# 01【熟悉】git简介

## 一，概述

Git是什么？

Git是目前世界上最先进的分布式版本控制系统（没有之一）。

Git有什么特点？简单来说就是：高端大气上档次！

哪些GIT网站？

<https://github.com/> 全球最大的开源项目网站。

<https://gitee.com/> 中国最大的开源项目网站。

## 二，git的诞生

        很多人都知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。

        Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

        事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

        你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。

        不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。

        安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

        Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：

        Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。

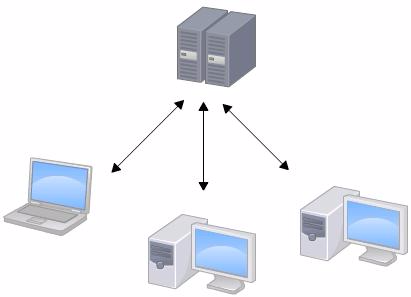
        Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

        历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了。

## 三，集中式VS分布式

        Linus一直痛恨的CVS及SVN都是集中式的版本控制系统，而Git是分布式版本控制系统，集中式和 分布式版本控制系统有什么区别呢？

        先说集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。

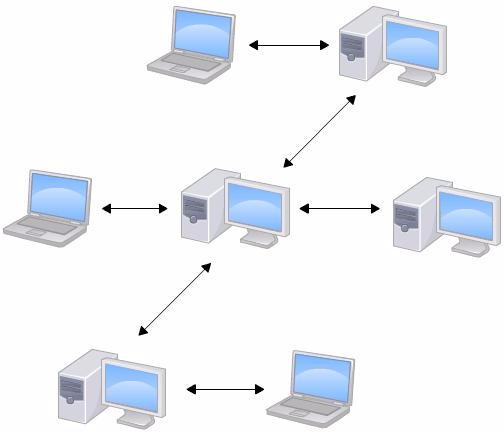


        集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊。

        那分布式版本控制系统与集中式版本控制系统有何不同呢？首先，分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

        和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。

        在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。



        当然，Git的优势不单是不必联网这么简单，后面我们还会看到Git极其强大的分支管理，把SVN等远远抛在了后面。

        CVS作为最早的开源而且免费的集中式版本控制系统，直到现在还有不少人在用。由于CVS自身设计的问题，会造成提交文件不完整，版本库莫名其妙损坏的情况。同样是开源而且免费的SVN修正了CVS的一些稳定性问题，是目前用得最多的集中式版本库控制系统。

        除了免费的外，还有收费的集中式版本控制系统，比如IBM的ClearCase（以前是Rational公司的，被IBM收购了），特点是安装比Windows还大，运行比蜗牛还慢，能用ClearCase的一般是世界500强，他们有个共同的特点是财大气粗，或者人傻钱多。

        微软自己也有一个集中式版本控制系统叫VSS，集成在Visual Studio中。由于其反人类的设计，连微软自己都不好意思用了。

        分布式版本控制系统除了Git以及促使Git诞生的BitKeeper外，还有类似Git的Mercurial和Bazaar等。这些分布式版本控制系统各有特点，但最快、最简单也最流行的依然是Git！

# 02【掌握】git的安装

## 一，下载

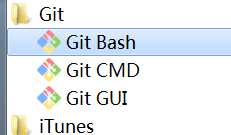
1，<https://git-scm.com/downloads>



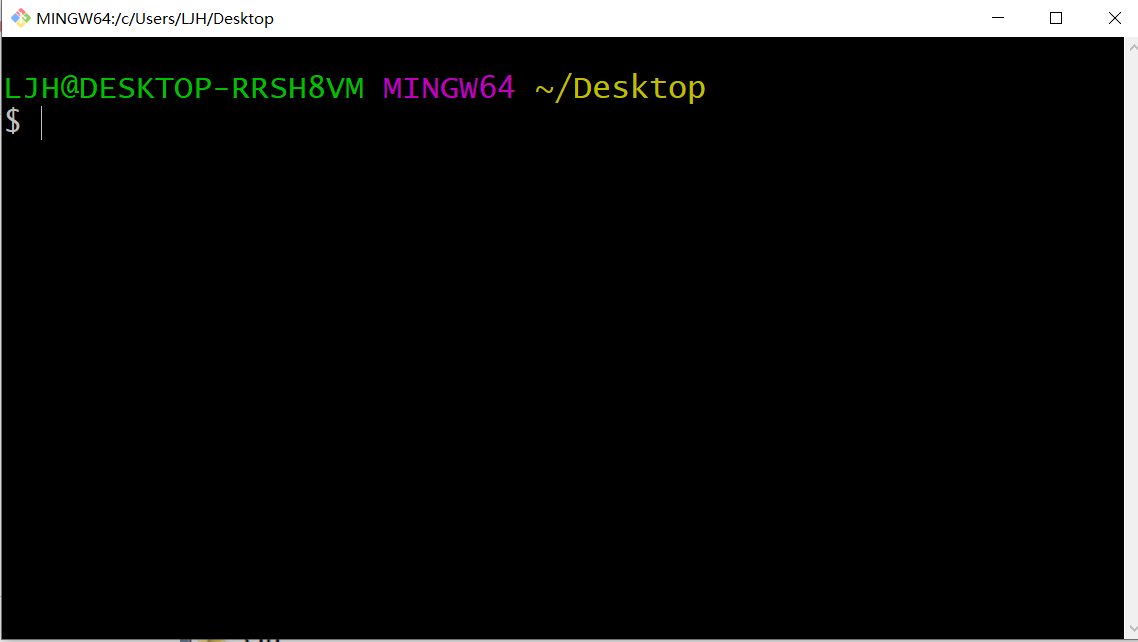


## 二，设置全局身份

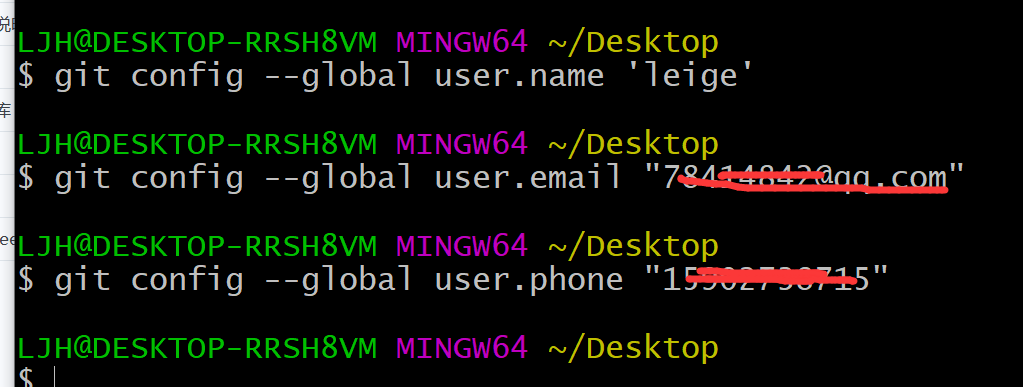
1，打开git Bash



如果出现以下界面说明安装成功了（除非脸黑，否则是没有任何问题的）



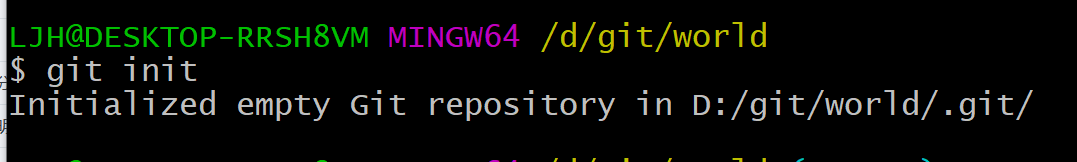
因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。



## 三，创建版本库

什么是版本库呢？版本库又名仓库，英文名repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

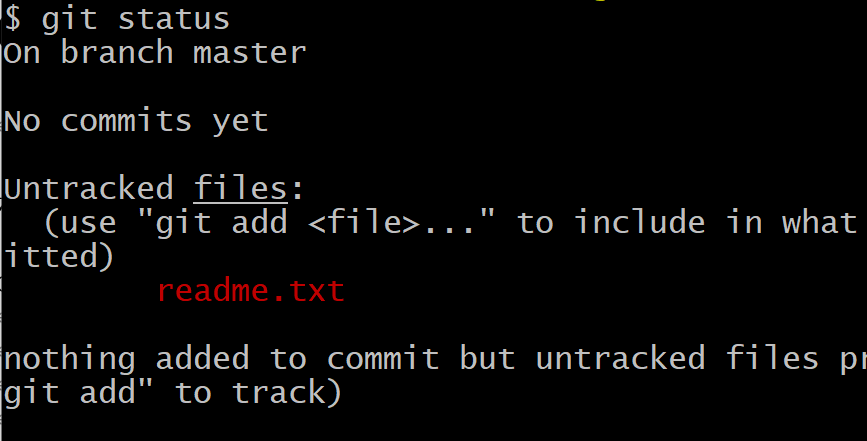
所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：



# 03【掌握】文件管理-添加文件

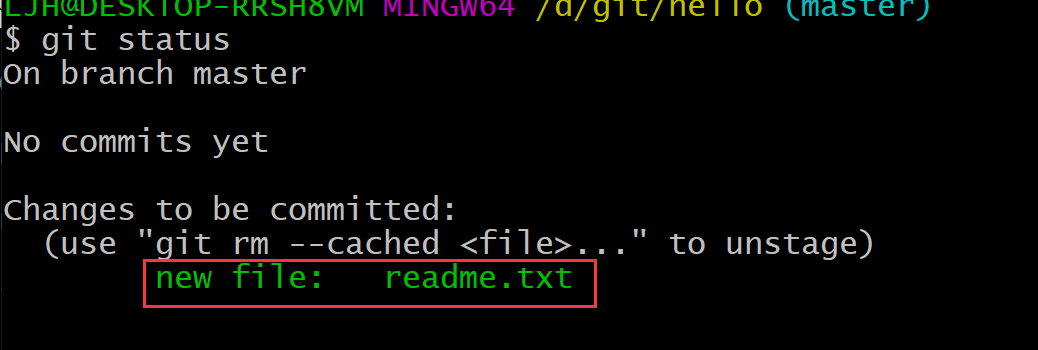
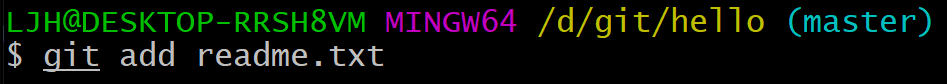
## 相关命令

### git status 查看当前git目录的版本状态



|-代表一个新的文件

### git add 文件名1 文件名2 把当前文件加入到暂存区



|--代表现在暂存区有一个新的文件没有提交 提交之后才会进入版本库

### git commit -m 提交记录

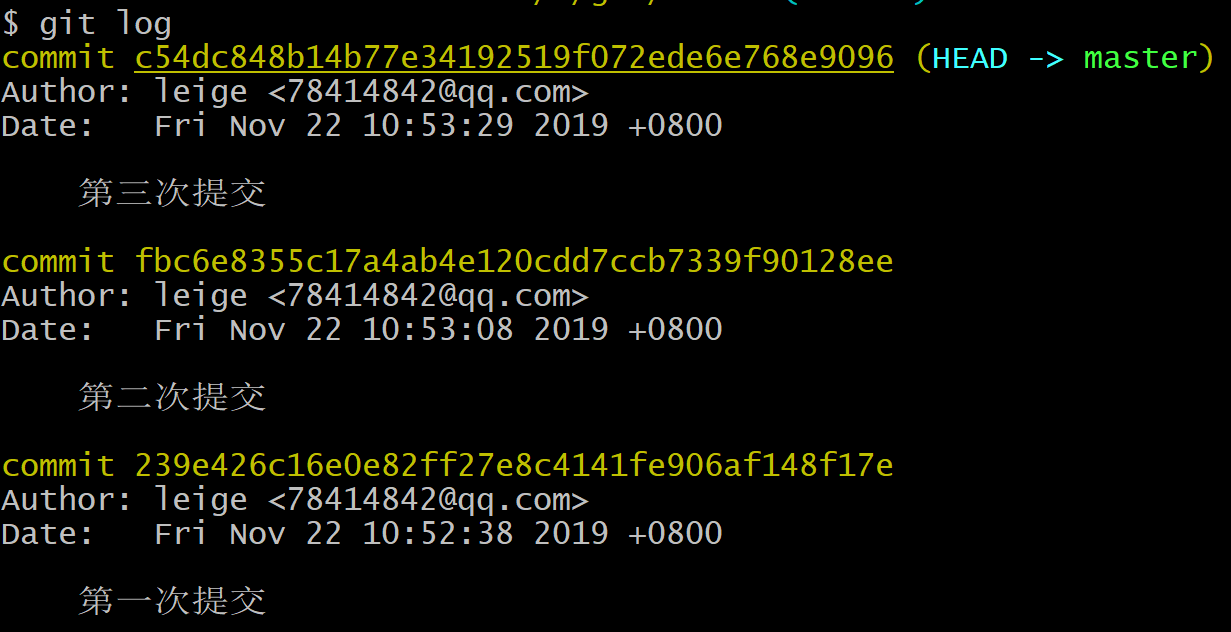
把暂存区里面的数据提交到当前所在的分支(master)

# 04【掌握】文件管理-版本回退

## 相关命令

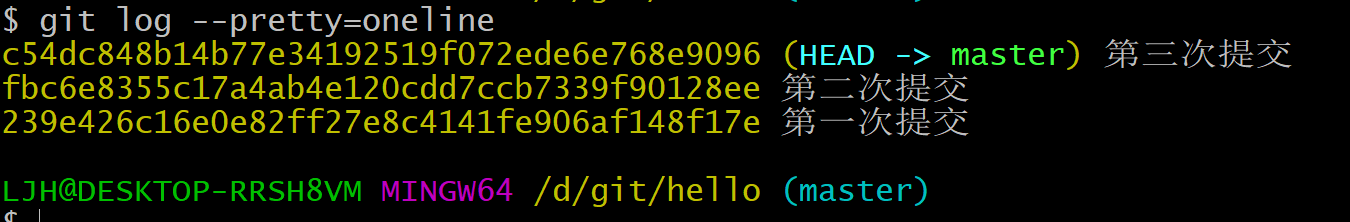
### git log

查看当前分支的版本提交记录



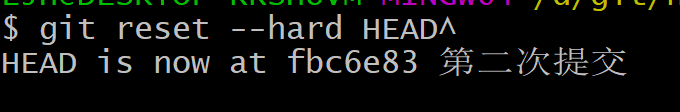
### git log --pretty=oneline

个行显示



### git reset --hard HEAD^

回退到上一个版本

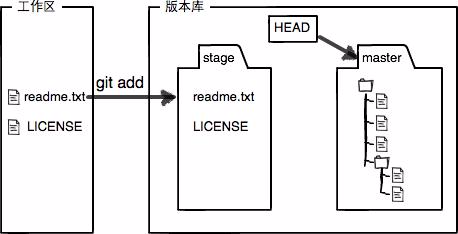


### git reset --hard cb5f63

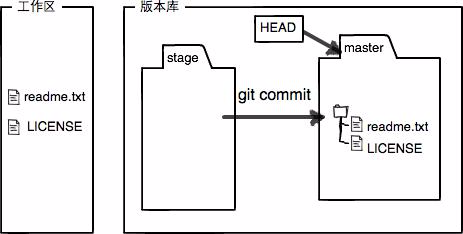
回退到指定的版本号



# 05【掌握】工作区和暂存区



使用commit



# 06【掌握】文件管理-文件修改

## 思路

在工作区创建文件---add 到暂存区---commit 到分支

## 相关命令

### git diff head -- 文件名

查看当前文件和版本库工区别

## 特点

Git 对文件修改跟踪的修改的记录 。不是整个文件

# 07【掌握】文件管理-撤销修改

## 情况1：数据只存在工作区

### git checkout -- readme.txt 丢弃工作区修改

## 情况2：数据使用add 添加到暂存区

思路：先从暂存区里面移除相应的文件



再使用情况1的方式丢弃

## 情况3：数据使用add 添加到暂存区并commit到版本库

使用04【掌握】文件管理-版本回退的方式

# 08【掌握】文件管理-删除文件

## 相关命令

先删除

git rm -rf 文件名

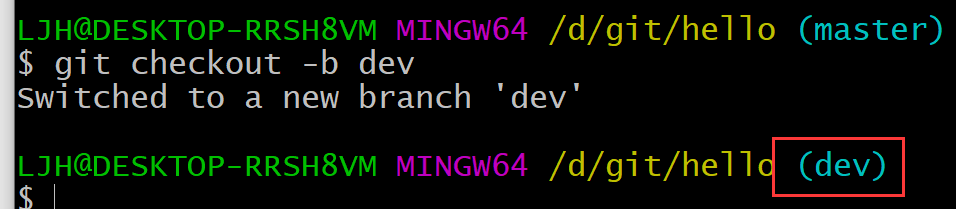
再提交

Git commit -m 备注

# 09【掌握】分支管理-创建与合并分支

## 相关命令

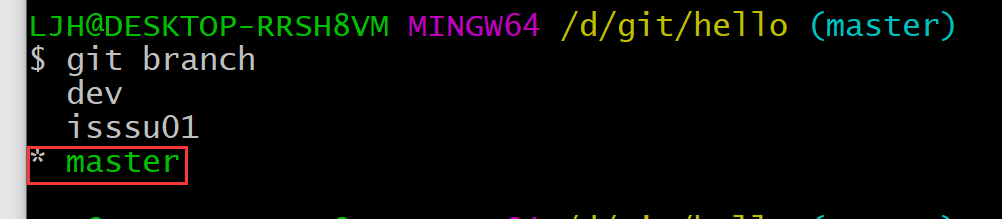
### git checkout -b dev 创建dev分支并切换到dev分支



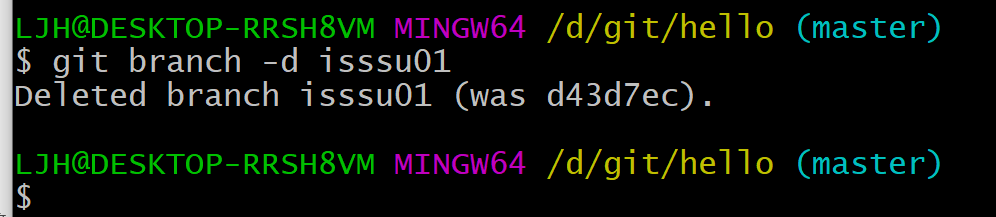
### git branch dev 创建一个叫dev的分支，不切换

### git checkout dev 切换到一个叫dev的分支

### git branch 查看所有分支

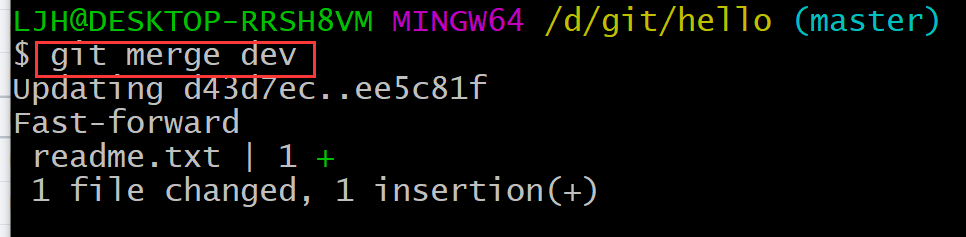


### git branch -d issue001 删除issue001分支

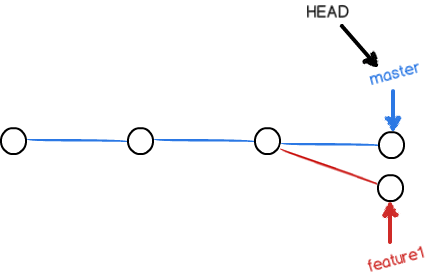


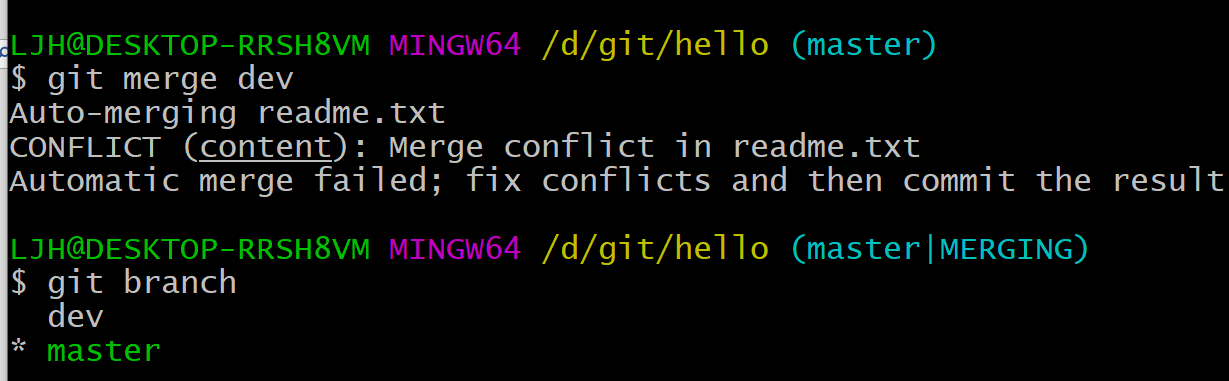
### git merge dev 合并分支

如把dev合并到master那么要在master分支上执行合并的命令



## 解决合并的冲突问题





因为两个分支的起点不一样了，所以要手动解决合并的冲突问题



# 10【熟悉】分支管理-分支策略说明

## 分支策略

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

下面我们实战一下--no-ff方式的git merge：

首先，仍然创建并切换dev分支：

$ git checkout -b dev

Switched to a new branch 'dev'

修改readme.txt文件，并提交一个新的commit：

$ git commit -m "添加dev上的修改内容"

[dev 8bee24d] 添加dev上的修改内容

1 file changed, 2 insertions(+)

现在，我们切换回master：

$ git checkout master

Switched to branch 'master'

准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

$ git merge --no-ff -m "merge with no-ff" dev

Merge made by the 'recursive' strategy.

readme.txt | 1 +

1 file changed, 1 insertion(+)

因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

合并后，我们用git log看看分支历史：

$ git log --graph --pretty=oneline --abbrev-commit

\* e1e9c68 (HEAD -> master) merge with no-ff

|\

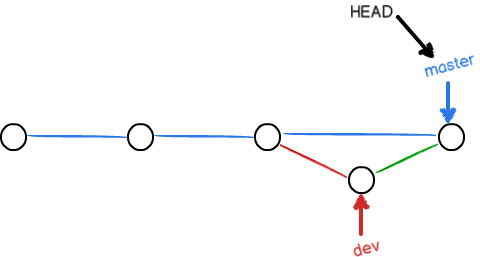
| \* f52c633 (dev) add merge

|/

\* cf810e4 conflict fixed

...

可以看到，不使用Fast forward模式，merge后就像这样：



分支策略

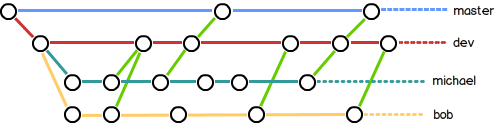
在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



## 小结

Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

# 11【了解】分支管理-Bug分支

## 使用场景

现有主分支master

从主分支切了一个dev分支开发一个新的NB的功能。。。但是还只写了一半

领导说线上和master项目有重大bug请速度解决。你看了一，这个bug5分钟可解决，

但是我还在开发新功能，但没有提交

需要把当前dev分支的工作现场储藏起来

切到master 切一个issue的bug分支出来解决bug

解决完之后回到master再合并issue分支。再删除issue的分支

## 是关命令

### git stash 储藏当前分支的工作区

储藏完成这后再执行git status发现工作区tree clean

### git stash list 显示当前分支储藏的工作区的集合

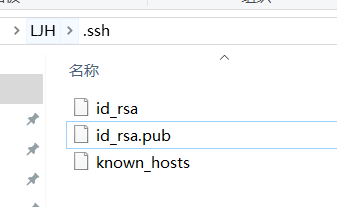
### git stash pop取出当前分支储藏的工作区栈顶的第一个储藏对象

# 12【掌握】使用GitHub远程仓库

## 1，远程库存配置

### 第1步：创建SSH Key。

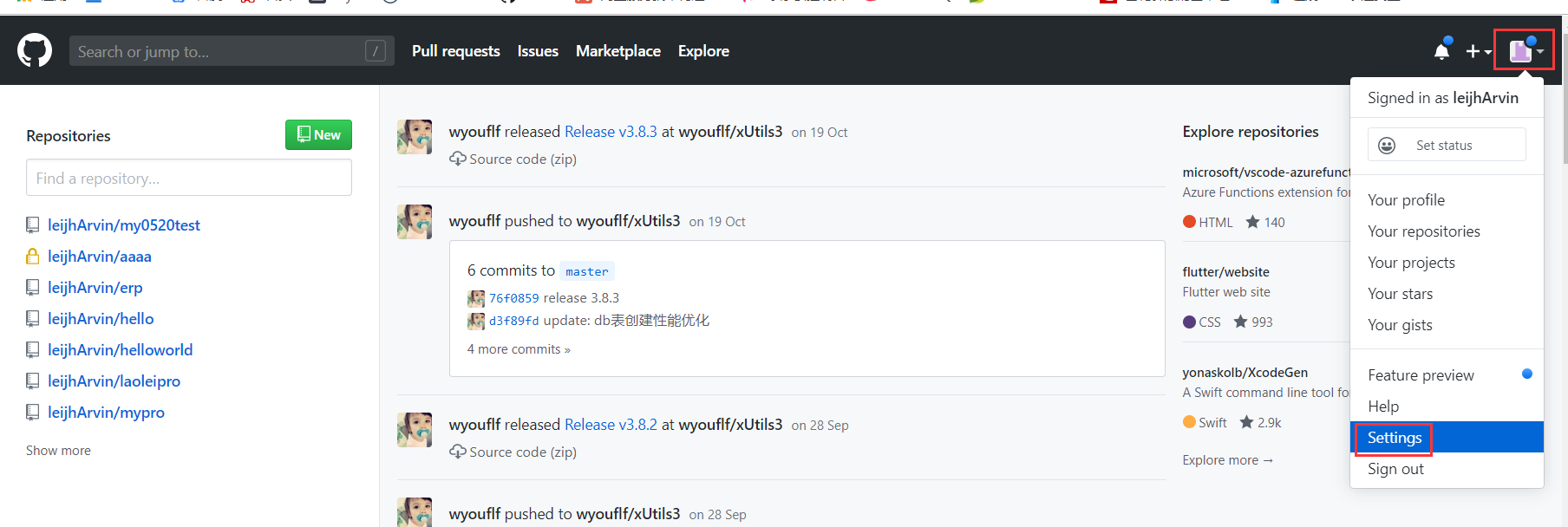
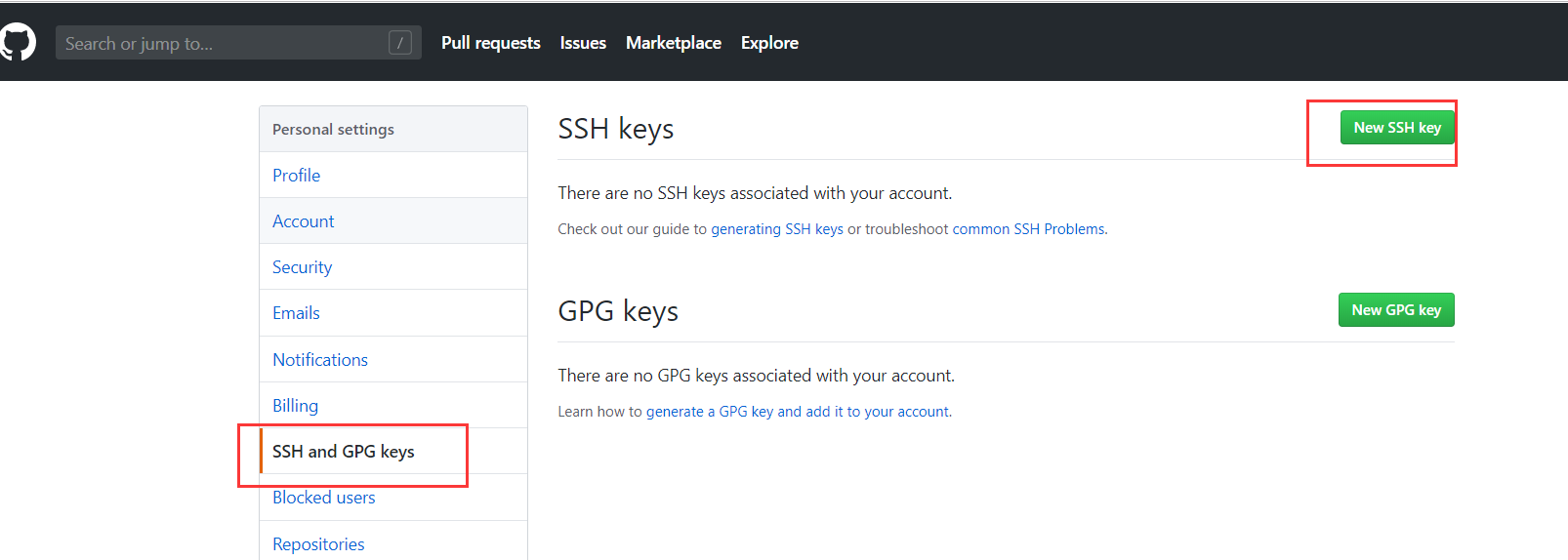


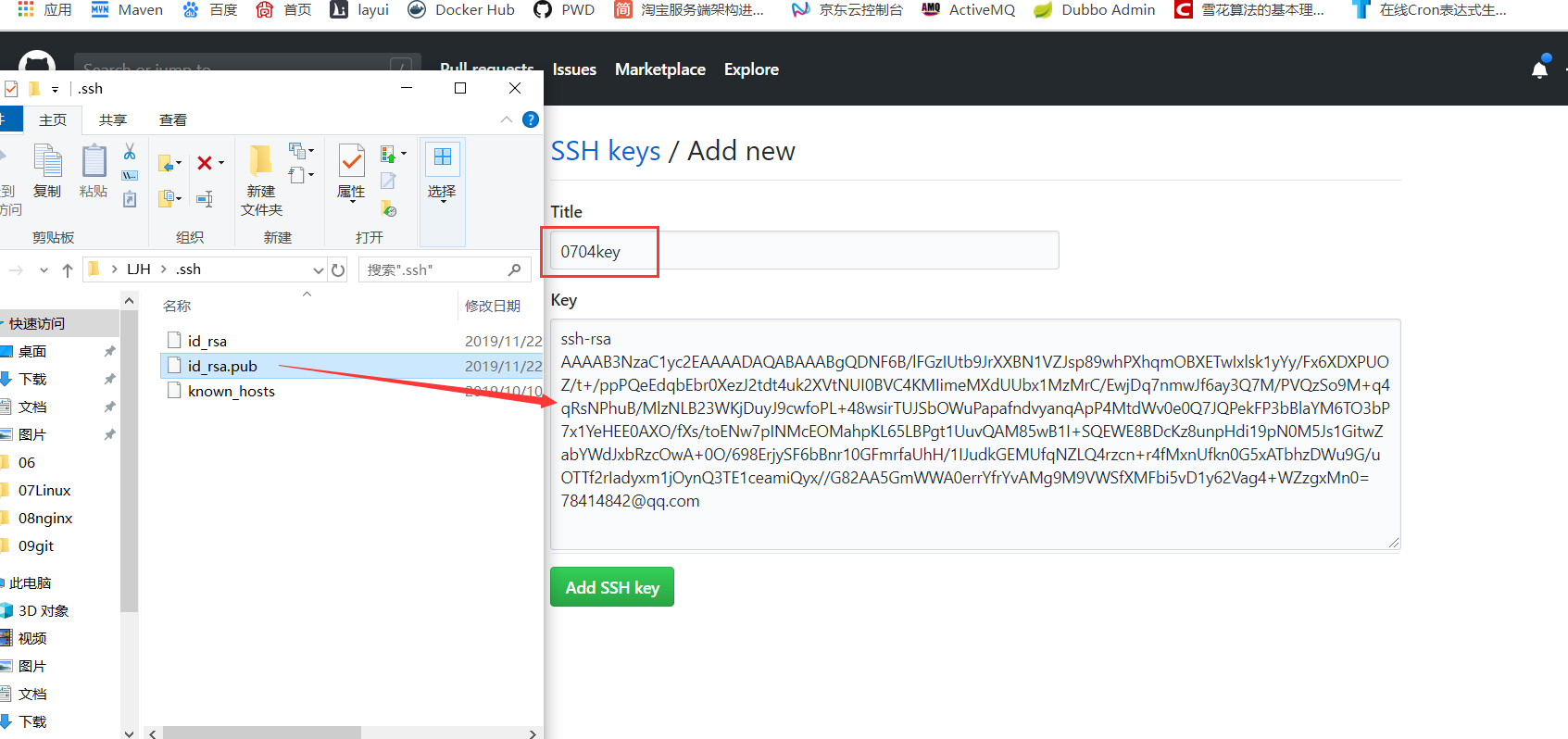


Id\_rsa 秘钥

Id\_rsa.pub 公钥

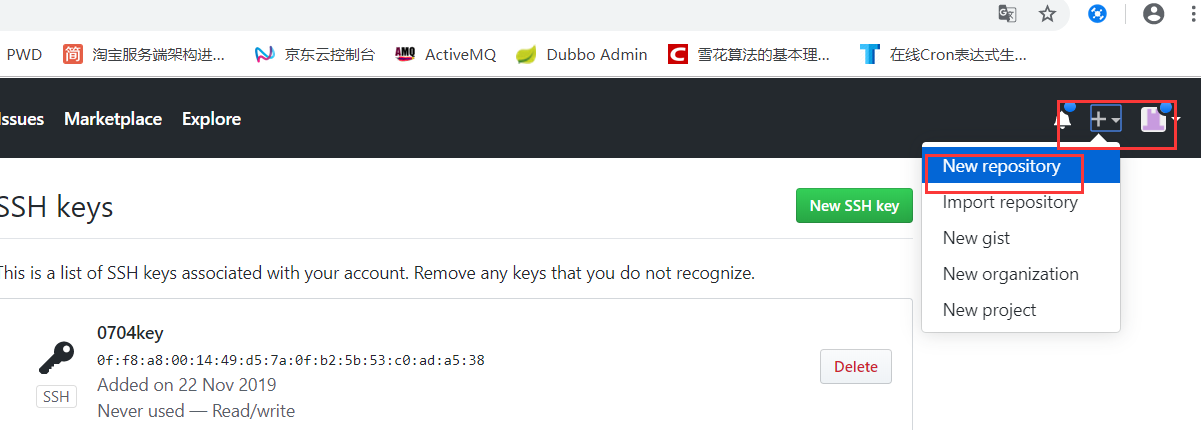
### 第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：

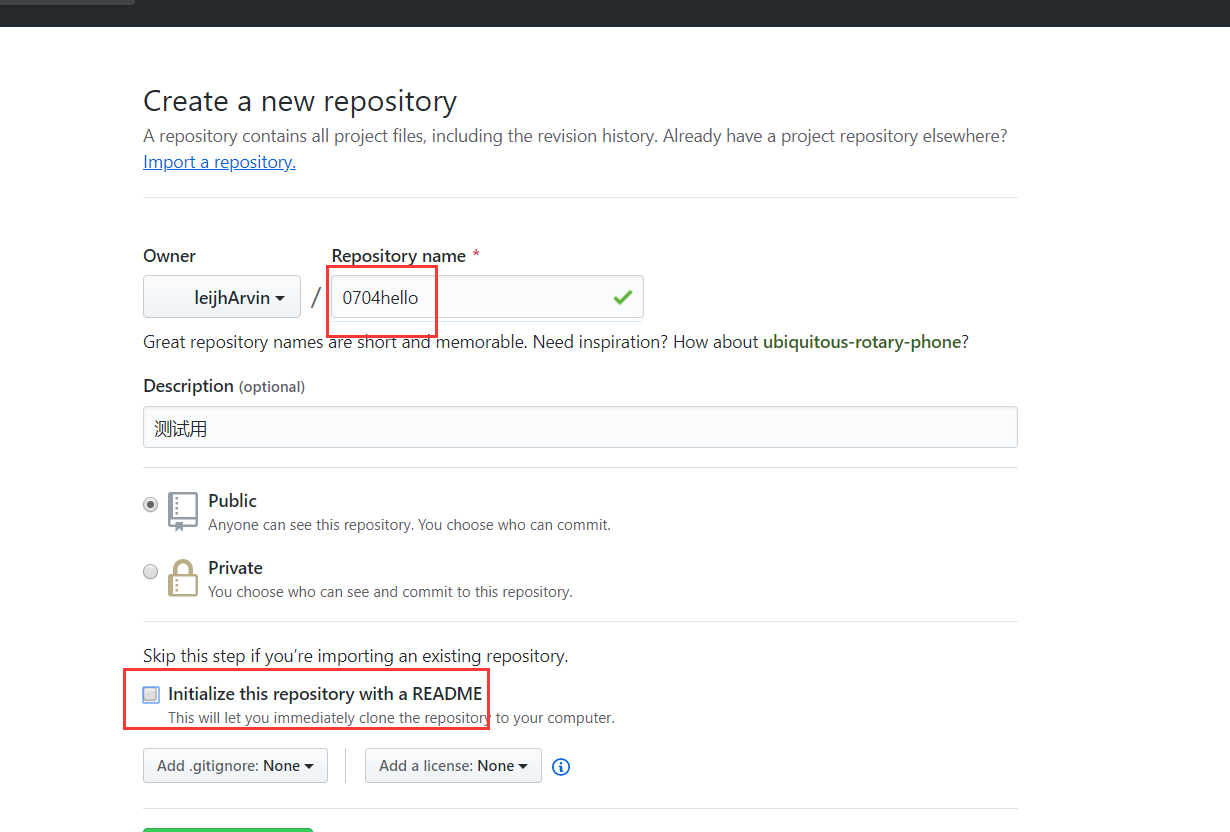
 



## 2，添加远程仓库

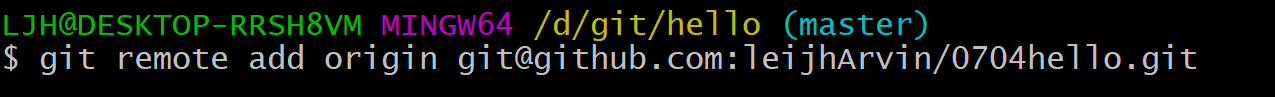
### 登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：





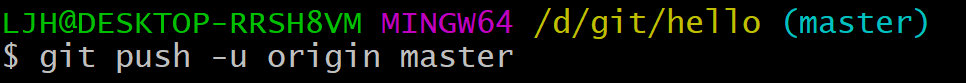
### 把本地仓库和远程仓库进行关联

git remote add origin git@github.com:leijhArvin/0704hello.git



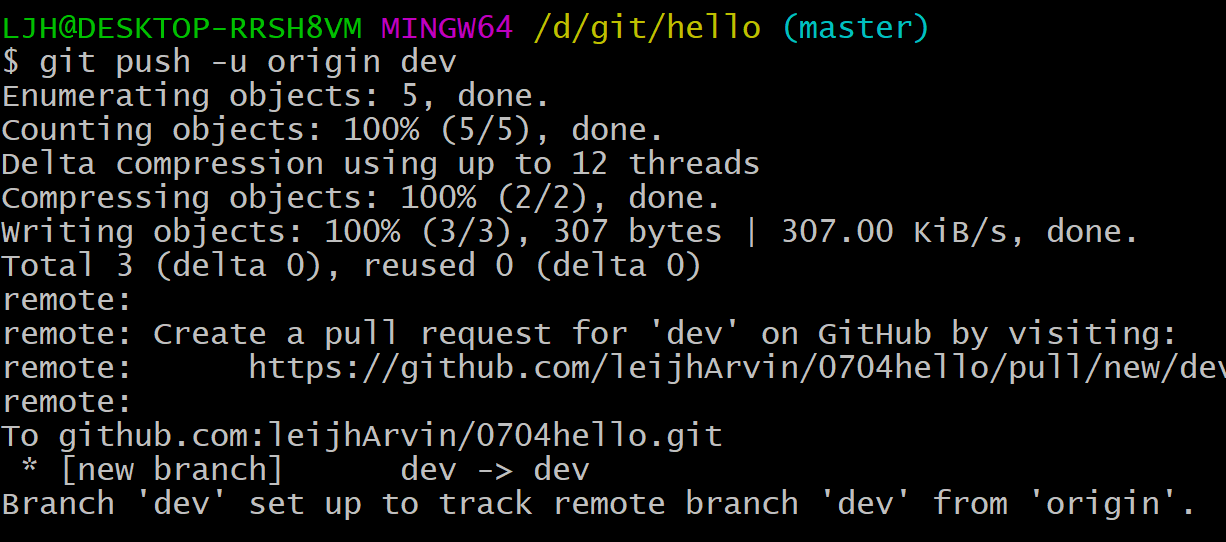
### 可以把本地库的master分支内容推送到远程库上：

git push -u origin master



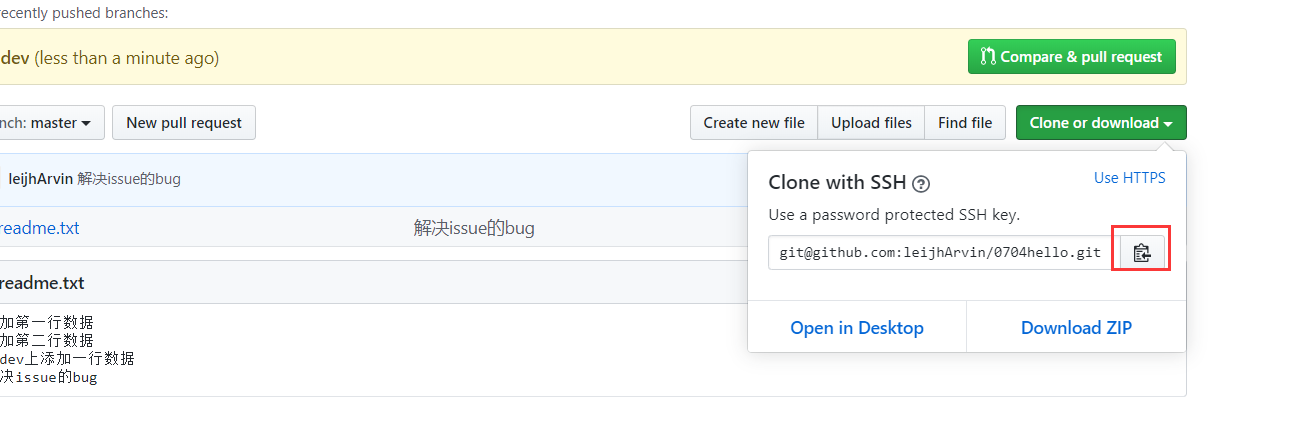
### 可以把本地库的dev分支内容推送到远程库上：

git push -u origin dev



## 3，从远程仓库克隆

### 找到地址

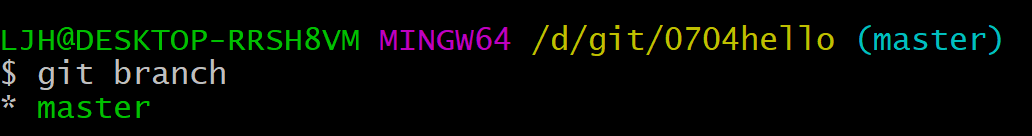


### 执行clone命令

git clone git@github.com:leijhArvin/0704hello.git

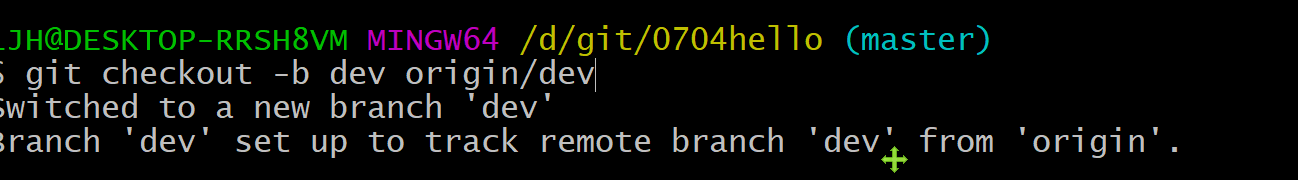
在clone时默认是把当前地址里面的所有分支全部clone下来，但是只有master显示

### 解决其它分支不显示的问题



是因为没有关联

git checkout -b dev origin/dev



# 13【掌握】分支管理-多人协作

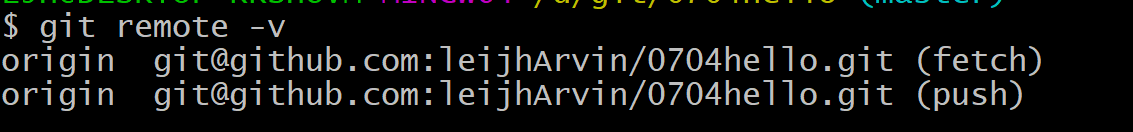
## 相关命令

git remote add origin [git@github.com:leijhArvin/0704hello.git](mailto:git@github.com:leijhArvin/0704hello.git) 关联

git push -u origin master 推送主分支

git clone Git地址 克隆远程库

git remote -v 查看远程库的信息



## 演示思路

创建项目目录

mkdir xiaoming

mkdir xiaohong

进入xiaoming

cd xiaoming

克隆git clone git@github.com:leijhArvin/0704hello.git

进入xiaohong

cd xiaohong

克隆git clone git@github.com:leijhArvin/0704hello.git

在小红里面修改并提交 并推送

Git add

Git commit

Git push -u -origin master

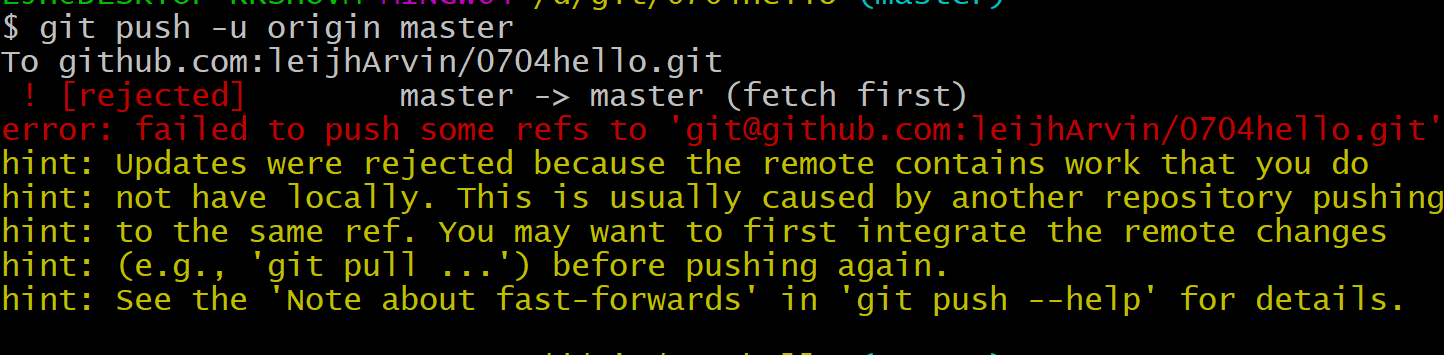
在小明里面修改关提交 ，会有冲突

Cd xiaoming

Git add

Git commit

Git push -u origin master



先拉取

Git pull

再解决冲突

再添加提交推送

## 作业

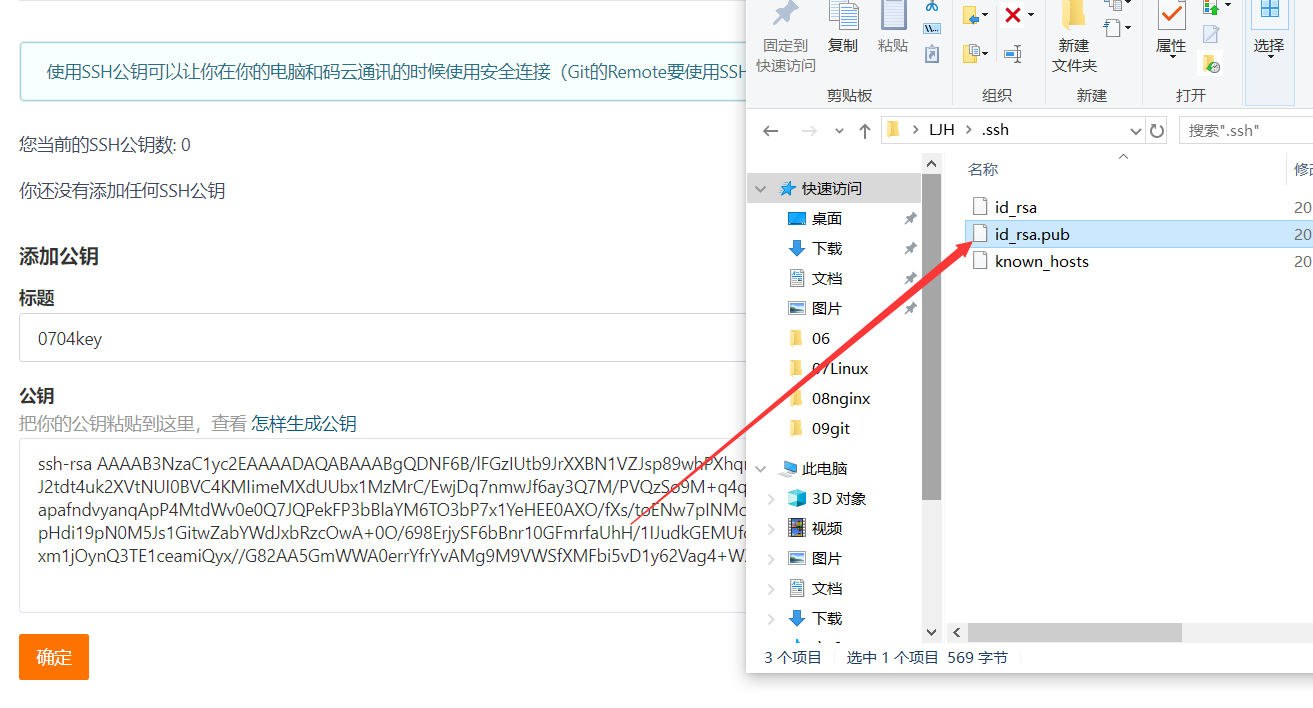
同桌之间相互推一下

# 14【掌握】使用码云

## 修改sshkey [上面已创建过]

## 在gitee上配置sshkey

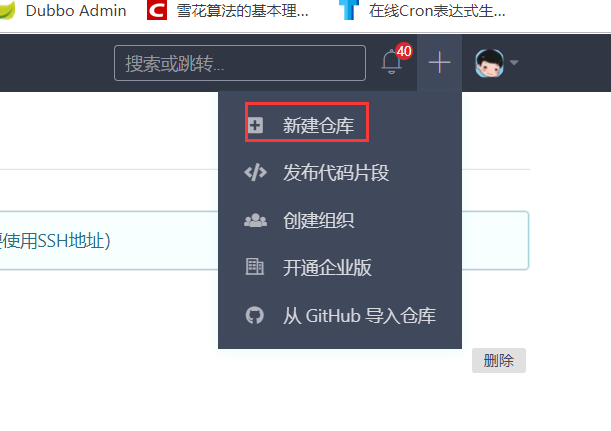






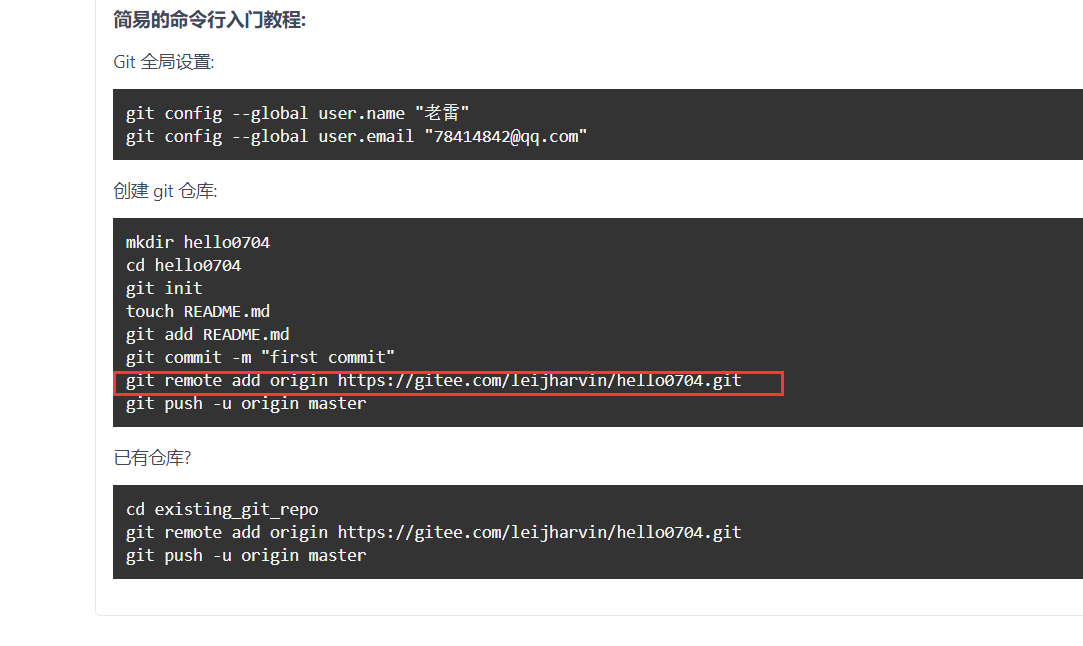
## 重新初始化本地仓库

## 在gitee上创建仓库





## 把本地仓库和gitee仓库关联

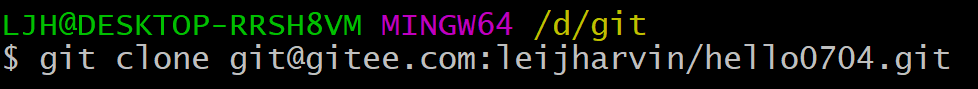




## 把本地仓库推送到gitee

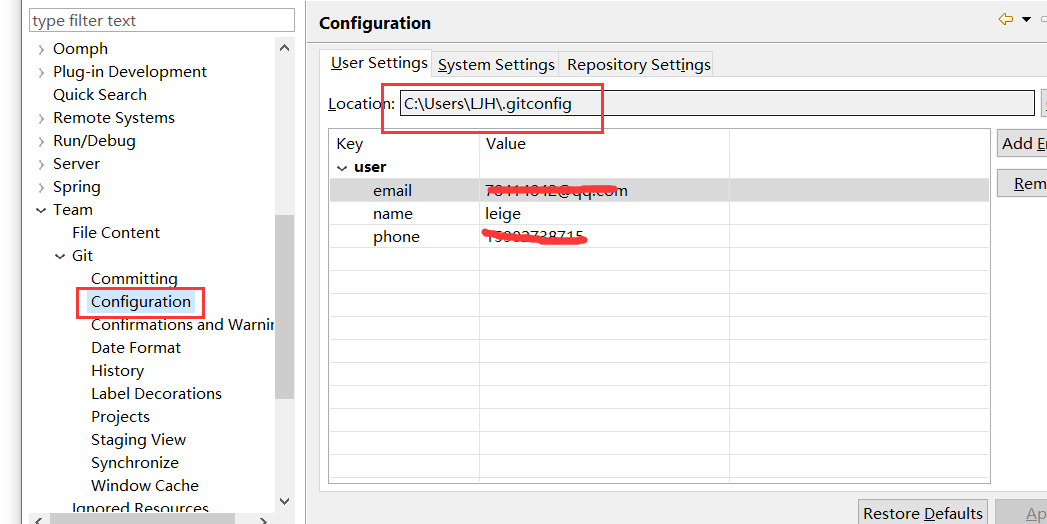
## 把远程仓库clone到本地





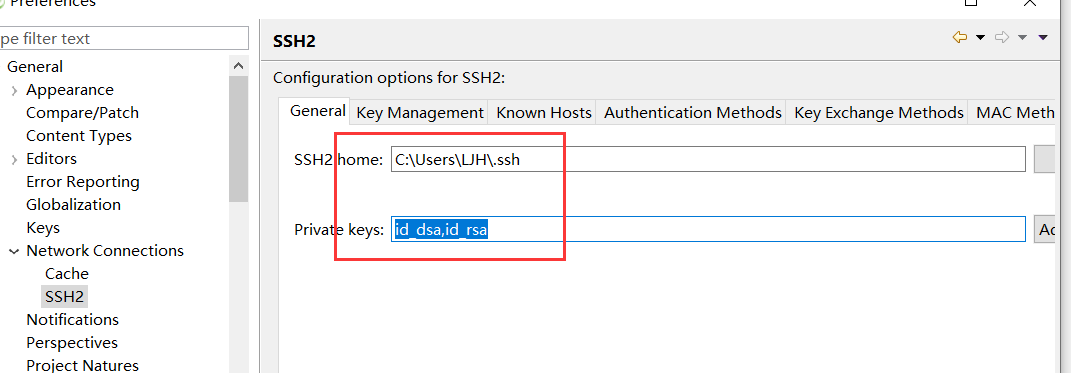
# 15【掌握】eclipse里面使用gitee

## 配置user[默认读取用户目录下的.gitconfig]

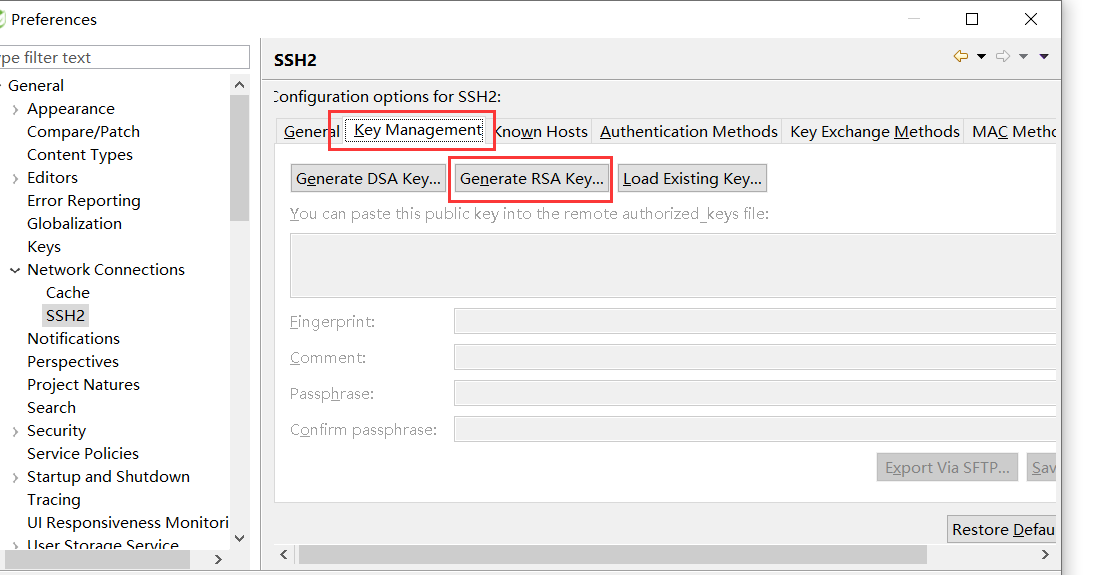


## 配置并生成公钥

默认已有



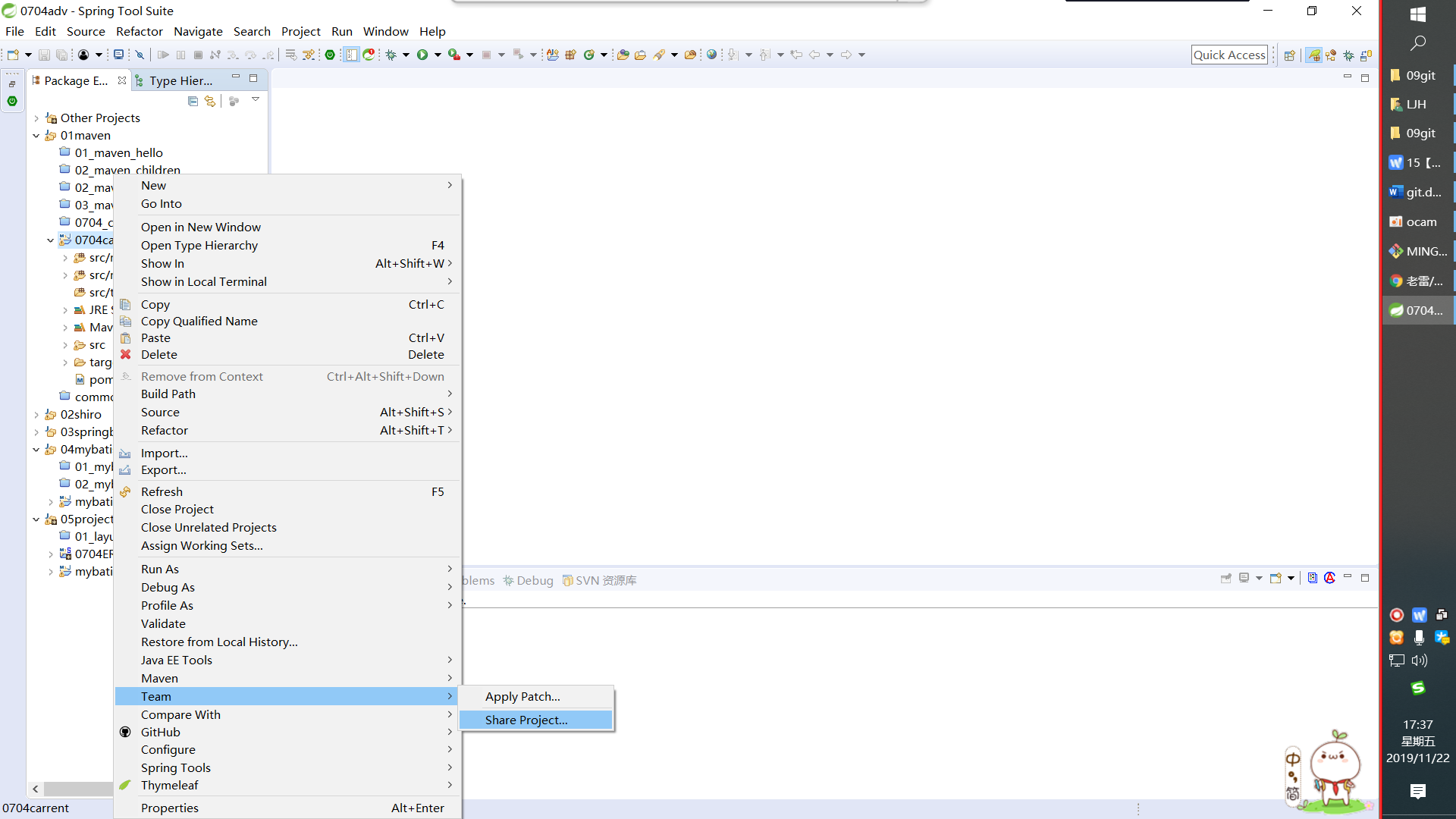
如果没有

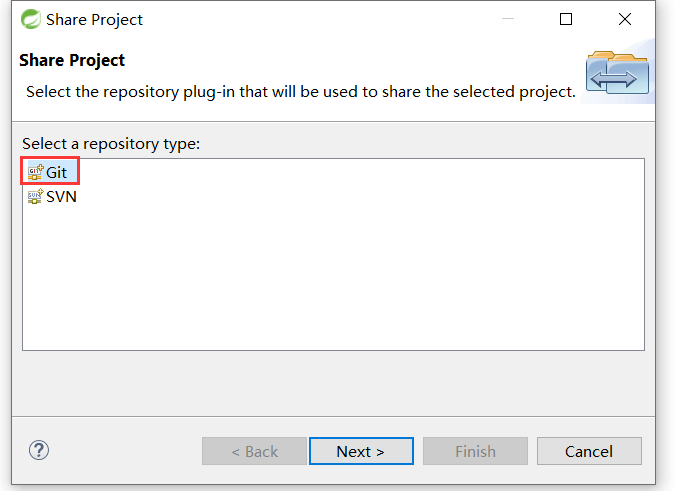


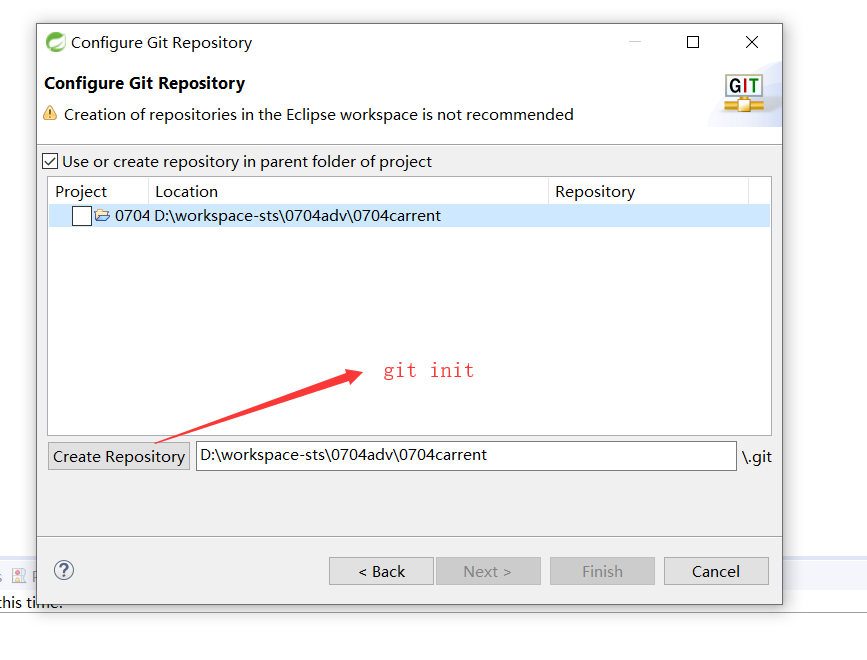
## 在gitee创建空仓库

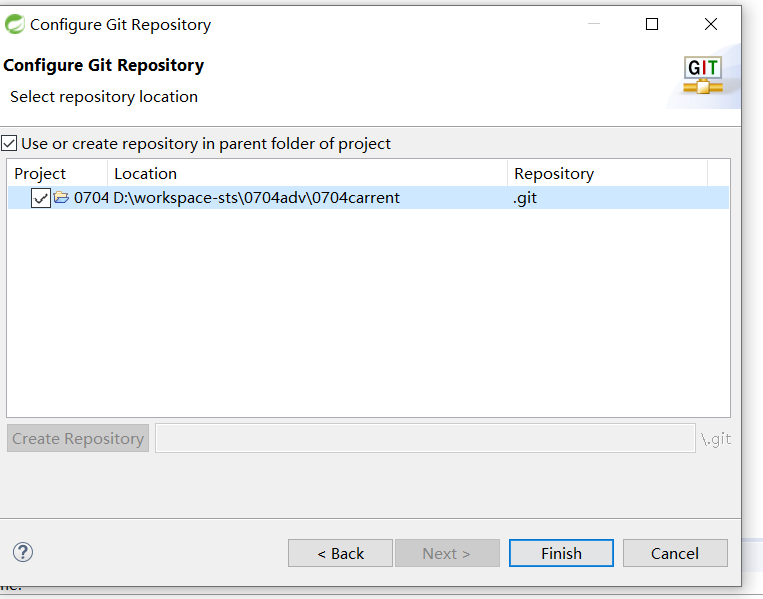


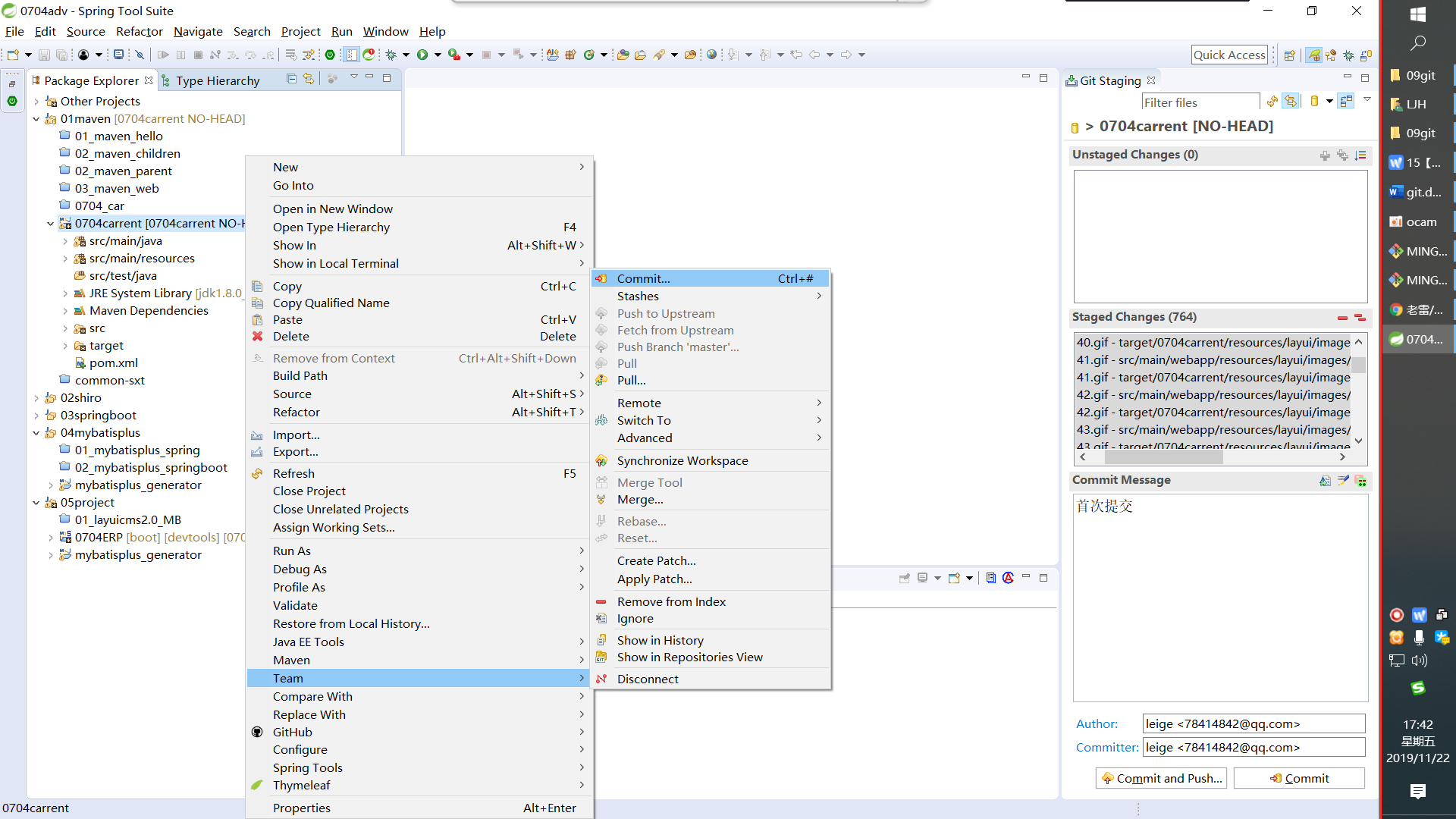
## 选择一个项目上传到gitee

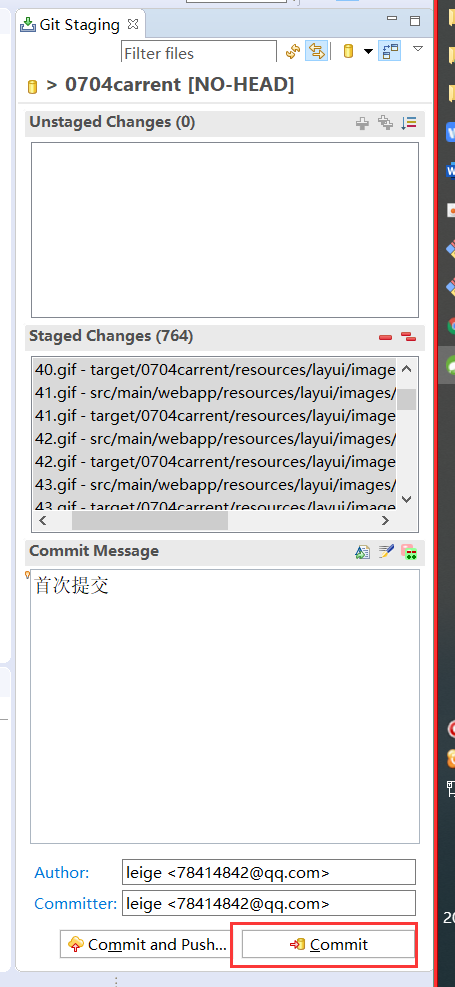




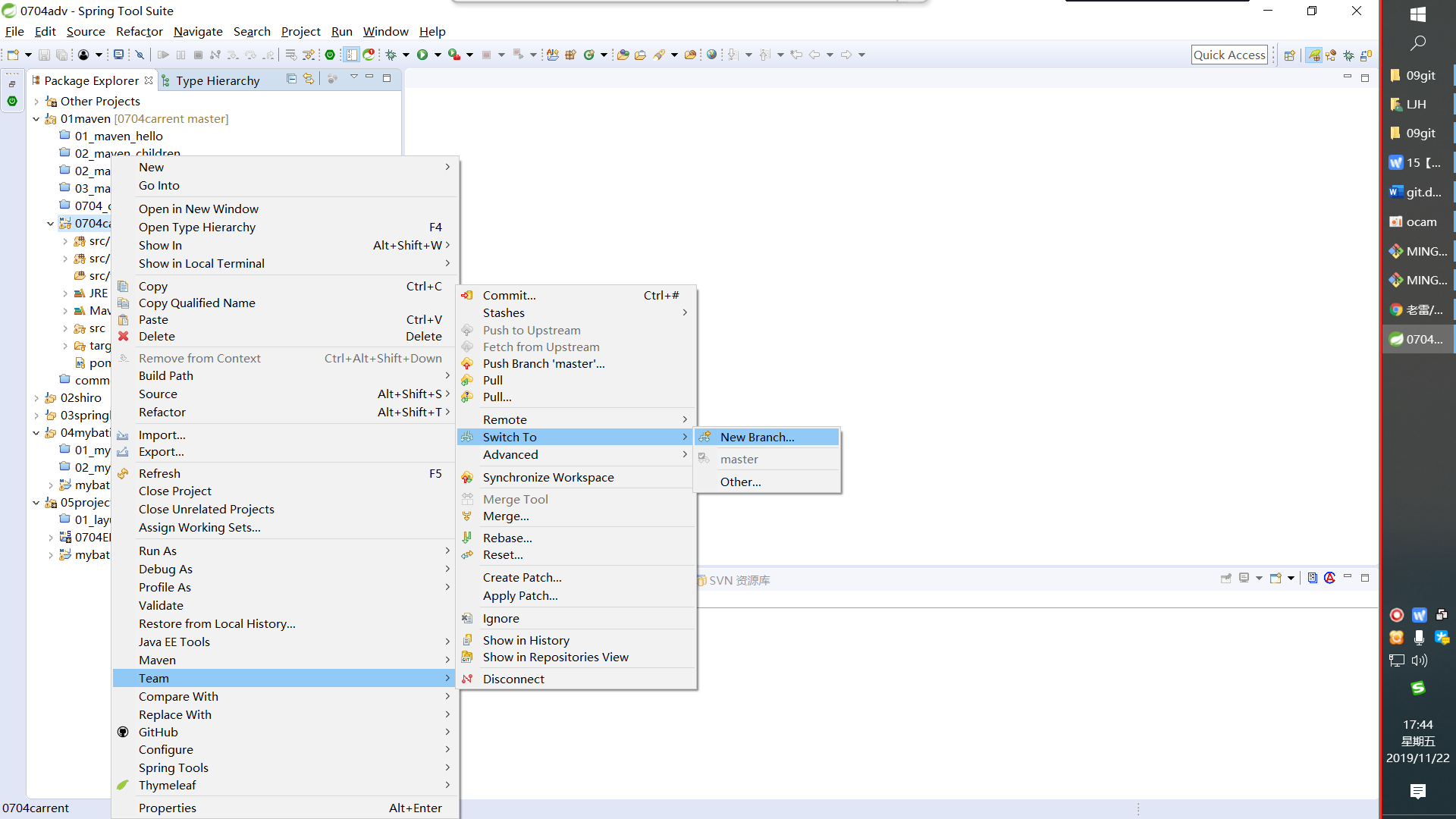




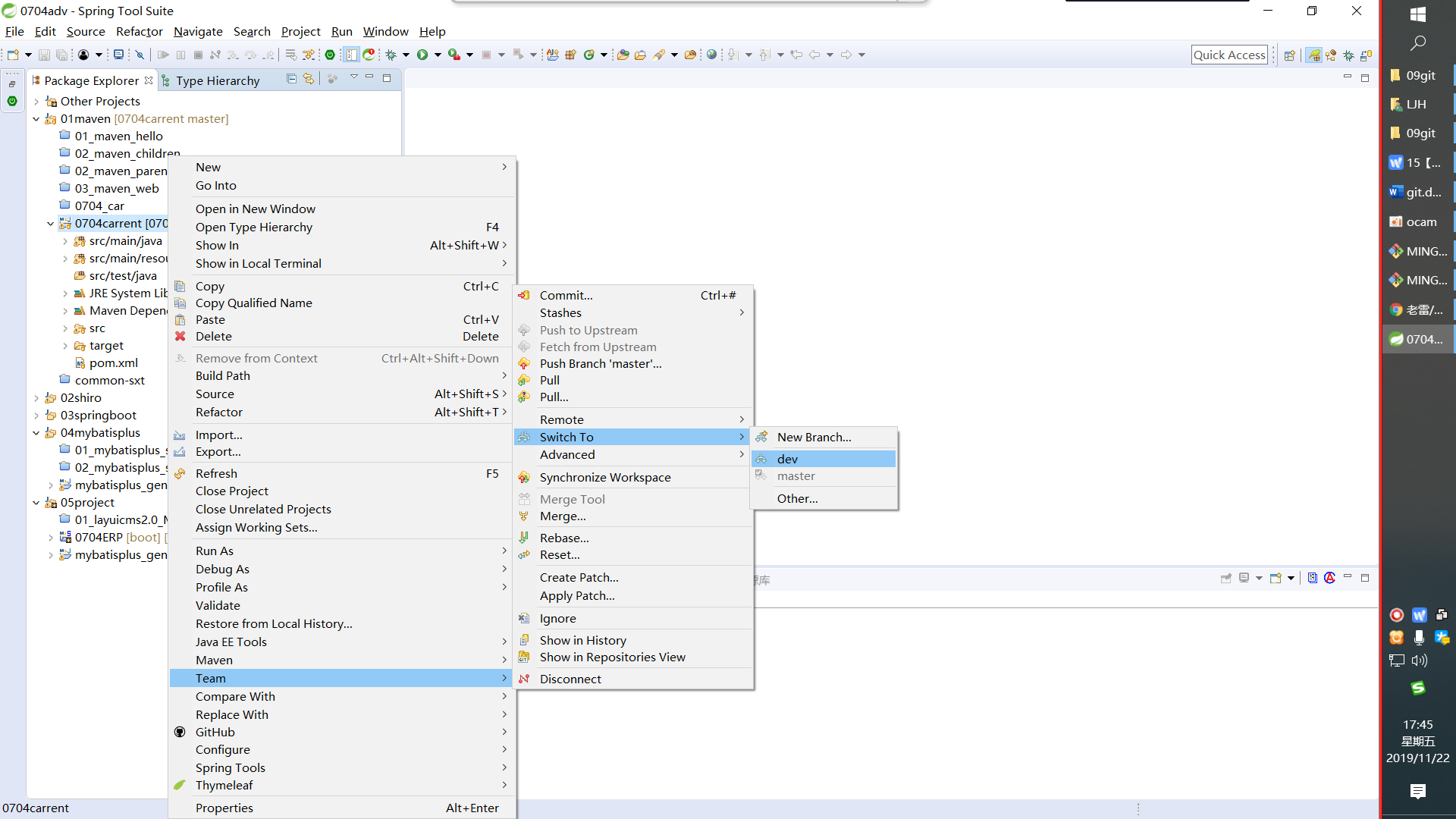




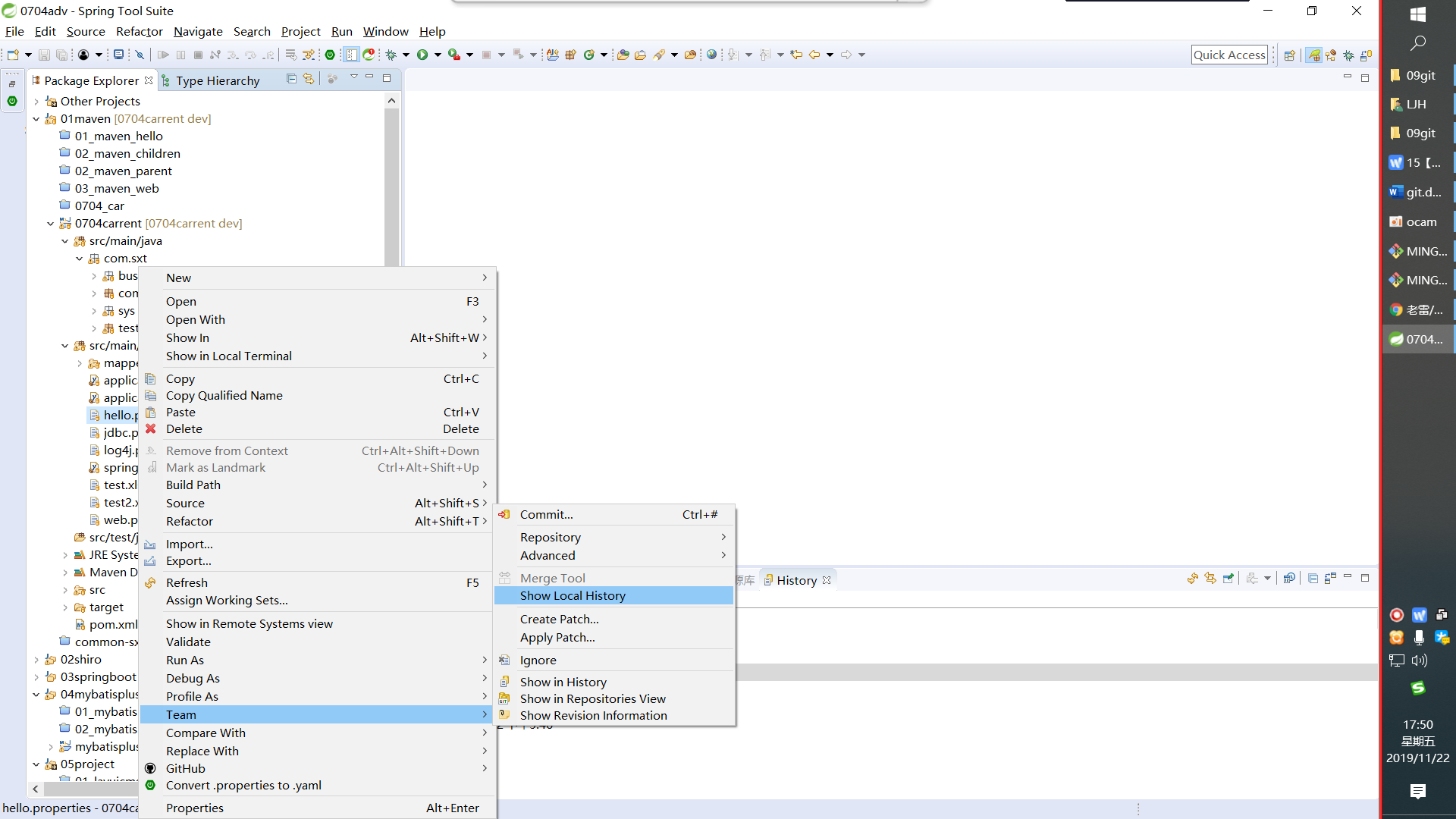
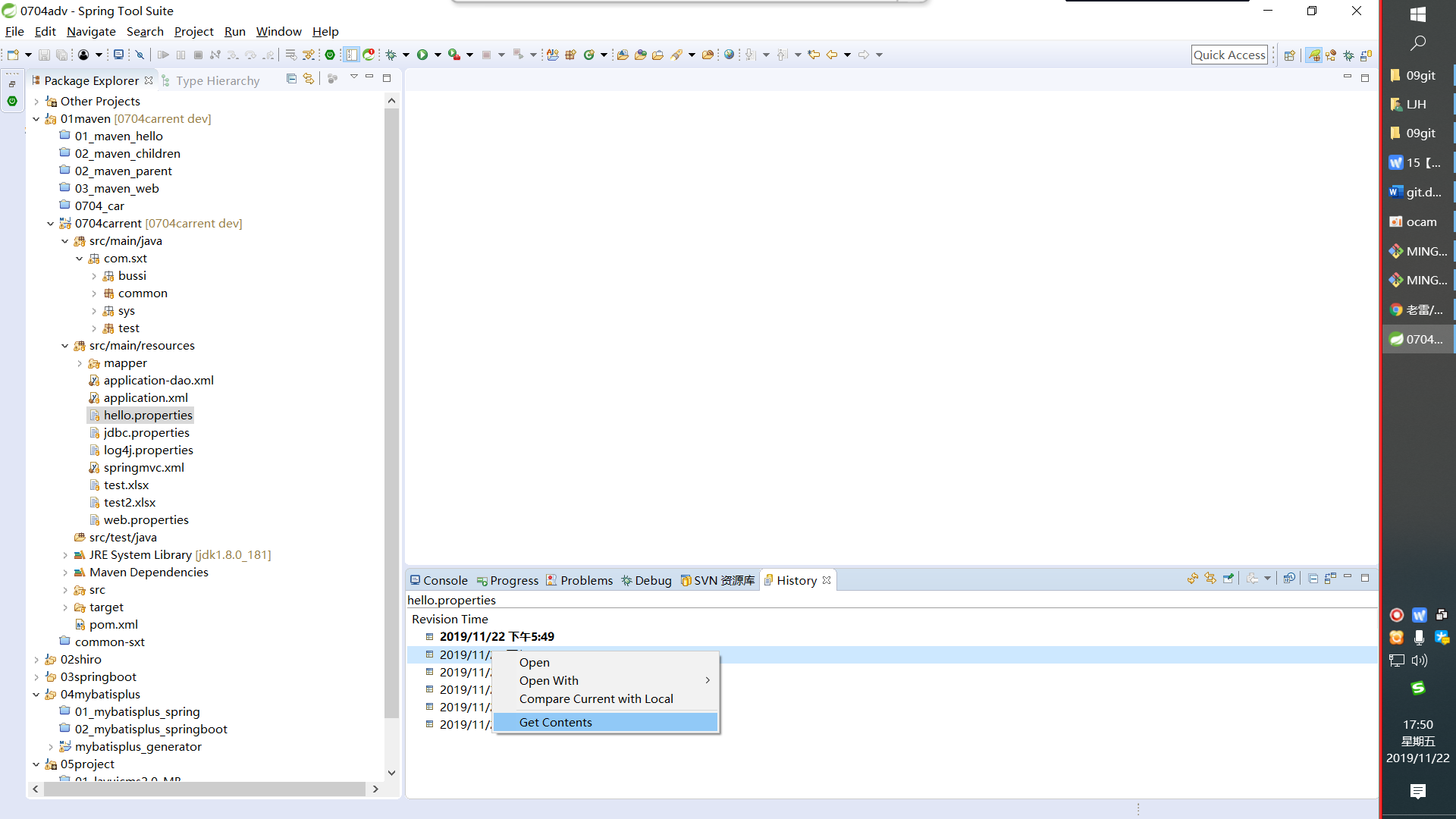
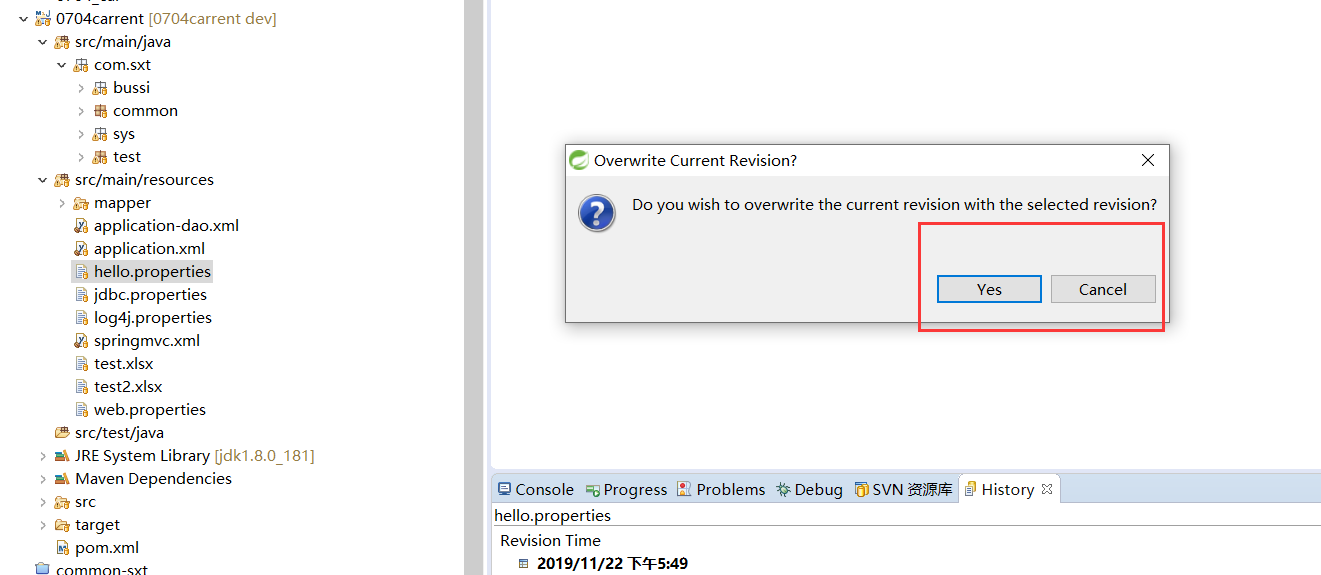
## 创建本地分支



## 切换分支

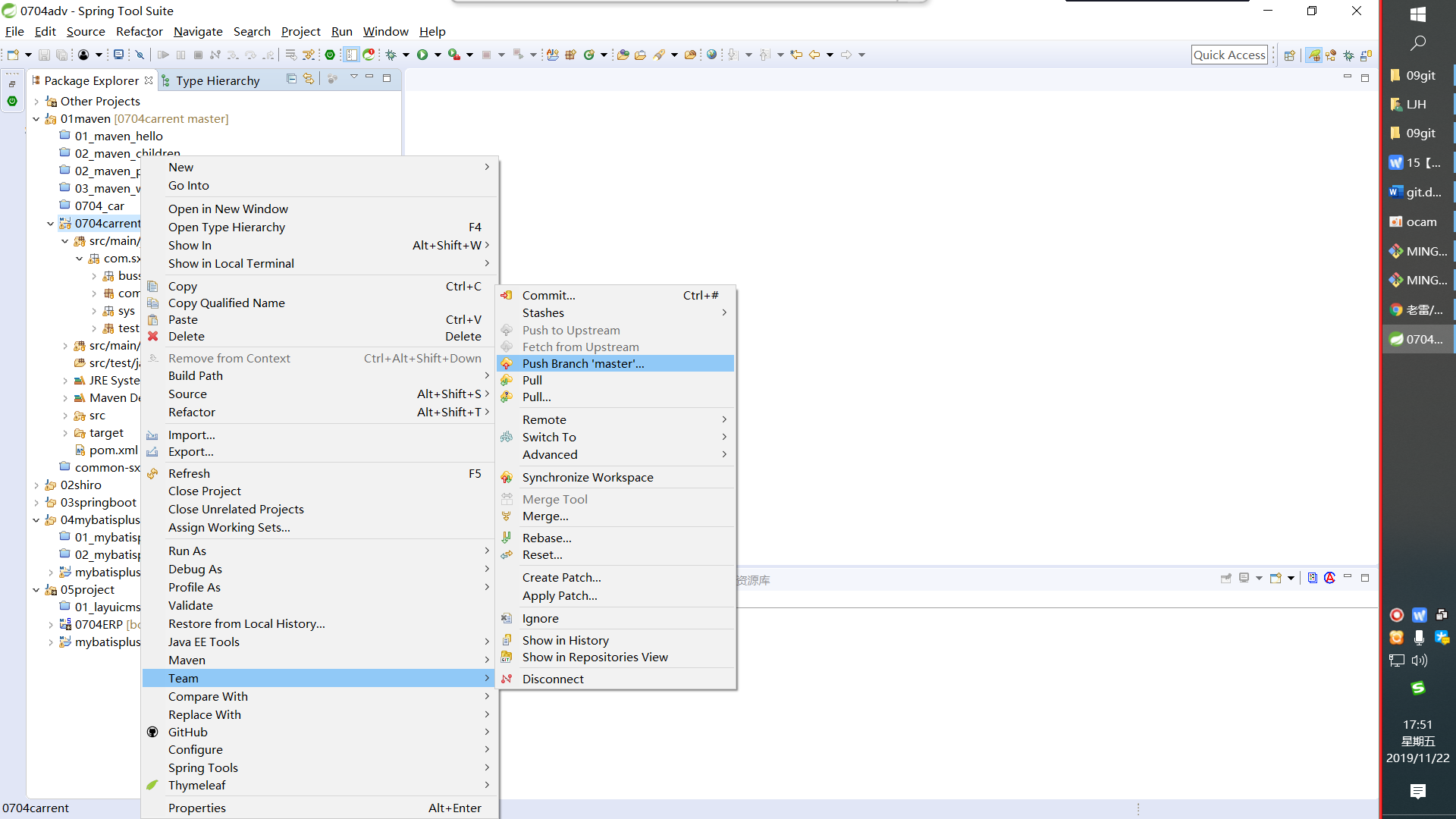


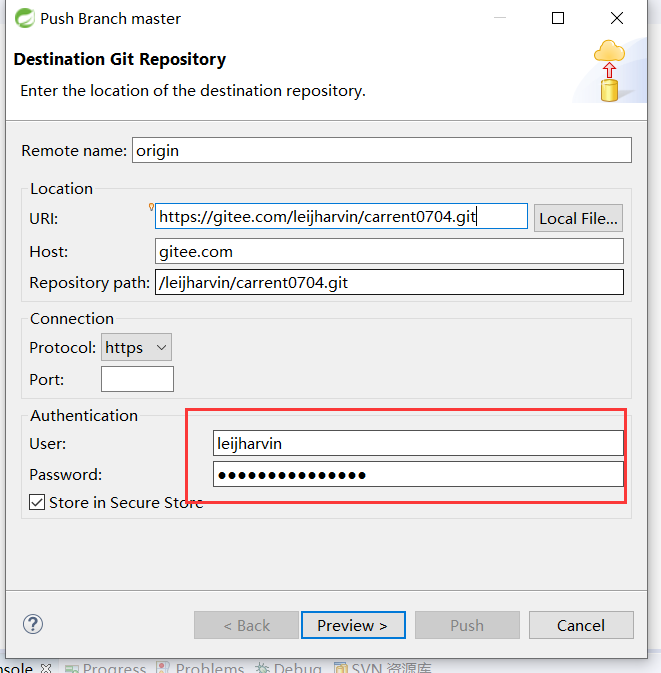
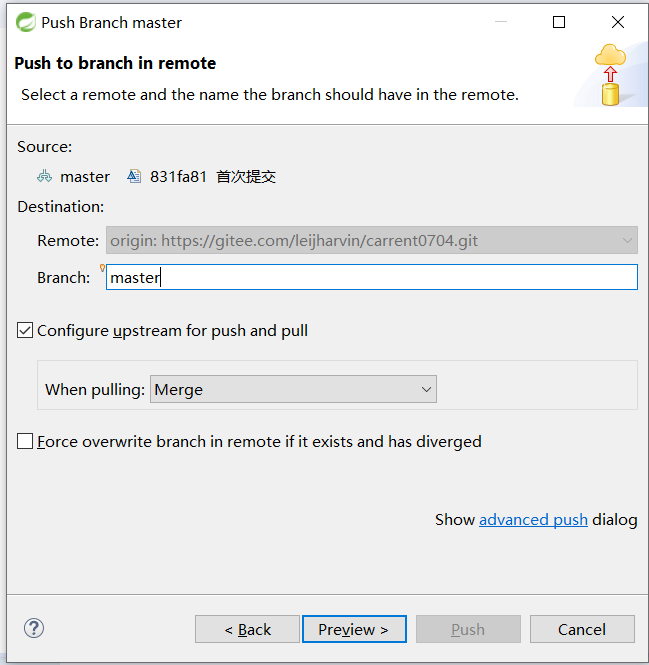
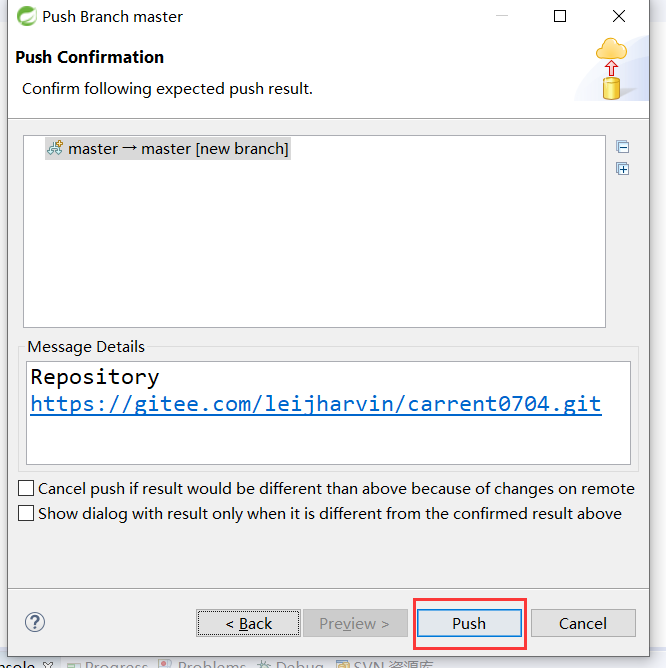
## 重置

## 把本地的项目推送到远程库

Git push -u origin …./



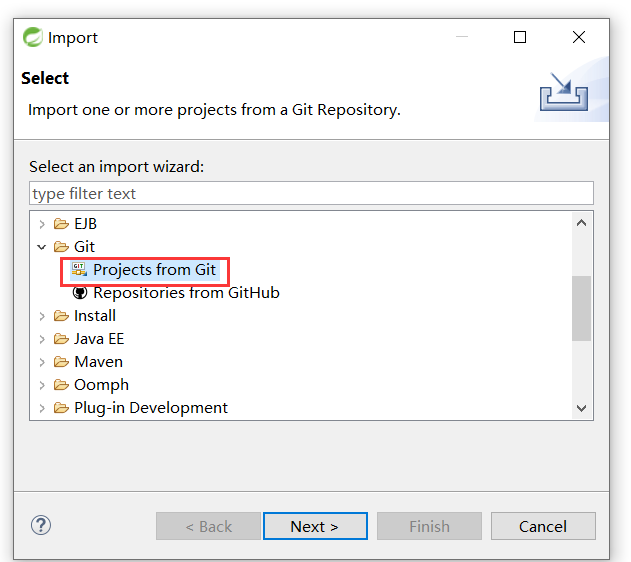
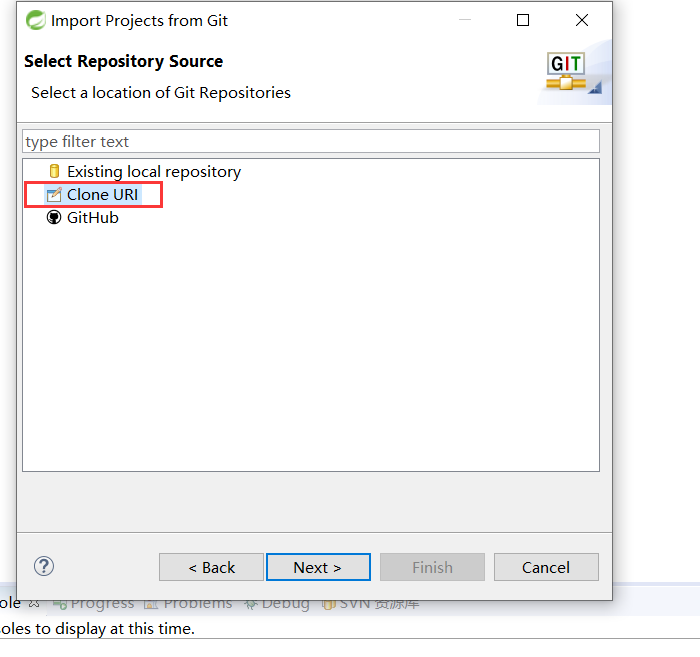
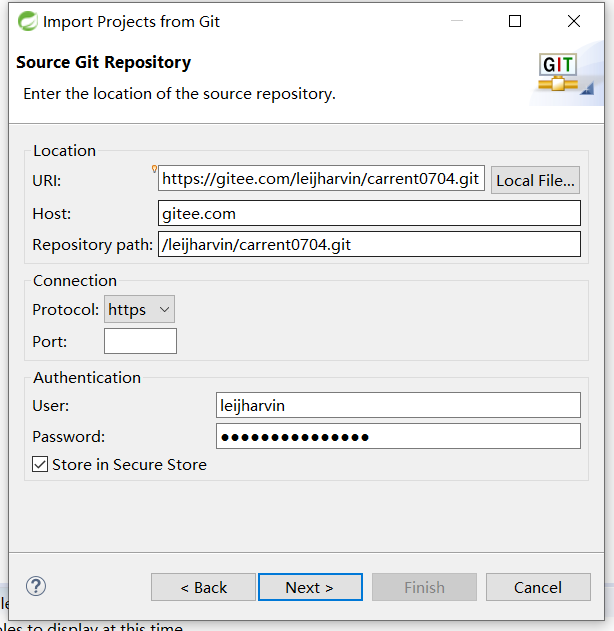
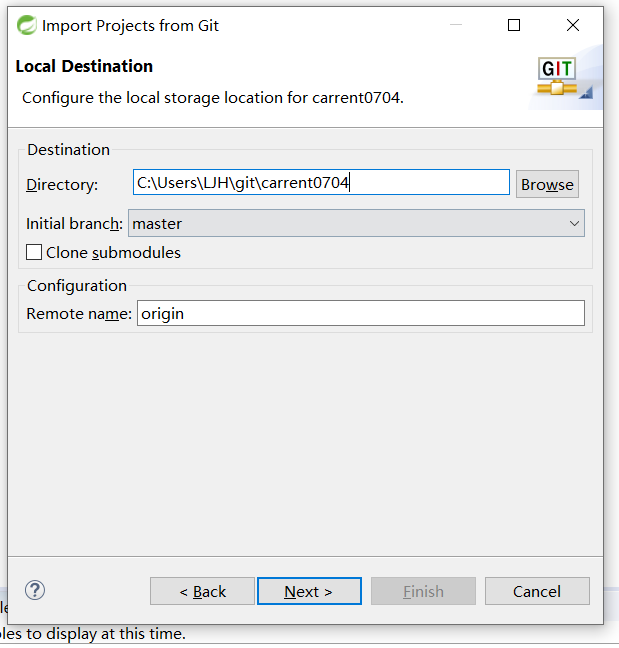
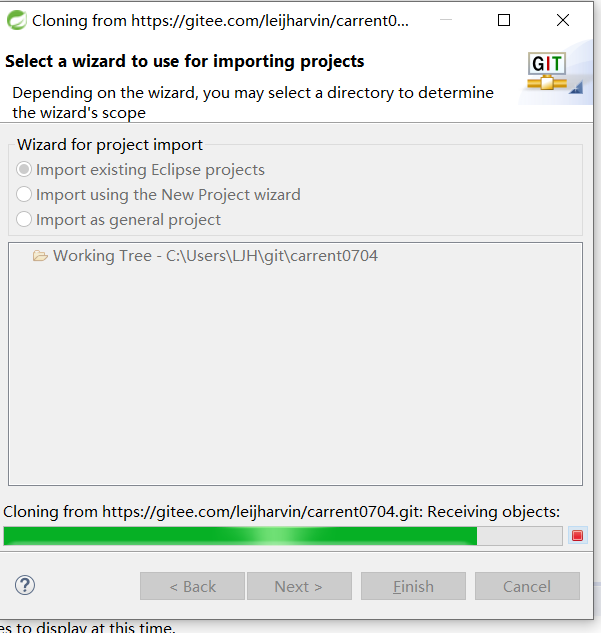
## 把远程库的项目克隆库

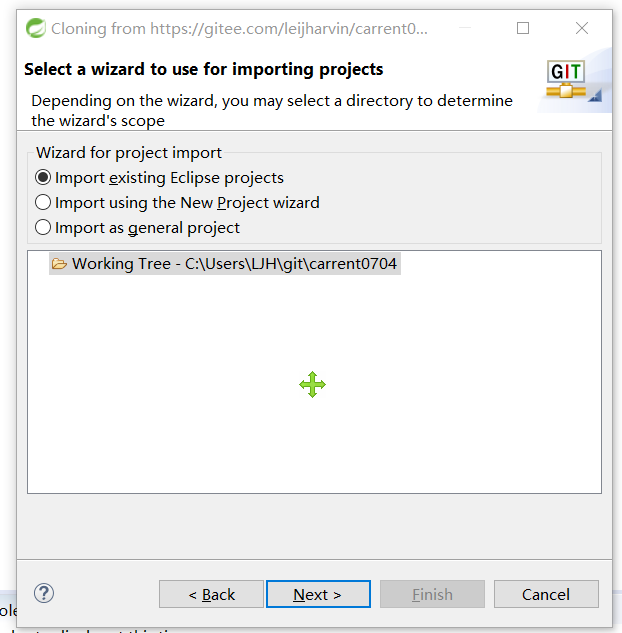
Git clone xxxxx

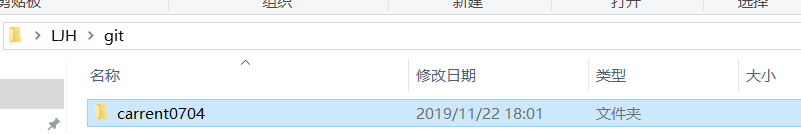
### 方式1 ，使用命令把项目clone下来再导入eclipse

git clone https://gitee.com/leijharvin/carrent0704.git

### 方式2，直接使用eclipse导入





再使用eclipse导入