

Git&GitHub

1 版本控制工具应该具备的功能

- 协同修改
 - 多人并行不悖的修改服务器端的同一个文件。
- 数据备份
 - 不仅保存目录和文件的当前状态，还能够保存每一个提交过的历史状态。
- 版本管理
 - 在保存每一个版本的文件信息的时候要做到不保存重复数据，以节约存储空间，提高运行效率。这方面 SVN 采用的是增量式管理的方式，而 Git 采取了文件系统快照的方式。
- 权限控制
 - 对团队中参与开发的人员进行权限控制。
 - 对团队外开发者贡献的代码进行审核——Git 独有。
- 历史记录
 - 查看修改人、修改时间、修改内容、日志信息。
 - 将本地文件恢复到某一个历史状态。
- 分支管理
 - 允许开发团队在工作过程中多条生产线同时推进任务，进一步提高效率。

2 版本控制简介

2.1 版本控制

工程设计领域中使用版本控制管理工程蓝图的设计过程。在 IT 开发过程中也可以使用版本控制思想管理代码的版本迭代。

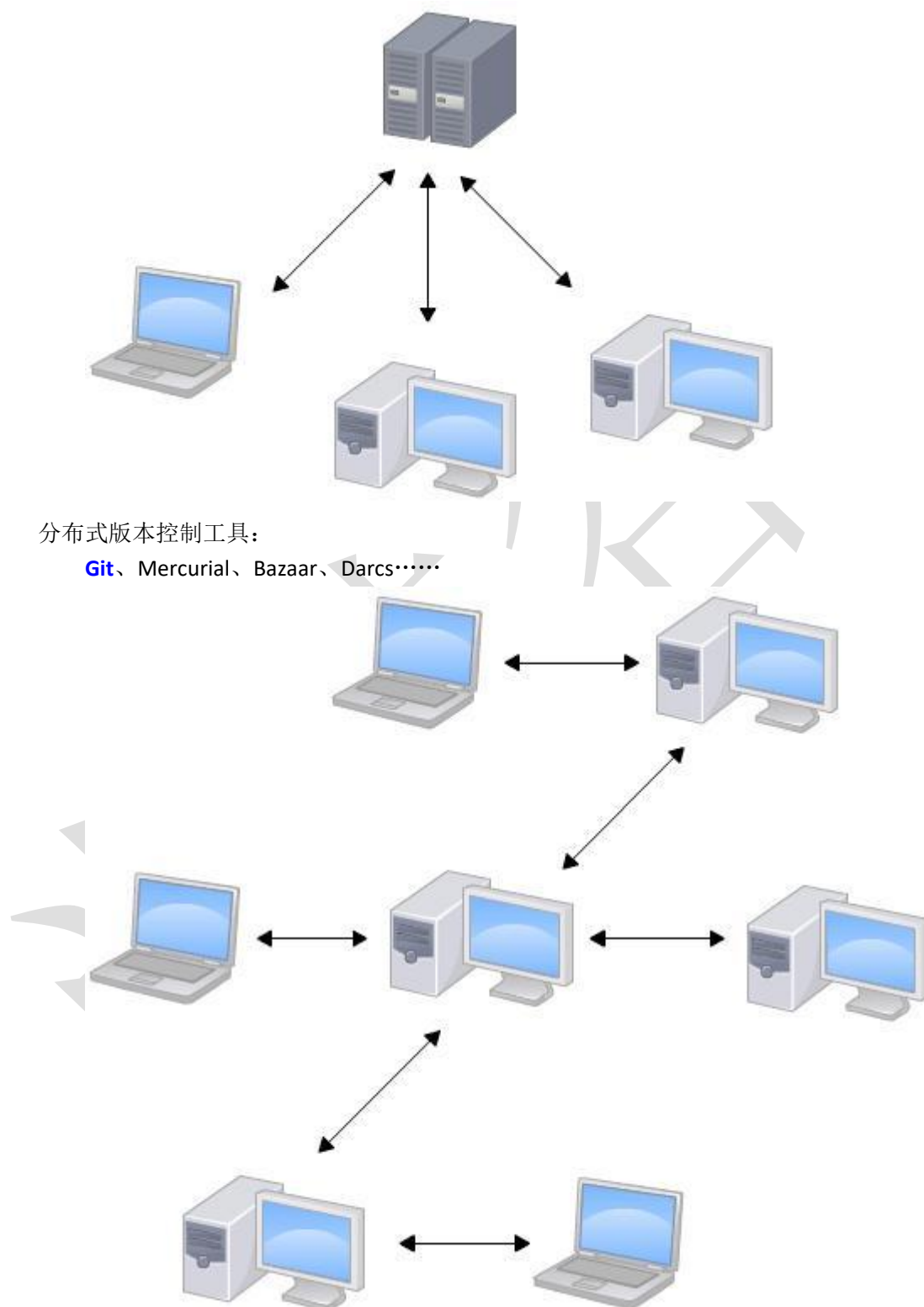
2.2 版本控制工具

思想：版本控制

实现：版本控制工具

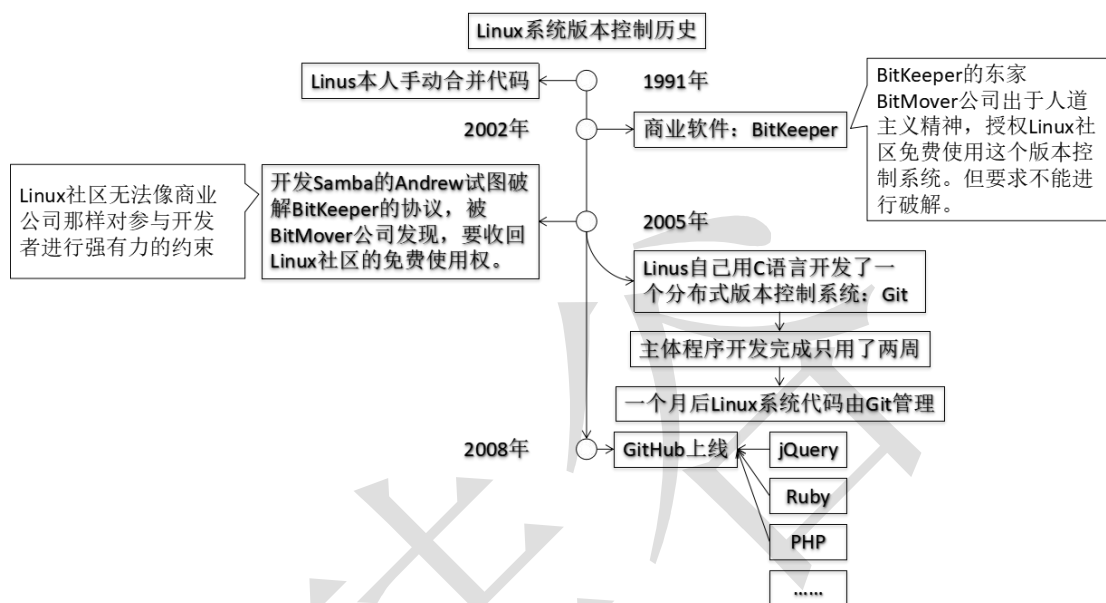
集中式版本控制工具：

CVS、SVN、VSS……



3 Git 简介

3.1 Git 简史



3.2 Git 官网和 Logo

官网地址：<https://git-scm.com/>

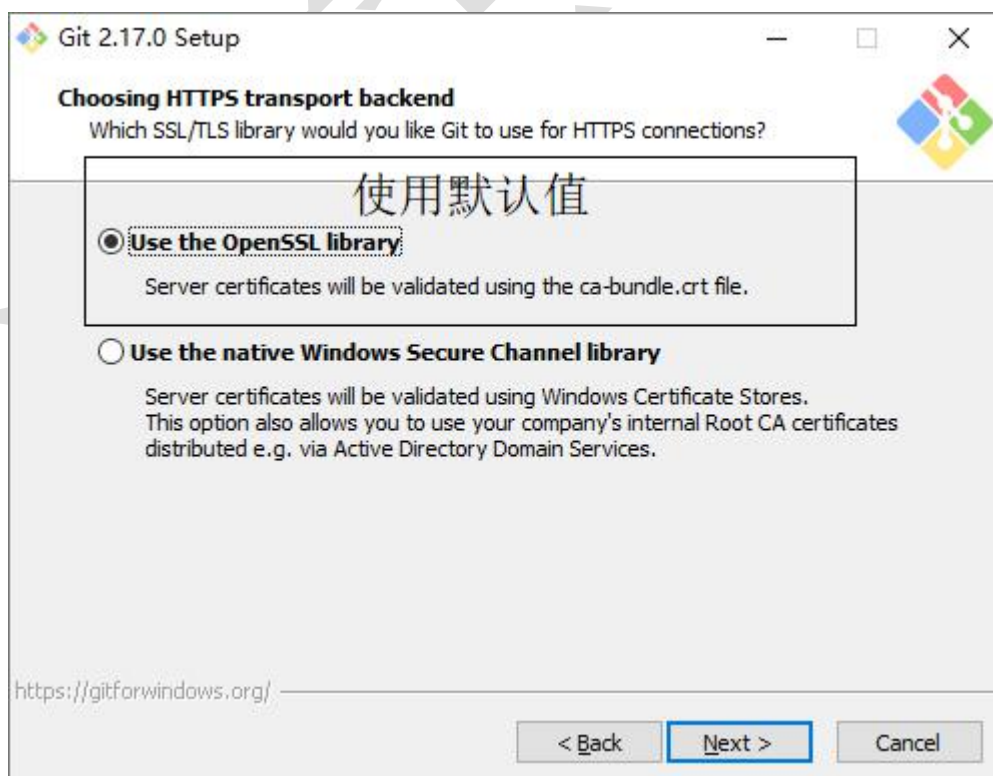


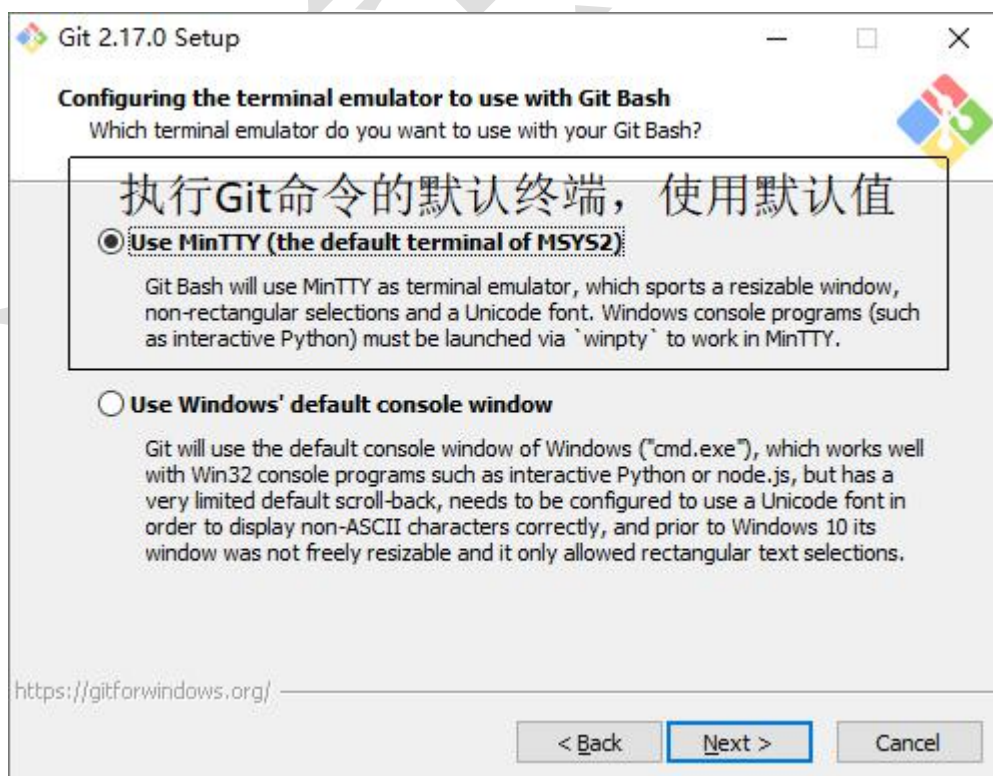
3.3 Git 的优势

- 大部分操作在本地完成，不需要联网
- 完整性保证
- 尽可能添加数据而不是删除或修改数据
- 分支操作非常快捷流畅
- 与 Linux 命令全面兼容

3.4 Git 安装











3.5 Git 结构



3.6 Git 和代码托管中心

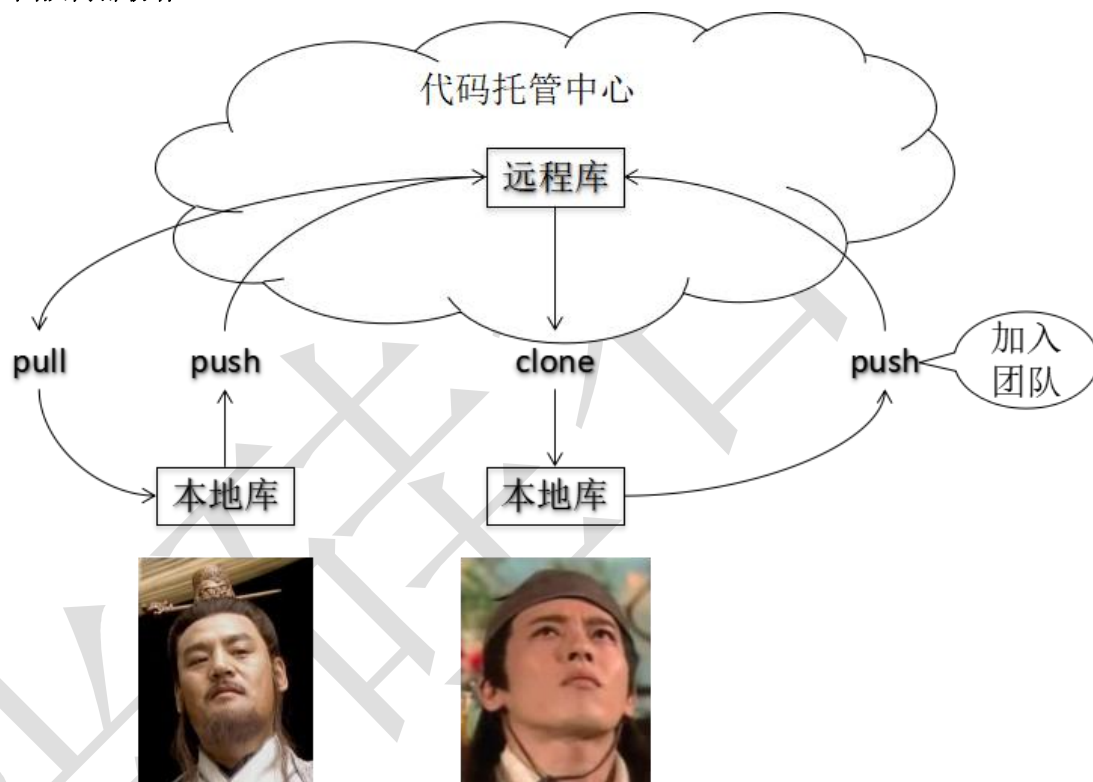
代码托管中心的任务：维护远程库

- 局域网环境下

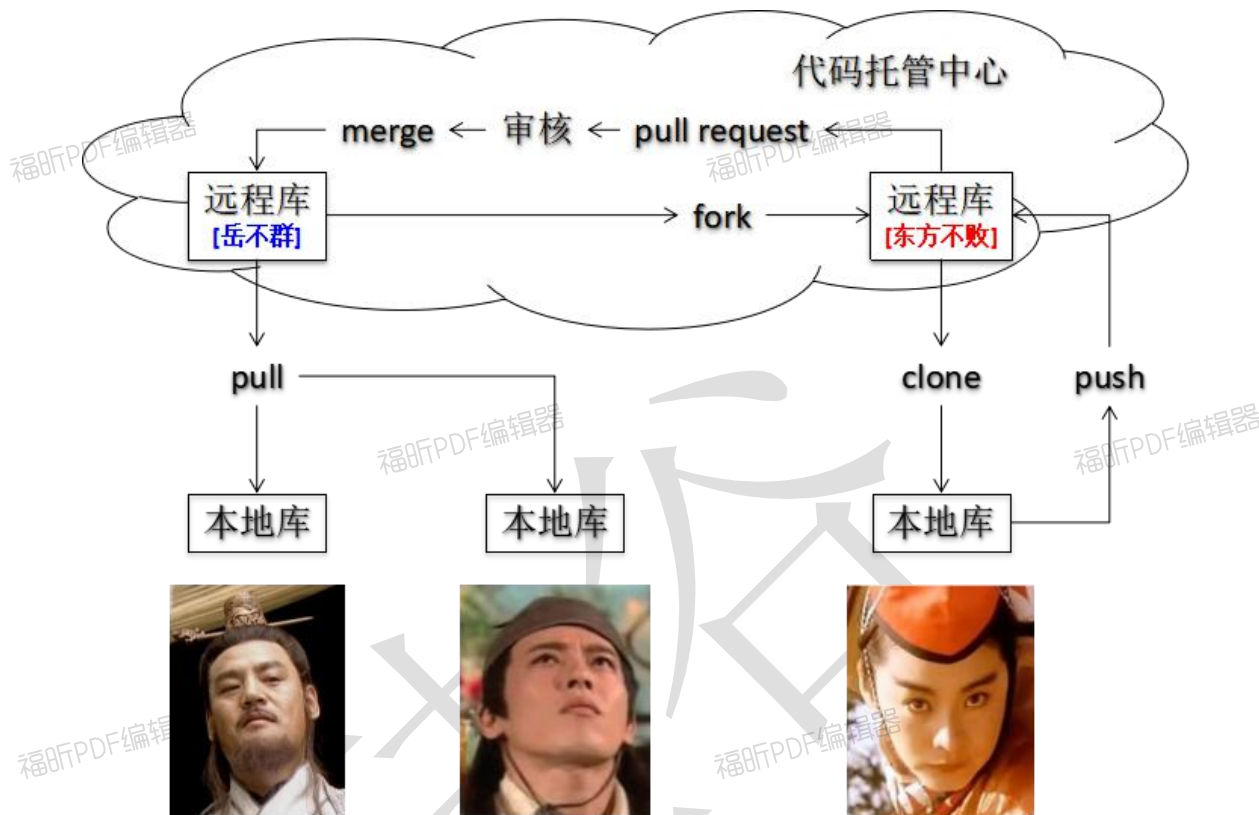
- GitLab 服务器
- 外网环境下
 - GitHub
 - 码云

3.7 本地库和远程库

3.7.1 团队内部协作



3.7.2 跨团队协作



4 Git 命令行操作

4.1 本地库初始化

- 命令：git init
- 效果

```
$ ll .git/
total 7
-rw-r--r-- 1 Lenovo 197121 130 5月 10 16:53 config
-rw-r--r-- 1 Lenovo 197121 73 5月 10 16:53 description
-rw-r--r-- 1 Lenovo 197121 23 5月 10 16:53 HEAD
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 hooks/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 info/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 objects/
drwxr-xr-x 1 Lenovo 197121 0 5月 10 16:53 refs/
```

- 注意：.git 目录中存放的是本地库相关的子目录和文件，不要删除，也不要胡乱修改。

4.2 设置签名

- 形式

用户名: tom

Email 地址: goodMorning@atguigu.com

- 作用: 区分不同开发人员的身份
- 辨析: 这里设置的签名和登录远程库(代码托管中心)的账号、密码没有任何关系。
- 命令
 - 项目级别/仓库级别: 仅在当前本地库范围内有效
 - ◆ git config user.name tom_pro
 - ◆ git config user.email goodMorning_pro@atguigu.com
 - ◆ 信息保存位置: `./.git/config` 文件

```
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true

[user]
    name = tom_pro
    email = goodMorning_pro@atguigu.com
```

- 系统用户级别: 登录当前操作系统的用户范围
 - ◆ git config --global user.name tom_glb
 - ◆ git config --global goodMorning_pro@atguigu.com
 - ◆ 信息保存位置: `~/.gitconfig` 文件

```
$ cat ~/.gitconfig
[user]
    name = tom_glb
    email = goodMorning_glb@atguigu.com
```

- 级别优先级
 - ◆ 就近原则: 项目级别优先于系统用户级别, 二者都有时采用项目级别的签名
 - ◆ 如果只有系统用户级别的签名, 就以系统用户级别的签名为准
 - ◆ 二者都没有不允许

4.3 基本操作

4.3.1 状态查看

git status

查看工作区、暂存区状态

4.3.2 添加

`git add [file name]`

将工作区的“新建/修改”添加到暂存区

4.3.3 提交

`git commit -m "commit message" [file name]`

将暂存区的内容提交到本地库

:set nu

合并后提交 `git commit -m "`

将文件夹提交到暂存区：`git add 文件夹名`

将暂存区的所有文件提交：`git commit -m &`

使用后如果出现行号，说明是vm编辑器

i：进入编辑模式；点击esc，输入:wq，则保存并退出

4.3.4 查看历史记录

`git log`

```
commit e5fee992adab620d433129318e85545f6376f7dc
Author: tom_pro <goodMorning_pro@atguigu.com>
Date:   Fri May 11 14:53:40 2018 +0800

insert nnnnnnnnn edit
```

多屏显示控制方式：

空格向下翻页

b 向上翻页

q 退出

`git log --pretty=oneline`

```
64011afc1b5fbdd35db8d55e52f824bdf819dee2 (HEAD -> master) insert qqqqqqqq edit
526eb78bd21618dda77a366edbc65a6e99ad63d7 insert pppppp edit
0a6fc6c266c2b7a6bc8839725344c93cce83db0c insert oooooo edit
```

只显示过去的记录（当前+当前记录之前的记录）

`git log --oneline`

```
64011af (HEAD -> master) insert qqqqqqqq edit
526eb78 insert pppppp edit
0a6fc6c insert oooooo edit
```

`git reflog` 显示所有的记录

```
64011af (HEAD -> master) HEAD@{0}: commit: insert qqqqqqqq edit
526eb78 HEAD@{1}: commit: insert pppppp edit
0a6fc6c HEAD@{2}: commit: insert oooooo edit
e5fee99 HEAD@{3}: commit: insert nnnnnnnnn edit
```

HEAD@{移动到当前版本需要多少步}

4.3.5 前进后退

➤ 本质

```

HEAD → fd83eb9 (HEAD -> master) HEAD@{0}: commit: insert qqqqqqqq edit
7bf0e31 HEAD@{1}: commit: insert pppppp edit
2679109 HEAD@{2}: commit: insert oooooo edit
9a9ebe0 HEAD@{3}: commit: insert nnnnnnnnn edit
49f1bd3 HEAD@{4}: commit: insert mmmmmmmm edit
a6ace91 HEAD@{5}: commit: insert llllllll edit
3dd95d7 HEAD@{6}: commit: insert kkkkkkkkk edit
42e7e84 HEAD@{7}: commit: insert jjjjjjjj edit
7c265b1 HEAD@{8}: commit: insert iiiiiiii
c309b92 HEAD@{9}: commit: insert hhhhhh edit
7305cd8 HEAD@{10}: commit: insert gggggggg edit
ede116d HEAD@{11}: commit: insert fffffff edit
6325c55 HEAD@{12}: commit: insert eeeeeee edit
a709ad9 HEAD@{13}: commit: for test history
bfb79b7 HEAD@{14}: commit: My second commit, modify good.txt
ac5c801 HEAD@{15}: commit (initial): My first commit. new file good.txt
    
```

- 基于索引值操作[推荐]
 - `git reset --hard [局部索引值]`
 - `git reset --hard a6ace91`
- 使用^符号：只能后退
 - `git reset --hard HEAD^`
 - 注：一个^表示后退一步，n个表示后退n步
- 使用~符号：只能后退
 - `git reset --hard HEAD~n`
 - 注：表示后退n步

4.3.6 reset 命令的三个参数对比

- `--soft` 参数
 - 仅仅在本地库移动 HEAD 指针



- `--mixed` 参数
 - 在本地库移动 HEAD 指针
 - 重置暂存区



- `--hard` 参数
 - 在本地库移动 HEAD 指针
 - 重置暂存区
 - 重置工作区

4.3.7 删除文件并找回

- 前提：删除前，文件存在时的状态提交到了本地库。
- 操作：git reset --hard [指针位置]
 - 删除操作已经提交到本地库：指针位置指向历史记录
 - 删除操作尚未提交到本地库：指针位置使用 HEAD

二者的区别：提交到本地库之后，执行git relog后有那条删除记录，我们需要将head指向他的前面的那条创建的记录的局部索引值。只提交到暂存区的话，git relog中只含有那条创建的记录，也是让head指向那条创建的记录的索引值

4.3.8 比较文件差异

- git diff [文件名]
 - 将工作区中的文件和暂存区进行比较
- git diff [本地库中历史版本] [文件名]
 - 将工作区中的文件和本地库历史记录比较
- 不带文件名比较多个文件

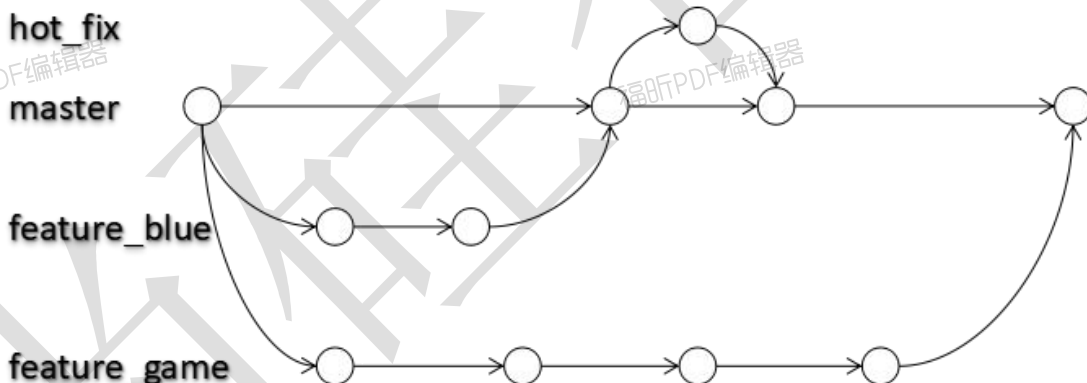
git diff HEAD [文件名]

- 和本地库的文件进行比较

4.4 分支管理

4.4.1 什么是分支？

在版本控制过程中，使用多条线同时推进多个任务。



4.4.2 分支的好处？

- 同时并行推进多个功能开发，提高开发效率
- 各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响。失败的分支删除重新开始即可。

4.4.3 分支操作

- 创建分支
 - git branch [分支名]
- 查看分支
 - git branch -v
- 切换分支
 - git checkout [分支名]
- 合并分支
 - 第一步：切换到接受修改的分支（被合并，增加新内容）上
 - git checkout [被合并分支名]

- 第二步：执行 merge 命令
git merge [有新内容分支名]

➢ 解决冲突

- 冲突的表现

```

8 gggggggg
9 <<<<<< HEAD
10 hhhhhhhh edit by hot_fix
11 =====
12 hhhhhhhh edit by master
13 >>>>>> master
14 iiiiiiii
15 jjjjjjjj
    
```

当前分支内容

另一分支内容

- 冲突的解决
 - ◆ 第一步：编辑文件，删除特殊符号
 - ◆ 第二步：把文件修改到满意的程度，保存退出
 - ◆ 第三步：git add [文件名]
 - ◆ 第四步：git commit -m "日志信息"
 - 注意：此时 commit 一定不能带具体文件名

5 Git 基本原理

5.1 哈希

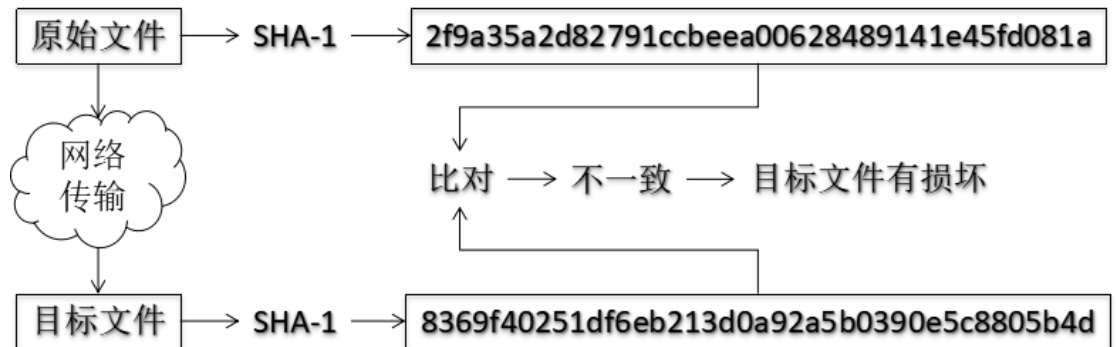


哈希是一个系列的加密算法，各个不同的哈希算法虽然加密强度不同，但是有以下几个共同点：

- ① 不管输入数据的数据量有多大，输入同一个哈希算法，得到的加密结果长度固定。
- ② 哈希算法确定，输入数据确定，输出数据能够保证不变
- ③ 哈希算法确定，输入数据有变化，输出数据一定有变化，而且通常变化很大
- ④ 哈希算法不可逆

Git 底层采用的是 SHA-1 算法。

哈希算法可以被用来验证文件。原理如下图所示：

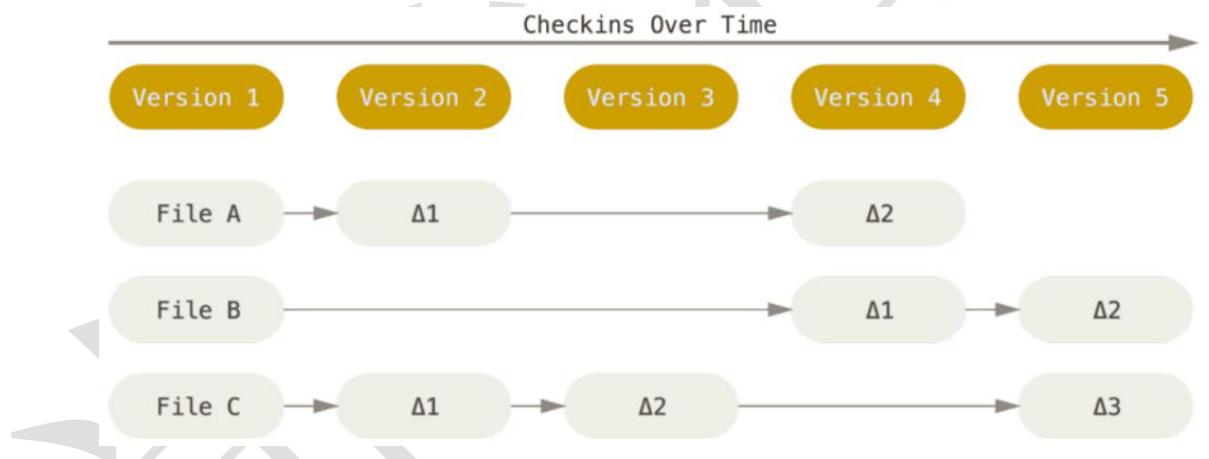


Git 就是靠这种机制来从根本上保证数据完整性的。

5.2 Git 保存版本的机制

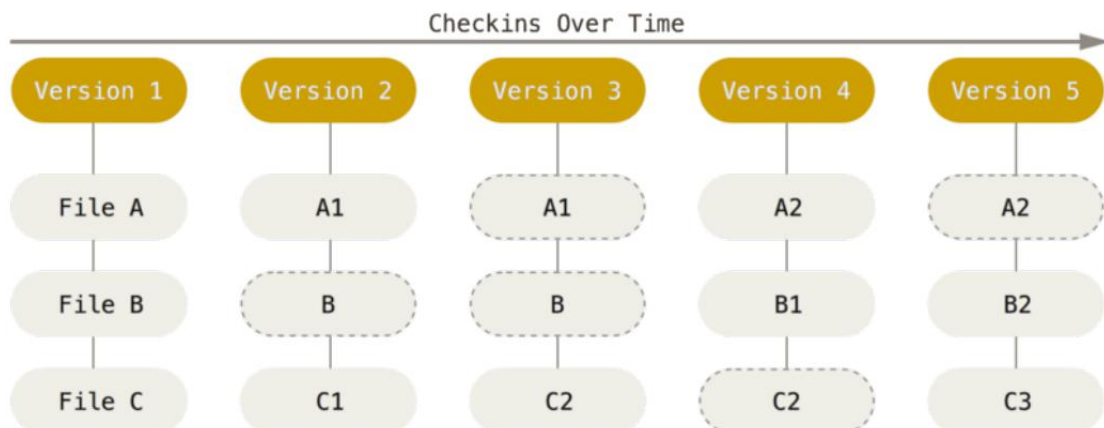
5.2.1 集中式版本控制工具的文件管理机制

以文件变更列表的方式存储信息。这类系统将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。



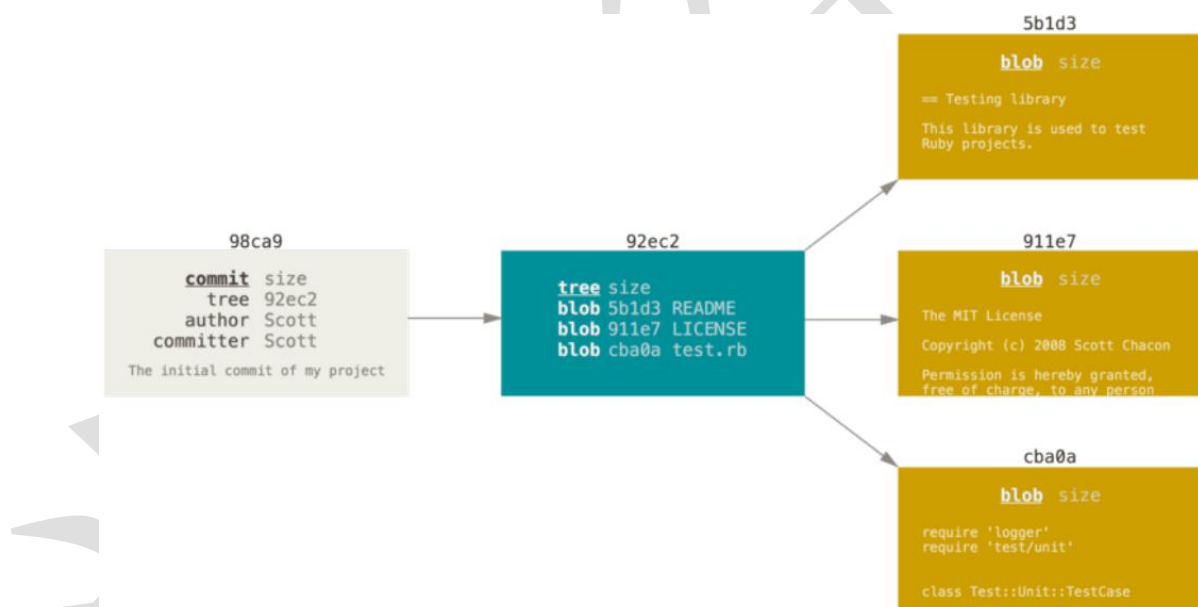
5.2.2 Git 的文件管理机制

Git 把数据看作是小型文件系统的一组快照。每次提交更新时 Git 都会对当前的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。所以 Git 的工作方式可以称之为快照流。

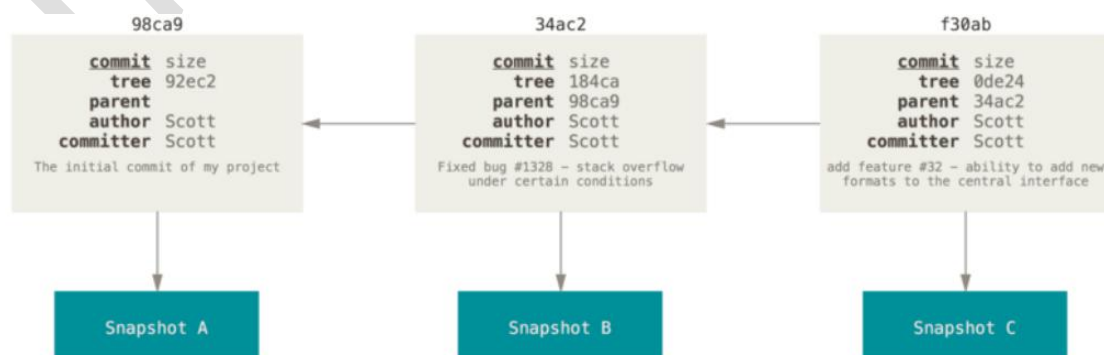


5.2.3 Git 文件管理机制细节

➤ Git 的“提交对象”

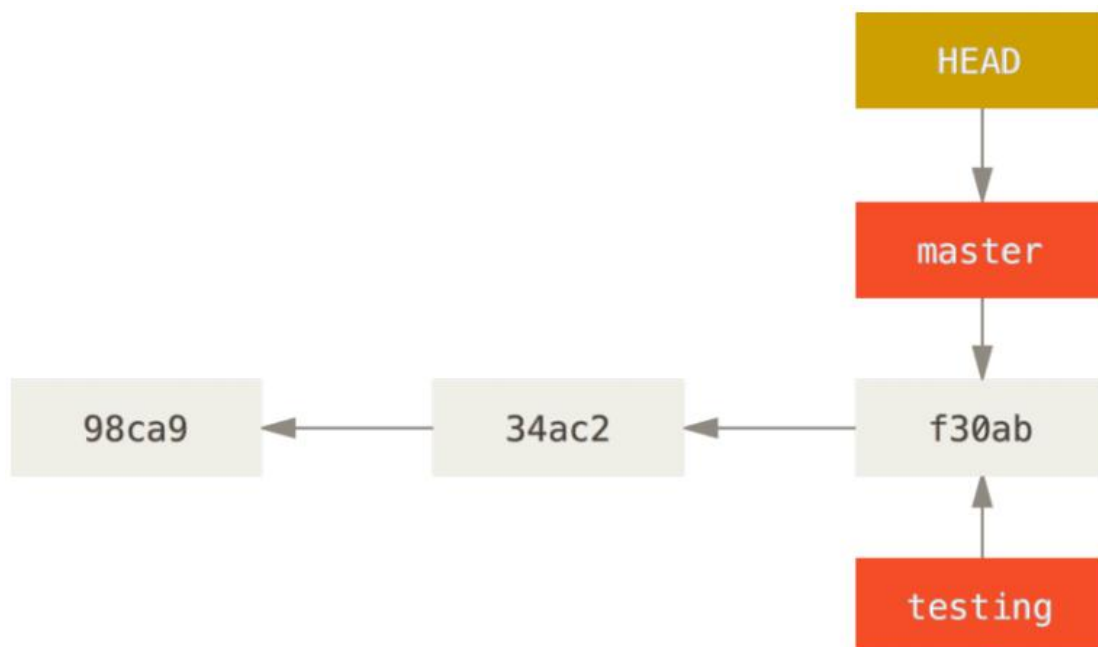


➤ 提交对象及其父对象形成的链条

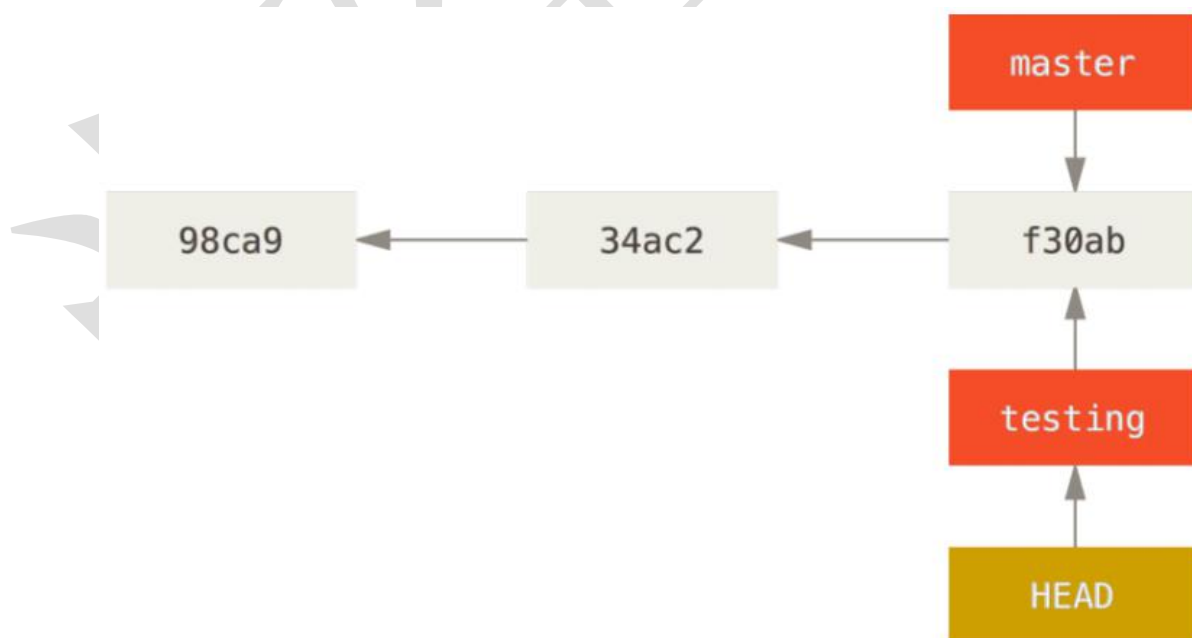


5.3 Git 分支管理机制

5.3.1 分支的创建



5.3.2 分支的切换





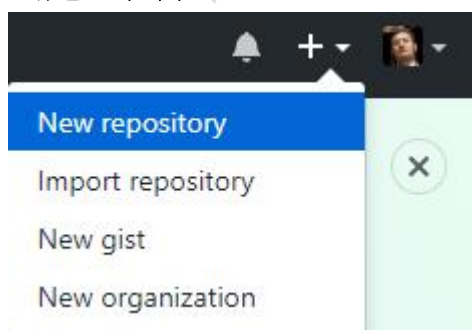
6 GitHub

6.1 账号信息

GitHub 首页就是注册页面: <https://github.com/>

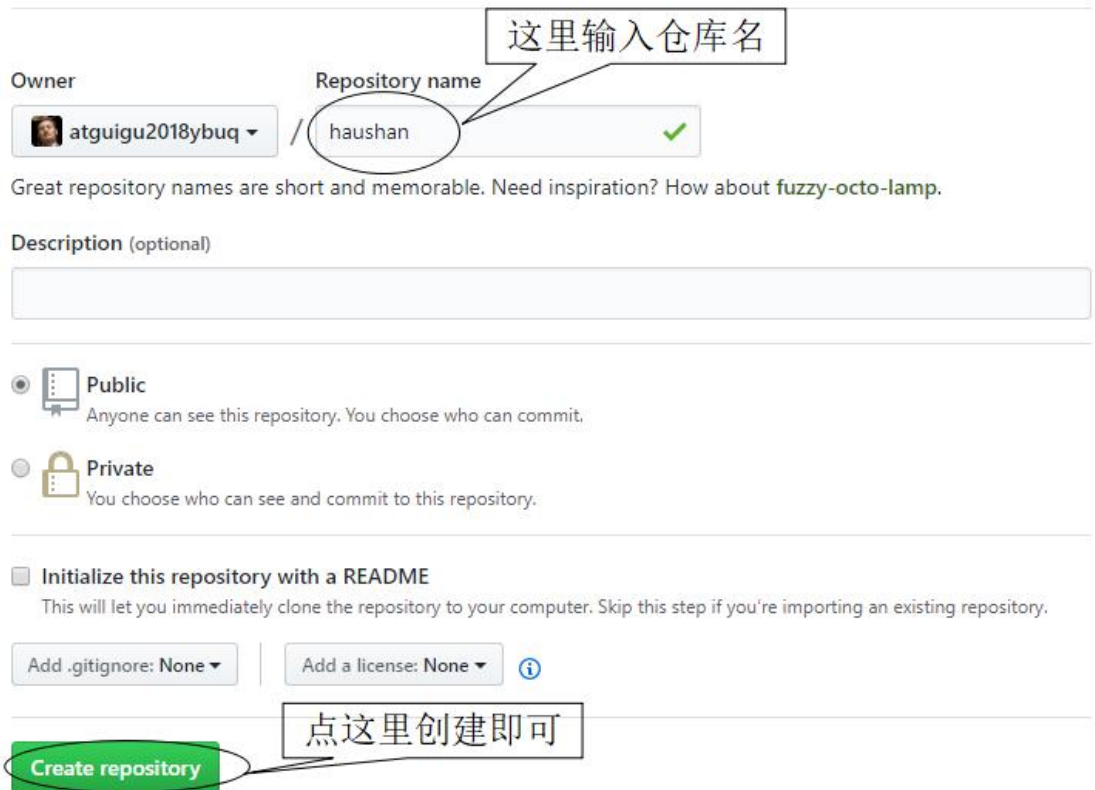
	Email 地址: atguigu2018ybuq@aliyun.com GitHub 账号: atguigu2018ybuq
	Email 地址: atguigu2018lhuc@aliyun.com GitHub 账号: atguigu2018lhuc
	Email 地址: atguigu2018east@aliyun.com GitHub 账号: atguigu2018east


6.2 创建远程库



Create a new repository

A repository contains all the files for your project, including the revision history.



Owner:  atguigu2018ybuq

Repository name:

Great repository names are short and memorable. Need inspiration? How about `fuzzy-octo-lamp`.

Description (optional):

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: | Add a license: ⓘ

6.3 创建远程库地址别名

`git remote -v` 查看当前所有远程地址别名

`git remote add [别名] [远程地址]`

```
$ git remote add origin https://github.com/atguigu2018ybuq/haushan.git
Lenovo@DESKTOP-SAV98C0 MINGW64 /d/workspaces/GitSpaceVideo/haushan (master)
$ git remote -v
origin https://github.com/atguigu2018ybuq/haushan.git (fetch)
origin https://github.com/atguigu2018ybuq/haushan.git (push)
```

6.4 推送

`git push [别名] [分支名]`

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 247 bytes | 247.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/atguigu2018ybuq/haushan.git
* [new branch]      master -> master
```

6.5 克隆

➤ 命令

■ git clone [远程地址]

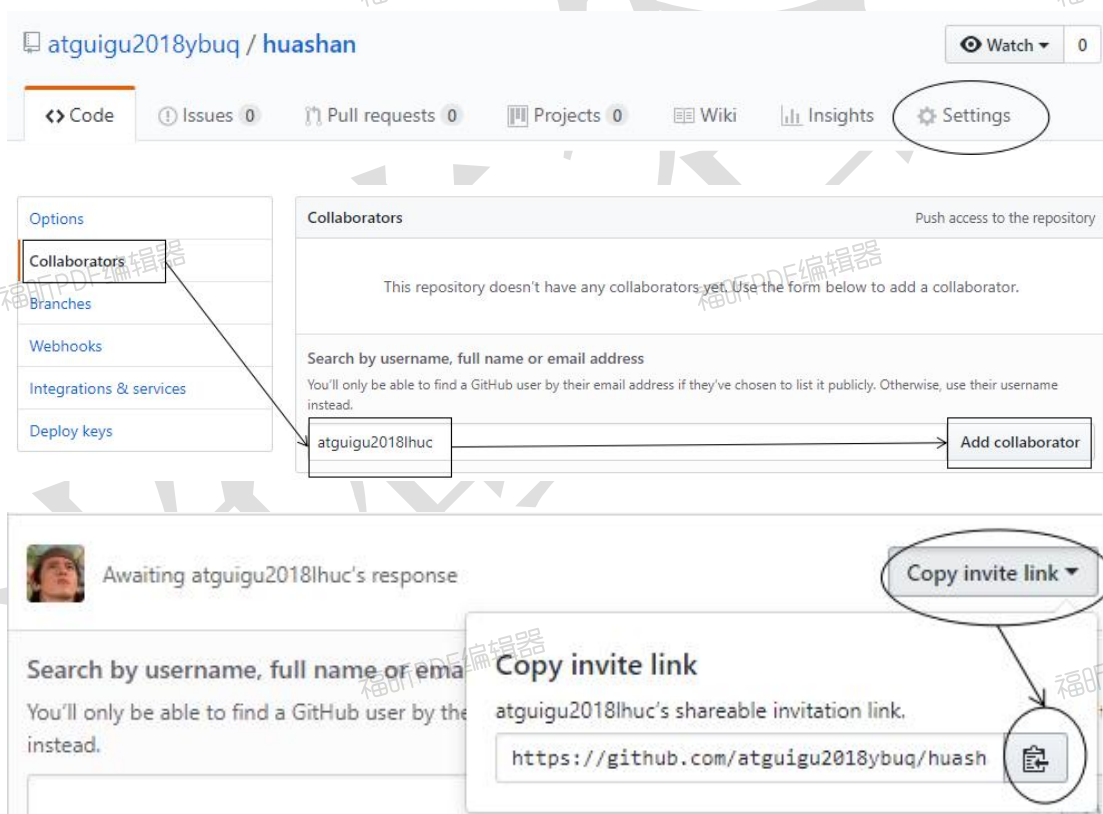
```
$ git clone https://github.com/atguigu2018ybuq/huashan.git
Cloning into 'huashan'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 10 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), done.
```

➤ 效果

- 完整的把远程库下载到本地
- 创建 origin 远程地址别名
- 初始化本地库

3、进入一个文件夹，通过git clone 远程地址的方式将仓库进行了克隆，我们需要cd 进入此仓库，才可进行下面的操作。

6.6 团队成员邀请



“岳不群”其他方式把邀请链接发送给“令狐冲”，“令狐冲”登录自己的 GitHub 账号，访问邀请链接。



atguigu2018ybuq invited you to collaborate

Accept invitation

Decline

6.7 拉取

- pull=fetch+merge
- git fetch [远程库地址别名] [远程分支名]
- git merge [远程库地址别名/远程分支名]
- git pull [远程库地址别名] [远程分支名]

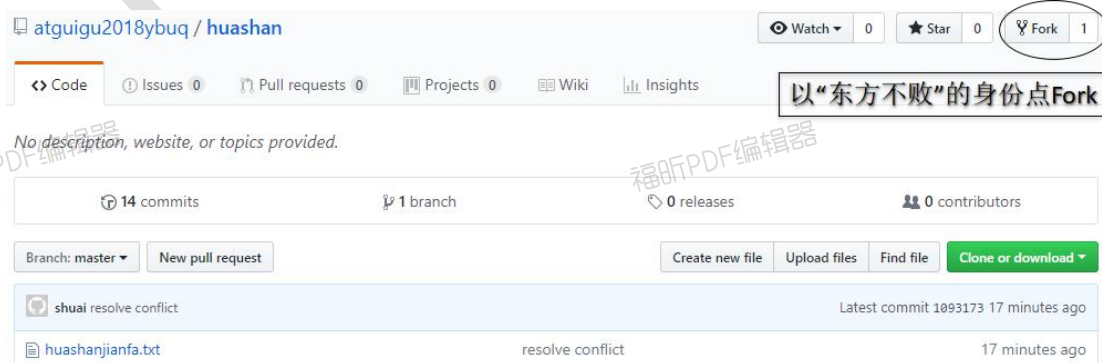
fetch一次便会把他人提交的内容下载下来，然后本人再根据情况决定是否merge合并。
但是fetch后，原本的分支master并不会改变，我们需要切换到origin/master分支才可以查看。
合并时，要切换到原本的分支，合并拉取下来的那个分支origin/master

6.8 解决冲突

- 要点
 - 如果不是基于 GitHub 远程库的最新版所做的修改，不能推送，必须先拉取。
 - 拉取下来后如果进入冲突状态，则按照“分支冲突解决”操作解决即可。
- 类比
 - 债权人：老王
 - 债务人：小刘
 - 老王说：10 天后归还。小刘接受，双方达成一致。
 - 老王媳妇说：5 天后归还。小刘不能接受。老王媳妇需要找老王确认后再执行。

6.9 跨团队协作

- Fork



Forking atguigu2018ybuq/huashan

It should only take a few seconds.

Refresh

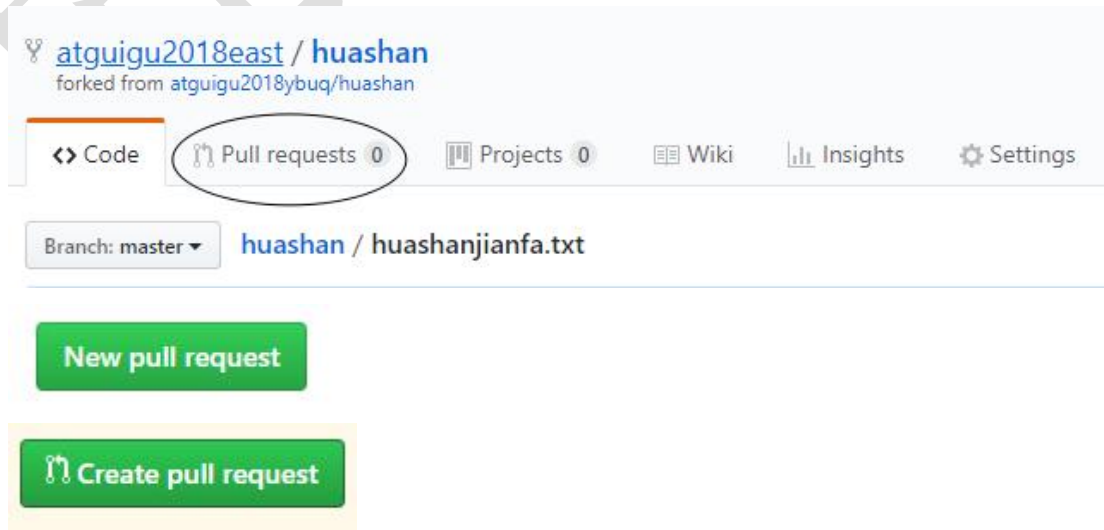


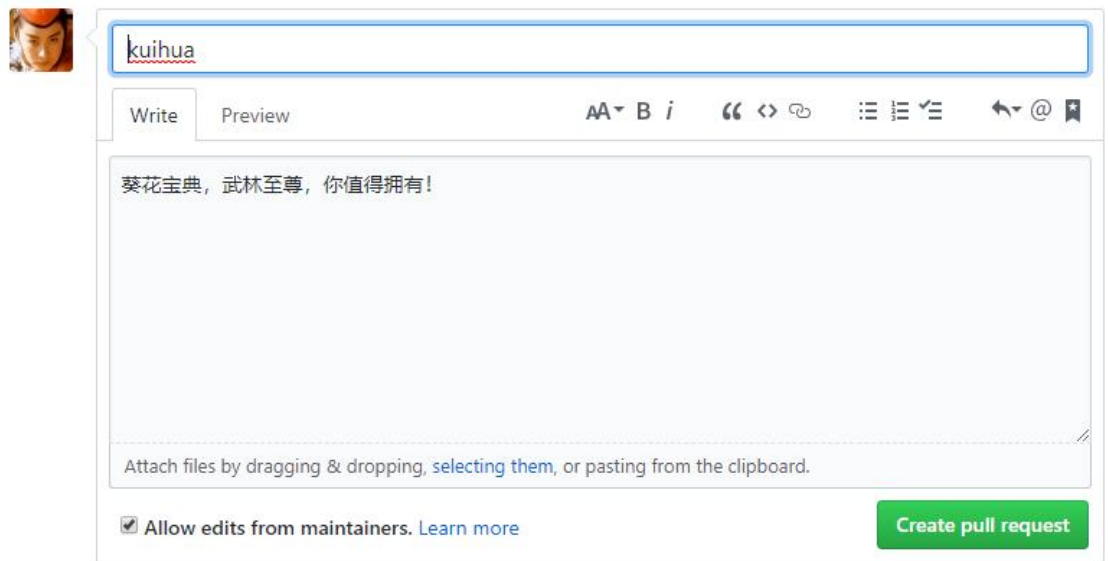
说明当前仓库的所有人是：东方不败

atguigu2018east / huashan
forked from atguigu2018ybuq/huashan

说明了fork的来源

- 本地修改，然后推送到远程
- Pull Request





➤ 对话





atguigu2018east commented 4 minutes ago

葵花宝典，武林至尊，你值得拥有！



kuihua



atguigu2018ybuq commented 12 seconds ago

老东，你这代码靠谱吗？练了会不会有什么危险？



atguigu2018east commented just now

你放心吧，我都亲自练过啦！

➤ 审核代码

 Conversation 2
  Commits 1
  Checks 0
  Files changed 1

Changes from all commits ▾ Jump to... ▾ +1 -0 ■■■■■■

1 ■■■■■■ huashanjianfa.txt



@@ -2,3 +2,4 @@

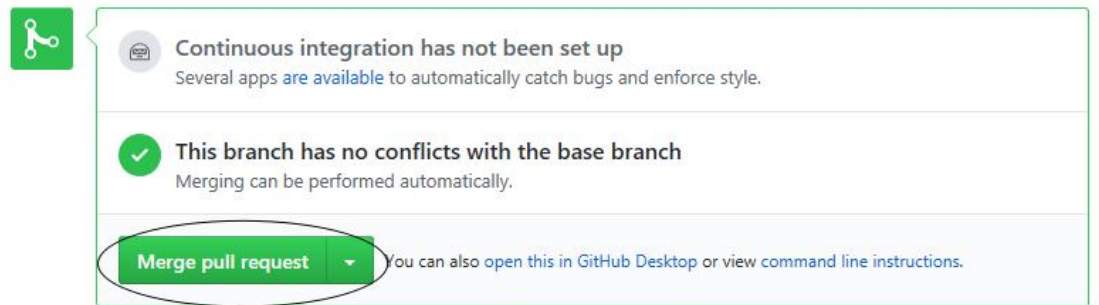
2 2 我是令狐冲，我比岳不群还厉害！ edit by lhuc

3 3 我是令狐冲，我比岳不群还厉害！ edit by ybuq

4 4 我会独孤九剑，哈哈，你不会！

5 +小葵花课堂开讲啦！孩子葵花宝典老练不好怎么办？多半是装的，打一顿就好啦！

➤ 合并代码

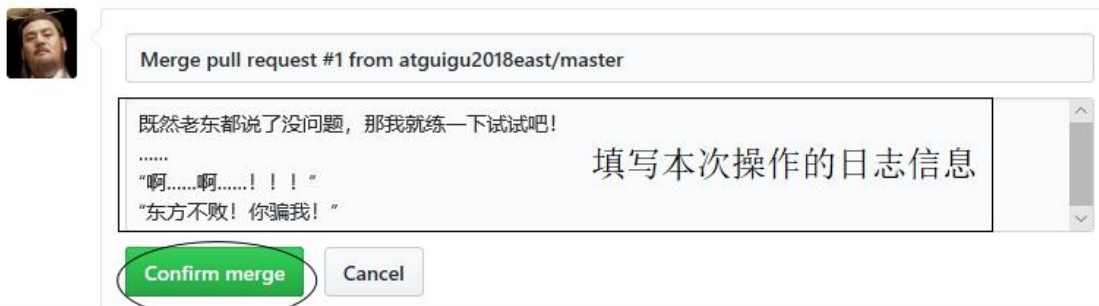


点这里合并代码

Add more commits by pushing to the **master** branch on **atguigu2018east/huashan**.



Add more commits by pushing to the **master** branch on **atguigu2018east/huashan**.



- 将远程库修改拉取到本地

6.10SSH 登录

- 进入当前用户的家目录
\$ cd ~
- 删除.ssh 目录
\$ rm -rvf .ssh
- 运行命令生成.ssh 密钥目录
\$ ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com
[注意：这里-c 这个参数是大写的 C]
- 进入.ssh 目录查看文件列表
\$ cd .ssh
\$ ls -lF
- 查看 id_rsa.pub 文件内容

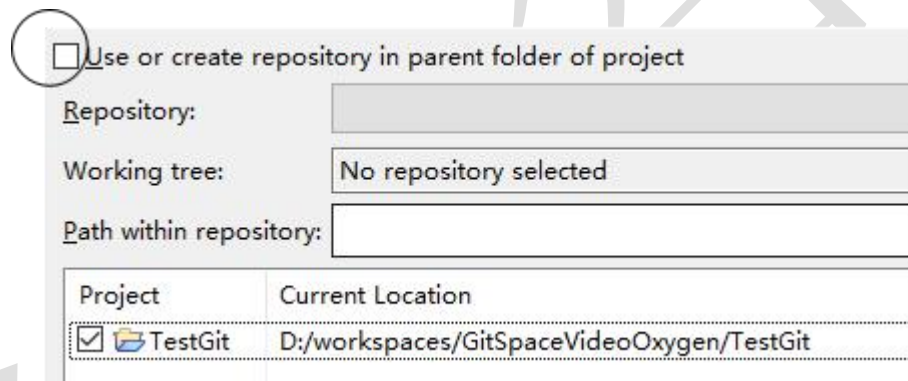
```
$ cat id_rsa.pub
```

- 复制 id_rsa.pub 文件内容, 登录 GitHub, 点击用户头像→Settings→SSH and GPG keys
- New SSH Key
- 输入复制的密钥信息
- 回到 Git bash 创建远程地址别名
`git remote add origin_ssh git@github.com:atguigu2018ybuq/huashan.git`
- 推送文件进行测试

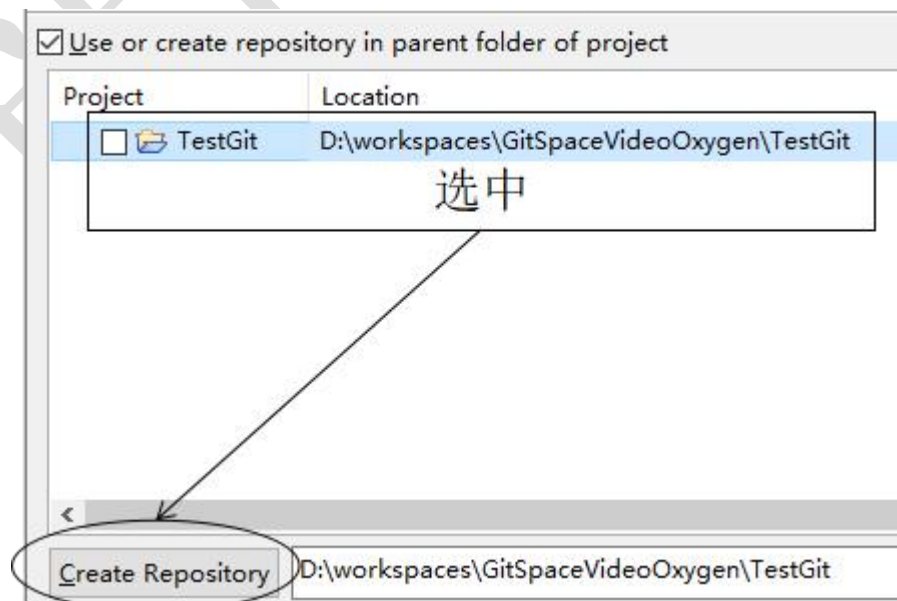
7 Eclipse 操作

7.1 工程初始化为本地库

- 工程→右键→Team→Share Project→Git



- Create Repository



- Finish

7.2 Eclipse 中忽略文件

- 概念：Eclipse 特定文件

这些都是 Eclipse 为了管理我们创建的工程而维护的文件，和开发的代码没有直接关系。最好不要在 Git 中进行追踪，也就是把它们忽略。

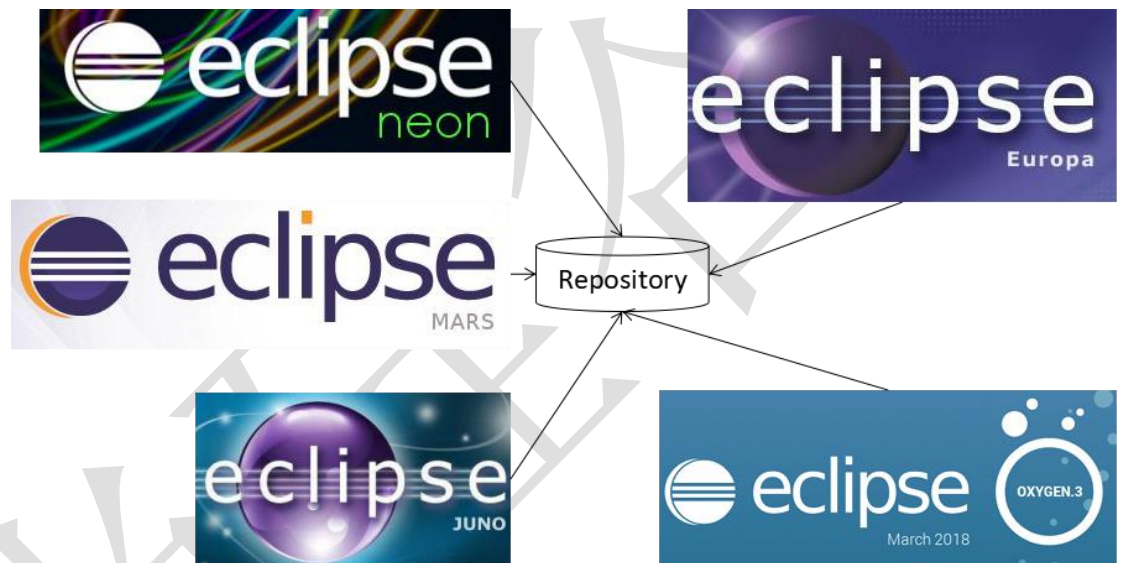
.classpath 文件

.project 文件

.settings 目录下所有文件

- 为什么要忽略 Eclipse 特定文件呢？

同一个团队中很难保证大家使用相同的 IDE 工具，而 IDE 工具不同时，相关工程特定文件就有可能不同。如果这些文件加入版本控制，那么开发时很可能需要为了这些文件解决冲突。



- GitHub 官网样例文件

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

<https://github.com/github/gitignore/blob/master/Java.gitignore>

- 编辑本地忽略配置文件，文件名任意

Java.gitignore
<pre># Compiled class file *.class # Log file *.log # BlueJ files *.ctxt # Mobile Tools for Java (J2ME) .mtj.tmp/</pre>

```
# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*

.classpath
.project
.settings
target
```

- 在~/.gitconfig 文件中引入上述文件
[core]
excludesfile = C:/Users/Lenovo/Java.gitignore

[注意：这里路径中一定要使用“/”，不能使用“\”]

7.3 推送到远程库



Location

URI:

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

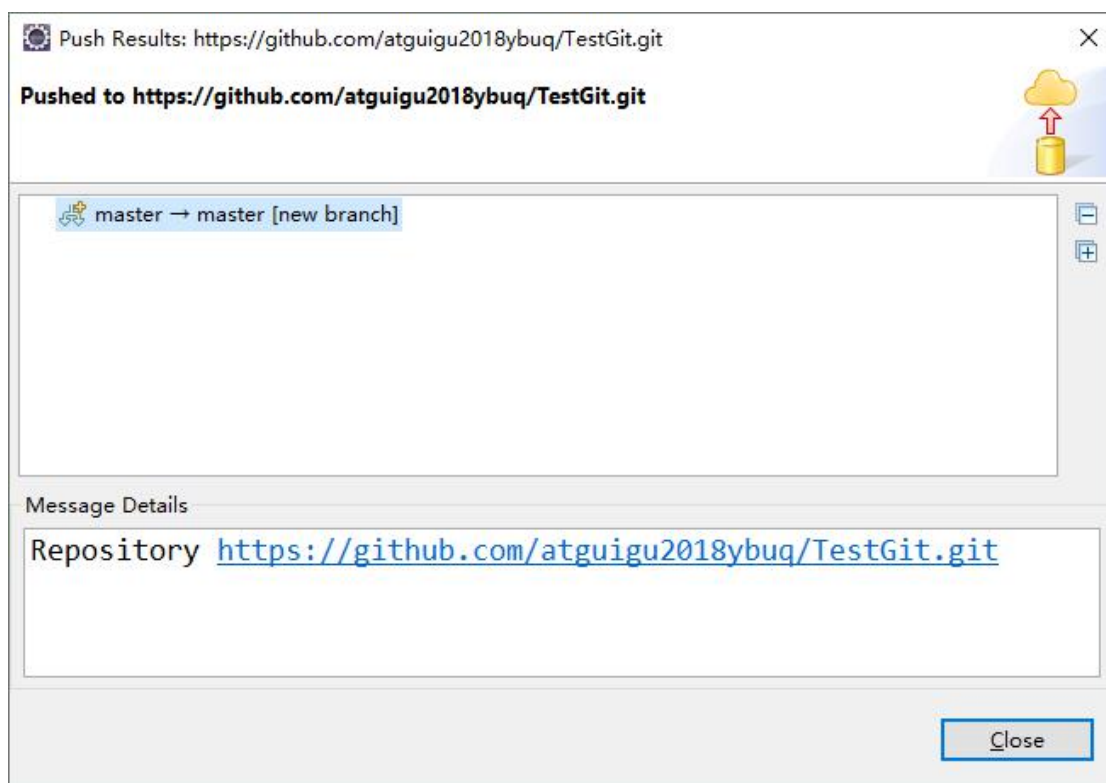
☒ Store in Secure Store

Add predefined specification

Specifications for push

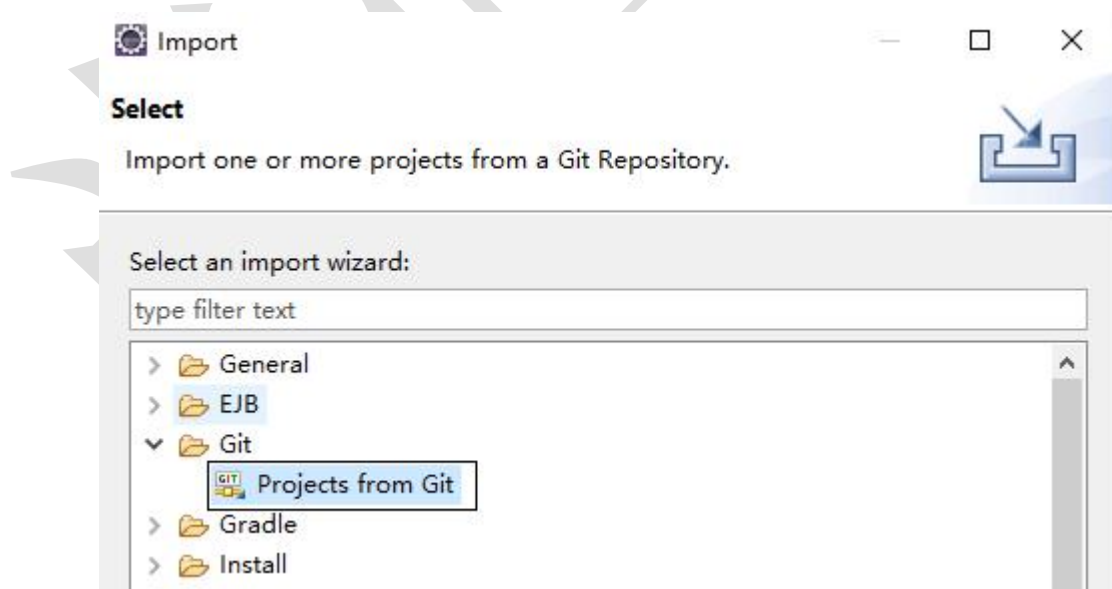
Mode	Source Ref	Destination Ref	Force Update	Remove
<input type="button" value="+"/>	Update	refs/heads/*	<input type="checkbox"/>	<input type="button" value="Remove"/>

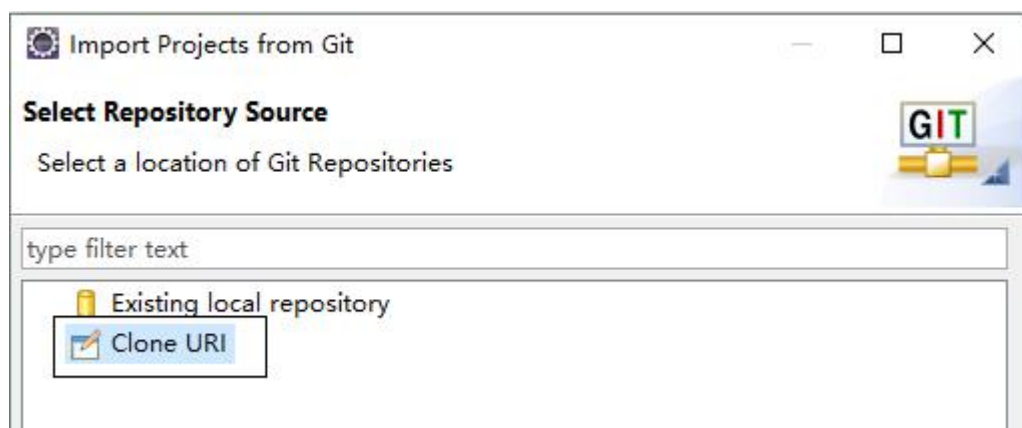




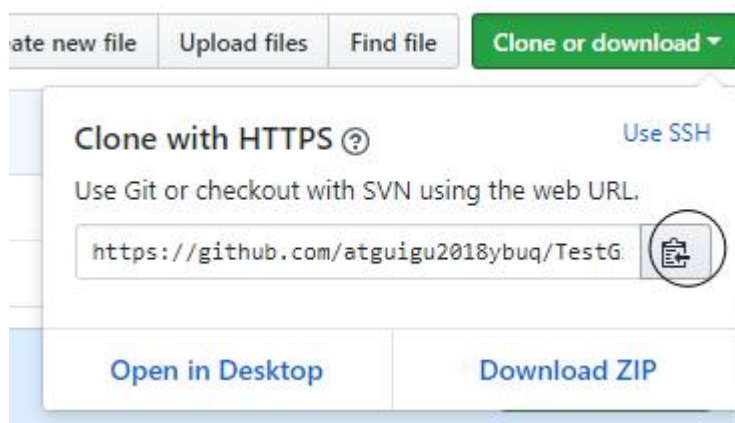
7.4 Oxygen Eclipse 克隆工程操作

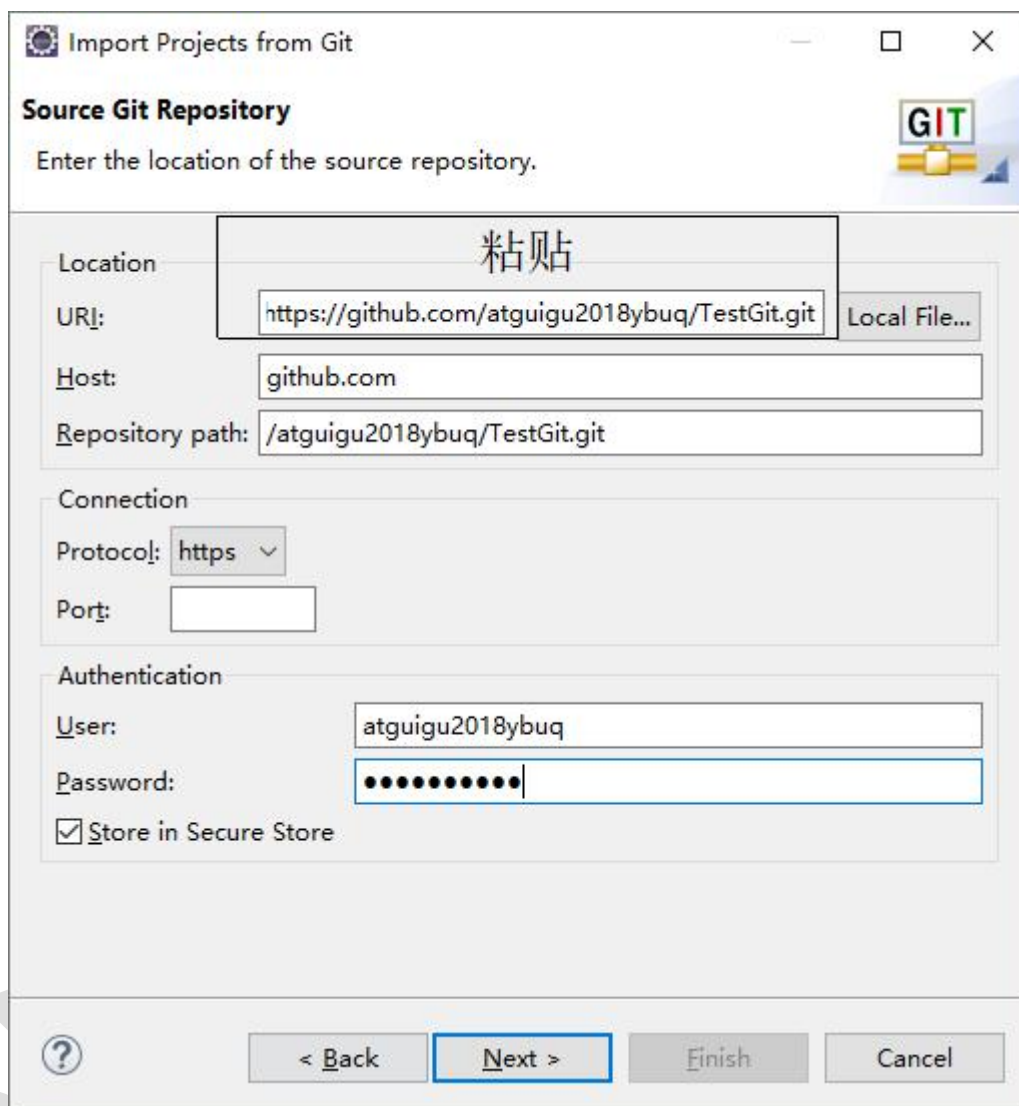
- Import...导入工程





- 到远程库复制工程地址





Import Projects from Git

Source Git Repository
Enter the location of the source repository.

Location

URI:

Host:

Repository path:

Connection

Protocol:

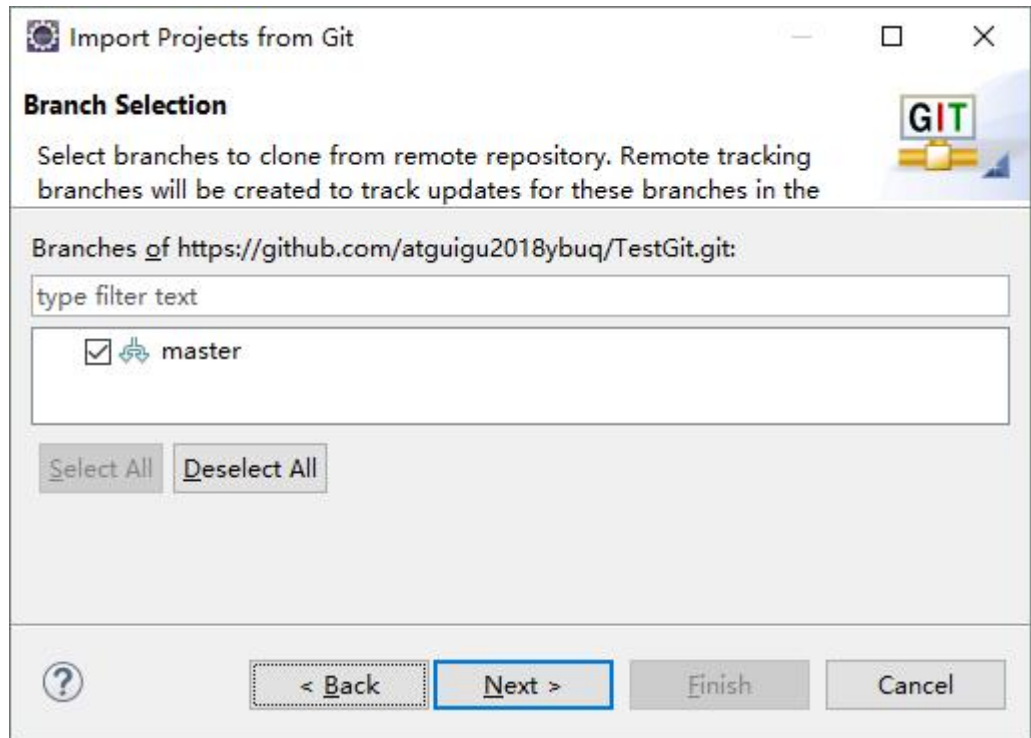
Port:

Authentication

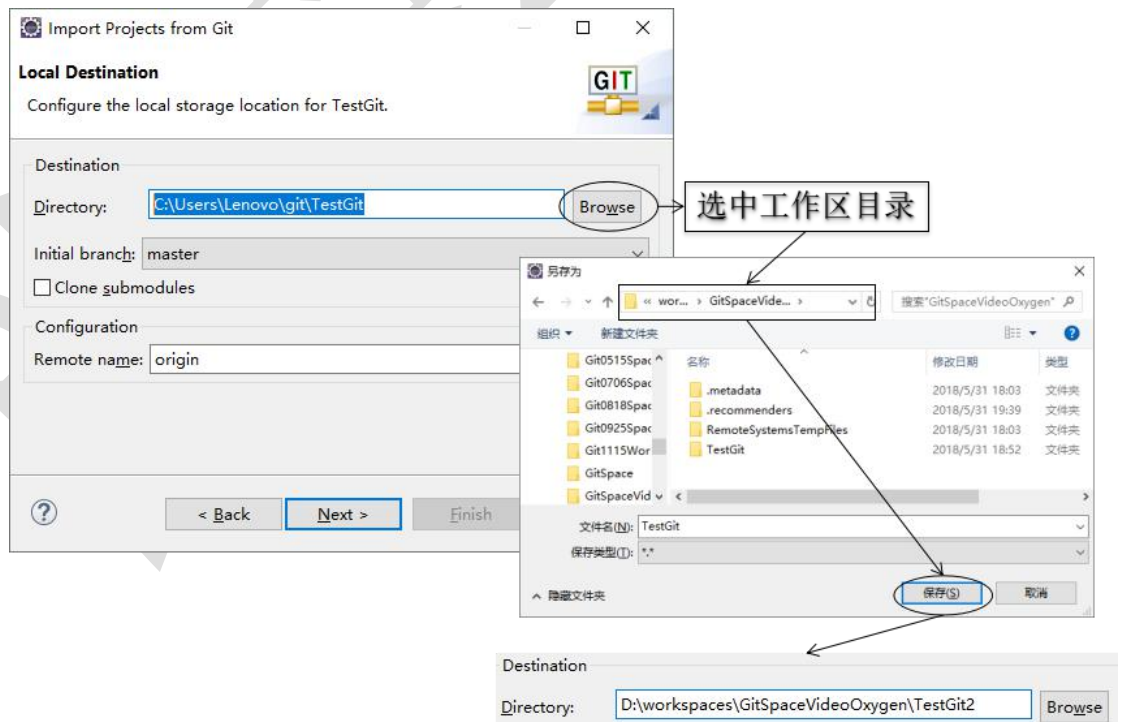
User:

Password:

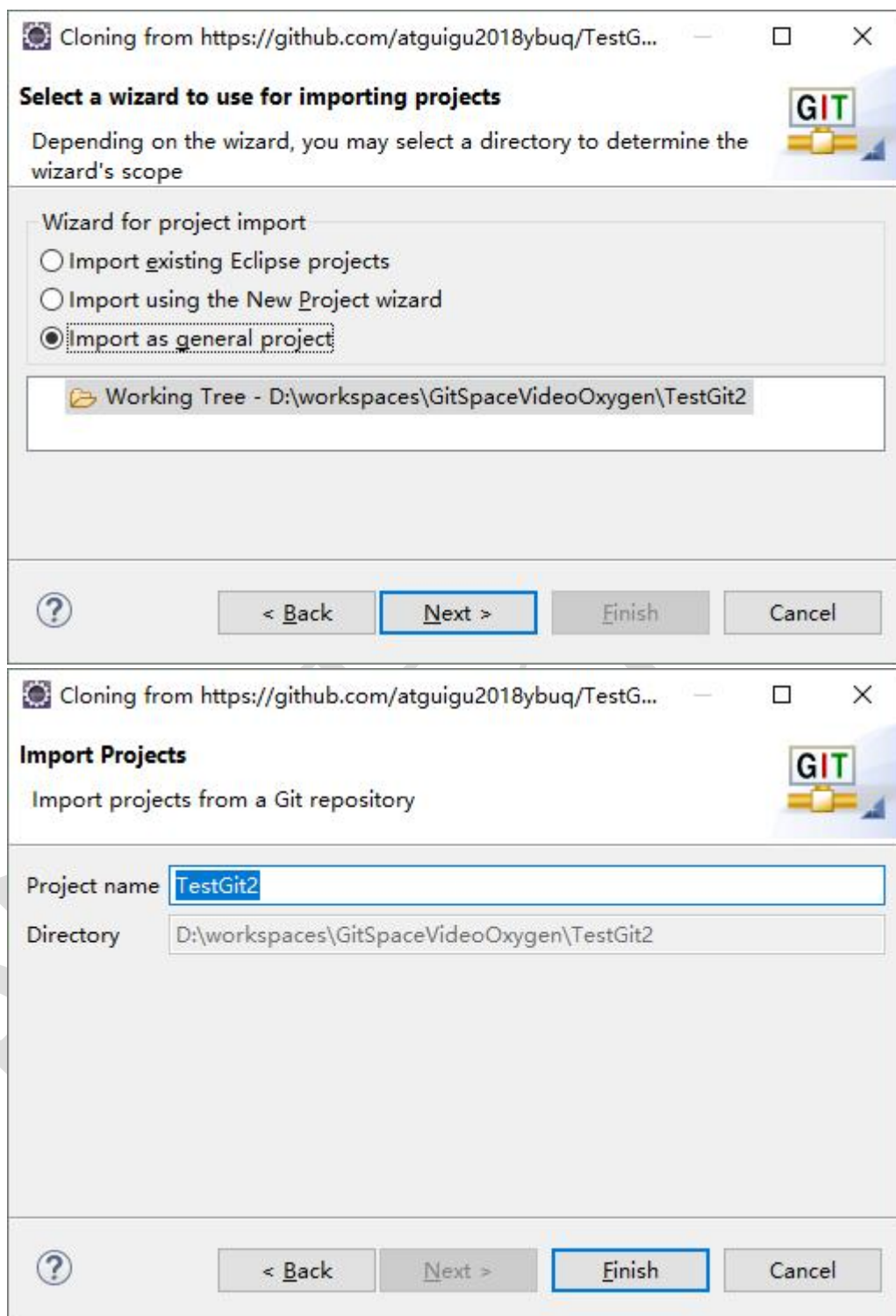
☒ Store in Secure Store



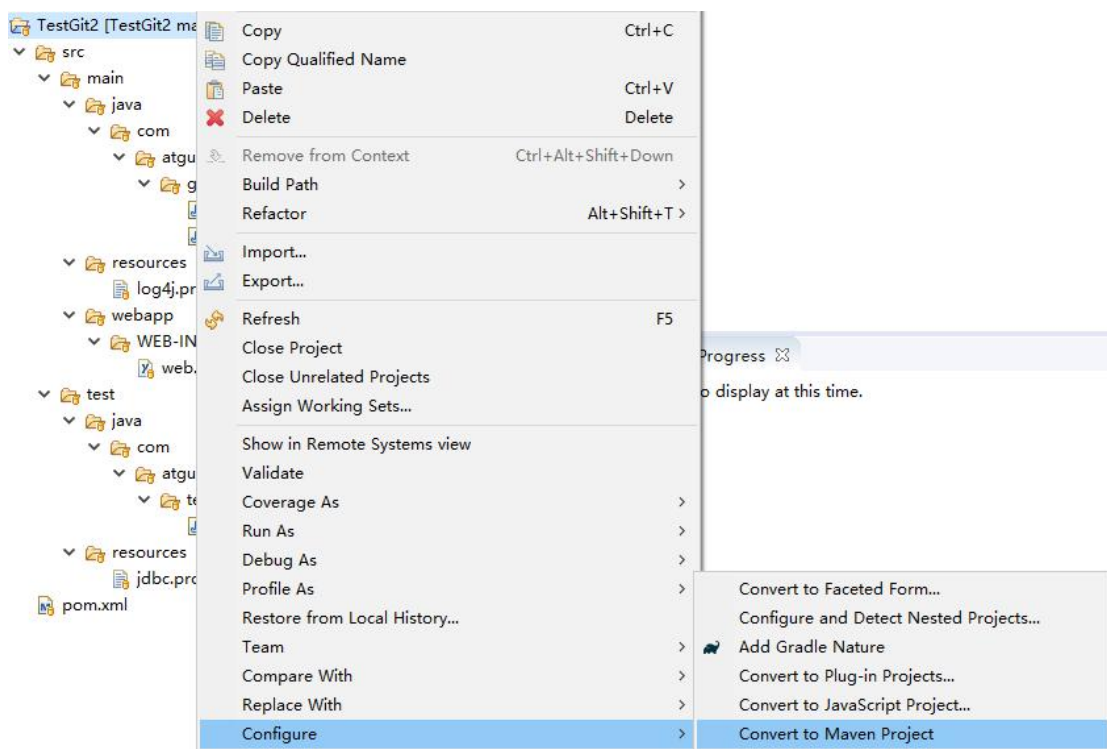
- 指定工程的保存位置



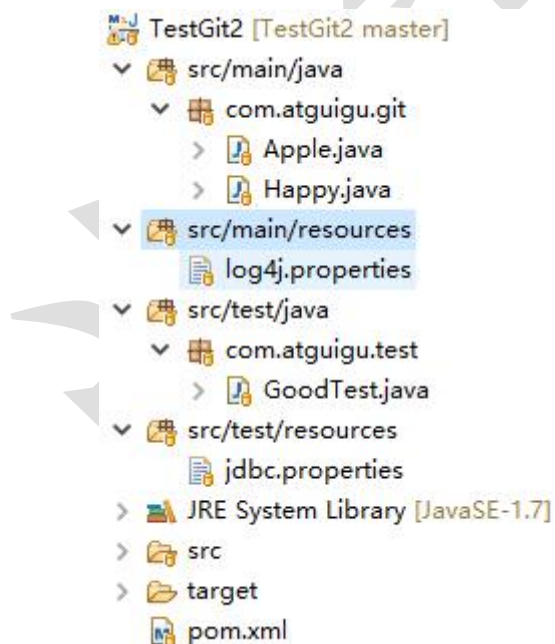
- 指定工程导入方式，这里只能用：Import as general project



➤ 转换工程类型

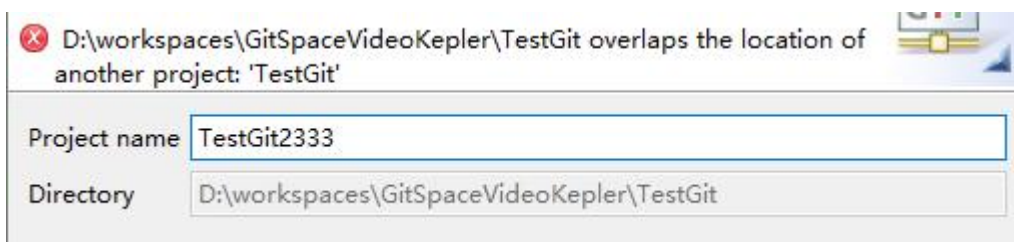


➤ 最终效果

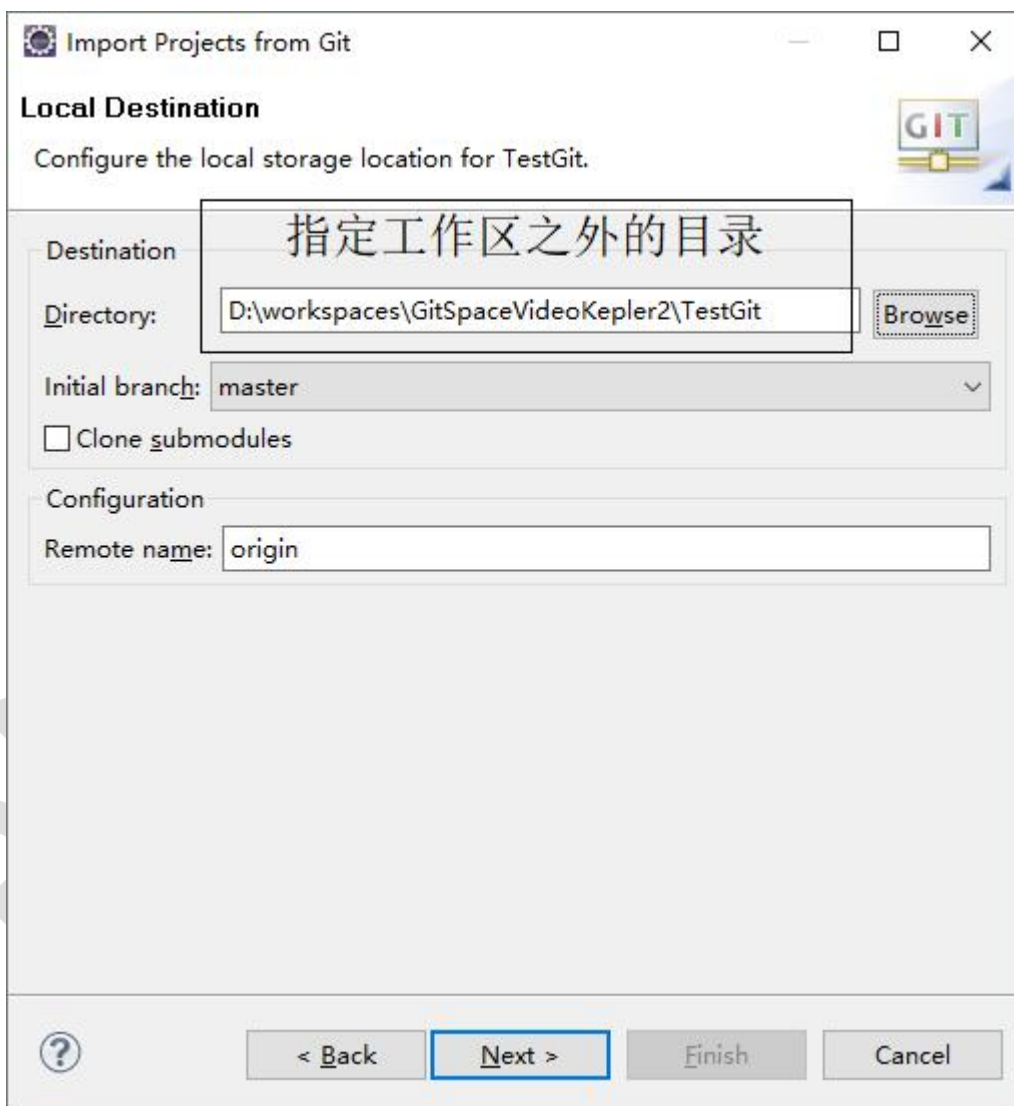


7.5 Kepler Eclipse 克隆工程操作

➤ 问题：不能保存到当前 Eclipse 工作区目录



- 正确做法：保存到工作区以外的目录中



7.6 解决冲突

冲突文件 → 右键 → Team → Merge Tool

修改完成后正常执行 add/commit 操作即可

8 Git 工作流

8.1 概念

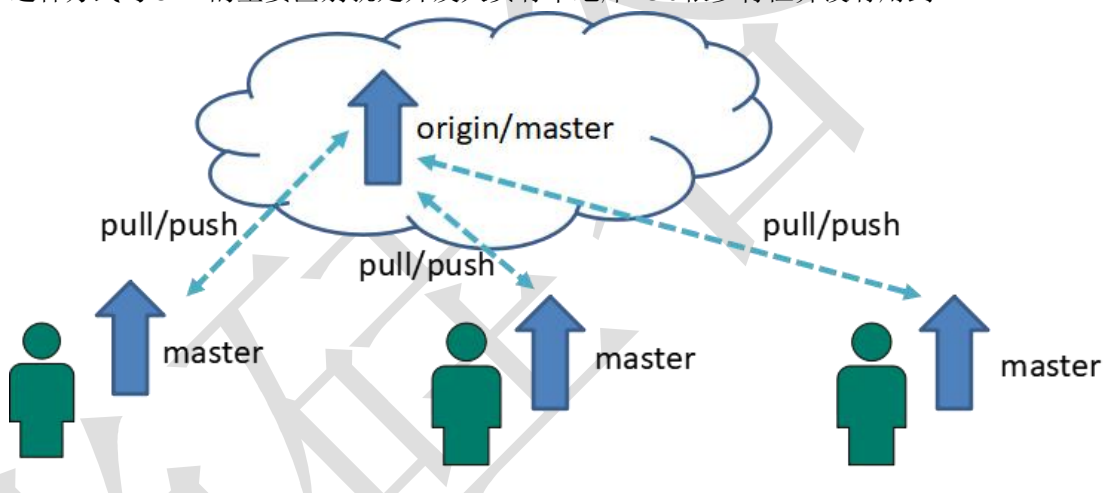
在项目开发过程中使用 Git 的方式

8.2 分类

8.2.1 集中式工作流

像 SVN 一样，集中式工作流以中央仓库作为项目所有修改的单点实体。所有修改都提交到 Master 这个分支上。

这种方式与 SVN 的主要区别就是开发人员有本地库。Git 很多特性并没有用到。



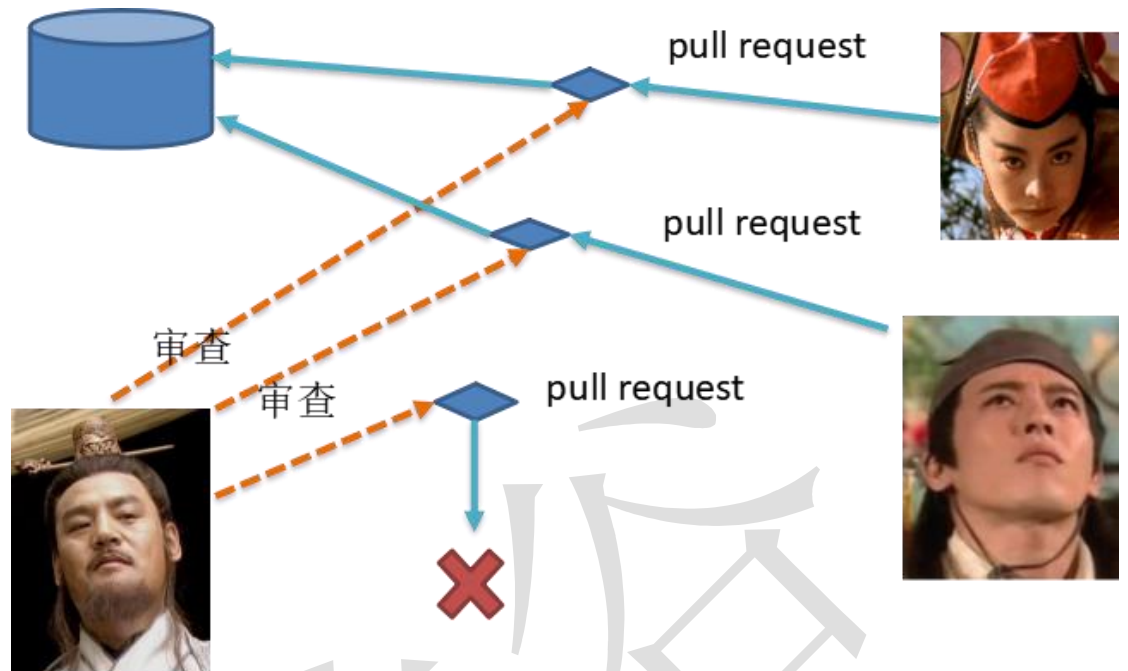
8.2.2 GitFlow 工作流

Gitflow 工作流通过为功能开发、发布准备和维护设立了独立的分支，让发布迭代过程更流畅。严格的分支模型也为大型项目提供了一些非常必要的结构。



8.2.3 Forking 工作流

Forking 工作流是在 GitFlow 基础上，充分利用了 Git 的 Fork 和 pull request 的功能以达到代码审核的目的。更适合安全可靠地管理大团队的开发者，而且能接受不信任贡献者的提交。

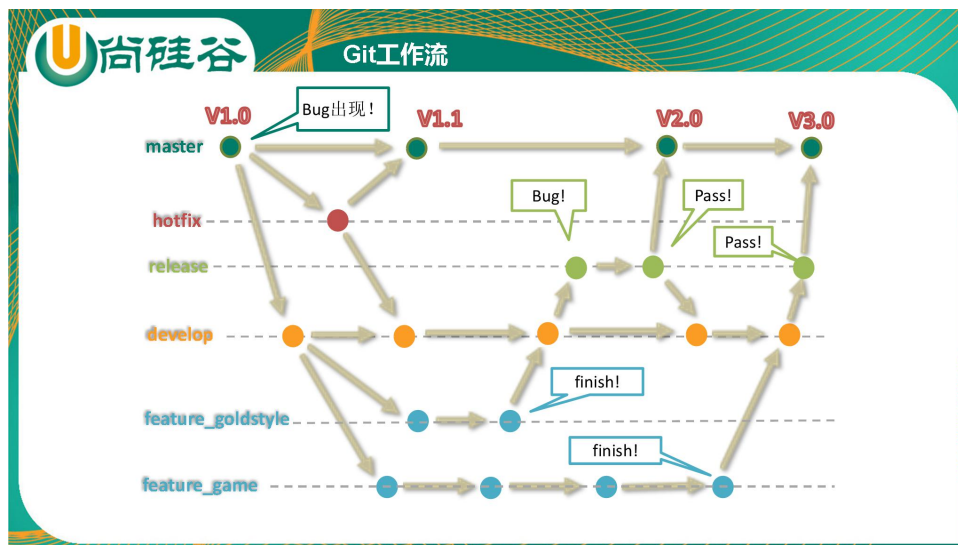


8.3 GitFlow workflow 详解

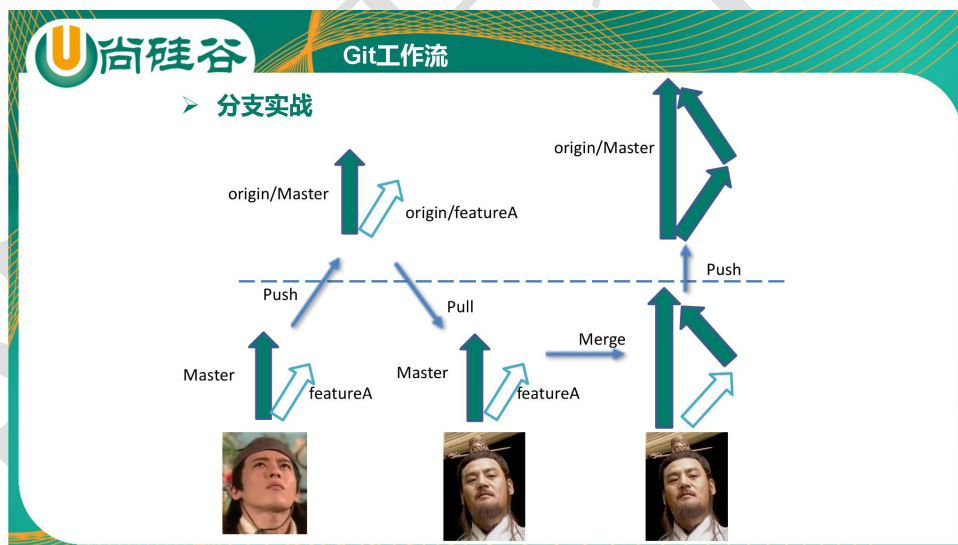
8.3.1 分支种类

- 主干分支 `master`
主要负责管理正在运行的生产环境代码。永远保持与正在运行的生产环境完全一致。
- 开发分支 `develop`
主要负责管理正在开发过程中的代码。一般情况下应该是最新的代码。
- bug 修复分支 `hotfix`
主要负责管理生产环境下出现的紧急修复的代码。从主干分支分出，修理完毕并测试上线后，并回主干分支。并回后，视情况可以删除该分支。
- 准生产分支（预发布分支） `release`
较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶段较稳定后可以视情况删除。
- 功能分支 `feature`
为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合并到开发分支。

8.3.2 GitFlow 工作流举例

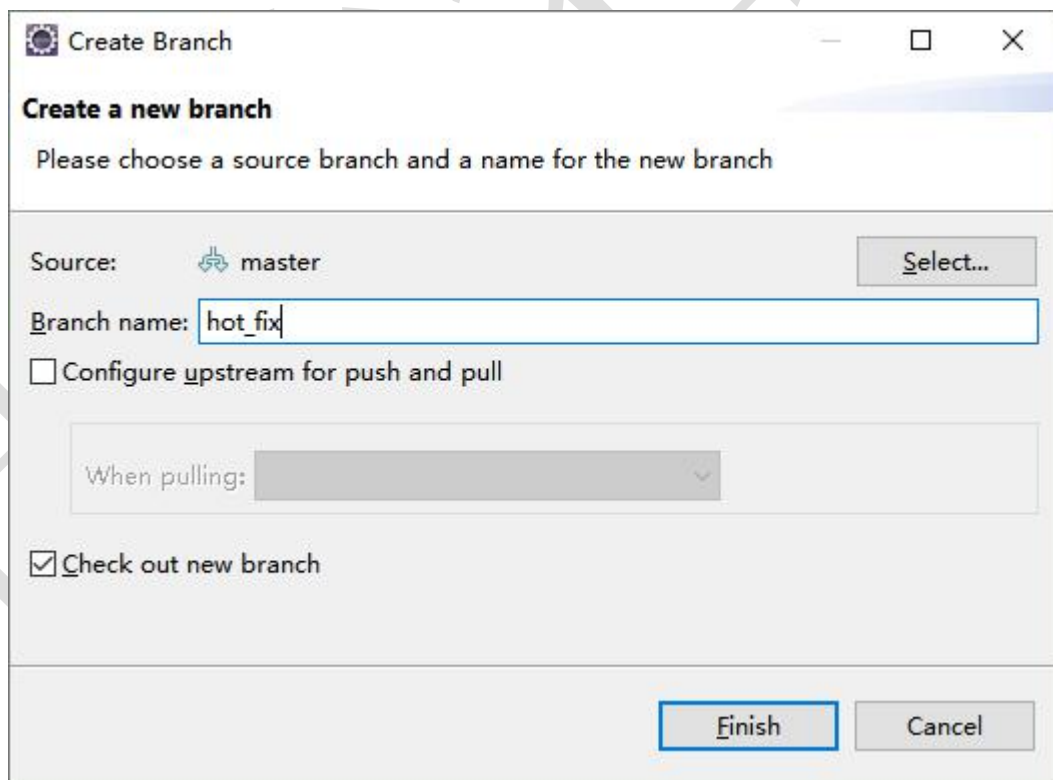
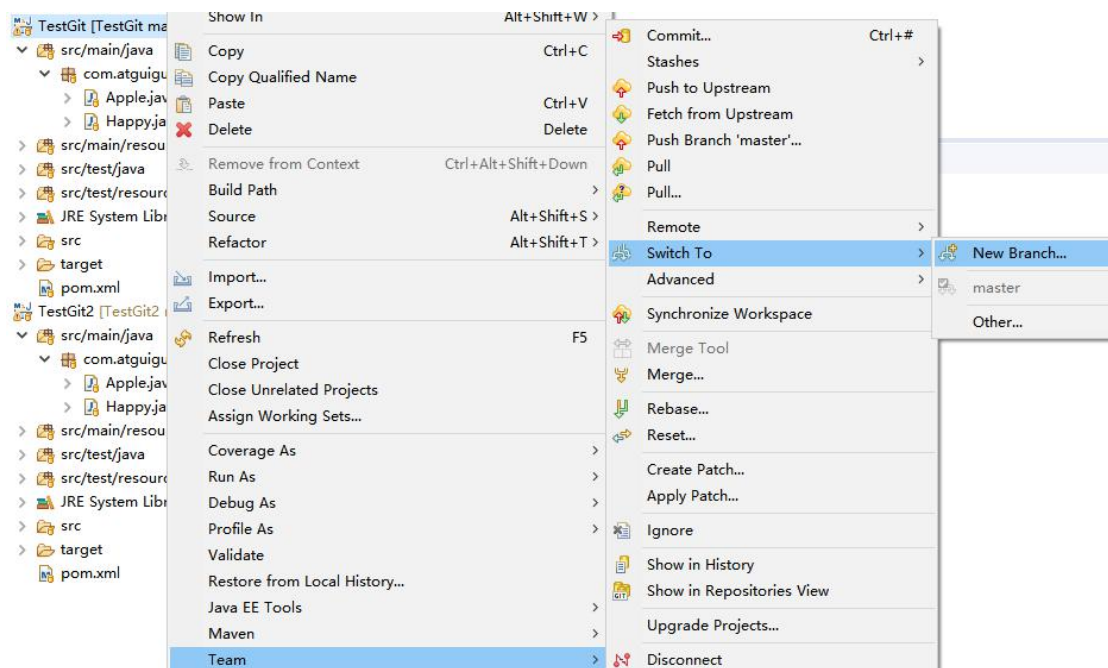


8.3.3 分支实战

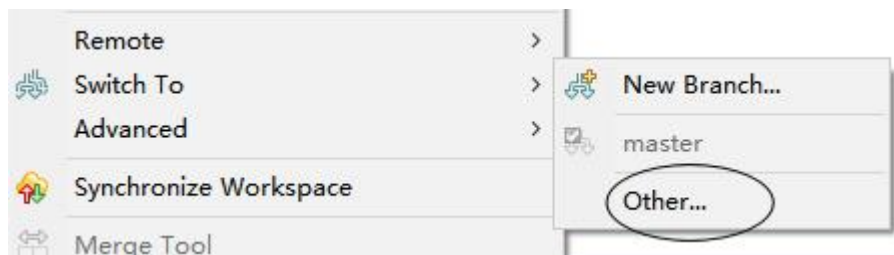


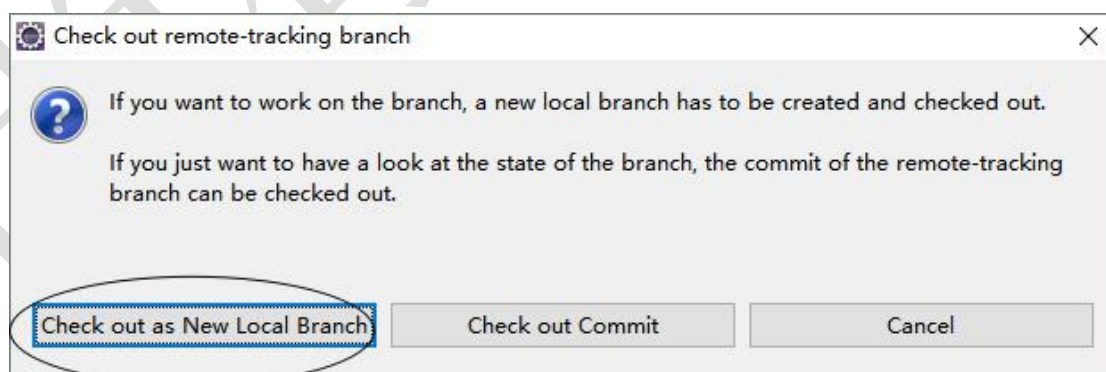
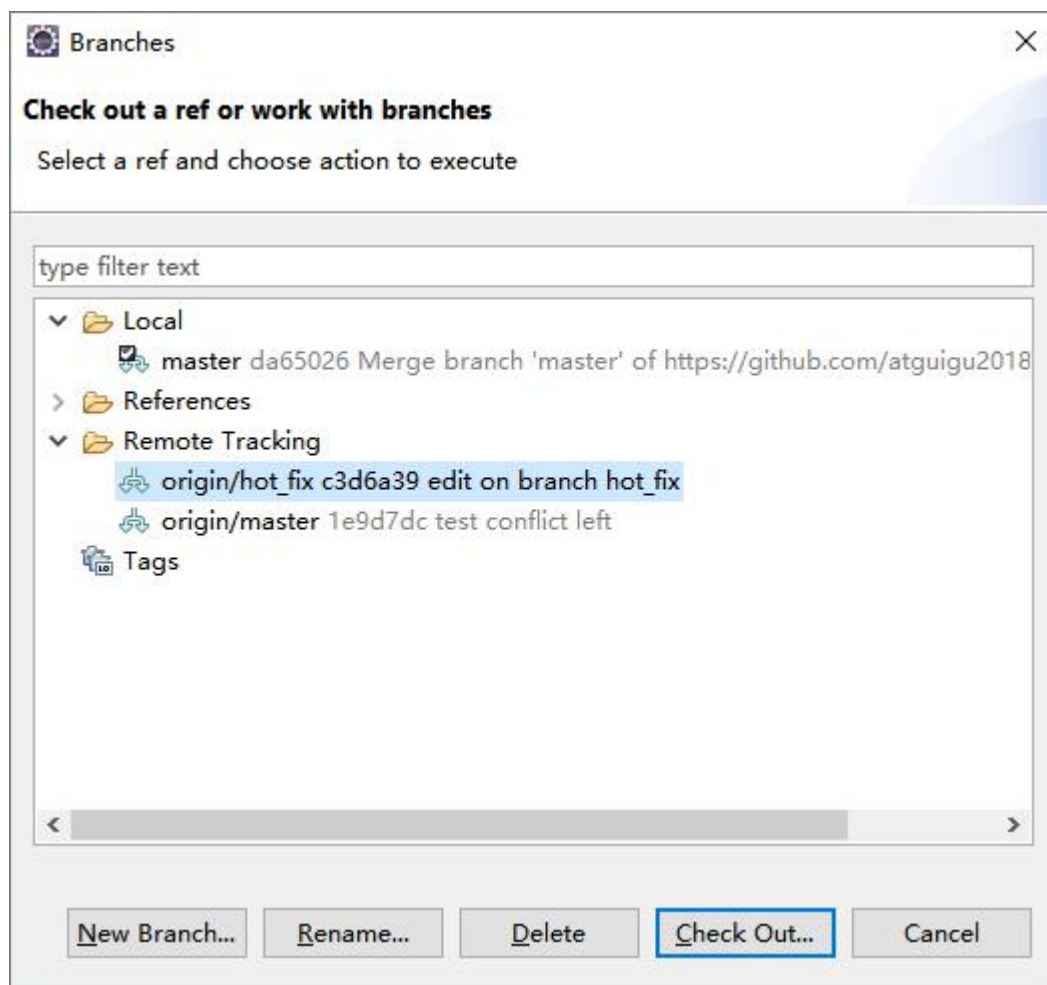
8.3.4 具体操作

- 创建分支

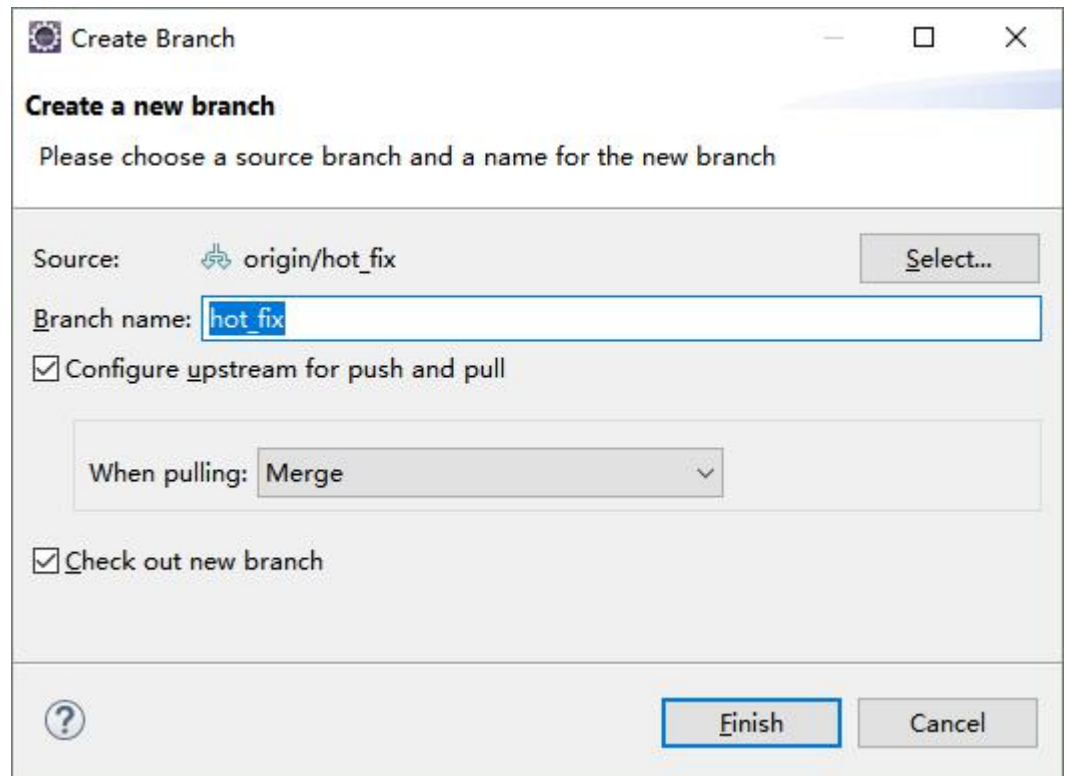


➤ 切换分支审查代码

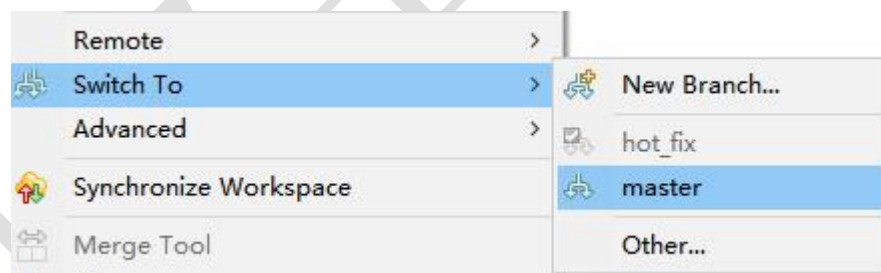




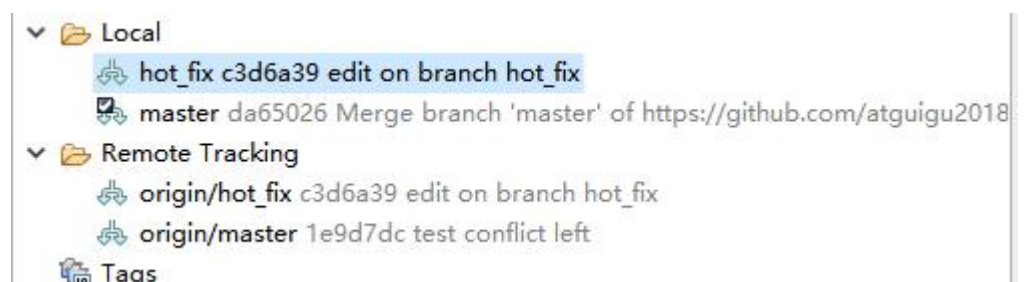
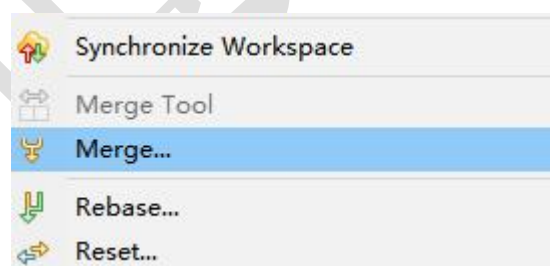
- 检出远程新分支



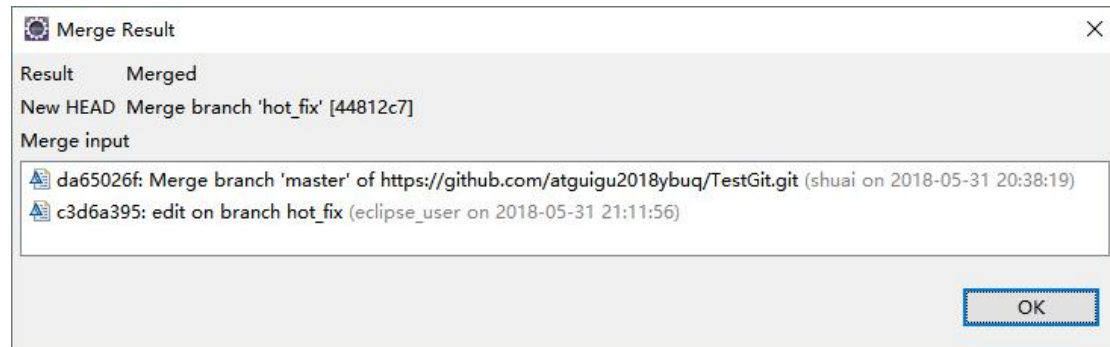
- 切换回 master



- 合并分支



➤ 合并结果



合并成功后，把 master 推送到远程。

9 Gitlab 服务器搭建过程

9.1 官网地址

首页: <https://about.gitlab.com/>

安装说明: <https://about.gitlab.com/installation/>

9.2 安装命令摘录

```
sudo yum install -y curl policycoreutils-python openssh-server crontab
sudo lokkit -s http -s ssh
sudo yum install postfix
sudo service postfix start
sudo chkconfig postfix on
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh | sudo bash
sudo EXTERNAL_URL="http://gitlab.example.com" yum -y install gitlab-ee
```

实际问题: yum 安装 gitlab-ee(或 ce)时, 需要联网下载几百 M 的安装文件, 非常耗时, 所以应提前把所需 RPM 包下载并安装好。

下载地址为:

```
https://packages.gitlab.com/gitlab/gitlab-ce/packages/el/7/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm
```

9.3 调整后的安装过程

```
sudo rpm -ivh /opt/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm
sudo yum install -y curl policycoreutils-python openssh-server crontab
sudo lokkit -s http -s ssh
sudo yum install postfix
sudo service postfix start
sudo chkconfig postfix on
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh | sudo bash
sudo EXTERNAL_URL="http://gitlab.example.com" yum -y install gitlab-ce
```

当前步骤完成后重启。

9.4 gitlab 服务操作

- 初始化配置 gitlab
gitlab-ctl reconfigure
- 启动 gitlab 服务
gitlab-ctl start
- 停止 gitlab 服务
gitlab-ctl stop

9.5 浏览器访问

访问 Linux 服务器 IP 地址即可，如果想访问 EXTERNAL_URL 指定的域名还需要配置域名服务器或本地 hosts 文件。

初次登录时需要为 gitlab 的 root 用户设置密码。

Please create a password for your new account.

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Change your password

New password

Confirm new password

Change your password

root/atguigu2018good

※应该会需要停止防火墙服务：

service firewalld stop

微信号：creathinFeng