

# Github

## 一、基本概念

### 1.1 简介

Github是一个代码托管平台，可以在这个平台协同开发项目，也拥有数不尽的资源。国内的平台gitee。

官网：<https://Github.com/>

### 1.2 相关介绍

**仓库(Repository):** 用于存放项目代码，每个项目对应一个项目，多个开源项目则有多个仓库

**收藏(Star):** 收藏项目，方便下次查看(如果在Github上，发现某个项目被很多人收藏，这个一定是好项目)

**复制克隆项目(Fork):** 复制别人的项目到我们自己的仓库

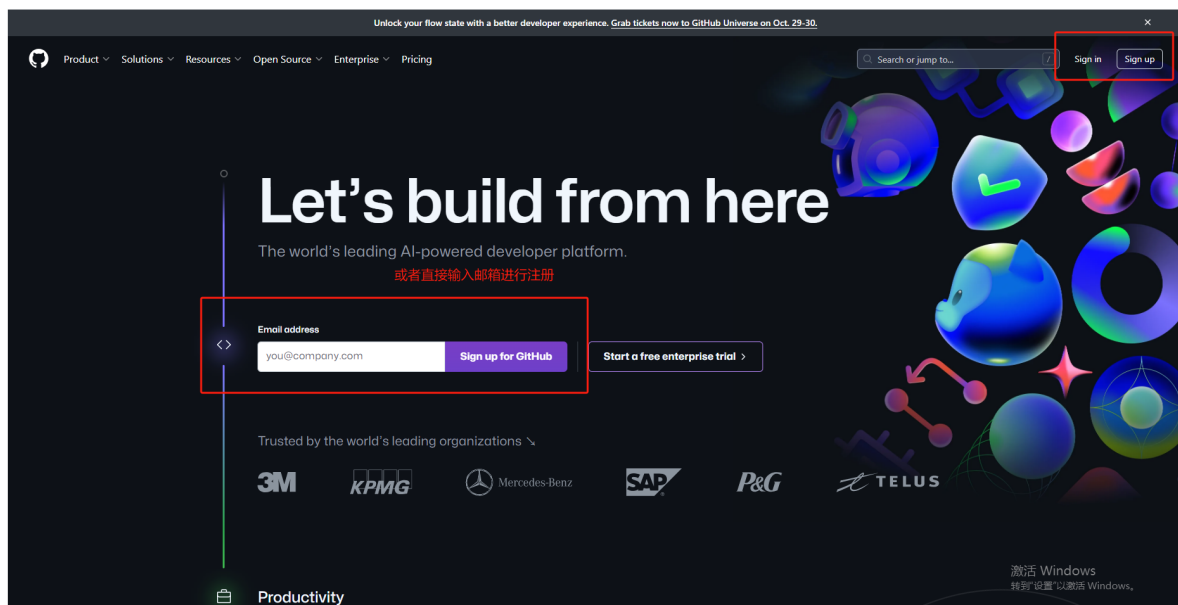
**观察(watch):** 关注某个项目，有更新会及时通知你

**事务卡片(Issue):**发现bug，但是目前没有成型代码，需要讨论的时候使用

## 二、Github使用

### 2.1 注册登入

选择Sign up进行注册（如果有账户直接sign in）




Welcome to GitHub!  
Let's begin the adventure

Enter your email\*

✓ 1942044422@qq.com

Create a password\*

→ ..... 

Continue

---

Password is strong

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter.

Welcome to GitHub!  
Let's begin the adventure

Enter your email\*

✓ 1942044422@qq.com 有效邮箱

Create a password\*

✓ ..... 密码 (8~16) 至少一位数字和小写字母

Enter a username\*

✓ bubbleflaw2 用户名

Email preferences

☐ Receive occasional product updates and announcements.

Continue

验证过后进入选择版本:

Free

The basics of GitHub for every developer

\$0

per month

免费版

Includes:

- ∞ Unlimited public and private repositories
- ✓ 3 collaborators for private repositories
- ✓ Issues and bug tracking
- ✓ Project management

Pro

Pro tools for developers with advanced requirements

\$7

per month

付费版

(view in CNY)

Includes:

- ∞ Unlimited public and private repositories
- ∞ Unlimited collaborators
- ✓ Issues and bug tracking
- ✓ Project management
- ✓ Advanced tools and insights

Are you a [student](#)? Get access to the best developer tools for free with the [GitHub Student Developer Pack](#).

☐ Help me set up an organization next  
Organizations are separate from personal accounts and are best suited for businesses who need to manage perm  
[Learn more about organizations](#)

☒ Send me updates on GitHub news, offers, and events  
Unsubscribe anytime in your email preferences. [Learn more](#)

Continue

邮箱验证完成选择操作:

Learn Git and GitHub without any code!

Using the Hello World guide, you'll [create a repository](#), start a branch, write comments, and open a pull request.

阅读用户指南

开始新项目

Read the guide

Start a project

可以点开profile修改个人简历

## 2.2 创建Github仓库

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* Bubbleflaw / Repository name \* github\_demo → 仓库名称  
✔ github\_demo is available.

Great repository names are short and memorable. Need inspiration? How about [expert-couscous](#) ?

Description (optional)  
这是一个用于测试的仓库 → 仓库介绍

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit. → 默认是公用，大家都可以看到

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

Initialize this repository with:  
☒ **Add a README file** → 初始化仓库的信息文件，建议勾选。  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore  
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license  
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set main as the default branch. Change the default name in your [settings](#).

! You are creating a public repository in your personal account.

Create repository

→ 点击创建仓库

## 相关介绍

github\_demo

Public

仓库名称 公有

main

1 Branch

0 Tags

当前分支是main 共有1个分支

1e73cd4 · 15 minutes ago

1 Commit

Initial commit

初次提交

README.md

Initial commit

15 minutes ago

提交时间

README

github\_demo

这是一个用于测试的仓库

Pin

Unwatch 1

Fork 0

Star 0

About

这是一个用于测试的仓库

Readme 描述

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

## 2.3 创建issues

这个是为了发现bug的时候和笔者讨论用的，问题得到解决的时候，笔者可以关闭issues

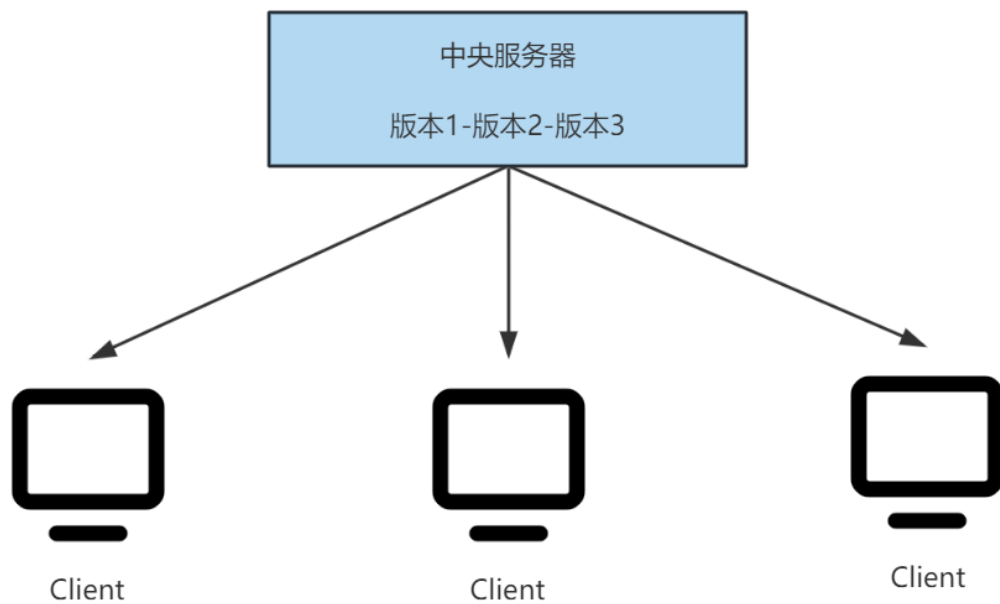
# Git

## 一、基本概念

## 1.1 什么是版本控制

定义：版本控制是一种记录一个或者若干个文件内容变化，以便将来查阅特定版本情况的系统

### 1. 集中化版本控制



所有文件都保留在中央服务器上，每个人的电脑都只保存一个副本，当你要修改文件的时候，首先去中央服务器下载最新的版本，修改完再上传回中央服务器。

#### 优点：

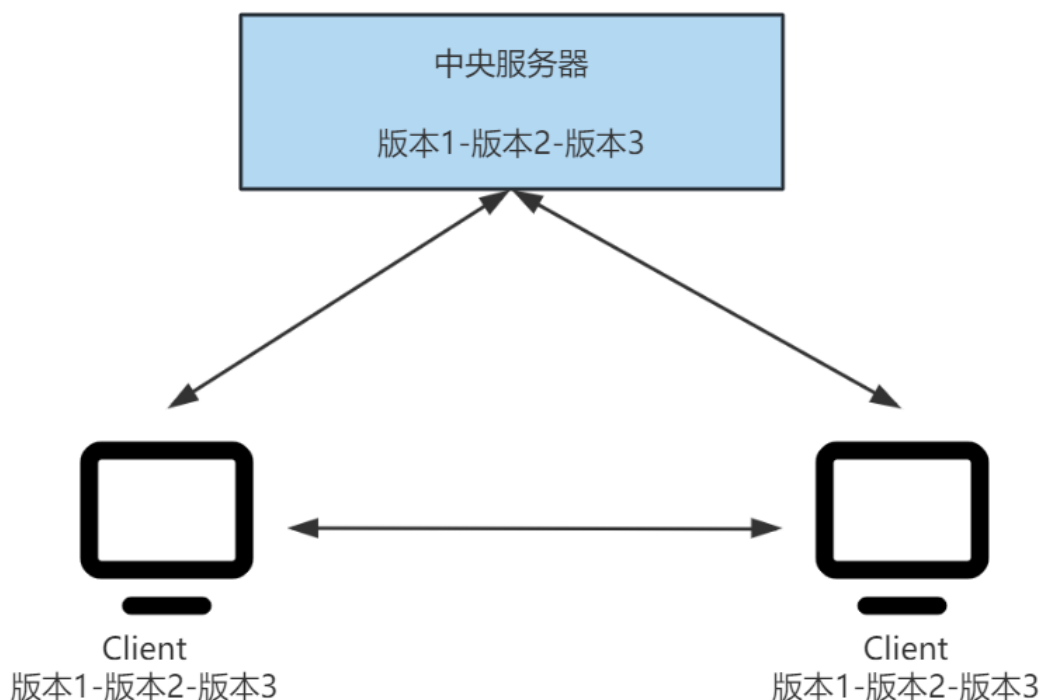
易维护、易掌握开发者权限

（这种做法带来了许多好处，每个人都可以在一定程度上看到项目中的其他人正在做些什么，而管理员可以轻松掌控每个开发者的权限，并且管理一个集中化的版本控制系统，要远比在各个客户端上维护本地数据库来得轻松容易）

#### 缺点：

中央服务器的单点故障时候无法协同工作

## 2. 分布式版本控制



像Git这种分布式版本控制工具，客户端提取的不是最新版本的文件快照，而是把代码仓库完整的镜像下来（本地库），这样任何一处协同工作用的文件发生故障，事后都可以用其他客户端的本地仓库进行修复，因为每个客户端的每一次文件提取操作，实际上都是一次对整个文件仓库的完整备份。

分布式的版本控制系统出现之后，解决了集中式版本控制系统的缺陷

- 服务器断网的情况下，也可以进行开发（因为版本控制是在本地进行的，只有push、pull时候需要联网）
- 每个客户端保存的也都是整个完整的项目（包含历史记录，更加安全）

## 1.2 Git相关介绍

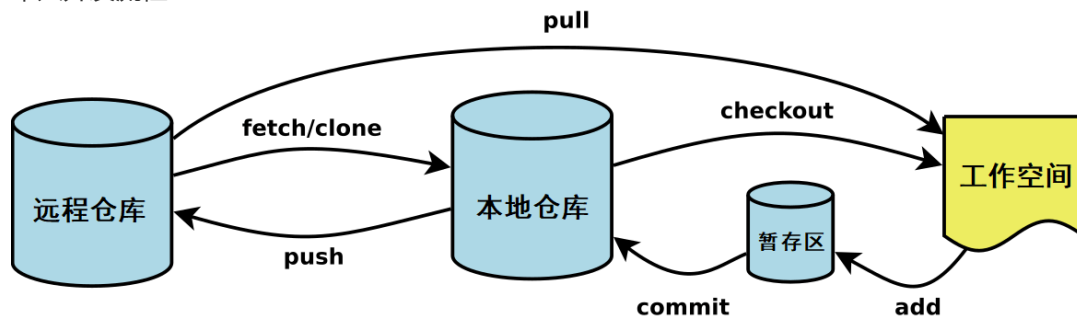
定义：Git是一个免费，开源的版本控制软件（SVN、CVS、Mercuria）

### 1.2.1 Git的四大工作区域

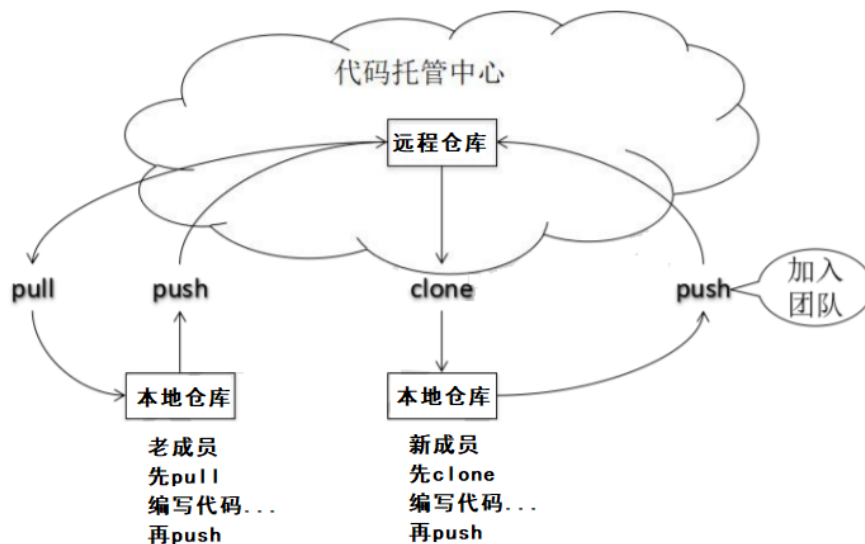
1. **工作目录 (Working Directory)**：你电脑本地看到的文件和目录。也就是.git所在的目录。
2. **暂存区 (Staging Area)**：暂存区，一般存放在.git目录下，即.git/index,它又叫待提交更新区，用于临时存放你未提交的改动。
3. **本地仓库 (Local Repository)** 本地仓库，你执行git clone 地址，就是把远程仓库克隆到本地仓库。它是一个存放在本地的版本库。
4. **远程仓库 (Remote Repository)**：远程仓库，就是类似github，码云等网站所提供的仓库，可以理解为远程数据交换的仓库。

### 1.2.2 Git的开发流程

## 1. 单人开发流程



## 2. 多人开发流程



Git的多人工作流程一般是这样的：

- 从远程仓库拉取文件代码回来，在工作目录中修改文件。 **pull**
- 将需要进行版本管理的文件放入暂存区。 **add**
- 将暂存区的文件提交到本地仓库。 **commit**
- 如果需要，可以将本地仓库的文件推送到远程仓库。 **push**

### 1.3 Git的文件状态

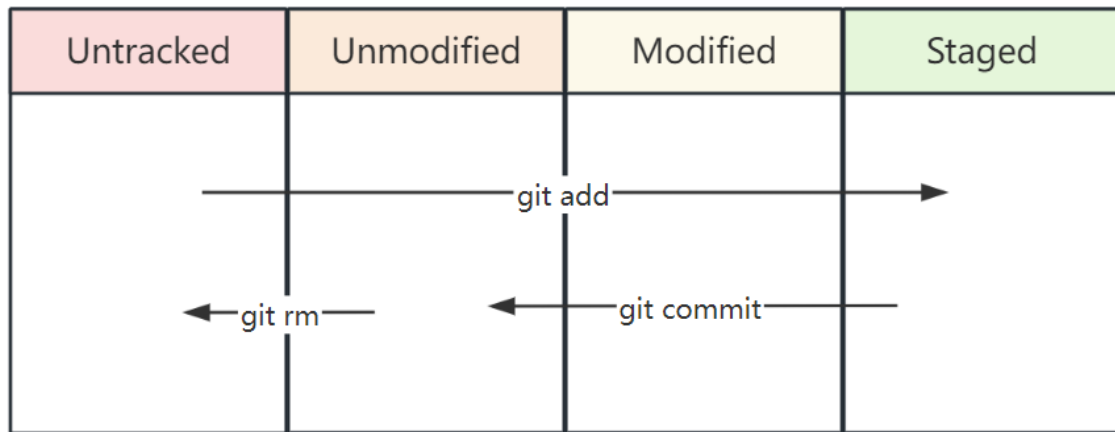
根据一个文件是否已加入版本控制，可以把文件状态分为：

Untracked (未跟踪)：新创建，没有被git管理起来的文件。

Unmodified (未修改)：已经被git管理起来的文件，但文件的内容还有被修改过。

Modified (已修改) : git管理起来已经修改了但还没有添加到暂存区的文件。

Staged (已暂存) : 修改后添加到暂存区 (git add )



## 1.4 Git安装下载

官方地址: <https://git-scm.com/>

下载地址: <https://git-scm.com/downloads>

差不多一直默认, 详细安装教程可参考: [https://blog.csdn.net/weixin\\_73670121/article/details/141176714](https://blog.csdn.net/weixin_73670121/article/details/141176714)

任务:

- 注册登入github、gitee、新建一个远程仓库
- 安装git

## 二、Git的初始化及仓库创建和操作

### 2.1 基本信息设置

```
1 //设置用户名
2 git config --global user.name "github上的用户名"
3 // 设置邮箱
4 git config --global user.email "github上的邮箱"
5 //查看所有的配置信息
6 git config --global --list
7
```

### 2.2 初始化Git仓库

```
1 cd: 跳转到目录
2 pwd: 查看当前在哪个目录下
3 ls /11: 查看当前目录下的文件
4 mkdir: 创建目录
5 git init (目录): 初始化git仓库
6 git status: 查看状态
7 ls -a/a1 :显示所有目录下文件 (包括隐藏文件)
```



## 2.3 添加/修改文件

```
1 #创建一个文件demo.txt
2
3 $ echo "这是第一个测试的文件" > demo.txt
4
5 #使用 'git add 文件名'命令将文件从工作区存到暂存区
6
7 $ git add readme.txt
8
9 #使用'git commit -m "备注信息"'将文件从暂存区提交到git仓库
10
11 $ git commit-m "write a readme file")
```

## 2.4 查看版本

```
1 #git log 查看历史记录
2
3 #git log --oneline 历史记录的缩减版
4
5 #git ls-files 查询暂存区文件
6
7 #vi 命令可以修改文件，按i进入插入模式，:wq保存退出
```

## 2.5 版本回退

git reset 后面可以有三种参数soft hard mixed

区别：

	工作区	暂存区
git reset --soft	✓	✓
git reset --hard	✗	✗
git reset --mixed	✓	✗

```
1 git reset --hard 版本号,写前面几位就可以了
2
3 git reset --hard HEAD^ 往回退一个 版本
4
5 用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个
  ^比较容易数不过来，所以写成HEAD~100
```

```
user@DESKTOP-9D8KAM4 MINGW64 /d/git (master)
$ git log
commit 45410a280ce66a18e46dbd5b691d80c96907f77c (HEAD -> master)
Author: Bubbleflaw <huangwenqing@oamicnet.com>
Date: Thu Sep 26 16:03:24 2024 +0800

    这是第二次提交

commit 532ac41775dfa6602f6f736c67c3b47d6b8b9c9e
Author: Bubbleflaw <huangwenqing@oamicnet.com>
Date: Thu Sep 26 16:00:27 2024 +0800

    这是第一次提交
```

## 2.6 删除文件

```
1 #先工作区删除，再git add删除暂存区
2 $ rm demo.txt
3 # git ls-files 查看暂存区文件
4 $ git add .
5
6 #git rm --cached xxx 工作区和暂存区都删除
7 $ git rm --cached demo.txt
8
9 #最后commit更新到本地仓库中
```

## 2.7 gitignore忽略文件

.gitignore是一个隐藏文件，一般我们默认会把它建立在仓库的根目录(也可以是仓库下的任意目录)

### 应该忽略哪些文件

- 系统或者软件自动生成的文件
- 编译产生的中间文件和结果文件
- 运行时生成日志文件、缓存文件、临时文件
- 涉及身份、密码、口令、秘钥等敏感信息文件

### 使用

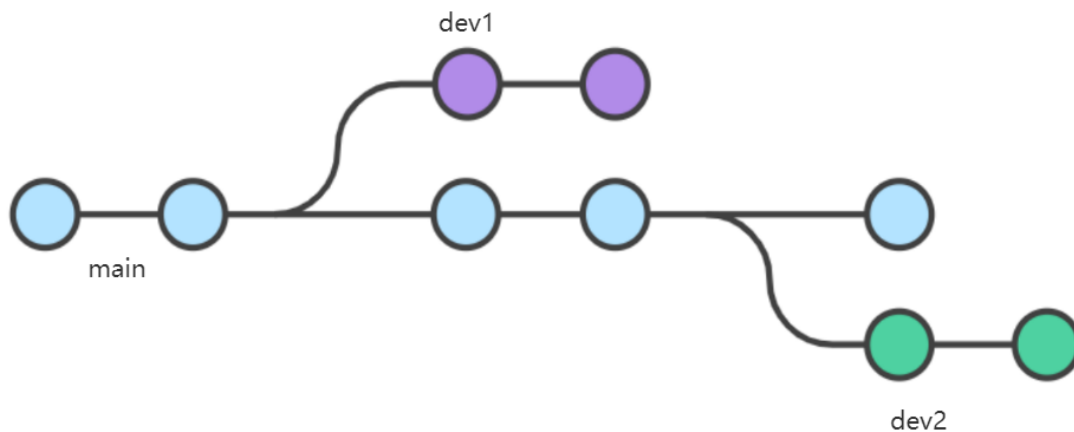
```
1 $ echo demo.txt > .gitignore #创建.gitignore文件，并且将demo.txt设置程忽略文件
2 #当某文件已经在暂存区的时候，再往.gitignore文件中添加，不生效!!! 需要先从暂存区删除才可以
```

### 规则

- **.a**: 忽略所有以.a为后缀的文件,
- **!ilb.a** : 不忽略文件ilb.a
- **/TODO** : 只忽略此目录下TODO文件,子目录的TODO不被忽略
- **build/** : 忽略build目录下的所有文件:
- **doc/\*.txt**: 只忽略doc/下所有的txt文件,但是不忽略doc/subdir/下的txt文件

提示: github上有许多gitignore的模板: <https://github.com/github/gitignore>

## 三、分支管理



### 3.1 创建和合并分支

```

1  git checkout -b <分支名称> :创建并切换分支
2
3  git branch : 查看有哪些分支, 当前分支前面有个*号
4
5  git branch <分支名称> : 创建分支
6
7  git checkout <分支名称> : 切换分支
8
9  git merge <分支名称> : 合并分支
10
11 git branch -d <分支名称> : 删除分支

```

### 3.2 冲突的发生和解决

git checkout -b feature 创建并切换分支

修改readme.txt文件, 添加, 提交

切换到master分支

修改readme.txt文件, 添加, 提交

git merge feature 合并分支, 这个时候会发生冲突

Git用 <<<<<<, =====, >>>>>> 标记出不同分支的内容

```

1  <<<<<< HEAD
2
3  当前分支的代码
4
5  =====
6
7  合并过来的代码
8
9  >>>>>> 另一分支名称

```

**解决方式:** 手动修改文件, 然后提交

使用git log 查看分支合并情况

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

```
$ git log --graph --pretty=oneline --abbrev-commit
* 1da6cfe (HEAD -> master) 冲突处理后提交
* 95738d1 (feature1) create by feature
* 9ee7b96 master 提交
* fbf39da add by dev2
* 1430192 (origin/master) commit by dev
* 2db2802 add merge
```

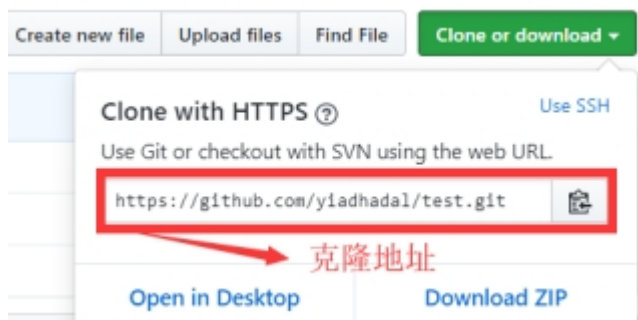
小结：当Git无法自动合并分支时，就必须首先解决冲突。步骤如下：

- 第一步：编辑文件，删除特殊符号
- 第二步：把文件手动编辑为我们希望的内容，保存退出
- 第三步：git add [文件名]
- 第四步：git commit -m “日志信息”

## 四、远程仓库（1、clone的项目怎么传到自己的远程仓库 2、本地没有新建且关联）

### 4.1 git的克隆操作

地址的由来：



```
1 $ git config --list 查看设置
2
3 # git clone 复制项目
4
5 # HTTPS拉到本地的时候需要验证用户名和密码，SSH协议这种方式在推送的时候不需要验证用户名和密码但是需要在github上系加SSH公钥的配置
6
7 $ git clone git@github.com:michaelliao/gitskills.git
8 （更加安全快捷，演示还没有配置ssh的时候）
```

（总结前面的四大工作区域，.git是工作区域 .git/index暂存区域 git clone 就是本地仓库的地方 远程就是github仓库，区域之间是怎么变化的，add commit push）

### 4.2 ssh免密登陆

ssh-keygen -t rsa 然后一路回车键（如果原本就存在ssh文件，会覆盖原本的，且不可逆）

```

$ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/74757/.ssh/id_rsa):
Created directory '/c/Users/74757/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/74757/.ssh/id_rsa.
Your public key has been saved in /c/Users/74757/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:i4bhCIYXGeBRw9x1KGLiPHSwSv7V67zJ2u3pnShK3NI 74757@DESKTOP-FJH8F1F
The key's randomart image is:
+---[RSA 3072]-----+
|. +*O. ....
|.ooO.o ..
|+* . .
|=+ . .
|+oo .. .S
|.o.oooo...
|.o+oE.
|. .*.o + .
|.ooB=*. o
+---[SHA256]-----+

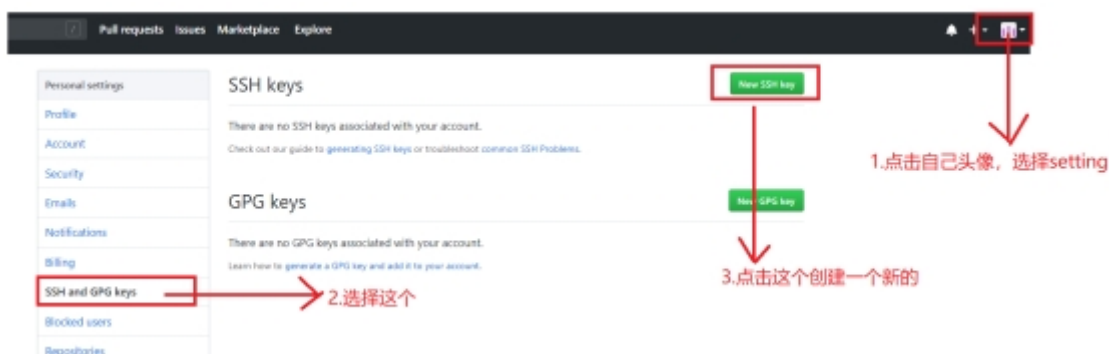
```

默认生成在当前目录底下的.ssh文件里面，会有两个文件

打开.ssh目录里面有两个文件，打开.pub（公钥）的那个，复制里面内容去github设置



github界面设置ssh免密码登陆



将id\_rsa.pub里面的内容复制到底下

Title: ssh-key1

Key: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQC7LddXSM0LaxT/VGT18ztj8fDABef9fuZYfUUVBEQHa5xPmeSgiaR8oR0szz UeKWcpzq3qNm/s8Scu84cQndbwAbOimWx+ynrOgNXnugtw+9OIV+5DksMeJe7uTecwypqJZ2Lu9DnVwe2RbjtlCewg sAzZCy+Aom87eGAZy6Zz12CtRBOCINHwXPTle9XycwCoq/IKQD68iAN0Ssx+SwqEgx1UZmu4/qOzFeMehI7pNmecN gvAnG/xpz47C7b8wlbm/WDwE1A/cuOpcriZQaUz3ws4tn4KrRjdFXDjYs4gyprVR6Y/Uv2sqfM2tM0Xhg2wtux3D8NNvC7IS RKID/9ODQ0A7RnqU9osmpiceDEgu3DP/ygOfnF2uioL6/BalEAUliq51fdeL2D46ZB/xayFXKuioXQmEys4iSyukqrGWH5 xDhiWOPi2qDnYp4MHQ2XBLq7FESLVirclH2lytz/SvEMKrg3+OB0VHYxhFA+mcuaVG/V3FmU4ypnvdRWKhs= 74757@DESKTOP-FJH8F1f

Add SSH key

## 4.3 关联远程仓库

```
1 $ git remote -v #查询当前仓库对应的远程仓库地址以及别名
2
3 $ git branch -M main #默认分支的名称为main
4
5 # 将远程的origin仓库和本地仓库关联 git remote add <远程仓库别名> <远程仓库地址>
6 $ git remote add origin git@github.com:Bubbleflaw/git_demo.git
7
8 # 远程分支和本地分支建立关联 git push -u <远程仓库别名> <远程分支名称>:<本地分支名称>
9 $ git push -u origin main:main
10 $ git push -u origin main
```

#### ...or create a new repository on the command line

```
echo "# git_demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:Bubbleflaw/git_demo.git
git push -u origin main
```

初始化

初始化本地仓库  
且关联远程仓库

#### ...or push an existing repository from the command line

```
git remote add origin git@github.com:Bubbleflaw/git_demo.git
git branch -M main
git push -u origin main
```

关联远程仓库

## 4.4 将本地仓库内容推送到远程

```
1 $ git remote add origin git@github.com:Bubbleflaw/git_demo.git
2
3 $ git push -u origin main
```

## 4.5 将远程内容拉取到本地

```
1 # git pull <远程仓库名称> <远程分支名称>:<本地分支名称>
2 # 后面的 <远程仓库名称> <远程分支名称>:<本地分支名称>可省略，默认拉去远程main分支
3 $ git pull
```

注意：pull 是拉取远程分支并合并，如何远程和本地有冲突的时候需要解决冲突，fetch 只获取远程分支的修改内容，不合并。

## 4.6 冲突的处理

- A-修改A文件内容-->提交到github
- B修改A文件内容-->提交到github-->会发生冲突
- B先拉取-->手动解决冲突-->在提交

## 4.7 Gitee和GitLab

Gitee：国内平台，都是开源的

GitLab：局域网，私有化部署

## 五、多人协助分支管理

```
1 git remove : 查看远程库信息 -v 查看详细信息
2
3 git push -u origin main: 把主分支内容推送到远程
4
5 git push -u origin dev : 把dev分支内容推荐到远程
6
7 git remote add origin 地址 : 让本地仓库和远程仓库发生关系
8
9 git fetch : 相当于是从远程获取最新到本地, 不会自动merge
10
11 git pull : 相当于是从远程获取最新到本地(合并), 会自动merge
```

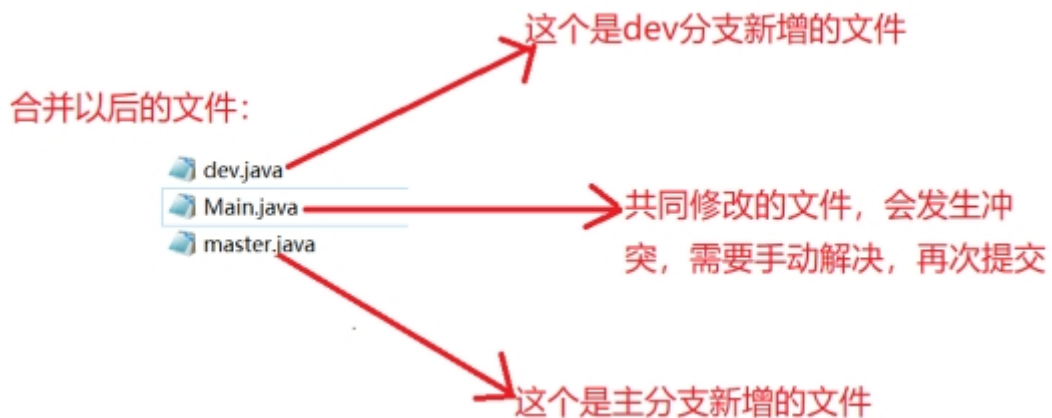
**注意: 本地新建的分支如果不推送到远程, 对其他人就是不可见的;**

本地推送分支, 使用 `git push -u <远程仓库别名> <远程分支名称>`, 如果推送失败, 先用 `git pull` 抓取远程的新提交;

建立本地分支和远程分支的关联, 使用 `git branch --set-upstream-to=origin/<远程分支名称> <本地分支名称>`

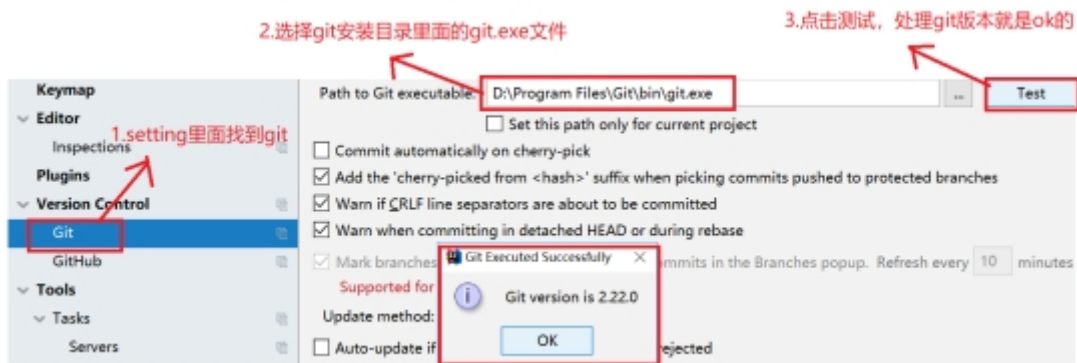
**实验步骤:**

1. `git checkout -b dev` 创建并切换到dev分支(相当于复制了一份主分支内容)
2. `git checkout main` 切换回主分支
3. 在主分支新增一个 `master.java` 文件, 然后修改一个 `Main.java` 的文件
4. `git checkout dev` 切换到dev分支
5. 在 dev 分支新增一个 `dev.java` 文件, 然后修改一个 `Main.java` 的文件
6. `git checkout master` 切换回主分支
7. `git merge dev` 合并 dev 分支

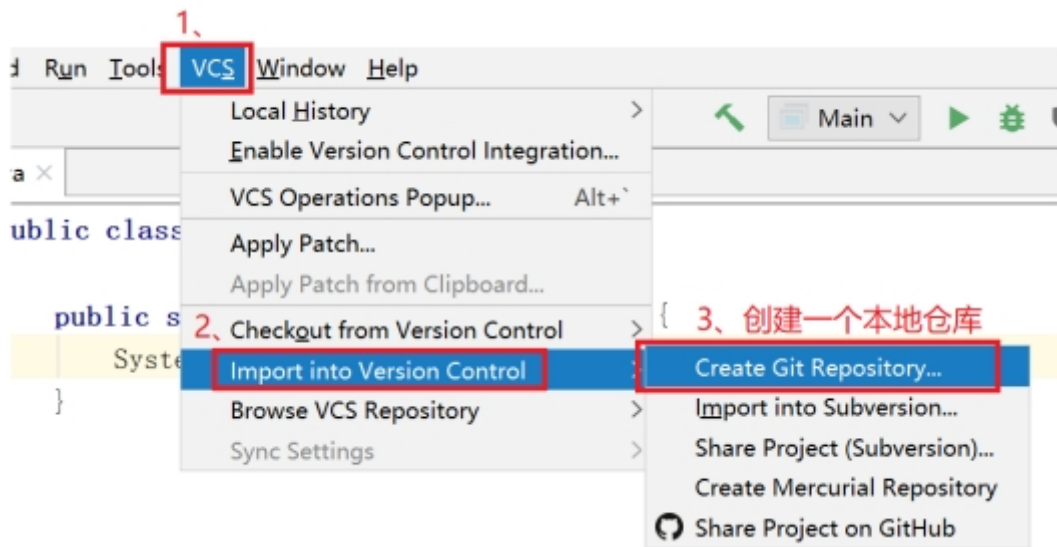


## 六、在idea里面使用git

### 6.1 idea配置git



## 6.2 将工程添加到本地仓库

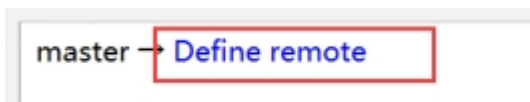
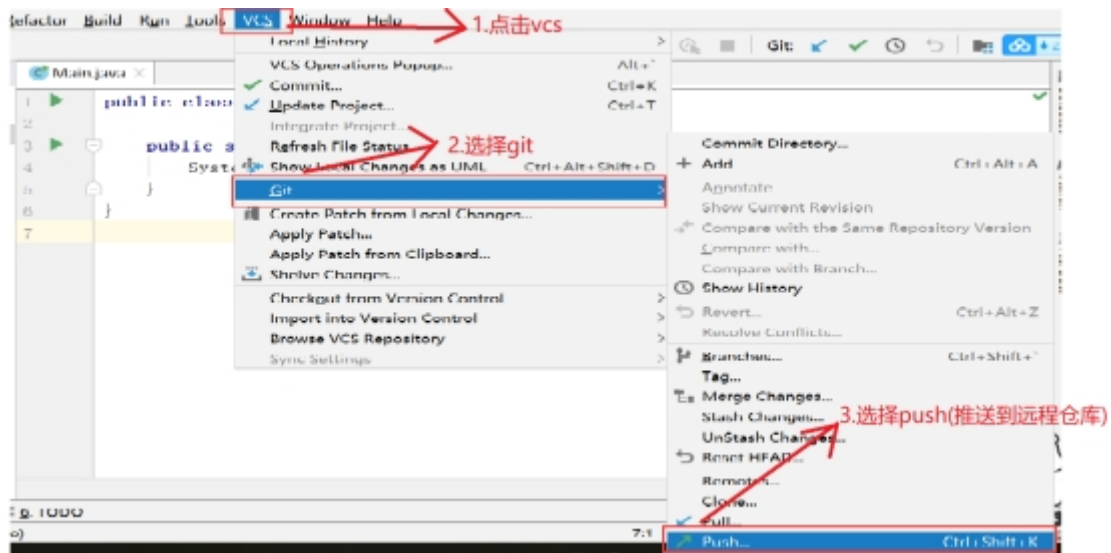


## 6.3 commot提交，文件识别

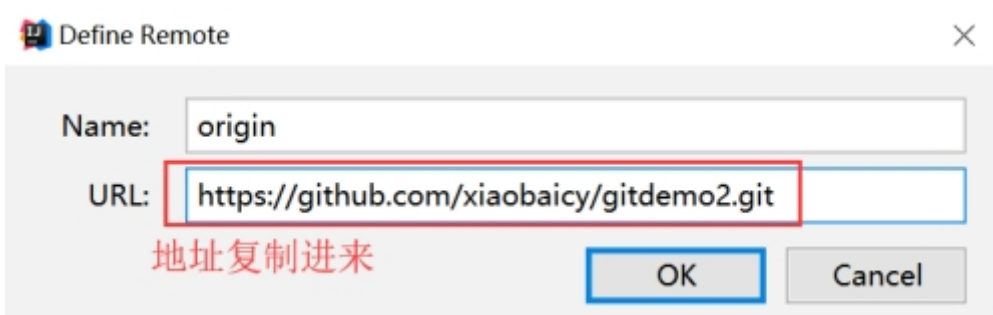


## 6.4 关联远程仓库(github)





第一次会要求输入用户名，密码



## 6.5 邀请多人协作

<https://jingyan.baidu.com/article/948f5924f43f47d80ff5f9f9.html>

## 七、习题

题目1：创建一个新的Git仓库，且关联远程仓库后提交一个简单的文件

题目2：找ruoyi-plus开源项目（或者其他开源项目），fork到自己远程仓库中，使用idea再克隆到本地进行修改后提交