

一、首先在 linux 操作系统上安装 git 和 vim 等，然后准备好 ssh 密钥：

git 和 vim 的安装省略。

首先需要进行简单的配置，填用户名和邮箱（随意）

```
git config --global user.name "name"
```

```
git config --global user.email "e-mail"
```

然后开始准备 ssh 密钥

```
cd ~/.ssh
```

```
ssh-keygen -t rsa -b 4096 -C "e-mail"
```

双引号内可以填任何东西，但是大家普遍填的是自己邮箱

然后给密钥文件起个名字 2023_summer_QYN，并设置密码

设置好之后列出（ls）所有文件，可以看到两个密钥文件 2023_summer_QYN 和 2023_summer_QYN.pub，打开 2023_summer_QYN.pub

```
cat 2023_summer_QYN.pub
```

复制文件内的内容

浏览器打开 github，页面右上角找到 settings

新建一个 ssh key，起个名字，然后将刚刚的内容粘贴到 Key 部分

The screenshot displays the GitHub 'SSH keys / Add new' interface. On the left, the user's profile 'P YG (996-PYG)' is shown, along with a sidebar menu containing options like 'Public profile', 'Account', 'Appearance', 'Accessibility', 'Notifications', 'Access', 'Billing and plans', 'Emails', 'Password and authentication', 'Sessions', 'SSH and GPG keys' (which is highlighted), 'Organizations', 'Moderation', 'Code, planning, and automation', 'Repositories', 'Codespaces', 'Packages', 'Copilot', 'Pages', 'Saved replies', and 'Security'. The main content area is titled 'SSH keys / Add new' and features a form with three sections: 'Title' (an empty text input), 'Key type' (a dropdown menu currently showing 'Authentication Key'), and 'Key' (a large text area containing a placeholder message about valid key prefixes). A green 'Add SSH key' button is positioned at the bottom of the form.

保存该 SSH key

这一部分的目的：让 github 这个 git 托管平台确定你的身份

二、编写文件

回到 linux 操作系统，创建想放项目的空文件夹，在该文件夹打开终端

git clone [git@github.com:996-PYG/QYN..git](https://github.com/996-PYG/QYN..git)

然后自己新建文件夹，编写各种文件。文件状态如下所示：



文件状态说明以及该部分会用到的命令（前几天我不知道怎么想的，反正就写成英文了）

###~LOCAL~###

Before using these commands, please make sure that you have understood the statuses of files.

untracked : the file exists in the working directory, but has not yet been added to the local Git repository

new file : after 'git add' the file

modified : the file has been modified and not staged yet

staged : ready to be committed

conflict : If conflicts occur while merging branches

unmodified : the tracked file in working directory is the same as the file in staging area

git add <filename>

git add .

rename a tracked file

`git mv <filename> <new_filename>`

`git commit -m '<commens>'`

remember that 'just do one thing and do it well'

<type>[optional]: <description>

for example `git commit -m '<docs>[git_study]: add useful_commands.txt'`

`git commit -m '<docs>[git_study]: add useful_commands.txt'`

`git commit`

only type 'git commit' will open a nano terminal to help you complete your commit.

'^' means Ctrl key, '-m' means Alt in nano

when you want to remove some files in staging area, use 'git reset'

the files you removed would become 'modified'

`git reset <filename>`

remove a file

`git rm <filename>`

see the differences between current files and the files in staging area

`git diff`

check the statuses of files

`git status`

history of commitments

`git log`

这一部分类似于写完文档之后保存到自己电脑的特定位置，每次写完一个文档就 commit 一次，commit 的时候要说明本次 commit 的内容。

`git add -> git commit`

三、提交流程(从本地到线上)

首先确定我们要提交的远程仓库

```
git remote add origin git@github.com:996-PYG/QYN..git
```

origin 是给 git@github.com:996-PYG/QYN..git 起的一个名字，也可以是其他的名字

git remote 查看一下仓库

可以用 git remote rename <current_name> <new_name> 来重命名

可以用 git remote set-url <new_url> 来更改 url

在准备好之后，我们就可以将本地的分支推送到 github 平台了

```
git push <repository_name> <local_branch>
```

上面三节是 Git 的基本使用流程（面向单个分支）。

四、分支（branch）相关内容

如果没有分支，那么我们其实并不需要使用 git，使用百度网盘就可以了。

为什么要有分支？

举个例子，一个项目，要添加两个新特性，这两个特性是并行开发的，那么就要有两个新分支，这两个特性开发和测试完成之后再分别合并到 master 分支上。

对于单个分支，有时候发现此次提交的内容有些问题，但是也不是全错，因此不能 ctrl+z 全删了，就可以回退到前几次提交，对照着进行一些修改。

查看自己所在的分支

```
git log
```

```
git status
```

```
git branch --list
```

创建新的分支

```
git branch <branch_name>
```

切换到某个分支

```
git checkout <branch_name>
```

创建并切换到某个分支

```
git checkout -b <branch_name>
```

合并分支

某分支与 master 分支合并可以用以下的命令

```
git merge <branch_name>
```

（本地分支的合并）合并多个分支到 master 分支上是很可能出问题的，因为有可能不同的分支对同一个文件进行了修改。

冲突的文件会显示为 both modified 状态。

这个时候需要我们自己去确定哪一个修改是最好的，直接去冲突的文件内查看具体情况，冲突的部分会自动被标出。修改完成后再按照第二节中的操作对文件进行 git add 和 git commit 操作。

拉取远程分支

```
git fetch
```

将远程分支做为本地分支

```
git checkout -b <branch_name_local> origin/<branch_name_online>
```

直接使用 `git checkout <branch_name_online>` 也可以

更新本地的分支，如果本地分支和远程更新的分支不太一致，git 会自动尝试合并二者，否则需要手动更改

```
git pull
```

切换分支可能会出现的问题：

一个分支的文件写到一半，但是想要切换到其他分支查看情况，但是文件处在 modified 状态，无法切换分支。我们称当前我们的工作目录不干净。

写到一半的文件不建议使用 git commit 命令，推荐使用 `git stash` 对当前的文件进行储藏（储藏之后该分支所做的修改就都存起来了）

在切换回存储的分支时，使用 `git stash apply` 重新找回自己的修改，如果目录不干净那么是不可以恢复的。

stash 也可以保存多次修改，可以通过 `git stash list` 查看保存的修改，最上面（第 0 个）的是我们最近一次存储的。

用 `git stash apply stash@{<number>}` 选择想要取回的存储。

删除某一次的储藏

```
git stash drop stash@{<number>}
```

将最近的一次储藏进行恢复到工作目录并将储藏删除

```
git stash pop
```

将对应文件所做的修改删除（方便恢复储藏用的）

```
git checkout --<file_name>
```

五、撤回 commit

在第二节中提到 `git reset` 可以将文件从暂存状态回退到 `modified` 状态。如果要撤销刚刚进行的提交

```
git reset head~<number> --soft
```

在前几节的操作中，应该可以注意到，`head` 是一个指向当前分支的指针，`head~`指上一次 `commit`（倒数第一次 `commit`），`head~2` 指倒数第二次 `commit`。`--soft` 会把文件恢复为 `staged` 状态，如果不加则恢复为 `modified` 状态。

六、变基

主要是好看，以及大型项目压缩 `commit`，暂不使用。

作业：

在本地 `master` 分支上新建一个分支，将笔记整理好并 `commit` 后，在远程仓库上也新建这样的分支。该分支需要修改 `PYG/Git/useful_commands.txt`，在该文档最后整理本次实验用到的 `git` 命令（从 `clone` 项目开始的命令，同时注意要隐去个人信息）。