



Git

寒假训练项目-Week1

清华大学自动化系学生科协

主讲人：赵子远

日期：2023.1.18



目录

is empty

- What is Git
- Git的工作原理
- Git本地操作
- Git远程操作
- Bonus Time: About encryption

Review



Linux

Command: pwd、ls、cd、touch、mkdir、rm、cat、echo、chmod

Tools: vim、ssh、tmux、.bashrc

Docker:

Concept: image、container、repository

Command: build、pull、run、images、exec、save、load

Dockerfile、Volume



Git

Story time...



Ben



Ben



Ben



Alice





Ben



Alice



Ben



Alice

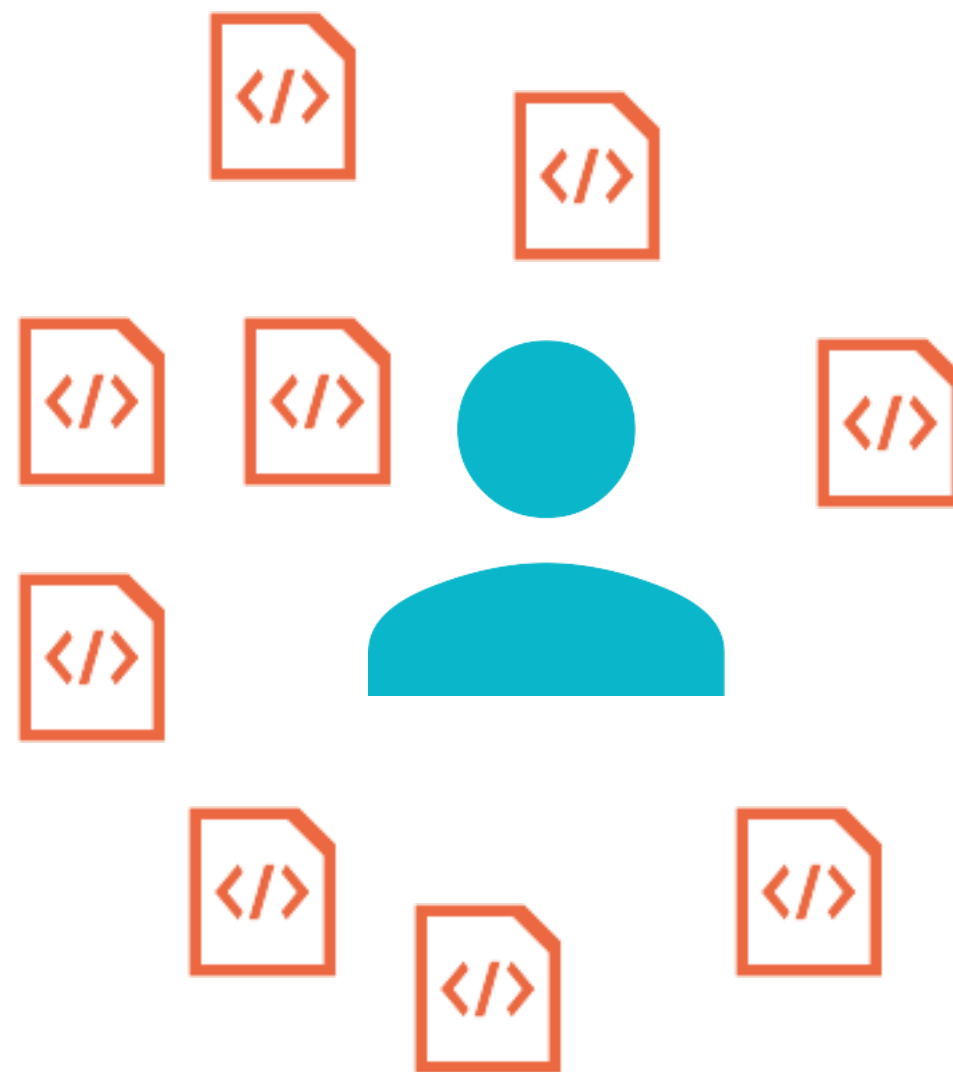
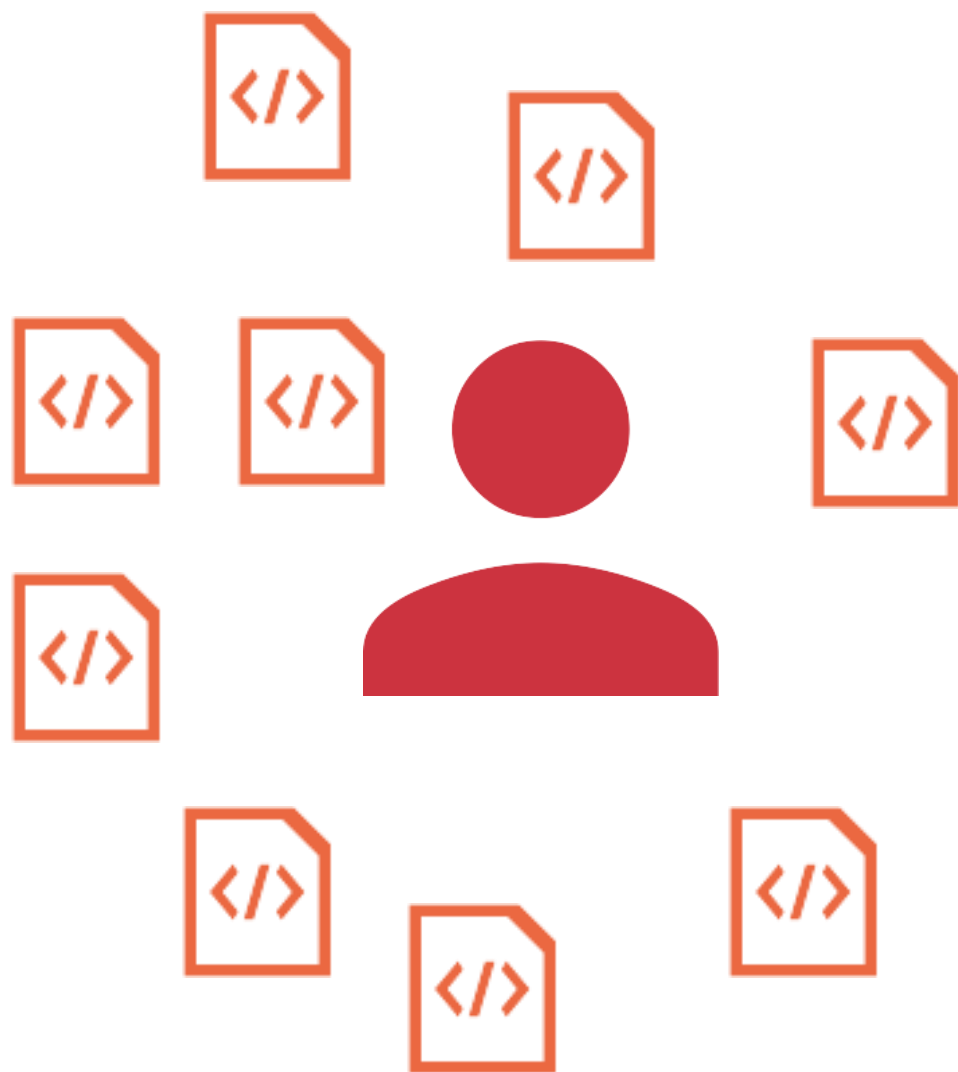


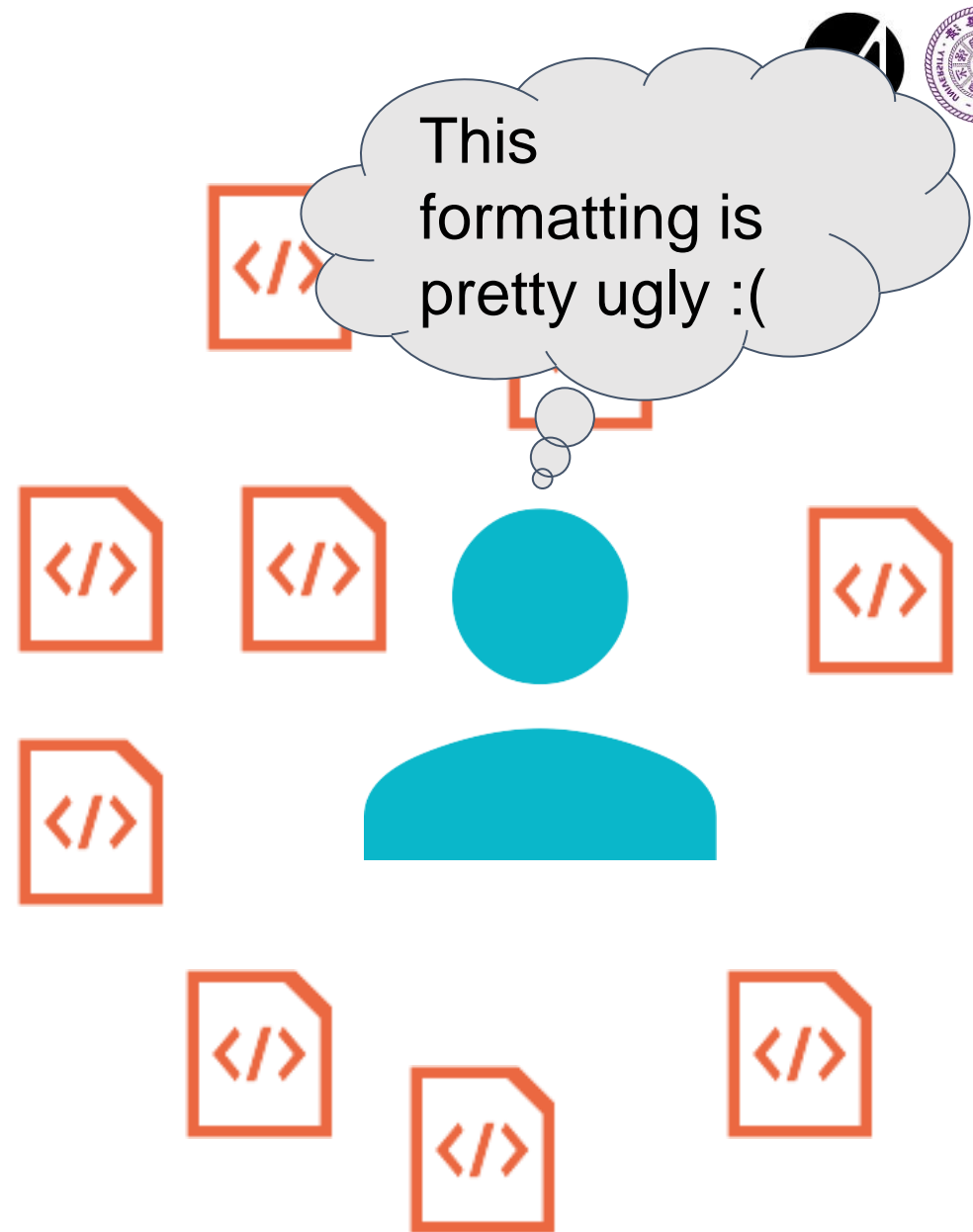


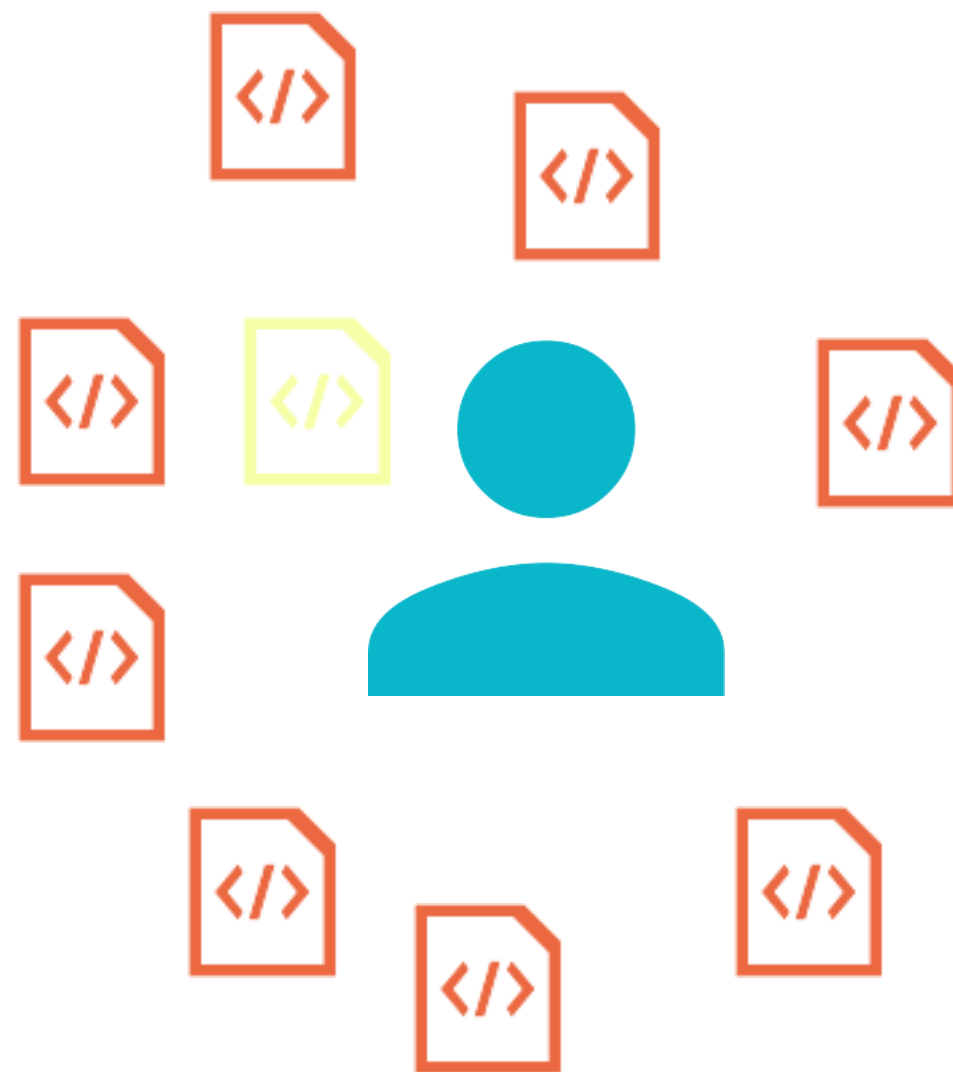
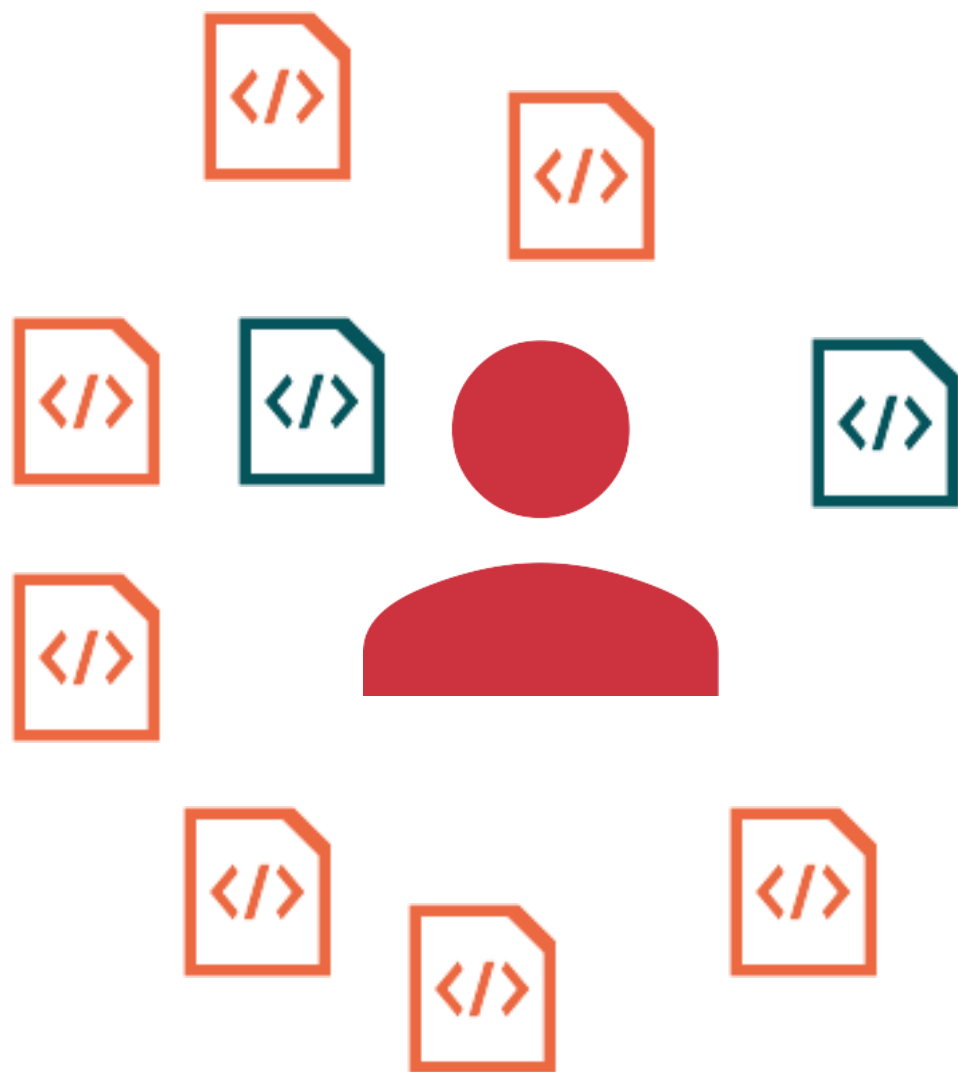
Ben



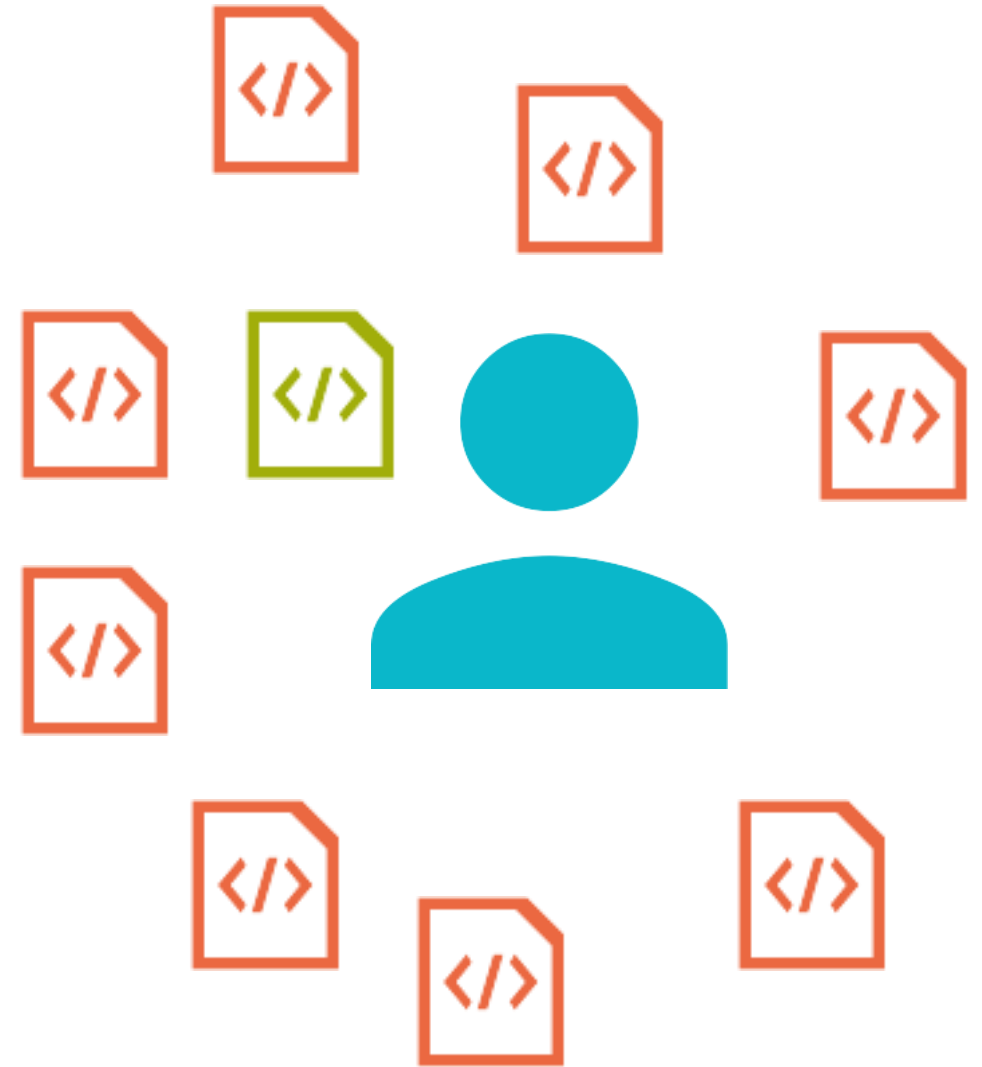
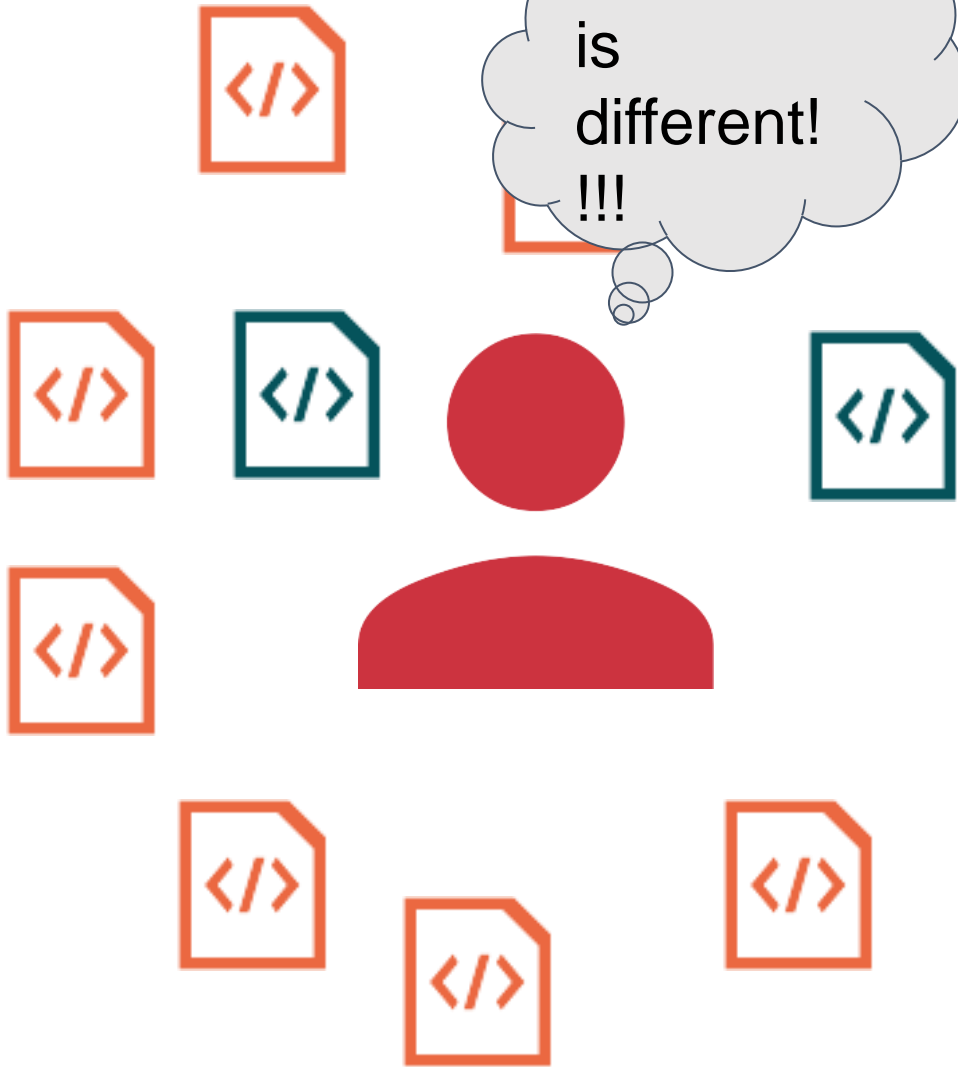
Alice







Our code
is
different!
!!!





What is Git?

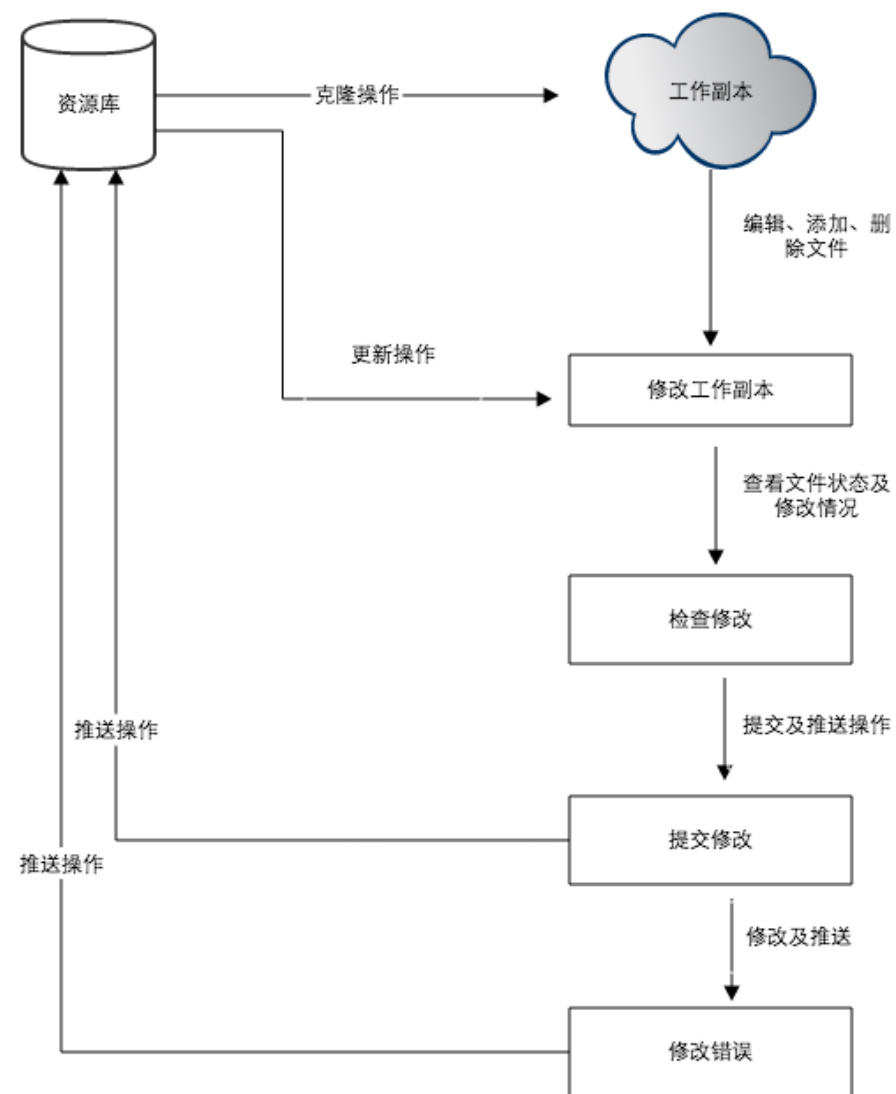


Git是一个开源的**分布式版本控制系统**，可以有效、高速地处理从很小到非常大的项目**版本管理**。这是Linus Torvalds为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。

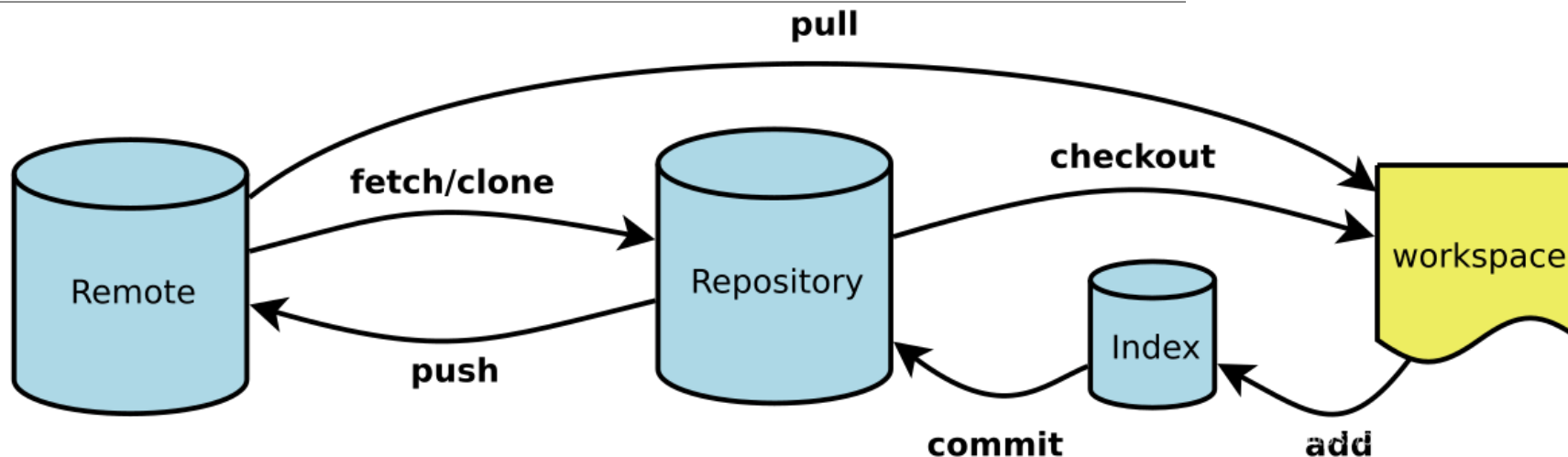
分布式：每个人在本地都拥有一个独立的代码仓库用于管理，各种版本控制的操作都可以在本地完成。

版本管理：对于文件修改的管理。

Git 工作流程



Git的工作原理



- **workspace**: 工作区，存放项目代码的地方。
- **Index/Stage**: 暂存区，用于临时存放你的改动，保存即将提交的文件列表信息。
- **Repository**: 仓库区，存放数据的位置，这里面有你提交到所有版本的数据。其中HEAD指向最新放入仓库的版本。
- **Remote**: 远程仓库，托管代码的远程服务器。

Git的本地操作——安装配置



Windows/MacOS: 官网下载安装。

Linux: `apt-get install git`

配置用户名和邮箱：

```
$ git config --global user.name <username>
$ git config --global user.email <email>
```

仅作为远程提交时的身份显示，无身份验证作用，但是不提倡随意设置，最好与Github的邮箱相同。

查看配置信息：

```
$ git config --list
$ git config user.name
$ git config user.email
```

Git的本地操作——常用操作



初始化仓库：

```
$ git init
```

添加文件到暂存区：

```
$ git add <filename>
```

删除文件：

```
$ git rm <filename> # 从暂存区和工作区删除  
$ git rm --cached <filename> # 只从暂存区中移除
```

查看仓库状态：

```
$ git status
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test  
$ git init  
Initialized empty Git repository in F:/Test/.git/  
  
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)  
$ git add test.py  
  
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)  
$ git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
    new file:   test.py  
  
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)  
$ git rm --cached test.py  
rm 'test.py'  
  
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)  
$ ls  
test.py  
  
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)  
$ git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be  
    test.py  
  
nothing added to commit but untracked files present
```

Git的本地操作——常用操作



比较差异：

```
$ git diff # 暂存区与工作区
$ git diff --cached # 暂存区与上一次提交
$ git diff <first-branch> <second-branch> # 两个分支的差异
```

提交暂存区到仓库：

```
$ git commit -m <message> # After $ git add
```

查看历史提交记录：

```
$ git log
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ echo "import torch" > 2.py

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ echo "import svox" > 3.py

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git add 2.py
warning: in the working copy of '2.py', LF will be
e Git touches it

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git commit -m "test"
[master 0276c38] test
1 file changed, 1 insertion(+)
create mode 100644 2.py
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git add 3.py
warning: in the working copy of '3.py', LF will be
e Git touches it

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git diff --cached
diff --git a/3.py b/3.py
new file mode 100644
index 0000000..d882bdf
--- /dev/null
+++ b/3.py
@@ -0,0 +1 @@
+import svox
```

Git的本地操作——常用操作



回退版本：

```
$ git reset HEAD^ # 回退到上一个版本
$ git reset --mixed <version> # 回退，修改内容进工作区
$ git reset --soft <version> # 回退，修改内容进暂存区
$ git reset --hard <version> # 彻底回退，修改内容清除
```

不知道版本号怎么办？ git log！

PS：回退时只需要版本哈希值的前七位即可

撤销add操作：

```
$ git reset HEAD <filename>
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git log
commit 41aee5981367e9204431b4140c6e8144edcdfef3 (HEAD -> master)
Author: Yuan78 <zhaoyiyu21@mails.tsinghua.edu.cn>
Date:   Wed Jan 18 14:47:55 2023 +0800

    2

commit 0276c3873314ba1746d2e789da5c8d562693ba95
Author: Yuan78 <zhaoyiyu21@mails.tsinghua.edu.cn>
Date:   Wed Jan 18 14:43:54 2023 +0800

    test

commit f0cf67becce3d2751afd234cc86f5427fe6f9b19
Author: Yuan78 <zhaoyiyu21@mails.tsinghua.edu.cn>
Date:   Wed Jan 18 14:42:25 2023 +0800

    first

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git reset --hard 0276c38
HEAD is now at 0276c38 test

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ ls
1.py 2.py test.py
```


Git的本地操作——分支管理



分支管理是实现多版本管理的关键

基础操作：

```
$ git branch <branch-name> # 创建分支
$ git branch -v # 查看所有分支
$ git branch -d <branch-name> # 删除分支
$ git branch -m <old-name> <new-name> #分支改名
$ git checkout <branch-name> # 切换分支
$ git checkout -b <branch-name> # 创建新分支并切换（以当前分支为基础）
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git checkout -b dev
Switched to a new branch 'dev'

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (dev)
$ git branch -v
* dev      0276c38 test
  master  0276c38 test

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (dev)
$ git branch -m dev asta

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (asta)
$ ls
1.py  2.py  test.py
```

Git的本地操作——分支管理



分支合并：

```
$ git merge <branch-name> # 将其他分支合并进本分支
```

分支冲突： 两分支对同一内容做了不同的修改

解决方法： 根据git的提示信息手动修改冲突位置，如VScode等编辑器中可以直接选择保留哪一个分支的内容

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
test
=====
get conflict
>>>>>> test_conf (Incoming Change)
```

ASTA

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (asta)
$ git reset --hard 41aee59
HEAD is now at 41aee59 2

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (asta)
$ git checkout master
Switched to branch 'master'

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git merge asta
Updating 0276c38..41aee59
Fast-forward
 3.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 3.py

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ ls
1.py 2.py 3.py test.py
```

Git的远程操作——托管平台



常用的托管平台

GitHub: <https://github.com/> (裸连丢包严重, 需要科学上网)

Gitee: <https://gitee.com/> (国产平台, 但是代码需要审核, 《开源》)

Tsinghua Gitlab: <https://git.tsinghua.edu.cn/> (和校内同学们合作开发还是很方便的)

身份验证

本地仓库和托管平台之间的身份验证一般是通过**SSH加密**实现的。

使用 (`ssh-keygen -t rsa -C "email"`) 生成SSH Key, 将本地的`.ssh/id_rsa.pub`中的内容复制到GitHub->setting->sshkeys即可。

Git的远程操作——基本操作



绑定远程仓库：

```
$ git remote add <name> <url> # name 通常为 origin  
$ git remote -v # 查看远程仓库地址  
$ git remote rm <name> # 解除绑定
```

拉取远程仓库：

```
$ git clone <url>  
$ git pull origin <branch-name> (:<branch-name>)  
$ git fetch origin <branch-name> (:<branch-name>)
```

推送到远程仓库

```
$ git push origin <branch>
```



```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git remote add origin https://git.tsinghua.edu.cn/zhaoyiyu21/test

23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git remote -v
origin https://git.tsinghua.edu.cn/zhaoyiyu21/test (fetch)
origin https://git.tsinghua.edu.cn/zhaoyiyu21/test (push)
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git pull --rebase origin master
warning: redirecting to https://git.tsinghua.edu.cn/zhaoyiyu21/test.git/
From https://git.tsinghua.edu.cn/zhaoyiyu21/test
 * branch      master      -> FETCH_HEAD
Current branch master is up to date.
```

```
23743@LAPTOPR4U3HLOE MINGW64 /f/Test (master)
$ git push origin master
warning: redirecting to https://git.tsinghua.edu.cn/zhaoyiyu21/test.git/
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
writing objects: 100% (2/2), 225 bytes | 225.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
To https://git.tsinghua.edu.cn/zhaoyiyu21/test
 a472cc3..38d127c  master -> master
```








3
赵子远 authored 26 minutes ago

README

Auto DevOps enabled

Add LICENSE

+

Name	Last commit
 1.py	first
 2.py	test
 3.py	3
 README.md	Initial commit
 test.py	first

Git的远程操作——补充



pull、clone、fetch的区别：

git clone是直接把远程仓库复制到本地，不需要git init；

git fetch是拉取远程分支，只拉取不合并，需要手动merge到本地分支；

git pull = git fetch + git merge，直接将本地分支更新到远程版本。

使用pull和fetch可能会出现**冲突**，原因是远程版本和本地版本对于**同一内容**有**不同修改**，解决方法是拉取到一个新的分支，手动解决冲突后再merge。

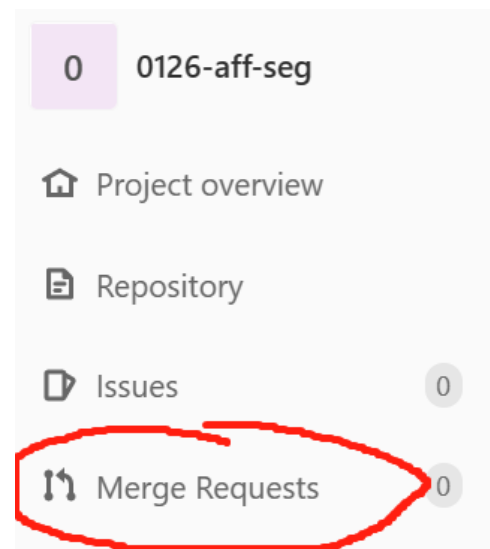
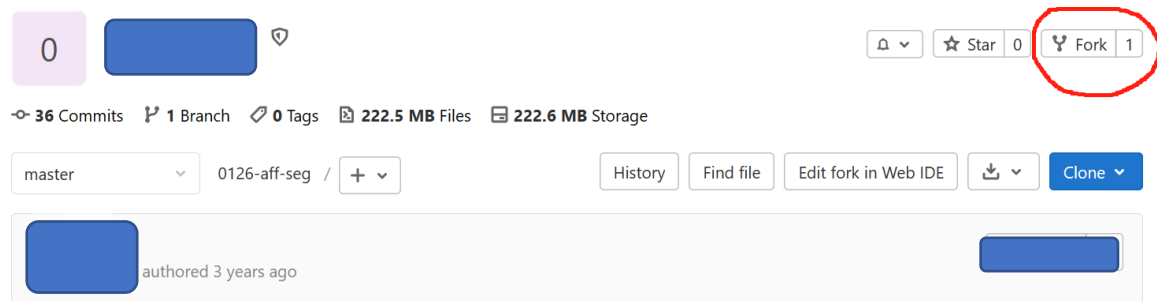
推送到别人的仓库：

别人的仓库是无法直接push的（不能往别人的仓库里随意丢东西），一种方法是让别人把自己加为collaborators，另一种方法则是发起pull request（github中为pull request，gitlab为merge request）。

Git的远程操作——PR



码农界的PR指的当然不是Adobe Premiere，而是github上的重要操作——**Pull Request**，即申请对方拉取自己的代码。当想要向别人的仓库发送代码时，就需要发起PR。



New Merge Request

Source branch

zhaoziyu21/0126-aff-seg master

8aac5513

authored 3 years ago

Target branch

0126-aff-seg master

8aac5513

authored 3 years ago

Compare branches and continue

Git的远程操作——issue



The screenshot shows the GitHub interface for the repository 'openai/gpt-2'. The top navigation bar includes links for Code, Issues (110), Pull requests (32), Actions, Security, and Insights. The 'Issues' tab is selected. The main heading is 'How to use gpt-2 for question answers task? #309'. Below the heading, it says 'Open' and 'ayush431 opened this issue on Nov 28, 2022 · 2 comments'. There are three comments listed:

- ayush431** commented on Nov 28, 2022: 'No description provided.'
- JsNx137** commented on Dec 7, 2022: 'G'
- Ashish-Waykar** commented 3 weeks ago: 'Create virtual environment by :
py- m venv env_name
Run :
Pip install -r requirements.txt
Now after installation packages activate your environment & run the project'

On the right side, there are sections for Assignees (No one assigned), Labels (None yet), Projects (None yet), Milestone (No milestone), and Development (No branches or pull requests). At the bottom right, there is a 'Subscribe' button.

提出bug，记录milestone，甚至作为blog，forum

Git的远程操作——练习



1. 将<https://git.tsinghua.edu.cn/zhaoziyu21/test>的仓库 Fork到自己的仓库
2. 拉取到本地，新建一个dev分支，添加一个含有姓名+学号的文本文件
3. Push到gitlab，向原仓库的master分支发起Merge Request

Gitignore——我不需要提交xxx



在实际开发时，我们并不需要提交所有的文件。例如我们使用Visual Studio写了一个C++程序，项目文件夹会相当大，但是我们其实只需要提交其中的.hpp和.cpp文件即可，这时我们如果直接使用`git add .`，就会提交很多不必要的文件。

Gitignore 的作用是设置文件的忽略规则，让 Git 根据该规则有选择地忽略文件的更改。使用 gitignore 的方式便是在目录中创建名为 `.gitignore` 的文本文件，在该文本文件里编写忽略规则。

Gitignore——我不需要提交xxx



最常见的规则：

直接写上文件名，例如：main.o

重名文件则需要给出目录，例如：./bin/main.o

忽略文件夹：

```
# 忽略所有名为 bin 的文件夹内的文件
bin/
# 忽略当前目录下的 bin 文件夹内的文件
./bin/
```

如果想要进一步了解（通配符，反向操作...）：

语法详解：https://blog.csdn.net/nyist_zxp/article/details/119887324

Gitignore模板：<https://github.com/github/gitignore>

Bonus time——RSA Encryption



原理： RSA加密是一种非对称加密。可以在不直接传递密钥的情况下，完成解密。这能够确保信息的安全性，避免了直接传递密钥所造成的被破解的风险。一对密钥分别称为公钥和私钥，两者之间有数学相关，该加密算法的可靠性是由对一极大整数做因数分解的困难性来保证的。通常个人保存私钥，公钥是公开的（可能同时多人持有）。

简而言之：通过公钥加密的信息只能通过私钥解开。

Bonus time——RSA Encryption



场景设想：一对男女/男男/女女AB在一个匿名的论坛上聊得颇为投机，他们希望**交换联系方式**以进行进一步的交流。

实现过程：A生成一对密钥，将公钥发在论坛；B用公钥加密自己的联系方式后发出；A用私钥对信息解密，得到了B的联系方式。在此过程中其他人无法得到B的联系方式。

然而这样能够确保安全吗？

Bonus time——RSA Encryption



场景设想：一位单身者C，对于他人成双成对的行为十分反感，希望能够拆尽天下有情人。C看到了帖子，希望能够破坏AB的联系。

实现过程：C用A的公钥加密了自己的联系方式，并在传输途中替换了B的信息。于是A实际上得到了C的联系方式。A在添加好友后受到了C的尖锐的言语攻击，从此对于网络社交陷入深深的失望.....

如何让AB二人得到HE?

Bonus time——RSA Encryption



场景设想： A决定把自己的联系方式发给B，并且希望这个联系方式是能够自证真实性的。

实现过程： A用私钥将信息加密，将密文与信息一起发出（这个密文称为签名），B用A的公钥解出密文后发现结果与信息相同，证明信息没有被篡改。这里的原理是公钥与私钥作用上是对称的，其他人即使篡改信息也无法完成加密。

但这时论坛上的人都得到了A的联系方式，C召唤水军对A进行了网暴。

Bonus time——RSA Encryption



场景设想：AB两人再次尝试传递联系方式。

实现过程：A和B都有一套自己的公钥和私钥，当A要给B发送消息时，先用B的公钥对消息加密，再对加密的消息使用A的私钥加签名，从而达到既不泄露也不被篡改的效果。

终于两人成功传递了联系方式，有情人终成眷属.....吗？

如果公钥也不能保证真实性怎么办？如果坏心眼的C可能会在传输过程中把A或B的公钥替换成自己的公钥？

Bonus time——RSA Encryption



公钥认证机构 (CA)： 实体(个人、网站、路由器)向CA注册其公钥，并向CA提供“身份证明”。CA创建证书，将E与E的公钥绑定。证书包含由CA进行了数字签名的E的公钥：CA可以证明“这是E的公钥”

实现过程： A和B都可以通过CA获得对方的真实公钥，两人终于能够免于C的打扰完成信息交换了。

Happy Ending!



谢谢大家