

第十八章：Git

第1节：概述

1.1 Git介绍

1. 什么是Git？

Git是一个开源的分布式版本控制系统(Distributed Version Control System, 简称DVCS)。可以有效、高速地处理从很小到非常大的项目版本管理。

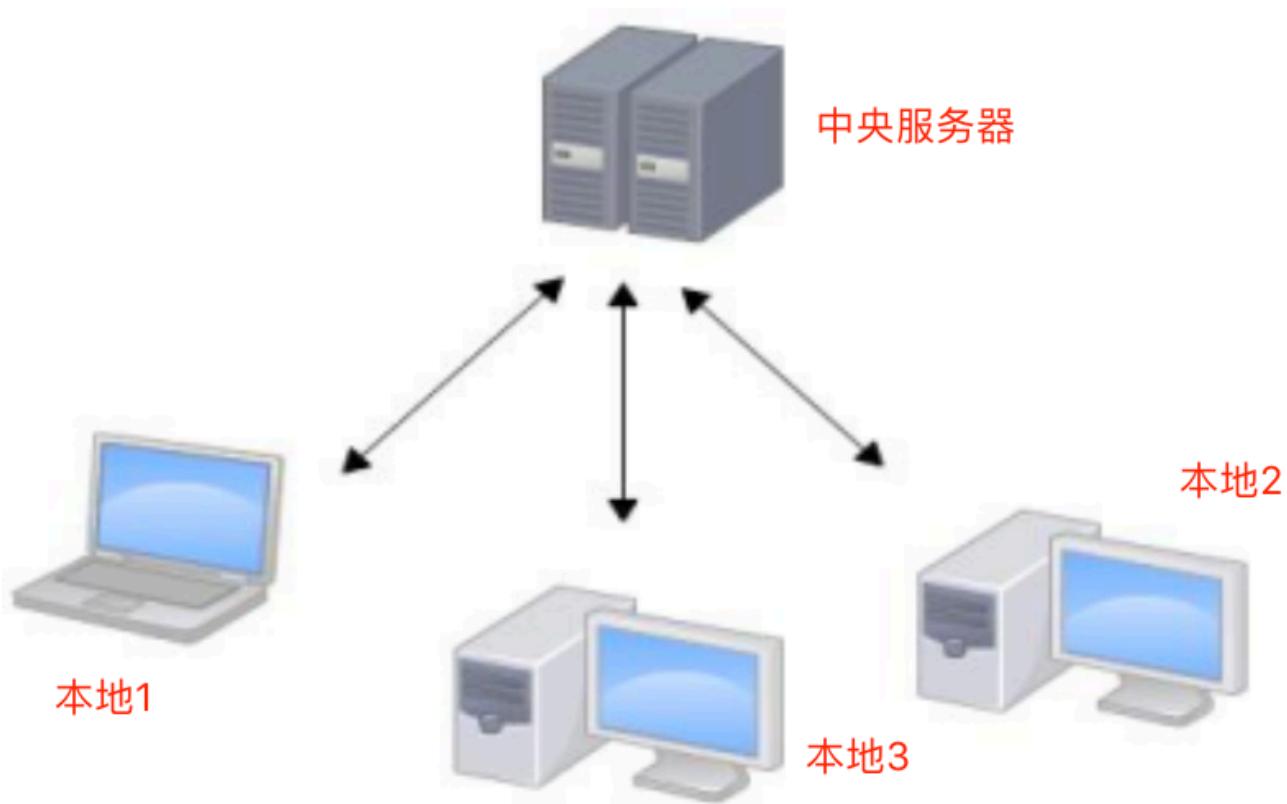
2. 什么是版本控制系统？

版本控制系统能追踪项目，从开始到结束的整个过程。对编程人员而言，版本控制技术是团队协作开发的桥梁，助力于多人协作同步进行大型项目开发。软件版本控制系统的核心任务：查阅项目历史操作记录、实现协同开发。

1.2 版本控制系统

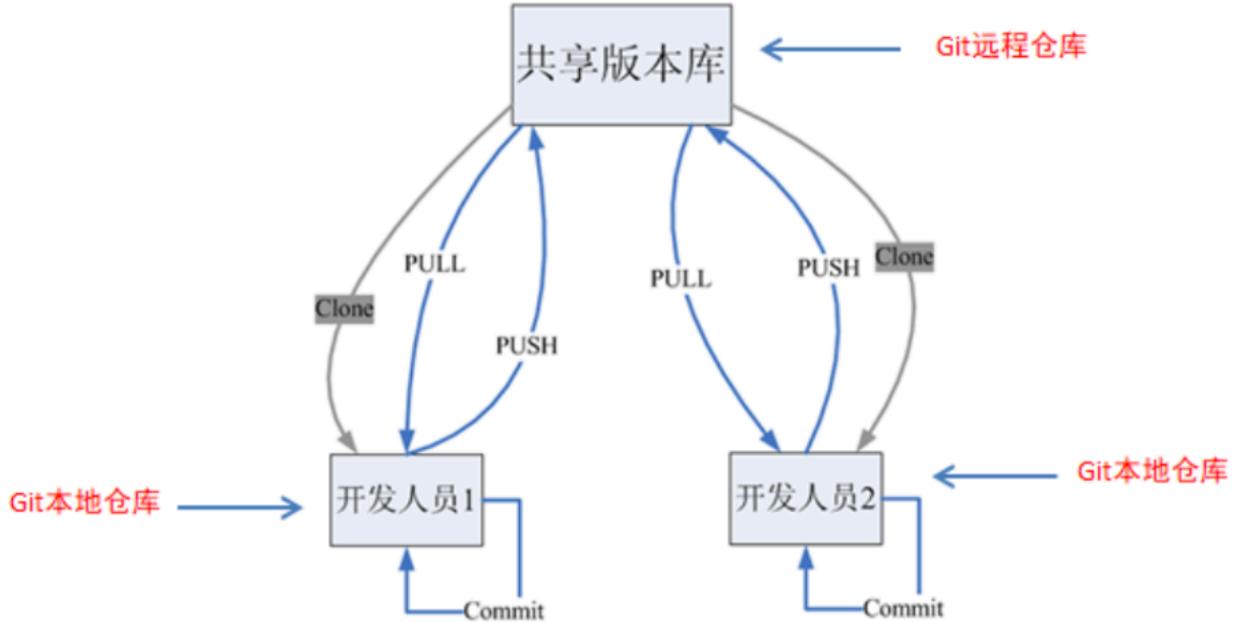
① 集中式版本控制工具

集中式版本控制工具，版本仓库是集中存放在中央服务器的，team里每个人工作时，从中央服务器下载代码。每个人个人修改后，提交到中央版本仓库。提交(commit)代码需要联网。如：svn
这会造成一个明显的问题：单点故障



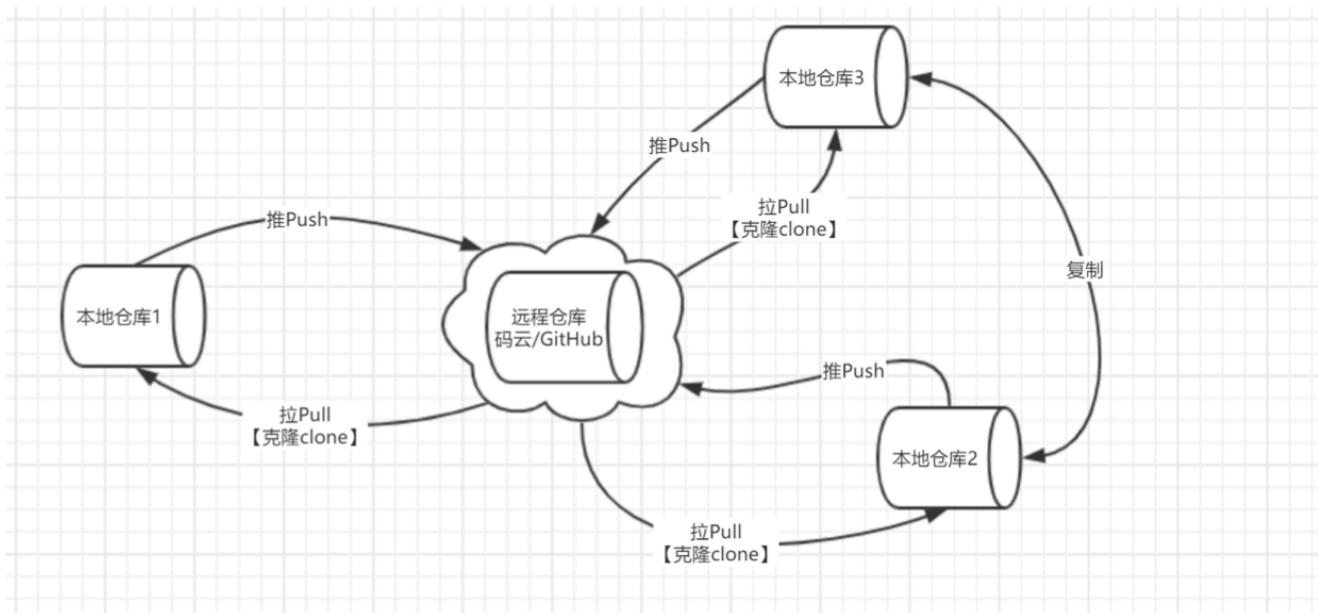
② 分布式版本控制工具

分布式版本控制系统可以没有“中央服务器”，每个人的电脑上都是一个完整的版本仓库，这样工作的时候，不需要联网。因为版本仓库就在你自己的电脑上。多人协作只需要各自修改，开发完成即可，推送给对方【联网】，推送的时候是将整个版本仓库推过去。如：Git



1.3 Git的特点

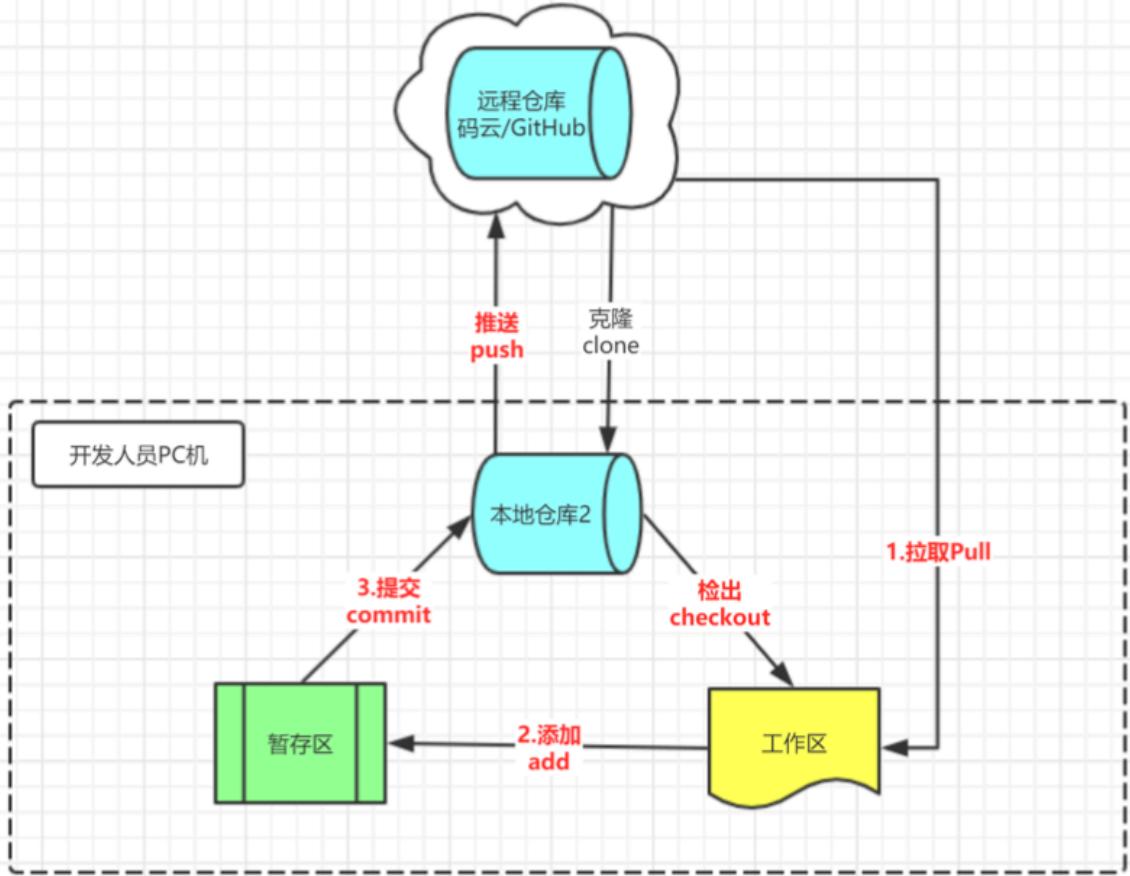
- ① 速度、简单的设计
- ② 对非线性开发模式的强力支持 (允许成千上万个并行开发的分支)
- ③ 完全分布式
- ④ 有能力高效管理类似 Linux 内核一样的超大规模项目 (速度和数据量)
- ⑤ 协同开发



Clone: 克隆，从远程仓库中克隆代码到本地仓库，第一次操作

Push: 推送，代码完成后，需要和团队成员共享代码时，将代码推送到远程仓库。

Pull: 拉取，从远程库拉代码到本地库，自动进行合并 (merge)，最后放到工作区。



checkout: 将本地仓库的内容检出到工作区
add: 在提交前先将代码提交到暂存区
commit: 提交到本地仓库

1.4 基本概念

本地仓库: 在本地主机上的一个代码库，可以独立存在，也可以与远程仓库进行关联

工作区: 对任何文件的修订(增删改)，都先放在工作区，工作区不与任何仓库分支进行关联

暂存区: 把修订的文件，从工作区经过**add**(添加)后与某一个仓库分支进行关联，只要进[入缓存区的文件才能commit](#)(提交)到本地仓库。

远程仓库 : 在局域网或互联网上的一个主机，存放代码库的主机或平台，比如GitHub、Gitee.com(码云)

分支: 代码存放在仓库，默认是主分支(master)，可以在主分支基础上创建很多子分支，比如[develop](#)(开发)、[bugfix](#)(bug修复)等。

第2节：下载和安装

2.1 下载

下载地址：<https://git-scm.com/download>

The screenshot shows the official Git website at <https://git-scm.com/>. The main navigation bar includes links for '关于' (About), '文献资料' (Documentation), '资料下载' (Downloads), 'GUI客户端' (GUI Client), '标志' (Logos), and '社区' (Community). A search bar is located at the top right. The central content area features a large '资料下载' (Downloads) section with three download links: '苹果系统' (Mac OS X), '视窗' (Windows), and 'Linux / Unix'. Below this, a note states: '可以使用旧版本，并且Git源存储库位于GitHub上。' To the right, there's a promotional section for the '最新源代码发布 2.30.0' (Latest Source Code Release 2.30.0) with a '发行说明 (2020-12-27)' (Release Notes (2020-12-27)) and a '下载Mac版2.27.0' (Download Mac version 2.27.0) button. Further down, sections for 'GUI客户端' (GUI Client) and '标志' (Logos) are visible, along with a note about using Git through its own interface.

2.2 安装

傻瓜式安装：一路下一步。安装完成后在电脑桌面右击显示

注：

Git GUI Here: Git提供的图形界面工具

Git Bash Here: Git提供的命令行工具



第3节：使用

3.1 配置

1. 安装完成 Git 后，正式使用git前，是需要进行一些全局设置的，如用户名、邮箱。

设置全局用户名

```
git config --global user.name "your name"
```

设置邮箱

```
git config --global user.email "your email"
```

```
MINGW64:/d/git_pro
junguang@junguang-PC MINGW64 ~/Desktop
$ cd D://git_pro
bash: cd: D://git_pro: No such file or directory
junguang@junguang-PC MINGW64 ~/Desktop
$ cd /d/git_pro
junguang@junguang-PC MINGW64 /d/git_pro
$ git config --global user.name "junguang"
junguang@junguang-PC MINGW64 /d/git_pro
$ git config --global user.email "zhangjunguang@163.com"
junguang@junguang-PC MINGW64 /d/git_pro
$
```

以上配置信息默认存储在用户目录下，如果设置错误，可以删除以下如图文件，重新操作以上命令即可。

	名称	修改日期	类型	大小
	.gitconfig	2021/1/23 10:17	GITCONFIG 文件	1 KB
	.minttyrc	2021/1/23 10:16	MINTTYRC 文件	1 KB
访问的位置	.npmrc	2021/1/19 15:19	NPMRC 文件	1 KB
云文档	.nrmrc	2021/1/19 15:19	NRMRC 文件	0 KB
	.node_repl_history	2021/1/19 14:59	NODE_REPL_HIS...	1 KB
	.vue-templates	2021/1/22 19:12	文件夹	
	桌面	2021/1/22 16:11	文件夹	

2. 查看配置信息: `git config --list`

```

MINGW64:/d/git_pro
junguang@junguang-PC MINGW64 /d/git_pro
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=D:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
credential.helper=manager
user.name=junguang
user.email=junguang@junguang-PC.m

```

3. 构建本地仓库: 要使用Git对我们的代码进行版本控制, 首先需要构建本地仓库

- ① 在本地初始化一个Git仓库
- ② 从远程仓库克隆一个仓库

3.2 本地操作

1. 在本地初始化一个Git仓库:

- 在电脑的任意位置创建一个空目录作为我们的本地Git仓库
 - 进入这个目录中, 点击右键打开Git bash窗口
 - 执行命令`Git init`
- 如果在当前目录中看到`.git`文件夹(此文件夹为隐藏文件夹)则说明Git仓库创建成功

2. 查看文件状态:

```
git status [-s]
```

```
MINGW64:/d/git_pro
junguang@junguang-PC MINGW64 /d/git_pro (master)
$ git status
on branch master

Initial commit                                     没有可跟踪文件

nothing to commit (create/copy files and use "git add" to track)

junguang@junguang-PC MINGW64 /d/git_pro (master)
$
```

3. 将文件添加(修改)到版本库

要将一个文件纳入到版本库管理，首先要将其添加到暂存区，然后才能提交到仓库中。

```
# 添加单个文件到暂存区
git add Readme.txt
# 将当前目录下所有修改添加到暂存区，除按照规则忽略的之外
git add .
```

4. 将暂存区中的文件，提交到仓库中

```
# 如果暂存区有文件，则将其中的文件提交到仓库
git commit
# 带评论提交，用于说明提交内容、变更、作用等
git commit -m 'your comments'
```

5. 查看提交历史记录

有的时候，是会需要查看自己做过哪些提交，来回顾自己完成的部分。或者需要寻找某个具体的提交来查看当时的代码。

```
git log # 显示所有提交的历史记录
git log --pretty=oneline # 单行显示提交历史记录的内容
```

6. 版本回退

有了 `git log` 来查看提交的历史记录，我们就可以通过 `git reset --hard` 来回退到我们需要的特定版本，然后使用当时的代码进行各种操作。

```
# 回退到 commit_id 指定的提交版本
git reset --hard 'commit_id'
```

7. 回到未来的某个提交

当退回到某个提交的版本以后，再通过 `git log` 是无法显示在这之后的提交信息的。但是，通过 `git reflog` 可以获取到操作命令的历史。因此，想要回到未来的某个提交，先通过 `git reflog` 从历史命令中找到想要回到的提交版本的 ID，然后通过 `git reset --hard` 来切换。

```
git reflog  
git reset --hard 'commit_id'
```

8. 删 除文件

在文件未添加到暂存区之前，对想删除文件可以直接物理删除。如果文件已经被提交，则需要 `git rm` 来删除

```
git rm Readme.txt // 删除已经被提交过的 Readme.txt  
注意：git rm 只能删除已经提交到版本库中的文件。其他状态的文件直接用这个命令操作是出错的。
```

9. 添加文件至忽略列表

一般在工作区中，并不是所有文件都需要纳入版本控制的。这种不需要进行版本控制的通常都是些自动生成的文件。比如：idea工程文件(`springmvc.iml`)、编译后文件`target`、系统上传的图片`img`。在这种情况下，我们可以在工作目录中创建一个名为 `.gitignore` 的文件(文件名称固定)，列出要忽略的文件。

第4节：分支管理

4.1 查看分支

```
# 查看本地分支信息  
git branch  
# 查看相对详细的本地分支信息  
git branch -v  
# 查看包括远程仓库在内的分支信息  
git branch -av
```

注意：前面带有*号，这标识我们当前所在的分支

4.2 创建分支

```
# 新建一个名称为 dev 的分支  
git branch dev
```

4.3 切换分支

```
# 新建完 dev 分支以后，通过该命令切换到 dev 分支  
git checkout dev
```

注意：当我们创建完分支以后，我们需要切换到新建的分支，否则，所有的修改，还是在原来的分支上。事实上，所有的改动，只能影响到当前所在的分支。

4.4 创建并切换分支

```
# 新建 dev 分支，并切换到该分支上  
git checkout -b dev
```

4.5 合并分支

```
# 切换回 master 分支  
git checkout master  
# 将 dev 分支中的修改合并回 master 分支  
git merge dev  
注意：分支修改文件中如果有换行的话会报错
```

```
junguang@junguang-PC MINGW64 /d/git_pro (dev1)  
$ git add Readme.txt  
warning: LF will be replaced by CRLF in Readme.txt.  
The file will have its original line endings in your working directory.
```

解决：Git默认配置替换成车换行成统一的CRLF，我们只需要修改配置禁用该功能即可。

```
git config --global core.autocrlf false
```

4.6 删除分支

当之前创建的分支，完成了它的使命，如 Bug 修复完，分支合并以后，这个分支就不在需要了，就可以删除它。

```
# 删除dev分支  
git branch -d dev
```

第5节：远程仓库

现在我们已经在本地创建了一个Git仓库，又想让其他人来协作开发，此时就可以把本地仓库同步到远程仓库，同时还增加了本地仓库的一个备份。那么我们如何搭建Git远程仓库呢？我们可以借助互联网上提供的一些代码托管服务平台来实现，其中比较常用的有GitHub、码云等。

-- GitHub(地址：<https://github.com/>)是一个面向开源及私有软件项目的托管平台，因为只支持 Git 作为唯一的版本仓库格式进行托管，故名GitHub。

-- 码云(地址：<https://gitee.com/>)是国内的一个代码托管平台，由于服务器在国内，所以相比于 GitHub，码云速度会更快。

接下来我们演示如何将本地仓库中的代码同步到码云。GitHub一样

5.1 注册账号

① 第一步，点击注册按钮

The screenshot shows the Gitee homepage. At the top right, there is a red arrow pointing to the "注册" (Register) button, which is highlighted with a red border. Other buttons at the top include "登录" (Login), "搜索" (Search), and user profile icons.

企业级 DevOps 研发管理平台

帮助开发者/团队/企业更好地管理代码，让软件研发更高效

注册 Gitee **免费开通企业版**

开发者 代码仓库 企业客户 高校
800 万+ 2000 万+ 20 万+ 3500 +

开源/个人代码管理 企业/机构研发管理 高校教学管理

5.2 登录使用

1. 登录

The screenshot shows the Gitee login page. On the left, there is a dark sidebar with the Gitee logo and a brief introduction. On the right, the main login form has fields for "手机 / 邮箱 / 个人空间地址" (Phone number / Email / Personal space address) and "请输入密码" (Enter password). There are checkboxes for "记住我" (Remember me) and "短信验证登录" (SMS verification login). A large orange "登 录" (Login) button is centered. Below it, links for "已有帐号, 忘记密码?" (Have an account, forgot password?) and "使用 OSChina 帐号登录" (Log in with OSChina account) are provided, along with other social media login options.

2. 创建仓库

The screenshot shows two pages from Gitee.com:

- User Profile Page:** The top half shows the user's profile (junguang00), activity feed for 2021, and a list of repositories. A modal window titled "关于暂停纸质专用发票邮寄的通知" is open. On the right, there are sections for recommended users and repositories.
- New Project Creation Page:** The bottom half shows the "New Repository" form. It includes fields for repository name, owner, path, description, visibility (private selected), and initialization options. A note about creating a public repository is highlighted in a red box.

第6节：同步远程仓库

6.1 HTTP方式

码云支持两种同步方式“`https`”和“`ssh`”。如果使用`https`很简单基本不需要配置就可以使用，但是每次提交代码和下载代码时都需要输入用户名和密码。而且如果是公司配置的私有git服务器一般不提供`https`方式访问，所以我们要来着重演示“`ssh`”方式。

The screenshot shows a Gitee repository page for 'junguang00 / smallU_1012'. The repository has 1 star, 0 forks, and 1 issue. It contains files like 'src', 'target', and 'pom.xml'. A README section is present with a placeholder '添加一个 README.md 文件，帮助感兴趣的人了解。添加 README'.

近期动态:

- J 3个月前推送了新的提交到 master 分支, 78e81c7..8df0ef3
- J 3个月前推送了新的提交到 master 分

6.2 SSH方式

SSH是英文Secure Shell的简写形式。通过使用SSH, 你可以把所有传输的数据进行加密, 这样"中间人"这种攻击方式就不可能实现了, 而且也能够防止DNS欺骗和IP欺骗。使用SSH, 还有一个额外的好处就是传输的数据是经过压缩的, 所以可以加快传输的速度。

注: 使用SSH同步方式需要先生成密钥并在Gitee配置公钥

执行命令, 生成公钥和私钥:

```
ssh-keygen -t rsa
```

执行命令完成后, 在window本地用户.ssh目录C:\Users\用户名.ssh下面生成如下名称的公钥和私钥

密钥生成后需要在gitee上配置密钥, 本地才可以顺利访问。

The screenshot shows the Gitee.com profile page for user 'junguang00'. On the left, there's a sidebar with sections for '仓库' (Repositories), 'Pull Requests', 'Issues', '代码片段' (Code Snippets), and '我的星选集' (My Star Collection). A note says '现在已支持通过「星选集」来分类管理「我 Star 的仓库」'. Below this are sections for '企业' (Enterprise) and '组织' (Organizations), with a note about creating a free enterprise edition. The main area shows '动态' (Activity) with posts from 'junguang00' dated '2021-12-28' and '2021-12-27'. Each post includes a commit message and a timestamp. To the right, there's a sidebar for '推荐关注' (Recommended Follow) with profiles like '叛道' (2D/Web/UI Designer), 'wangzhonnew', '夏楚', 'smalljop', and '王斌'. There's also a section for '推荐仓库' (Recommended Repositories) with '大屏数据展示模板' (Large-screen data display template) and 'deepin-terminal'.

The screenshot shows the 'SSH公钥' (SSH Keys) section of the Gitee.com profile page for user 'junguang00'. On the left, there's a sidebar with '消息中心', '基本设置', '安全设置', and 'SSH公钥' (selected and highlighted with a red border). Under 'SSH公钥', there's a note: '使用SSH公钥可以让你在你的电脑和 Gitee 通讯的时候使用安全连接 (Git的Remote要使用SSH地址)'. It lists two existing keys: 'junguang@junguang-PC' and 'junguang@junguang-PC'. Below this is a form for adding a new key, with fields for '标题' (Title) containing '公钥标题(key)' and '公钥' (Key) containing placeholder text. A '确定' (Confirm) button is at the bottom right of the form.

第7节：远程仓库的操作

7.1 查看关联的远程仓库

如果想查看已经配置的远程仓库服务器，可以运行 `git remote` 命令。它会列出指定的每一个远程服务 器的简写。如果已经克隆了远程仓库，那么至少应该能看到 `origin`，这是 Git 克隆的仓库服务器的默认名字

```
# 命令形式:  
git remote -v  
# origin —仓库服务器的默认名称  
注: 如果显示空, 没有添加远程仓库。如果添加显示远程仓库地址
```

7.2 添加远程仓库

如果已经有了一个本地仓库, , 然后打算将它发布到远程, 供其他人协作。

那么使用命令:

```
# 为本地仓库添加远程仓库  
git remote add origin 远程仓库地址  
如: git remote add origin https://gitee.com/junguang00/git_demo01.git
```

注:

提示出错信息: fatal: remote origin already exists.

先输入

```
$ git remote rm origin
```

再输入:

```
git remote add origin 远程仓库地址
```

7.3 推送本地内容到远程仓库

```
# 在本地仓库更新内容 添加暂存区  
git add .  
# 提交内容  
git commit -m '信息'  
# 当本地仓库中, 代码完成提交, 就需要将代码等推送到远程仓库, 这样其他协作人员可以从远程仓库同步内容。
```

第一次推送时使用, 可以简化后面的推送或者拉取命令使用

```
git push -u origin master
```

将本地 master 分支推送到 origin 远程分支

```
git push origin master
```

注意:

① `git push -u origin master` , 第一次使用时, 带上 `-u` 参数, 在将本地的 `master` 分支推送到远程新的 `master` 分支的同时, 还会把本地的 `master` 分支和远程的 `master` 分支关联起来。

② 推送之前, 需要先pull远端仓库, 如果发现提交版本不一致, 出现错误

```
git pull --rebase origin master  
git push origin master
```

7.4 移除无效的远程仓库

命令形式:

```
git remote rm <shortname>
```

注意:此命令只是从本地移除远程仓库的记录，并不会真正影响到远程仓库

7.5 从远程仓库克隆

Git 克隆的是该 Git 仓库服务器上的几乎所有数据(包括日志信息、历史记录等)，而不仅仅是复制工作所需要的文件。当你执行 `git clone` 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。

如果你本地没有仓库，希望从已有的远程仓库上复制一份代码，那么你需要 `git clone`。

通过 https 协议，克隆 Github 上 git 仓库的源码

```
git clone https://github.com/lagou-zimu/repo1.git
```

通过 ssh 协议，克隆 Github 上 git 仓库的源码

```
git clone git@github.com:lagou-zimu/repo1.git
```

7.6 从远程仓库中拉取

拉取 命令形式:git pull 【远程仓库名称】 【分支名称】

在多人协作过程中，当自己完成了本地仓库中的提交，想要向远程仓库推送前，需要先获取到远程仓库的最新内容。使用如下命令

```
git fetch origin master
```

```
git pull origin master
```

git fetch 和 git pull 之间的区别：

git fetch 是仅仅获取远程仓库的更新内容，并不会自动做合并。

git pull 在获取远程仓库的内容后，会自动做合并，可以看成 git fetch 之后 git merge。

7.7 解决合并冲突

在一段时间，A、B用户修改了同一个文件，且修改了同一行位置的代码，此时会发生合并冲突。

例如：

A用户在本地修改代码后优先推送到远程仓库，此时B用户在本地修订代码，提交到本地仓库后，也需要推送到远程仓库，此时B用户晚于A用户推送，故需要先拉取远程仓库代码，经过合并后才能推送到代码。在B用户拉取代码时，因为A、B用户同一时间段修改了同一个文件的相同位置代码，故会发生合并冲突。

解决：

- ① 先拉取代码
- ② 打开代码解决冲突
- ③ 再提交

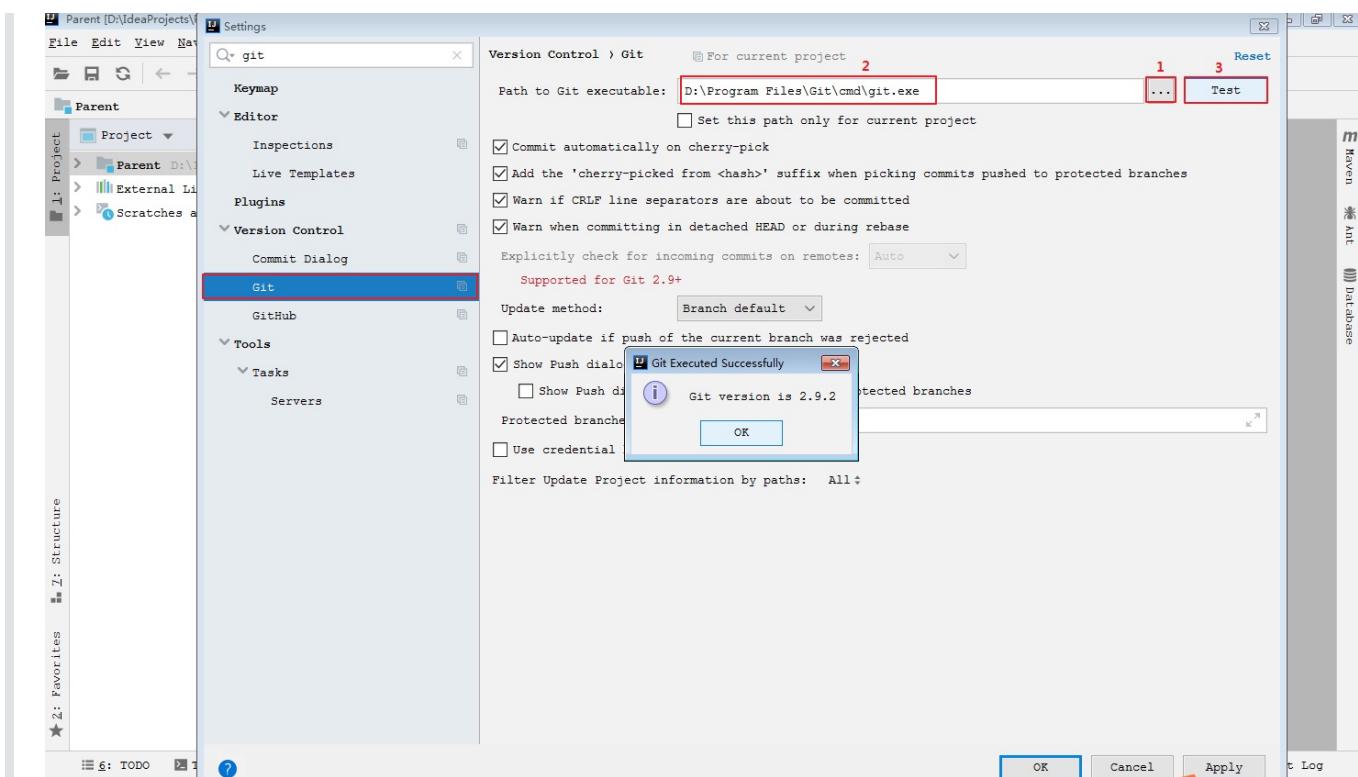
7.8 小结

远程仓库常见操作命令：

```
git remote #查看所有远程仓库名称  
git remote -v #查看远程仓库缩略信息  
git push origin master # 将本地仓库代码推送到远程仓库  
git clone https://github.com/lagou-zimu/repol.git # 克隆远程仓库代码到本地 git pull  
origin master # 拉取远程仓库代码到本地:(fetch+merge)
```

第8节：Idea中集成Git

8.1 在idea中配置Git



8.2 Idea的Git操作

8.2.1 初始化并提交项目到远程仓库 【项目leader操作】

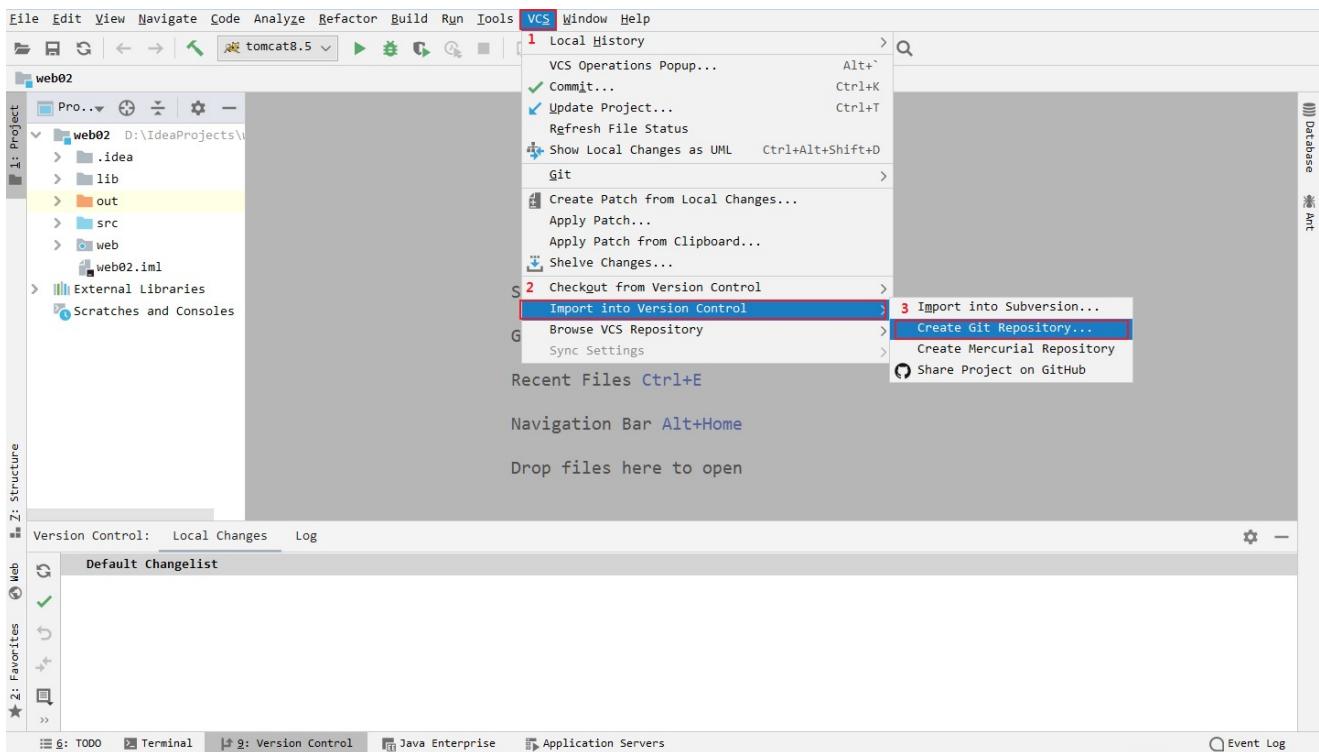
① 初始化并提交项目到远程仓库 【项目leader操作】

执行步骤：

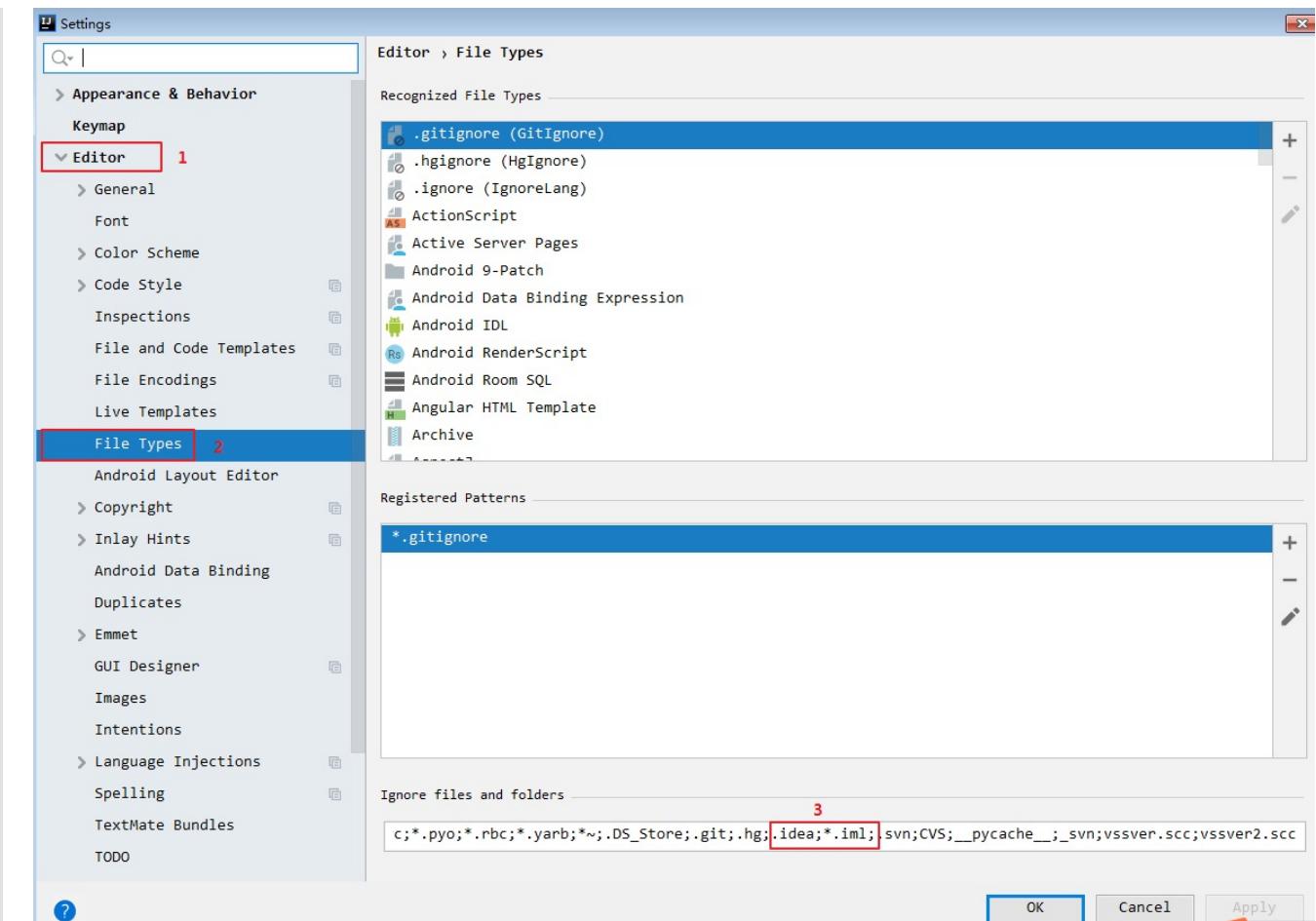
1. 在GitHub/码云中创建远程仓库
2. 将工程交给Git管理
3. 配置忽略文件
4. 提交到本地仓库
5. 推送到远程仓库

② 先在码云上创建仓库

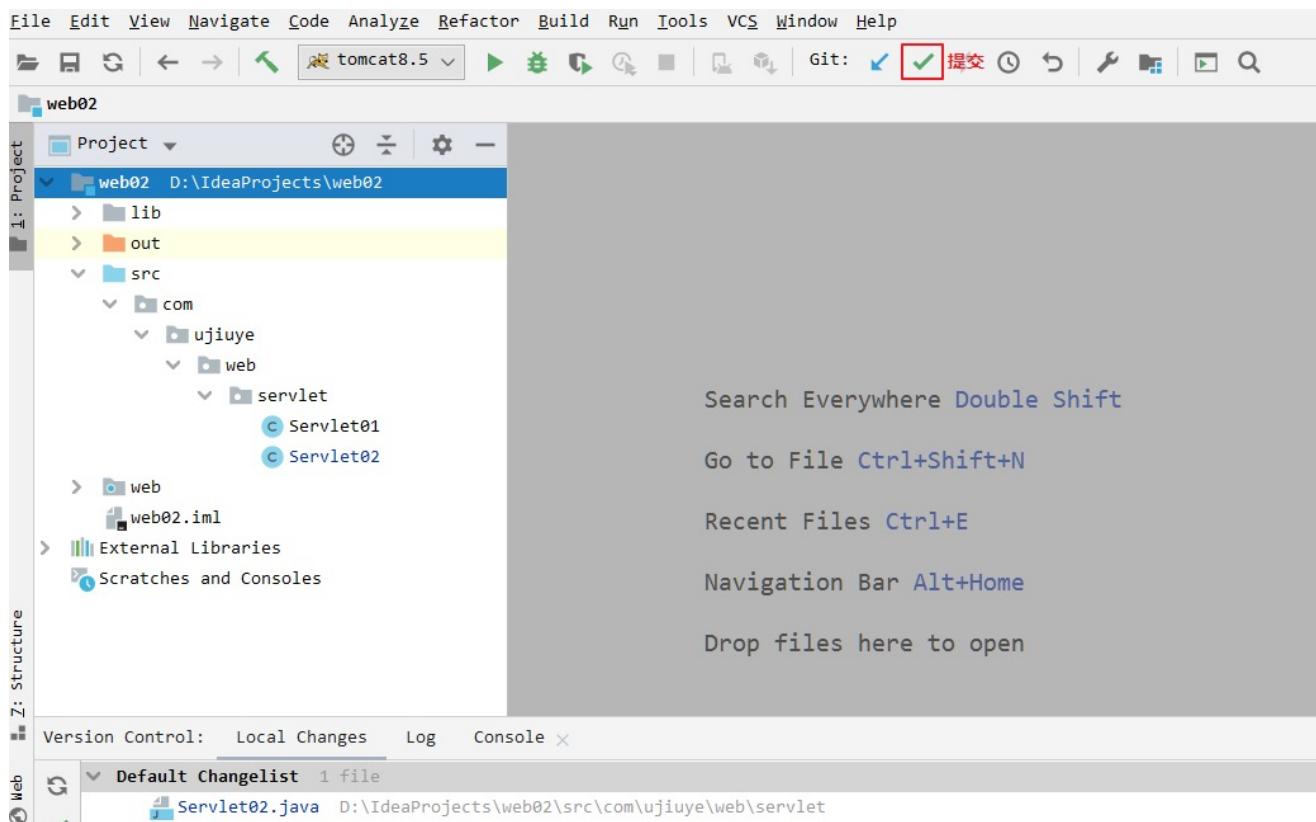
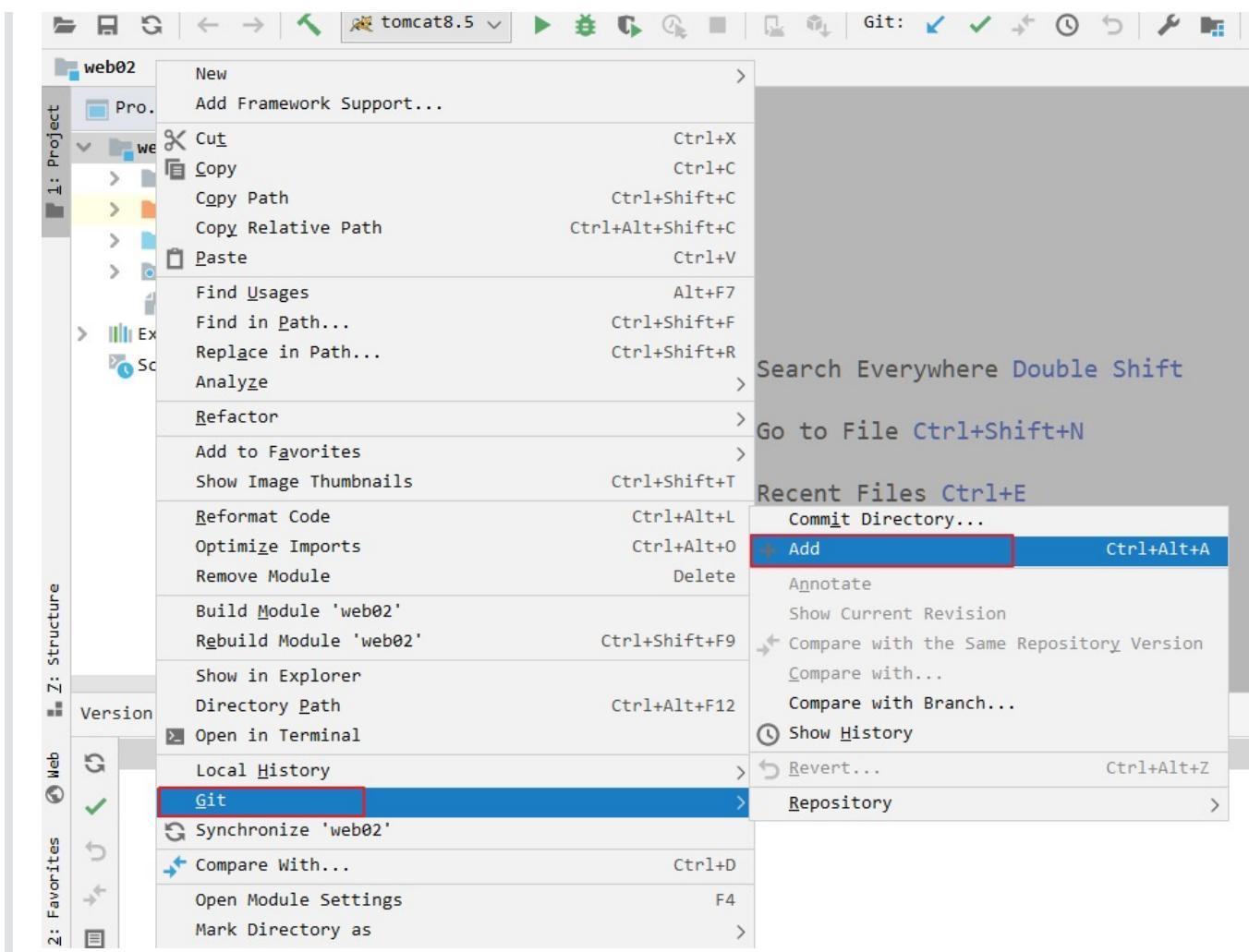
③ 将工程交给Git管理



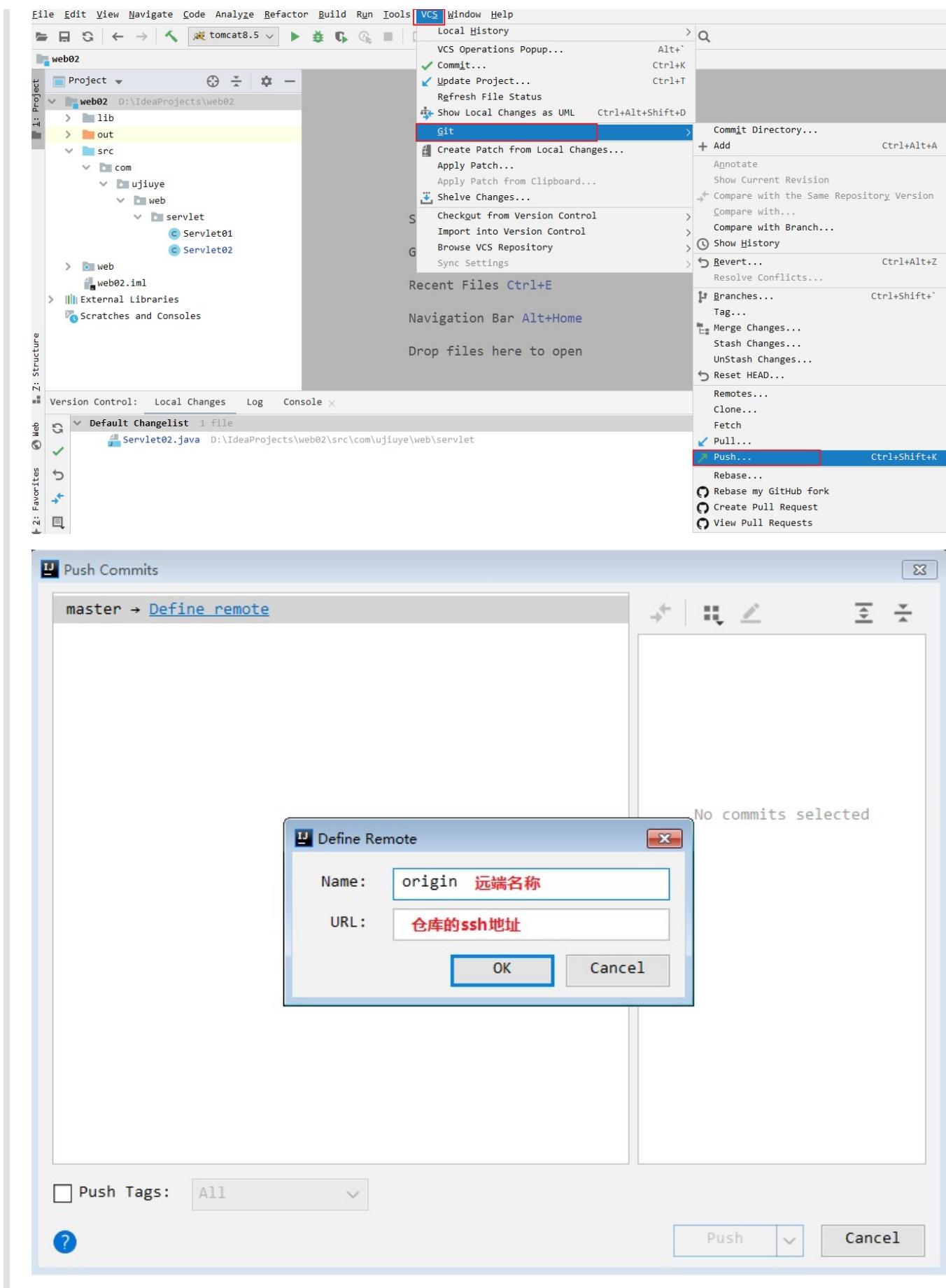
④ 配置忽略文件



⑤ 提交到本地仓库

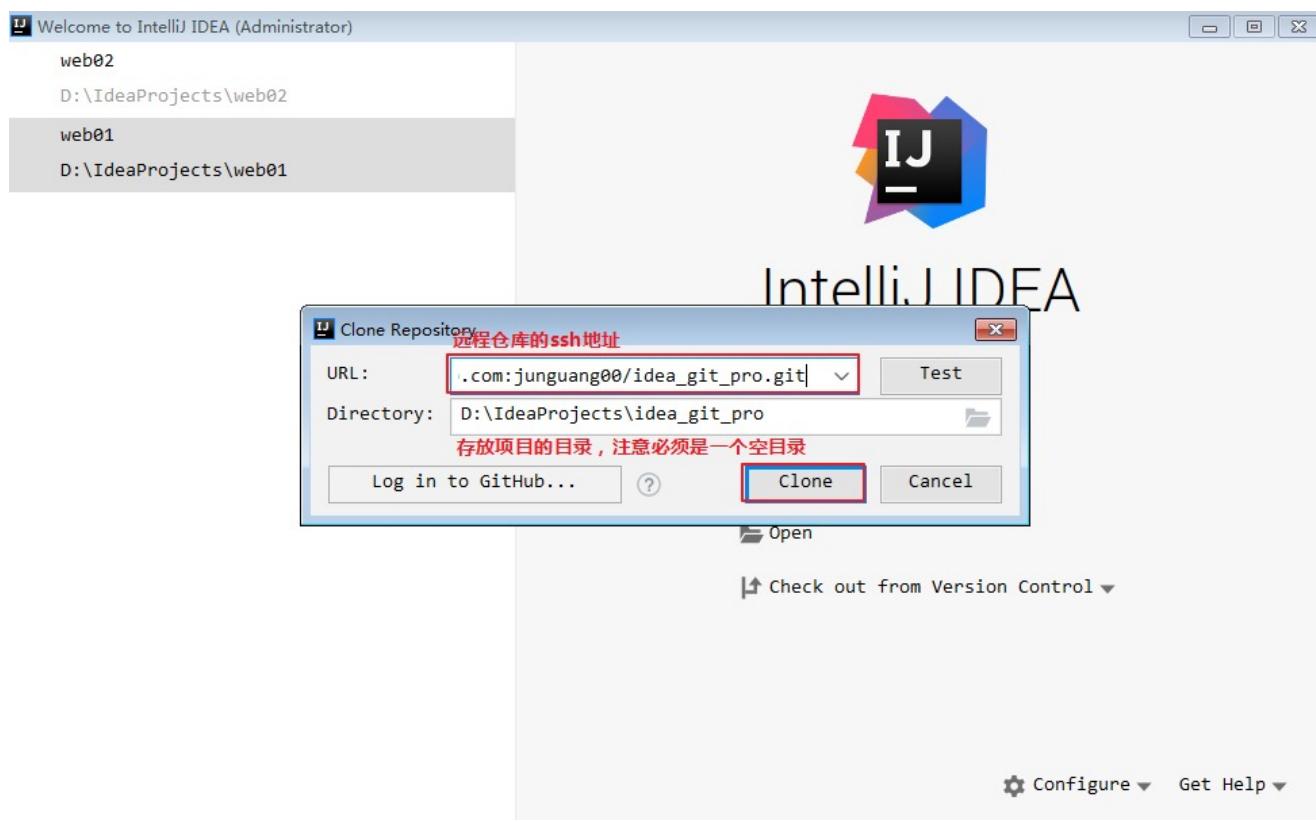
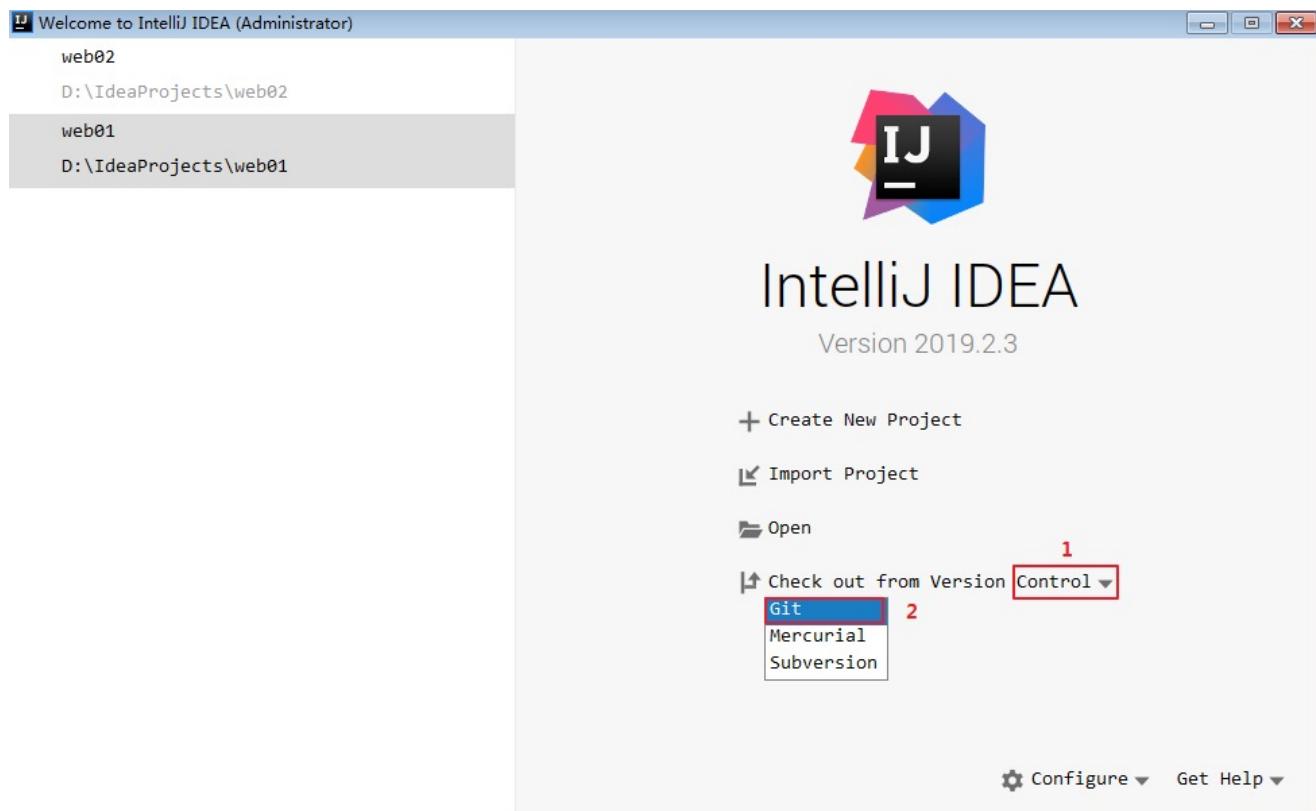


⑥ 推送到远程仓库



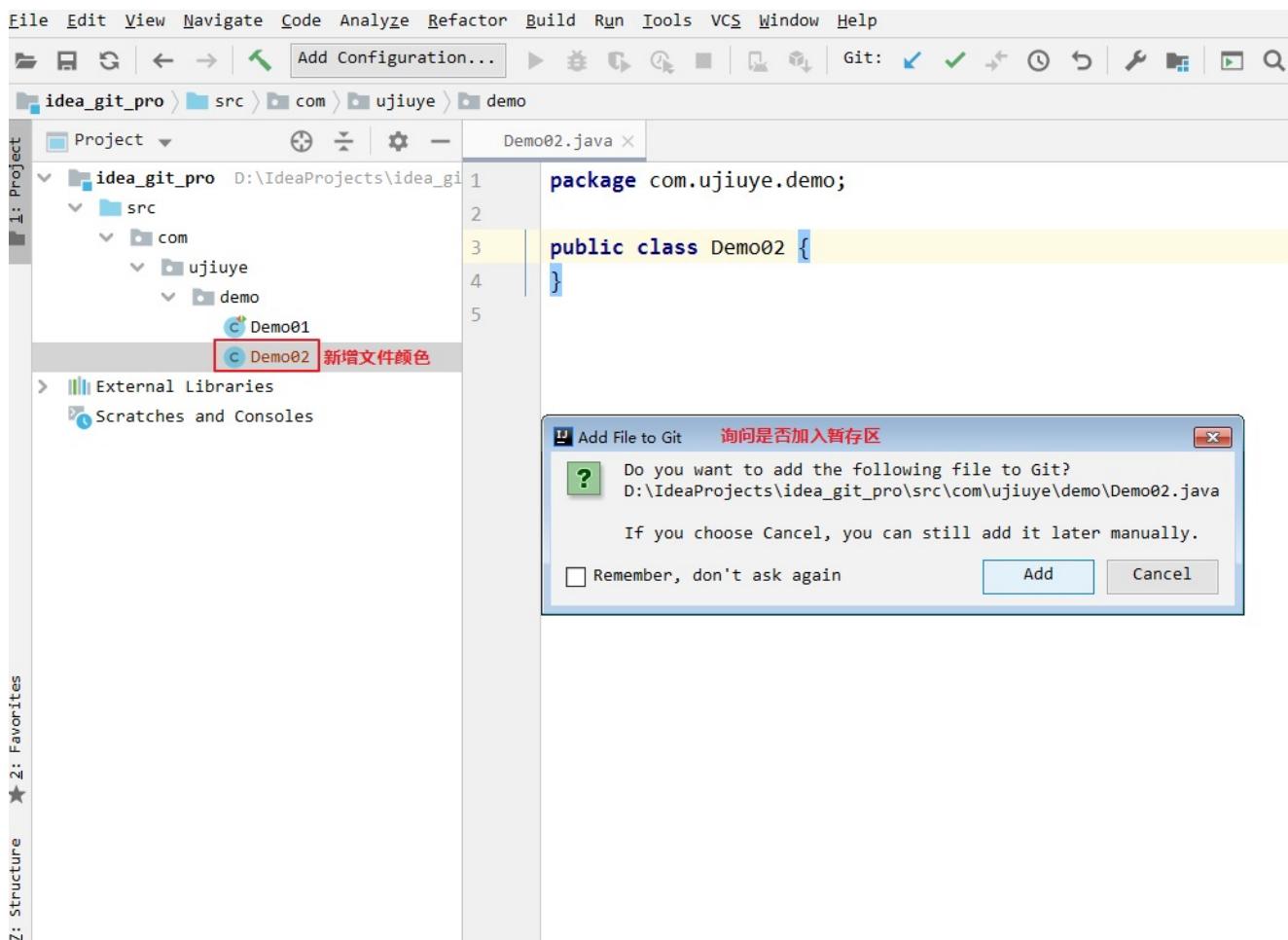
8.2.2 克隆远程仓库到本地【开发人员】

① 从远程仓库克隆



8.2.3 Idea中Git的常见操作【开发人员重点】

① 新增文件：新文件状态红色，未进入暂存区



加入git之后，红色变绿色，已经进入暂存区

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Add Configuration...
idea_git_pro > src > com > ujiuye > demo > Demo02
Project Demo01.java Demo02.java
1 package com.ujiuye.demo;
2
3 public class Demo02 {
4
5 }
```

1: Project

idea_git_pro D:\IdeaProjects\idea_git
src
com
ujiuye
demo
Demo01
Demo02 进入暂存区之后

External Libraries
Scratches and Consoles

② 编辑文件：修改文件 变成蓝色

正常编辑的文件默认放在暂存区，不需要再添加到暂存区

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Add Configuration...
idea_git_pro > src > com > ujiuye > demo > Demo02
Project Demo01.java Demo02.java
1 package com.ujiuye.demo;
2
3 import org.junit.Test;
4
5 public class Demo02 {
6     @Test
7     public void test1(){
8
9     }
10 }
```

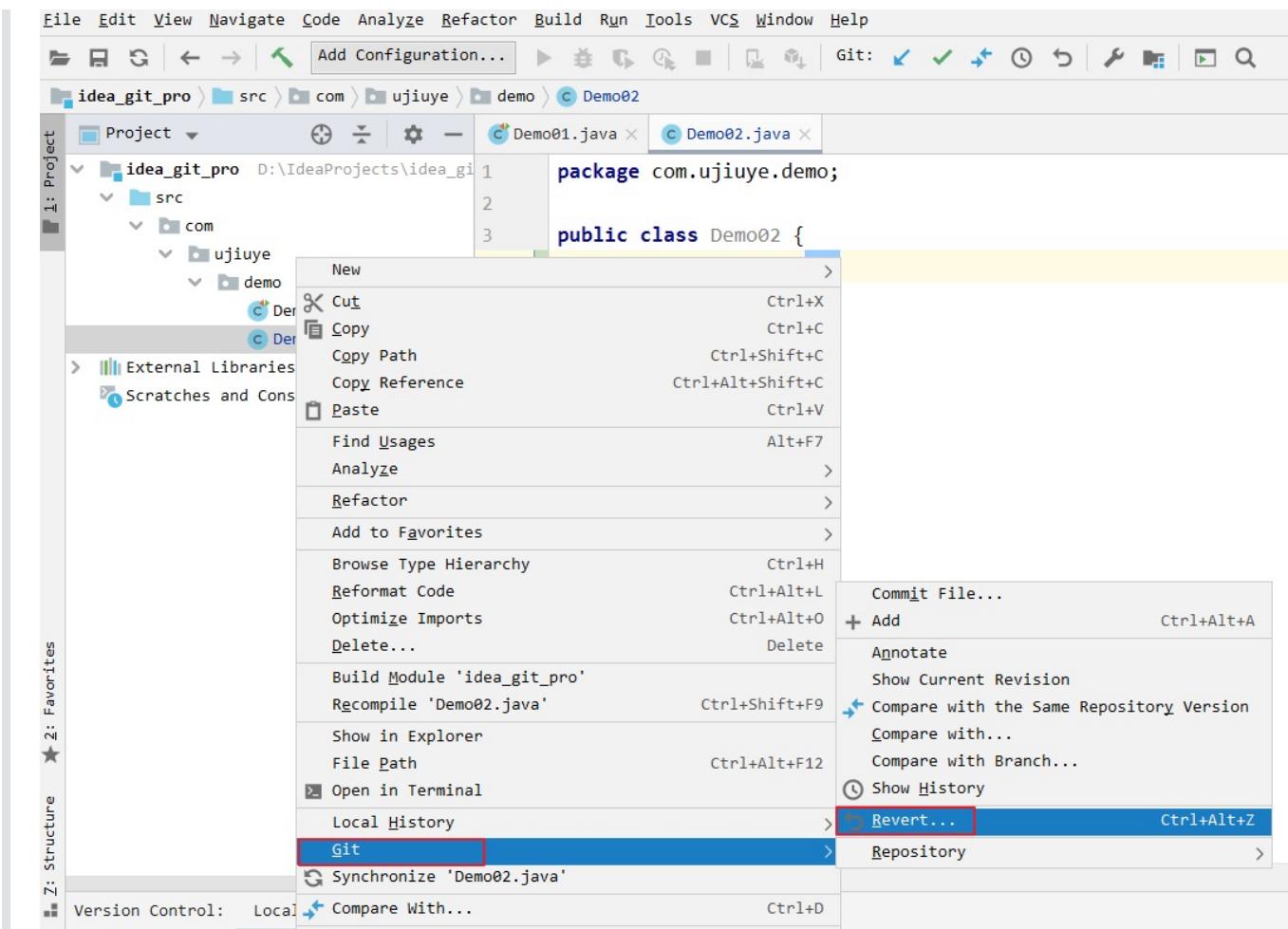
1: Project

idea_git_pro D:\IdeaProjects\idea_git
src
com
ujiuye
demo
Demo01
Demo02 修改文件

External Libraries
Scratches and Consoles

③ 重置文件到修改前

比如修订了某一文件，需要重置到修改文件之前的状态，选择文件，右键菜单：选择Git--->Revert
重置后，文件颜色自动消失，说明已重置到修改之前的状态。



④ 提交当前文件

⑤ 本地仓库推送到远程仓库

操作步骤：

1. 推送前一定要先拉取远程仓库对应分支
2. 如果有冲突，先解决冲突，并提交到本地仓库
3. 推送当前分支到远程仓库

冲突：如果A、B两个分支都在操作同一文件。当A分支提交远程仓库之后，B分支再提交的话会产生冲突

解决：先拉取，解决完冲突，再push

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Add Configuration... Git:

idea_git_pro > src > com > ujiuye > demo > Demo02 拉取

Project Demo01.java Demo02.java

1 package com.ujiuye.demo;

2

3 public class Demo02 {

4 public void test1(){

5 }

6 }

7

1: Project

idea_git_pro D:\IdeaProjects\idea_gi 1

src

com

ujiuye

demo

Demo01

Demo02

External Libraries

Scratches and Consoles

The screenshot shows the IntelliJ IDEA interface with a Java file named Demo02.java open. The code contains a public class Demo02 with a test1() method that prints "2222". A 'Push Rejected' dialog box is displayed in the foreground, stating: 'Push of current branch master was rejected. Remote changes need to be merged before pushing.' It includes a checkbox for remembering the update method choice and buttons for Cancel, Merge (which is highlighted with a red border), and Rebase.

```
package com.ijiuye.demo;

public class Demo02 {
    public void test1(){
        System.out.println("2222");
    }
}
```

Push Rejected

Push of current branch master was rejected.
Remote changes need to be merged before pushing.

Remember the update method choice and silently update in future
(you may change this in the Settings)

Cancel Merge Rebase

Conflicts

Merge conflicts detected. Resolve them before continuing update.

Name	Yours (mas...)	Theirs (ori...	
Demo02.java	D:\IdeaProjects\Modified	Modified	Accept Yours Accept Theirs Merge...

Group files by directory

Close

The screenshot shows a merge conflict in IntelliJ IDEA for a Java file named Demo02.java. The interface is divided into three main sections: 'Your version, branch master' (left), 'Result' (center), and 'Changes from branch origin/master, rev...' (right). The 'Result' section contains the merged code:

```
package com.ujkiye.demo;
public class Demo02 {
    public void test1(){
        System.out.println();
    }
}
```

The 'Your version' section (A分支) has changes 1 through 8. The 'Changes from branch' section (B分支) has changes 1 through 8. A conflict is indicated at line 4 of the 'Result' section, where the code 'System.out.println()' is present in both branches. The 'Accept Left' and 'Accept Right' buttons are at the bottom left, and 'Apply' and 'Abort' buttons are at the bottom right.

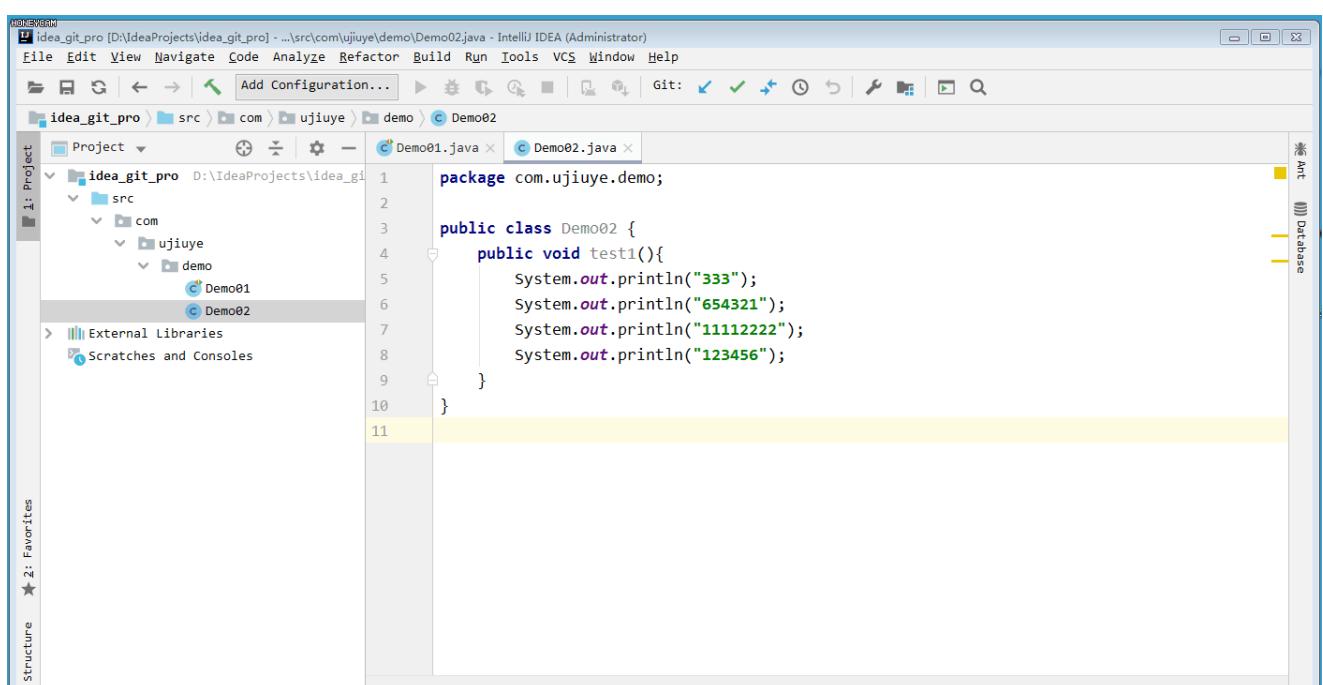
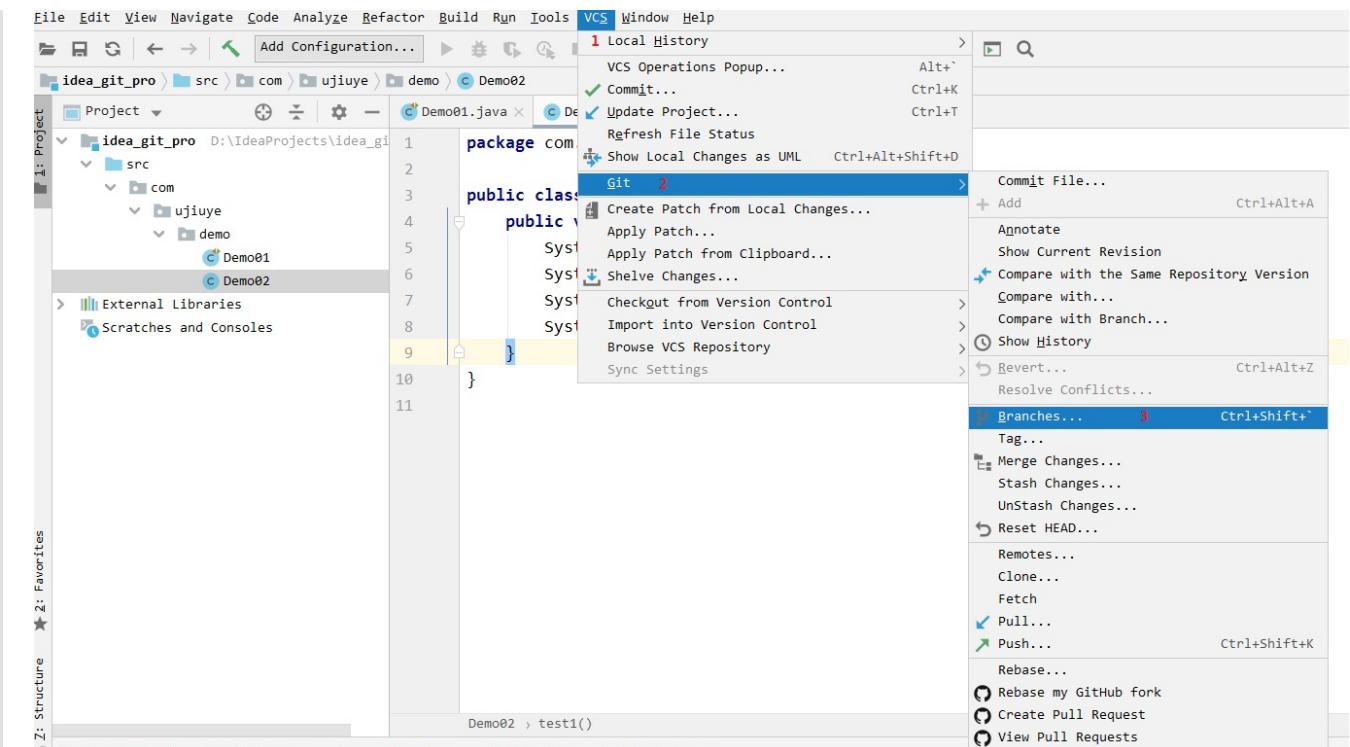
再推送到远程仓库

8.2.3 分支操作

操作步骤：

1. 创建分支
2. 切换分支执行操作
3. 执行合并操作，master合并dev
4. 同步远程仓库

① 创建分支



② 切换分支执行操作

The screenshot shows the IntelliJ IDEA interface with a Java file named Demo02.java open. A context menu is displayed over the line of code where the variable 'System.out.println("dev1");' is located. The menu is titled 'Git Branches' and includes options like '+ New Branch', 'Checkout Tag or Revision...', 'Local Branches', 'master', 'origin/master', 'dev' (selected), 'Remote Branches', and 'origin/master'. Other options include 'Checkout As...', 'Checkout and Rebase onto Current', 'Compare with Current', 'Show Diff with Working Tree', 'Rebase Current onto Selected', 'Merge into Current', 'Rename...', and 'Delete'.

③ 执行合并操作(将dev分支合并到master分支)

The screenshot shows the IntelliJ IDEA interface after the merge operation. The Java file Demo02.java now contains additional print statements from the merged code. The 'Event Log' panel at the bottom left shows a message: '11:12 Workspace associated with branch "dev" has been restored'. The bottom navigation bar includes tabs for 'TODO', 'Terminal', and 'Version Control'.

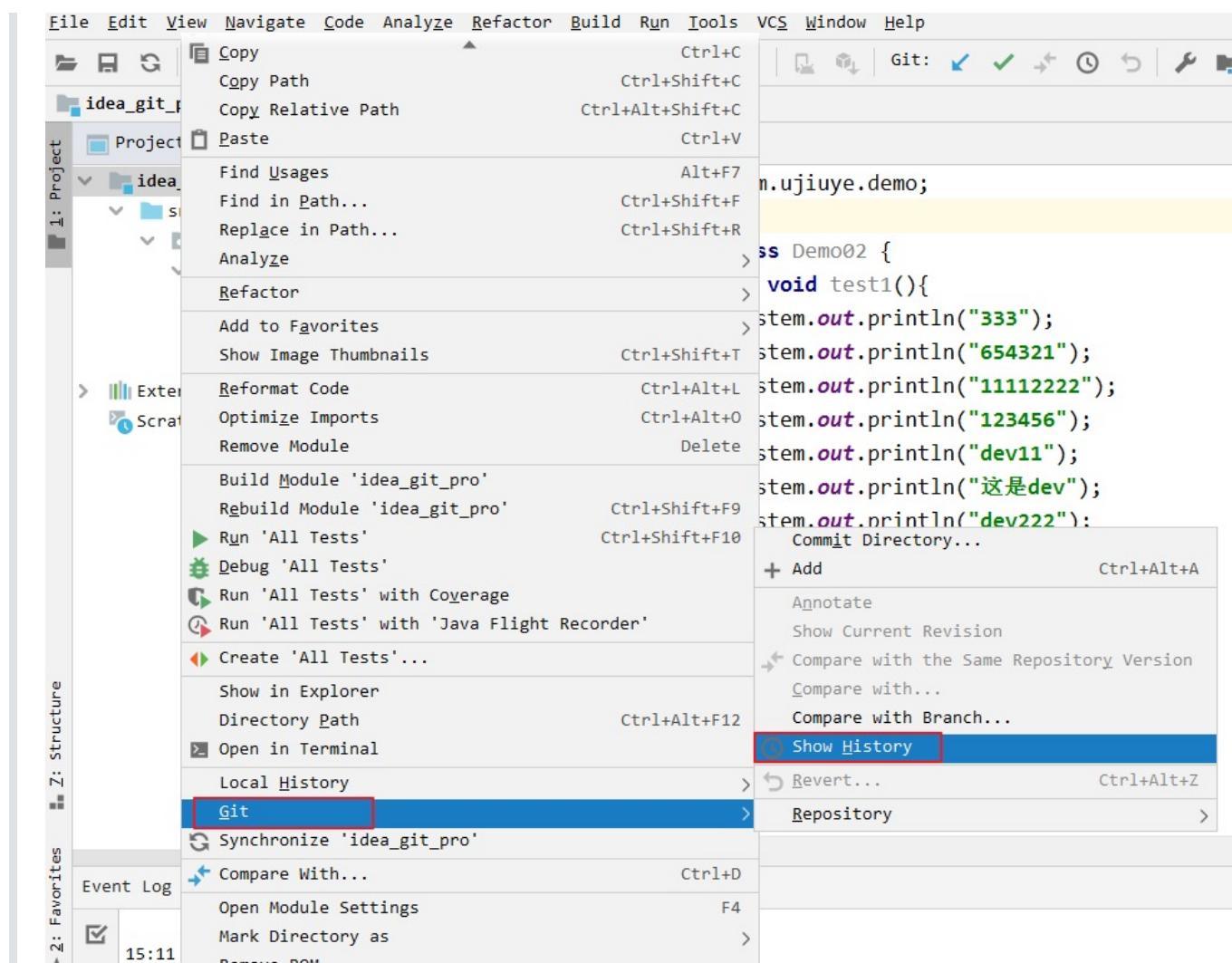
The screenshot shows the IntelliJ IDEA interface with a Java file named Demo02.java open. A context menu is displayed over the line of code where the variable System.out is used. The menu is specifically for the 'dev' branch, which is highlighted in blue. The menu options include:

- Checkout
- Checkout As...
- Checkout and Rebase onto Current
- Compare with Current
- Show Diff with Working Tree
- Rebase Current onto Selected
- Merge into Current (this option is selected and highlighted in red)
- 将dev分支合并到master (Chinese translation of the merge option)
- Rename...
- Delete

④ 同步远程仓库

The screenshot shows the IntelliJ IDEA interface with the 'Merge into Current' operation in progress. The status bar at the bottom indicates 'Merging dev into master'. The code editor shows the 'test1()' method with several println statements. The 'Event Log' panel at the bottom left shows the message 'Checked out master (today 11:19)'. The 'Favorites' section also has a checked entry for 'Checked out master (show_balloon)'.

8.2.4 查看提交历史提交



Version Control: Local Changes Log Log: 2 × Console ×

Branch: HEAD User: All Date: All > 🔍 ⚡ ⚡

操作	分支	提交者	日期
添加了注释	origin & master	junwei liu	2021/1/26 15:12
啦啦	dev	junwei liu	2021/1/25 16:51
合并提交		junwei liu	2021/1/25 16:45
合并分支		junwei liu	2021/1/25 16:29
第三次修改修改demo02 pro1		junwei liu	2021/1/25 15:47
第三次修改demo2 pro2		junwei liu	2021/1/25 15:45
Merge remote-tracking branch 'ori		junwei liu	2021/1/25 15:35
又一次修改demo2 pro2		junwei liu	2021/1/25 15:35
又一次修改修改demo02 pro1		junwei liu	2021/1/25 15:34
Merge remote-tracking branch 'or		junwei liu	2021/1/25 15:31
修改demo02 pro1		junwei liu	2021/1/25 15:21
修改demo2 pro2		junwei liu	2021/1/25 15:20
修改demo02		junwei liu	2021/1/25 15:02
新增demo02		junwei liu	2021/1/25 14:55
删除demo02		junwei liu	2021/1/25 14:50
改变demo01		junwei liu	2021/1/25 14:36
创建了demo01		junwei liu	2021/1/25 11:47

idea_git_pro
src\com\u...
Demo02

添加了注释

84cacc78
junwei liu
<215780883@qq.com> on