

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

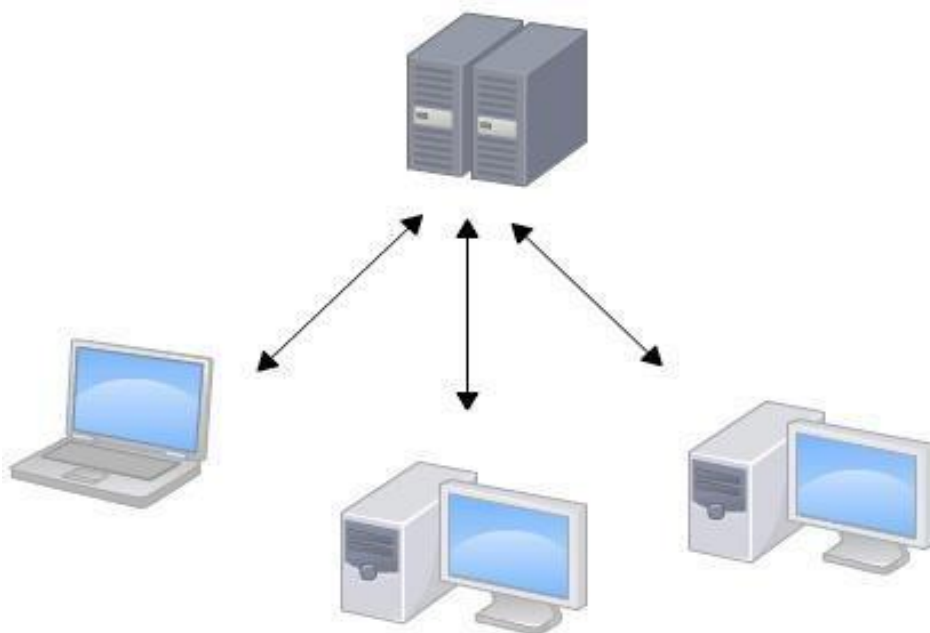
思想：版本控制

实现：版本控制工具

集中式版本控制工具：

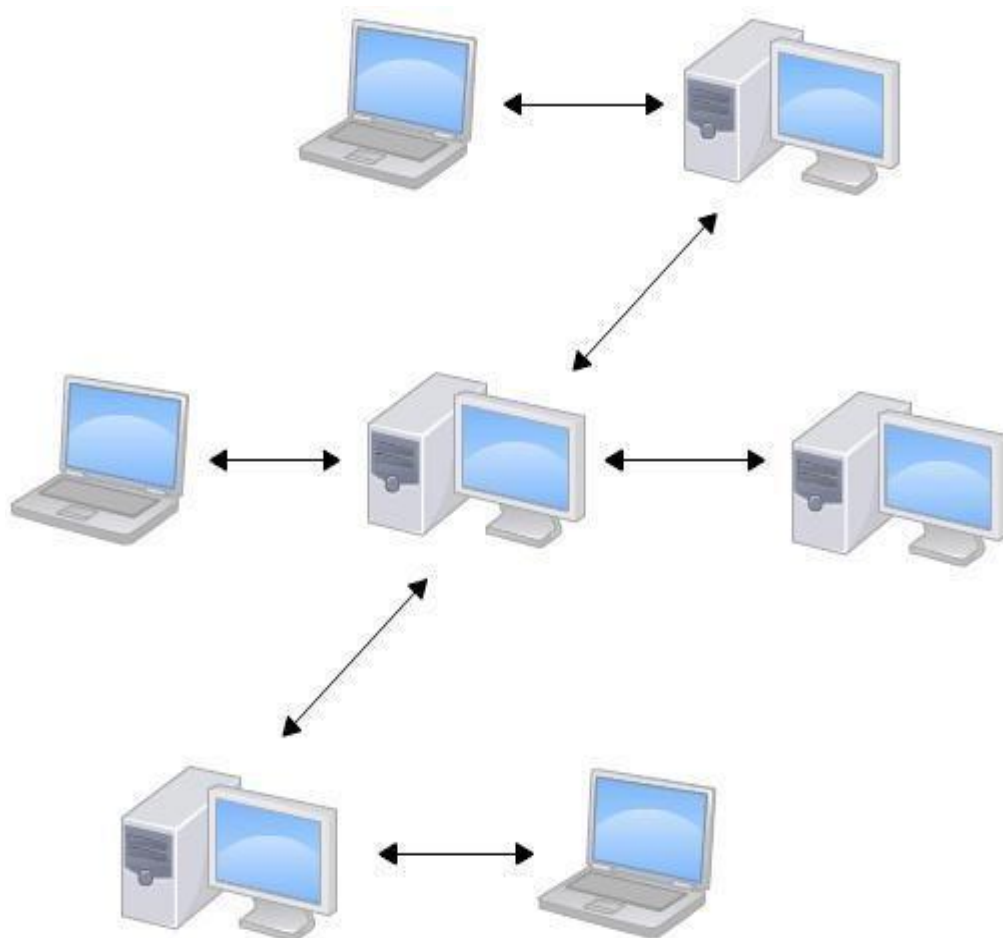
CVS、SVN、VSS……

单点故障



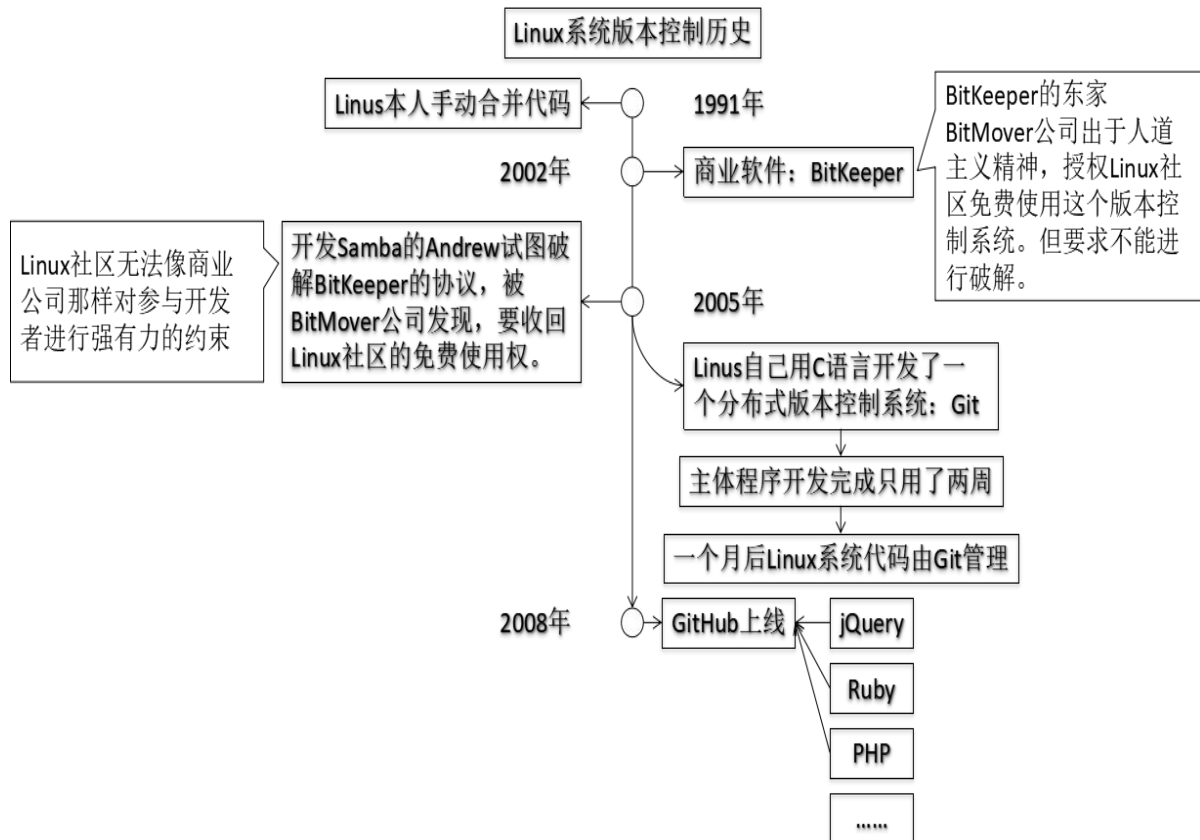
分布式版本控制工具：（每个用户机, 也都有版本历史）意味可以本地库传本地不建议这么做

Git、Mercurial、Bazaar、Darcs……



3 Git 简介 p4-p9

3.1Git 简史 p4



3.2Git 官网和 Logo

官网地址: <https://git-scm.com/>

Logo:



3.3Git 的优势 p5

大部分操作在本地完成, 不需要联网

完整性保证 (提交每条数据进行 hash 运算)

尽可能添加数据而不是删除或修改数据

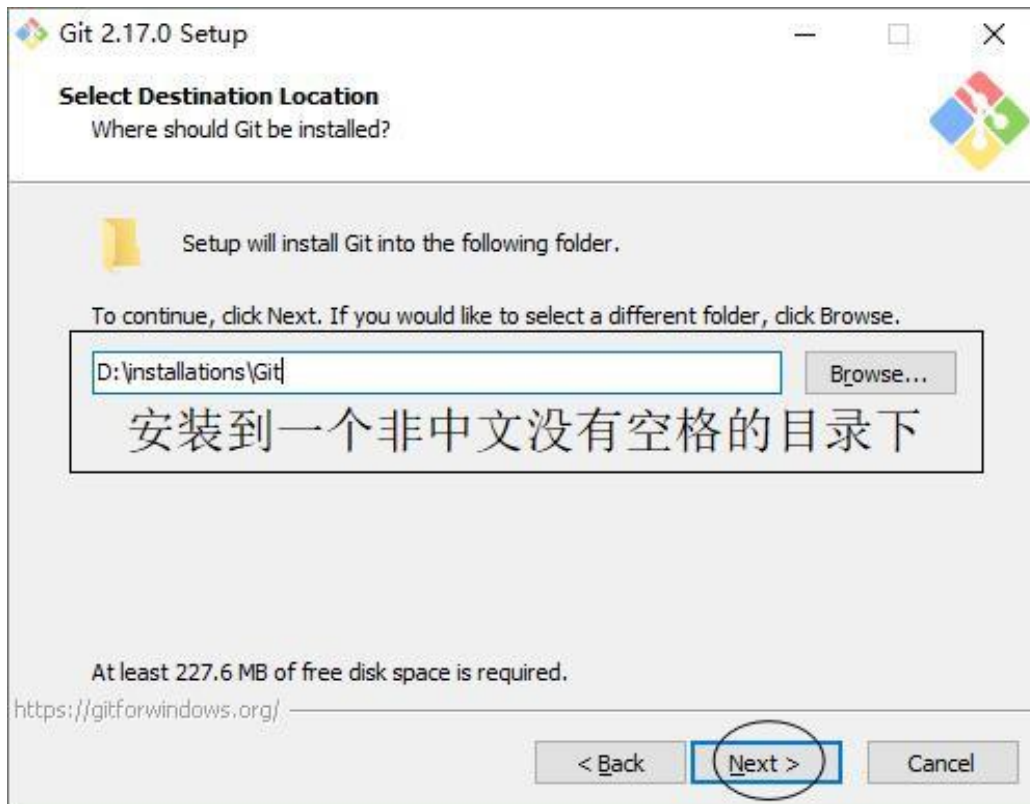
分支操作非常快捷流畅 (1 因为用快照, 2 每个分支只是创建一个指针)

与 Linux 命令全面兼容 ()

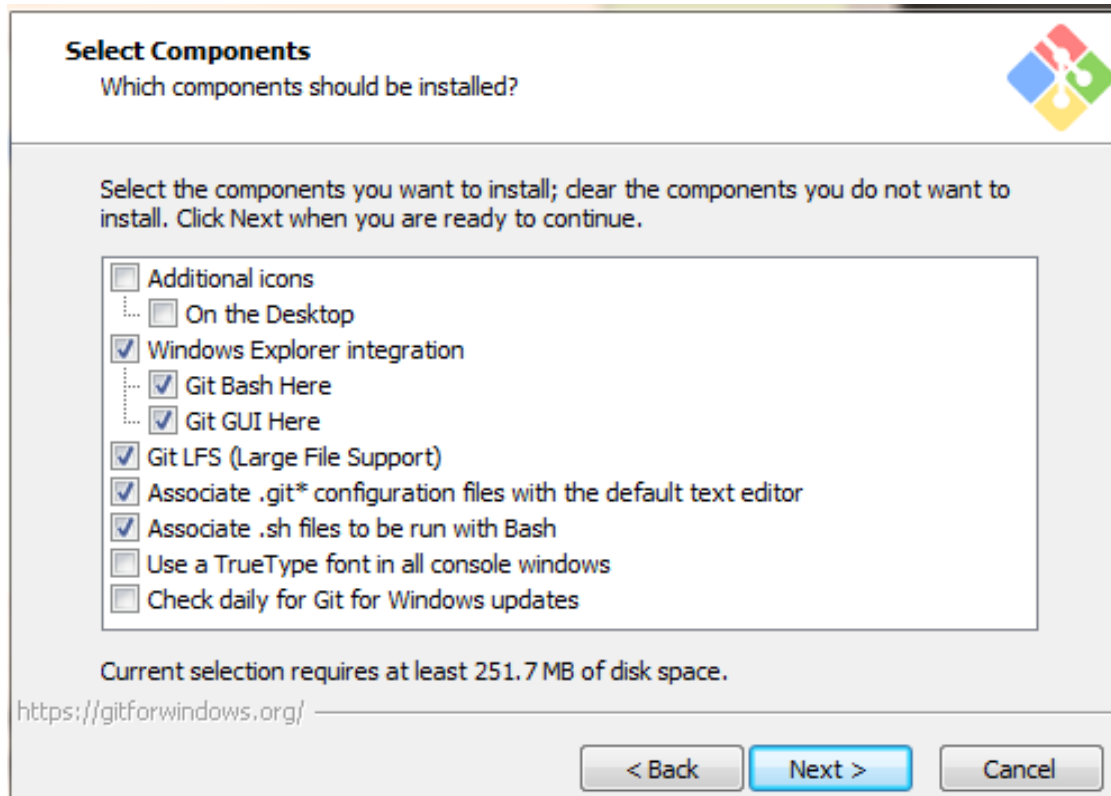
3.4Git 安装 windows 安装 p6

<https://www.cnblogs.com/wlming/p/12213876.html> Windows 安装

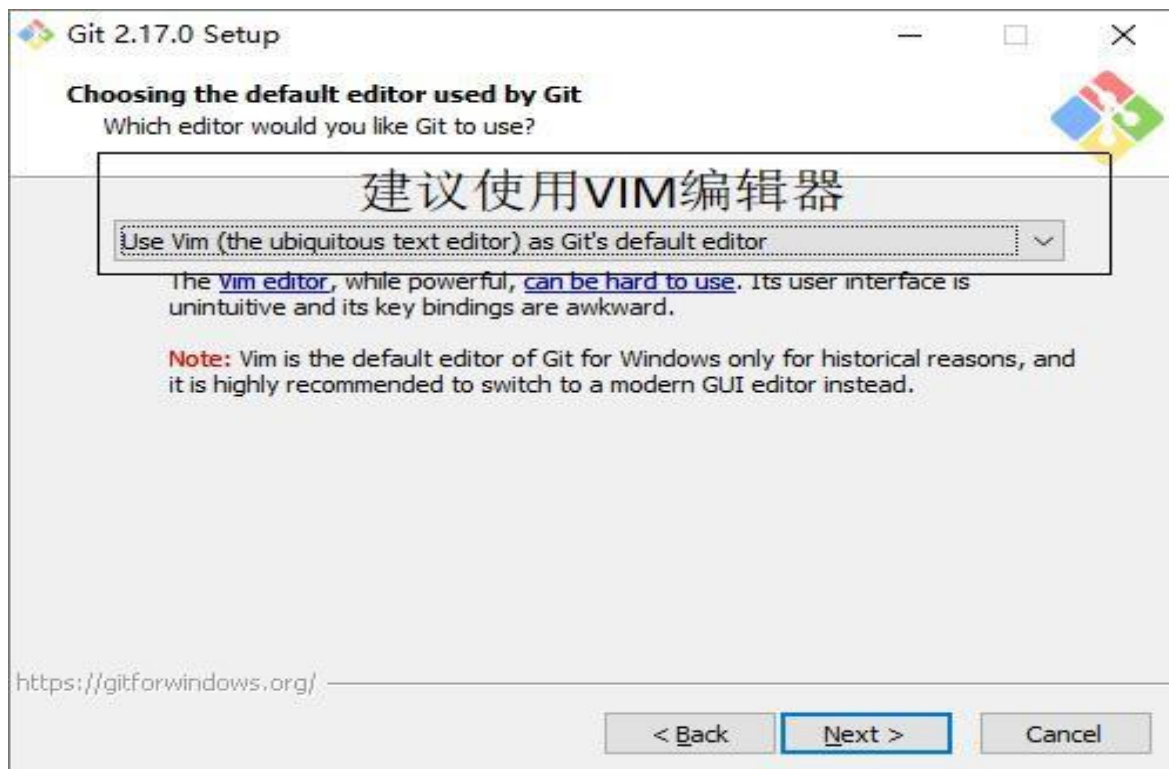
(linux 直接 `yum install -y git`),加-y 自动选择 y, 全自动



2 下面默认设置就行: 下图(下一步) 3 这个的下一步也使用默认 直接下一步



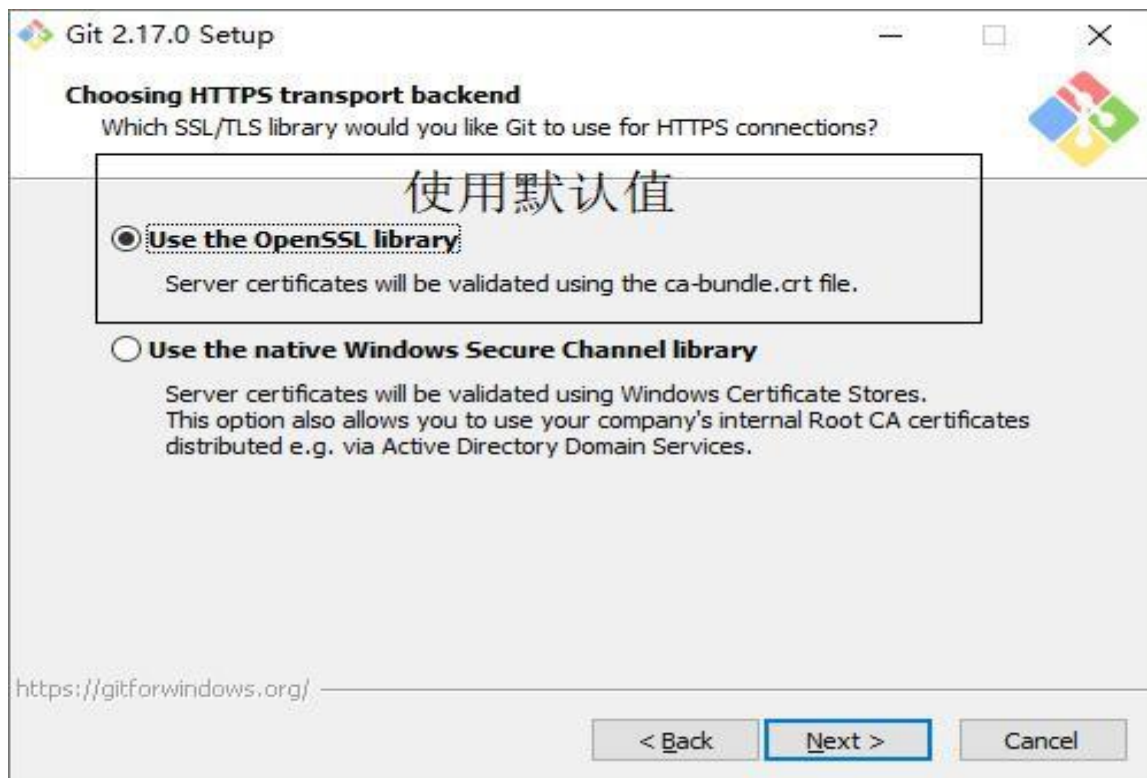
4 选择默认文本编辑器



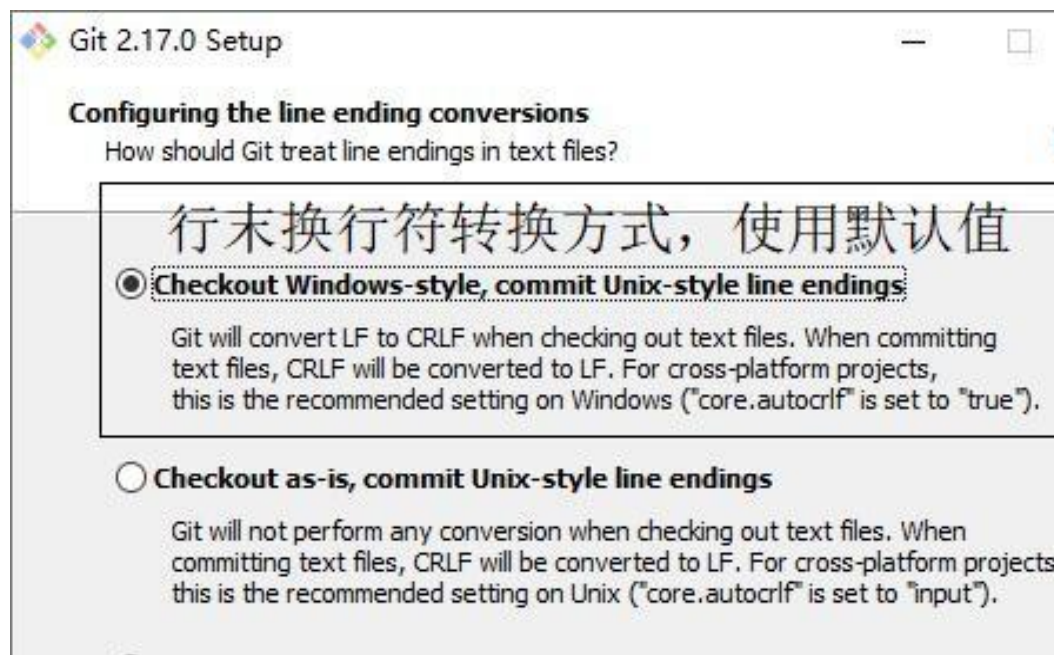
5 然后修改环境变量(选第一完全不修改),下面选项第二个(是被认为安全的)



6 选择服务端本地库和远程库连接方式(1 通用连接 2 使用 Windows 连接方式)



7 选择换行符的方式(1 检查文件时 LF 转为 CRLF 提交相反)



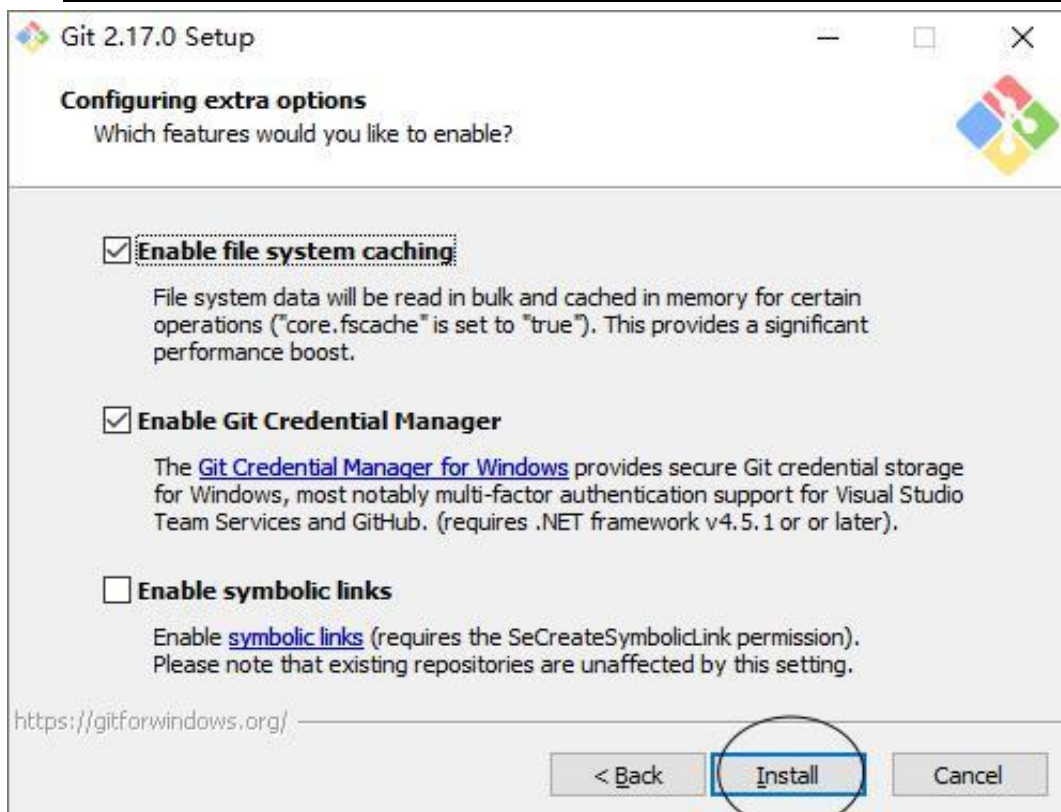
8 选择终端(1Git 默认终端(是 liunx 命令)2 选择 Windows 终端(wind 命令))



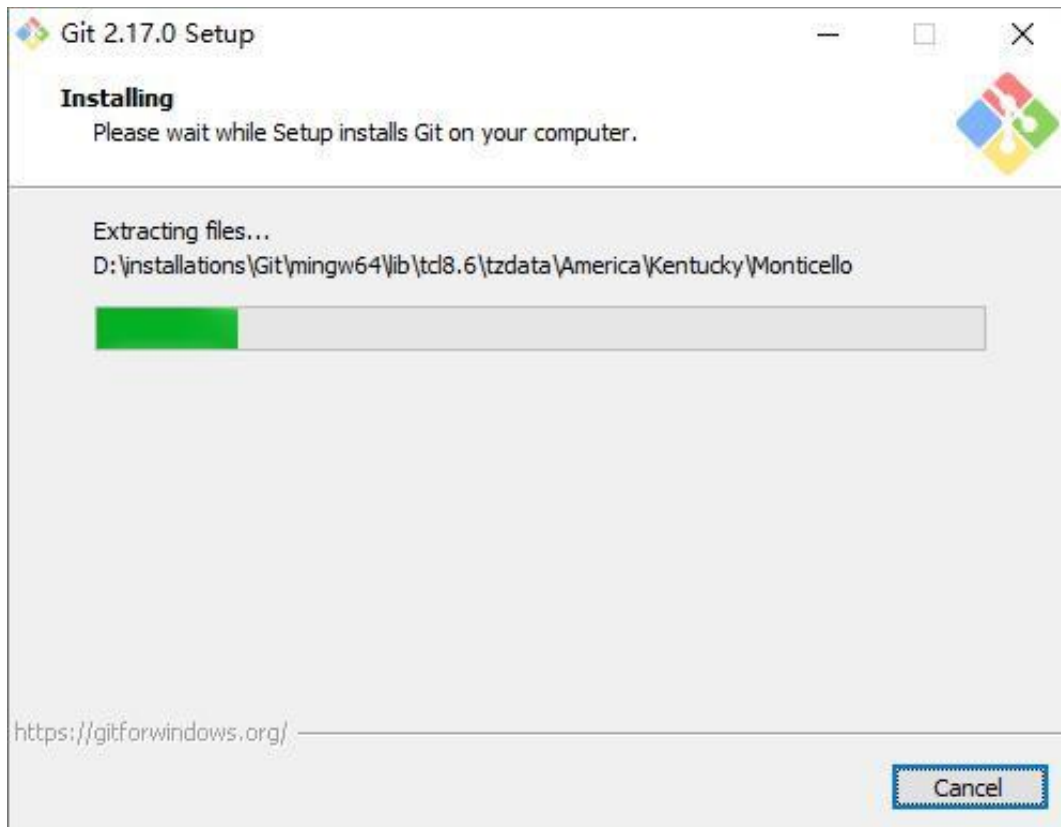
9 使用默认(选择第二个需要安装.NET framework c4.5.1 以上版本)

NET framework 安装失败解决方案:

<https://jingyan.baidu.com/article/fb48e8bee50ebf6e632e1464.html>



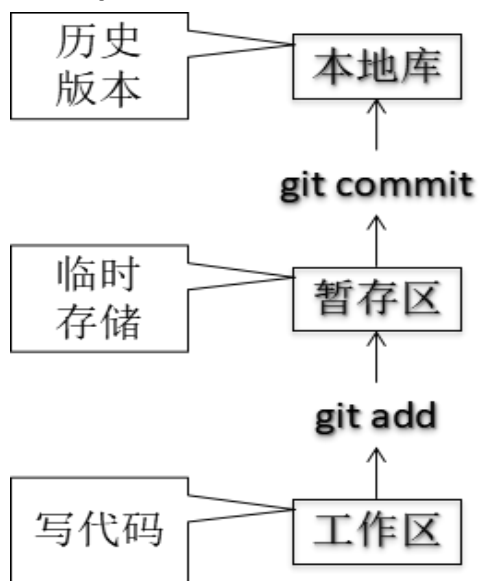
10



11 (1 加载他的 git Bash 终端 2 查看更新的文档)



3.5Git 结构 p7



3.6Git 和代码托管中心 p8

代码托管中心的任务: [维护远程库](#)

局域网环境下

[GitLab 服务器](#)

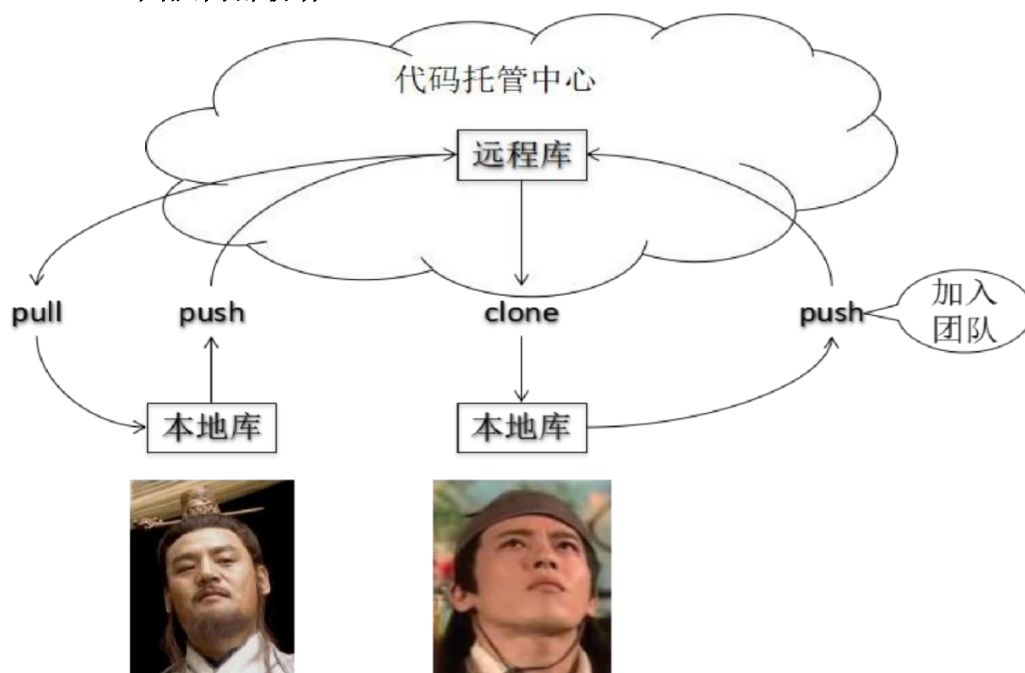
外网环境下

[GitHub](#)

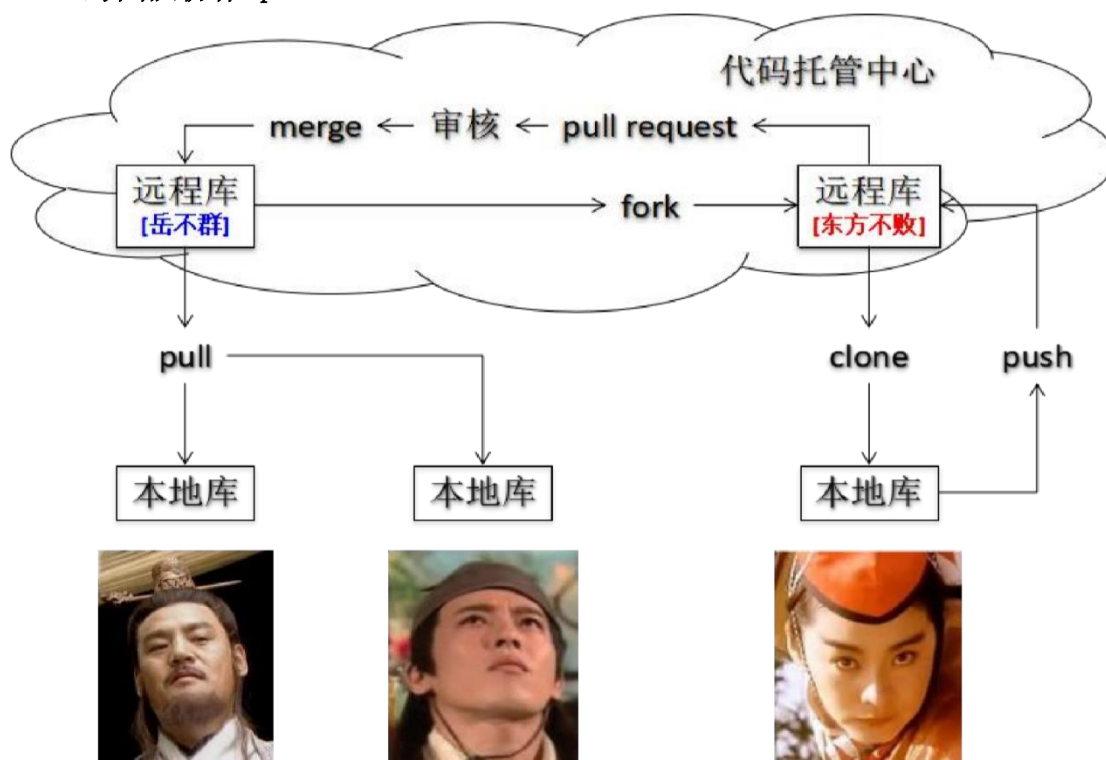
[码云](#)

3.7 本地库和远程库(上面 3.5 是 git 结构)p9

3.7.1 团队内部协作



3.7.2 跨团队协作 p9



4 Git 命令行操作 p10-p2

4.1 本地库初始化 p10 (我的:e 盘下/may/GitSpaceVideo/WeChat)

命令: `git init` (切换到目录>右键 Git Bash Here>用 liunx 命令到对应目录下>初始化)

效果: 会在当前目录生成.git 目录(隐藏的)

```
$ 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 (本地库初始化后, 要执行的)p11

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: (不带参数-) 仅在当前本地库范围内有效

`git config user.name tom_pro`

`git config user.email goodMorning_pro@taku.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 user.email goodMorning_glb@taku.com`

信息保存位置: `~/.gitconfig` 文件 (家目录下 `c/user`)

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

级别优先级

- 1 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
- 2 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
- 3 二者都没有不允许

4.3 基本操作(查看状态.提交暂存区和本地库.历史查看.) p12-p22

4.3.1 状态查看

`git status` /*第一代表是那个分区的,第2 是否提交 3 有没有可提交的文件和提示
查看工作区、暂存区状态

4.3.2 添加/撤回>暂存区 (已经 add 的文件,修改后可以直接 commit)

`git add [file name]` /*提交到暂存区,并且转换换行符

将工作区的“新建/修改”**添加到暂存区**

`git rm --cached good.txt` **从暂存区撤回**

4.3.3 提交(这里和安装是选择 vim 编辑器有关)

修改后 4.3.1 查看提示(`git status` 命令)**modified: good.txt**

`git commit file` /*需要输入提交信息日志>写完 `wq`(下面不需要进入 vim)

`git commit -m "commit message" [file name]` 将暂存区的内容提交到本地库

`git restore <file>` 撤销某个文件修改的操作

`git reset HEAD <file>` 可以把暂存区的修改撤销掉(unstage),重新放回工作区

4.3.4 查看历史记录(4 中查询) p14-p15

git log 查看版本记录

```
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(超过了自动多屏)

git log --pretty=oneline 每个历史只显示一行(hash 值和日志)

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

git log --oneline 每个历史只显示一行且显示 hash 的部分值

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


git reflog 显示历史只显示一行,并且显示指针(要移动到版本多少步)

```
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 (三方式) p16-18

本质: 指针移动



```
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 mmmmmmmmm 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
```

1 基于索引值操作[推荐] p17

git reset --hard [局部索引值]

git reset --hard a6ace91

2 使用^符号: 只能后退 p18

git reset --hard HEAD^^

注: 一个^表示后退一步, n 个^表示后退 n 步

3 使用~符号: 只能后退 p18

git reset --hard HEAD~n 注: 表示后退 n 步

git reset --hard HEAD~3 表示后退 3 步

4.3.6 reset 命令的三个参数对比 p19

(查看命令帮助文档: **git help reset**)

--soft 参数

仅仅在本地库移动 HEAD 指针 (查看状态时,绿色提示,本地库和暂存区不同步)



--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 p20-p21,p22 笔记总结

前提: 删除前, 文件存在时的状态提交到了本地库。

进入的文件目录>**rm 文件名**(**rm aaa.txt**)删除本地文件>然后提交到暂存区 **git add aaa.txt**>然后提交到本地仓库 **git commit -m "delete aaa" aaa.txt** (删除完成)

找回操作: **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(回到之前未删除版本)

删除操作尚未提交到本地库: 指针位置使用 HEAD(**git reset --hard HEAD**)

4.3.8 比较文件差异 p23

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]

eg: **git diff HEAD^ apple.txt** 可以用 HEAD 或者版本索引值

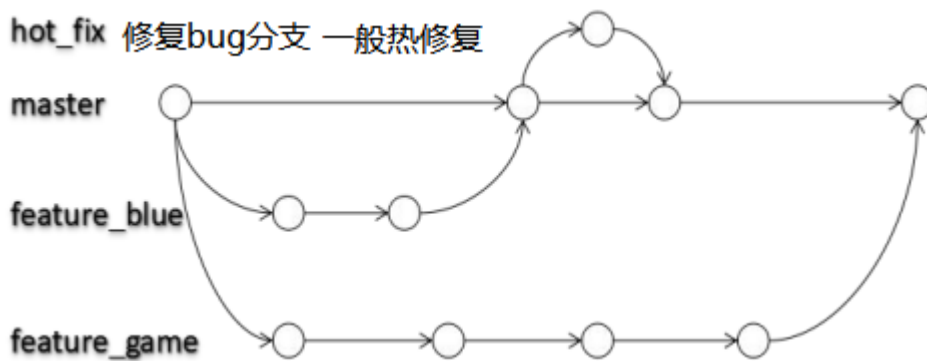
将工作区中的文件和本地库历史记录比较

不带文件名比较多个文件

4.4 分支管理 p24-p26

4.4.2 什么是分支?

1 在版本控制过程中, 使用多条线同时推进多个任务。



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 p25

o1 创建分支: **git branch [分支名]** /*例: **git branch hot_fix**

o2 查看分支: **git branch -v**

o3 切换分支: **git checkout [分支名]** /***git checkout hot_fix**

o4 合并分支: **git merge [有新内容分支名]**

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 master] /*切换分支

第二步: 执行 merge 命令 (合并分支指令)

git merge [有新内容分支名]

hot_fix a865afb hot_fix commit apple /*合并后查询分支效果

* master a865afb hot_fix commit apple

o5 解决冲突 p26

冲突原因: 2 个分支,修改同一文件,同一位置,修改内容不一样时.

冲突的表现:

```

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

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

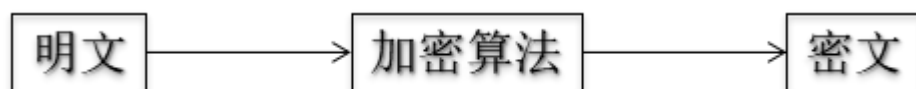
第三步: **git add [文件名]**

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 P27-p29

5.1 哈希 p27

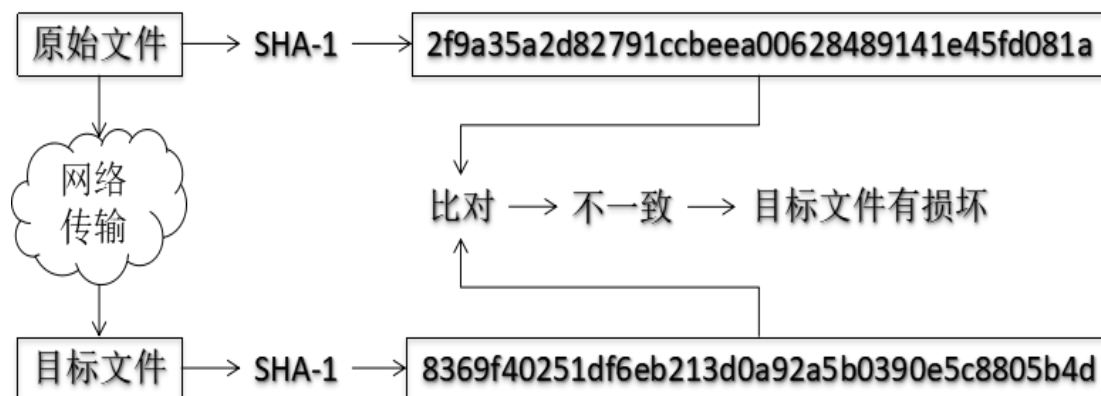


哈希是一个系列的加密算法，各个不同的哈希算法虽然加密强度不同，但是有以下几个共同点：

- 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：(传输前后 hash 值对比)

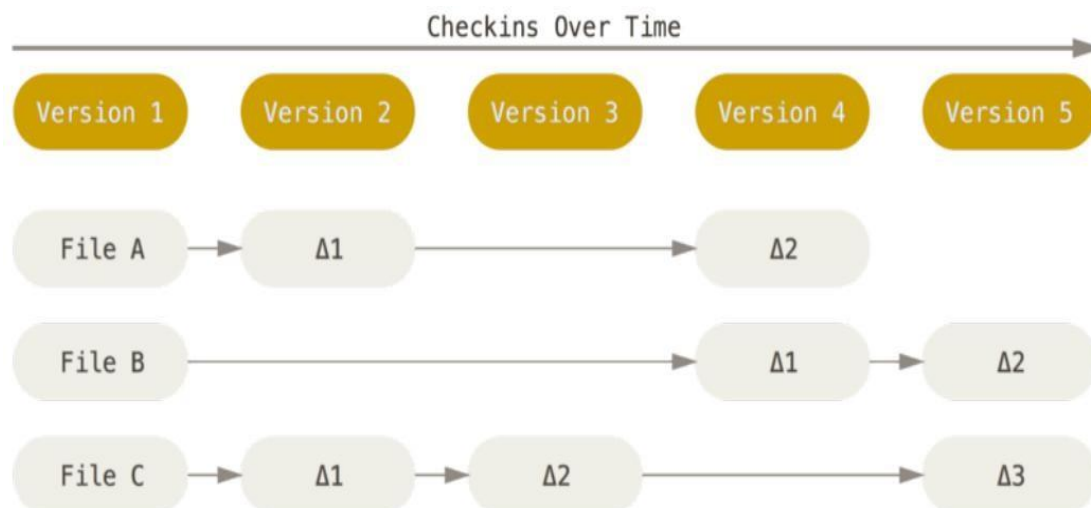


Git 就是靠这种机制来从根本上保证数据完整性的。

5.2 Git 保存版本的机制

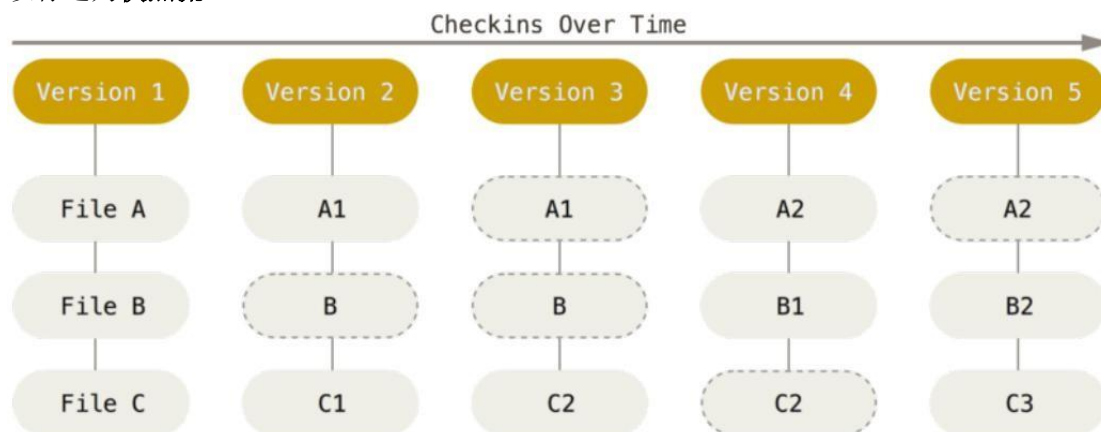
5.2.1 集中式版本控制工具的文件管理机制

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。SVN



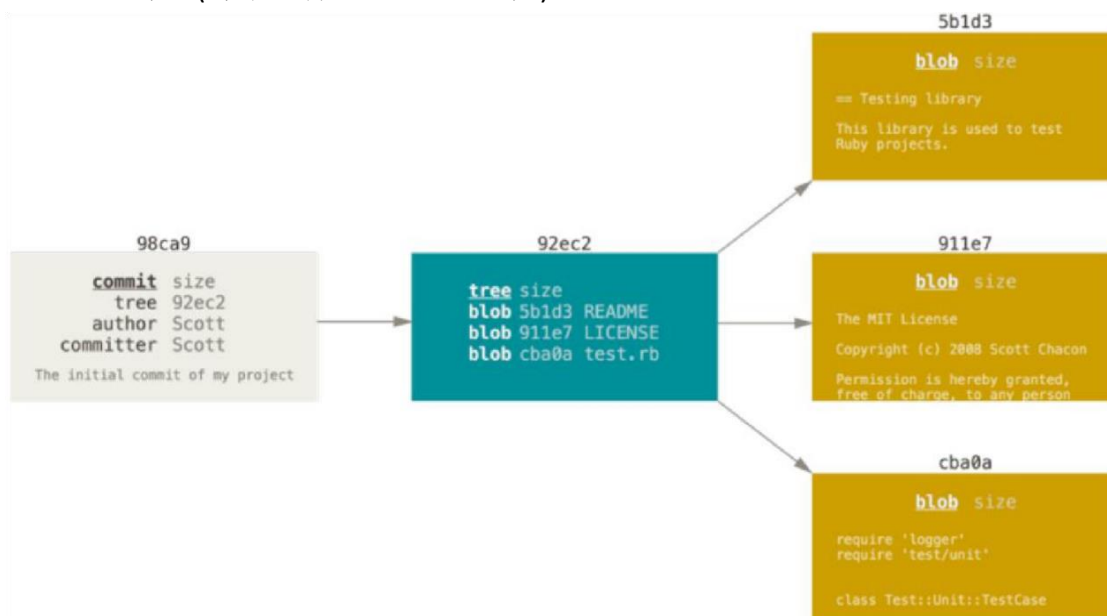
5.2.2 Git 的文件管理机制

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。

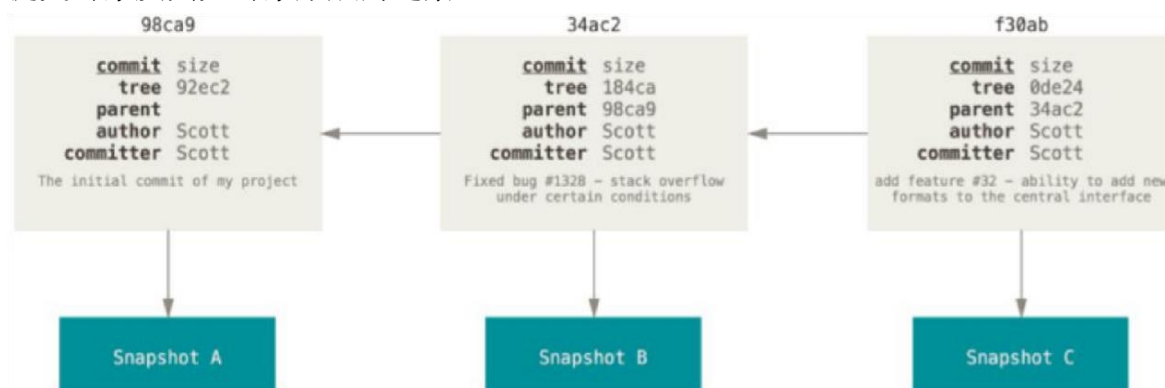


5.2.3 Git 文件管理机制细节

Git 的“提交对象” (每个文件对应的 hash 值)



提交对象及其父对象形成的链条

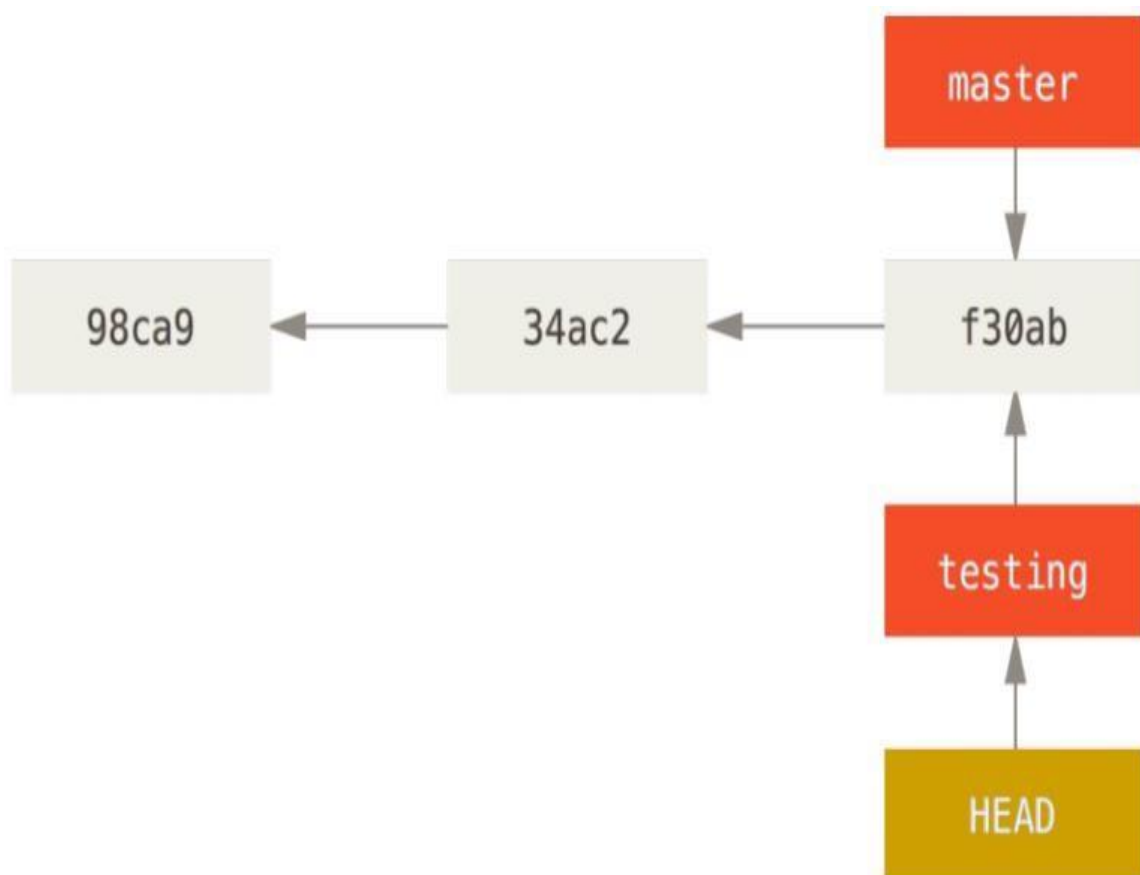


5.3Git 分支管理机制 p29

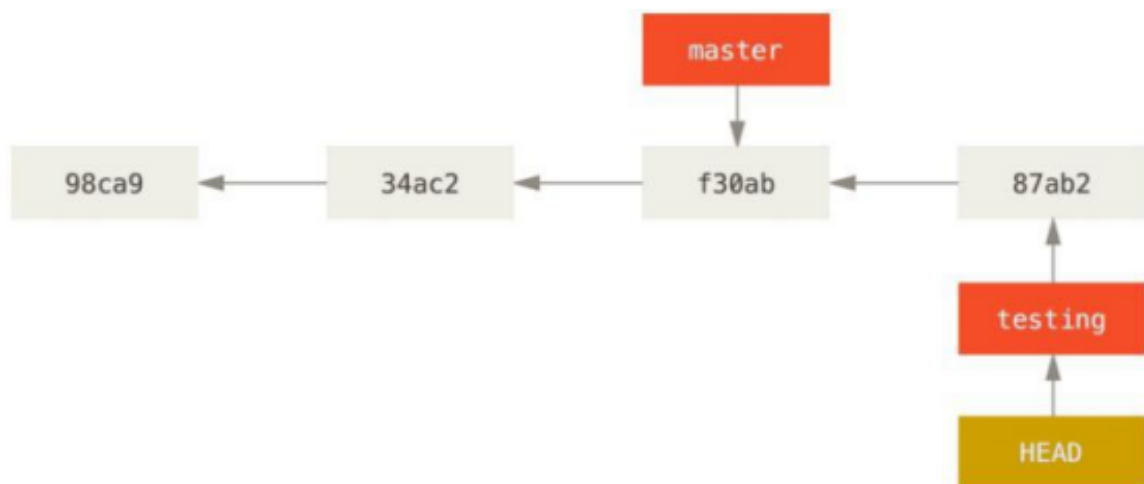
5.3.1 分支的创建(就是新建一个指针)



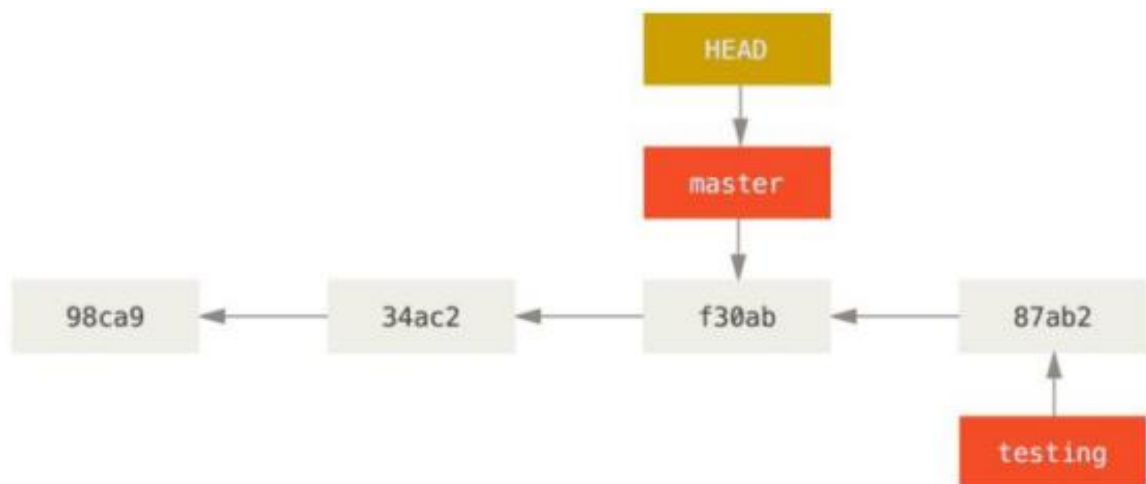
5.3.2 分支的切换 (HEAD 指向)



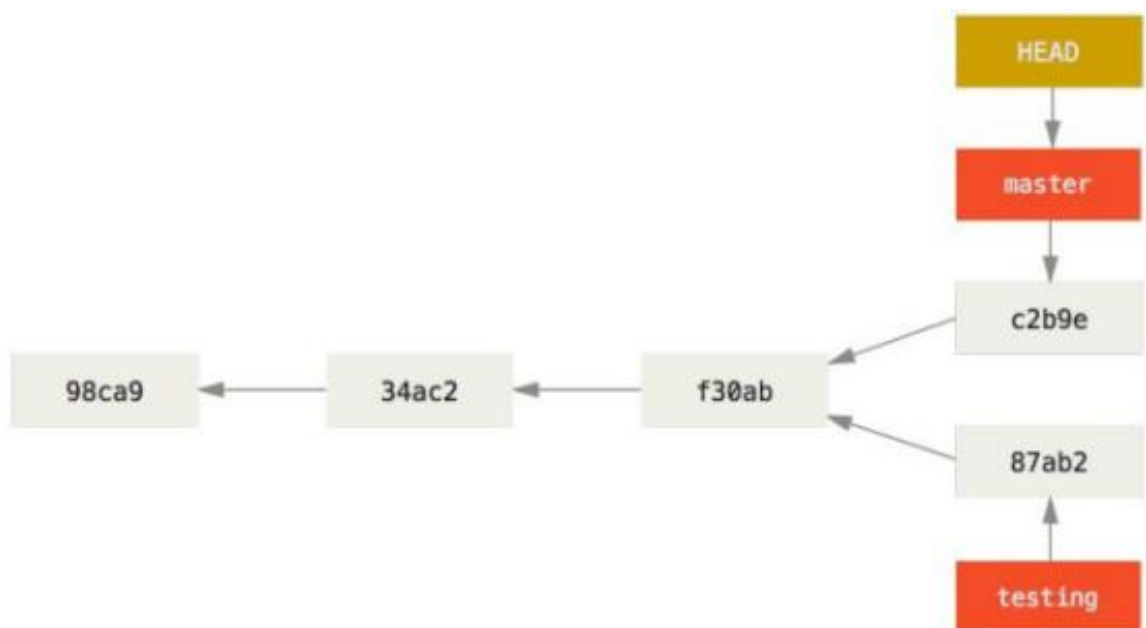
3 HEAD 指向 testing 时提交了内容



4 切换回 master



5 HEAD 指向 master 时提交了数据



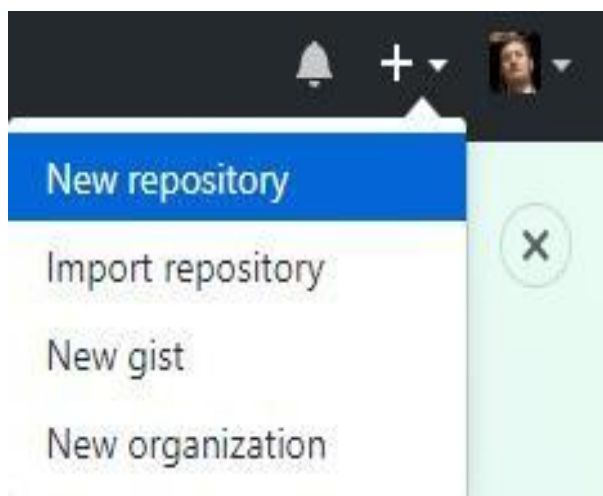
6 GitHub p30-p42

6.1 账号信息

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

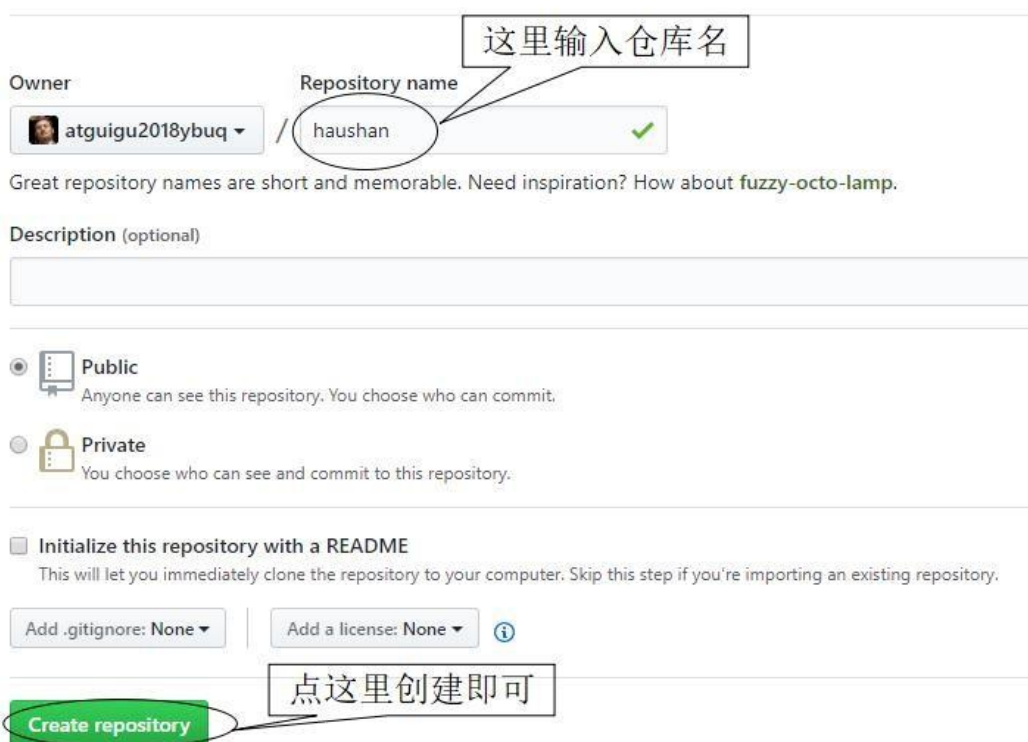
	Email 地址: atguigu2018ybuq@aliyun.com GitHub 账号: atguigu2018ybuq
	Email 地址: atguigu2018lhuc@aliyun.com GitHub 账号: atguigu2018lhuc
	Email 地址: atguigu2018east@aliyun.com GitHub 账号: atguigu2018east

6.2 创建远程库 p34

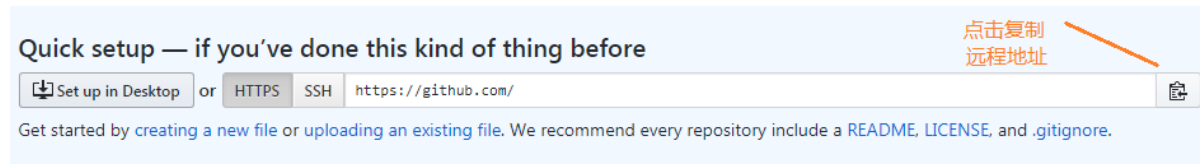


Create a new repository

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



6.3 创建远程库地址别名 p35



git remote -v 查看当前所有远程地址别名

git remote add [别名 origin] [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)
```

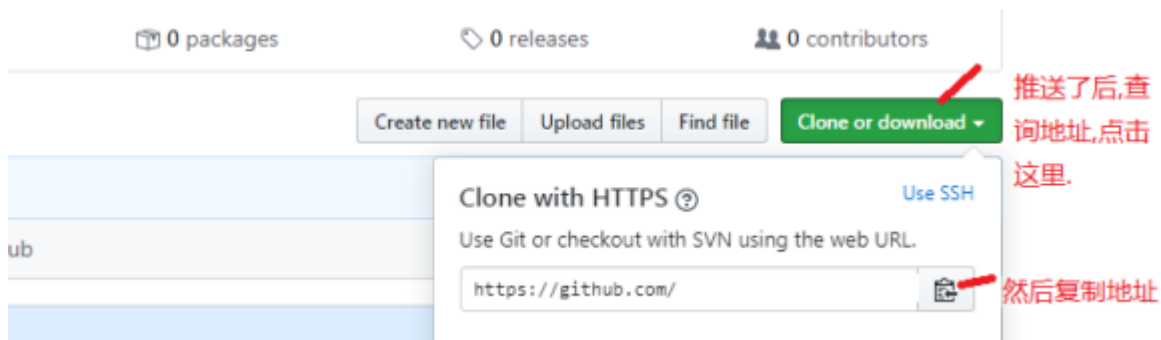
6.4 推送 p36

git push [别名] [分支名] **git push origin master** /*回车可能需要等待一会会,弹出对话框>输入用户和密码)

```
$ 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 p37

命令 **git clone [远程地址]** /*



成功:

```
$ 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.
```

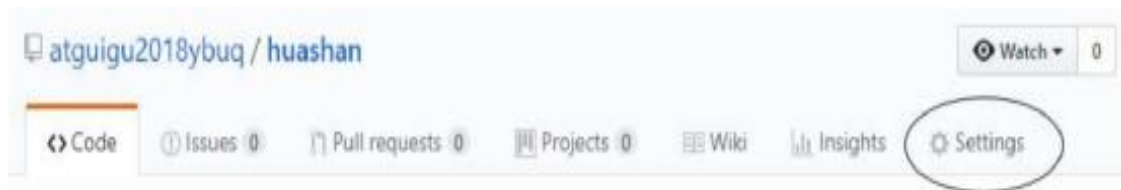
克隆效果:

完整的把远程库下载到本地

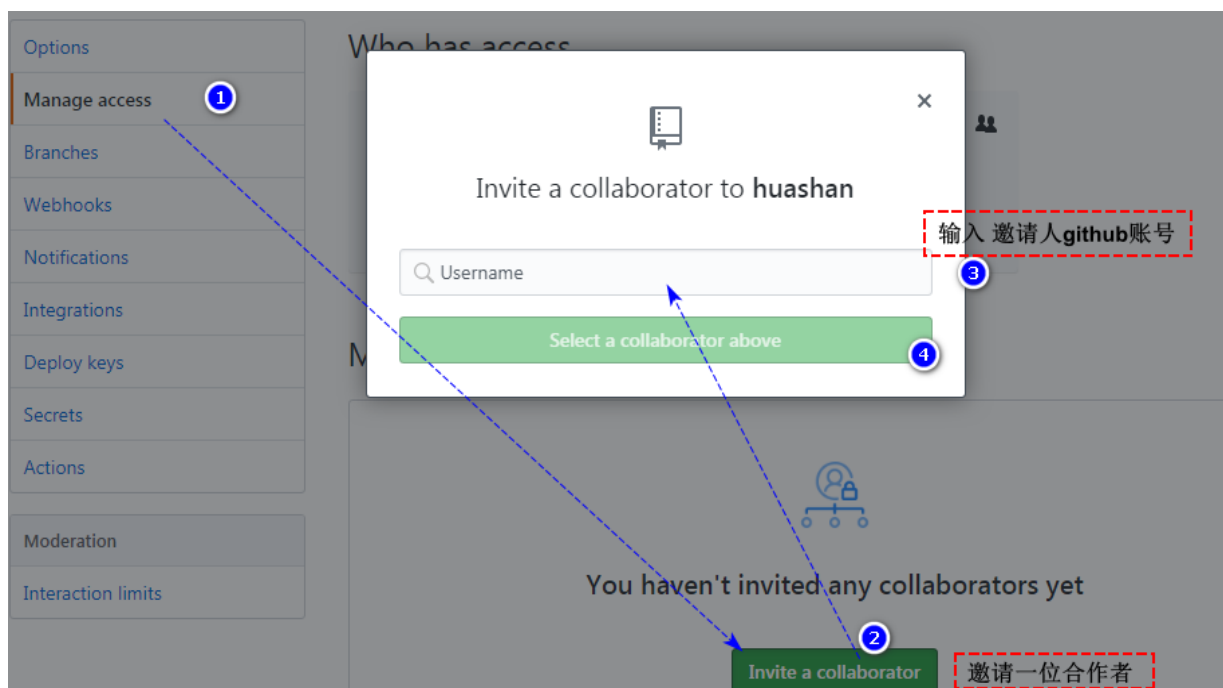
创建 **origin** 远程地址别名 (git remote -v 查看远程库别名)

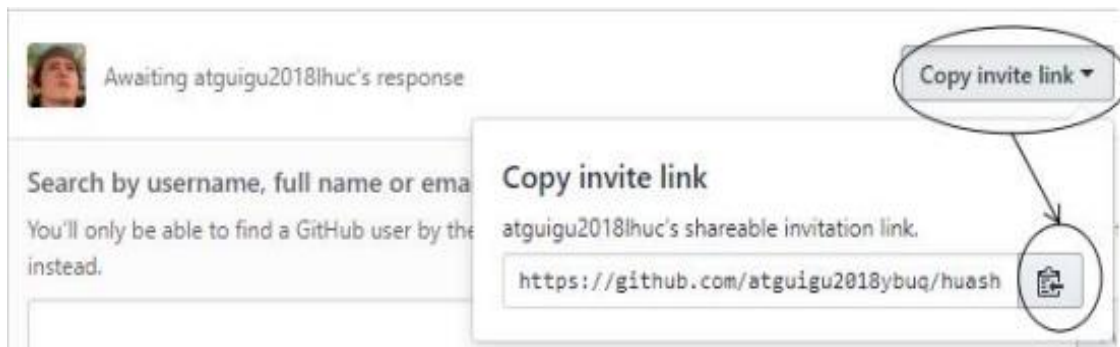
初始化本地库(就是:git init)

6.6 团队成员邀请(邀请用户才能提交)p38



和老师笔记不一样的地方





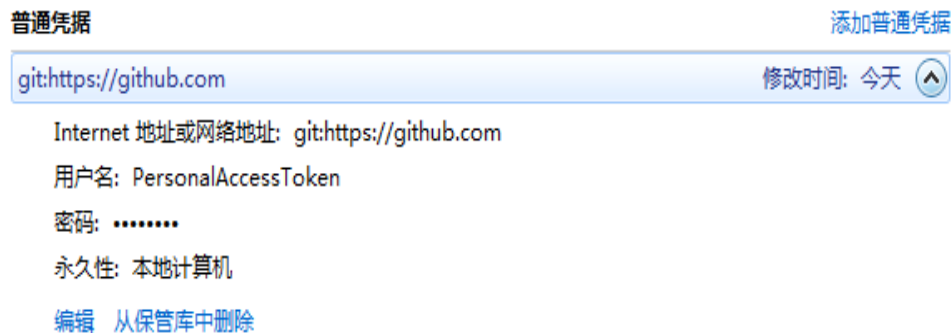
“岳不群”其他方式把邀请链接发送给“令狐冲”，“令狐冲”登录自己的 GitHub 账号，访问邀请链接。

点击接受>然后在执行推送



:推送了第一次在此推送不要输入用户名:git 本身不具备记录功能,Windows 中凭据管理器记录用户名和密码

控制面板\所有控制面板项\凭据管理器(如果想切换用户:删除记录)



lhc>提交后>>然后推送 `git push origin master`

6.7 拉取 p39

pull=fetch+merge

`git fetch [远程库地址别名 origin] [远程分支名 master]` /*抓去下来

`git checkout origin/master` /*切换到链接地址(别名)的 master(可查看抓取下来内容
切换回 `git checkout master`

`git merge [远程库地址别名 origin/master 远程分支名]` /*合并

`git pull [远程库地址别名] [远程分支名]` /*等于上面步骤

6.8 解决冲突 p40

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

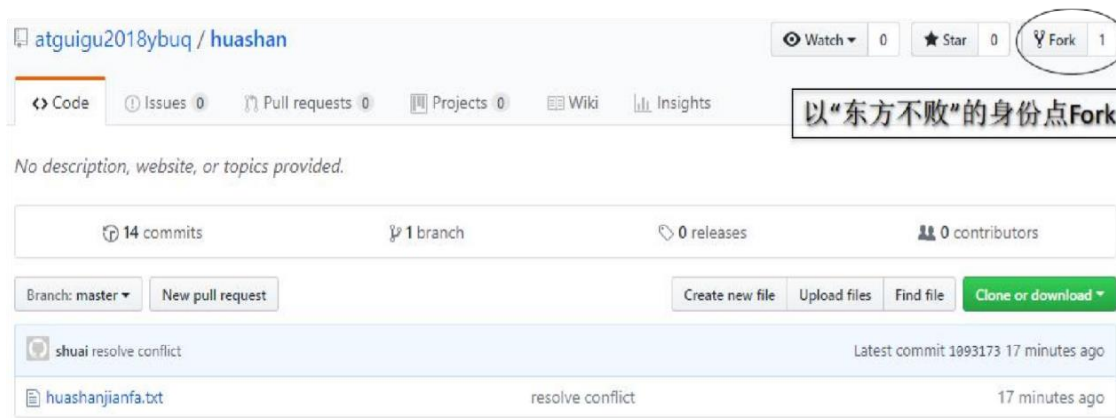
债务人：小刘

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。

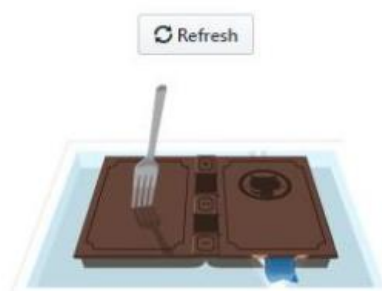
6.9 跨团队协作 p41

1(先复制**当前库地址**,发式给 dfbb,然后有 dfbb 登录访问这个地址)>然后 **Fork**



Forking atquigu2018ybuq/huashan

It should only take a few seconds.



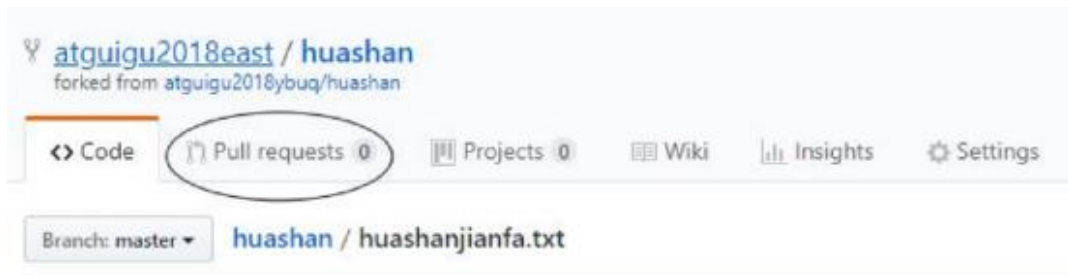
正在 fork 的界面

fork 过来的仓库说明 回多下面一行(forked from at...)说明 fork 来源



2 dfbb("东方不败")本地修改，然后推送到远程 `git push origin master`

3 dfbb 在远程库中选择 Pull Request



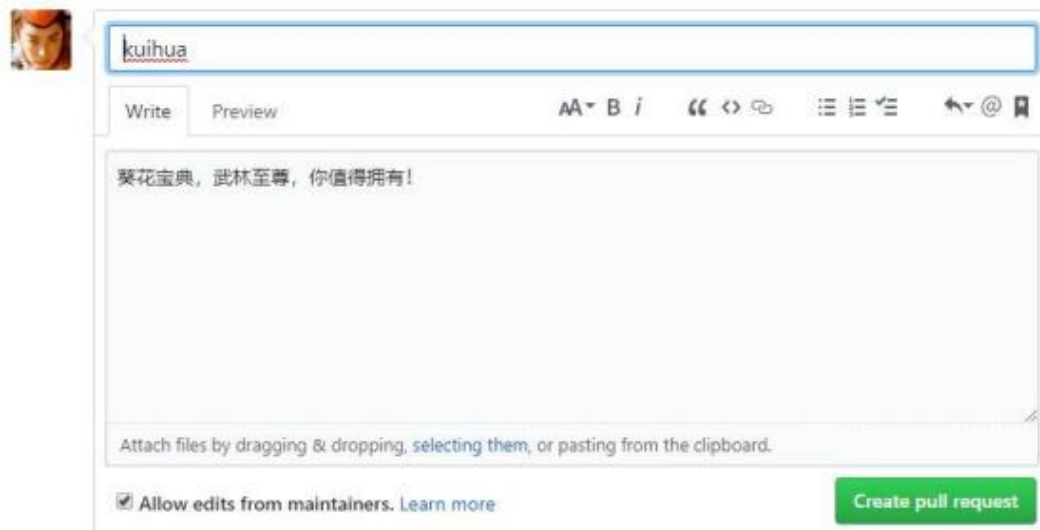
New pull request

3.2 然后点击里面的 New pull request

Create pull request

3.3 然后点击 Create pull request

3.4 然后发送消息给 fork 的库(ybq(岳不群))



4 ybq 操作



5

5.2 对话（这时还可以相互对话）



Write

Preview

AA B i “ ” ↻ ☰ ☷ ☹ ↶ @

老东，你这代码靠谱吗？练了会不会有什么危险？

Attach files by dragging & dropping or [selecting them](#).

 Styling with Markdown is supported

Close and comment

Comment



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？

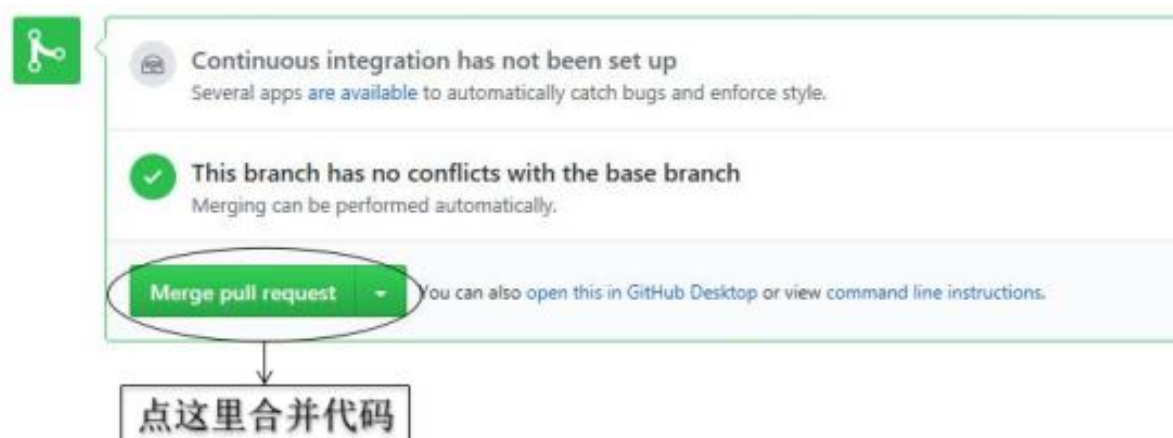


atguigu2018east commented just now

你放心吧，我都亲自练过啦！

6 审核代码

合并代码 (回到对话 Conversation>>合并操作如图)



上面操作完了就远程库就有合并内容>然后>将远程库修改拉取到本地
6.10SSH 登录记录用户(http 地址如果不能记录登录用户) p42

1 进入当前用户的家目录

```
$ cd ~
```

2 删除.ssh 目录

```
$ rm -rvf .ssh
```

3 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] 3.2 后面直接回车(使用默认)

4 进入.ssh 目录查看文件列表

```
$ cd .ssh
```

```
$ ls -lF
```

5 查看 id_rsa.pub 文件内容

```
$ cat id_rsa.pub
```

6 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

然后>>key 中输入复制的密钥信息 Title 自定义输入标题

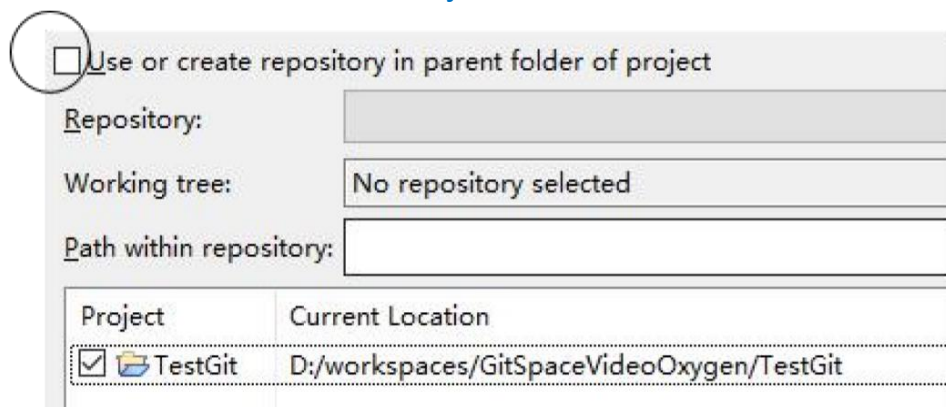
7 回到工作区 cd > 创建远程地址别名

```
git remote add origin_ssh git@github.com:atguigu2018ybuq/huashan.git
```

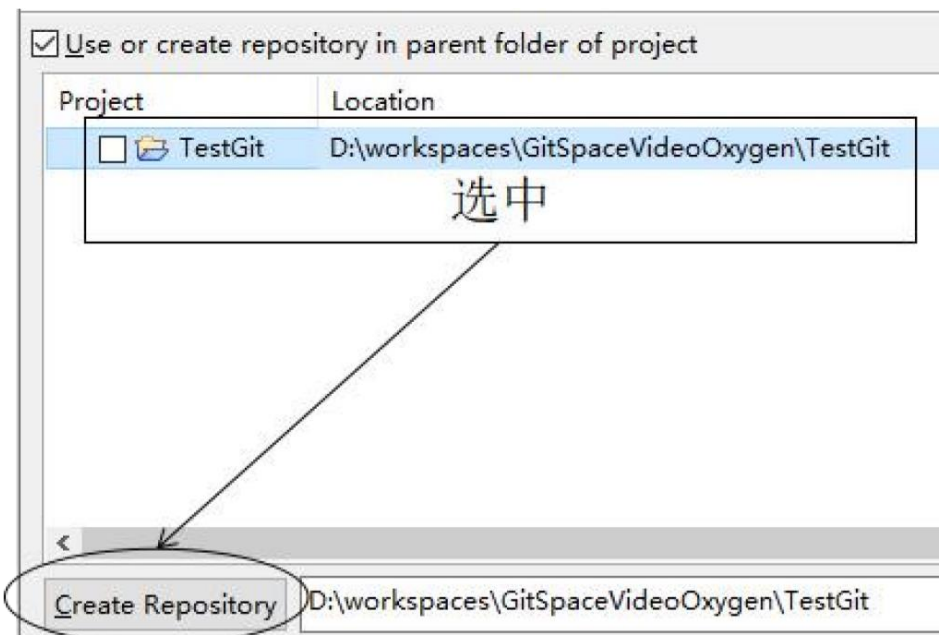
8 推送文件进行测试

7 Eclipse 操作 p43-p53

7.1 工程初始化为本地库 (p44) 先创建一个 Maven 工程,
工程→右键→Team→Share Project→Git



2 Create Repository

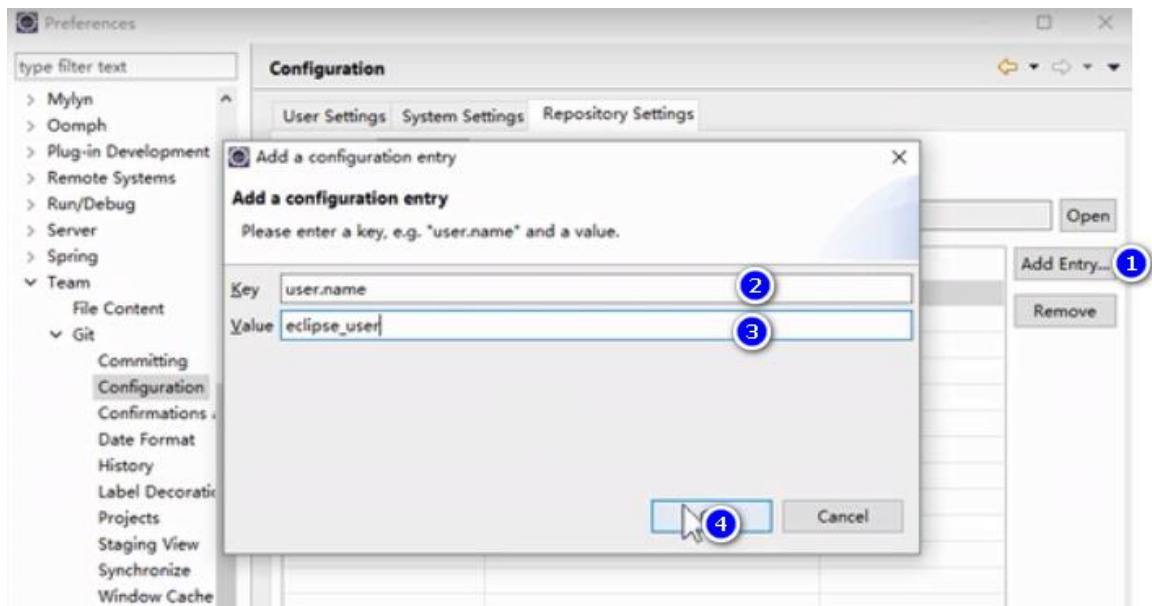


3 Finish

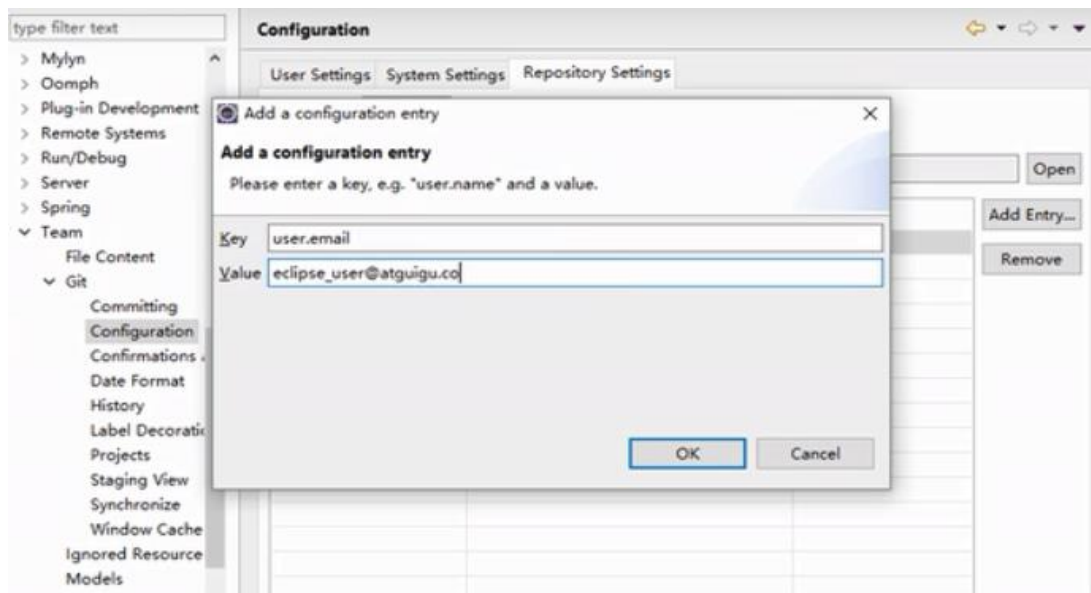
7.1.2 Eclipse 中设置 签

名:window>prefer.>Team>Git>Configuration>Reposit....

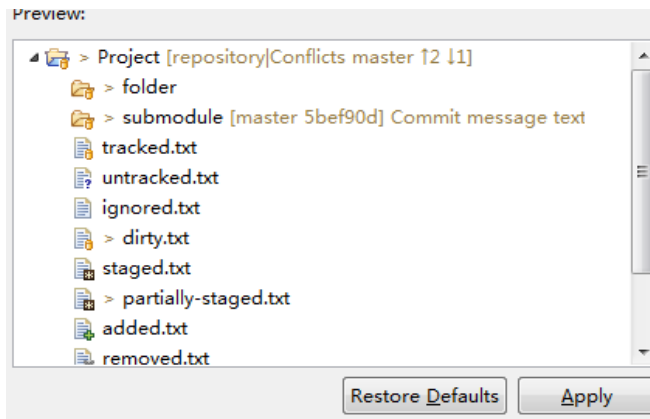
1 设置用户名字 p45



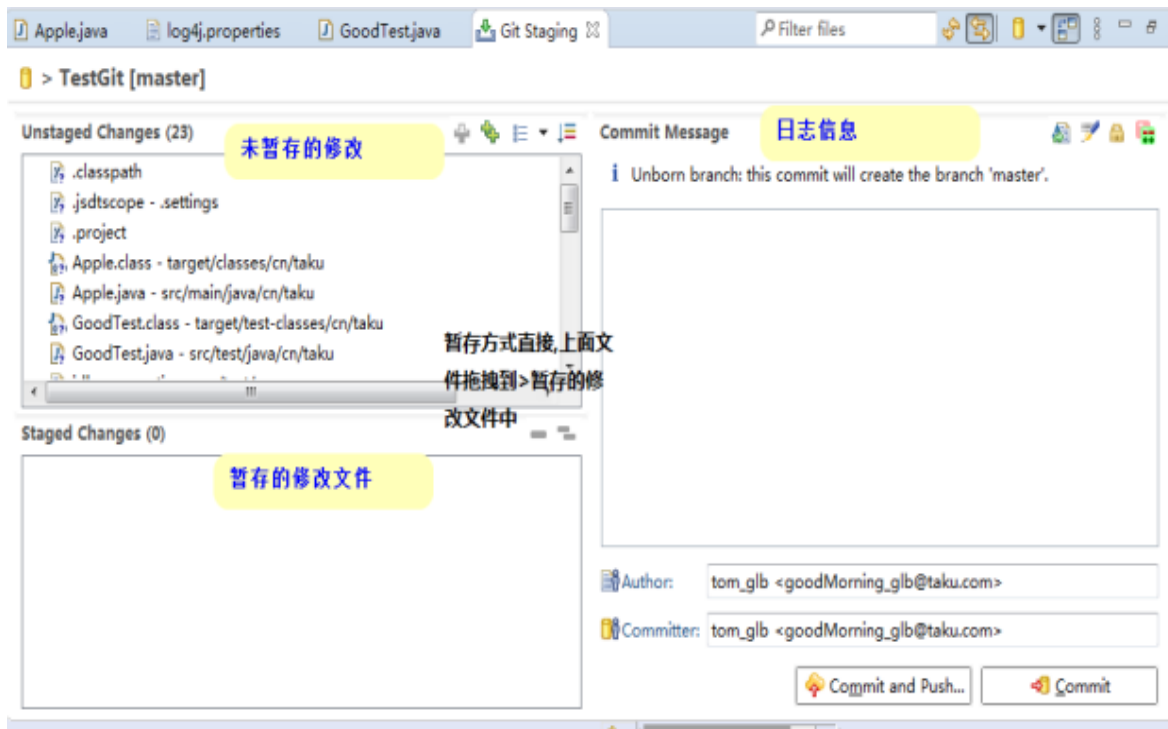
2 设置用户 email



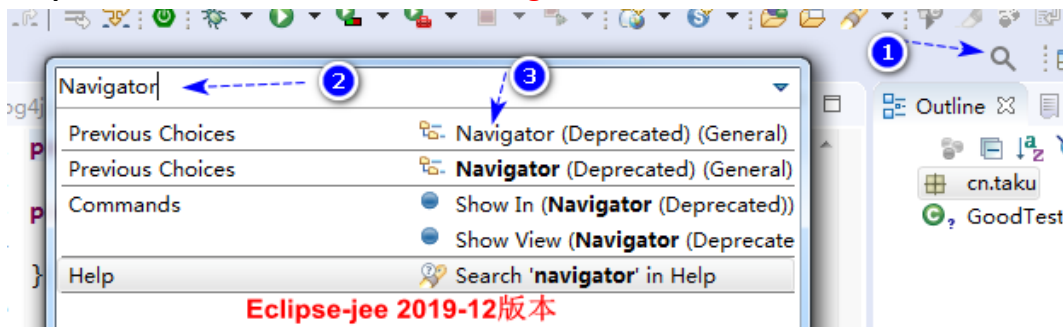
Eclipse 查看 git 文件标识说明: window>prefer.>Team>Git>Label D.. p46



Eclipse:文件追踪添加到缓存区:文件/项目> 右键>>Team>Commit p47



Eclipse 查看项目中的所有文件的 **Navigator** 窗口打开



7.2Eclipse 忽略文件 p47-49

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。

(文件忽略)GitHub 官网样例文件 p48

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

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

文件忽略具体步骤 p48

1 编辑本地忽略配置文件，文件名任意(eg:在家目录下创建 Java.gitignore)

1.2 Java.gitignore 文件编辑如下：

```
# Compiled class file *
.class

# Log file *
.log

# BlueJ files *
.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# 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
```

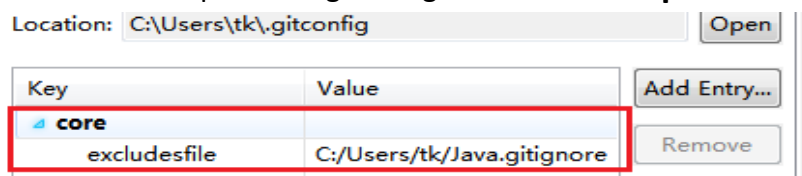
2 在~/.gitconfig 文件中引入上述文件 :如下

[core]

excludesfile = C:/Users/Lenovo/Java.gitignore

[注意：这里路径中一定要使用“/”，不能使用“\”]

3 完成后在 Eclipse 查看.gitconfig 配置:window>prefer.>Team>Git>Configuration

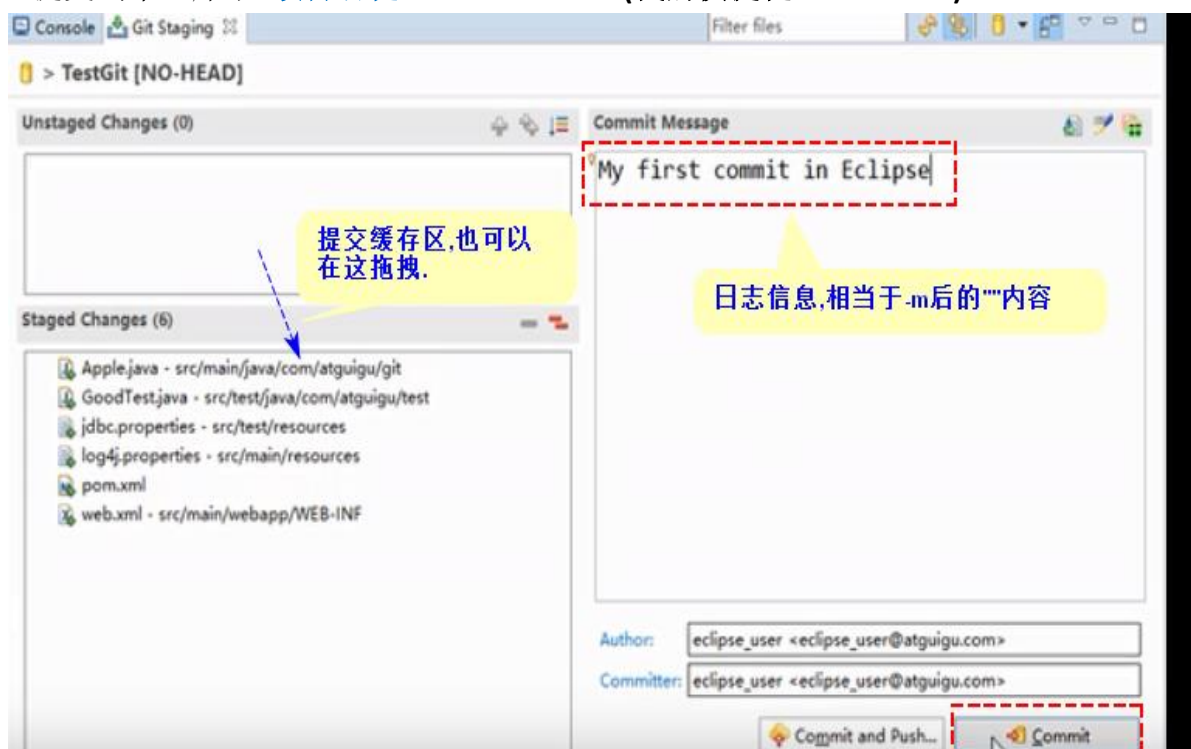


3.2 重启 eclipse >Navigator 窗口查看忽略是否成功

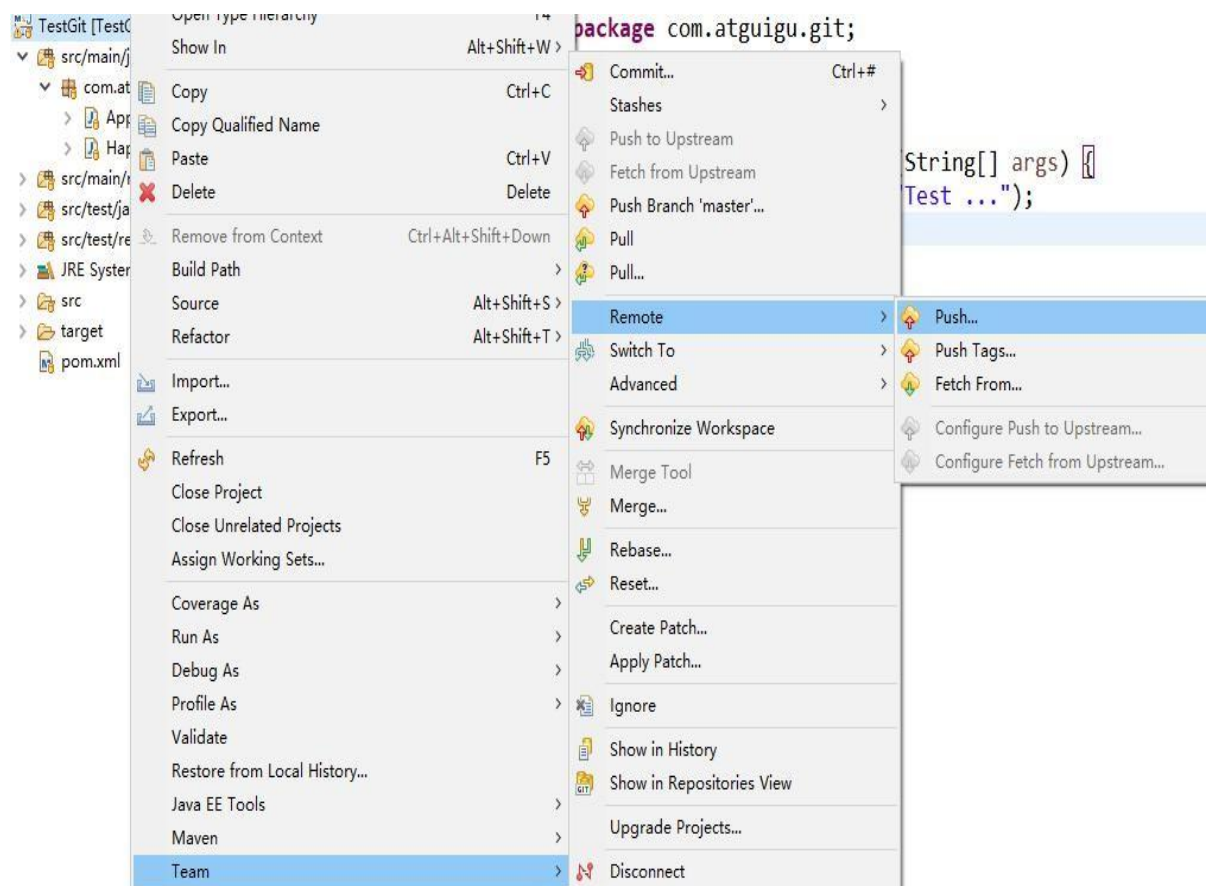
7.2.2 提交暂存区/和提交到库 p49

1 提交到暂存区:项目右键>Team> add to Index (或者上面 p47 添加到缓存区)

2 提交到本地库中: 项目右键>Team>Commit (我的快捷键:ctrl+shift+#)



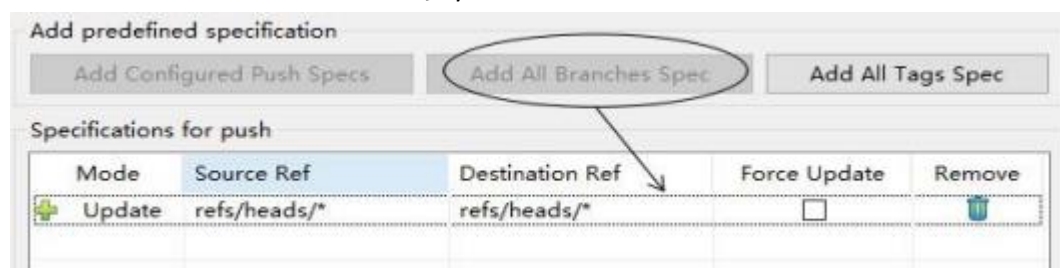
7.3 推送到远程库 p50 (:在 GitHub 创建新的远程库, 名为 TestGit)



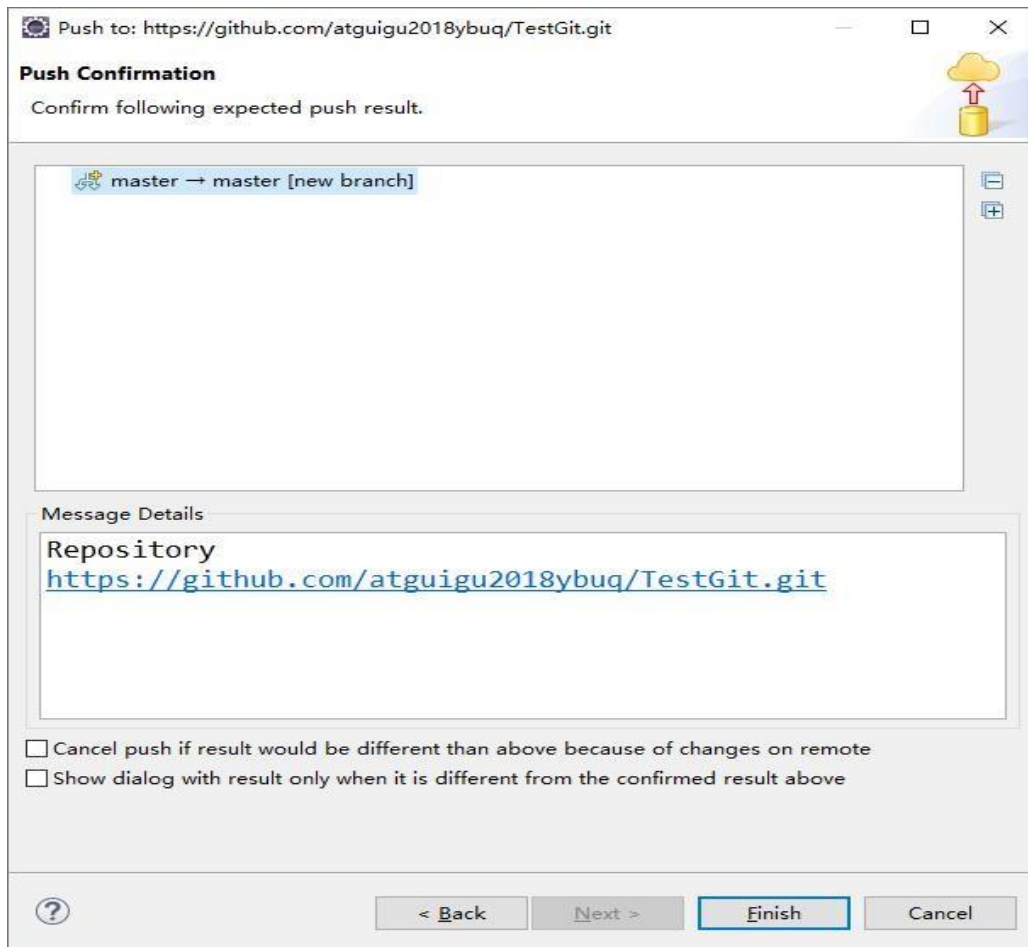
URI: 粘贴在 GitHub 复制的地址,后面 2 个默认;User 账号,下面:密码



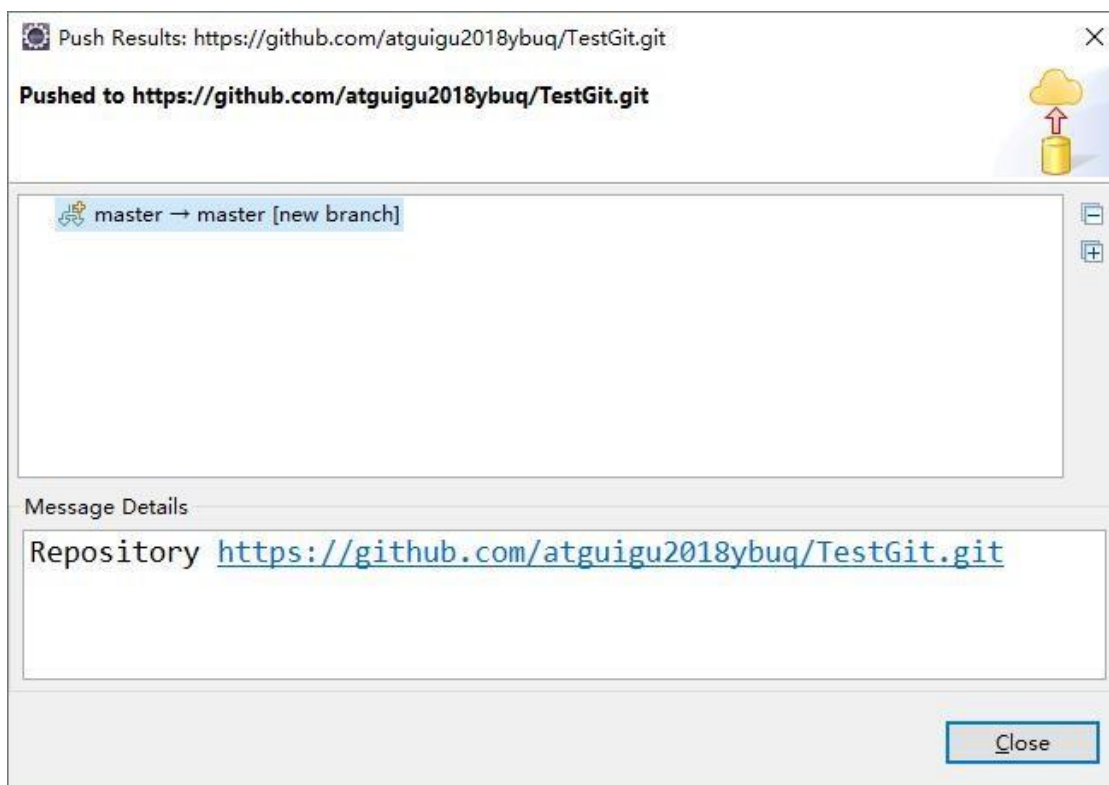
点击 Add All Bran... > 然后下一步/或者直接 Finish



可以添加一些日志信息>Finis

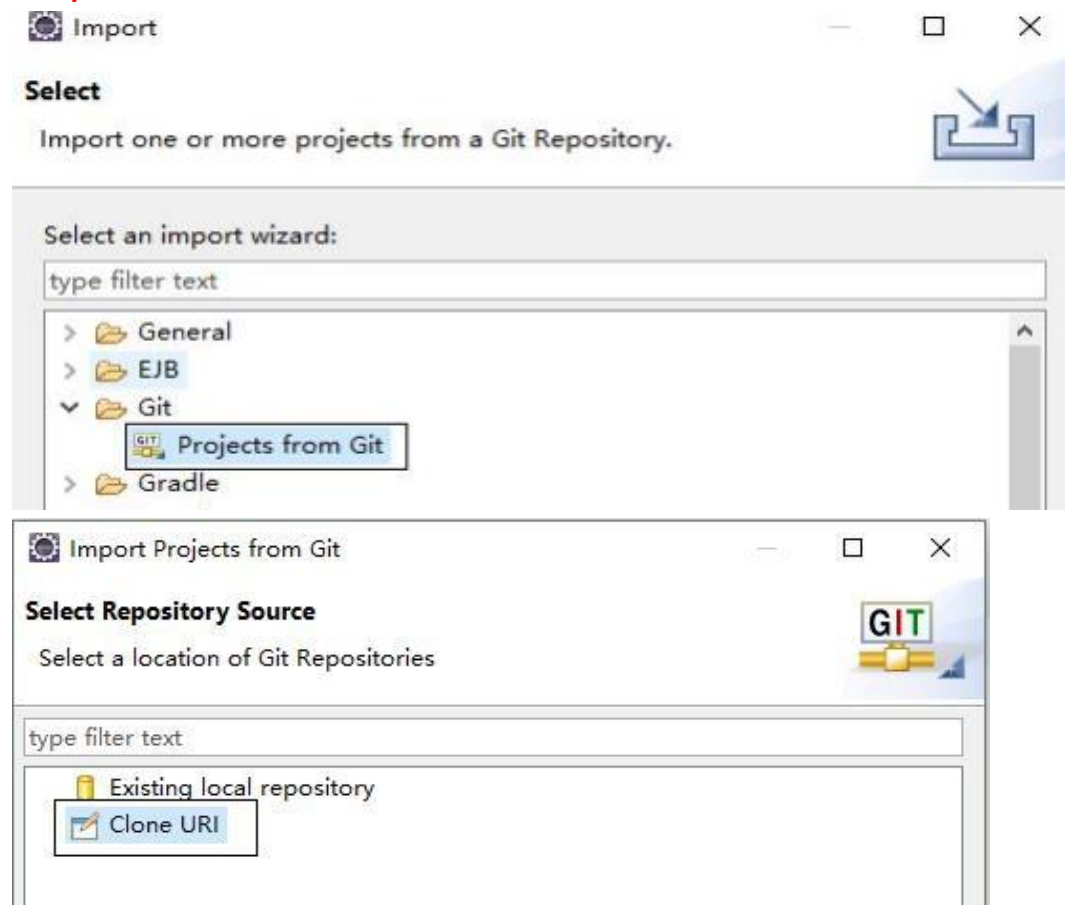


执行成功的结果

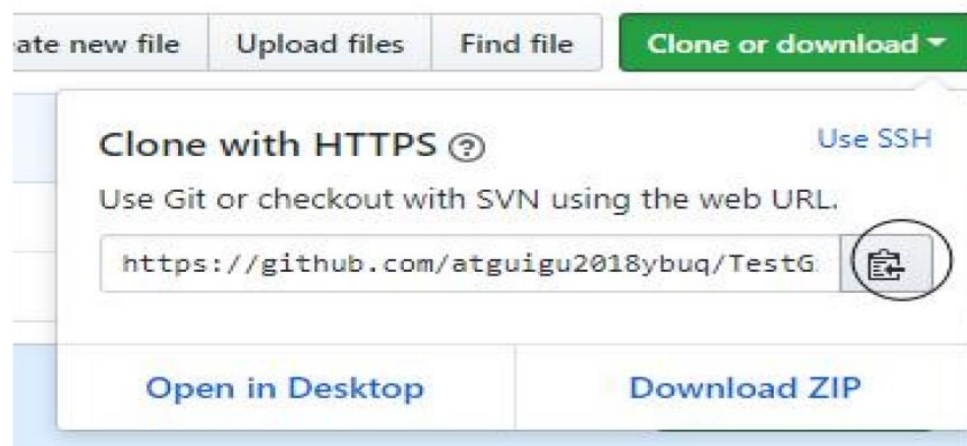


7.4Oxygen Eclipse 克隆工程操作 p51


1 Import...导入工程




2 到远程库复制工程地址



3 粘贴到 URL 如下:>然后点击 next

 Import Projects from Git

Source Git Repository
Enter the location of the source repository.



Location

粘贴

URI:

Host:

Repository path:

Connection

Protocol:


Port:

Authentication

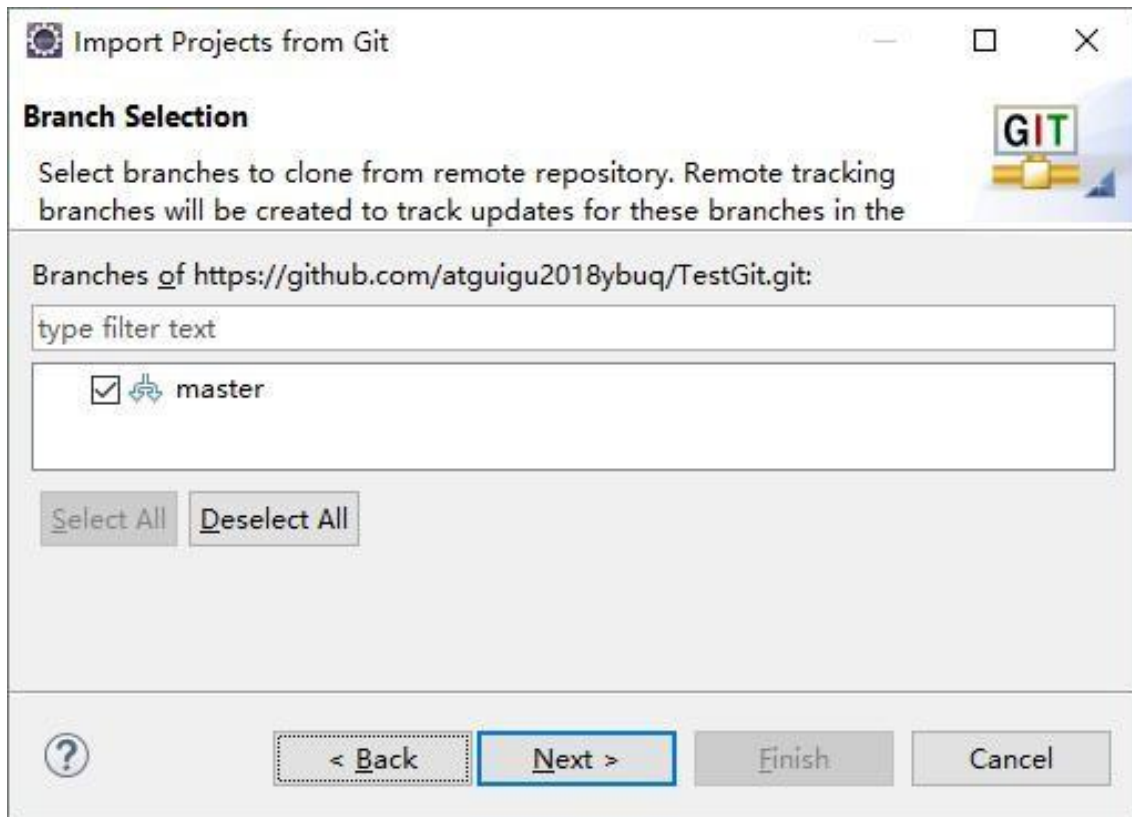
User:

Password:

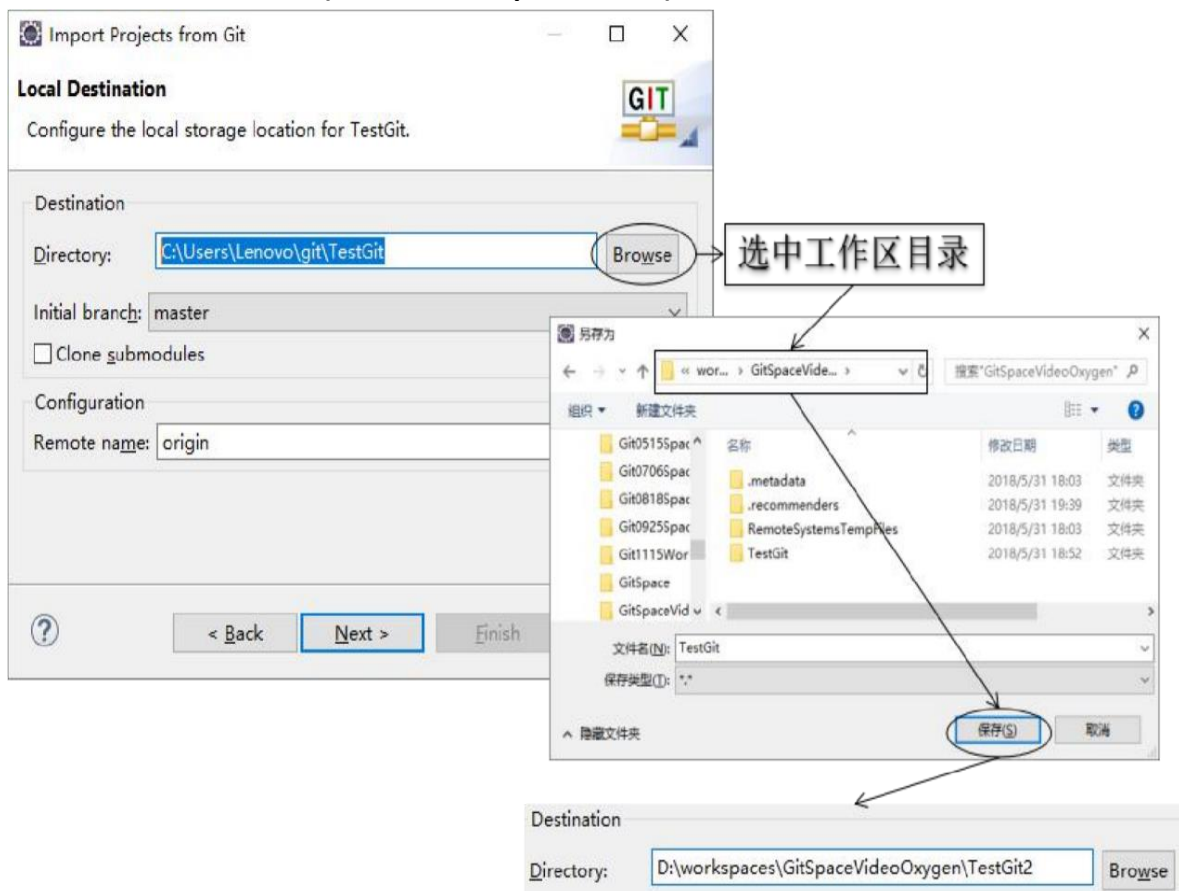
☒ Store in Secure Store



4 选择分支 Next

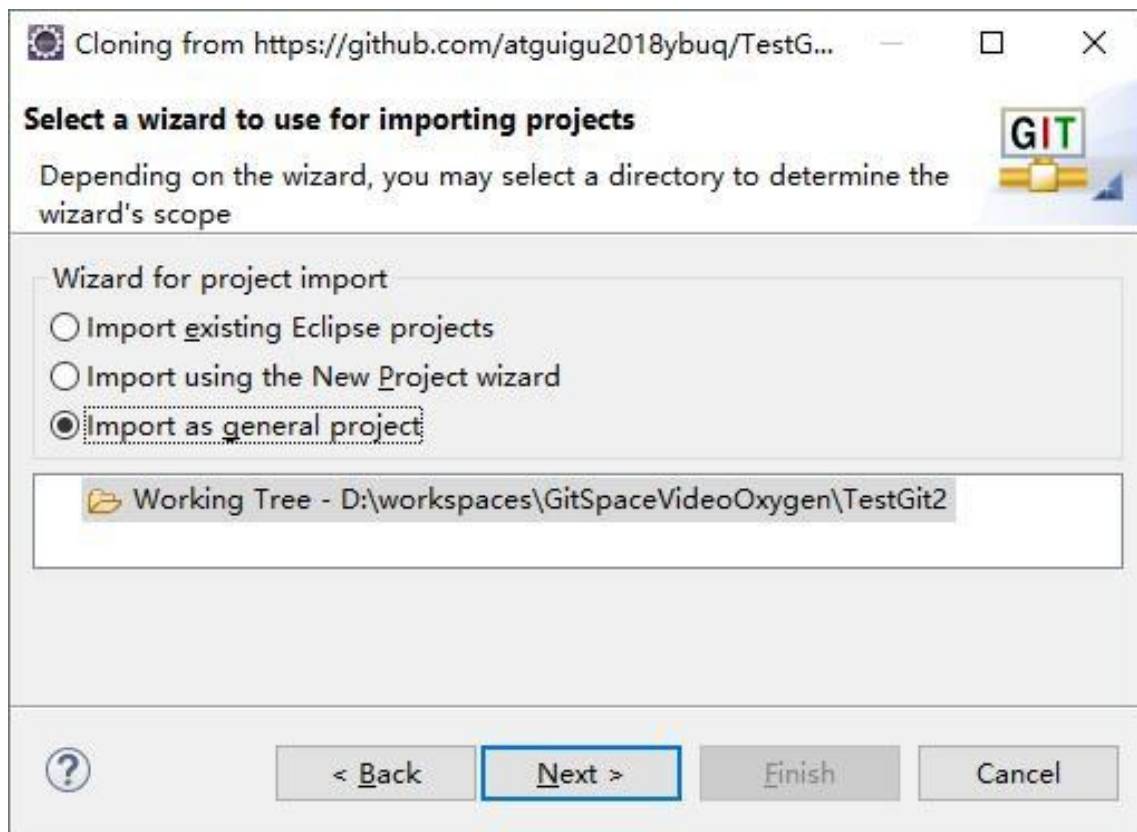


5 指定工程的保存位置(最好选择 eclipse 工作区) 然后 next

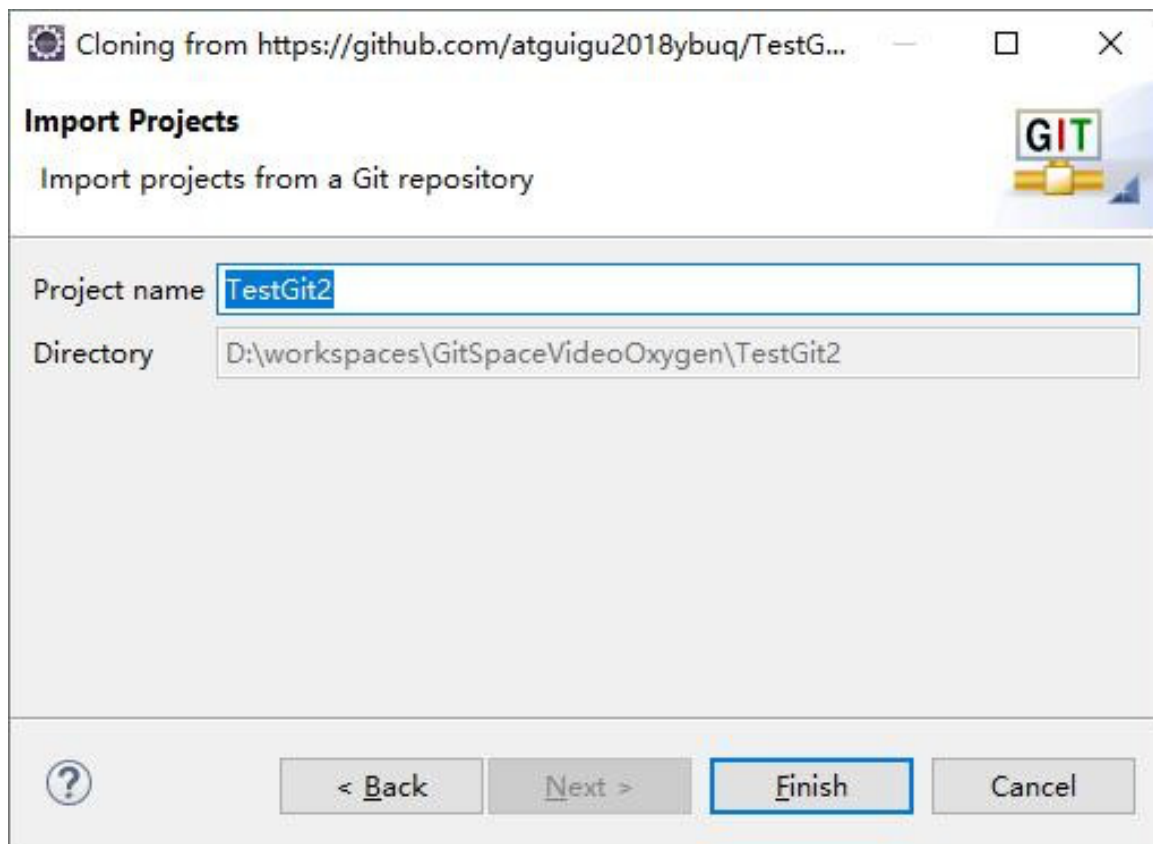


等待下载

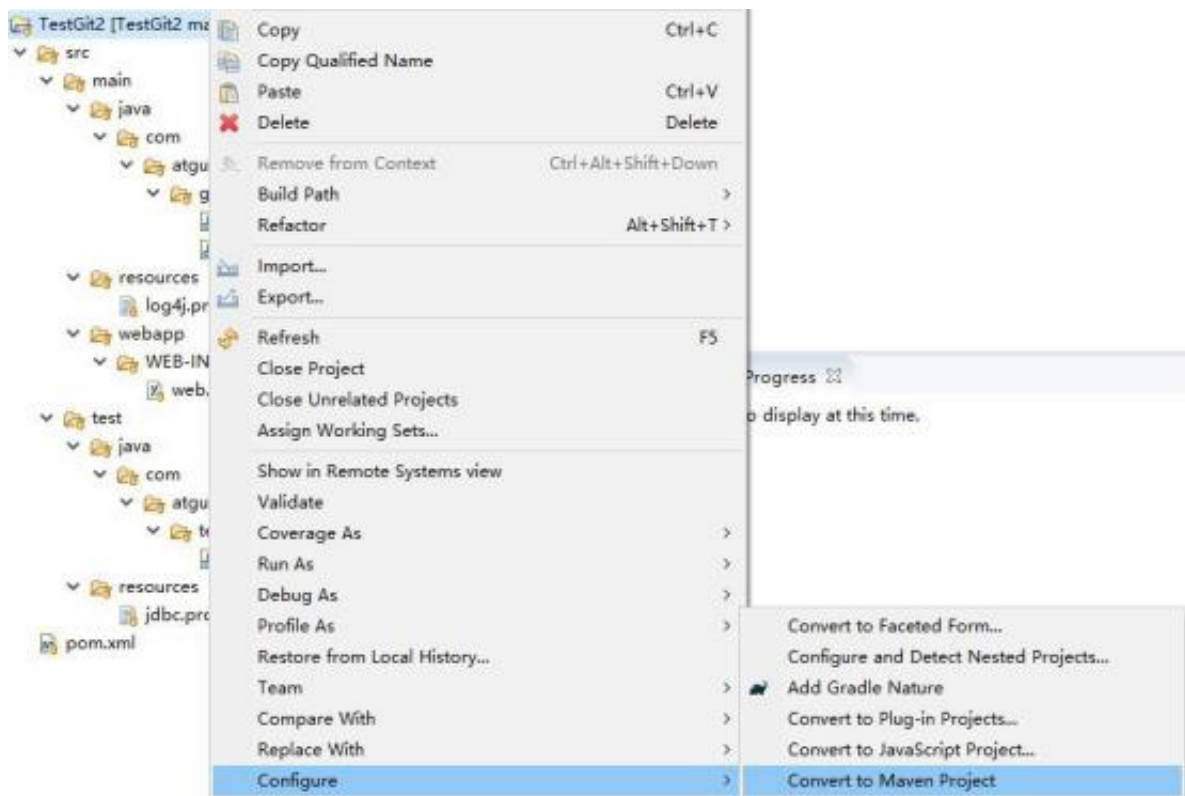
6 指定工程导入方式, 这里只能用: **Import as general project** (作为普通工程导入)



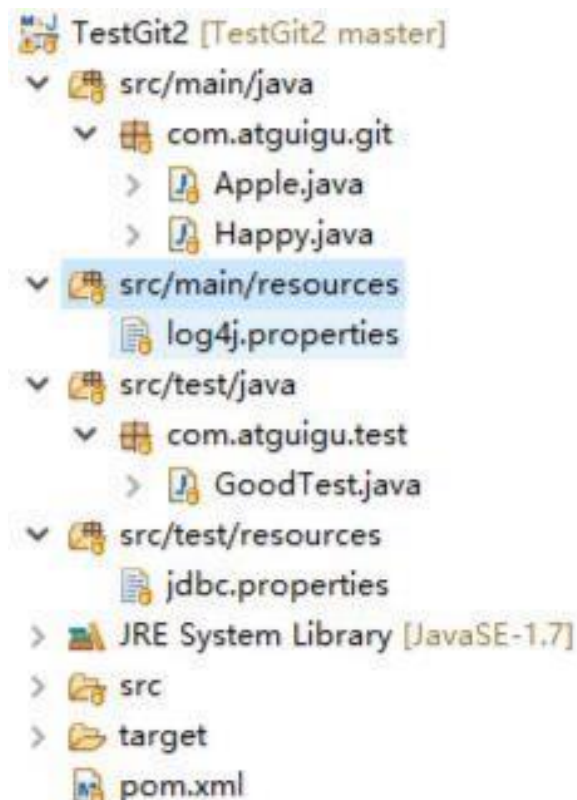
7 点击 Finish



8 转换工程类型 (导入进来的不适合编写.需要修改 Eclipse 工程)



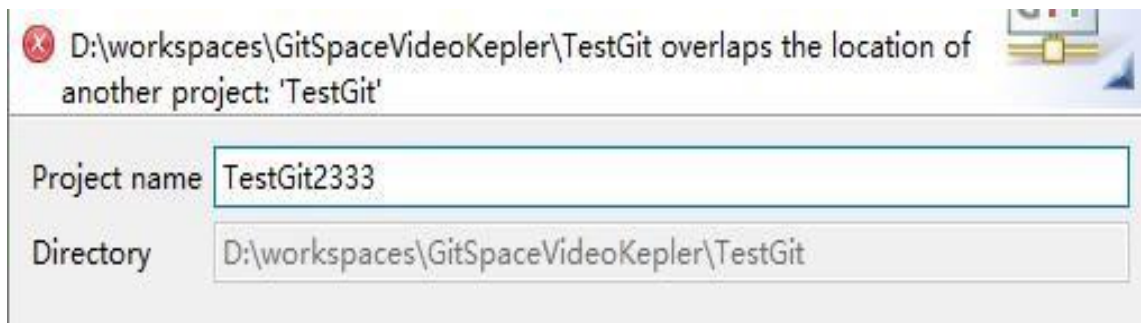
9 最终效果



7.5Kepler Eclipse 克隆工程操作 p52(版本比较低的 Eclipse)

问题：不能保存到**当前 Eclipse 工作区**目录 (和上面第 5 步不同)

放在工作区[第 7 步时]不能导入,如下图:



正确做法：保存到工作区以外的目录中,其他的和上面都一样



7.6 解决冲突 p53

Eclipse 制造冲突

- 1 分别修改 TestGit 和 TestGit2 同一文件同一位置
- 2 然后:都提交到本地库,快捷 **ctrl+shift+#**(他们两人个是个的本地库)
- 3 然后 TestGit 推送到远程库(这里不是首次推送和上面不同 如下:)

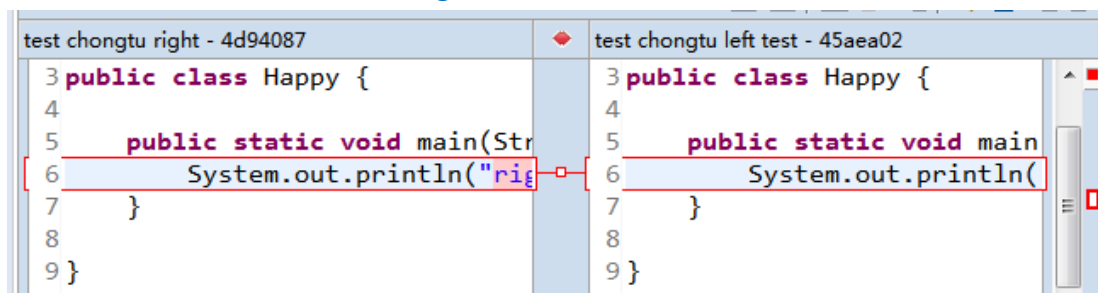
工程右键>Team>Push Branch 'master')(所以默认直接下一步)>>等条读完(弹出对话框)

4 这时 TestGit2 不能推送,会出现这个图 ,因为不是最新版本,

更新:右键>Team>pull

Eclipse 解决冲突

1 冲突文件→右键→Team→Merge Tool



2 修改完成后 Eclipse 正常执行[该文件] **add/commit** 操作即可

8 Git 工作流(3 种)p54-

8.1 概念在项目开发过程中使用

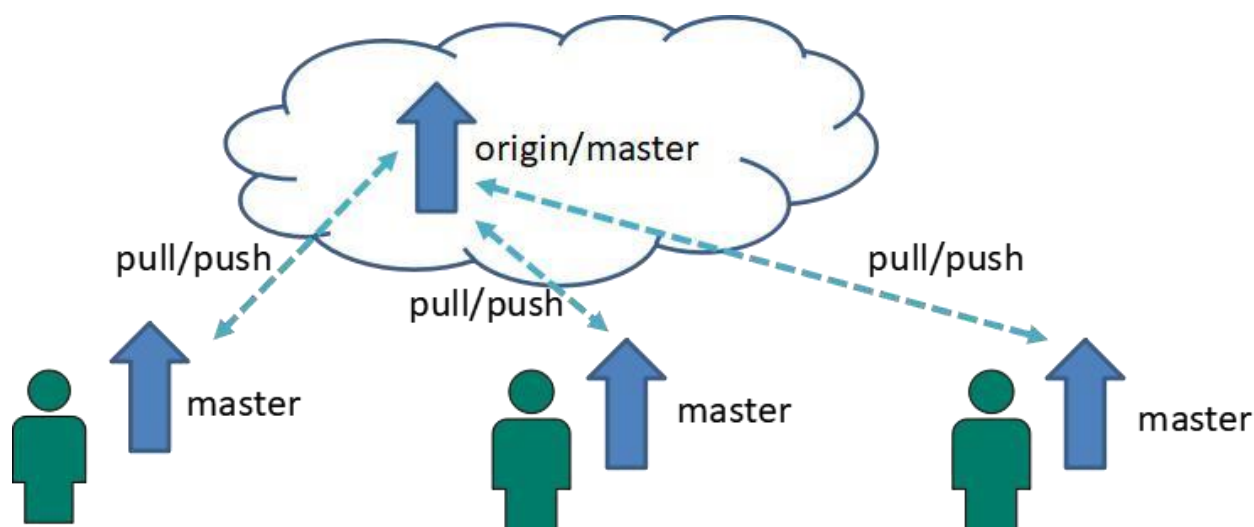
Git 的方式

8.2 分类

8.2.1 集中式工作流像 () p54

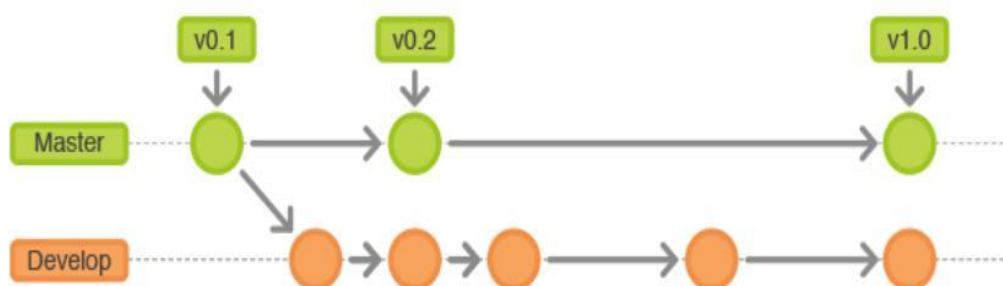
SVN 一样，集中式工作流以中央仓库作为项目所有修改的单点实体。所有修改都提交到 **Master 这个分支上**。

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。



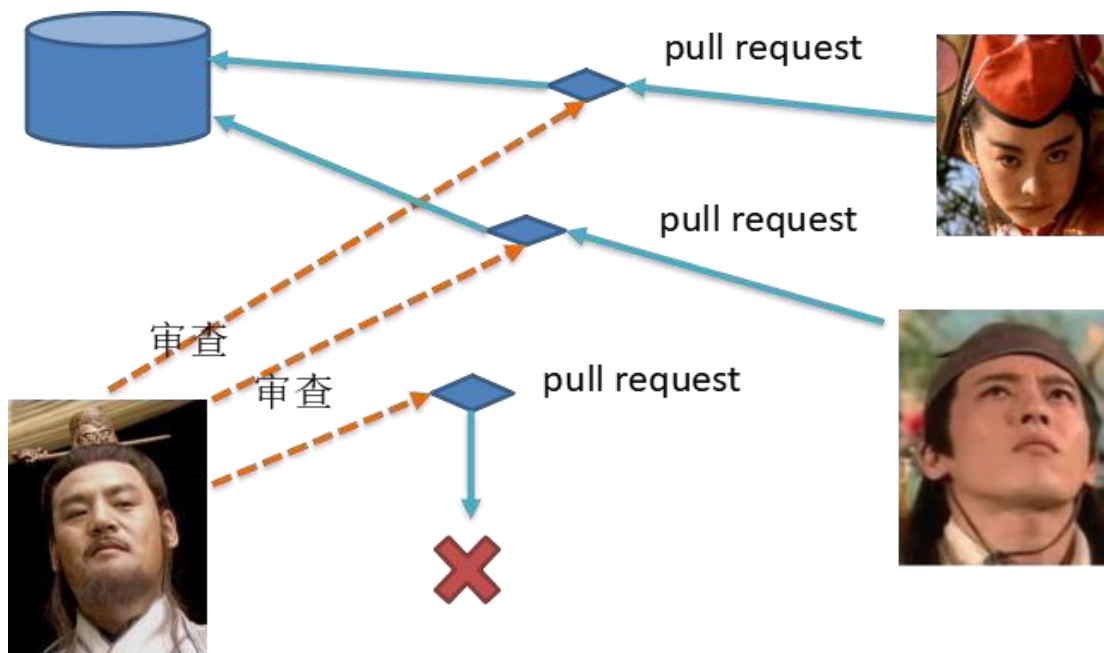
8.2.2 GitFlow 工作流(最金典,用得最多)

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**

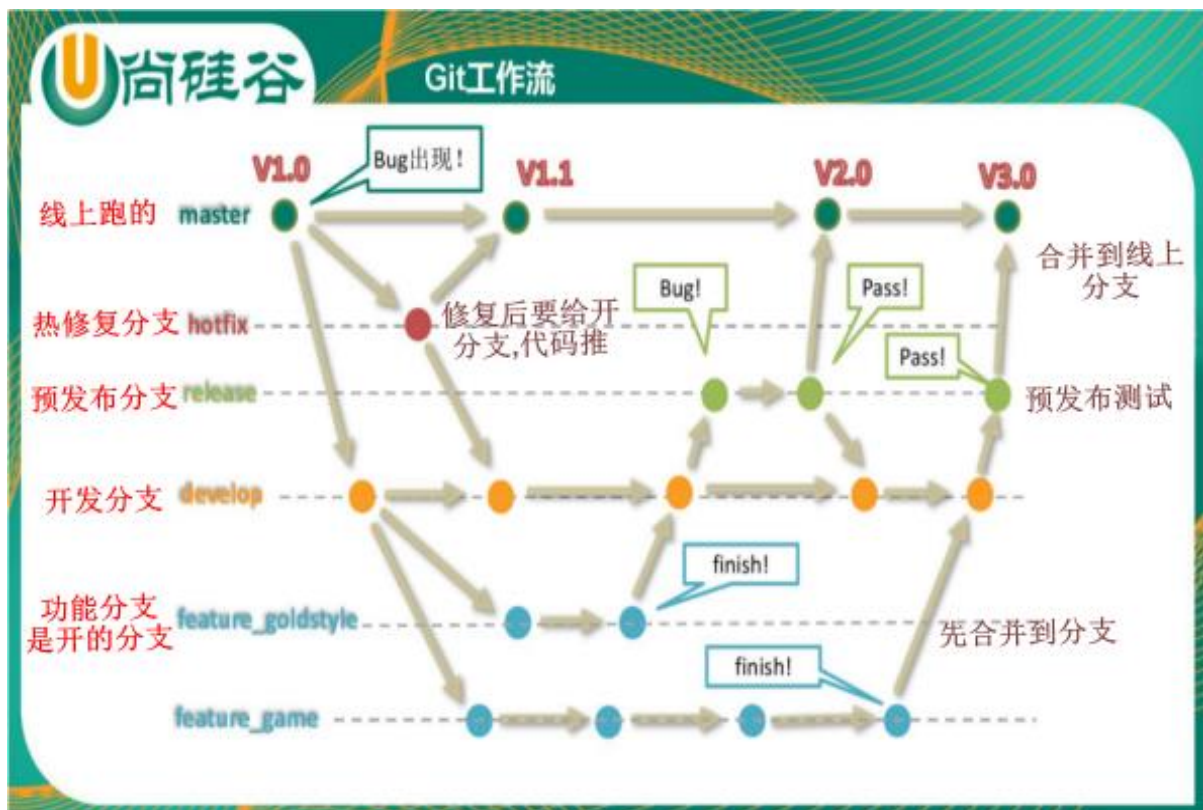
较大的版本上线前，会从开发分支中分出准生产分支，进行最后阶段的集成测试。该版本上线后，会合并到主干分支。生产环境运行一段阶段较稳定后可以视情况删除。

功能分支 **feature**

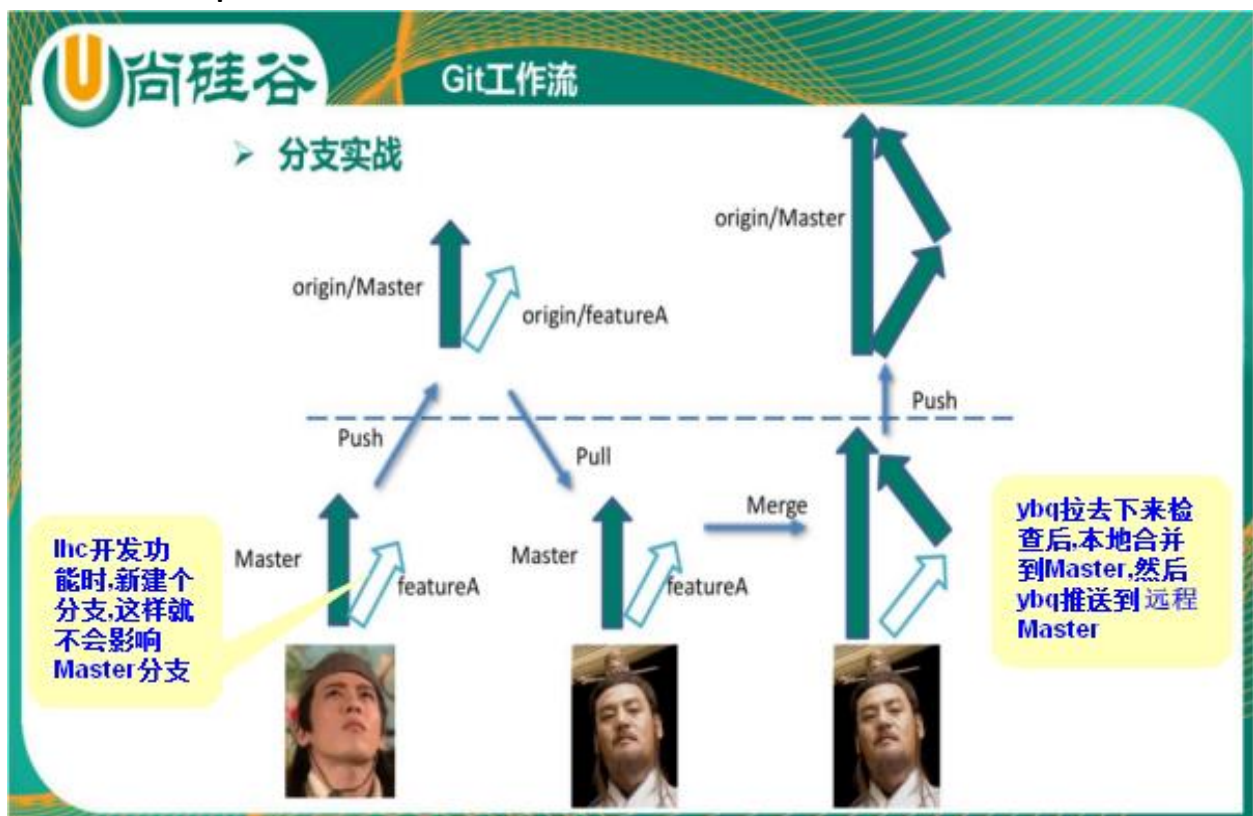
为了不影响较短周期的开发工作，一般把中长期开发模块，会从开发分支中独立出来。开发完成后会合并到开发分支。

8.3.2 GitFlow workflow 举例 **p54**

下面是 **GitFlow** workflow 举例图:

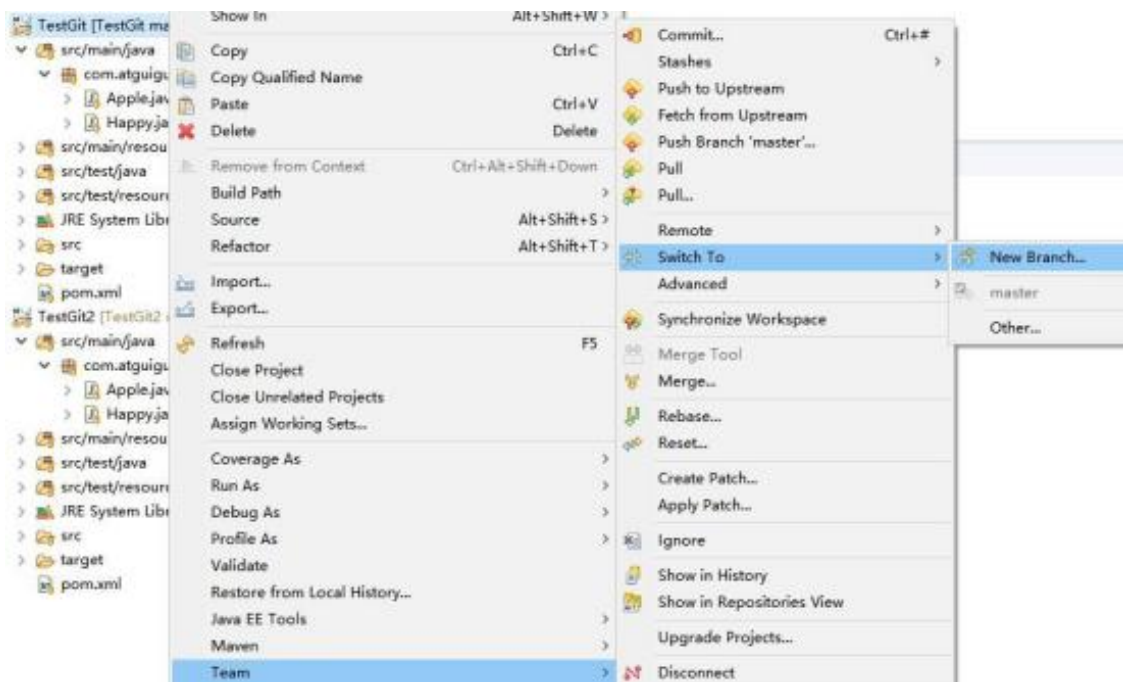


8.3.3 分支实战 p55

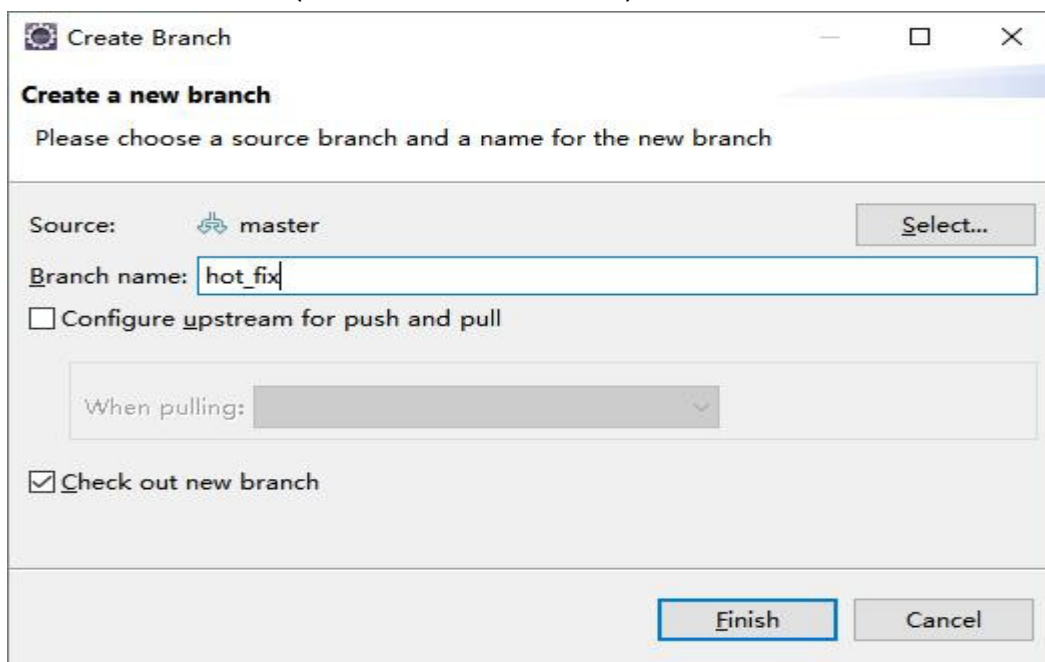


8.3.4 具体操作 p56

- 1 创建分支: 右键>Team>Switch To >New Branch



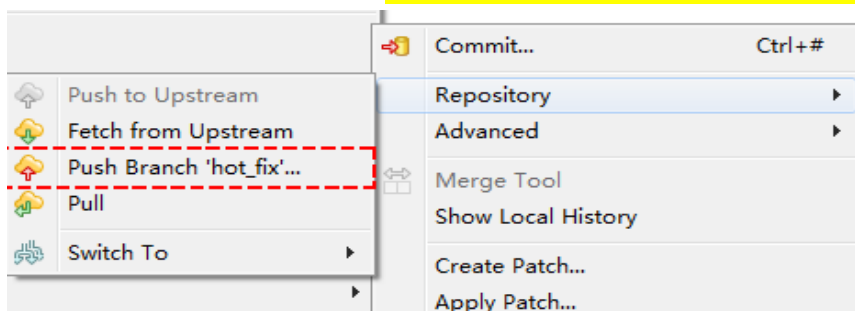
2 给分支命名>Finish (创建后自动切换到分支)



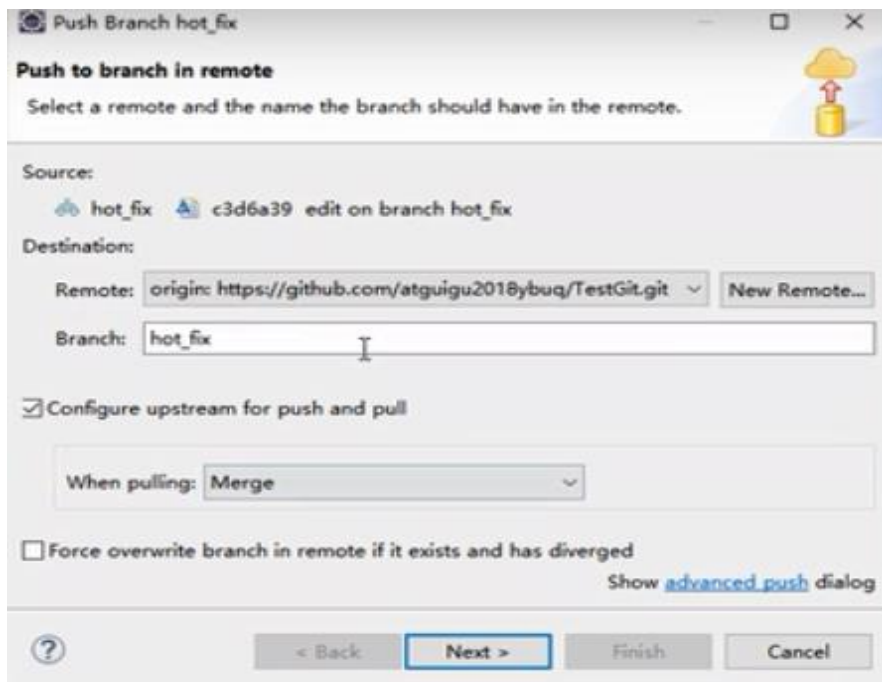
3 修改分支 hot_fix 的 happy.java 的内容(要避免冲突), 提交到本地库

4 远程推送到远程库

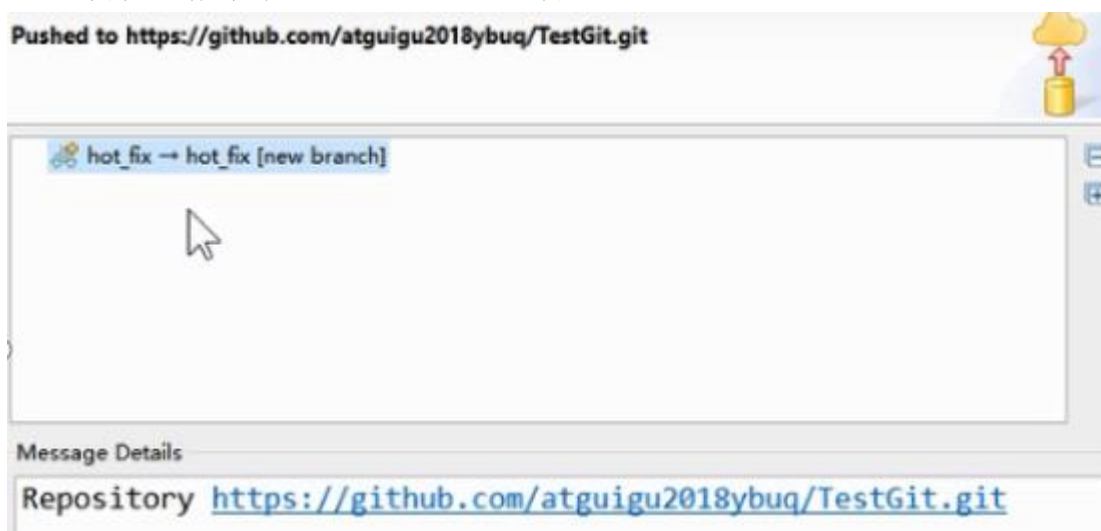
选择文件 happy.java >> 右键 >> Repository >> Push Branch 'hot_fix'



4.2 然后下一步>>然后(没截图)push



4.3 等待一会儿, 弹出下面对话框说明成功了

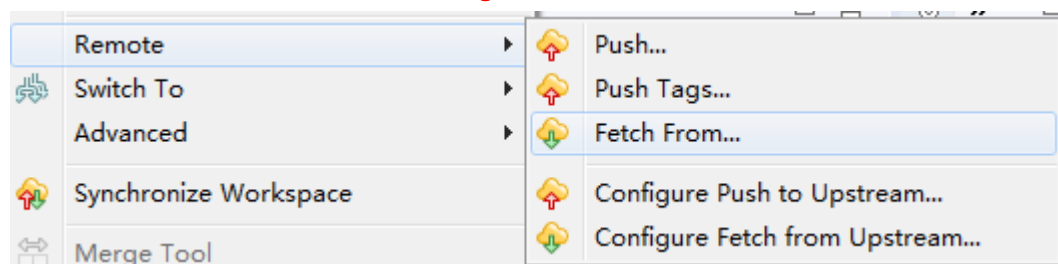


o5 ybq 拉取操作不需要选择分支:(这里有 TestGit2 模拟岳不群)

项目右键>Team>Pull

补充:/*上面操作如果说没有更新(上面没问题从这到>o6 之间不用看)

项目右键>Team>Remote>Fetch Tags



☐ Configured remote repository:

origin:

☒ Custom URI:

Location

URI:

Host:

Repository path:

Fetch Ref Specifications

Select refs to fetch.



Add create/update specification

Source ref:

Destination ref:

Add predefined specification

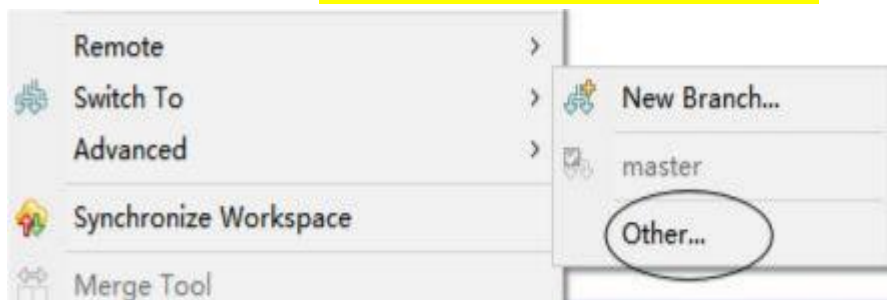
Specifications for fetch

Source Ref	Destination Ref	Force Update	Remove
refs/heads/*	refs/remotes/choose_re...	<input type="checkbox"/>	<input type="button" value="🗑"/>

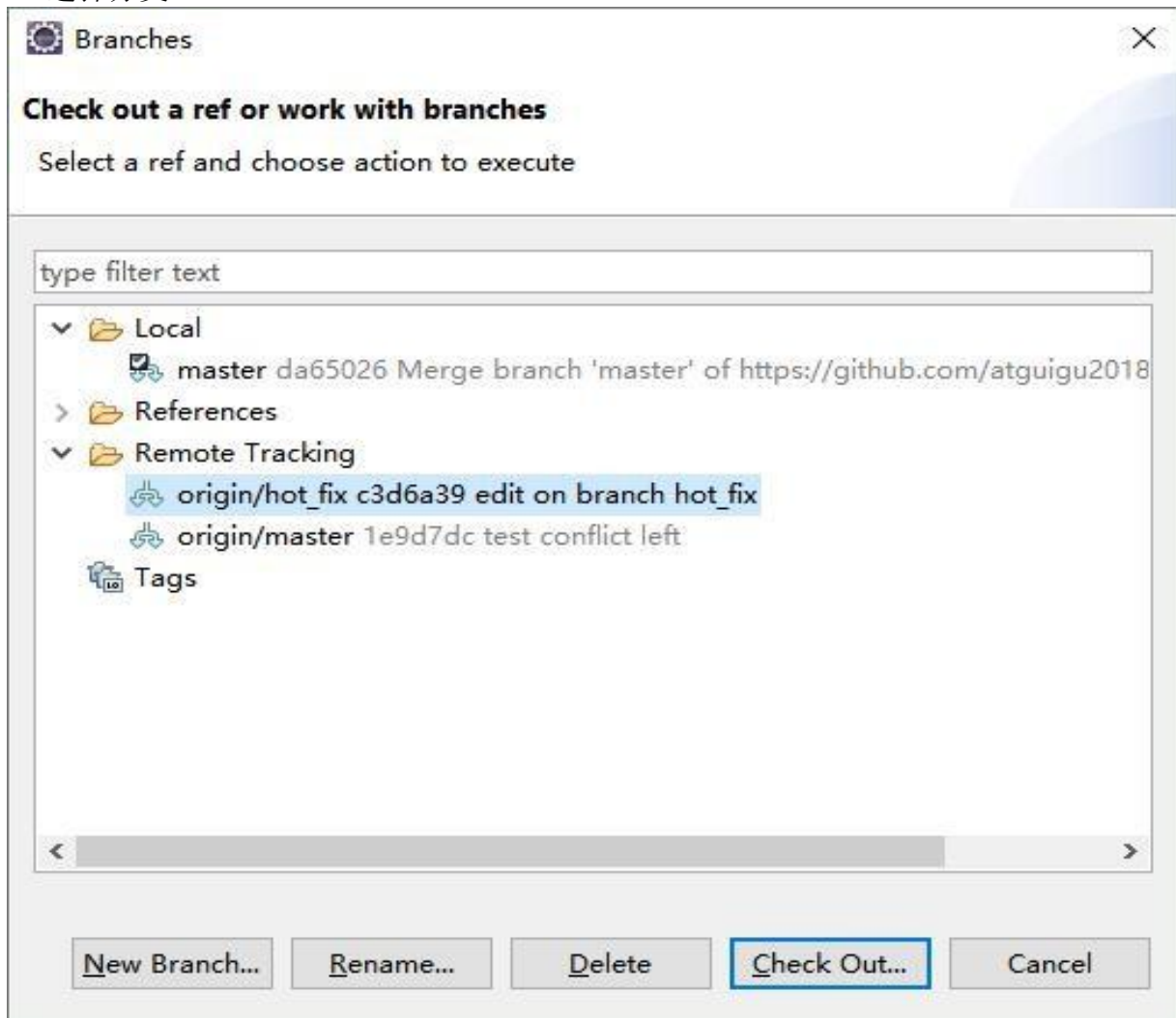
Annotated tags fetching strategy

- ☒ Automatically follow tags if we fetch the thing they point at
- ☐ Always fetch tags, even if we do not have the thing it points at
- ☐ Never fetch tags, even if we have the thing it points at

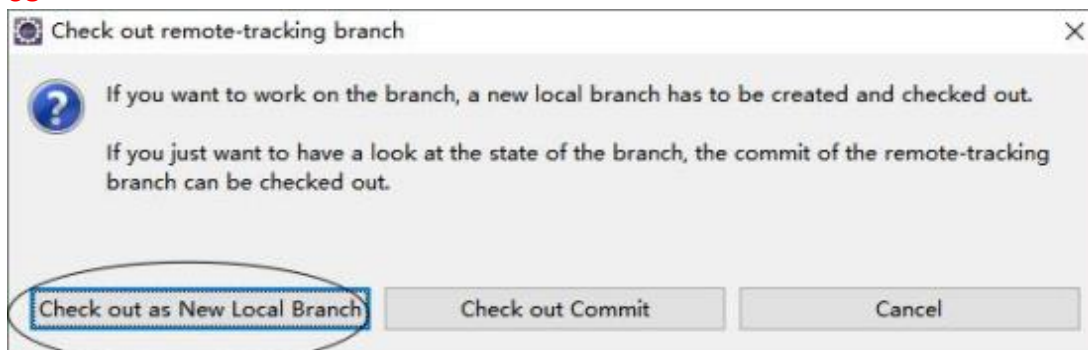
o6 切换分支审查代码: 项目右键>Team>Switch To > Other



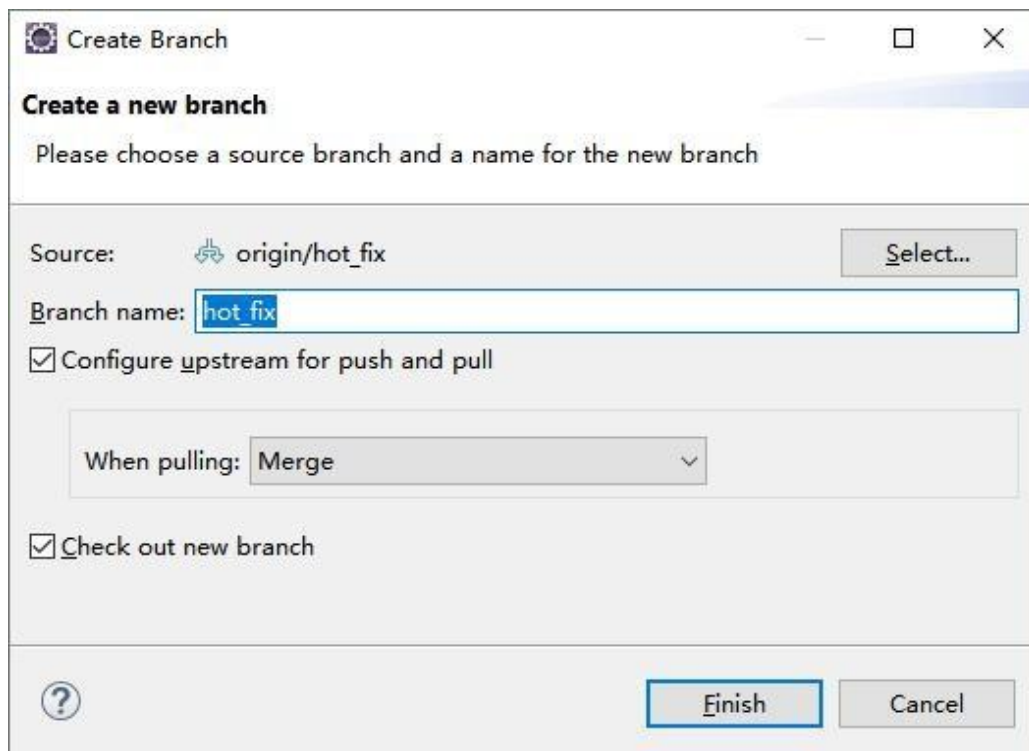
o7 选择分支



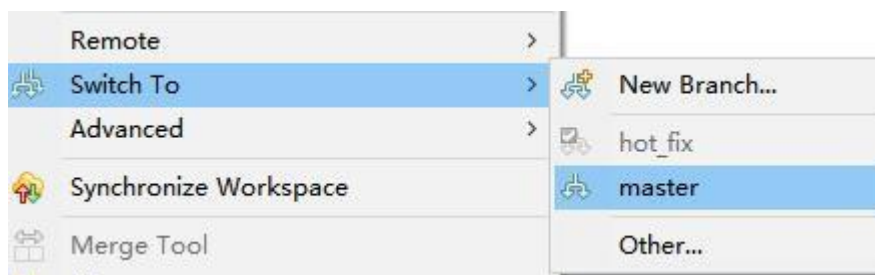
o8



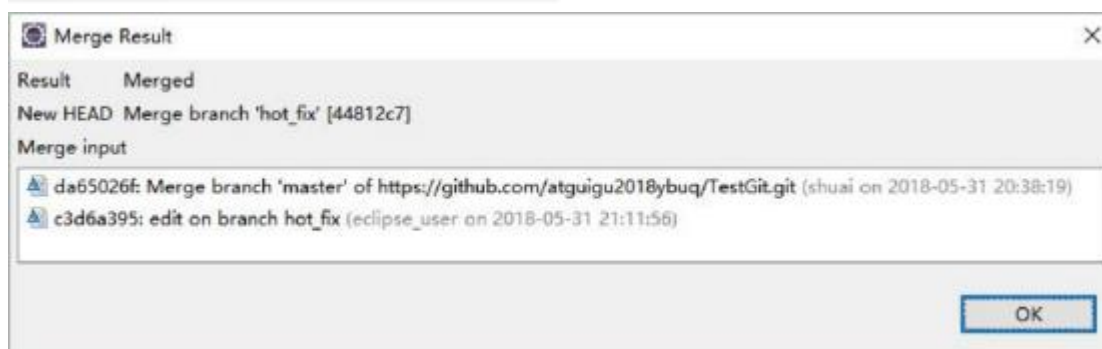
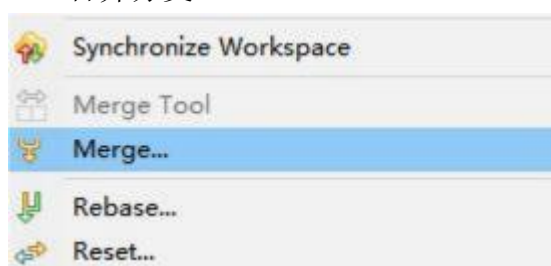
o9 创建新分支,选择 **Check out new branch** 为检出远程新分支



o10 切换回 master



o11 合并分支



合并结果



合并后直接提交到远程库(本地库已经提交)

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 cronie
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 ee 是企业版本 ce 是社区版本

```
sudo rpm -ivh /opt/gitlab-ce-10.8.2-ce.0.el7.x86_64.rpm
sudo yum install -y curl policycoreutils-python openssh-server cronie
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 60

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 61

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 firewallld stop**

微信号：creathinFeng