on 2020-07-28.

[Click here to download manually](#), if your download hasn't started.

### Other Git for Windows downloads

**Git for Windows Setup**  
**32-bit Git for Windows Setup.**  
**64-bit Git for Windows Setup.**

**Git for Windows Portable ("thumbdrive edition")**  
**32-bit Git for Windows Portable.**  
**64-bit Git for Windows Portable.**

The current source code release is version 2.28.0. If you want the newer version, you can build it from [the source code](#).

官网速度慢可以使用淘宝镜像：<http://npm.taobao.org/mirrors/git-for-windows>

Mirror index of <https://github.com/git-for-windows/git/releases/>

../		
2.10.0.windows.1/	2016-09-03T08:24:42Z	-
2.10.1.windows.1/	2016-10-04T16:13:02Z	-
2.10.1.windows.2/	2016-10-13T18:24:05Z	-
2.10.2.windows.1/	2016-11-02T20:43:32Z	-
2.11.0-rc0.windows.1/	2016-11-04T19:05:05Z	-
2.11.0-rc0.windows.2/	2016-11-11T18:32:38Z	-
2.11.0-rc1.windows.1/	2016-11-15T16:55:46Z	-
2.11.0-rc2.windows.1/	2016-11-19T10:49:16Z	-
2.11.0-rc3.windows.1/	2016-11-24T23:52:36Z	-
2.11.0.windows.1/	2016-12-01T12:31:04Z	-
2.11.0.windows.2/	2017-01-13T15:42:38Z	-
2.11.0.windows.3/	2017-01-14T12:09:10Z	-
2.11.1.mingit-prerelease.1/	2017-01-04T17:39:02Z	-
2.11.1.windows-prerelease.1/	2016-12-23T16:21:33Z	-
2.11.1.windows-prerelease.2/	2017-01-20T16:31:36Z	-
2.11.1.windows.1/	2017-02-03T09:18:55Z	-
2.12.0.windows.1/	2017-02-25T16:55:20Z	-
2.12.0.windows.2/	2017-03-15T18:30:19Z	-
2.12.1.windows.1/	2017-03-21T16:06:44Z	-
2.12.2.windows.1/	2017-03-27T19:47:36Z	-
2.12.2.windows.2/	2017-04-05T16:08:51Z	-
2.12.2.windows.3/	2017-08-10T18:39:17Z	-
2.13.0.windows.1/	2017-05-10T08:55:53Z	-
2.13.1.windows.1/	2017-06-13T12:12:11Z	-
2.13.1.windows.2/	2017-06-15T17:01:25Z	-
2.13.1.windows.3/	2017-08-10T18:39:43Z	-
2.13.2.windows.1/	2017-06-26T14:00:55Z	-
2.13.3.windows.1/	2017-07-13T12:46:46Z	-

安装成功，目录显示Git相关的应用程序

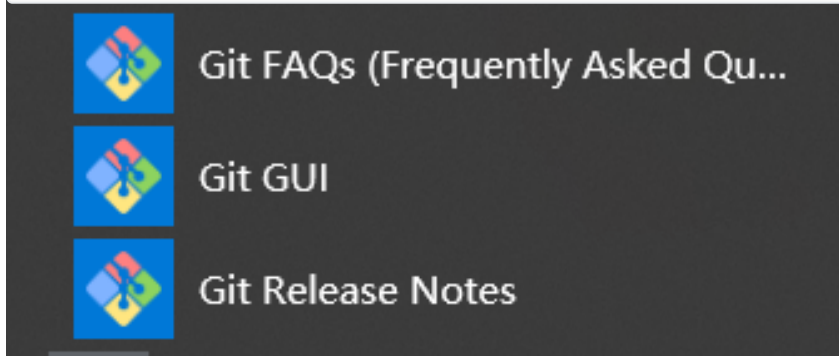
## Git相关配置

Git系统配置

```
# 查看配置
git config -l

# 查看系统配置，除去自定义配置
git config --system -l

# 查看用户自定义的配置
git config --global -l
```



Git相关配置文件：

1) 、Git\etc\gitconfig : Git  
安装目录下的gitconfig --  
system 系统级

2) 、  
C:\Users\TRacy\.gitconfig :  
只使用于当前登录的用户的基本配置 --global 全局

也可以直接编辑文件，通过命令设置会响应到这里。

#### Git用户配置

#### 设置用户名和邮箱（必要）

```
# 设置用户名
git config --global user.name "username"

# 设置邮箱
git config --global user.email "youremail"
```

## Git基本原理

#### 工作区域

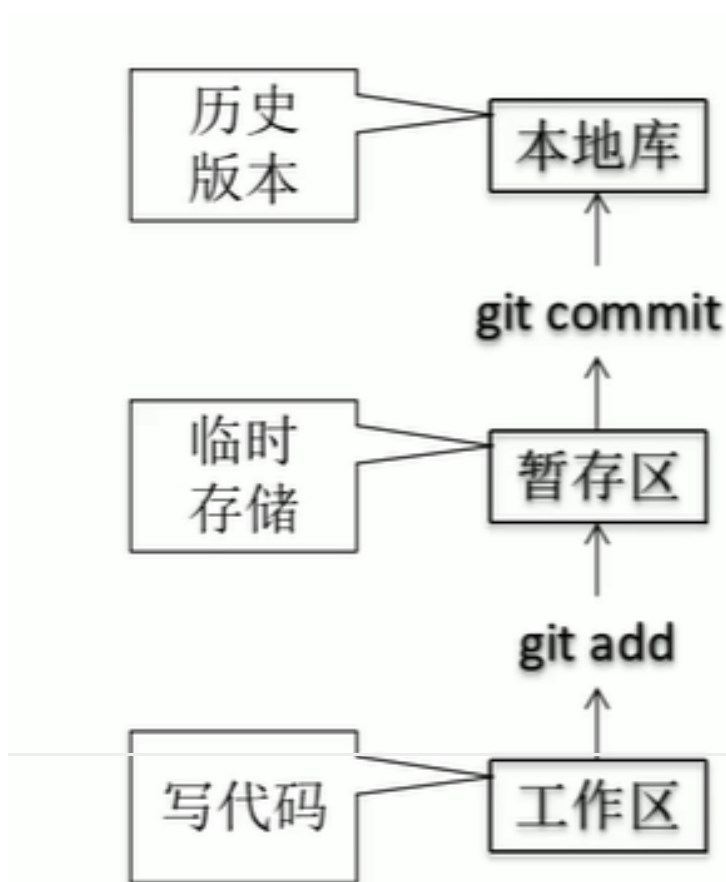
Git本地有三个工作区域：工作目录（Work Dictionary），暂存区（Stage/index），资源库（Repository）。如果加上远程的Git仓库（Remote Dictionary）就可以划分为四个工作区域。文件在这四个区域间的转化关系如下：

工作区：平时存放代码的地方。

暂存区：用于临时存放一些改动，可以保存即将提交到文件列表里的信息。

仓库区（本地库）：安全存放数据的位置，里面有提交的所有版本的数据。其中HEAD指向最新放入仓库的版本。

远程仓库：托管代码的服务器，可以进行远程的数据交换。



## 工作流程

Git的工作流程：

- 1、创建本地仓库。
- 2、在工作目录中添加/修改文件。
- 3、将需要进行版本管理的文件放入暂存区域。
- 4、将暂存区的内容提交到git本地仓库。
- 5、将本地仓库的代码推送至远程仓库。

## Git项目搭建

本地仓库搭建

- 1、建立全新的项目，使用git管理的项目的根目录执行

```
# 在当前目录新建一个Git代码库
$ git init
```

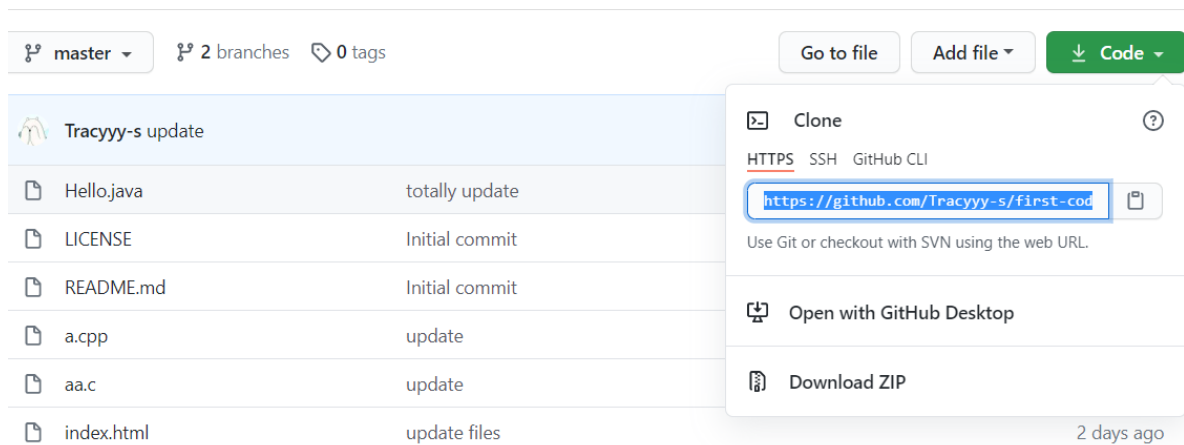
- 2、执行完毕后可以看到在根目录下多出了一个.git目录，关于版本的所有信息都在这个目录下。

### 克隆远程仓库

另一种方式是克隆远程仓库，是将远程服务器上的仓库完全镜像拉取至本地。

- 1、使用HTTPS URL进行克隆

```
# 克隆镜像至本地
$ git clone [url]
```



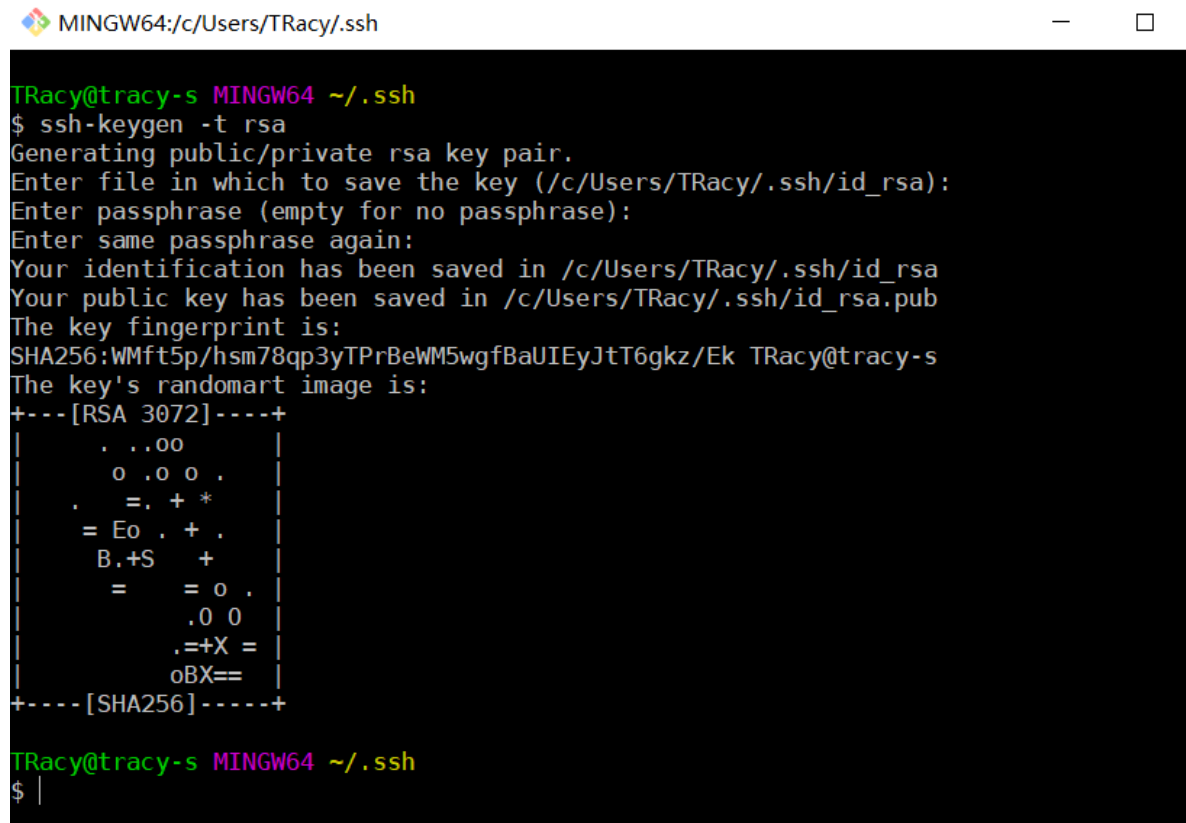
- 2、使用SSH URL进行克隆

①进入用户主目录进入.ssh文件夹（如果没有就创建一个）

进入bash输入命令：

```
# 获取本机公钥和私钥
ssh-keygen -t rsa
```

连接三次回车：

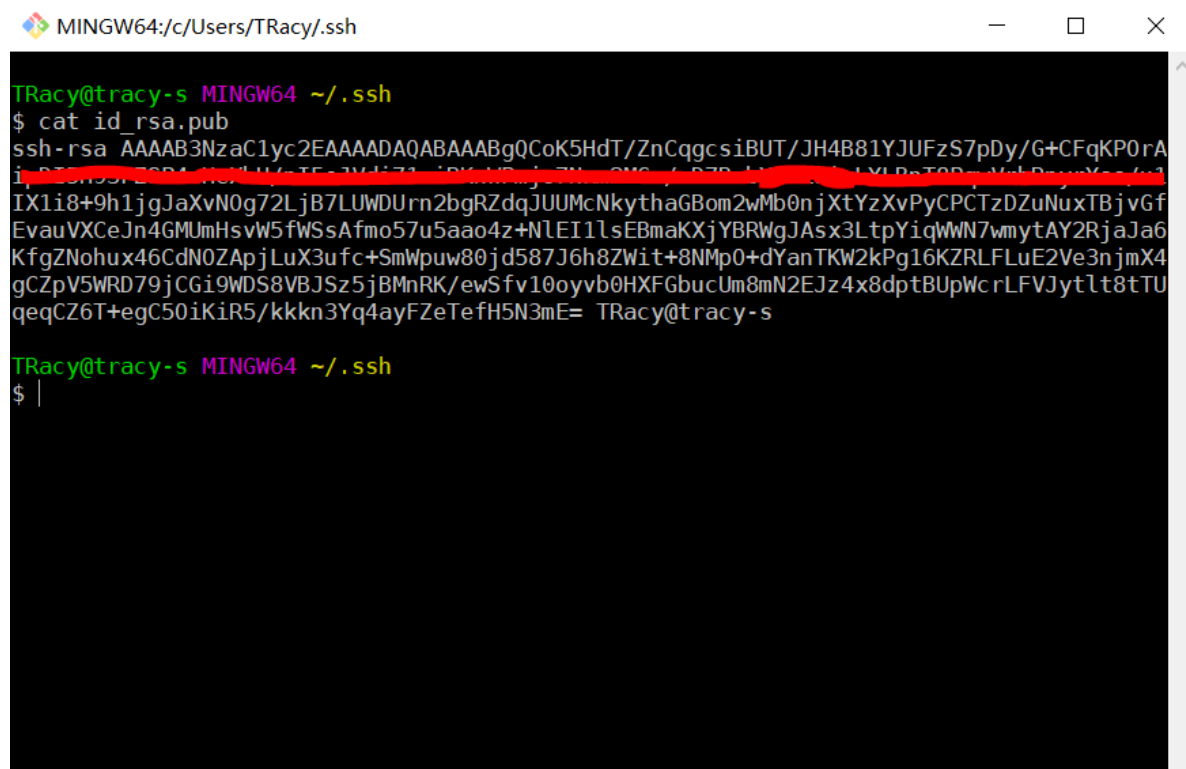


```
MINGW64:/c/Users/TRacy/.ssh

TRacy@tracy-s MINGW64 ~/.ssh
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/TRacy/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/TRacy/.ssh/id_rsa
Your public key has been saved in /c/Users/TRacy/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:WMft5p/hsm78qp3yTPrBewM5wgfBaUIEyJtT6gkz/Ek TRacy@tracy-s
The key's randomart image is:
+---[RSA 3072]---+
|      . . .oo      |
|      o .o o .     |
|      . =. + *      |
|      = Eo . + .    |
|      B.+S +        |
|      = = o .       |
|      .o o          |
|      .+=X =        |
|      oBX==         |
+-----[SHA256]-----+

TRacy@tracy-s MINGW64 ~/.ssh
$ |
```

②查看生成的id\_rsa.pub文件，选中本机的公钥



```
MINGW64:/c/Users/TRacy/.ssh

TRacy@tracy-s MINGW64 ~/.ssh
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCoK5HdT/ZnCqgcsiBUT/JH4B81YJUFzS7pDy/G+CFqKP0rA
IX1i8+9h1jgJaXvN0g72LjB7LUWdUrn2bgRZdqJUUMcNkythaGBom2wMb0njXtYzXvPyCPCTzDZuNuxTBjvGf
EvauVXCeJn4GMUmHsvW5fWSsAfmo57u5aao4z+NLEI1lsEBmaKXjYBRWgJAsx3LtpYiqWWN7wmytAY2RjaJa6
KfgZNohux46CdN0ZApjLuX3ufc+SmWpuw80jd587J6h8ZWit+8NMp0+dYanTKW2kPg16KZRLFLuE2Ve3njmX4
gCZpV5WRD79jCGi9WDS8VBJSz5jBMnRK/ewSfv10oyvb0HXFGbucUm8mN2EJz4x8dptBUUpWcrLFVJytlt8tTU
qeqCZ6T+egC50iKiR5/kkkn3Yq4ayFZeTefH5N3mE= TRacy@tracy-s

TRacy@tracy-s MINGW64 ~/.ssh
$ |
```

③在Github中添加创建的公钥并命名

④使用公钥clone项目

## SSH keys / Add new

Title

TRacy@tracy-s

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGCok5HdT/ZnCqgcsiBUT/JH4B81YJUFzS7pDy/G+CFqKPOrAipDI5H93FZSP
4cHeXbU/nI5eJVdi71wiRKnWPmjo7Nxm2MGm/vD7BubYVVLaIrLYLBnT8PqwVrhRnyrYoo/u1IX1i8+9h1jgJaXvNOg72
g57LUWb0mEbgRZLpUUMfNlykhaSBom2n4N0-jYX-X-7
sAfmo57u5aao4z+NIEl1sEBmaKXjYBRWgJAsx3LtpYiqWWN7wmytAY2RjaJa6KfgZNohux46CdNOZApjLuX3ufc+Sm
Wpuw80jd587J6h8ZWit+8NMPo+dYanTKW2kPg16KZRLFLuE2Ve3njmX4gCZpV5WRD79jCGi9WDS8VBJSz5jBMnRK
/ewSfv10oyvb0HXFGbucUm8mN2EJz4x8dptBUpWcrLFVJytl8tTUqeqCZ6T+egC5OiKiR5/kkkn3Yq4ayFZeTefH5N3mE
= TRacy@tracy-s
```

Add SSH key

master 2 branches 0 tags

Tracyyy-s update

Hello.java	totally update
LICENSE	Initial commit
README.md	Initial commit
a.cpp	update
aa.c	update
index.html	update files

Clone

HTTPS SSH GitHub CLI

git@github.com:Tracyyy-s/first-code.git

Use a password-protected SSH key.

Open with GitHub Desktop

Download ZIP

2 days ago

## Git文件操作

查看文件状态

```
# 查看指定文件状态
git status [filename]

# 查看所有文件状态
git status

# 将指定文件添加到暂存区
git add [filename]

# 将所有文件提交到暂存区
git add .

# 提交暂存区的内容到本地仓库，并添加说明 -m 提交信息
git commit -m "message"

# 将文件提交至远程仓库
git push
```

```
# 将文件移出暂存区
git restore --staged [filename]
```

## Git冲突解决

如果不是基于GitHub远程库的最新版本进行修改，则无法推送，必须先进行拉取

产生错误后对冲突的代码块进行修改，然后提交至远程仓库。

## Git分支

git中分支的常用命令

```
# 列出所有本地分支
git branch

# 列出所有远程分支
git branch -r

# 新建一个分支，但依然停留在当前分支
git branch [branch-name]

# 新建一个分支，并切换到该分支
git checkout -b [branch]

# 在指定分支提交文件
git push --set-upstream origin dev

# 合并指定分支到当前分支
git merge [branch]

# 删除分支
git branch -d [branch-name]

# 删除远程分支
git push origin --delete [branch-name]
git branch -dr [remote/branch]
```

如果同一个文件在合并时都被修改了则会引起冲突：解决方法是可以修改冲突文件后重新提交。

master主分支应该非常稳定，用来发布新版本，一般情况下不允许在上面工作，工作一般情况在新建的dev环境下，

若要发布，可以将dev分支合并到master上来。

## Git版本控制

查看本地库历史版本

```
# 查看历史记录
git log

# 多屏显示控制方式:
#   空格向下翻页
#   b  向上翻页
#   q  退出

# 显示完整的版本号，并且每个版本只打印一行
git log --pretty=oneline

git log --oneline

# 显示部分版本号，以及当前版本指针所指向的位置
git reflog
```

前进后退

MINGW64:/e/gitcode/first-code

```
TRacy@tracy-s MINGW64 /e/gitcode/first-code (master)
$ git reflog
890b290 (HEAD -> master) HEAD@{0}: reset: moving to 890b290
daf025d (origin/master, origin/HEAD) HEAD@{1}: reset: moving to daf025d
067515a HEAD@{2}: reset: moving to 067515a
daf025d (origin/master, origin/HEAD) HEAD@{3}: checkout: moving from dev to master
4170c0d (origin/dev, dev) HEAD@{4}: commit: touch a file
daf025d (origin/master, origin/HEAD) HEAD@{5}: checkout: moving from master to dev
daf025d (origin/master, origin/HEAD) HEAD@{6}: commit: update
5df6390 HEAD@{7}: commit: totally update
73c05f0 HEAD@{8}: commit: a cpp code
067515a HEAD@{9}: commit: update all
890b290 (HEAD -> master) HEAD@{10}: commit: update a cpp code
4f5e88a HEAD@{11}: commit: update files
37a5015 HEAD@{12}: clone: from git@github.com:Tracyyy-s/first-code.git

TRacy@tracy-s MINGW64 /e/gitcode/first-code (master)
$ |
```



# 基于索引值操作【推荐】

`git reset --hard [局部索引值]`

`git reset --hard 890b290`

# 使用^符号，只能后退

# 一个^表示后退一步，n个表示后退n步

`git reset --hard HEAD^`

# 使用~符号，只能后退

# ~后表示后退n步

`git reset --hard HEAD~n`