

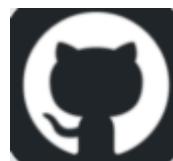
# Github以及Git的普识

---

## 一、通识

这是**Github**，它是

**基于Web的代码托管平台**



---

这是**Git**，它是

**一个开源的分布式版本控制系统**



- 所以，我们为什么要用github以及git？

现在假设一个情景，我们在自己电脑的小熊猫上写好了一个.c的代码程序，

为了方便在机房使用，你复制了一份到U盘。

你上机课的时候发现原来的代码在机房的dev C++跑不通，但是在你自己电脑上是跑的通的，

于是你改了一下代码，把这个.c文件命名为代码2，并且又复制一份放到了U盘。

这个U盘相当于**仓库**，

而现在你想把这个代码分享给朋友，但是一个人一个人地借U盘似乎有点太麻烦，

于是你把你的“U盘”挂到了网上，并且允许其他人也上传自己的“U盘”，

大家就能在网上共享代码，互相帮助互相学习。

而每个人的“U盘”(仓库)都放在网上集成在一起的地方，就是我们的**github**。

这种开放自己的**仓库**、供大家学习使用的行为也就是**开源**。

github也是**基于Web的代码托管平台**。

---

但是，仅仅如此，似乎百度网盘也能做到。那为什么我们要用github来分享代码，而不用百度网盘呢？

刚才的例子，我们是用 U 盘和文件来回拷贝。如果只是“代码1.c”、“代码2.c”这样简单命名，勉强还能应付。

但如果你改了很多次，变成了“代码最终版.c”、“代码真的最终版.c”、“代码再也不改版.c”……时间一长，你自己都分不清哪个是哪个，更别提想找回三天前能运行的那个版本了。

**Git** 就是来解决这个问题的。

它不仅能帮你把代码“备份”到仓库，还可以让你清晰地知道：

**谁，在什么时间，改了哪几行代码。**

并且每一次修改都会保存。

在**Git**的帮助下，你也可以将不同人的、多次的“修改”合并为同一个版本。

或者在可以跑的版本的基础上，建立一个分支，在分支上进行修改，以确保不会影响到可以跑的版本。

通过**Git**，我们成功实现了版本之间的控制。

而**Git**允许不同的人都可以参与进来，所以也就实现了**分布式**版本控制。

## 二、基础的上传和下载

大概了解了**Git**以及**Github**的概念后，我们来学习，如何实现传到网上，以及如何从网上传下来：

此处介绍最原始的命令行操作形式，后续再介绍图形化管理软件

### 1. 如何把自己的代码传到网上

(1) 在**github**上创建一个仓库

主页的左上角



或者右上角：



在设置好合法的仓库名字后，按需勾选 配置：

自述文件即README.md，是每个项目中的介绍文档

.gitignore 则用于隐藏一些文件，不上传到github上，比如临时文件，配置文件等等（因为可能带有密钥等等），可以自选模板（如果你是写Python的，.gitignore 会帮你排除 \_\_pycache\_\_ 文件夹；如果你是做前端的，会排除 node\_modules）

许可证则是法律文件，用于定义一些规则，常用的有：

**MIT许可证**：最宽松，别人可以随便用，只需要保留你的版权声明

Apache License 2.0

GNU General Public License v3.0

MIT License

**GPL许可证**：要求使用者如果修改你的代码，也必须开源

✓ 无

Apache License 2.0

GNU General Public License v3.0

MIT License

## 2 配置

选择可见性 \* 公共

添加自述文件 关

添加 .gitignore 文件 无

添加许可证 无

## (2) 对本地进行初始化

创建好后，我们能看到这个：

The screenshot shows the GitHub repository creation interface. It includes sections for configuration (visibility, README, .gitignore, license), code space (create workspace), collaboration (invite collaborators), and initialization (fast install via desktop app or command line, or from command line).

**使用代码空间开始编程**  
添加 README 文件并在安全、可配置和专用的开发环境中开始编码。  
[创建代码空间](#)

**将协作者添加到此仓库**  
使用 GitHub 用户名或电子邮件地址搜索人员。  
[邀请协作者](#)

**快速安装 - 如果您以前做过这样的事**  
安装到 GitHub Desktop 或 HTTPS SSH git@github.com:qwsaszl/PCA\_random\_forest\_25samples.git  
通过[创建一个新文件](#)或[上传一个现有的文件](#)来开始。我们推荐每个仓库都包括[自述文件](#), [LICENSE](#), 和[.gitignore](#)。

**...或在命令行上创建一个新的仓库**  

```
echo "# PCA_random_forest_25samples" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:qwsaszl/PCA_random_forest_25samples.git
git push -u origin main
```

**...或从命令行中推送现有的仓库**  

```
git remote add origin git@github.com:qwsaszl/PCA_random_forest_25samples.git
git branch -M main
git push -u origin main
```

这里分为三块：

**左上角：**在云端开发，相当于github给你一个浏览器里运行的VScode，比较适合你想要快速编程的情况。

但是我们的目的是，把本地的项目/文件传到github，所以暂时不考虑这个

**右上角：**纯字面意思，不过显然也不是我们现在需要的

**下方：**这是我们需要的

## ...或在命令行上创建一个新的仓库

```
echo "# PCA_random_forest_25samples" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:qwsaszl/PCA_random_forest_25samples.git
git push -u origin main
```

可能有的同学会疑惑，我们不是已经在github上新建了仓库吗？为什么这里还叫我们创建一个新的仓库？

这里需要厘清两个概念：

### 远程仓库和本地仓库

远程仓库，就是我们在github上创建的仓库，已经创建。

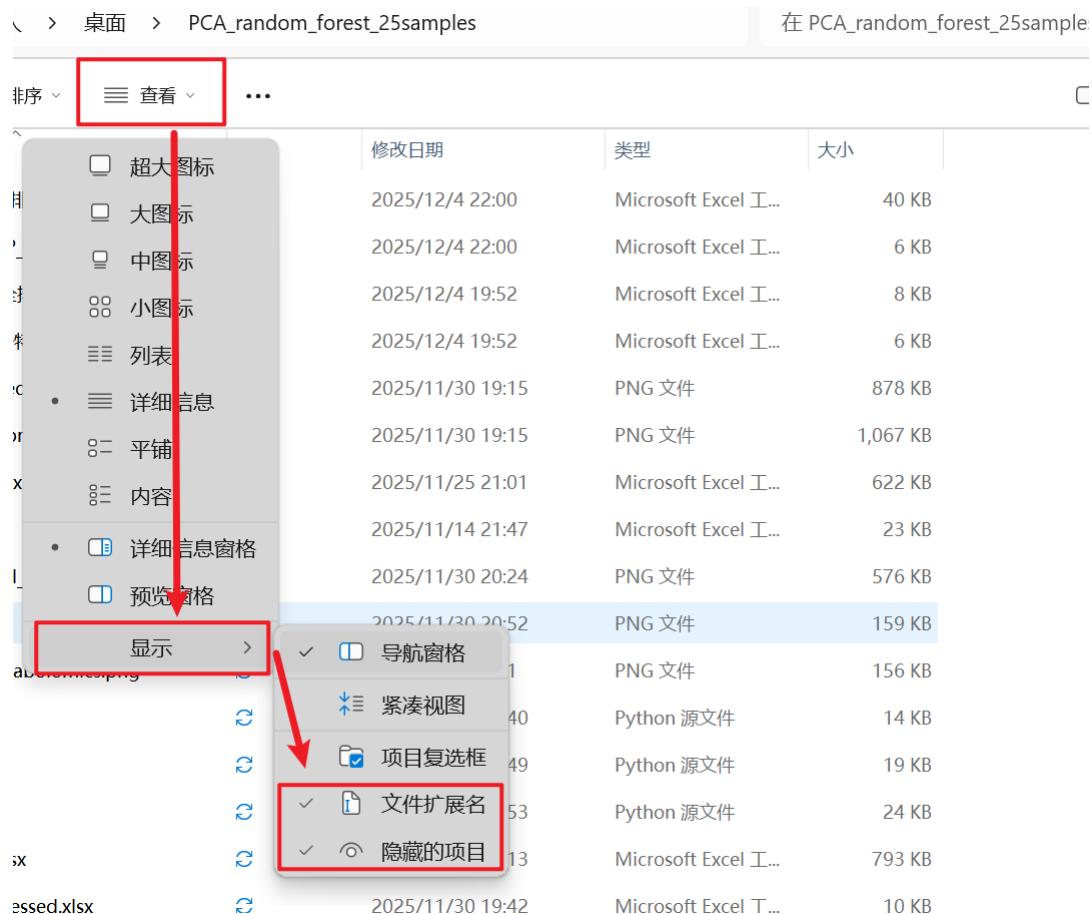
此处叫我们创建的是**本地仓库**。

我们想要传到云端的项目，在此时还只是一个文件夹，你需要先告诉Git，这个是我需要传的项目。

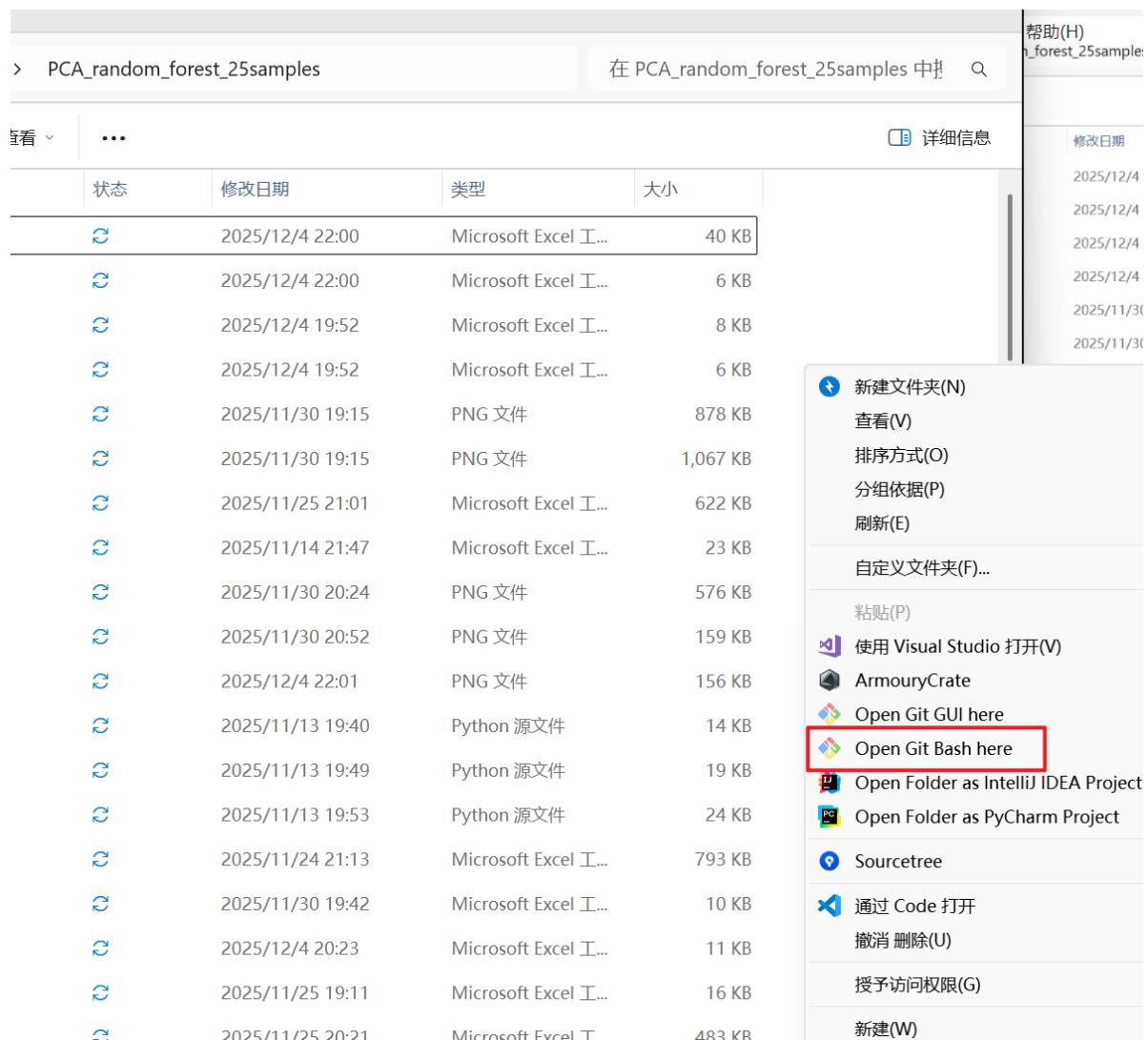
也就是**初始化本地仓库**。

首先，先打开你的项目文件夹，

在这里我们先做一个基本的配置：



在打开的项目文件夹的空白处，右键，选择这个



在这里，我们就能在这个文件夹下，打开命令行（Bash）了

在Windows系统上，命令行为cmd，但是Git最初是为Linux开发的，所以Git的命令行就是Linux的命令行 Bash。

```
MINGW64:/c/Users/14394/OneDrive/桌面/PCA_random_forest_25samples
14394@cloudback MINGW64 ~/OneDrive/桌面/PCA_random_forest_25samples
$
```

在该页面，我们把刚刚那一串东西复制粘贴过来，即：

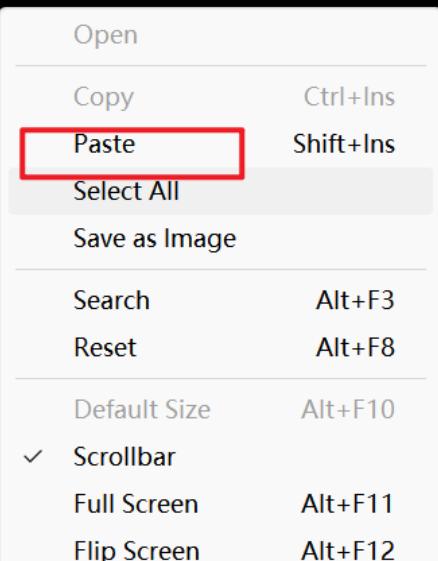
...或在命令行上创建一个新的仓库

```
echo "# PCA_random_forest_25samples" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:qwsasz1/PCA_random_forest_25samples.git
git push -u origin main
```



回到Bash，右键，选择Paste：

```
14394@cloudback MINGW64 ~/OneDrive/桌面/PCA_random_forest_25samples
$ |
```



再按回车。

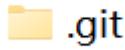
【注意，此处bash内不可以Ctrl+V粘贴，也不可以Ctrl+C复制，毕竟bash不是Windows的命令行】

出现下面这些就意味着成功的基础的步骤：

```
echo "# PCA_random_forest_25samples" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:qwsasz1/PCA_random_forest_25samples.git
git push -u origin main
Initialized empty Git repository in C:/Users/14394/OneDrive/桌面/PCA_random_forest_25samples/.git/
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
[main (root-commit) db6ed5d] first commit
 1 file changed, 1 insertion(+)
  create mode 100644 README.md
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 243 bytes | 243.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:qwsasz1/PCA_random_forest_25samples.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

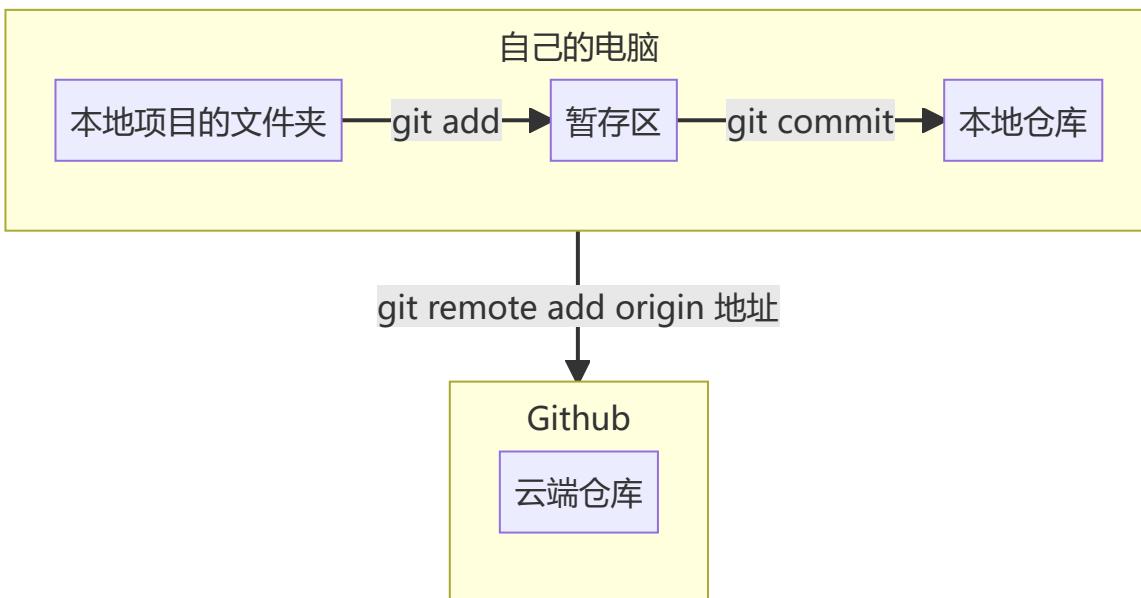
下面来依次介绍，这些命令的含义：

```
1 echo "# PCA_random_forest_25samples" >> README.md  
2 #在PCA_random_forest_25samples文件夹下，创建一个文件README.md  
3  
4 git init  
5 #初始化(initialize)本地仓库，成功后，会在我们项目文件夹下出现这个（只有开启文件管理器 查看->显示->隐藏的项目 才能看见）：
```



```
1 git add README.md  
2 #将刚刚创建的README.md添加到暂存区
```

看到这里，可能又有疑惑：暂存区又是什么？



暂存区 (stage area)，是我们提交到本地仓库前的一个缓存区域。

当然，除了指令中的README.md以外，我们也可以把当前文件夹下其他文件也提交到暂存区。

```
1 git add 文件名.后缀 #提交某一个文件  
2 git add . #提交该文件夹下所有文件
```

在把所有提交完毕后，我们再把暂存区的东西，提交到本地仓库：

```
1 git commit -m "first commit" #将暂存区所有文件一起保存到本地仓库，并且为这次提交添加备注first commit
```

在这次保存到本地仓库时，同时也会记录谁，**什么时候提交了什么**。

当你下一次commit时，就是下一个版本（谁，在另一时候，提交了什么）的事情了。

---

我们可以commit多次，这样就有多个“版本”了，

但是仅仅按照上面这个流程图来走，这些版本之间还是**没有关联、离散的**。

我们如何构建这些“版本”之间的联系呢？

这就是下面这种指令做的事情了，当然，关于这一块有专门的讲解，我们放在后面讲，

```
1 | git branch -M main # 如果只是初次提交到github, 仅这条就够了, 意思是: 强制将当前分支重命名为 main
```

### (3) 将本地仓库和远程仓库建立连接

在通过git branch等指令、创建好commit的版本与版本之间的关系后，我们可以：

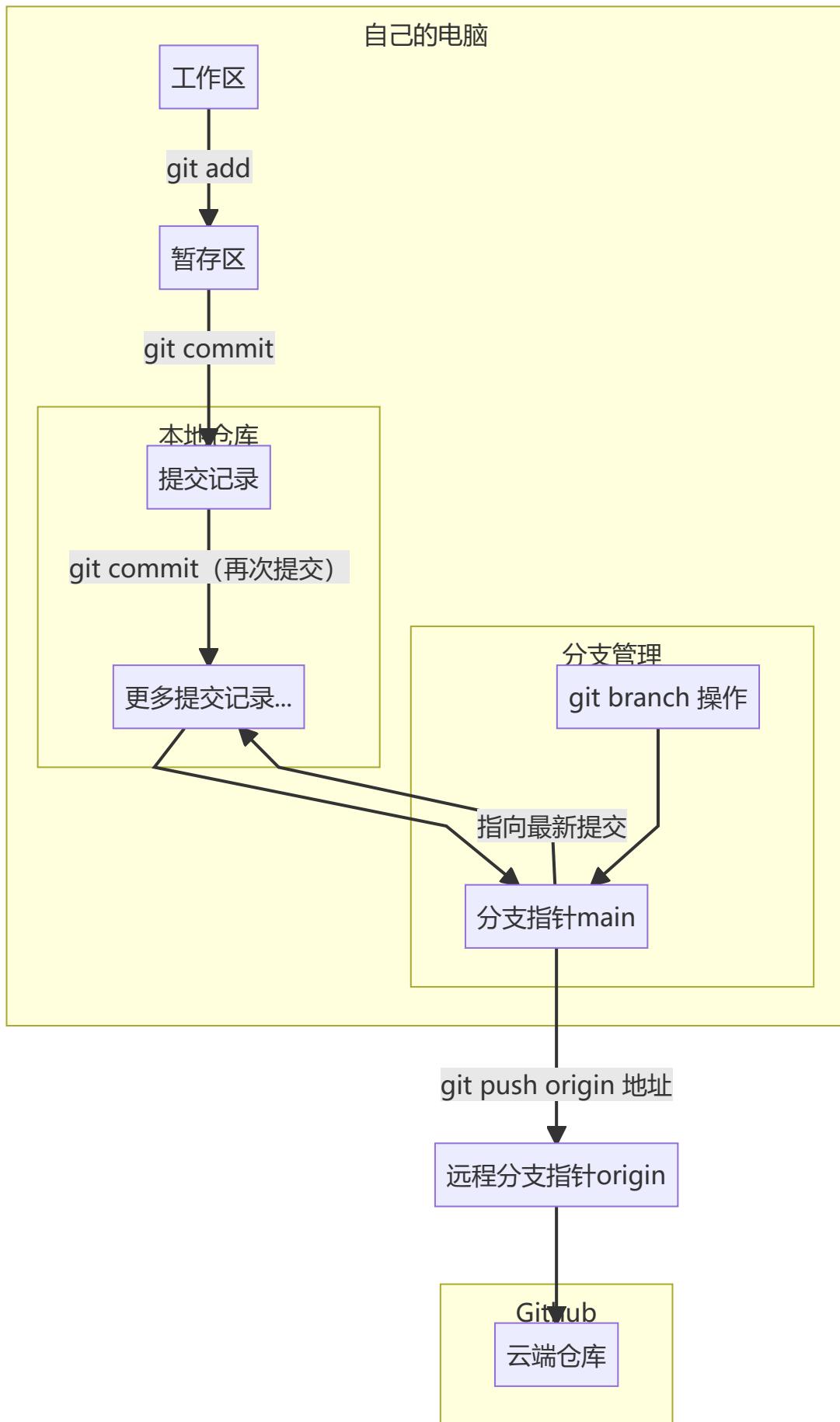
```
1 | git remote add origin git@github.com:qwsaszl/PCA_random_forest_25samples.git
```

它的意思是：告诉本地仓库，我现在要添加（add）一个与远程仓库（[git@github.com](mailto:git@github.com):qwsaszl/PCA\_random\_forest\_25samples.git，即地址）的连接了。

由于这个地址（[git@github.com](mailto:git@github.com):qwsaszl/PCA\_random\_forest\_25samples.git）过长，在该**本地仓库**内，我们为这一长串地址取了个别名 `origin`，用于代指这一长串地址。

提问：对于git remote add origin 地址，

该指令是发给谁的？是自己的电脑？还是Github？



答案：发给自己电脑的。

```
1 # 你执行:  
2 git remote add origin git@github.com:qwsasz1/PCA_random_forest_25samples.git  
3  
4 # Git 在本地做的:  
5 1. 打开当前项目的 .git/config 文件，就是我们一开始git init创建的那个隐藏的文件夹，里面的 config  
6 2. 添加一段配置:  
7 [remote "origin"]  
8     url = git@github.com:qwsasz1/PCA_random_forest_25samples.git  
9     fetch = +refs/heads/*:refs/remotes/origin/*  
10 3. 保存，完成
```

所以这个origin，只是你自己创建的这个本地仓库，给地址取的一个别名。

- 关于地址，我们有两种形式：

SSH:

The screenshot shows the GitHub Desktop interface for creating a new repository. At the top, there's a section titled '快速安装 - 如果您以前做过这样的事' (Quick Setup - If you've done something like this before). It has three options: '安装到 GitHub Desktop' (Install to GitHub Desktop), '或' (Or), and 'SSH'. The 'SSH' option is highlighted with a red circle and a red arrow points from it to the command-line section below. A text input field contains the URL 'git@github.com:qwsasz1/testteach.git'. Below this, instructions say '通过 创建一个新文件 或 上传一个现有的文件 来开始。我们推荐每个仓库都包括 自述文件, LICENSE, 和 .gitignore.' (Through Create a new file or Upload an existing file to start. We recommend including a README file, LICENSE, and .gitignore in each repository.)

...或在命令行上创建一个新的仓库

```
echo "# testteach" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin git@github.com:qwsasz1/testteach.git  
git push -u origin main
```

HTTPS:

This screenshot is similar to the previous one but for HTTPS. The 'SSH' button is now highlighted with a red circle and a red arrow points from it to the command-line section below. The URL input field now contains 'https://github.com/qwsasz1/testteach.git'. The rest of the interface is identical to the SSH version.

...或在命令行上创建一个新的仓库

```
echo "# testteach" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/qwsasz1/testteach.git  
git push -u origin main
```

由于网络环境差异，SSH 连接方式在某些情况下可能比 HTTPS 更稳定，在后续使用中大家可能也可以感受到，

所以我们在最开始建议大家把ssh配置好，以便于以后使用Github。

#### (4) 建立连接后，把本地仓库的这个版本上传到云端：

```
1 | git push -u origin main
```

- git push: 意为推送
- -u是 set-upstream (设置上游)
- 在此处, 将我们的main支和origin (即长串地址) 连接起来了, 并且实行了push (推送)

至此, 我们便完成了初步的上传 (即新建仓库教程中的README.md的上传)

如果需要后续的上传:

- 在已有版本上更新

```
1 git branch #显示一下目前在什么分支下, 是不是自己想要储存的本地仓库对应的分支  
2  
3 git add . #将文件夹所有东西放在暂存区  
4 git commit -m "这里放关于这一次提交, 你的备注"  
5 git push
```

- 创建新的版本

```
1 git checkout -b 新分支名 #创建并切换到分支  
2  
3 git add .  
4 git commit -m "本次修改的备注"  
5  
6 git push -u origin 新分支名
```

## 2. 如何把网上的代码下载到本地

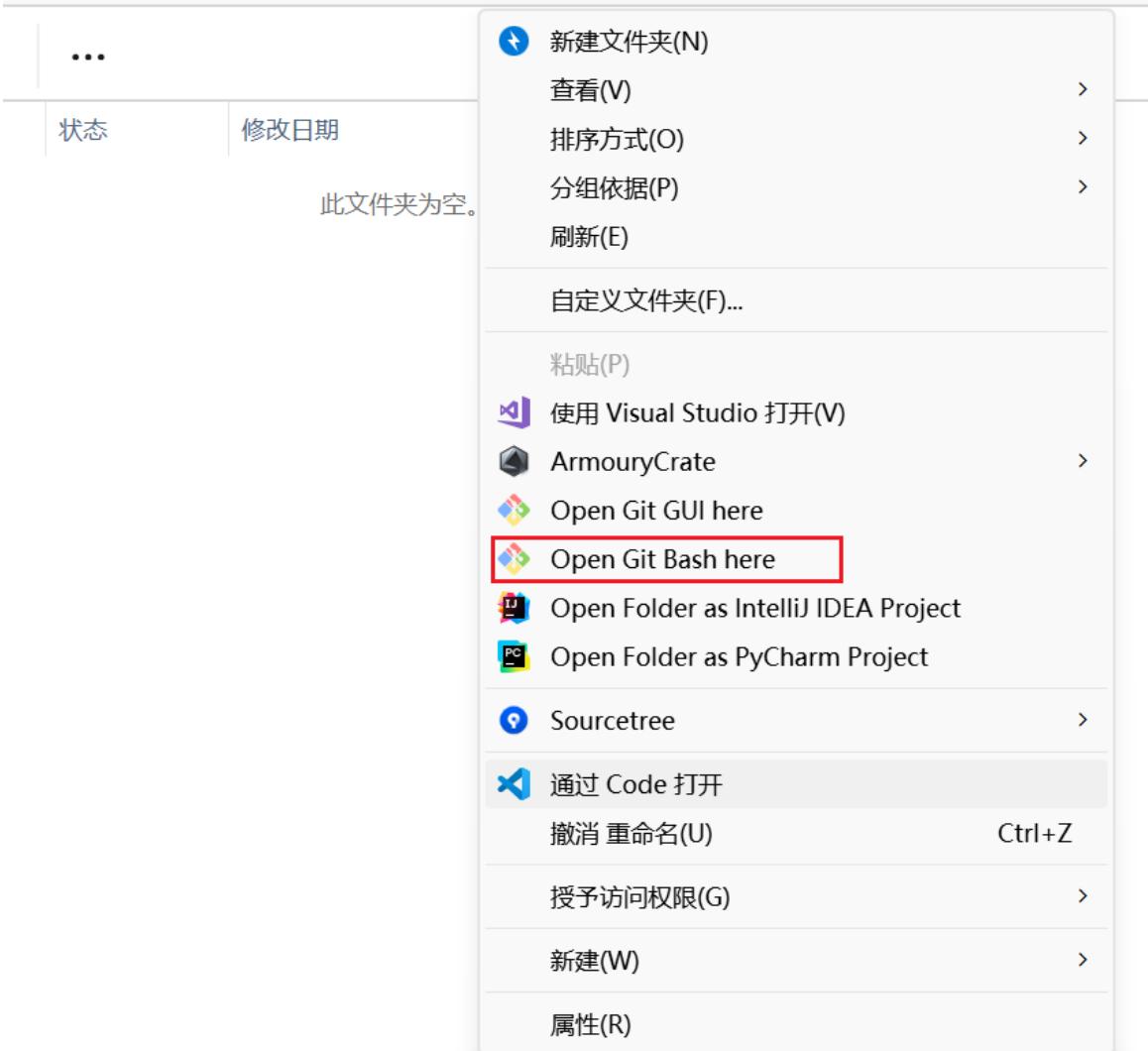
- 直接下载zip压缩包最简单,



- 或者使用指令, 在电脑自己创建一个文件夹, 在空白处右键, 打开Bash

新建文件夹 (2)

在 新建文件夹 (2) 中搜索



输入git clone (克隆) +地址，地址为ssh或者https。

1 | git clone 地址(自己复制自己的地址)



按回车

就能下载到本地了。

以上两种方法，都把所有代码（源代码）打包到本地了，但有的时候我们并不需要下载最新的代码，我们只想要使用里面的部分功能，

只需要下载发行版（即release）就行了

添加文件
代码

object.toml ✓ 251955f · 昨天 4,637 次提交

add extensions in devcontainer.json 7个月前

feat: 更新 Dockerfile，添加 git 安装步骤以支持插件安装 2周前

feat:更新readme和更新日志 3天前

将文件全部归进docs 7个月前

rename: 改一下docs名称方便更新文档 2周前

log:修改一些log 3个月前

feat: 优化任务调度和插件管理，支持路径规范化及插件 ID 自... 2周前

Ruff fix 2周前

fix:ruff 3天前

feat: 统一对task中过慢的模型进行警告，并在model\_config.t... 5天前

WebUI 0.11.6 3天前

test(docker): 分段构建 上个月

更新 .gitattributes 文件，添加 webui/dist/\*\* 为二进制文件 2周前

feat: 记忆查询能力提升 上周

添加 .pre-commit-config.yaml (这并不会启用hook) 9个月前

feat: 添加贡献者契约行为准则 6个月前

自述文件

GPL-3.0 许可证

行为准则

活动

自定义属性

3.8k 星标

18 关注

419 复刻

举报仓库

**发行版 26**

0.11.6 最新 3 days ago

+ 25 个发行版

**软件包**

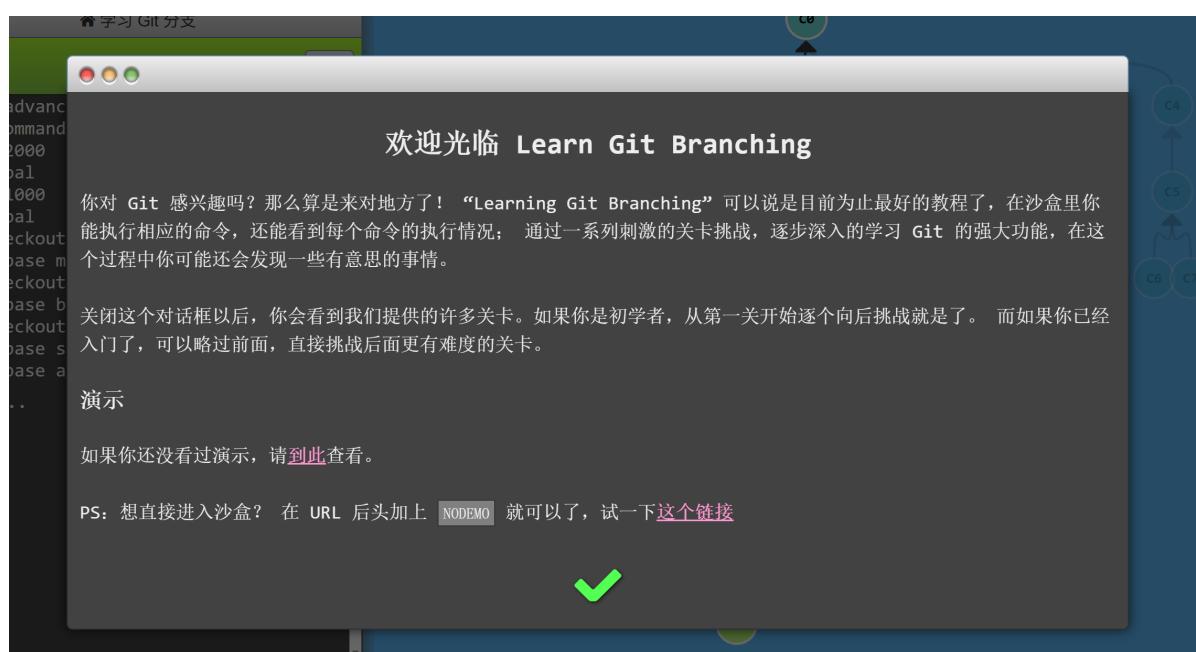
未发布软件包

### 三、如何管理各个版本的关系

这里给大家推荐一个可视化Git教学的平台

[Learn Git Branching](#)

【等它演示动画播放完毕后，会弹出基本介绍，点绿色的√，就可以选择关卡了】

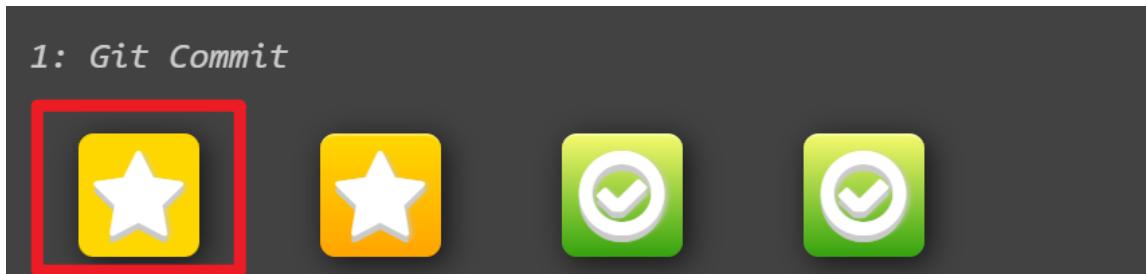


进阶内容大家课后掌握，课上我们主要介绍基础内容：

## 1.git commit

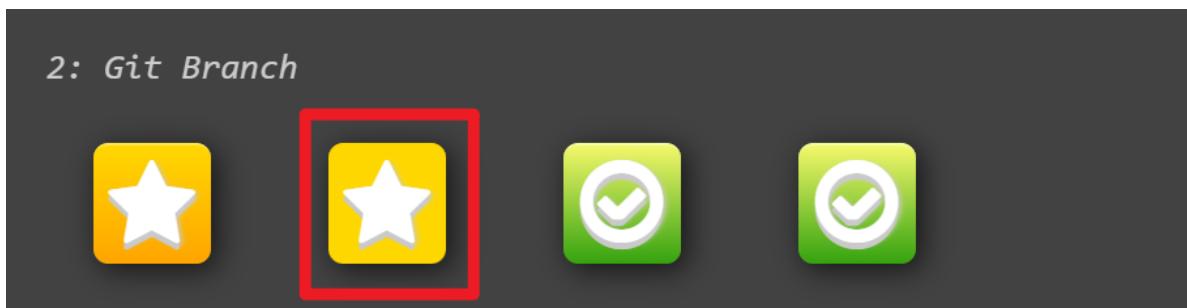
提交

这个在前面已经讲解了，大家可以自己实操一下



## 2.git branch

分支



- 什么是分支？

字面意思，我们可以简单地理解为分叉路，

在main分支上，我们发布稳定版本，

而在main分支中延伸出来的其他分支上（比如你建立了一个develop分支），

我们在这条分支下进行修改和提交，等修改整合完毕后，在merge（合并）到主分支（main）上。

---

查看当前所有分支

```
1 | git branch
2 | # 显示:
3 | # * main
4 | #   a/login
5 | #   a/search
6 | # (* 表示当前所在分支)
```

当你确保自己在稳定的main分支下后，再git branch，才能在main的基础上创建分支。

创建分支：

```
1 | git branch 分支名
```

但如果发现自己不在main上呢？

比如

```
1 | git branch
2 | # 显示:
3 | #   main
4 | # * a/login
5 | #   a/search
6 | # (* 表示当前所在分支)
```

那我们就切换到main分支

```
1 | git checkout main
2 |
3 | #通过checkout, 我们可以切换到任意分支, 即:
4 | #git checkout 分支名
```

再创建一个develop分支, 以后的新功能的添加尝试就可以在develop上进行了

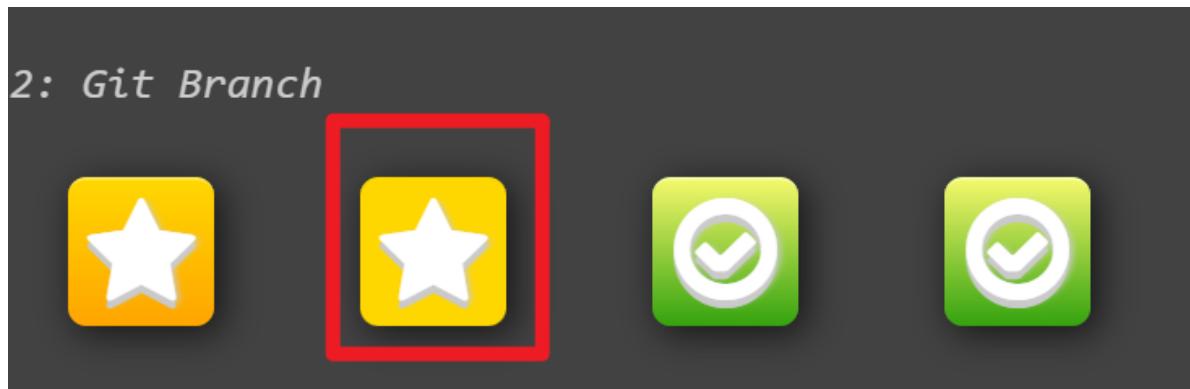
```
1 | git branch develop
```

当然, 你以后要在develop分支上提交(commit)东西, 也需要先确保你在develop下, 才能提交

```
1 | git checkout develop #切换到develop
2 | git commit -m "你的备注"
```

现在大家可以去尝试一下了:

【如果不知道怎么进入关卡选择页面, 在输入栏, 输入levels即可】



### 3.git merge

合并

比如说, 我们在分支develop上已经开发完毕, 下一个版本想加的功能都加好了, 并且发现也没有什么问题, 现在我需要发布这个版本了, 就可以通过git merge, 把develop的内容合并到main分支。

```
1 | git merge develop
2 | #意为, 把develop合并到本分支下
```

把develop合并到**本分支**

这意味着, 你得先checkout到main下, 才能通过merge, 把develop分支合并到main中。

不知道自己在什么分支下, 可以输入

```
1 | git branch
```

就能显示本地的所有分支以及自己在哪个分支下了。

然后再checkout到main,

最后再git merge develop。

了解基本操作后，大家到这里实操一下。

【输入levels跳转到关卡选择】

## 循序渐进地介绍 Git 主要命令

### 3: Git Merge



## 【扩展部分】4.git rebase

由于对于初学者来说可能会混淆，所以这一块作为扩展部分，感兴趣的同學可以尝试一下

简单来说，git rebase可以说是嫁接。

主要分为两步：

1. 复制某个分支（比如说develop分支）
2. 把复制的分支接到你想嫁接的分支下（比如说，把develop嫁接到main）

比如，我们要把develop最新的那个版本嫁接到main时：

对于1：

先确保你在你想要嫁接的分支下（确保你在develop）

```
1 | git checkout develop
```

再复制分支过去

```
1 | git rebase main #意味着你把develop最新的点 复制了一个副本 打算接到main上
```

现在我们需要接到main：

```
1 | #切换到main分支下:  
2 | git checkout main  
3 |  
4 | #再接上刚刚复制的副本  
5 | git rebase develop
```

就实现了rebase。

大家可以去实操一下【依旧是输入levels，打开关卡选择页面】

