

# PHOENIX嵌入式培养之Git的使用

---

## PHOENIX嵌入式培养之Git的使用

### 版本管理之Git使用

为什么要使用Git

一个版本管理的教训

版本控制软件

基本功能

分布式版本控制软件

Git和Github Desktop安装

Windows与Ubuntu的安装

设置Git用户名

Windows进命令行的方式

Git的理论知识

Git 记录的是什么？

三个区域

实战一下

初始化Git

将文件添加到暂存区

将文件提交到 Git 仓库

状态

一步完成添加到暂存区和提交命令

回到过去

内容回顾

回滚快照

版本比较

比较暂存区域与工作目录

比较两个历史快照

比较当前工作目录与快照

修改最后一次提交、删除文件、重命名文件

删除文件

修改最后一次提交

重命名文件

创建和切换分支

分支是什么？

创建分支

切换分支

合并和删除分支

合并分支

删除分支

从历史中拷贝一文件

本地仓库连接到Github

生成本地秘钥

在github中添加公钥

上传本地库文件

Git的基础命令

## 版本管理之Git使用

---

# 为什么要使用Git

Git 是一款分布式的代码版本控制工具，Linux 之父 Linus 嫌弃当时主流的中心式的版本控制工具太难用还要花钱，就自己开发出了 Git 用来维护 Linux 的版本。Git 的设计非常优雅，但初学者通常因为很难理解其内部逻辑因此会觉得非常难用。



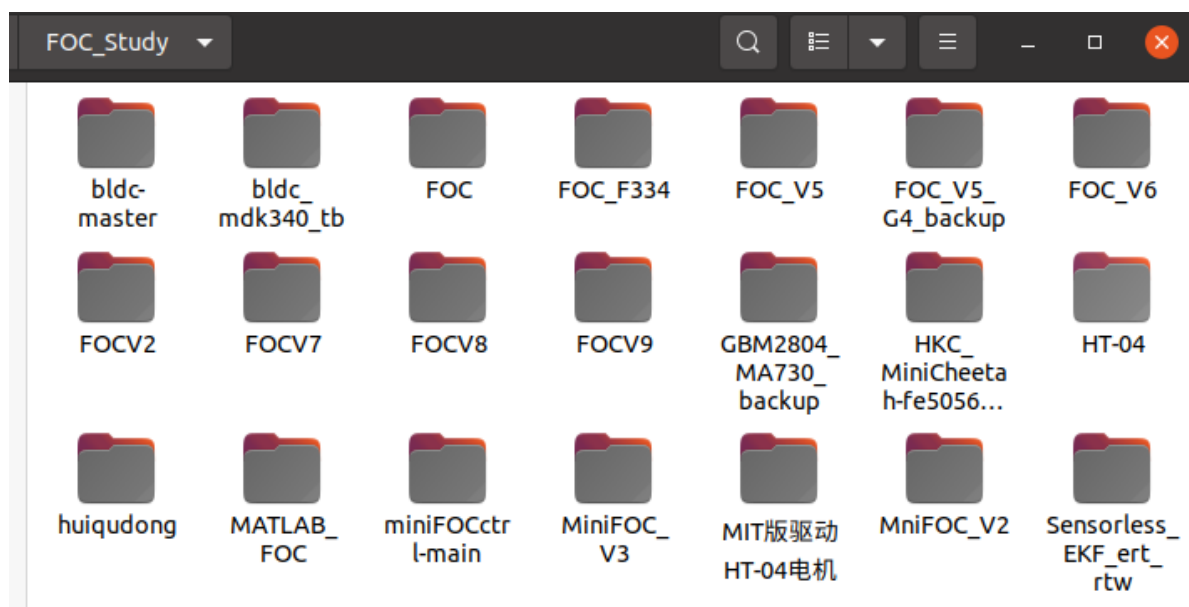
对 Git 不熟悉的初学者很容易出现因为误用命令将代码给控制版本控制没了的状况。但相信我，和 Vim 一样，Git 是一款你最终掌握之后会感叹“它值得！”的神器。

而对于我们嵌入式而言，在大学里掌握Git无论是对如今学习代码整理，还是对以后的求职或者是做更大的

项目都是百利而无一害的！



## 一个版本管理的教训



之前我在B站发了一个基于stm32f1芯片的FOC跑到10KHZ的视频，在那之后就大概有半年没管FOC了，最近当我想找到那个工程文件将它开源时发现已经不知道哪个版本是能用的了。上面的文件夹图片里可以看到我的FOC版本从V1到了V9，而在我记忆里并不是最新版本是能用的，每一个版本只是代表我要进行一次重要操作时的备份，并没有详细记录每个版本有什么区别，所有现在回过头来发现了一堆麻烦。

所以文件版本管理固然重要，很多同学可能都是像我一样备份，这样至少保证了重要的历史记录，同时可以恢复数据，只是回过头来比较不好找。而且文件多了要改很多个V几V几的就很麻烦，所以需要有一个版本控制软件。

## 版本控制软件

### 基本功能

#### 1、保存和管理文件

也就是有一个软件可以自动帮我们生成上面那样的版本号，然后把每个版本的文件存在一个仓库里。

#### 2、提供客户端工具进行访问

因为既然是版本管理，那么每个存下来的版本不应该是我们能随便更改的，所以存在仓库里要提供给用户一个客户端工具进行访问这些文件进行操作。

#### 3、提供不同版本的比对功能

如果只是标记个版本号就会像上面说的那样找不到该要的版本，所以版本管理软件还应该给我们一个版本比对功能，让我们知道每个版本的差异，到时候回来找版本就一目了然了。

## 分布式版本控制软件

Git属于分布式版本控制软件，就是云端有一个仓库，本地也有一个跟云端一模一样的仓库，我们修改内容不直接修改云端服务器的内容，而是修改本地仓库的内容，等到连网的时候再把本地的仓库同步到云端去。

## Git和Github Desktop安装

### Windows与Ubuntu的安装

Windows版本的话只需要安装<https://git-scm.com/downloads>进这个网站安装即可，Github Desktop就在<https://desktop.github.com/>这里下载安装。



两个都是国外网站下载慢，我这里提供一下百度网盘下载链接：

64位 -> 链接：<http://pan.baidu.com/s/1nv6l9rr> 密码：4plg


32位 -> 链接：<http://pan.baidu.com/s/1bp3Flnx> 密码：l0qi

Ubuntu的话就执行 `# apt-get install git` 就可以安装Git了。而Github Desktop并没有Linux版本，可以用鱼香ROS的一键安装，终端执行“`wget http://fishros.com/install -O fishros && . fishros`”然后看出来的菜单，选Github 桌面版即可。

### 设置Git用户名

成功安装完 Git 的第一件事儿就是让它知道主人是谁！

这个操作非常重要，因为每一个 Git 的提交都会使用到这些信息，**一旦确定不可更改**。

在命令行模式里输入以下命令（不懂怎么进入命令行的妹子看最下边 ）：

```
> git config --global user.name "your name"
> git config --global user.email "your email"
```

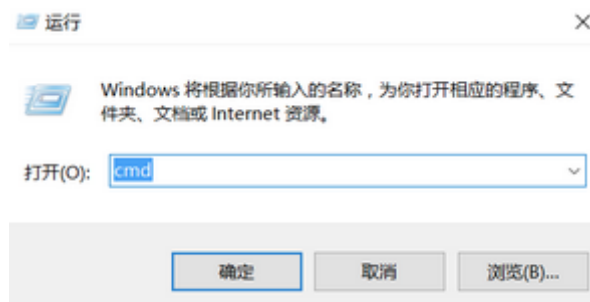
注：双引号中的内容改为自己的信息(๑๐๑)哦，否则你今后的劳动成果可就要算我的头上了~

使用 `git config --list` 命令可以查看信息是否写入成功：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM: ~  
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ git config --global user.name "171. 1713"  
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ git config --global user.email "141. 141@qq.com"  
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ git config ==list  
error: key does not contain a section: ==list  
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ git config --list  
user.name=171. 1713  
user.email=141. 141@qq.com  
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$
```

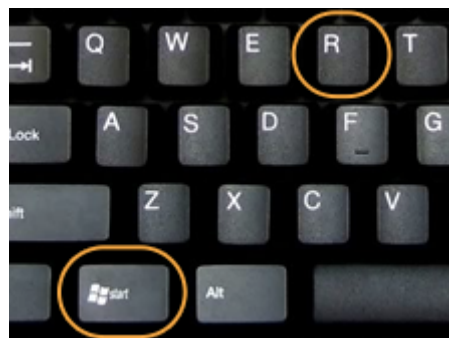
## Windows进命令行的方式

点击开始菜单（Win10 是在左下角点右键） -> 点击“运行”-> 输入 cmd -> 回车



会弹出来一个带文字的大黑框，就是了.....

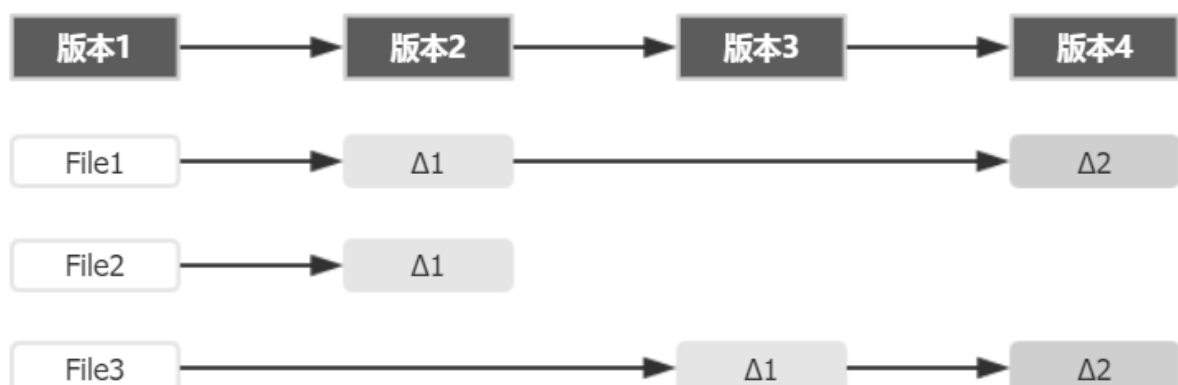
噢，对了，使用快捷键 Win + r 也可以直接打开“运行”



## Git的理论知识

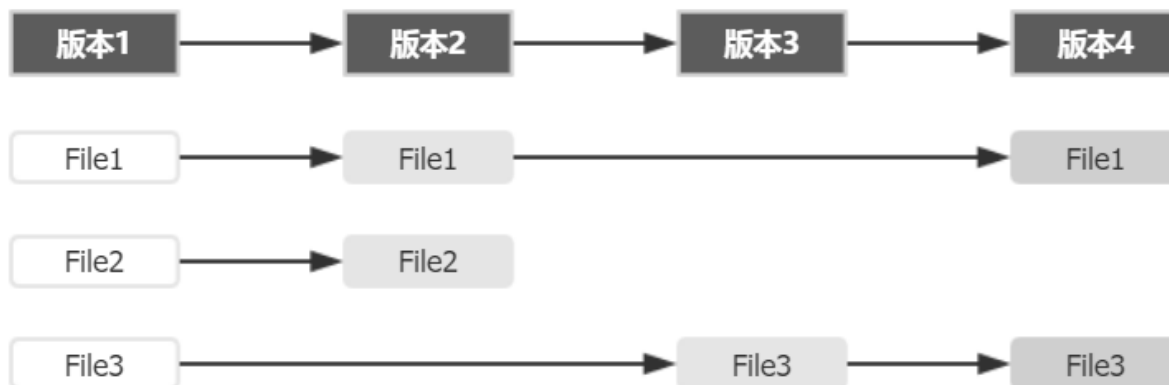
### Git 记录的是什么？

如果你有使用 SVN 等其他版本控制系统的经验，你应该知道它们的工作原理是记录每一次的变动。



差不多就是上面酱紫，每一次版本迭代，SVN 记录的是文件的变化内容。通常让我们自己来写一个版本管理工具也会首选这样的思维吧？就像写小说一样，每次就增加一个章节，修改若干错别字，最终装订成册.....没必要为每次的修改都拷贝一整本书！这种存储方式也是有名堂的，叫增量文件系统（Delta Storage systems）。

而 Linux 童鞋这次却决定剑走偏锋，以一种看似“异端”的方式来处理版本迭代：



如上，如果每个版本中有文件发生变动，Git 会将整个文件复制并保存起来。这种设计看似会多消耗更多的空间，但在分支管理时却是带来了很多的益处和便利（分支管理我们后边会讲，不急）。

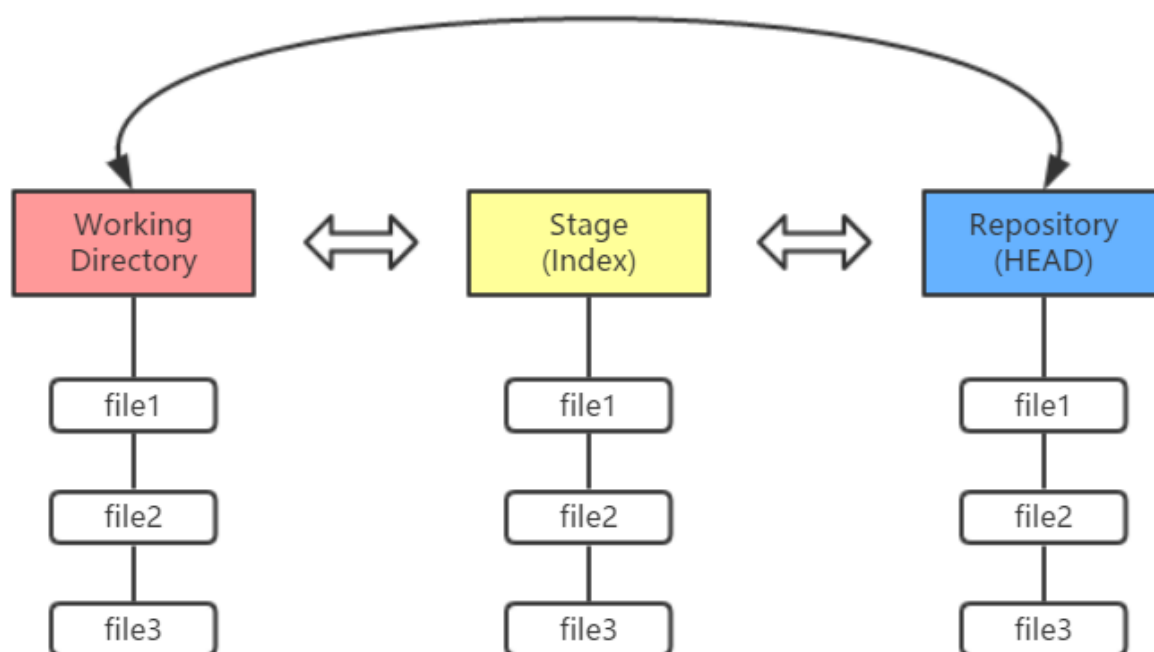
突然想到了一句话：普通的程序员是把很多的时间放在写代码和调 Bug 上，而优秀的程序员是将更多的精力放在设计上.....🧐

警告：前方高能，请先洗脸！！

## 三个区域

你的本地仓库有 Git 维护的三棵“树”组成，这是 Git 的核心框架。

这三棵树分别是：工作区域、暂存区域和 Git 仓库。




工作区域（Working Directory）就是你平时存放项目代码的地方。暂存区域（Stage）用于临时存放你的改动，事实上它只是一个文件，保存即将提交的文件列表信息。Git 仓库（Repository）就是安全存放数据的位置，这里边有你提交的所有版本的数据。其中，HEAD 指向最新放入仓库的版本（这第三棵树，确切的说，应该是 Git 仓库中 HEAD 指向的版本）。

OK，Git 的工作流程一般是酱紫：

1. 在工作目录中添加、修改文件；
2. 将需要进行版本管理的文件放入暂存区域；
3. 将暂存区域的文件提交到 Git 仓库。

因此，Git 管理的文件有三种状态：已修改（modified）、已暂存（staged）和已提交（committed），依次对应上边的每一个流程。

等等.....看到这里，你们肯定有疑惑：“你说 Git 仓库用于存放每次的版本迭代，我可以理解。但为何还要多增加一个暂存区域呢？”

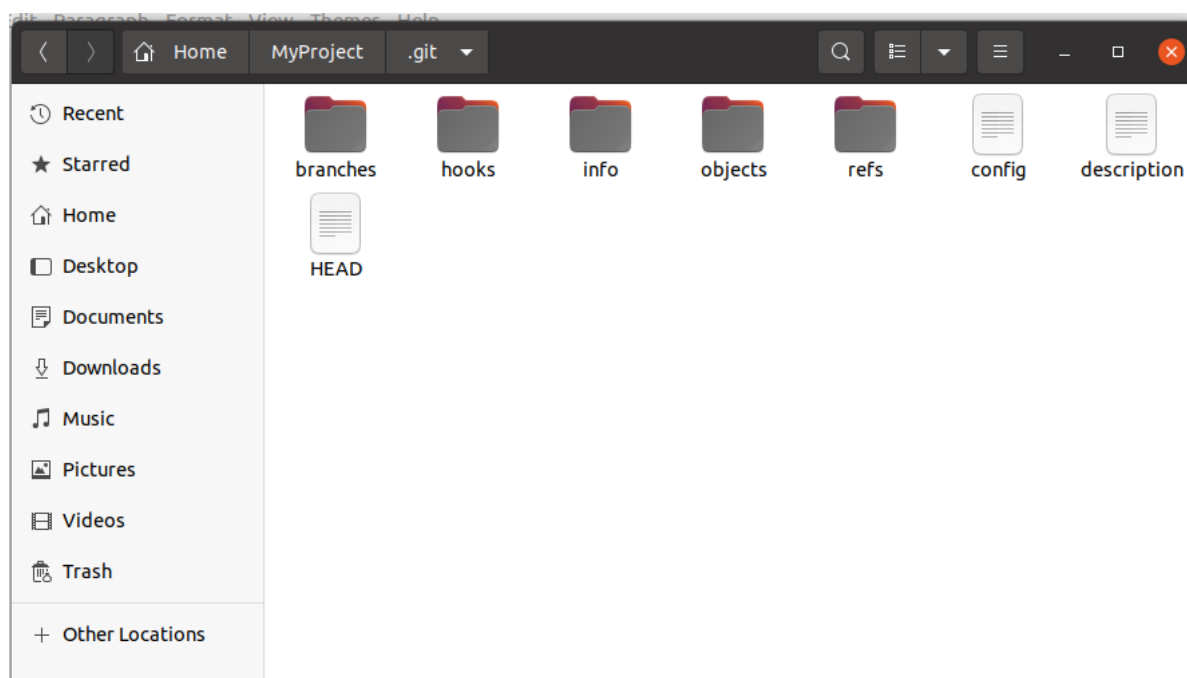
我这里打个比方：像某些厂家开发一个产品，一般他们都留有一手，不会把该产品的所有特性一次性发布。通过产品的迭代，每年秋季你就可以开开心心地买到又有一两项新功能的“新”产品了。 我好像知道的太多了.....So，有时你并不想把工作目录中所有的新功能都提交到最新版本，你就可以先添加一些本次需要提交的文件到暂存区，然后从暂存区中提交它们.....所以暂存区在江湖中有个外号叫“索引”（Index）。



记住这三棵树，因为后边教的所有 Git 操作基本上都是在这三棵树之间搞来搞去！

## 实战一下

### 初始化Git

首先要建立一个大本营（确定你的工作目录），这里我们在新建一个叫做 MyProject 的新文件夹（这里你可以自己找个位置实验，但路径中尽量不要出现中文符号）。然后将命令行窗口的工作路径切换到刚才创建的 MyProject 中，输入命令 **git init** 即可初始化 Git。然后你会看到 MyProject 文件夹中出现一个叫做 .git 的**隐藏文件**（这个文件夹就是 Git 用来跟踪管理版本迭代的）：



然后我们新建一个README文件吧，写任何大项目之前，先要写一个 README.md 的文档（md 后缀是 Markdown 语言写的文本，现在可火啦 ）不过前面在交笔记的时候已经叫你们学啦。

## 将文件添加到暂存区


回到命令行窗口，输入 `git add README.md` 命令：

```
~/MyProject$ touch README.md
~/MyProject$ git add README.md
```

## 将文件提交到 Git 仓库

输入 `git commit -m "add a readme file"` 将暂存区域里的东西提交到 Git 仓库中：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "add a readme file"
[master (root-commit) da71e70] add a readme file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

好了，这次有消息了。但并不是坏消息，只是 Git 告诉你有 1 个文件被改动（README.md），插入了 1 行内容。及时反馈工作进度，是一个程序对用户的基本尊重！对了，commit 是提交的意思，-m


选项后边跟着的是本次提交的说明，就是大概描述这一版本做了哪些改动，以便今后可以迅速查看。

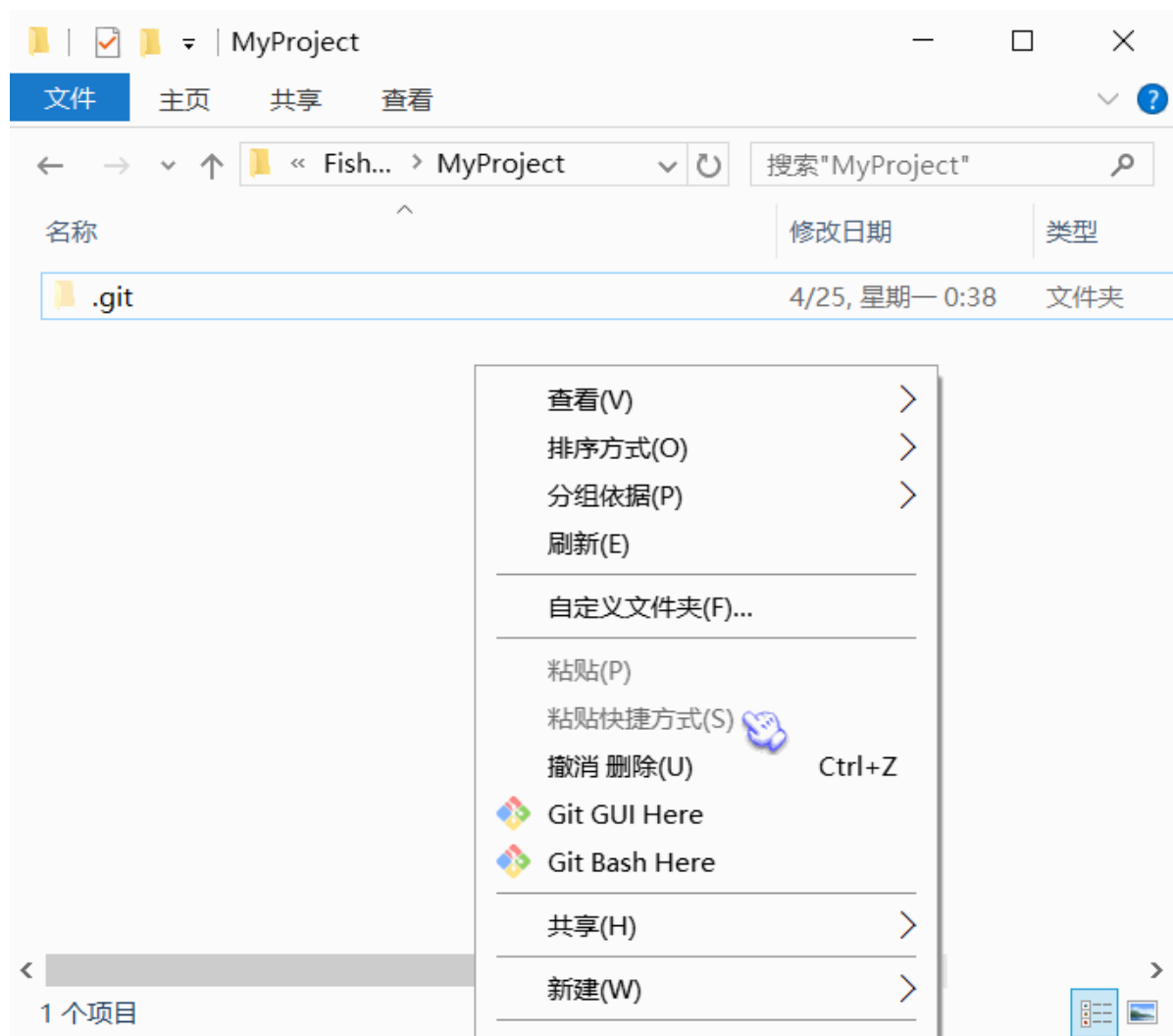
需要注意的是：对于这个提交的说明，Git 是强制要求你必须写的。如果没有使用 -m 选项，Git 会自动打开一个编辑器，要求你在其中输入提交说明，输入完毕后保存退出即可（操作命令与 vim 编辑器一致）。

总结一下，将工作目录的文件放到 Git 仓库只需要两步：

**Step One -> git add 文件名**

**Step Two -> git commit -m "你干了啥"**

有些同学是不是有点 Yin 乱了？没关系，献上我的动图：



## 状态

既然大家都是程序猿，当然要以代码担当。我们可忙了，一个项目百八十个文件，怎么知道哪些文件是新添加的，哪些文件已经加入了暂存区域呢？总不能让我们自己拿个本本记下来吧？🤖

当然不，作为世界上最伟大的版本控制系统，你能遇到的困境，Git 早已有了相应的解决方案。你唯一需要做的，就是.....学习...🧐🎵

随时随地，你都可以使用 **git status** 命令查看当前的状态.....上面的代码如果你没有动过，那么输入 **git status** 命令后应该是下边这样：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
nothing to commit, working tree clean
```

**On branch master** 说明我们位于一个叫做“master”的分支里，这是默认的分支（相比其他版本管理程序，Git 的分支理念领先世界好几年🧐放心，在适当的时候小甲鱼会详细地给大家讲解分支的）。

**nothing to commit, working directory clean** 说明你的工作目录目前是“干净的”，没有需要提交的文件（意思就是自上次提交后，工作目录中的内容压根儿就没改动过）。

既然要做大项目，那么应该增加一个版权声明。为了显得大公无私，我们采用 MIT 许可证给予用户最大的权利，让全世界的开发者都参与进来！在工作目录中增加 LICENSE 文件，内容如下：



Copyright (C) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

现在我们输入 **git status** 命令，看下 git 有什么想法：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LICENSE

nothing added to commit but untracked files present (use "git add" to track)
```

**Untracked files** 说明存在未跟踪的文件（下边红色的那个）

所谓的“未跟踪”文件，是指那些新添加的并且未被加入到暂存区域或提交的文件。它们处于一个逍遥法外的状态，但你一旦将它们加入暂存区域或提交到 Git 仓库，它们就开始受到 Git 的“跟踪”。

这里圆括号中的英文是 git 给我们的建议：使用 **git add** 命令将待提交的文件添加到暂存区域。（然后

Git 就可以对它们嘿嘿嘿



那我们不妨按照建议来操作：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add LICENSE
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   LICENSE
```


add 命令如果执行成功，并不会有提示消息。

大家看到，首先是文件被绿了（都说是给 Git 嘿嘿嘿了嘛）

use **"git reset HEAD .." to unstage** 的意思是“如果你反悔了，你可以使用 **git reset HEAD** 命令恢复暂存区域”。试试看吧

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git reset HEAD
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LICENSE

nothing added to commit but untracked files present (use "git add" to track)
```

还真回去了！ 好吧，再次添加到暂存区域，然后执行 `git commit -m "add a license file"` 命令：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add LICENSE
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m LIC
ENSE
[master bbb372f] LICENSE
1 file changed, 7 insertions(+)
create mode 100644 LICENSE
```

然后我们修改一下LICENSE里面的内容（随便改几个字吧），再看一下 `git status`：



```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LICENSE

no changes added to commit (use "git add" and/or "git commit -a")
```

由于你对工作目录的文件进行了修改，导致这个文件和暂存区域的对应文件不匹配了，所以 Git 又给你提出两条建议：

- 使用 `git add` 命令将工作目录的新版本覆盖暂存区域的旧版本，然后准备提交
- 使用 `git checkout` 命令将暂存区域的旧版本覆盖工作目录的新版本（危险操作：相当于丢弃工作目录的修改）

还有一种情况我们没分析，大家先把新版本的文件覆盖掉暂存区域的旧版本：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add LICENSE
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   LICENSE
```


然后我们打开 LICENSE 文件，将 **ChenLZ** 改为 **ChenLZ.com**，保存.....再次查看状态：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   LICENSE

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LICENSE
```

这次诡异了：被绿的 LICENSE 说明文件存放在暂存区域（待提交），同时红色的 LICENSE 说明文件还在工作目录等待添加到暂存区域。

这种情况你应该意识到这里存在两个不同版本的 LICENSE 文件，这时如果你直接执行 `commit` 命令，那么提交的是暂存区域的版本（ChenLZ），如果你希望提交工作目录的新版本（ChenLZ.com），那么你需要先执行 `add` 命令覆盖暂存区域，然后再提交.....

烦不烦？！ 老是要 `add` 再 `commit`，很烦恼吧？有没有办法一步到位？答案当然是

**Yes !**

## 一步完成添加到暂存区和提交命令

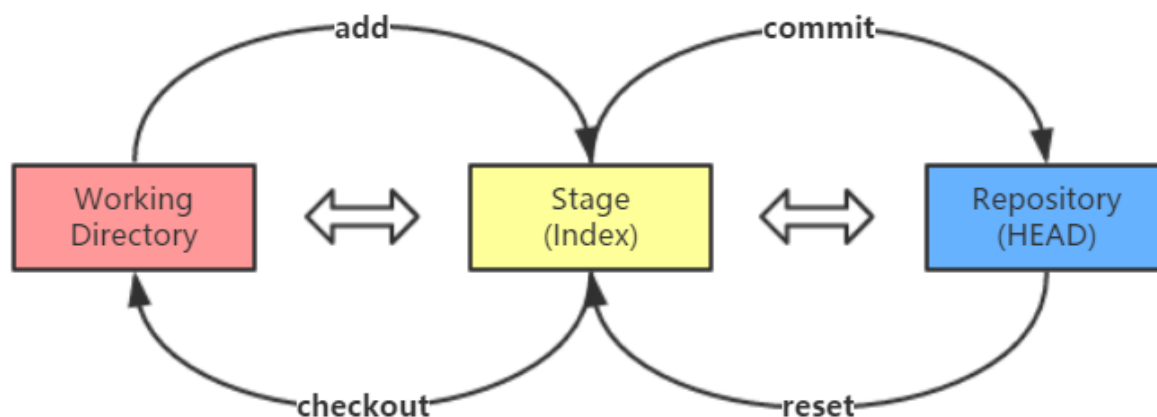
执行 `git commit -am "提交描述"` 即可将add和commit操作合并, 不需要先`git add file` 再 `git commit -m "提交描述"` 了

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -am "
改了权限文件，再一步完成提交"
[master 87f2214] 改了权限文件，再一步完成提交
1 file changed, 1 insertion(+), 1 deletion(-)
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
nothing to commit, working tree clean
```

## 回到过去

### 内容回顾

我们先理一下上面讲的几个命令：



现在几个命令应该相当清晰了：

- `git add` 命令用于把工作目录的文件放入暂存区域
- `git commit` 命令用于把暂存区域的文件提交到 Git 仓库
- `git reset` 命令用于把 Git 仓库的文件还原到暂存区域
- `git checkout` 命令用于把暂存区域的文件还原到工作目录

前边两个命令我有信心你已经相当熟悉了，但后边两个千万别说你已懂，因为它们是 Git 里边最复杂的命令（它们的功能可不止上边文字描述的这么简单 🧐）

**先给大家重点讲解 `reset` 命令，`checkout` 命令在分支管理中再细讲。**

先执行 `git log` 命令查看历史提交：

```

chenliangzhi@chenliangzhi-R0G-Strix-G513QM-G513QM:~/MyProject$ git log
commit 87f22145dfb88b3567e6db4249ea49eb7474fc78 (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 12:27:54 2023 +0800

    改了权限文件，再一步完成提交

commit bbb372f52442640bb849f53dc417ca312a706cb0
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 12:17:22 2023 +0800

    LICENSE

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file

```

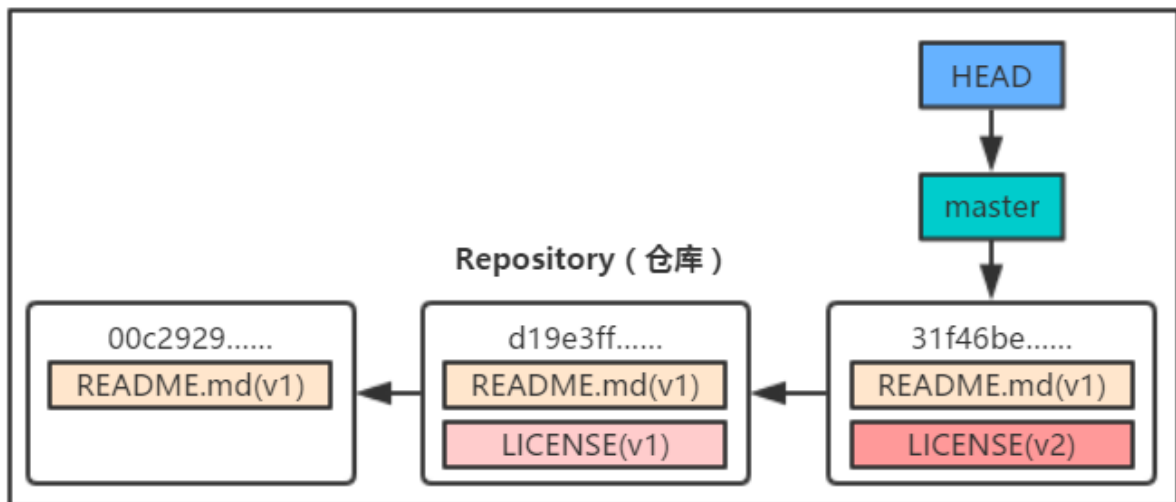
这里记录了我们之前的 3 次提交（排序是按时间从近到远的），**Author** 后边是提交者，**Date** 后边是提交日期，下边是当次提交的说明。那.....草黄色的那个 commit +“乱码”是什么鬼？🤔 咳咳~~

这里并没有什么“乱码”，这个是 Git 为每次提交计算出来的 ID，它其实一个完整的 SHA-1 校验和（尽管你的文件内容可能跟我的完全一致，但这个值却不一样，这是因为账号、时间不同而导致）。你不需要知道 SHA-1 的原理，只需要知道它在任何时候都是唯一的，通过这个 ID，你就可以找到对应的那个版本。

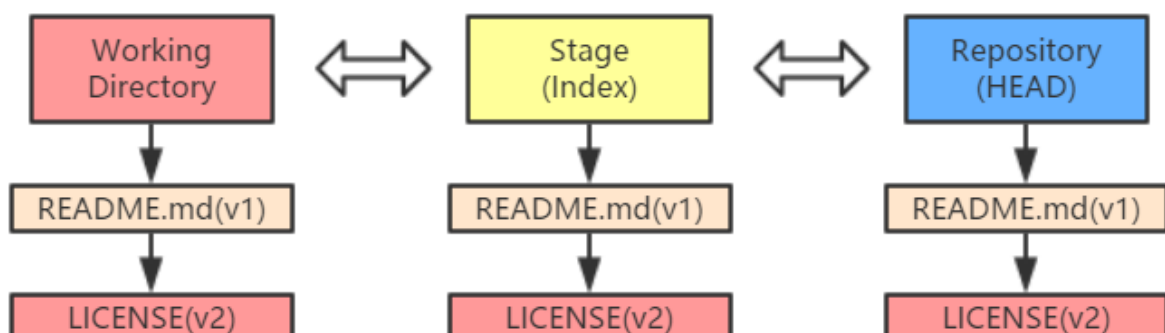
为何 ID 不是传统的 1、2、3、4、5..... 难道作者是为了装 X？

非也非也，因为 Git 是分布式的版本控制系统，如果多人同时工作，那么使用 1、2、3、4、5 就很容易产生冲突，继而打群架.....

根据 log 记录，现在我们将 Git 仓库如果可视化，应该是这样子：



三棵树现在应该是下面酱紫：



## 回滚快照


注：快照即提交的版本，每个版本我们称之为一个快照。

现在我们利用 reset 命令回滚快照，并看看 Git 仓库和三棵树分别发生了什么。

执行 **git reset HEAD~** 命令：

注：HEAD 表示最新提交的快照（87f22），而 HEAD~ 表示 HEAD 的上一个快照（bbb37）

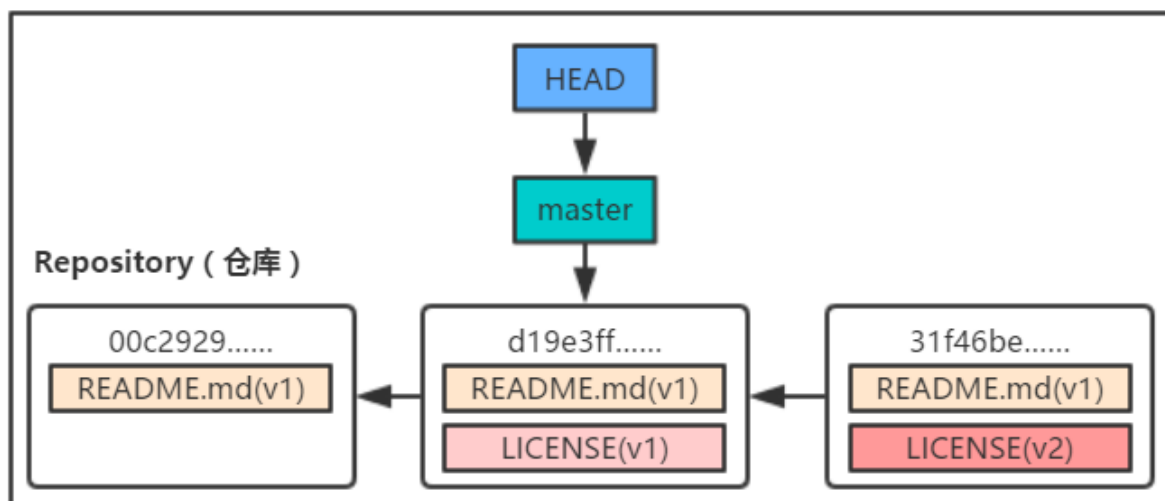
```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git reset HEAD~
Unstaged changes after reset:
M    LICENSE
```

然后执行 **git status** 命令查看现在  的状态：

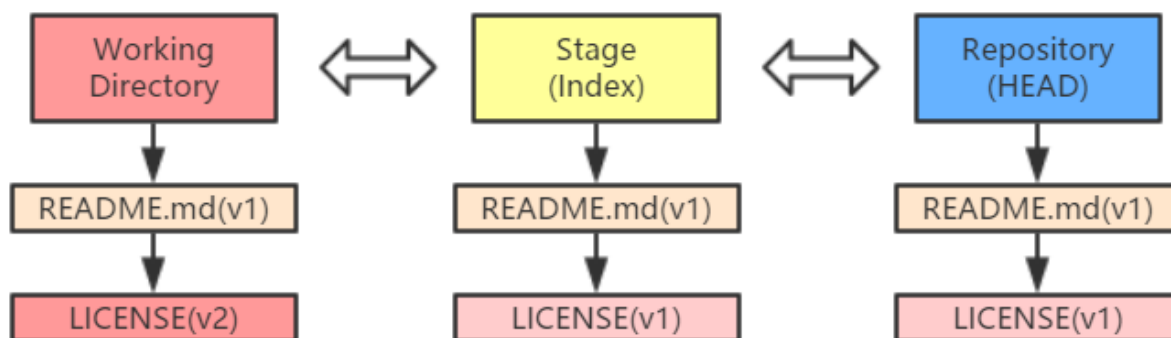
```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LICENSE


no changes added to commit (use "git add" and/or "git commit -a")
```

好了，现在执行完 **git reset HEAD~** 命令之后，Git 仓库应该是这样子：



三棵树现在应该是下面酱紫：



这里有一点要补充的：HEAD~ 表示 HEAD 的上一个快照（d19e3），HEAD~~（00c29）则表示 HEAD 的上上一个快照，如果希望表示上上上上上上上上上一个快照（数了一下，这里有 10 个“上” ），那么可以直接用 HEAD~10 来表示。

git reset HEAD~ 命令其实是 git reset --mixed HEAD~ 的缩写，因为 --mixed 选项是默认的，所以我们可以偷懒。我们发现，git reset HEAD~ 命令其实影响了两棵树：首先是移动 HEAD 的指向，将其指向上一个快照（HEAD~）；然后再将该位置的快照回滚到暂存区域。为了灵活地操纵这三棵树，Git 还为 reset 命令安排了 --soft 和 --hard 选项，可谓软硬兼施，不到你不服~

### --soft 选项

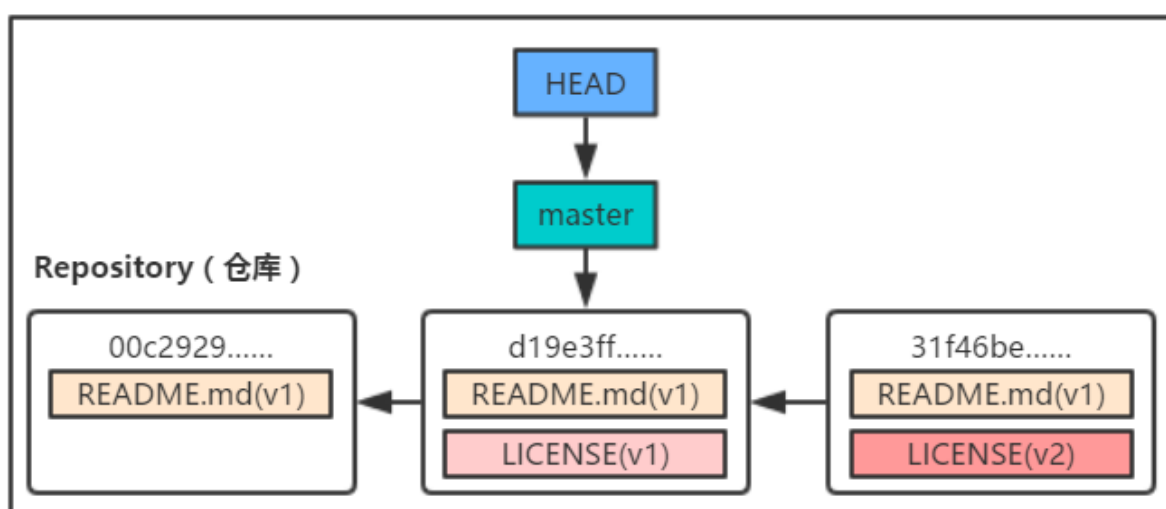
加上 --soft 选项的结果是使得 reset 变“软”了，也就没有原来那么持久..... So, git reset --soft

HEAD~ 命令就相当于只移动 HEAD 的指向，但并不会将快照回滚到暂存区域。这个选项有什么作用呢？事实它就是相当于撤消了上一次的提交（commit）。一不小心提交了，后悔了，那么你就执行 git reset --soft HEAD~ 命令即可（此时执行 git log 命令，也不会再看到已经撤消了的那个提交）。

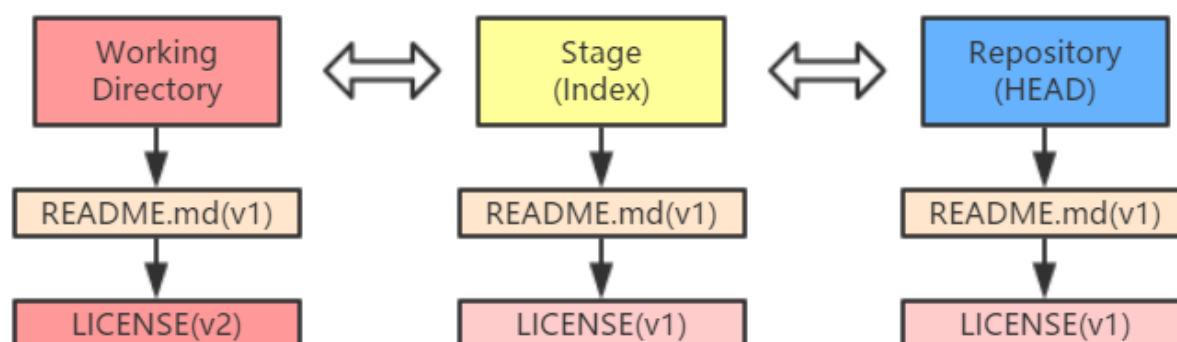
### --hard 选项

加上 --hard 选项的结果是使得 reset 变“硬”.....你猜的不错，加上 --hard 选项，reset 不仅移动 HEAD 的指向，将快照回滚到暂存区域，它还将暂存区域的文件还原到工作目录。

来，上点图吧！刚才执行完 git reset HEAD~ 命令后，Git 仓库里的数据是这样：



三棵树是这样：



那么在这种状态下，我再执行 git reset --hard HEAD~ 命令：

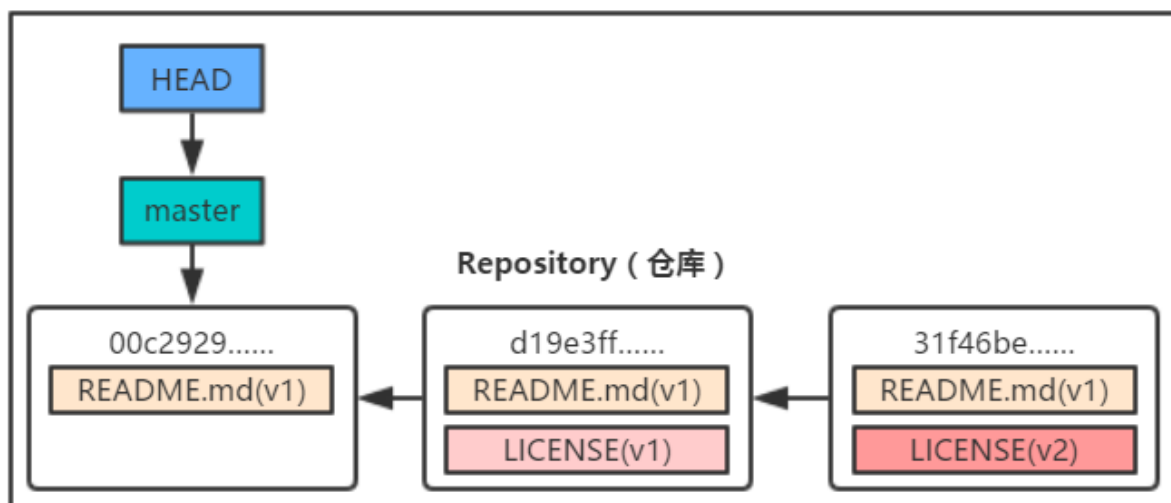
```
chenliangzhi@chenliangzhi-R0G-Strix-G513QM-G513QM:~/MyProject$ git reset --hard HEAD~  
HEAD is now at da71e70 add a readme file
```

Git 仓库中就剩下最后一个快照了：

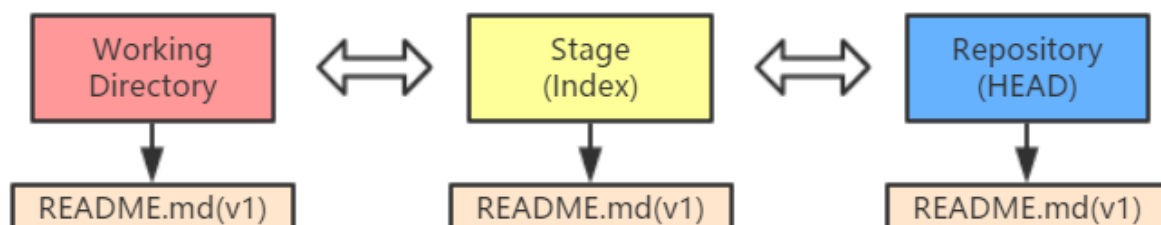
```
chenliangzhi@chenliangzhi-R0G-Strix-G513QM-G513QM:~/MyProject$ git log  
commit da71e700fac826ae53944662574a8311da1b75de (HEAD -> master)  
Author: 17805818393 <1411904617@qq.com>  
Date: Thu Aug 3 01:03:37 2023 +0800  
  
add a readme file
```



还原案发现场，Git 仓库现在应该是这样：



而三棵树现在应该都被回归到第一个版本：将快照回滚动到暂存区域，它还将暂存区域的文件还原到工作目录



最后总结一下：**reset 回滚快照三部曲**

1. 移动 HEAD 的指向 (--soft)
2. 将快照回滚到暂存区域 ([--mixed], 默认)
3. 将暂存区域还原到工作目录 (--hard)


**回滚指定快照**

如果快照比较多，你又懒得去数有多少个“上”，那么你可以通过指定具体的快照 ID 来回滚该快照。比如 `git reset 00c2929`，你不必把辣么长的 ID 号都给输入进去，一般只要输入前几位（5 位或以上吧）就可以了。

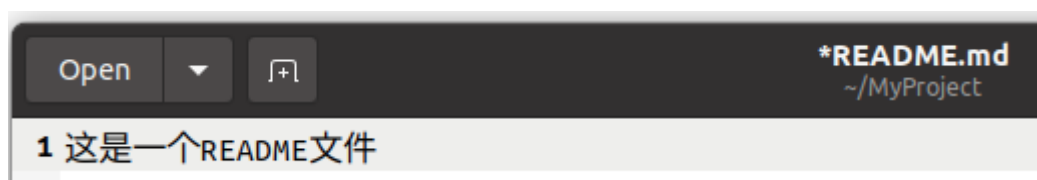


## 版本比较

### 比较暂存区域与工作目录

`status` 命令虽然可以查看到当前的工作状态，也有 Git 给你提供友情提示，但它只会告诉你版本发生了改变（哪个文件有变化，增加、减少了多少行），但它没法告诉你版本之间到底有哪些不同。而我们缺的，正正就是这个功能！！ (o\_v\_o) 嗯，缺啥来啥，今天我们要讲的命令叫 `diff`，就是 `different` 的缩写，这个命令是专门用来找茬的。


我们在刚刚的 `README.md` 文件中写一点文字：



再执行以下 `git status`：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

看到文件进行了修改，但是我们不知道他修改了什么哎，不知道的话我们不能确定这个版本是否是可行的，那怎么绝对是要备份呢？那我们直接执行 **git diff** 命令是比较暂存区域与工作目录的文

件内容：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git diff
diff --git a/README.md b/README.md
index e69de29..e360e56 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+这是一个README文件
```

What???这河里吗？ 不急，我一个个给你解释.....

首先，diff 命令是 linux 上非常重要的工具，用于比较文件的内容，特别是比较两个版本不同的文件以找到改动的地方。diff 程序的输出被称为补丁 (patch)，因为 Linux 系统中还有一个 patch 程序，可以根据 diff 的输出将 a.c 的文件内容更新为 b.c。

diff 现在已经是 Git 不可或缺的一部分了！值得一提的是，**现在的 diff 已经可以很好的支持二进制文件了**，比如 docx 文件以前不支持，现在都支持啦~(≥▽≤)/~

第一行：diff --git a/README.md b/README.md。表示对比的是存放在暂存区域的 README.md 和工作目录的 README.md

第二行：index e69de29..e360e56 100644。表示对应文件的 ID 分别是 e69de29 和 e360e56，左边暂存区域，后边当前目录。最后的 100644 是指定文件的类型和权限。喏，具体的定义我也帮你找出来了（Linux 系统的定义，看不懂没关系，我知道有人需要，所以顺便贴出来）：


```
The following flags are defined for the st_mode field:
S_IFMT      0170000  bit mask for the file type bit fields
S_IFSOCK    0140000  socket
S_IFLNK     0120000  symbolic link
S_IFREG     0100000  regular file
S_IFBLK     0060000  block device
S_IFDIR     0040000  directory
S_IFCHR     0020000  character device
S_IFIFO     0010000  FIFO
S_ISUID     0004000  set UID bit
S_ISGID     0002000  set-group-ID bit (see below)
S_ISVTX     0001000  sticky bit (see below)
S_IRWXU     00700    mask for file owner permissions
S_IRUSR     00400    owner has read permission
S_IWUSR     00200    owner has write permission
S_IXUSR     00100    owner has execute permission
S_IRWXG     00070    mask for group permissions
S_IRGRP     00040    group has read permission
S_IWGRP     00020    group has write permission
S_IXGRP     00010    group has execute permission
```




S_IRWXO	00007	mask for permissions for others (not in group)
S_IROTH	00004	others have read permission
S_IWOTH	00002	others have write permission
S_IXOTH	00001	others have execute permission

第三行：--- a/README.md。--- 表示该文件是旧文件（存放在暂存区域）

第四行：+++ b/README.md。+++ 表示该文件是新文件（存放在工作区域）

第五行：@@ -0 0,+1 @@。以 @@ 开头和结束，中间的“-”表示旧文件，“+”表示新文件，后边的数字表示“开始行号，显示行数” 一脸懵逼。。。不急我们继续看。

第六行：+这是一个README文件。这是将两个文件合并显示的结果，前边有个+ 的绿色的那一行说明是新文件独有的，浅灰色的则是两个文件所共有的内容。 现在懂了吧，0,+1就是说在第0行开始显示1行。

## 比较两个历史快照

我们先把刚刚的修改提交（对的，git一直是支持中文的😄）：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -am "
改了一下readme"
[master 8e7a488] 改了一下readme
1 file changed, 1 insertion(+)
```

看一下现在的日志：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit 8e7a488e3791982b598084fa6cc9cfc7cb5b5e1f (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 17:51:20 2023 +0800

    改了一下readme

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file
```

现在我们对比一下这两个快照有什么区别，执行git diff 8e7a488 da71e70：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git diff 8e7a488
da71e70
diff --git a/README.md b/README.md
index e360e56..e69de29 100644
--- a/README.md
+++ b/README.md
@@ -1 +0,0 @@
-这是一个README文件
```

## 比较当前工作目录与快照

我们再对readme修改一下不提交：



输入 `git diff 8e7a488`：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git diff 8e7a488
diff --git a/README.md b/README.md
index e360e56..4712474 100644
--- a/README.md
+++ b/README.md
@@ -1,1 @@
-这是一个README文件
+这是一个README文件，我又改了一点
```

## 修改最后一次提交、删除文件、重命名文件

### 删除文件

首先看一下我现在的日志是这样的：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme

commit 8e7a488e3791982b598084fa6cc9cfc7cb5b5e1f
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 17:51:20 2023 +0800

    改了一下readme

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file
```

然后我们把README.md 删了，在看一下状态：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

他说我删掉了这个文件，但暂存区里还有（工作区和暂存区不同是红色的），但这个时候我们肯定不能再用`git add`来把文件加到暂存区了对吧，那怎么办呢？我们执行`git rm 文件名`：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git rm README.md
rm 'README.md'
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   README.md
```

这个时候就把这个文件从暂存区也删掉了，但快照里还没删掉，所以我们再提交一遍即可：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "删除了README.md"
[master 8a10aeb] 删除了README.md
1 file changed, 1 deletion(-)
delete mode 100644 README.md
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
nothing to commit, working tree clean
```

看看现在的日志：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit 8a10aeb65a83efbf84d4939545cd76f4372944b7 (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:08:29 2023 +0800

    删除了README.md

commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme

commit 8e7a488e3791982b598084fa6cc9cfc7cb5b5e1f
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 17:51:20 2023 +0800

    改了一下readme

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file
```

嘿，突然发现README还是需要的，我们还是把它找回来吧



执行前面讲的快照回滚就可以啦：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git reset --hard
8c799f3
HEAD is now at 8c799f3 又改了一下readme
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
nothing to commit, working tree clean
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ ls
README.md
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme

commit 8e7a488e3791982b598084fa6cc9cfc7cb5b5e1f
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 17:51:20 2023 +0800

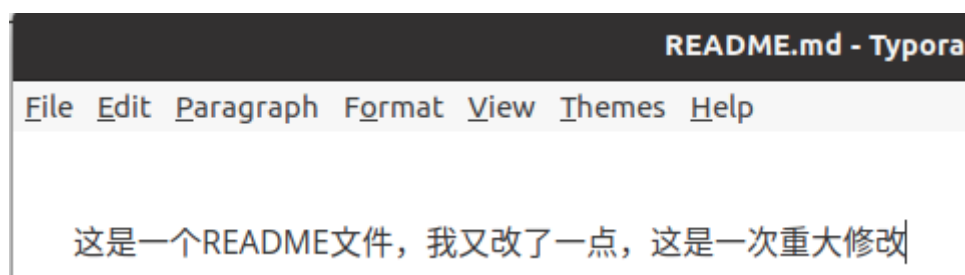
    改了一下readme

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file
```

## 修改最后一次提交

我们再修改一下README文件，然后提交一下：




```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -am "
又又改了一下README"
[master f7ead0d] 又又改了一下README
1 file changed, 1 insertion(+), 1 deletion(-)
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit f7ead0db377e366b5b5ed8cd0e0b6cb6674bc657 (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:21:47 2023 +0800

    又又改了一下README

commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme
```

但这个时候我们发现我们的描述还不合适呀，没有突出“**重大修改**”这个意思，别人不仔细看还以为我在摸鱼呢！所以我们要修改一下这次提交的描述，执行`git commit --amend -m "改了一下`

README，是一次重大修改”：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit --amend -m "改了一下README，是一次重大修改"
[master 70e5440] 改了一下README，是一次重大修改
Date: Thu Aug 3 18:21:47 2023 +0800
1 file changed, 1 insertion(+), 1 deletion(-)
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log
commit 70e5440742f0469fdcdab8f76737c0a5b3014d68 (HEAD -> master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:21:47 2023 +0800

    改了一下README，是一次重大修改

commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme
```

## 重命名文件

我们先把刚刚的README.md文件重命名为README1.md，看一下git状态：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ ls
README1.md
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README1.md

no changes added to commit (use "git add" and/or "git commit -a")
```

git认为我新建了一个README1.md文件而删除了一个README.md文件，那怎么进行重命名呢？我们先换回原来的名字，在 Git 里重命名，需要让 Git 来帮你做，这样它才便于跟踪。执行 **git mv README.md README1.md** 命令：

```
commit 8e7a488e3791982b598084fa6cc9cfc7cb5b5e1f
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 17:51:20 2023 +0800

    改了一下readme

commit da71e700fac826ae53944662574a8311da1b75de
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 01:03:37 2023 +0800

    add a readme file
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
nothing to commit, working tree clean
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ ls
README1.md
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README1.md

no changes added to commit (use "git add" and/or "git commit -a")
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git mv README.md README1.md
```

# 创建和切换分支

## 分支是什么？

假设你的大项目已经上线了（有上百万人在使用呢），过了一段时间你突然觉得应该添加一些新的功能，但是为了保险起见，你肯定不能在当前项目上直接进行开发，这时候你就有另（创）起（建）炉（分）灶（支）的需要了。放个大图先让你知道分支大概是咋回事：



因为对于其它版本控制系统而言，创建分支常常需要完全创建一个源代码目录的副本，项目越大，耗费的时间就越多；而 Git 由于每一个结点都已经是一个完整的项目，所以只需要创建多一个“指针”（像 master）指向分支开始的位置即可。

## 创建分支

创建分支，使用 `git branch 分支名` 命令：

```
chenliangzhi@chenliangzhi-R0G-Strix-G513QM-G513QM:~/MyProject$ git branch branch2
```

没有任何提示说明分支创建成功（一般也不会失败啦，除非创建了同名的分支会提醒你一下），此时可以执行 `git log --decorate` 命令查看：

如果希望以“精简版”的方式显示，可以加上一个 `--oneline` 选项（即 `git log --decorate --oneline`），这样就只用一行来显示一个快照记录。

```
chenliangzhi@chenliangzhi-R0G-Strix-G513QM-G513QM:~/MyProject$ git log --decorate
commit 1b3fcf1b01430e9aa9f5c7e258681de6748b2f5d (HEAD -> master, branch2)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 21:25:31 2023 +0800

    修改README

commit 70e5440742f0469fdcdab8f76737c0a5b3014d68
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:21:47 2023 +0800

    改了一下README，是一次重大修改

commit 8c799f392d20e7d0d85e5cbaf8d2323c70af6eff
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:06:39 2023 +0800

    又改了一下readme
```

可以看到最新的快照后边多了一个 (HEAD -> master, branch2)

它的意思是：目前有两个分支，一个是主分支（master），一个是刚才我们创建的新分支（branch2），然后 HEAD 指针仍然指向默认的 master 分支。



## 切换分支

现在我们需要将工作环境切换到新创建的分支（branch2）上，使用的就是之前我们欲言又止的 checkout 命令。

执行 `git checkout branch2` 命令：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git checkout branch2
Switched to branch 'branch2'
```

什么？！口说无凭！ 好吧，你们要的证据在这里：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --decorate
commit 1b3fcf1b01430e9aa9f5c7e258681de6748b2f5d (HEAD -> branch2, master)
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 21:25:31 2023 +0800

    修改README

commit 70e5440742f0469fdcdab8f76737c0a5b3014d68
Author: 17805818393 <1411904617@qq.com>
Date: Thu Aug 3 18:21:47 2023 +0800

    改了一下README，是一次重大修改
```

然后我们在这个分支中修改一下README文件，再提交一下：



```

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch branch2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add README.md

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "在分支更新README"
[branch2 858fed1] 在分支更新README
 1 file changed, 1 insertion(+), 1 deletion(-)
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --decorate
commit 858fed1371a52b95f41243cd7709ff7d528846b2 (HEAD -> branch2)
Author: 17805818393 <1411904617@qq.com>
Date:   Thu Aug 3 21:31:46 2023 +0800

    在分支更新README

commit 1b3fcf1b01430e9aa9f5c7e258681de6748b2f5d (master)
Author: 17805818393 <1411904617@qq.com>
Date:   Thu Aug 3 21:25:31 2023 +0800

    修改README

```

现在我们切换回主分支，执行 **git checkout master**：

```

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git checkout master
Switched to branch 'master'

```

我们看到README又变回了在分支里更改之前的样子：



然后我们在主分支里对README修改一下再提交：



```

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -am "主分支里更改README"
[master 321012b] 主分支里更改README
 1 file changed, 2 insertions(+)

```



执行 `git log --oneline --decorate --graph --all` 命令查看一下：

--graph 选项表示让 Git 绘制分支图，--all 表示显示所有分支


```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --oneline
--decorate --graph --all
* 321012b (HEAD -> master) 主分支里更改README
| * 858fed1 (branch2) 在分支更新README
|/
* 1b3fcf1 修改README
* 70e5440 改了一下README，是一次重大修改
* 8c799f3 又改了一下readme
* 8e7a488 改了一下readme
* da71e70 add a readme file
```

## 合并和删除分支

### 合并分支

当一个子分支的使命完结之后，它就应该回归到主分支中去。合并分支我们使用 merge 命令，执行 `git merge branch2` 命令，将 branch2 分支合并到 HEAD 所在的分支（master）上：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git merge branch2
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Holy crap!!  这就叫出师不利.....从 Git 提示的内容来看，我们知道这次的合并并没有成功，Git 意

思是说现在你需要先解决冲突的问题，Git 才能进行合并操作。所谓冲突，无非就是像两个分支中存在同名但内容却不同的文件，Git 不知道你要舍弃哪一个或保留哪一个，所以需要你自己来决定。此时执行 `git status` 命令也会显示需要你解决的冲突：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

然后 Git 会在有冲突的文件中加入一些标记，不信你打开 README.md 文件看看：

```
1 这是一个README文件，我又改了一点，这是一次重大修改
2
3 <<<<<<< HEAD
4 11111
5
6 master
7 =====
8 111113243
9 >>>>>>> branch2
```

以“=====”为界，上到“<<<<<<< HEAD”的内容表示当前分支，下到“>>>>>>> branch2”表示待合并的 branch2 分支，之间的内容就是冲突的地方。

现在我们将 README.md 统一修改如下（同时去掉了 <<<<<<< HEAD 等内容）保存文件，然后提交快照：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add README.md
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "解决冲突"
[master 34cda3f] 解决冲突
```

执行 `git log --decorate --all --graph --oneline` 命令，可以看到此时的分支已经自动合并了：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --oneline
--decorate --graph --all
* 34cda3f (HEAD -> master) 解决冲突
| \
| * 858fed1 (branch2) 在分支更新README
* | 321012b 主分支里更改README
| /
* 1b3fcf1 修改README
* 70e5440 改了一下README，是一次重大修改
* 8c799f3 又改了一下readme
* 8e7a488 改了一下readme
* da71e70 add a readme file
```

当然，如果不存在冲突，就不用搞这么多了.....

## 删除分支

对于不再需要的分支，我们还是把它们删了吧！删除分支，使用 `git branch -d 分支名` 命令：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git branch -d branch2
Deleted branch branch2 (was 0205dc3).
```

执行 `git log --decorate --all --graph --oneline` 命令：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --oneline
--decorate --graph --all
* d91b0a5 (HEAD -> master) Merge branch 'branch2'
| \
| * 0205dc3 合并分支3到分支2
| | \
| | * fae44cb 在分支3中更细README
| * | f16066e 在分支2中更新README
| * | 6e30790 在分支3中修改README
| /
* 34cda3f 解决冲突
| \
| * 858fed1 在分支更新README
* | 321012b 主分支里更改README
| /
* 1b3fcf1 修改README
* 70e5440 改了一下README，是一次重大修改
* 8c799f3 又改了一下readme
* 8e7a488 改了一下readme
* da71e70 add a readme file
```

可以看到branch2和branch3都没了（branch3是我自己测试时写的，在文章中没体现出来）。由于 Git 的分支原理实际上只是通过一个指针记载，所以创建和删除分支都几乎是瞬间完成。

注意：如果试图删除未合并的分支，Git 会提示你“该分支未完全合并，如果你确定要删除，请使用 `git branch -D 分支名` 命令”。

## 从历史中拷贝一文件

我们先新建一个文件，并提交上去：



```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git add 这是一个会被删除的测试文件.txt
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "添加一个用来删除的文件"
[master 08b2a8d] 添加一个用来删除的文件
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "\350\277\231\346\230\257\344\270\200\344\270\252\344\274\232\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\226\207\344\273\266.txt"
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --oneline
08b2a8d (HEAD -> master) 添加一个用来删除的文件
0205dc3 合并分支3到分支2
fae44cb 在分支3中更细README
f16066e 在分支2中更新README
6e30790 在分支3中修改README
858fed1 在分支更新README
1b3fcf1 修改README
70e5440 改了一下README，是一次重大修改
8c799f3 又改了一下readme
8e7a488 改了一下readme
da71e70 add a readme file
```

然后我们删除这个文件（删除方法看前面的文件删除栏目）：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "删除了会被删除的文件"
[master c61b289] 删除了会被删除的文件
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 "\350\277\231\346\230\257\344\270\200\344\270\252\344\274\232\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\226\207\344\273\266.txt"
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git log --oneline
c61b289 (HEAD -> master) 删除了会被删除的文件
08b2a8d 添加一个用来删除的文件
0205dc3 合并分支3到分支2
fae44cb 在分支3中更细README
f16066e 在分支2中更新README
6e30790 在分支3中修改README
858fed1 在分支更新README
1b3fcf1 修改README
70e5440 改了一下README，是一次重大修改
8c799f3 又改了一下readme
8e7a488 改了一下readme
da71e70 add a readme file
```

接着我们可以用git diff语句看一下当前工作区与某个快照的区别：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git diff 08b2a8d
diff --git "a/\350\277\231\346\230\257\344\270\200\344\270\252\344\274\232\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\226\207\344\273\266.txt" "b/\350\277\231\346\230\257\344\270\200\344\270\252\344\274\232\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\226\207\344\273\266.txt"
deleted file mode 100644
```

可以看到唯一的区别就是快照里多了那个被我们删掉的文件呢，那我们就用 **git checkout ID 文件名** 把快照里的文件拷贝回来：

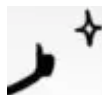
```

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git checkout 08b2
a8d 这是一个会被删除的测试文件.txt
Updated 1 path from 2299b89
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   "\350\277\231\346\230\257\344\270\200\344\270\252\344\274\23
2\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\22
6\207\344\273\266.txt"

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/MyProject$ git commit -m "又
恢复会被删除的文件"
[master d4e2fb8] 又恢复会被删除的文件
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 "\350\277\231\346\230\257\344\270\200\344\270\252\344\274\23
2\350\242\253\345\210\240\351\231\244\347\232\204\346\265\213\350\257\225\346\22
6\207\344\273\266.txt"

```

这样文件夹里可以看到那个被我们删掉的文件又回来啦！



到这里为止我们的本地Git教程就差不多结束了，后面还有连接Github使用的教程！

## 本地仓库连接到Github

### 生成本地秘钥

在终端里执行`ssh-keygen -t rsa -C "youremail@example.com"` 需要将引号部分替换成自己的邮箱（注册Github的）。

在生成过程中，会让你填写想要储存的路径，这里会有一个默认地址，照着输就行了。确定地址后，会提示让你填写passphrase，直接敲Enter跳过即可。完成后，它会提示你密钥和私钥的保存路径位置。在这里会出现一个SHA256加密的fingerprint，可以记一下用于之后的验证。

```

chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ ssh-keygen -t rsa -C "14119
04617@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/chenliangzhi/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/chenliangzhi/.ssh/id_rsa
Your public key has been saved in /home/chenliangzhi/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:KKg/TF9RVvgOYEUI...SAnwL...QcxLLkD4 1411904617@qq.com
The key's randomart image is:
+---[RSA 307]---+
|                 o+*   . . .                 |
|                o  -   . . o                 |
|   +  C_   .   .   .   .   o                 |
| o * . r   .   .   .   +                   |
| .X.c   . .   .   .   .                   |
| +* =  .   .   .   .   .                   |
| Oo.o = . .   .   .   .                   |
| =o + . .   .   .   .                   |
| . .   .   .   .   .   .                   |
+---[SHA256]-----+

```

现在我们已经生成了密钥，然后查看一下公钥，把公钥放到github里就可以了。



分别执行 `cd ~/.ssh`, `ls -a`, `cat id_rsa.pub`：

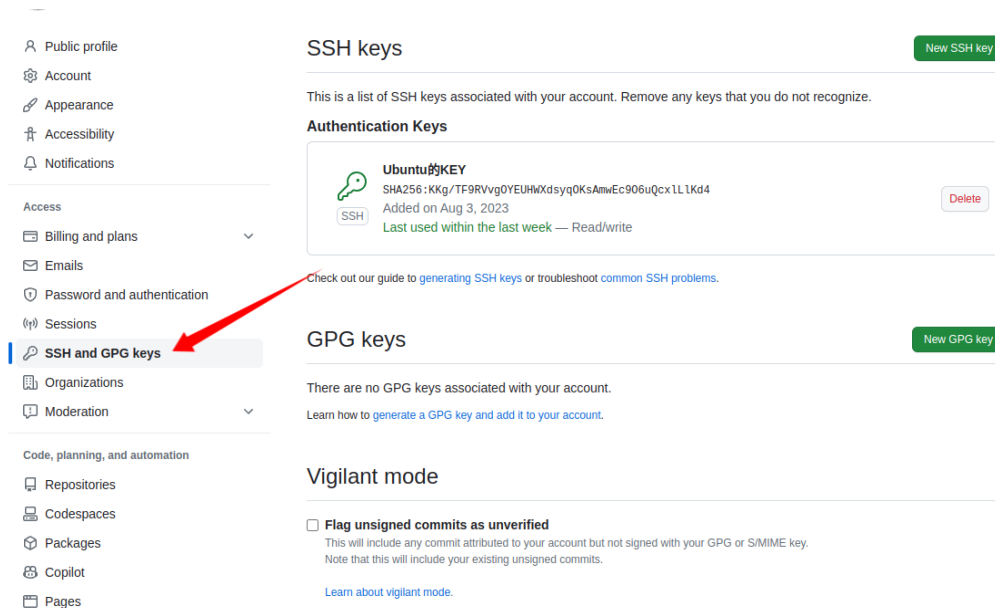
```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~$ cd ~/.ssh
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/.ssh$ ls -a
.  ..  id_rsa  id_rsa.pub
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQDa5oKNwTVL/vQuQfNzM8RQX4dv0LVUyk7DgFLUZxSG
lAfcTl4lKA+7L1GcrHOPv3NJOQ4c/nSIJkc1qzjl0YbpnsPuJ2QSe8kP2HCB7QFyKWeFAahLXKq6+q8R
9npVsaRlnvH6IqYtJPAY3uMxkF7fwduHEhcwro6foBZh4CogxELMm93bjwfU0cmKe57wWM5PVCNK/md
1XIs1Uwv/UY9HM0NMFL20FGFEW0NWXH+hF1k0WuIdUyooLNk5Bpg5hp9RULi8PKCc0IsPnmBjTnrJTdw
Kj5uU9uWsiF8g10X/d5hM5RKVtoFs8GJevK9Tuonqf6C2pEdPvy4HAIjZDG2t2PIHxvDL9XNpEHfFpOb
VL9GdtCwYkwELCqFiMm6oFL0EMmiqwzKz7GiVFXF+t03C0mnHhyD3gL1BqRpNyjWQB/wfr0BLAUv2pHp
nQJ7Jo55wcM7wjEj3igDlKEo/lFT6Szhox2pyI+luZtZzsRUNo7XwNPW6O4JLQ0wVNC4U= 1411904
617@qq.com
```

私钥存在放自己系统里的，只读取公钥（用于身份验证），将“cat id\_rsa.pub”得到的整段结果复制，黏贴到自己的Github中。

到此为止，Linux就可以通过SSH建立本机Git和Github的连接啦（下载与传输）。

## 在github中添加公钥

进入github的setting页面，点击SSH and GPG keys，点击SSH keys右边绿色（New SSH key）的按钮



跳转到新页面将你复制的内容（id\_rsa.pub）粘贴到Key的文本框中，Title可以随便取，点击绿色（Add SSH key）按钮，页面会跳转回之前的界面，如果成功会显示一串字符（SHA256算法加密的身份验证），可以看到这串字符和之前创建ssh时的fingerprint是完全一致的。

Title

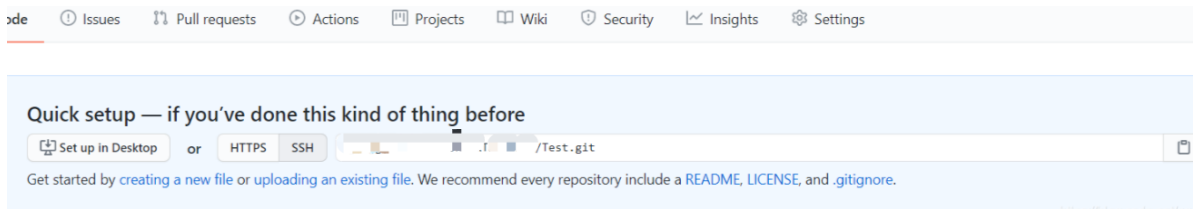
Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQgQDa5oKNwTVL/vQuQfNzM8RQX4dv0LVUyk7DgFLUZxSG
GoWD'
+GJfexijBG0MK7MMK
reobJ7tDv
P5f
```



## 上传本地库文件

在Github上建立一个新的Repository，创建后可以点击选项栏SSH，会出现一个URL地址，把它复制下来



在本地执行 **git remote add origin** 刚刚拷贝的链接（以下图片的链接跟上面图片的链接不一样是因为我用两个仓库写教程，混乱了🤪）：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/RM$ git remote add origin git@github.com:17805818393/PHOENIX_Control.git
```

最后执行 **git push -u origin 分支名** 就能把该分支上传上去啦：

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/RM$ git push -u origin master
Enumerating objects: 59, done.
Counting objects: 100% (59/59), done.
Delta compression using up to 16 threads
Compressing objects: 100% (58/58), done.
Writing objects: 100% (59/59), 637.16 KiB | 2.05 MiB/s, done.
Total 59 (delta 0), reused 0 (delta 0)
To github.com:17805818393/PHOENIX_Control.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

如果本地仓库修改了一下，而分支已经提交过了，那就直接 **git push** 就能上传了。

```
chenliangzhi@chenliangzhi-ROG-Strix-G513QM-G513QM:~/RM$ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 16 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (12/12), 3.57 MiB | 462.00 KiB/s, done.
Total 12 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To github.com:17805818393/PHOENIX_Control.git
 d4717f2..1cb0c03  master -> master
```

至此Git的所有教程就结束啦，上面讲的只是我略懂的皮毛，Git的水是很深的，在使用的过程中也会遇到其他上面未提到的问题，那就要靠你们勤劳的双手自己百度咯！🤪 感谢任何一个努力学习的你！🙏

## Git的基础命令

- 1、git init（初始化git仓库，将目录指到要做仓库的文件夹后执行即可，会在文件夹生成一个.git文件夹）。
- 2、git config --global user.name "your name"
- 3、git config --global user.email "your email"

- 4、git config --list
- 5、git add 文件
- 6、git commit -m "描述"
- 7、git checkout （命令将暂存区域的旧版本覆盖工作目录的新版本（危险操作：相当于丢弃工作目录的修改））
- 8、git reset HEAD （恢复暂存区）
- 9、git commit -am "提交描述"（将add和commit操作合并）
- 10、git reset HEAD~ （回滚到上一个快照到暂存区，--soft 只回滚快照，不将仓库的文件放到暂存区，--hard 将快照回滚动到暂存区域，它还将暂存区域的文件还原到工作目录）
- 11、git reset ID （回滚指定快照）
- 12、git diff （比较暂存区域与工作目录的文件内容）
- 13、git diff ID1 ID2 （比较两个快照的内容）
- 14、git diff ID1 （比较工作区与指定快照的内容）
- 15、git rm 文件名 （删除暂存区的文件）
- 16、git commit --amend -m "描述" （修改最后一次提交的描述）
- 17、git mv 旧文件名 新文件名 （重命名文件）
- 18、git branch 分支名 （创建分支）
- 19、git checkout 分支名 （切换分支）
- 20、git log --oneline --decorate --graph --all （绘制分支图的日志）
- 21、git merge 分支名 （把分支名合并到当前所在的分支）
- 22、git branch -d 分支名 （删除分支）
- 23、git checkout ID 文件名 （拷贝快照中的文件）
- 24、git remote add origin 仓库链接 （本地仓库关联到云端仓库）
- 25、git push -u origin 分支名 （将本地的分支上传到云端上）