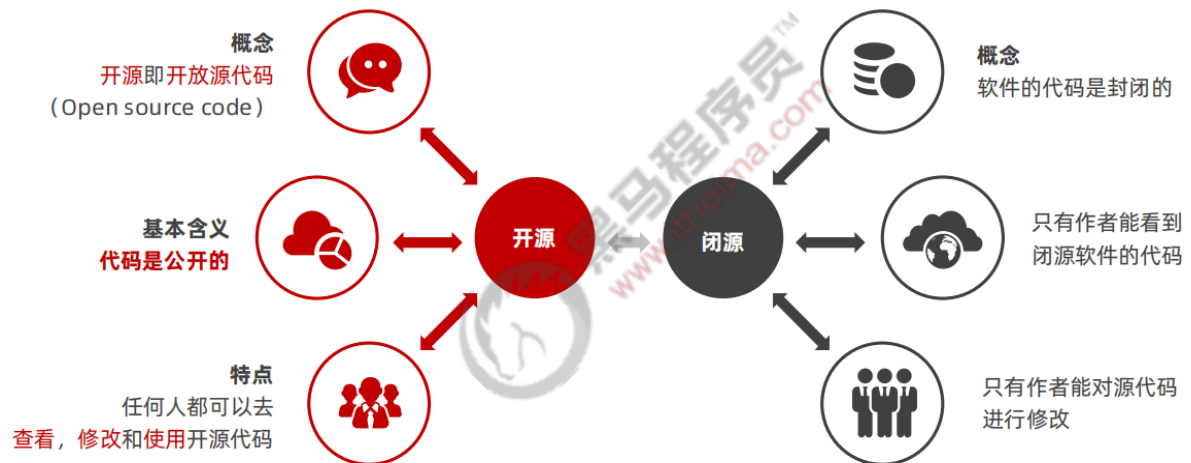


# 目标

- 能够使用 `Github` 创建和维护远程仓库
- 能够掌握 `Git` 分支的基本使用

## 了解开源相关的概念

### 什么是开源



#### 通俗的理解

- **开源**是指不仅提供程序还提供程序的源代码
- **闭源**是只提供程序, 不提供源代码

### 什么是开源许可协议

开源并不意味着完全没有限制, 为了**限制使用者的使用范围**和**保护作者的权利**, 每个开源项目都应该遵守开源

许可协议 ( `Open Source License` ) 。

#### 常见的 5 种开源许可协议

- `BSD` (Berkeley Software Distribution)
- `Apache Licence 2.0`
- `GPL` (GNU General Public License) (☆☆☆)
- 具有传染性的一种开源协议, 不允许修改后和衍生的代码做为闭源的商业软件发布和销售
  - 使用 `GPL` 的最著名的软件项目是: `Linux`
- `LGPL` (GNU Lesser General Public License)
- `MIT` (Massachusetts Institute of Technology, MIT) (☆☆☆)
- 是目前限制最少的协议, 唯一的条件: 在修改后的代码或者发行包中, 必须包含原作者的许可信息
- 使用 MIT 的软件项目有: `jquery`、`Node.js`

# 为什么要拥抱开源

---

开源的核心思想是“我为人人，人人为我”，人们越来越喜欢开源大致是出于以下 3 个原因：

- ① 开源给使用者更多的控制权
- ② 开源让学习变得容易
- ③ 开源才有真正的安全

开源是软件开发领域的大趋势，**拥抱开源就像站在了巨人的肩膀上**，不用自己重复造轮子，让开发越来越容易

## 开源项目托管平台

---

专门用于免费存放开源项目源代码的网站，叫做**开源项目托管平台**。目前世界上比较出名的开源项目托管平台

主要有以下 3 个：

- **Github**（全球最牛的开源项目托管平台，没有之一）
- **Gitlab**（对代码私有性支持较好，因此企业用户较多）
- **Gitee**（又叫做码云，是国产的开源项目托管平台。访问速度快、纯中文界面、使用友好）

**注意：**以上 3 个开源项目托管平台，只能托管以 **Git** 管理的项目源代码，因此，它们的名字都以 **Git** 开头

## Github

---

### 什么是 Github

---

**Github** 是全球最大的**开源项目**托管平台。因为只支持 **Git** 作为唯一的版本控制工具，故名 **GitHub**。

在 **Github** 中，你可以：

- ① 关注自己喜欢的开源项目，为其点赞打 **call**
- ② 为自己喜欢的开源项目做贡献（**Pull Request**）
- ③ 和开源项目的作者讨论 Bug 和提需求（**Issues**）
- ④ 把喜欢的项目复制一份作为自己的项目进行修改（**Fork**）
- ⑤ 创建属于自己的开源项目
- ⑥ etc...

So, **Github** ≠ **Git**

## 注册

---

### 注册 Github 账号的流程

- ① 访问 **Github** 的官网首页 <https://github.com/>
- ② 点击“**Sign up**”按钮跳转到注册页面

- ③ 填写可用的用户名、邮箱、密码
- ④ 通过点击箭头的形式，将验证图片摆正
- ⑤ 点击“Create account”按钮注册新用户
- ⑥ 登录到第三步填写的邮箱中，点击激活链接，完成注册

## Create your account

Username \*

Email address \*

Password \*


Make sure it's **at least 15 characters** OR **at least 8 characters including a number and a lowercase letter**. [Learn more](#).

Email preferences


☐ Send me occasional product updates, announcements, and offers.

Verify your account

点击箭头，滚动图片



## 激活 Github 账号




### Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.

An email containing verification instructions was sent to `liu@itcast.cn`.

[Resend verification email](#) [Change your email settings](#)



Almost done, **@teacher-liu!** To complete your GitHub sign up, we just need to verify your email address: `liu@itcast.cn`.

[Verify email address](#)

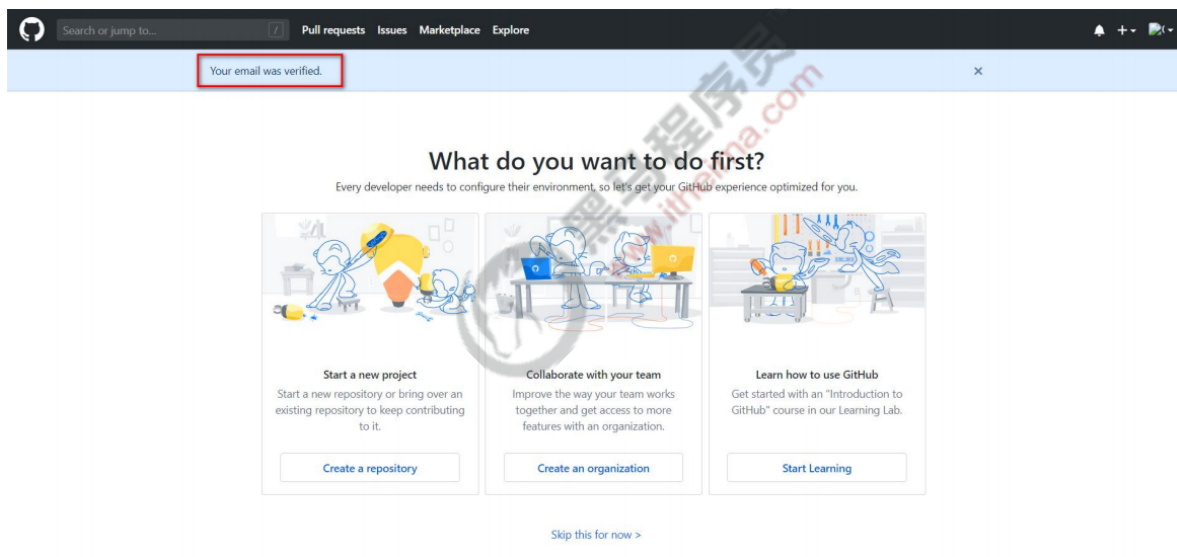
Once verified, you can start using all of GitHub's features to explore, build, and share projects.

Button not working? Paste the following link into your browser: [https://github.com/users/teacher-liu/emails/103277464/confirm\\_verification?token=12ca3b0](https://github.com/users/teacher-liu/emails/103277464/confirm_verification?token=12ca3b0)

You're receiving this email because you recently created a new GitHub account or added a new email address. If this wasn't you, please ignore this email.

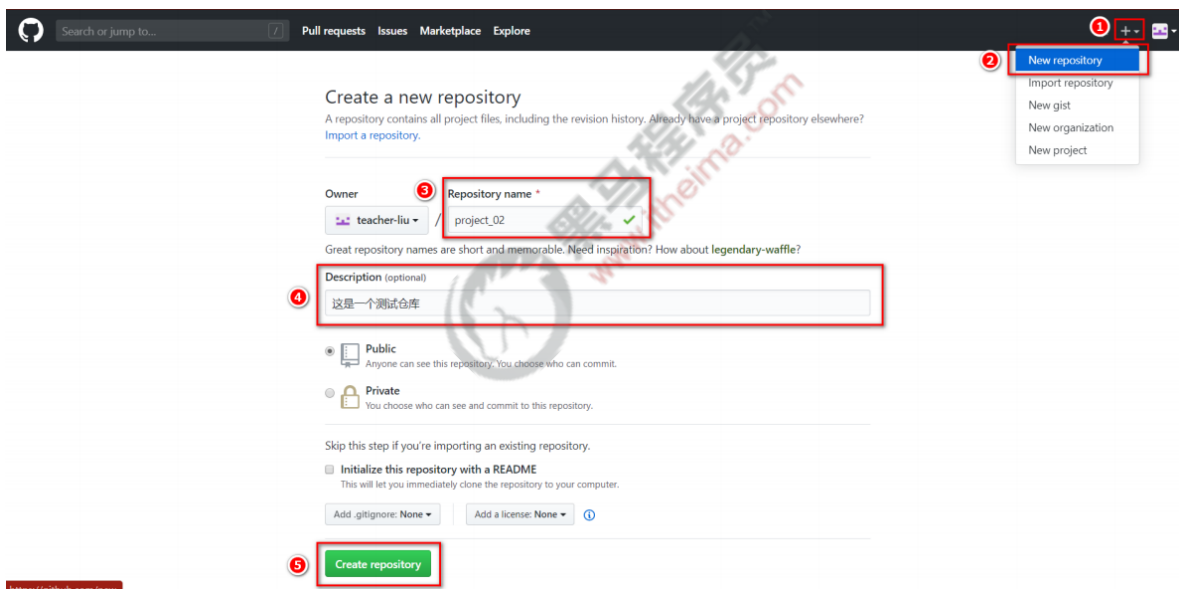
[Email preferences](#) - [Terms](#) - [Privacy](#) - [Sign into GitHub](#)

## 完成注册

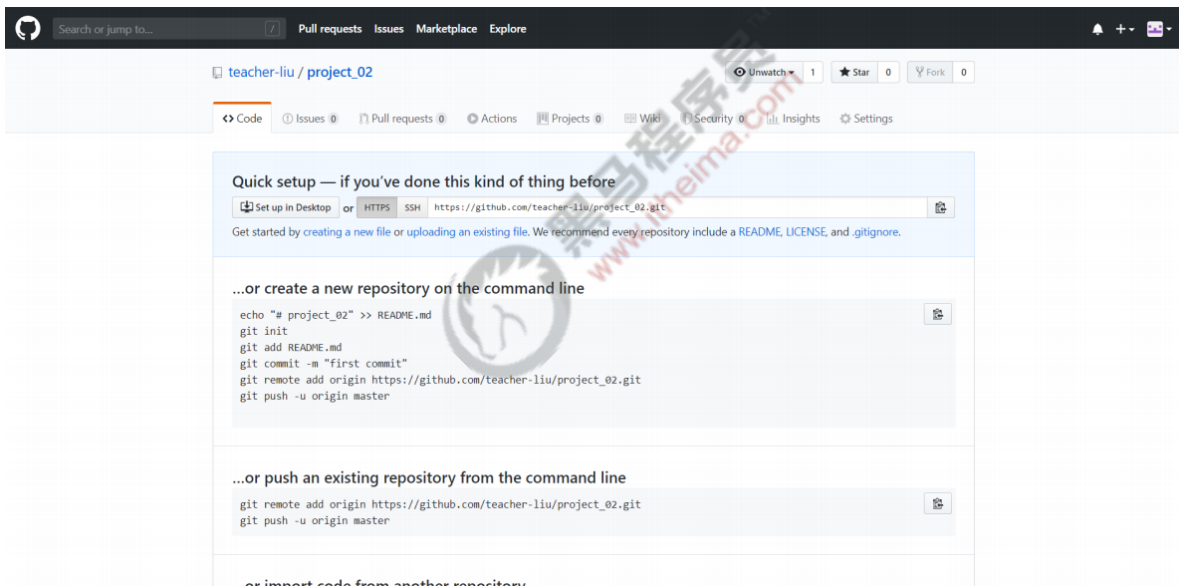


## 远程仓库的使用

### 新建空白远程仓库



### 新建空白远程仓库成功



# 远程仓库的两种访问方式

Github 上的远程仓库，有两种访问方式，分别是 HTTPS 和 SSH。它们的区别是：

- ① HTTPS：零配置；但是每次访问仓库时，需要重复输入 Github 的账号和密码才能访问成功
- ② SSH：需要进行额外的配置；但是配置成功后，每次访问仓库时，不需重复输入 Github 的账号和密码

注意：在实际开发中，推荐使用 SSH 的方式访问远程仓库。

## 基于 HTTPS 将本地仓库上传到 Github



Quick setup — if you've done this kind of thing before

Set up in Desktop or **HTTPS** SSH `https://github.com/teacher-liu/project_02.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

① 本地没有现成的 Git 仓库

...or create a new repository on the command line

```
echo "# project 02" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/teacher-liu/project_02.git
git push -u origin master
```

1. 使用终端命令创建 README.md 文档，并写入初始内容为 # project 02

2. 初始化本地 Git 仓库，并将文件的修改提交到本地的 Git 仓库中

3. 将本地仓库和远程仓库进行关联，并把远程仓库命名为 origin

4. 将本地仓库中的内容推送到远程的 origin 仓库中

② 本地有现成的 Git 仓库

...or push an existing repository from the command line

```
git remote add origin https://github.com/teacher-liu/project_02.git
git push -u origin master
```

1. 将本地仓库和远程仓库进行关联，并把远程仓库命名为 origin

2. 将本地仓库中的内容推送到远程的 origin 仓库中

## 基于 SSH key(☆☆☆)

### SSH key

SSH key 的作用：实现本地仓库和 Github 之间免登录的加密数据传输。

SSH key 的好处：免登录身份认证、数据加密传输。

SSH key 由两部分组成，分别是：

- ① id\_rsa（私钥文件，存放于客户端的电脑中即可）
- ② id\_rsa.pub（公钥文件，需要配置到 Github 中）

### 生成 SSH key

① 打开 Git Bash

② 粘贴如下的命令，并将 your\_email@example.com 替换为注册 Github 账号时填写的邮箱：

- `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`

③ 连续敲击 3 次回车，即可在 C:\Users\用户名文件夹\.ssh 目录中生成 id\_rsa 和 id\_rsa.pub 两个文件

### 配置 SSH key

① 使用记事本打开 id\_rsa.pub 文件，复制里面的文本内容

- ② 在浏览器中登录 Github, 点击头像 -> Settings -> SSH and GPG Keys -> New SSH key
- ③ 将 id\_rsa.pub 文件中的内容, 粘贴到 Key 对应的文本框中
- ④ 在 Title 文本框中任意填写一个名称, 来标识这个 Key 从何而来

## 检测 Github 的 SSH key 是否配置成功

- 打开 Git Bash, 输入如下的命令并回车执行:

```
ssh -T git@github.com
```

- 上述的命令执行成功后, 可能会看到如下的提示消息:

```
1 The authenticity of host 'github.com (IP ADDRESS)' can't be established.
2 RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IG0CspRomTxdCARLviKw6E5SY8.
3 Are you sure you want to continue connecting (yes/no)?
```

- 输入 yes 之后, 如果能看到类似于下面的提示消息, 证明 SSH key 已经配置成功了:

```
1 Hi username! You've successfully authenticated, but GitHub does not
2 provide shell access.
```

## 基于 SSH 将本地仓库上传到 Github

Quick setup — if you've <sup>①</sup> done this kind of thing before

☐ Set up in Desktop or ☒ HTTPS ☒ SSH git@github.com:teacher-liu/project\_03.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# project_03" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:teacher-liu/project_03.git
git push -u origin master
```

将本地现成的仓库推送到 Github

...or push an existing repository from the command line

<sup>②</sup> git remote add origin git@github.com:teacher-liu/project\_03.git  
git push -u origin master

1. 将本地仓库和远程仓库进行关联, 并把远程仓库命名为 origin  
2. 将本地仓库中的内容推送到远程的 origin 仓库中

注意: `git push origin master` 也能进行提交, `git push origin -u` 的话可以提交代码, 并且把 origin 当作默认的主机, 后续直接 `git push` 就可以提交到 origin 对应的主机

## 将远程仓库克隆到本地

打开 Git Bash, 输入如下的命令并回车执行:

```
git clone 远程仓库的地址
```

# Git 分支

# 分支的概念

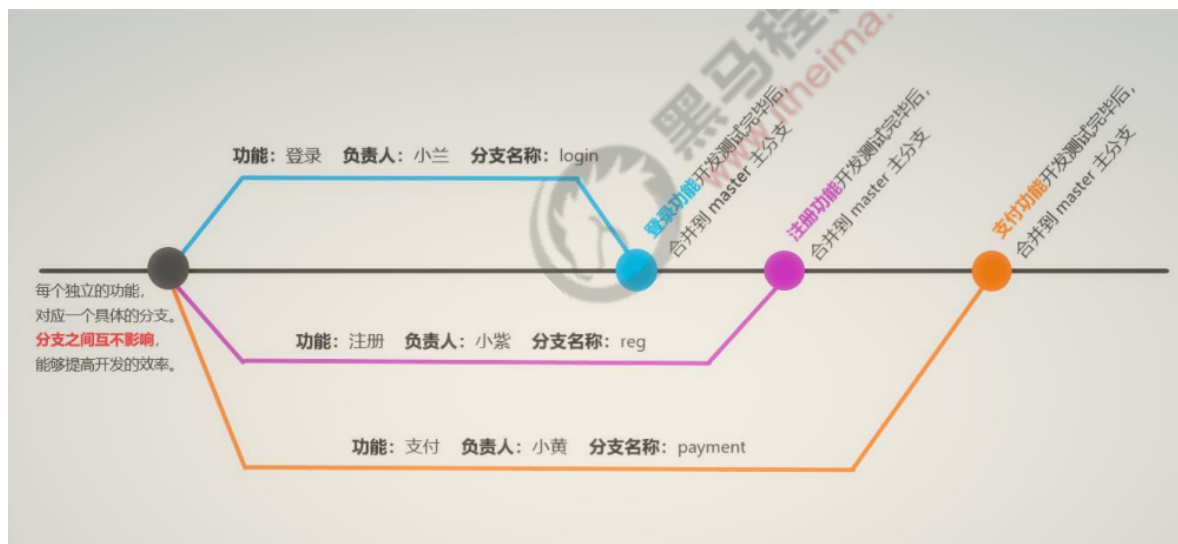
分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！



## 分支在实际开发中的作用

在进行多人协作开发的时候，为了防止互相干扰，提高协同开发的体验，建议每个开发者都基于分支进行项目

功能的开发，例如：



## master 主分支

在初始化本地Git仓库的时候，Git默认已经帮我们创建了一个名字叫做master的分支。通常我们把这个

master分支叫做主分支。





在实际工作中, `master` 主分支的作用是: 用来保存和记录整个项目已完成的功能代码。

因此, 不允许程序员直接在 `master` 分支上修改代码, 因为这样做的风险太高, 容易导致整个项目崩溃。

## 功能分支

由于程序员不能直接在 `master` 分支上进行功能的开发, 所以就有了功能分支的概念。

**功能分支**指的是专门用来开发新功能的分支, 它是临时从 `master` 主分支上分叉出来的, 当新功能开发且测试

完毕后, 最终需要合并到 `master` 主分支上, 如图所示:



## 查看分支列表(☆☆☆)

使用如下的命令, 可以查看当前 Git 仓库中所有的分支列表:

```
git branch
```

运行的结果如下所示:

```
C:\Windows\System32\cmd.exe

E:\code2>git branch
* master

E:\code2>
```

**注意:** 分支名字前面的 \* 号表示当前所处的分支。

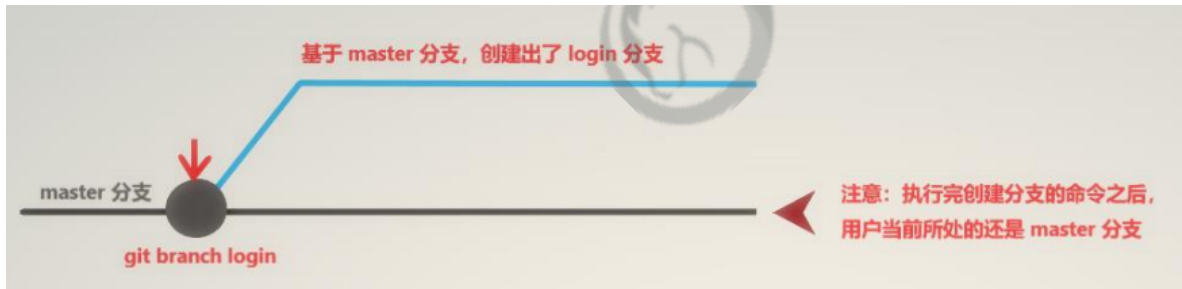
## 创建新分支



使用如下的命令，可以**基于当前分支，创建一个新的分支**，此时，新分支中的代码和当前分支完全一样：

```
git branch 分支名称
```

图示如下：

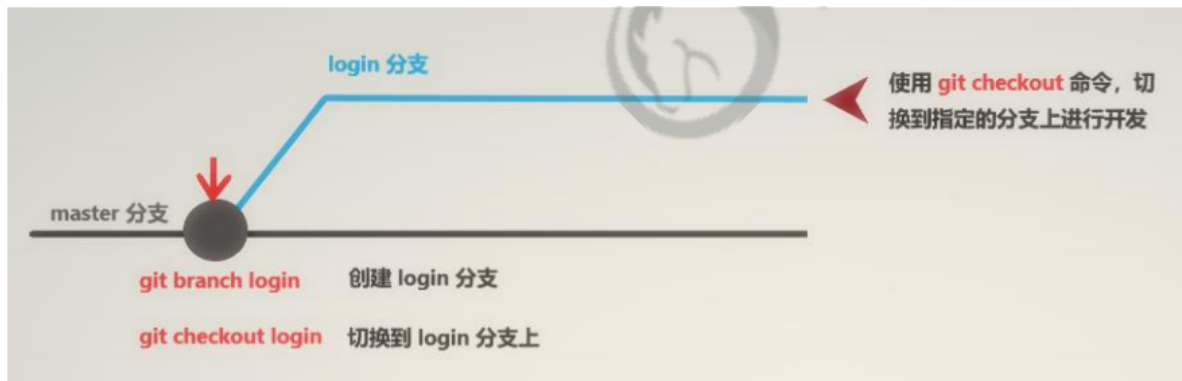


## 切换分支

使用如下的命令，可以**切换到指定的分支上**进行开发：

```
git checkout login
```

图示如下：

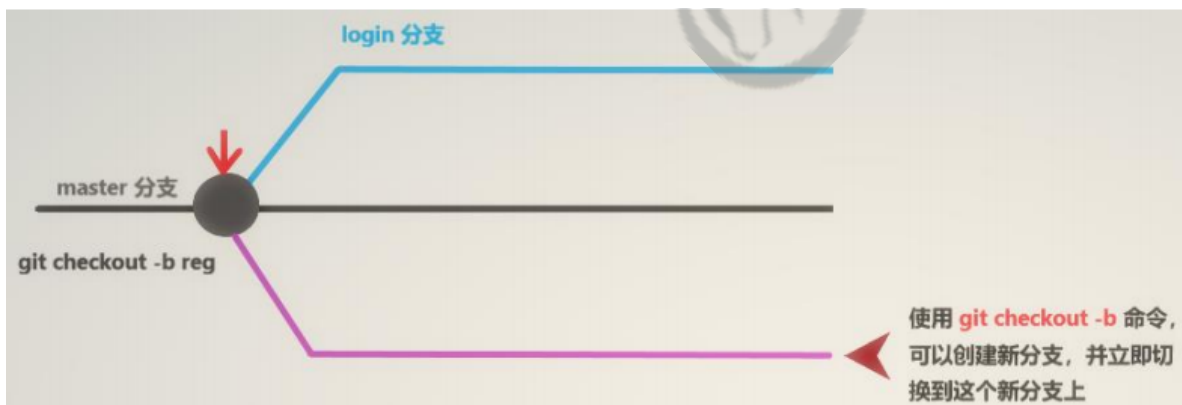


## 分支的快速创建和切换(☆☆☆)

使用如下的命令，可以**创建指定名称的新分支，并立即切换到新分支上**：

```
# -b 表示创建一个新分支  
# checkout 表示切换到刚才新建的分支上  
git checkout -b 分支名称
```

图示如下：



注意:

"`git checkout -b 分支名称`" 是下面

两条命令的简写形式:

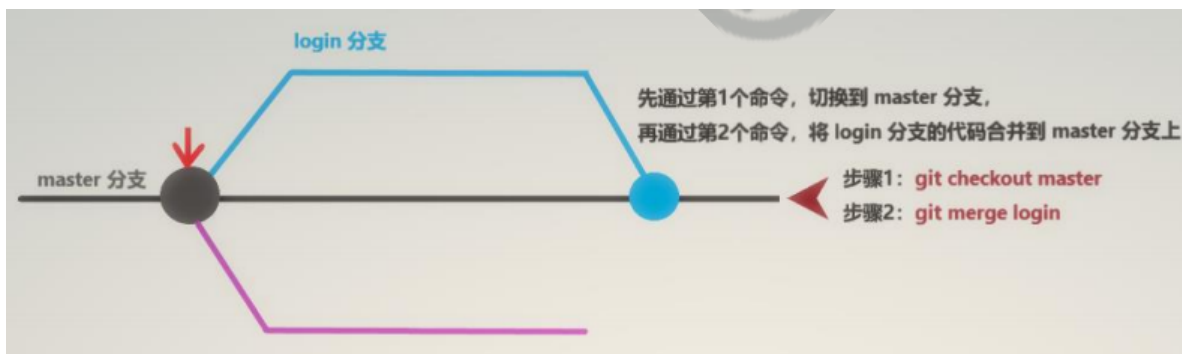
- ① `git branch 分支名称`
- ② `git checkout 分支名称`

## 合并分支

功能分支的代码开发测试完毕之后, 可以使用如下的命令, 将完成后的代码合并到 `master` 主分支上:

```
# 1. 切换到 master 分支
git checkout master
# 2. 在 master 分支上运行 git merge 命令, 将 login 分支的代码合并到 master 分支
git merge login
```

图示如下:



合并分支时的注意点:

假设要把 C 分支的代码合并到 A 分支,

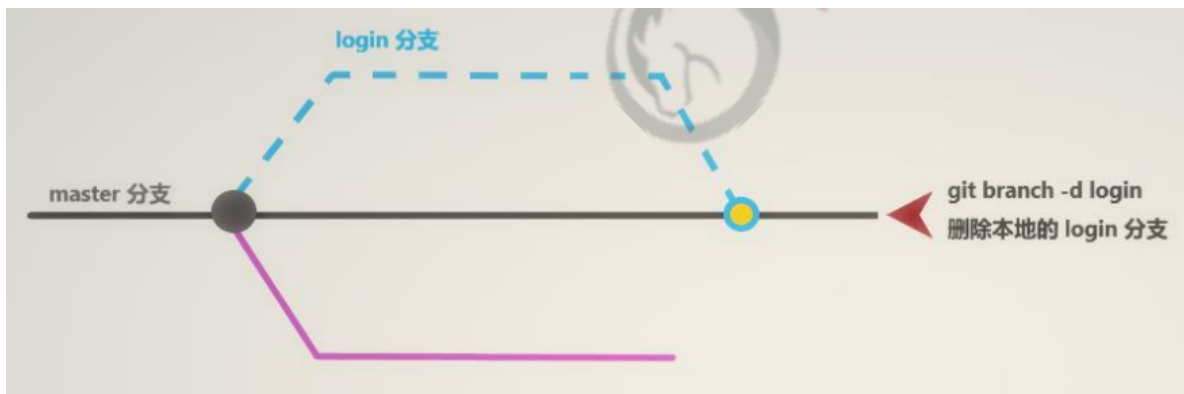
则必须先切换到 A 分支上, 再运行 `git merge` 命令, 来合并 C 分支!

## 删除分支

当把功能分支的代码合并到 `master` 主分支上以后, 就可以使用如下的命令, 删除对应的功能分支:

```
git branch -d 分支名称
```

图示如下:



## 遇到冲突时的分支合并

如果在两个不同的分支中，对同一个文件进行了不同的修改，Git 就没法干净的合并它们。此时，我们需要打开

这些包含冲突的文件然后手动解决冲突。

```
# 假设：在把 reg 分支合并到 master 分支期间
git checkout master
git merge reg

# 打开包含冲突的文件，手动解决冲突之后，再执行如下命令
git add .
git commit -m "解决了分支合并冲突的问题"
```

## 远程分支操作

### 将本地分支推送到远程仓库(☆☆☆)

如果是第一次将本地分支推送到远程仓库，需要运行如下的命令：

```
# -u 表示把本地分支和远程分支进行关联，只在第一次推送的时候需要带 -u 参数
git push -u 远程仓库的别名 本地分支名称:远程分支名称

# 实际案例
git push -u origin payment:pay

# 如果希望远程分支的名称和本地分支名称保持一致，可以对命令进行简化
git push -u origin payment
```

**注意：**第一次推送分支需要带 `-u` 参数，此后可以直接使用 `git push` 推送代码到远程分支。

### 查看远程仓库中所有的分支列表

通过如下的命令，可以查看远程仓库中，所有的分支列表的信息：

```
git remote show 远程仓库名称
```

### 跟踪分支(☆☆☆)

跟踪分支指的是：从远程仓库中，把远程分支下载到本地仓库中。需要运行的命令如下：

# 示例

```
git checkout pay
```

# 从远程仓库中，把对应的远程分支下载到本地仓库，并把下载的本地分支进行重命名

```
git checkout -b 本地分支名称 远程仓库名称/远程分支名称
```

# 示例

```
git checkout -b payment origin/pay
```

## 拉取远程分支的最新的代码

可以使用如下的命令，把远程分支最新的代码下载到本地对应的分支中：

# 从远程仓库，拉取当前分支最新的代码，保持当前分支的代码和远程分支代码一致

```
git pull
```

## 删除远程分支

可以使用如下的命令，删除远程仓库中指定的分支：

# 删除远程仓库中，制定名称的远程分支

```
git push 远程仓库名称 --delete 远程分支名称
```

# 示例

```
git push origin --delete pay
```

## 总结

- 能够掌握 Git 中基本命令的使用
  - `git init`
  - `git add .`
  - `git commit -m "提交消息"`
  - `git status` 和 `git status -s`
- 能够使用 Github 创建和维护远程仓库
  - 能够配置 Github 的 SSH 访问
  - 能够将本地仓库上传到 Github
- 能够掌握 Git 分支的基本使用
  - `git checkout -b 新分支名称`
  - `git push -u origin 新分支名称`
  - `git checkout 分支名称`
  - `git branch`