

CS100 Recitation 3

Git Basics

Speaker Yuxuan Li (liyx22025@shanghaitech.edu.cn)

VSPLab@SIST



上海科技大学

ShanghaiTech University

目录

1 Git 是什么

2 Git：在本地仓库使用

3 Git：和远程仓库交互

4 总结

Git 是什么

Git 是什么

Git 是一个版本控制软件，它以增量方式记录代码修改，保存完整版本历史！

立志成才 报国裕民

Git 与 GitHub

Git 并不等同于 GitHub！你完全可以用 Git 对本地项目进行版本控制，并不需要注册任何账号。

- Git 是一个运行在本地的版本控制软件。
- GitHub 则是一个很流行的 Git 仓库托管平台。

Git 很适合用于多人协作的代码开发，它可以很好地解决版本同步和合并。不过，这需要一个所有人可访问的远程仓库，而 GitHub 的作用就是提供一个开放的平台用于创建和托管远程仓库。

许多著名的开源项目都会将它们的仓库托管在 GitHub 上！

为什么需要版本控制？

哈哈，看起来很眼熟？**请使用 Git 来管理你的代码吧**！

- 我尝试修改了一些代码，一切都崩溃了，但刚刚明明都还是好的！
- 我想做几个实验性的修改，但很害怕把项目搞乱而不敢尝试！
- 我想查看上个月的一个问题是如何修复的，但我完全忘记当时改了什么！
- 我和我的合作者用微信来传递彼此的代码，两个人还碰巧改了同一个文件！
- 我将我的合作者的代码缝合至项目后，无法运行了，搞不清到底是谁的问题！

版本控制是体面地进行代码开发不可或缺的工具，即便是本地的个人项目也是如此！

立志成才 报国裕民

Git：在本地仓库使用

Git 安装

Git 可以通过 apt 安装

```
sudo apt install git
```

Git 有时需要一个文本编辑器，因此安装一下 Vim

```
sudo apt install vim
```


立志成才 报国裕民

Git 配置

Git 在第一次创建提交前需要设置用户名和邮箱

```
git config --global user.name "liyuxuan"
git config --global user.email "liyuxuan3003@outlook.com"
```

Git 默认搭配的文本编辑器是 Nano，可以更换为 Vim

```
git config --global core.editor vim
git config --global merge.tool vimdiff
```

Git 创建仓库

Git 仓库（Repository）是 Git 用来管理一个项目的版本历史的核心结构。

若希望对某个项目使用 Git 版本控制，就要在该项目的目录下初始化 Git 仓库，命令是

```
1 git init
```

完成后，当前目录就会出现一个 `.git` 隐藏目录，所有和版本控制相关的数据都会被存储在该目录中。因此，如果想要移除某个项目的版本控制，只需要简单地删除 `.git` 目录即可。

立志成才 报国裕民

Git 提交更改

Git 仓库由以下三个重要部分组成

- **工作区**（Workspace）
- **暂存区**（Index）
- **版本库**（Repository）

工作区就是你能看到的工作目录。当希望创建一个提交（Commit）时[1]，需要两个步骤：首先需要将工作区的改动添加到暂存区（**工作区**→**暂存区**），暂存区提供了一个中间缓冲区，用于存放准备提交的更改。随后需要将暂存区的改动提交到版本库（**暂存区**→**版本库**），形成一个提交。

[1] Git 的版本被称为“提交”，这不涉及到任何远程。

立志成才 报国裕民

Git 提交更改

将改动添加到暂存区（Add），其中 . 代表添加当前目录下的所有更改

```
git add .
```

将改动提交到版本库（Commit），其中选项 -m 用于写入提交的备注

```
git commit -m "fix bugs"
```


立志成才 报国裕民

Git 提交更改

Git 为何要引入一个中间的暂存区的概念？这是因为，Git 期望每一个提交是最小化的，但实践中，代码修改经常是一个连贯的过程，可能同时修改了多处代码。这时，暂存区的意义就体现出来了，使多个修改可以分步提交：先将一处修改暂存后提交，再将另一处修改暂存后提交。

然而，通过命令行完成很精细地暂存是比较麻烦的，推荐在 VSCode 中以图形化方式（侧边栏的 Source Control）交互完成，你可以精细地选择，甚至可以只暂存同一文件中的一处修改！

Git 提交更改

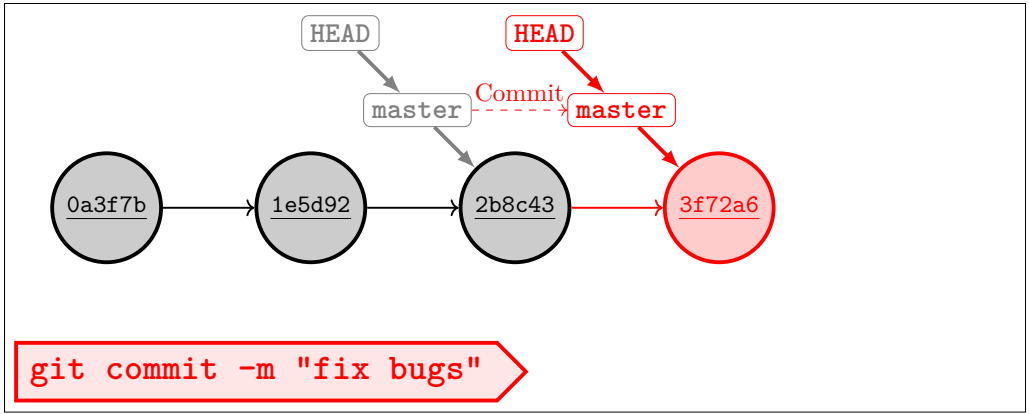
Git 只应当追踪源文件的变化，任何编译输出都应该被忽略

- 编译结果可以自动化产生（如果不能，请让它能），删除后随时可以复现。
- 编译的输出文件的体积往往远远大于源代码，且每次编译后都可能会发生变化。
- 编译的输出文件通常是二进制，而 Git 的增量记录不适用于二进制文件！

这需要在项目的根目录创建一个名为 `.gitignore` 的文件，并写入忽略的内容

```
1 build
```

创建提交


HEAD
HEAD
master
Commit
master
0a3f7b
1e5d92
2b8c43
3f72a6
git commit -m "fix bugs"

Git 添加标签

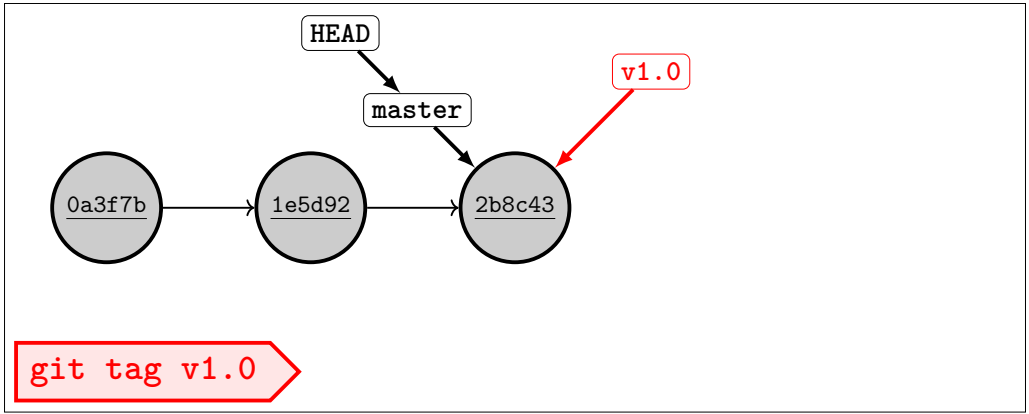
Git 允许对一些重要的版本添加标签，以便后续更容易地找到这个版本

```
git tag v1.0
```

选项 -d 用于删除标签

```
git tag -d v1.0
```

添加标签



Git 查看状态

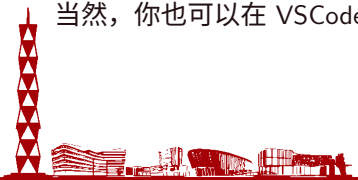
查看当前 Git 仓库的状态和所在分支（若没有更改，会提示“干净的工作区”）

```
git status
```

查看当前 Git 仓库的版本历史，选项 --graph 可以提供更漂亮的输出

```
git log
git log --graph
```

当然，你也可以在 VSCode 的图形界面中更直观地看到这些信息。



Git 分支管理

Git 的版本库从数据结构的角度，可以视为一个有向无环图：

- 每个提交都是图的一个节点。
- 每个提交都会记住其父提交，即上一个提交。
- 提交对应的节点以类似链表的方式组织起整个版本库。

Git 分支的本质，就是指向提交的一个指针。分支指向的提交以及该提交向前溯及的每一个提交，就反映了这个分支的历史。创建多个 Git 分支，就可以使仓库获得多个不同的发展方向。

Git 默认会创建一个称为 `master` 的分支[2]，作为主分支。

[2]由于一些原因，部分平台改用 `main` 取代 `master` 作为主分支名。

Git 分支管理

创建一个分支

```
git branch dev
```

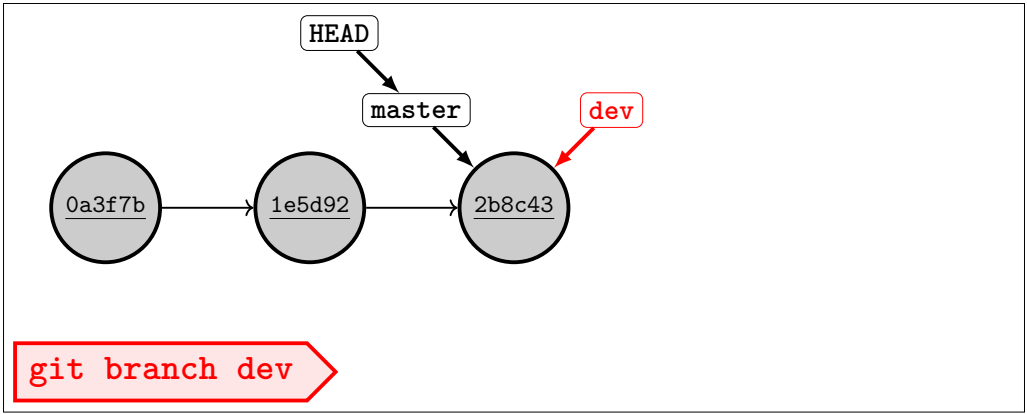
切换至一个分支

```
git checkout dev
```

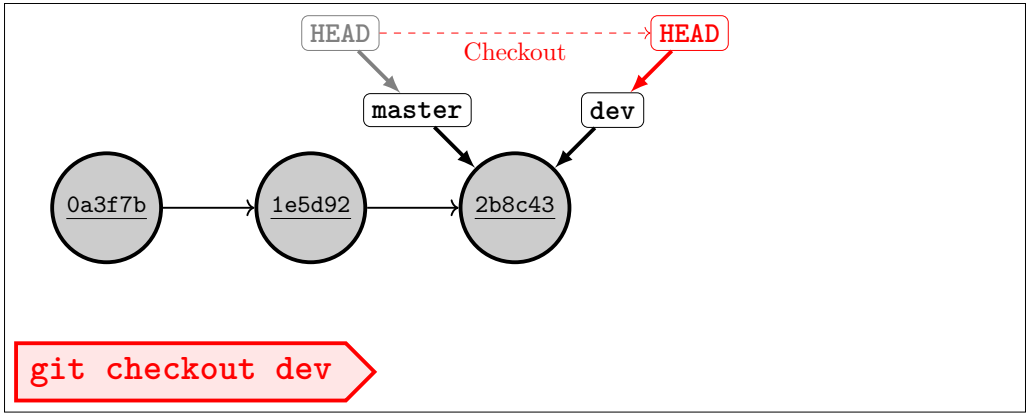
Git 切换分支的本质，就是在切换头指针 HEAD 所指向的分支指针！

通常来说，头指针并不直接指向提交：**头指针一般先指向分支指针，这代表当前所处的分支**。

创建分支



切换分支


HEAD
Checkout
HEAD
master
dev
0a3f7b
1e5d92
2b8c43
git checkout dev

Git 分支管理

查看所有分支

```
git branch
```

有时需要删除分支，可以使用 branch 的 -d 选项

```
git branch -d dev
```

有时需要重命名分支，可以使用 branch 的 -m 选项

```
git branch -m dev test
```


立志成才 报国裕民

Git 分支管理

有时需要创建一个新分支并立即切换，可以使用 `checkout` 的 `-b` 选项

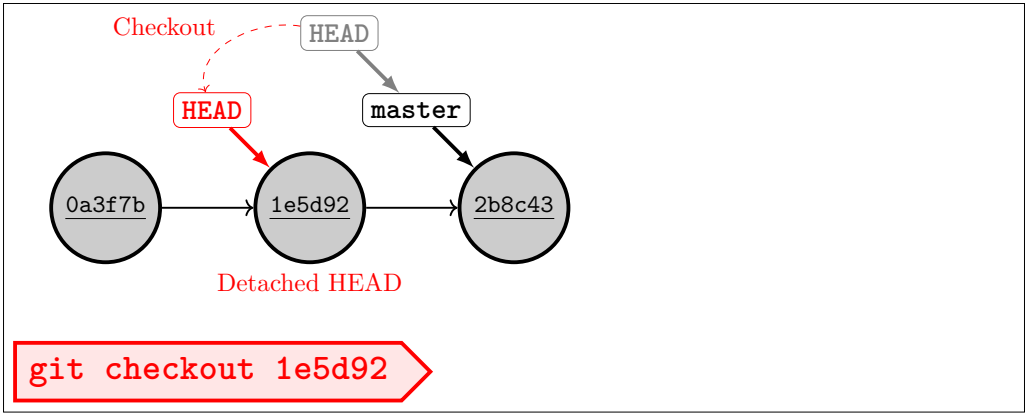
```
git checkout -b dev
```

实际上，直接切换到一个提交而非分支也是被允许的，这种头指针绕过分支指针直接指向提交的特殊状态被称为“**分离的头指针**”，该技巧常用于查看一个特定的历史版本，例如

```
git checkout 1e5d92
```

但请注意，切勿在“分离的头指针”的状态下创建提交！

查看历史版本


Checkout
HEAD
HEAD
master
0a3f7b
1e5d92
2b8c43
Detached HEAD
git checkout 1e5d92

Git 分支合并

Git 允许在若干不同分支上进行不同的开发，**需要时再将各分支的工作合并**。

Git 在合并中能自动处理大部分情况，确有冲突时，会让用户来决定如何解决冲突。

在 `master` 分支上合并 `dev` 分支

```
1 git merge dev
```

Git 分支合并

运行 `git merge` 后，通常可能会出现两种情况

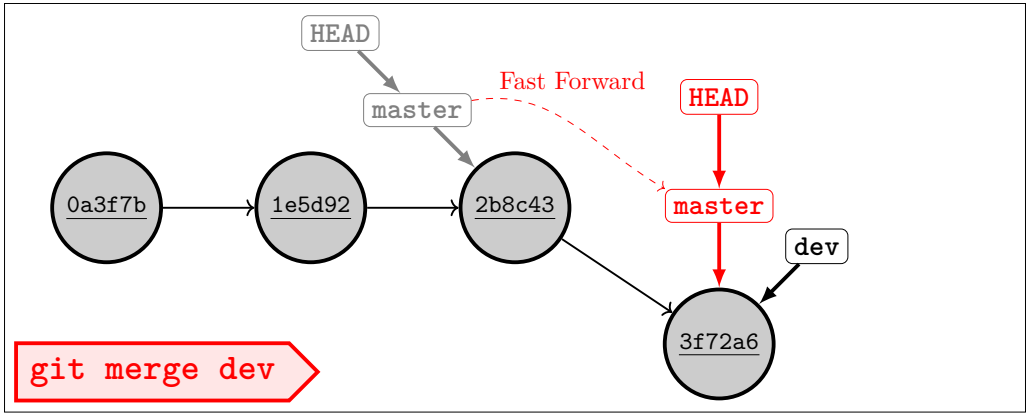
- **快进**（Fast Forward）
- **合并**（Merge）：自动合并 / 手动合并

快进是一种特殊的情形，这适用于 `master` 分支位于 `dev` 分支向前溯及的某个历史提交上。

在这种情况下，只需要将 `master` 分支指针移动到 `dev` 分支指针的位置，就可以完成合并。

立志成才 报国裕民

分支快进


HEAD
master
Fast Forward
HEAD
0a3f7b
1e5d92
2b8c43
master
dev
3f72a6
git merge dev

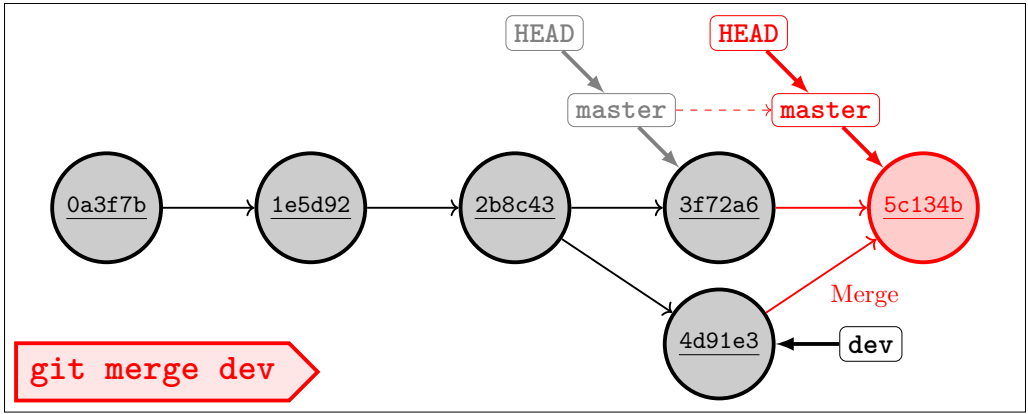
Git 分支合并

自动合并意味着 `master` 和 `dev` 各有新提交，但这些提交的改动并不冲突，例如修改两个文件或修改同一个文件的不同位置，因此 Git 可以自动处理合并。这种情况下 Git 会立即创建一个新提交作为合并结果，这个提交非常特殊，它有两个父提交，可以溯及两段不同的历史。

自动合并的过程中会弹出一个 Vim 界面，这相当于让你输入 `git commit -m` 的提交说明，但是以文件的形式输入。保存文件就可以写入提交说明，默认说明是 `Merge branch 'dev'`。

立志成才 报国裕民

分支合并


HEAD
HEAD
master
master
0a3f7b
1e5d92
2b8c43
3f72a6
5c134b
Merge
4d91e3
dev
git merge dev

Git 分支合并

手动合并意味着 master 和 dev 各自的新提交存在 Git 无法自动处理的冲突，例如修改同一个文件的同一位置。这种情况下，合并仍然会进行，命令行会提示需要手动解决冲突，同时文件中会以一种特殊标记标注所有的冲突点。若使用 VSCode 查看，这类标记会获得特殊高亮。

```
<<<<<<< HEAD
Hi (from master)
=======
Hello World! (from dev)
>>>>>>> dev
```

在逐一解决完所有冲突后，需要手动执行 `git add` 和 `git commit` 创建提交，结束合并窗口。

Git 版本回退

回退至一个特定版本

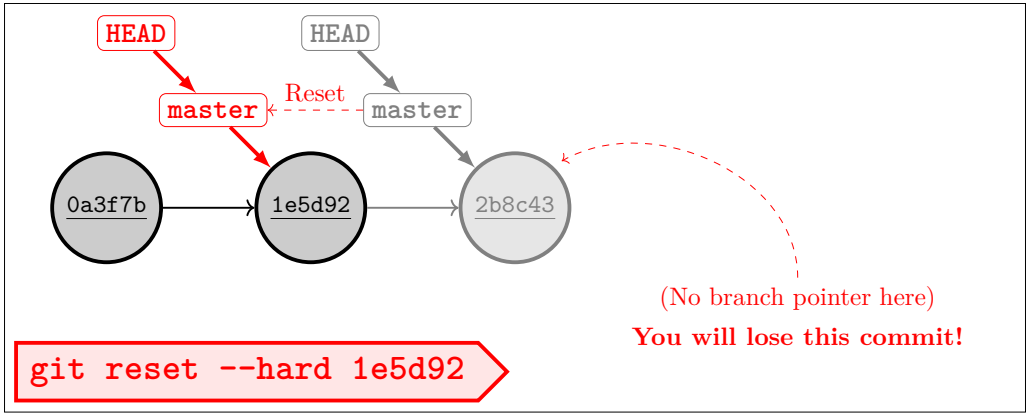
```
git reset --hard 1e5d92
```

请注意，在 Git 中回退版本可能是危险的！你无法回退到一个早期版本然后再重新回到最新版本。因为一个提交能被访问的前提是这条提交链的顶端需要有一个分支指针，若一个提交链的顶端没有分支指针，这个提交链上的所有提交就都消失了（仍存在，但不能用常规方式访问）！

因此，若目的是查看历史版本，请使用 `git checkout` 替代 `git reset --hard`。

若确实需要回退且不希望丢失提交，**请在回退前创建一个额外的分支挂在提交链顶端**。

回退历史版本


HEAD
HEAD
master
Reset
master
0a3f7b
1e5d92
2b8c43
(No branch pointer here)
You will lose this commit!
git reset --hard 1e5d92

Git 版本回退

回退可以用选项指定行为，但通常 `--hard` 可以提供预期中最干净的回退

表: Git 回退的选项

选项	效果
`--soft`	将版本库回退至指定提交。
`--mixed`	将版本库回退，并使暂存区与其保持一致（这是默认选项）。
`--hard`	将版本库回退，并使暂存区和工作区与其保持一致。

Git 版本回退

如果希望撤回刚刚的 `git commit`（例如提交说明写错了）

```
git reset --soft HEAD^
```

如果希望撤回刚刚的 `git commit` 和 `git add`（例如跟踪了不该跟踪的文件）

```
git reset --mixed HEAD^
```

其中 `HEAD^` 代表头指针 `HEAD` 的上一提交。

Git：和远程仓库交互

GitHub 注册与配置

GitHub 是目前最流行的 Git 仓库托管平台[3]，可以通过 GitHub 创建远程仓库。

GitHub 注册现在强制要求启用 2FA 认证，可以选择 Microsoft Authenticator，具体来说，你需要在手机上安装这个应用，它会动态生成密钥，你在某些情况下需要输入这组密钥。推荐立即在 Settings → General 中将默认的主分支名由 `main` 改为 `master`，避免不一致带来的麻烦。

[3] https://www.github.com

GitHub 注册与配置

GitHub 允许通过 HTTPS 和 SSH 两种方式与远程仓库交互，**但你只应当使用 SSH 方式**，连接较为稳定，且免于在每次访问远程时输入密码。为此，你需要先在本地创建一对 SSH 密钥。

SSH 密钥采用 RSA 或 ECC 的加密算法[4]，是一种非对称密钥。

- 密钥是成对的，一把是公钥，一把是私钥。
- 你应当将公钥提供给别人，而小心地保管私钥。
- 用于签名：使用你的私钥加密的数据，都只能用你的公钥解密。
- 用于加密：使用你的公钥加密的数据，都只能用你的私钥解密。

[4] RSA 基于大质数分解，ECC 基于椭圆曲线。

GitHub 注册与配置

用以下命令生成 SSH 密钥。选项 `-t rsa` 代表使用 RSA，选项 `-b 4096` 代表使用 4096 位的密钥，`-C "my-laptop"` 代表密钥的注释，交互中，对每个问题直接按回车保持默认即可。

这会生成公钥 `~/.ssh/id_rsa.pub` 和私钥 `~/.ssh/id_rsa` 两个文件。

```
ssh-keygen -t rsa -b 4096 -C "my-laptop"
```

查看公钥文件

```
cat ~/.ssh/id_rsa.pub
```

复制显示的内容，在 GitHub 的 Settings → SSH and GPG keys 中点击 New SSH Key，将复制的内容粘贴到 Key 中，将 Title 留空（Title 会自动从密钥注释中被识别），随后点击确定就可以了。

Git 克隆远程仓库

你可以将远程仓库克隆到本地

```
git clone git@github.com:liyuxuan3003/ComputerArchitecture.git
```

部分 Git 仓库内会存在 Git 子模块，你需要在 `git clone` 时添加 `--recursive` 选项。

如果在克隆时忘记加上这个选项，可以在克隆后执行以下命令拉取子模块。

```
git submodule init
git submodule update
```

Git 添加远程仓库

你需要首先在 GitHub 上手动创建一个远程仓库，为了后续使用的便利，**不要让它自动生成任何文件**，包括 README 和 LICENSE（这些可以在本地添加），确保这是一个干干净净的空仓库。

添加一个远程仓库

```
1 git remote add origin git@github.com:liyuxuan3003/GitTest.git
```

查看所有远程仓库

```
1 git remote -v
```

远程仓库的名称可以是任意的，但 origin 是默认的远程仓库名称。

Git 添加远程仓库

删除一个远程仓库

```
git remote remove origin
```

修改一个远程仓库的名称

```
git remote rename origin github
```

修改一个远程仓库的目标

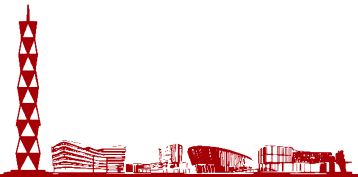
```
git remote set-url origin git@github.com:liyuxuan3003/GitTest26.git
```

Git 使用远程仓库

Git 和远程仓库的交互主要有下面三条命令

- `git push`，将本地分支的提交，**推送**到远程分支，用于上传（Push）。
- `git pull`，将远程分支的提交，**拉取**到本地分支，用于下载（Pull）。
- `git fetch`，**获取**远程分支的状态，用于刷新下载源（Fetch）。

总结：上传用 `git push` 命令，下载用 `git pull` 命令，刷新用 `git fetch` 命令！



Git 使用远程仓库

推送本地分支到远程

```
git push
```

拉取远程分支到本地

```
git pull
```

获取远程分支（但不更新本地分支）

```
git fetch
```


立志成才 报国裕民

Git 使用远程仓库

当一个分支第一次向远程推送时，需要手动关联本地分支和远程分支

```
git push --set-upstream origin master
```

通过下面的配置，可以自动关联远程分支

```
git config --global push.autoSetupRemote true
```


立志成才 报国裕民

Git 使用远程仓库

实际上，`git pull` 可以分解为 `git fetch` 和 `git merge` 两个步骤

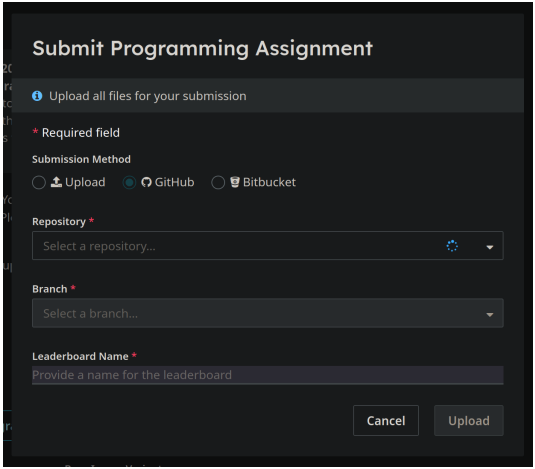
- 第一步，`git fetch` 的目的是刷新远程分支的本地副本 `origin/master`。
- 第二步，`git merge` 的目的是将 `origin/master` 合并到 `master`。

如果远程分支和本地分支间发生了偏离，直接使用 `git pull` 命令将无法完成，其默认只允许快进。这时，你就需要通过 `git fetch` 命令和 `git merge` 命令手动拉取并合并远程分支。

在实践中，请尽量避免这种情况！在多人合作中，每个人应当只在自己的分支上工作（例如命名为 `liyx` 和 `lichf`）。由一个人负责定期合并到 `master` 分支，随后各自再同步 `master` 分支。

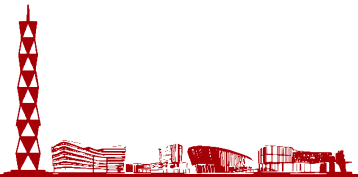
关于 Gradescope 的使用提示

- 在 GitHub 上创建一个**私有**仓库。
- 提交方式选择 GitHub。
- Repository 选择该次作业的仓库。
- Branch 选择用于提交的分支。



总结

欢迎关注我的 GitHub 账号：https://github.com/liyuxuan3003



Thank You