

CSAPP 实践课2 Git 基本使用

2023年3月3日 13:06

[1 Git 简介](#)

[1.1 为什么需要版本控制工具](#)

[1.2 操作：创建第一个仓库，体会版本控制](#)

[2 工作区与暂存区](#)

[2.1 基本概念](#)

[2.2 操作：将多个更改添加到暂存区一并提交](#)

[2.3 【自学】放弃更改](#)

[情况 1：没有添加到暂存区](#)

[情况 2：添加到暂存区](#)

[2.4 【自学】版本回退](#)

[3 分支](#)

[3.1 分支的基本概念](#)

[3.2 操作：创建分支并切换分支](#)

[3.3 【自学】其他关于分支的内容](#)

[4 远程仓库](#)

[4.1 远程仓库的概念](#)

[4.2 操作：生成公钥并添加到 Gitea](#)

[4.3 操作：推送本地分支到远程](#)

[4.4 【自学】其他关于远程仓库的内容](#)

[备注：关于作业提交](#)

1 Git 简介

1.1 为什么需要版本控制工具

Git 是一个版本控制工具。能够自动跟踪文件的更改，无需手动管理。

为什么需要版本控制工具？

例子：现在有一个文件需要多次修改

版本	文件名	用户	说明	日期
----	-----	----	----	----

1	service.txt	张三	删除了软件服务条款5	7/12 10:38
2	service.txt	张三	增加了License人数限制	7/12 18:09
3	service.txt	李四	财务部门调整了合同金额	7/13 9:51
4	service.txt	张三	延长了免费升级周期	7/14 15:17

现在我知道每一次修改做了什么操作，保存每一次操作的结果，那么就需要有 4 个文件，分别对应这 4 个版本，但是有了 git 后，我不需要手动管理 4 个文件，git 会帮我记录每一次的更改，我们可以了解到每一个版本的更改内容以及回退到之前的版本。

需要由 Git 进行管理的目录，称之为**仓库 (repository)**，这个目录下所有文件都可以被 Git 管理起来，每个文件的修改、删除都可以被跟踪，任何时刻都可以追踪历史，在将来的某个时刻可以回退。

注意

所有的版本控制系统，其实**只能跟踪文本文件的改动**，比如文本文件（TXT 格式），网页（HTML 格式），所有的程序代码（C 源码，Python 源码）等等。文本文件有编码，通常是使用标准的 UTF-8 编码。

而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从 100 KiB 改成了 120 KiB，但到底改了啥，版本控制系统不知道，也没法知道。

* 微软的 Word（后缀名为 .doc 或 .docx）是二进制文件。

1.2 操作：创建第一个仓库，体会版本控制


容器内已经安装好了 git，检查是否能够使用 `git` 命令：

```
1 $ which git
2 /usr/bin/git
```

由于 Git 跟踪仓库更改时，要求记录这次更改的信息以及用户名，因此需要进行一下配置，使用 `git config` 命令

```
1 $ git config --global user.name "Tong-yu Liu"
2 $ git config --global user.email "tyliu@stu.ecnu.edu.cn"
```

这里的 `--global` 表明对全局生效，这台机器上所有的 Git 仓库都会使用这个配置，请将引号里面的内容替换为自己的用户名与学校邮箱。

 `git config --list` 列出已有的配置信息。

这一次的练习我们视作第 0 次实验，我们先创建一个名为 lab0 的目录，后面的实验（lab1 到 lab5）也可以按照这个方式来命名，创建完成后，进入该目录。

```
1 $ mkdir lab0
2 $ cd lab0
3 $ pwd
4 /home/jovyan/lab0
```

使用 `git init` 命令初始化一个仓库

```
1 $ git init
2 hint: Using 'master' as the name for the initial branch. This default branch name
3 hint: is subject to change. To configure the initial branch name to use in all
4 hint: of your new repositories, which will suppress this warning, call:
5 hint:
6 hint:   git config --global init.defaultBranch <name>
7 hint:
8 hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
9 hint: 'development'. The just-created branch can be renamed via this command:
10 hint:
11 hint:   git branch -m <name>
12 Initialized empty Git repository in /home/jovyan/lab0/.git/
```

Git 提示我新建的是一个空仓库，这个目录下多了一个 `.git` 目录（这个目录是隐藏的）。

可以使用 `ls -A` 命令显示当前目录下所有文件与目录（包括以 `.` 开头的隐藏文件或目录，但不包括当前目录 `.` 与父目录 `..`）。

```
1 $ ls
2 <no output>
3 $ ls -A
4 .git
```

这个隐藏目录是 Git 来跟踪管理版本库的，请不要手动修改这个目录里面的文件。

不一定必须在空目录下创建 Git 仓库，选择一个已经有文件的目录也是可以的。

使用 `git status` 命令检查仓库的状态

```
1 $ git status
2 On branch master
3 No commits yet
4 nothing to commit (create/copy files and use "git add" to track)
```

平时在仓库中进行操作前与操作后可以多使用这条命令进行检查，这条命令是能够给出一些提示信息指导用户下一步操作。

添加一个文件到仓库中：新建一个 README 文件，写一句话到这个文件中，例如 "This is the repository for Lab 0."。

💡 请用 vim 进行操作

创建完成后，再检查仓库的状态，会发现有了变化。

```
1 $ vim README
2 $ ls
3 README
4 $ git status
5 On branch master
6 No commits yet
7 Untracked files:
8   (use "git add <file>..." to include in what will be committed)
9     README
10 nothing added to commit but untracked files present (use "git add" to track)
```

这里提示我们，刚才新建的 README 文件，是“没有跟踪的”。

如何让 Git 跟踪这个文件的更改呢？使用 `git add` 命令将更改暂存，完成后再检查仓库的状态

```
1 $ git add README
2 $ git status
3 On branch master
4 No commits yet
5 Changes to be committed:
6   (use "git rm --cached <file>..." to unstage)
7       new file:   README
```

可以发现仓库状态有了变化，提示“更改可以被提交”，那么我们进行一次提交，使用 `git commit` 命令进行。

```
1 $ git commit -m "add a README file"
2 [master (root-commit) c25a3c4] add a README file
3  1 file changed, 2 insertions(+)
4  create mode 100644 README
```

这里的 `-m` 选项表示“message”，用于记录这次提交的信息，一般用于描述这一次提交做了什么更改，方便追溯。

★ 完成上述操作后，自行完成下面的练习：

1. 在仓库中创建一个 Python 脚本 `hello.py`，使用 vim 进行编辑，编辑完成之后，使用 `python` 命令执行这个脚本；
2. 使用 `git add` 命令，让 Git “跟踪”该 Python 脚本的更改；
3. 使用 `git commit` 命令，进行一次提交，提交信息为“add a Python script.”

操作过程中，使用 `git status` 检查仓库的状态。

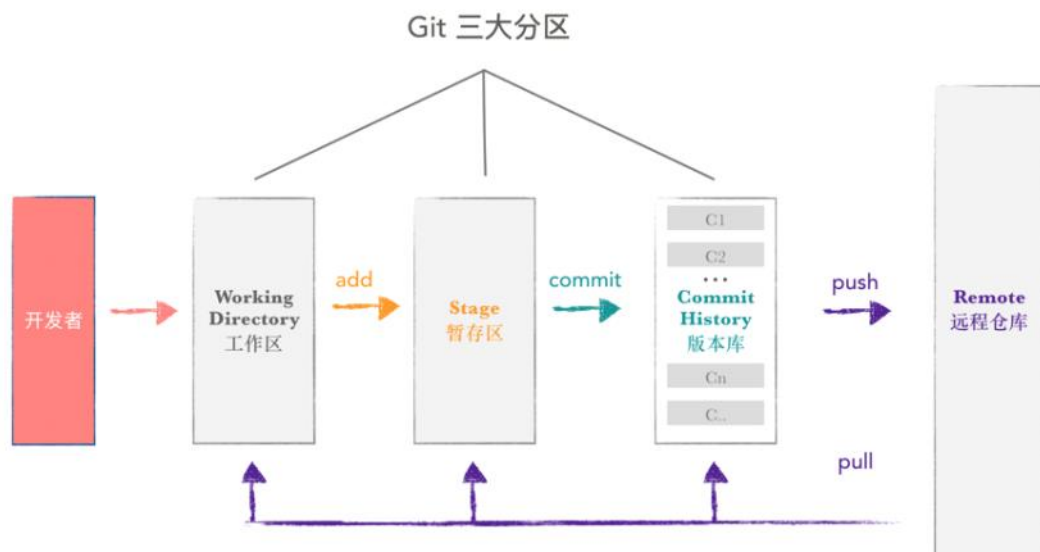
完成上述操作后，使用 `git log` 检查是否已经有了 2 条提交记录，如下所示：

```
1 $ git log
2 commit 0e03d19ce2570ee8da2975cfda2a6fabd4c9a15c (HEAD -> master)
3 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
4 Date:   Sun Mar 5 04:56:12 2023 +0000
5
6     add a Python script.
7
8 commit c25a3c4e6539c42ff087c777112faaf05ec64615
9 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
10 Date:   Sat Mar 4 16:19:49 2023 +0000
11
12     add a README file
```

2 工作区与暂存区

2.1 基本概念

Git 版本控制工具的一个特点是引入了**暂存区 (stage)** 的概念。



如果希望我们的更改被版本库跟踪，需要进行 2 步操作：

1. 用 `git add`，实际上就是把修改添加到暂存区；
2. 用 `git commit`，实际上就是把暂存区中的所有内容提交到仓库的当前分支；

关于分支，上面 `git status` 命令输出的第一行 "on branch master"，是在提示我们当前正处在 master 分支上。因为初始化仓库时，Git 自动创建了唯一的一个 master 分支，`git commit` 就是往 master 分支上提交更改。

关于分支以及这幅图涉及的远程仓库的话题，我们后面会继续讨论。

有些时候，我们可能需要修改仓库的多个文件之后做一次提交。

举例：正在做一个项目，需要修复一个 bug，这个 bug 的原因是调用了一个错误的函数，这个函数在多个源文件中被使用，因此需要同时修改这些源文件。

对于这种情况，可以将多个文件的更改全部放到暂存区，然后，一次性提交暂存区的所有修改，即可以 `git add` 好多次然后 `git commit` 一次。

Git 的特点在于：**跟踪并管理的是更改，而非文件**。每次更改，如果不使用 `git add` 添加到暂存区，就不会被提交。

2.2 操作：将多个更改添加到暂存区一并提交

★ 练习

1. 在代码仓库中新增一个 `LICENCES` 文本文件，随便往里面写入一些内容；
2. 修改 `README` 文件，任意增加一行文字；
3. 将上面 2 个文件的更改添加到暂存区；
4. 提交。

操作过程中，使用 `git status` 检查仓库的状态。

2.3 【自学】放弃更改

关于工作区与暂存区，建议还需要掌握放弃更改的操作方法。

放弃更改分为 2 种情况：

1. 没有 `git add` 添加到暂存区;
2. 已经 `git add` 添加到暂存区。

情况 1：没有添加到暂存区

没有 `git add` 添加到暂存区，如果没有修改很多，直接手动修改删除即可。

在这种情况下，使用 `git status` 时，会提示 "Changes not staged for commit"，括号内注明了提示：

```
1 $ git status
2 On branch master
3 Changes not staged for commit:
4   (use "git add <file>..." to update what will be committed)
5   (use "git restore <file>..." to discard changes in working directory)
6       modified:   hello.py
7
8 no changes added to commit (use "git add" and/or "git commit -a")
```

这里我修改了 `hello.py` 的 Python 脚本，会显示上面的信息。

提示信息给出了 2 个选项：要么使用 `git add` 将更改添加至暂存区，要么使用 `git restore <file>` 撤销更改。

如果修改很多记不得原来是什么样子的话，可以使用 `git restore <file>` 进行恢复，撤销在工作区的更改。

特别注意，这边又可以区分出来 2 种情况：

1. 该文件自修改后还没有被放到暂存区，使用上述指令就回到和版本库一模一样的状态，即回到最近一次 `git commit` 的状态；
2. 该文件的修改已经添加到暂存区，又作了修改，使用上述指令就回到添加到暂存区后的状态，即回到最近一次 `git add` 的状态。

情况 2：添加到暂存区

已经 `git add` 添加到暂存区，但是还没有提交。

在这种情况下使用 `git status` 时，会提示 "Changes to be committed"，括号内注明了提示：

```
1 $ git status
2 On branch master
3 Changes to be committed:
4   (use "git restore --staged <file>..." to unstage)
5       modified:   hello.py
```

这里我将刚才对 `hello.py` 文件的修改使用 `git add` 进行了提交，之后会显示上面的信息。

这提示我们使用 `git restore --staged <file>` 可以把暂存区的修改撤销（称之为 "unstage"），重新放回工作区。

这时候再使用 `git status`，就会发现工作区是有未暂存的修改，因为 `unstage` 是回退到工作区。

这样就回到了情况 1，可以用 `git restore <file>` 放弃工作区的更改。

2.4 【自学】版本回退

每一次提交都会生成一个 commit id，是一串十六进制字符串，该字符串是唯一的，利用这个字符串就可以进行版本回退。

例如，先使用 `git log --pretty=oneline` 列出提交记录：

```
1 $ git log --pretty=oneline
2 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL
3 e475afc93c209a690c39c13a46716e8fa000c366 add distributed
4 eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0 wrote a readme file
```

实际操作中，不需要全部输入全部的 commit id，只需要键入前几个，保证 Git 能够找到对应的提交记录即可。

使用下面的命令，回退到上一个提交：

```
1 $ git reset --hard e475af
2 HEAD is now at e475afc add distributed
```

可以看到此时代码仓库（工作区，暂存区，版本库）都回到了这一次提交的状态，此时如果使用 `git log` 只能看到此提交之前的提交记录，更新的版本已经看不到了。

那么怎样做才能恢复到最新版本呢？

使用 `git reflog` 调出每一次的 git 命令记录，可以看到

```
1 $ git reflog
2 e475afc HEAD@{1}: reset: moving to e475afc
3 1094adb (HEAD -> master) HEAD@{2}: commit: append GPL
4 e475afc HEAD@{3}: commit: add distributed
5 eaadf4e HEAD@{4}: commit (initial): wrote a readme file
```

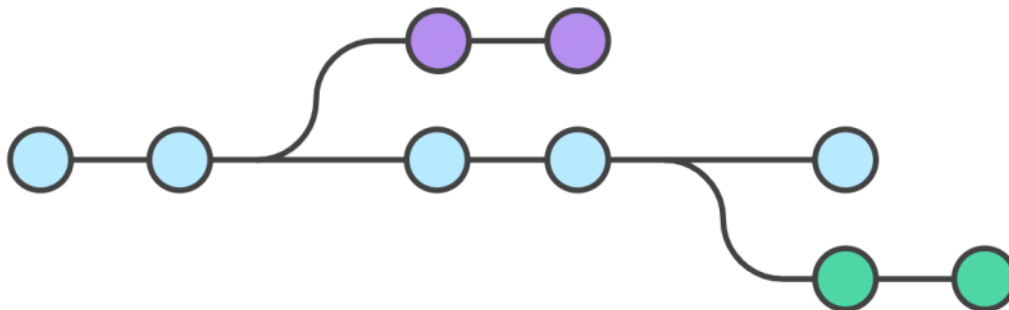
看输出的第 2 行，可以找到最新一次提交的 commit id，因此可以使用

```
1 $ git reset --hard 4d85b3a
2 HEAD is now at 4d85b3a append GPL
```

3 分支

3.1 分支的基本概念

几乎每一种版本控制系统都以某种形式支持分支，一个分支代表一条独立的开发线。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的时候继续工作。



之前我们的提交都是在 master 分支上进行的，这个分支称之为主分支，是不可删除的，相当于上面这幅图的淡蓝色主线。但是我们可以在 master 分支某个提交的基础上创建分支，相当于上面这幅图从主线分叉出来的紫色与蓝色支线。在某一个分支上进行的提交，不会影响其他分支。

3.2 操作：创建分支并切换分支

在 master 分支上，使用 `git branch <branch_name>` 的命令创建分支。创建分支后，再分别使用 `git status` 检查所在分支，`git branch`（不加参数）列出当前仓库的分支。

```
1 $ git branch lab0
2 $ git status
3 On branch master
4 nothing to commit, working tree clean
5 $ git branch
6   lab0
7 * master
```

这样创建分支之后，实际上我仍然在 master 分支上，可以从 `git status` 的输出中看出，同时 `git branch` 输出中前面加上 “*” 的分支表示的是当前的分支。

那么如何才能切换到刚刚创建的 lab0 分支上呢？

```
1 $ git checkout lab0
2 Switched to branch 'lab0'
3 $ git status
4 On branch lab0
5 nothing to commit, working tree clean
6 $ git branch
7 * lab0
8   master
```

使用 `git checkout` 之后，完成了切换分支的操作。

在 lab0 分支上进行一次提交，例如新增了一个 Python 脚本 `add.py`，提交后，使用 `git log` 查看提交记录。

```
1 $ git log
2 commit 1123131a9b3c51ce85d5c4228e8e3a8539b626ec (HEAD -> lab0)
3 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
4 Date:   Mon Mar 6 02:25:58 2023 +0000
5
6     add a Python file with a an 'add' function
7
8 commit b8c9b019aca38e5394ade8c9105fcadfb3413faf (master)
9 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
10 Date:   Mon Mar 6 02:13:59 2023 +0000
11
12     add a LICENCES file and modify README file.
13
14 commit 0e03d19ce2570ee8da2975cfda2a6fabd4c9a15c
15 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
16 Date:   Sun Mar 5 04:56:12 2023 +0000
17
18     add a Python script.
19
20 commit c25a3c4e6539c42ff087c777112faaf05ec64615
21 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
22 Date:   Sat Mar 4 16:19:49 2023 +0000
```



```
23
24 add a README file
```

如果我们切回 master 分支，可以看到刚才新建的 `add.py` 不见了，再使用 `git log` 查看提交记录

```
1 $ git checkout master
2 Switched to branch 'master'
3 $ ls
4 hello.py  LICENCES  README
5 $ git log
6 commit b8c9b019aca38e5394ade8c9105fcadfb3413faf (HEAD -> master)
7 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
8 Date: Mon Mar 6 02:13:59 2023 +0000
9
10 add a LICENCES file and modify README file.
11
12 commit 0e03d19ce2570ee8da2975cfda2a6fabd4c9a15c
13 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
14 Date: Sun Mar 5 04:56:12 2023 +0000
15
16 add a Python script.
17
18 commit c25a3c4e6539c42ff087c777112faaf05ec64615
19 Author: Tong-yu Liu <tyliu@stu.ecnu.edu.cn>
20 Date: Sat Mar 4 16:19:49 2023 +0000
21
22 add a README file
```

可见分支上的提交不会影响到其他分支。

3.3 【自学】其他关于分支的内容

关于 Git 的分支，其他建议学习的主题：

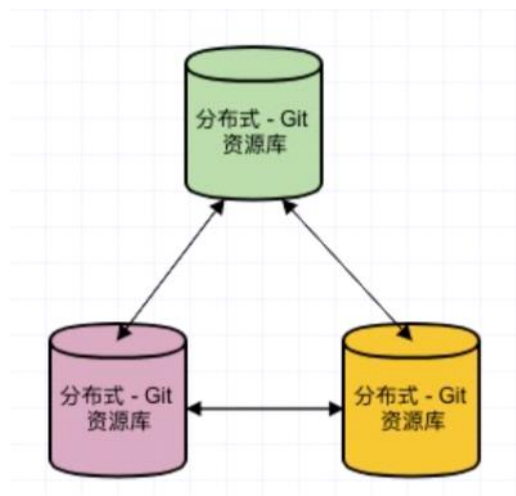
- 合并分支： `git merge <branch_name>` 合并指定分支到当前分支
 - 进阶：理解 HEAD 指针与 Git 创建分支与合并分支的实现原理
- 合并分支时的冲突解决
- 多人协作的流程与变基： `git rebase`

推荐学习资料：[分支管理 - 廖雪峰的官方网站 \(liaoxuefeng.com\)](https://liaoxuefeng.com)

4 远程仓库

4.1 远程仓库的概念

Git 是分布式版本控制系统，所谓分布式，指的是同一个仓库可以分布到不同的机器上，这样即使自己机器的仓库因故障导致数据丢失，也可以通过在其他机器上的副本进行恢复。



事实上 Git 这样分布式的设计非常便于团队协作开发，在这种场合，通常由一台机器充当“服务器”的角色，长时间开机，每个开发成员都从这个“服务器”仓库克隆一份到自己的机器上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

运行 Git 的服务器实际上可以自行搭建，例如 Gitea，不过个人使用服务器成本还是比较高的，好在有许多提供 Git 仓库托管服务的网站，例如最知名的 GitHub，只要注册一个 GitHub 账号，就可以免费获得 Git 远程仓库。

学院也提供了代码仓库的托管平台“水杉码园”，这是基于开源的 Gitea 开发的学习平台，学生能够将自己的代码推送到自己课程的远程仓库中。

登录水杉码园，进入课程后能够看到自己的仓库，进入后是这样的界面。

CSAPPJames.2023Spring.DaSE / 10225501448 Private

取消关注 5 点赞 0 派生 0

<> 代码 0 话题 0 合并请求 0 版本发布 0 百科 动态 仓库设置

student 10225501448 李度's repo

管理主题

1 提交 21 分支 18 KiB

分支: master 创建合并请求

新建文件 新建文件夹 上传文件 HTTPS SSH https://gitea.shuishan.net.cn/

苏斌 91b0212e0d Initial commit 43 分钟前

README.md Initial commit 43 分钟前

README.md

10225501448

student 10225501448 李度's repo

在右侧有一栏是一个地址，只要知道这个地址，就可以使用 `git push` 将本地的仓库推送到远程，或者使用 `git pull` 将远程的仓库拉回本地。

这样的操作是需要身份验证的，那么 Git 是如何进行身份验证的呢？是通过本地机器生成的密钥对（公钥和私钥），将公钥上传到 Gitea 上使得个人账户与本地机器绑定，私钥则保留在本地机器上，访问 Gitea 上托管的远程仓库时会进行验证。

4.2 操作：生成公钥并添加到 Gitea

按如下命令生成用于SSH连接的密钥对：

```
1 $ ssh-keygen -t ed25519 -C "tyliu@stu.ecnu.edu.cn"
```

请将引号里面的内容替换为自己的学校邮箱。

执行上述命令时，会询问一些配置，这里均使用默认配置，因此三次回车即可。

此时会在 `~/.ssh` 目录下生成名为 `id_ed25519` 与 `id_ed25519.pub` 的 2 个文本文件，分别对应私钥与公钥。

输出公钥的内容

```
1 $ cat ~/.ssh/id_ed25519.pub
2 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI... tyliu@stu.ecnu.edu.cn
```

进入 Gitea 平台，登陆后点击右上角用户头像，点击“设置”，切换到“SSH/GPG密钥”选项卡，将会列出所有已添加的密钥。

点击“增加密钥”，将上述公钥的内容粘贴到“密钥内容”框中，并为这个密钥命名，完成后点击“增加密钥”。

[个人信息](#) [账号](#) [安全](#) [应用](#) **SSH / GPG 密钥** [课程](#) [仓库列表](#)

您的 SSH 密钥 'Shuishan Jupyter' 添加成功。

管理 SSH 密钥 [增加密钥](#)

这些 SSH 公钥已经关联到你的账号。相应的私钥拥有完全操作你的仓库的权限。

 devcloud SHA256:XtRdxqA8qIfC28SV7Gkggq9hEyWo9NQx/FYG53jwpvo 增加于 Dec 01, 2022 — ① 没有最近活动	删除
 Shuishan Jupyter SHA256:OtgBE4Mlayzz0VGgSPZt7+kebVqUCssYGpDaiyEmy0 增加于 Mar 06, 2023 — ① 没有最近活动	删除

需要帮助？请查看有关 [如何生成 SSH 密钥](#) 或 [常见 SSH 问题](#) 寻找答案。

增加 SSH 密钥

密钥名称
tyliu@stu.ecnu.edu.cn

密钥内容
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI... tyliu@stu.ecnu.edu.cn

[增加密钥](#)

📌 可添加多个公钥与Gitea账户绑定，每一台本地机器对应一个公钥，建议公钥的标题能够标识是哪一台机器。

4.3 操作：推送本地分支到远程

推送前，需要配置远程仓库的地址，从课程个人仓库的界面，右侧的栏目可以找到，有 HTTPS 协议与 SSH 协议的，都是可以的。命令格式为：`git remote add <remote_name> <git_url>`

```
1 $ git remote add origin
https://gitea.shuishan.net.cn/CSAPP.James.2023Spring.DaSE/TA_repo.git
```

其中 `<remote_name>` 是远程仓库的名称，通常叫做 `origin`，上面的 `git_url` 请替换为自己仓库的地址。

使用 `git remote -v` 检查已经添加的远程仓库信息

```
1 $ git remote -v
2 origin https://gitea.shuishan.net.cn/CSAPP.James.2023Spring.DaSE/TA_repo.git (fetch)
3 origin https://gitea.shuishan.net.cn/CSAPP.James.2023Spring.DaSE/TA_repo.git (push)
```

添加后，可以使用 `git push` 将本地分支推送到远程。命令格式为：`git push <remote_name> <branch_name>`


这里将刚刚建立的，本地仓库的 lab0 分支，推送到远程仓库

```
1 $ git push origin lab0
2 Username for 'https://gitea.shuishan.net.cn': tyliu_stu_ecnu_edu_cn
3 Password for 'https://tyliu_stu_ecnu_edu_cn@gitea.shuishan.net.cn':
4 Enumerating objects: 13, done.
5 Counting objects: 100% (13/13), done.
6 Delta compression using up to 4 threads
7 Compressing objects: 100% (9/9), done.
8 Writing objects: 100% (13/13), 1.20 KiB | 136.00 KiB/s, done.
9 Total 13 (delta 1), reused 0 (delta 0), pack-reused 0
10 remote: . Processing 1 references
11 remote: Processed 1 references in total
12 To https://gitea.shuishan.net.cn/CSAPP.James.2023Spring.DaSE/TA_repo.git
13 * [new branch] lab0 -> lab0
```

此时要求用户输入用户名与密码，请使用登录水杉码园的用户名与密码。

最后会提示，`lab0 -> lab0`，该输出的含义是将本地的 lab0 分支推送到远程的 lab0 分支。

这时候再登录到课程仓库，可以看到本地仓库的提交能够在远程仓库中看见。

 CSAPP.James.2023Spring.DaSE / TA_repo Internal

取消关注 5 点赞 0 派生 0

<> 代码 0 话题 0 合并请求 0 版本发布 0 百科 动态 仓库设置

这是用于计算机系统实践课教学的仓库

[管理主题](#)

4 提交 1 分支 18 KiB

分支: lab0 创建合并请求

新建文件 新建文件夹 上传文件 HTTPS SSH https://gitea.shuishan.net.cn/ 仓库设置

过滤分支或标签...

分支列表 标签列表

lab0

Python file with an 'add' function 40 分钟前

add a LICENCES file and modify README file. 52 分钟前

add a LICENCES file and modify README file. 52 分钟前

add.py 40 分钟前

hello.py 22 小时前

README

This is the repository for Lab 0.
We have a README file and a LICENCES file.

★ `git push` 命令的另一种格式

```
git push <remote_name> <local_branch_name>:<remote_branch_name>
```

分区助教 的第 12 页

这样的命令是用于本地分支的名称与远程分支的名称不一致的情况下使用的。例如：本地仓库我在 master 分支上完成了作业，但是作业要求提交在课程仓库的 homework01 分支上，那么这个时候就应当使用上面这种格式来进行。不过实际开发过程中，一般本地仓库与远程仓库的分支名称应当保持一致。

4.4 【自学】其他关于远程仓库的内容

关于 Git 的分支，其他建议学习的主题：

- 从远程仓库克隆：`git clone`
- 从远程仓库拉取：`git pull`，其命令格式与 `git push` 基本一致
- 删除远程仓库：`git remote rm <remote_name>`
- 多人协作开发的流程

推荐学习资料：[远程仓库 - 廖雪峰的官方网站 \(liaoxuefeng.com\)](http://liaoxuefeng.com)，[多人协作 - 廖雪峰的官方网站 \(liaoxuefeng.com\)](http://liaoxuefeng.com)