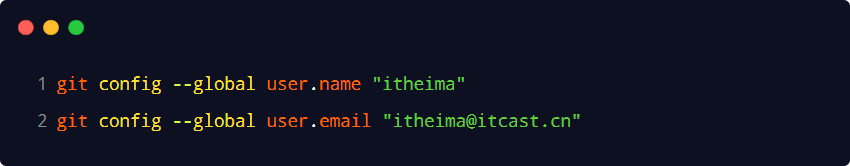
**下载**：在开始使用 Git 管理项目的版本之前，需要将它安装到计算机上。可以使用浏览器访问如下的网址，根据自己的操作系统，选择下载对应的 Git 安装包：

<https://git-scm.com/downloads>

**配置用户信息**

（注意：如果使用了 --global 选项，那么该命令只需要运行一次，即可永久生效。）

通过 git config --global user.name 和 git config --global user.email 配置的用户名和邮箱地址，会被写入到 C:/Users/用户名文件夹/.gitconfig 文件中。这个文件是 Git 的**全局配置文件**，**配置一次即可永久生效**。

可以使用记事本打开此文件，从而查看自己曾经对 Git 做了哪些全局性的配置

**检查配置信息**

git config --list --global

查看指定的全局配置项

git config user.name

git config user.email

**获取帮助信息**

可以使用 git help config 命令，无需联网即可在浏览器中打开帮助手册，

如果不想查看完整的手册，那么可以用 -h 选项获得更简明的“help”输出：

git config -h

**获取 Git 仓库的两种方式**

1. 将尚未进行版本控制的本地目录**转换**为 Git 仓库
2. 从其它服务器**克隆**一个已存在的 Git 仓库

如果自己有一个尚未进行版本控制的项目目录，想要用 Git 来控制它，需要执行如下两个步骤：

1. 在项目目录中，通过鼠标右键打开“Git Bash”
2. 执行 **git init** 命令将当前的目录转化为 Git 仓库

git init 命令会创建一个名为 .git 的隐藏目录，**这个 .git 目录就是当前项目的 Git 仓库**，里面包含了初始的必要文件，这些文件是 Git 仓库的必要组成部分。

**检查文件的状态**---未跟踪文件前面有红色的 ?? 标记

git status

git status -s

**跟踪新文件** ---新添加到暂存区中的文件前面有绿色的 A 标记

git add

**提交更新**

git commit -m

**对已提交的文件进行修改**---修改过的、没有放入暂存区的文件前面有红色的 M 标记

**暂存已修改的文件**

要暂存这次修改，需要再次运行 git add 命令

**向暂存区中一次性添加多个文件**

**git add .** “提交信息描述”

今后在项目开发中，会经常使用这个命令，将新增和修改过后的文件加入暂存区。

**取消暂存的文件**

git reset HEAD 要移除的文件名

**跳过使用暂存区域**

git commit -a -m “提交信息描述”

**移除文件**

从 Git 仓库中移除文件的方式有两种：

1. 从 Git 仓库和工作区中同时移除对应的文件 **git rm -f 文件名**
2. 只从 Git 仓库中移除指定的文件，但保留工作区中对应的文件 **git rm --cached 文件名**

**忽略文件**

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。 在这种情况下，我们可以创建一个名为 .gitignore 的配置文件，列出要忽略的文件的匹配模式。

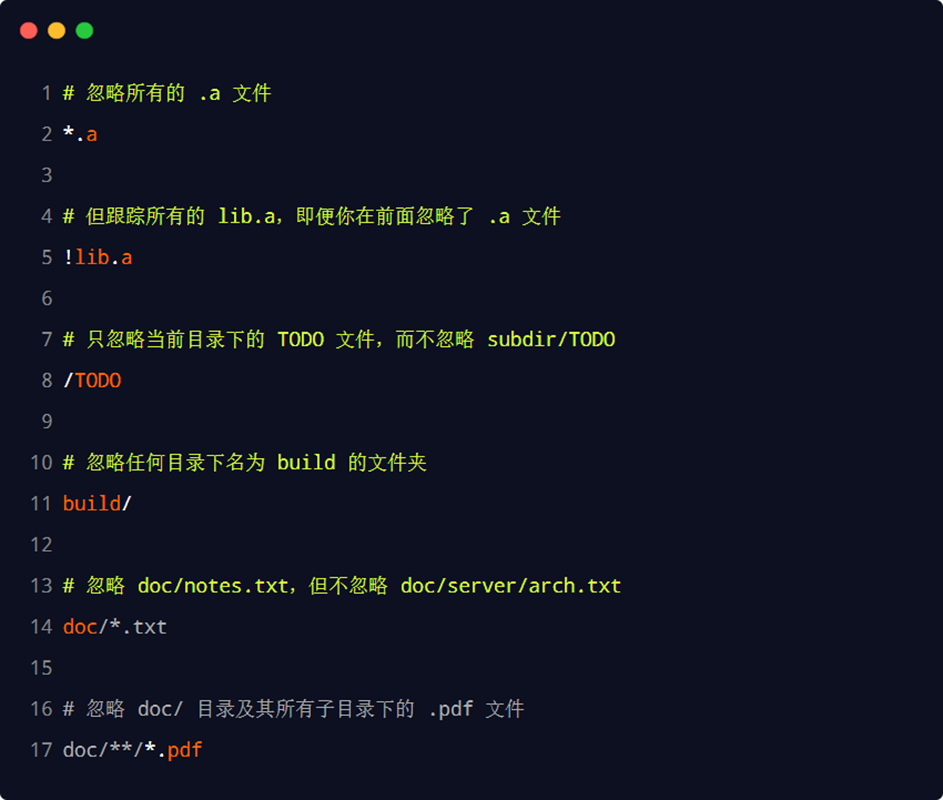
文件 .gitignore 的格式规范如下：

1. 以 # 开头的是注释
2. 以 / 结尾的是目录
3. 以 / 开头防止递归
4. 以 ! 开头表示取反
5. 可以使用 glob 模式进行文件和文件夹的匹配（glob 指简化了的正则表达式）

所谓的 glob 模式是指简化了的正则表达式：

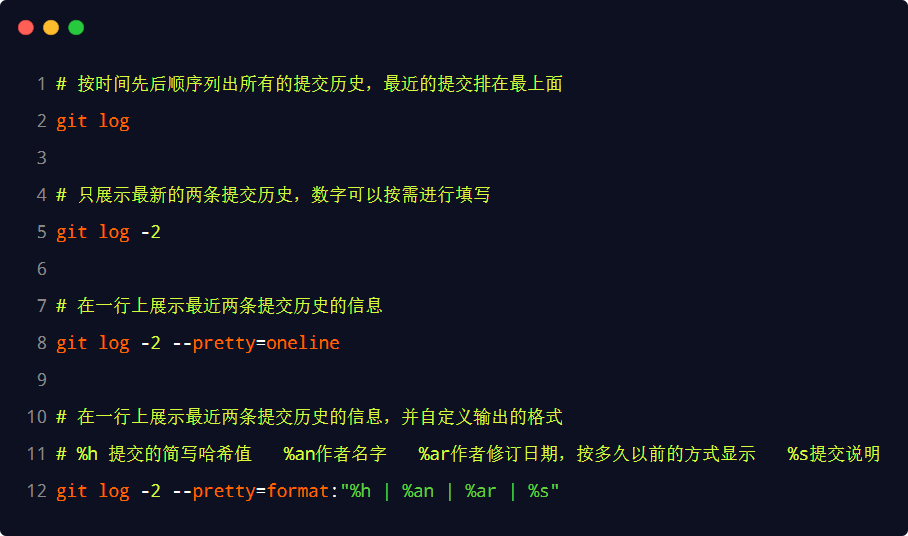
1. **星号 \*** 匹配零个或多个任意字符
2. **[abc]** 匹配任何一个列在方括号中的字符 （此案例匹配一个 a 或匹配一个 b 或匹配一个 c）
3. **问号 ?** 只匹配一个任意字符
4. 在方括号中使用**短划线**分隔两个字符， 表示所有在这两个字符范围内的都可以匹配（比如 [0-9] 表示匹配所有 0 到 9 的数字）

**两个星号 \*\*** 表示匹配任意中间目录（比如 a/\*\*/z 可以匹配 a/z 、 a/b/z 或 a/b/c/z 等）



**查看提交历史**

git log



**回退到指定的版本**



**小结**

1. 初始化 Git 仓库的命令
   * git init
2. 查看文件状态的命令
   * git status 或 git status -s
3. 一次性将文件加入暂存区的命令
   * git add .
4. 将暂存区的文件提交到 Git 仓库的命令
   * git commit -m "提交消息"

**Github - 了解开源相关的概念**

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 ）。

**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

**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

**Github - 远程仓库的使用**

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。

**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**

打开 Git Bash

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"

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**

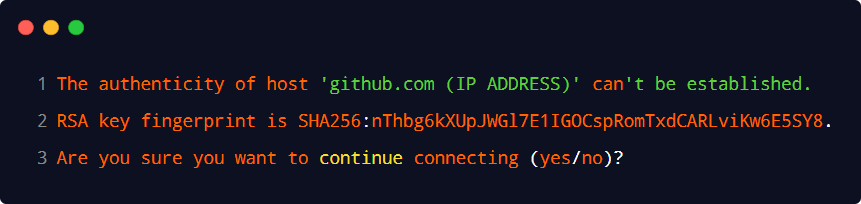
1. 使用记事本打开 id\_rsa.pub 文件，复制里面的文本内容
2. 在浏览器中登录 Github，点击头像 -> Settings -> SSH and GPG Keys -> New SSH key
3. 将 id\_rsa.pub 文件中的内容，粘贴到 Key 对应的文本框中
4. 在 Title 文本框中任意填写一个名称，来标识这个 Key 从何而来

**检测 Github 的 SSH key 是否配置成功**

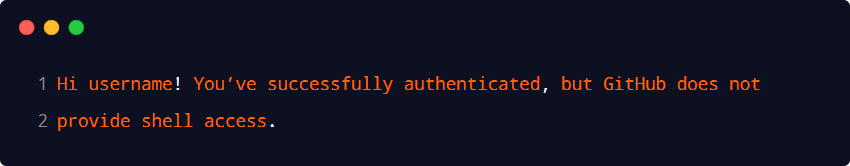
打开 Git Bash，输入如下的命令并回车执行：



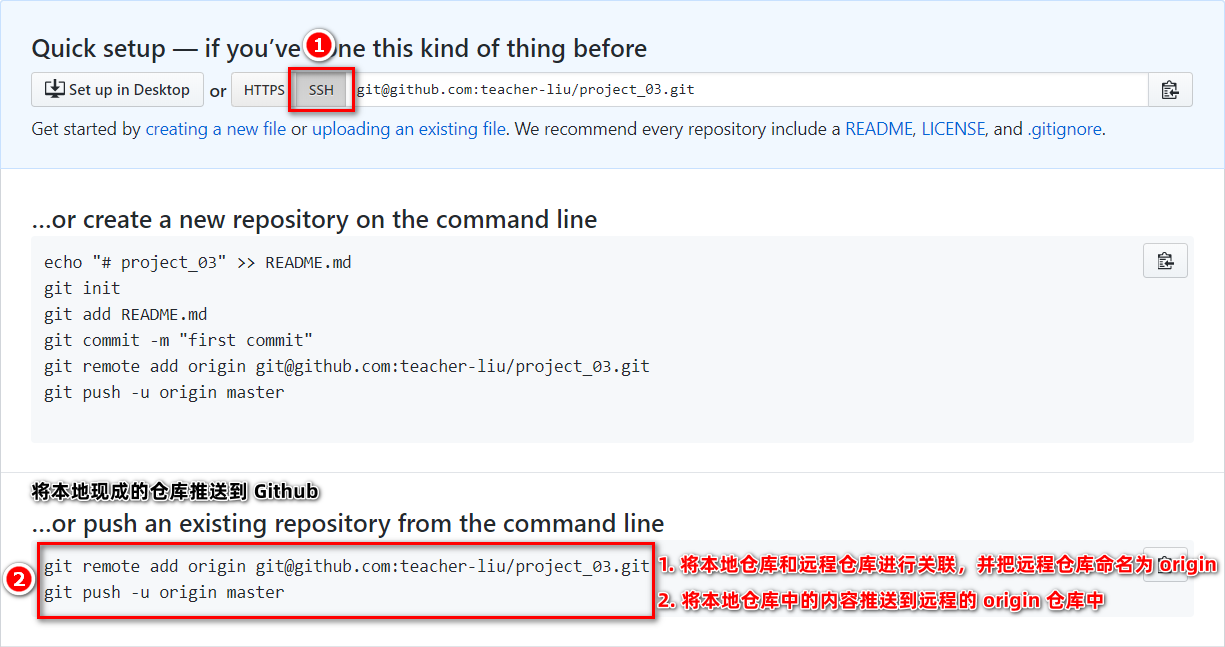
上述的命令执行成功后，可能会看到如下的提示消息：



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：



**基于 SSH 将本地仓库上传到 Github**



**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 分支**

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，

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。

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

**查看分支列表**

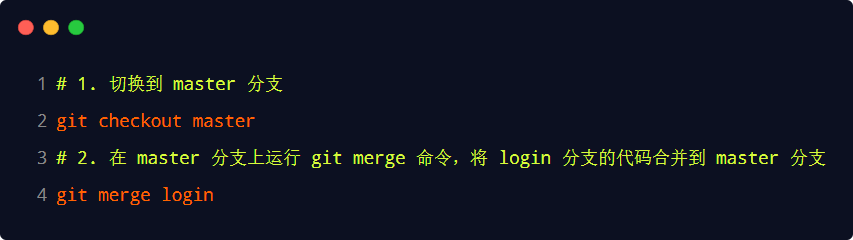
**git branch** (分支名字前面的 **\*** 号表示当前所处的分支。)

**创建新分支 ------ git branch 分支名称**

**切换分支 ---- git checkout 分支名称**

**分支的快速创建和切换 ---** **git checkout -b 分支名称**

**合并分支**

****

合并分支时的注意点：

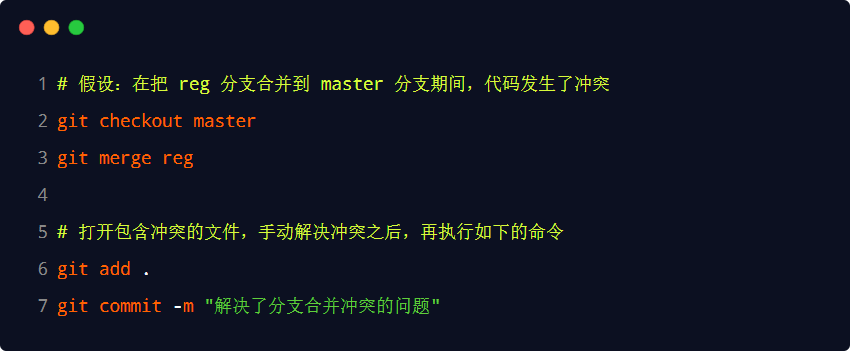
假设要把 C 分支的代码合并到 A 分支，则必须**先切换到 A 分支**上，**再运行 git merge 命令**，来合并 C 分支！

**删除分支--- git branch -d 分支名称**

在主分支上删除login分支

**遇到冲突时的分支合并**

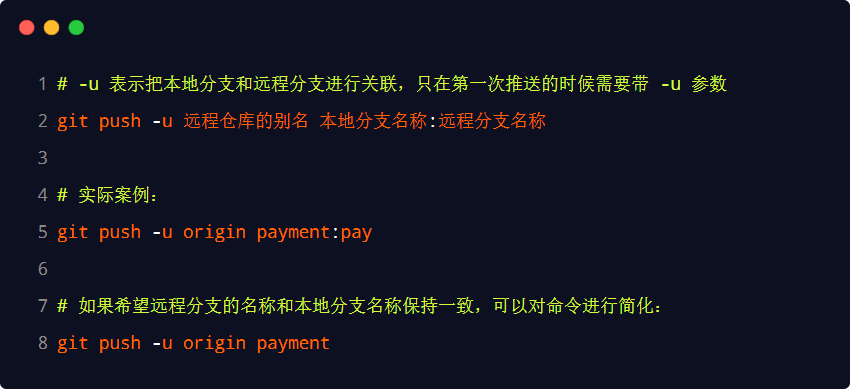
如果在两个不同的分支中，对同一个文件进行了不同的修改，Git 就没法干净的合并它们。 此时，我们需要打开这些包含冲突的文件然后**手动解决冲突**。



**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：

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。

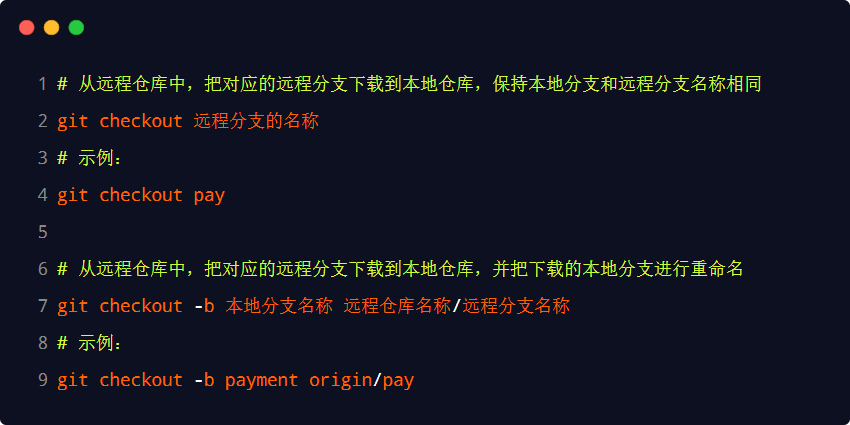


**查看远程仓库中所有的分支列表**

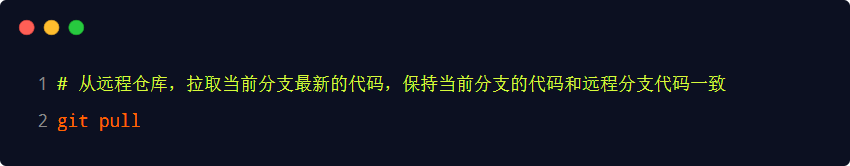


**跟踪分支**

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：



**拉取远程分支的最新的代码**



**删除远程分支**

