

# 1.引言

在单人开发过程中，需要进行版本管理，以利于开发进度的控制。

在多人开发过程中，不仅需要版本管理，还需要进行多人协同控制。

## 2.介绍

Git是一个开源的 **分布式** 版本控制系统，用于敏捷高效地处理任何或小或大的项目。

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

官网：<https://git-scm.com/>

## 3.Git安装

下载git <https://git-scm.com/downloads>

下载git



安装，除了安装位置外，其他一直下一步即可

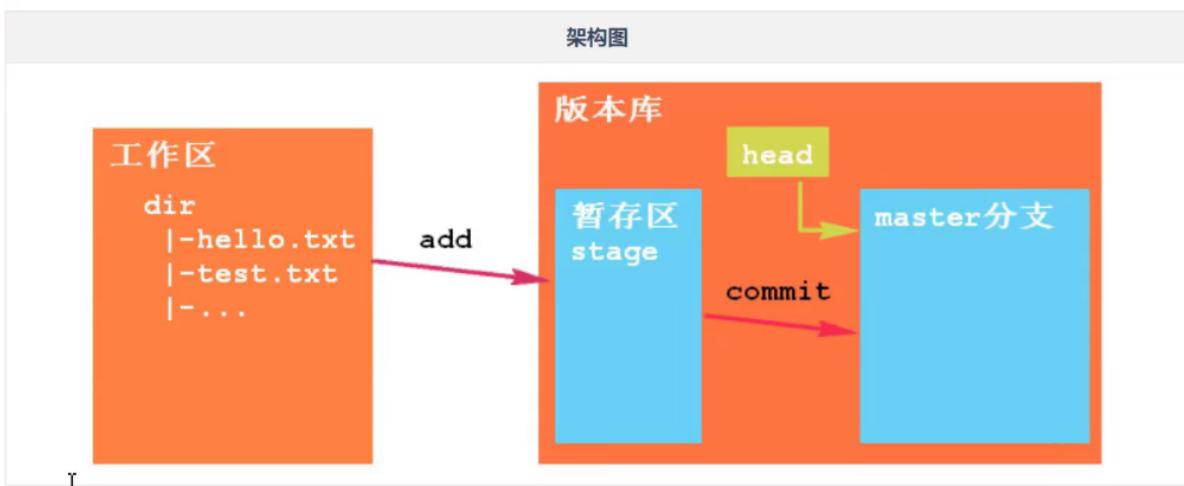
```
1 # 安装后，打开cmd，自报家门
2 # 如下信息会在提交代码时要使用，记录在你的每次提交中。以后才知道哪次提交是谁做的，“git log”可以查看
3 # 执行：
4 git config --global user.name "Your Name" #用户名
5 git config --global user.email "email@example.com" #邮箱
6 # 查看信息
7 git config -l
8 # 测试：cmd中执行，查看git版本
9 git version
```

## 4.架构

版本库：工作区中有一个隐藏目录 `.git`，这个目录不属于工作区，而是git的 **版本库**，是git管理的所有内容

暂存区：版本库中包含一个临时区域，保存下一步要提交的文件。

分支：版本库中包含若干分支，提交的文件存储在分支中



## 5.仓库

对应的就是一个 **目录**，这个目录中的所有文件被git管理起来。

以后会将一个 **项目的根目录**，作为仓库。

仓库中的每个文件的改动 都由git跟踪。

### 5.1新建仓库

选择一个目录，执行指令：`git init`

新建仓库	仓库目录
<pre>C:\Windows\System32\cmd.exe D:\repo1&gt;git init Initialized empty Git repository in D:/repo1/.git/ D:\repo1&gt;</pre> <p>D:\repo1目录中，会新建一个.git目录 .git目录即仓库对应目录</p>	<pre>电脑 &gt; D盘 (D:) &gt; repo1 名称 .git</pre> <p>仓库</p>

## 6.基本操作

### 6.1查看仓库状态

执行 `git status` 可以看到工作区中文件的状态

未记录过的文件，是 **未跟踪** 状态

电脑 > D盘 (D:) > repo1 >

名称

.git  
abc.txt  
def.txt

新建文件

D:\repo1>`git status` 执行

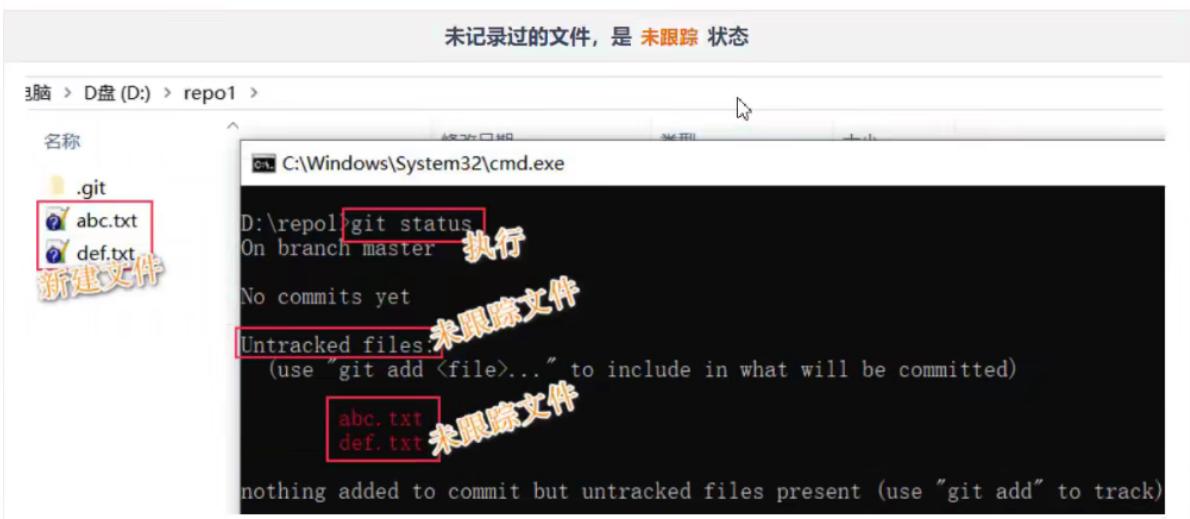
On branch master

No commits yet

Untracked files 未跟踪文件 (use "git add <file>..." to include in what will be committed)

abc.txt 未跟踪文件  
def.txt 未跟踪文件

nothing added to commit but untracked files present (use "git add" to track)



## 6.2暂存文件

执行 `git add .` 将工作区中的文件全部 **存入暂存区**

将工作区中的文件存入暂存区

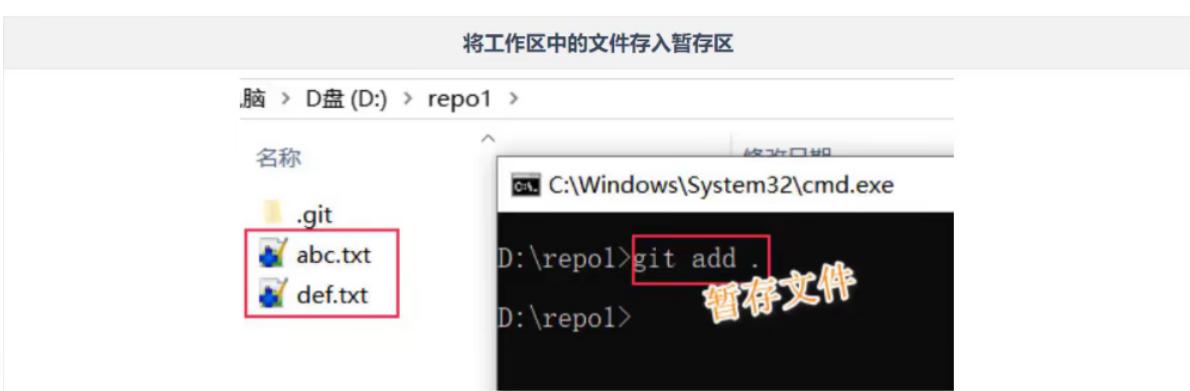
电脑 > D盘 (D:) > repo1 >

名称

.git  
abc.txt  
def.txt

D:\repo1>`git add .` 暂存文件

D:\repo1>



## 6.3提交文件

执行 `git commit -m "这里写提交的描述信息"` 作用是将暂存区的文件存入分支，形成一个版本

提交文件，形成一个版本

电脑 > D盘 (D:) > repo1

名称	修改日期	类型	大小
.git abc.txt def.txt			

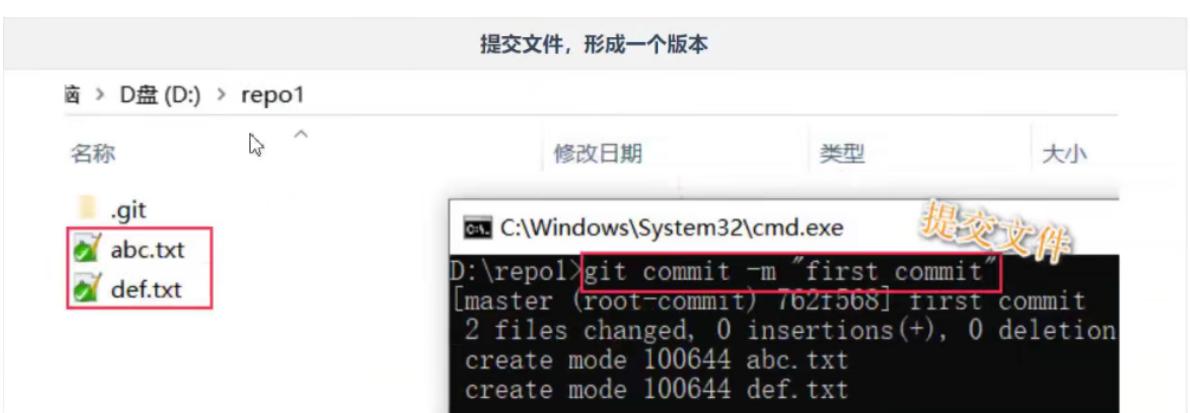
D:\repo1>`git commit -m "first commit"` 提交文件

[master (root-commit) 762f568] first commit

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 abc.txt

create mode 100644 def.txt



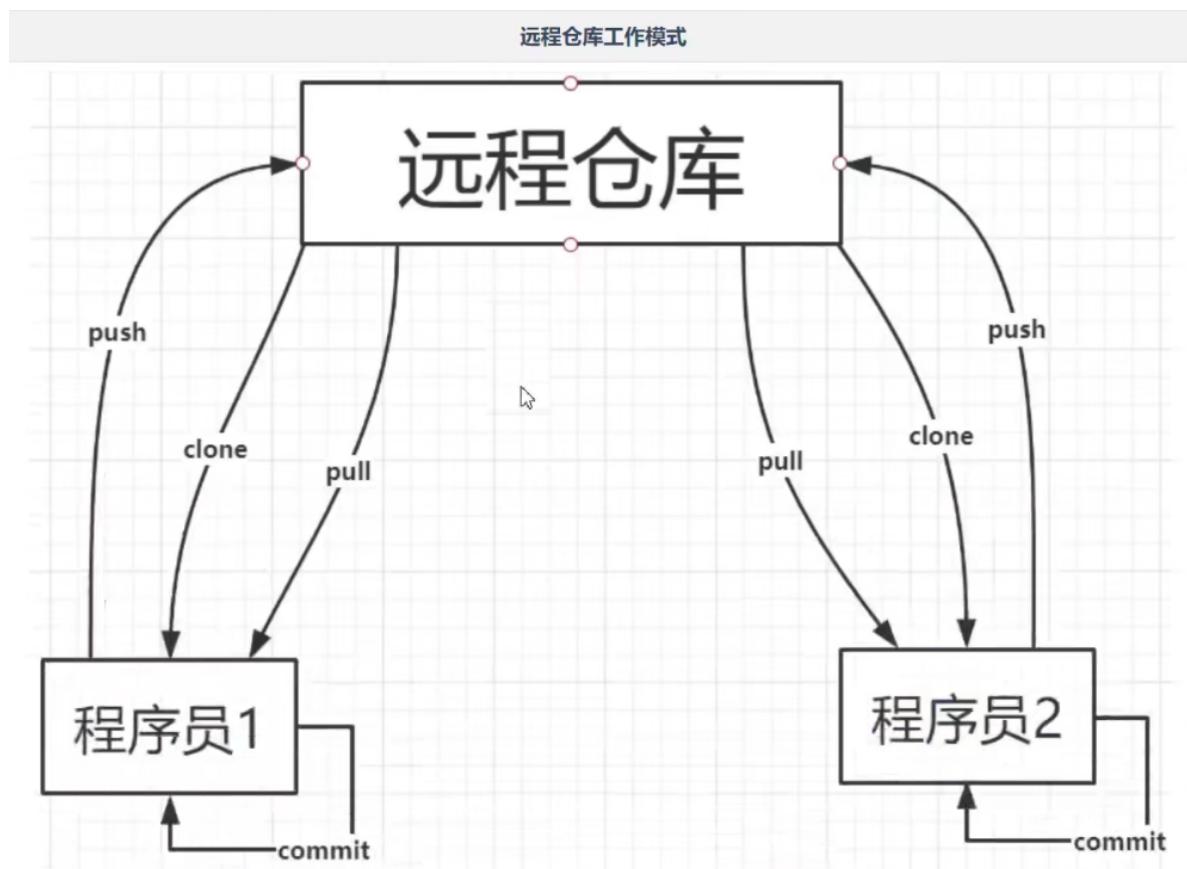
## 7.远程仓库

第5章中的仓库，其实是本地仓库。

当多人协同开发时，每人都在自己的本地仓库维护版本。

但很重要的一点是，多人之间需要共享代码、合并代码，此时就需要一个 **远程仓库**。

### 7.1远程仓库工作模式



### 7.2远程仓库选型

有很多远程仓库可以选择，比如 github(<https://github.com/>)，码云(<https://gitee.com/>)；

此两种可以注册自己测试使用，但如果是商业项目，需要更多支持需要付费。

公司内部也可以有自己构建的远程仓库([http://qianfeng.qfjava.cn:8087/users/sign\\_in](http://qianfeng.qfjava.cn:8087/users/sign_in))。

### 7.3基本操作

每个开发人员，在面对远程仓库时，会面临的一些基本操作。

#### 7.3.1注册git服务器账号

在 码云 注册账号，并登录。

进入公司后，很可能会使用公司自己搭建的git服务器，则账号向领导索要即可

### 点击注册，完成注册过程

### 登录后显示主页



### 7.3.2新建远程仓库

A screenshot of the 'Create Repository' page on Gitee. The page has a header '新建仓库' (Create Repository) and a sub-header '在其他网站已经有仓库了吗？点击导入' (Import from other website). The main form includes fields for '仓库名称' (Repository Name), '归属' (Owner), '路径' (Path), and '仓库介绍' (Repository Description). There are also several checkboxes for repository settings like '开源' (Open Source), '私有' (Private), and '企业内部开源' (Enterprise Internal Open Source). At the bottom right of the form, there is a blue '创建' (Create) button.

### 7.3.3本地关联远程仓库

```
D:\repo1>git remote add congyu https://gitee.com/cytus2/git_congyu.git  
D:\repo1>git remote -v  
congyu  https://gitee.com/cytus2/git_congyu.git (fetch)  
congyu  https://gitee.com/cytus2/git_congyu.git (push)
```

### 7.3.4推送文件到远程仓库

将本地仓库中已经commit的内容push到远程仓库，以共享自己的代码。

```
push  
C:\WINDOWS\system32\cmd.exe  
D:\repo1>git push origin master  
将本地master分支，上传到远程master分支  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (5/5), 381 bytes | 381.00 KiB/s, done.  
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0  
remote: Powered by GITEE.COM [GITEE-5.0]  
To https://gitee.com/shine10/git_shine.git  
 * [new branch]      master -> master  
本地master>远程master  
D:\repo1>
```

### 7.3.5 克隆远程仓库内容

如果仓库已经由别人创建完毕，我们需要其中的内容，则可以通过 `git clone` 将其复制到本地。

新建目录“repo2”，然后在其中执行 `git clone`

```
D:\> D盘 (D:) > repo2  
名称  
git_shine  
从远程仓库  
I 复制来的内容  
C:\Windows\System32\cmd.exe  
从远程仓库复制内容  
D:\repo2>git clone https://gitee.com/shine10/git_shine.git  
Cloning into 'git_shine'...  
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 5 (delta 1), reused 0 (delta 0), pack-reused 0  
Receiving objects: 100% (5/5), done.  
Resolving deltas: 100% (1/1), done.  
D:\repo2>
```

远程仓库，复制到本地，并自动初始化为一个本地仓库

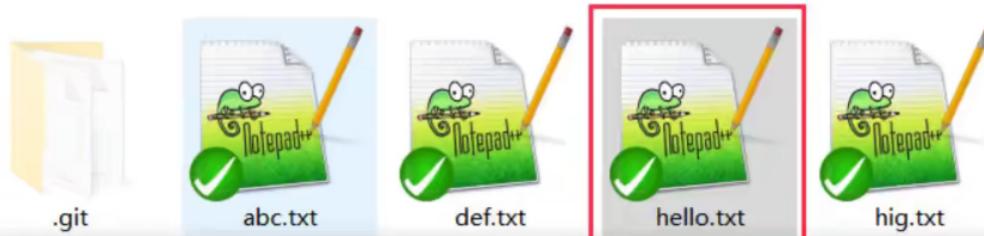
```
脑 : D盘 (D:) > repo2 > git_shine  
名称  
.git  
abc.txt  
def.txt  
复制来的内容，  
自动形成了本地仓库
```

### 7.3.6 代码共享

多人协同开发时，写好代码的 `git push` 上传到远程仓库；需要代码的 `git pull` 拉取代码即可。

有人再次将本地仓库内容，上传到了远程仓库

D 盘 (D:) > repo1 >



C:\WINDOWS\system32\cmd.exe

D:\repo1>git add .

**本地新增版本**

D:\repo1>git commit -m "new file hello.txt"

[master b73883e] new file hello.txt

1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 hello.txt

D:\repo1>git push origin master

**将本地的master分支的内容  
同步到远程仓库的master分支**

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Delta compression using up to 12 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (2/2), 230 bytes | 230.00 KiB/s, done.

Total 2 (delta 1), reused 0 (delta 0), pack-reused 0

remote: Powered by GITEE.COM [GSK-5.0]

To https://gitee.com/shinel0/git/shine.git

218ec77..b73883e [master -> master]

**本地master分支>远程master分支**

D:\repo1>

### 7.3.7 命令汇总

git remote add 标识名(origin) 远程地址	本地关联远程仓库
git push 标识名 master	将本地仓库内容上传到远程仓库
git pull 标识名 master	从远程仓库下载内容到本地仓库
git clone 远程地址	将远程仓库复制到本地，并自动形成一个本地仓库

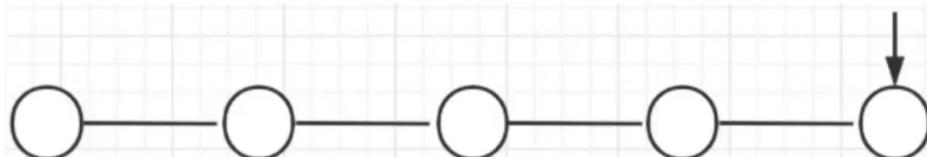
## 8. 分支

### 8.1 分支简介

分支，是一个个版本最终存储的位置。

分支，就是一条时间线，每次 **git commit** 形成一个个版本，一个个版本依次存储在分支的一个个提交点上。

分支由多个提交点组成，分支上会有一个指针，默认总是指向最新的提交点



## 8.2分支基操

### 8.2.1查看分支

查看当前仓库的分支 `git branch`

仓库中默认只有 `master` 分支

执行 `git commit` 时，默认是在 `master` 分支上保存版本。

默认只有 `master` 分支

```
D:\repo1>git branch
* master
```

### 8.2.2创建分支

在商业项目开发过程中，我们不会轻易的在 master分支 上做操作。

我们会新建一个 `开发用的分支`，在此分支上做版本的记录。

当代码确实没有问题时，才会将开发分支上成熟的代码版本添加到 `master分支` 上。

既保证开发过程中，可以及时记录版本，有保证master分支上每个提交点都是稳健版本。

创建分支

```
D:\repo1>git branch dev
Create a new branch
```

### 8.2.3切换分支

默认情况下，当前使用的分支是 **master分支**

可以切换到 **dev分支**，则后续的 `git commit` 便会在 **dev分支** 上新建版本(提交点)

切换分支

```
C:\Windows\System32\cmd.exe
D:\repo1>git checkout dev
Switched to branch 'dev'
切换到dev分支
D:\repo1>
```

## 8.3新建分支细节

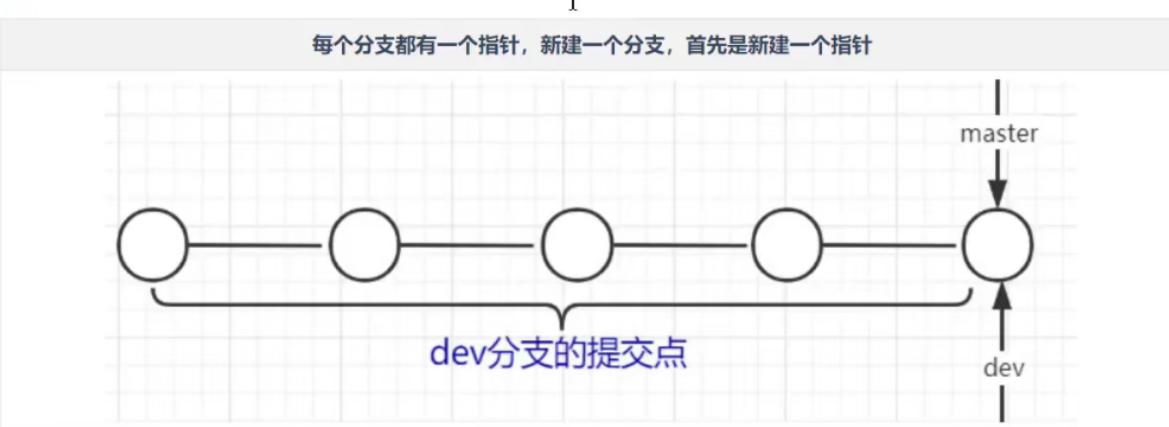
新建分支时，新分支，默认有哪些内容？分支中包含了哪些次提交？

### 8.3.1新分支初始内容

每个分支都有一个 **指针**，新建一个分支，首先是新建一个 **指针**。

而且新分支的指针会和当前分支指向 **同一个提交点**。

新分支包含的提交点就是从第一个提交点到分支指针指向的提交点。

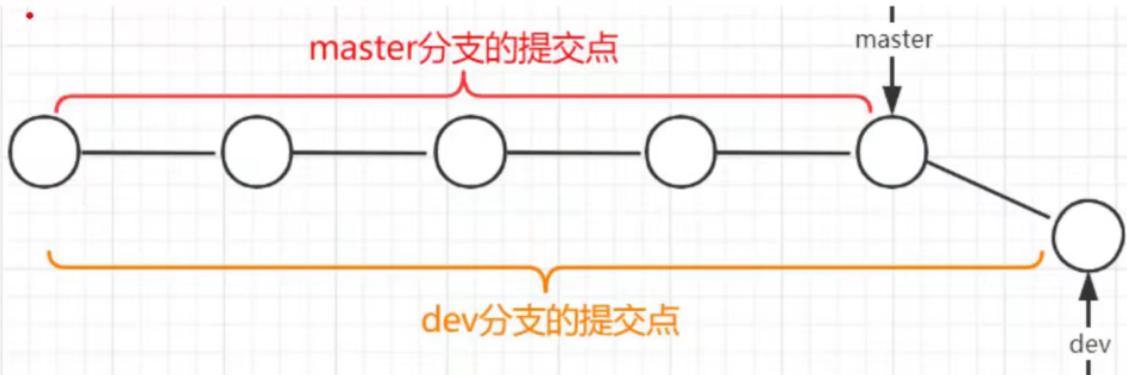


### 8.3.2多分支走向

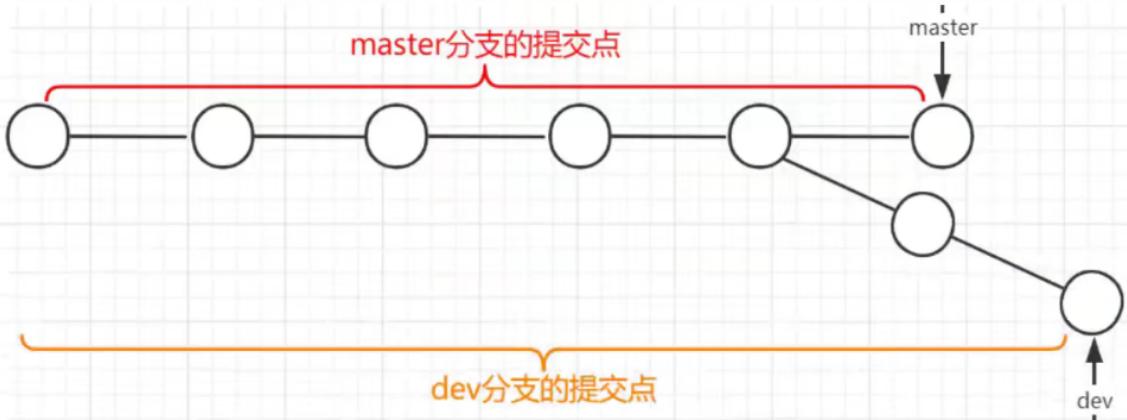
在master分支和新分支，分别进行 `git add` 和 `git commit`

分支情况如下图：

master分支未动，在dev分支增加一次commit



master分支增加一个commit，dev分支再增加一个commit



### 8.3.3 分支提交日志

查看分支的提交日志，进而看到分支中提交点的详细情况。

提交情况如下

插 > D盘 (D:) > repo1

名称	修改日期	类型	大小
.git			
abc.txt			
def.txt			
hig.txt			

C:\Windows\System32\cmd.exe

```
D:\repo1>git commit -m "first commit"
[master (root-commit) 762f568] first commit
 2 files changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 abc.txt
  create mode 100644 def.txt
两次提交，两个提交点
D:\repo1>git add .

D:\repo1>git commit -m "second commit"
[master 6185b10] second commit
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 hig.txt
```

查看当前分支的 提交日志

```
C:\Windows\System32\cmd.exe
create mode 100644 abc.txt
create mode 100644 def.txt

D:\repol>git add .

D:\repol>git commit -m "second commit"
[master 6185b10] second commit
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 hig.txt

D:\repol>git log --oneline
6185b10 (HEAD -> master) second commit
762f568 first commit

D:\repol>git log
commit 6185b10f77ccf6664c73cca83d78cfe7055a4fa0 (HEAD -> master)
Author: zanghongjiu <zanghongjiu@sina.com>
Date:   Mon Mar 30 19:24:59 2020 +0800

second commit

commit 762f568e640c7c9e602b5791da6865dac53784fb
Author: zanghongjiu <zanghongjiu@sina.com>
Date:   Mon Mar 30 19:24:37 2020 +0800

first commit
```

### 简易日志

### 完整日志

## 8.4 分支合并

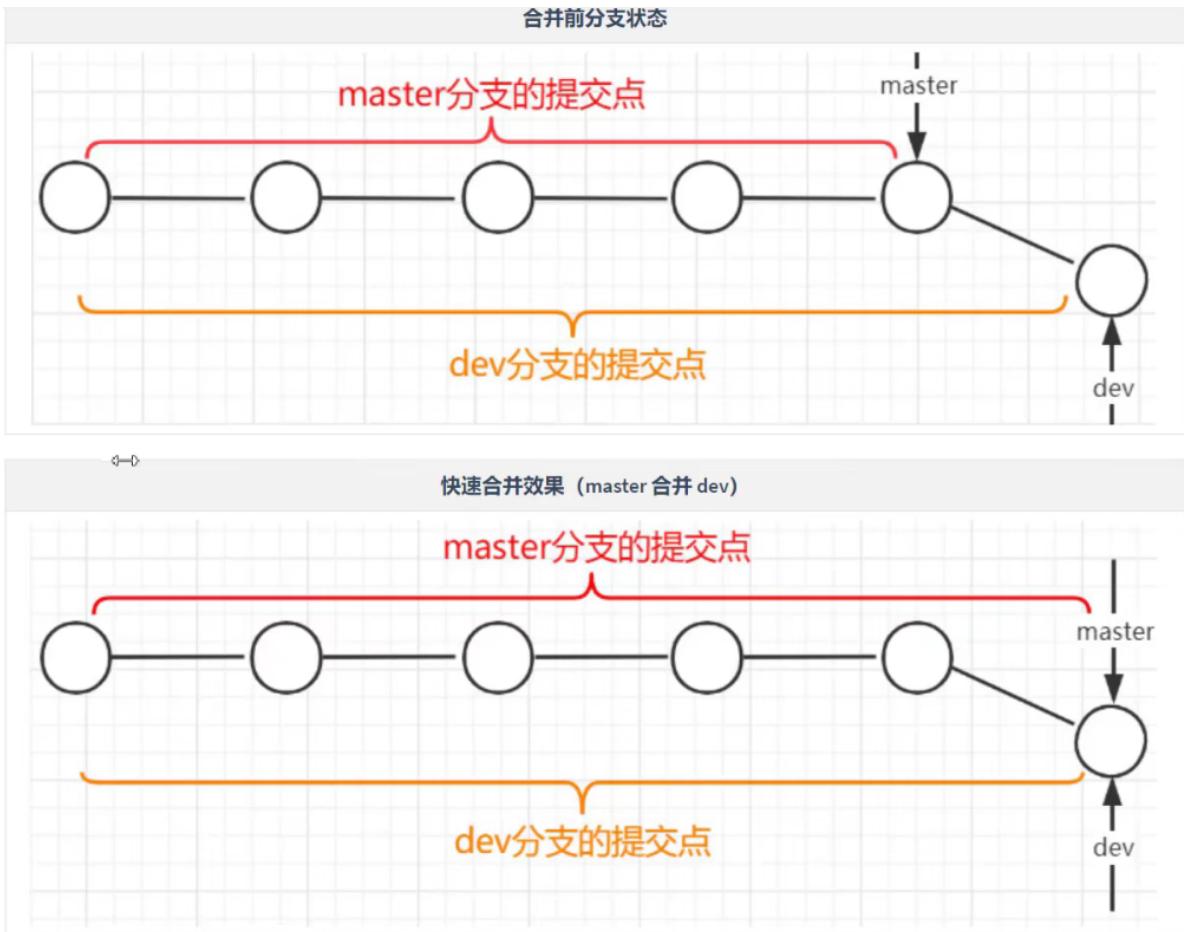
两个分支内容的合并

git merge 分支a 合并分支a

合并的方式有两种：快速合并 和 三方合并。

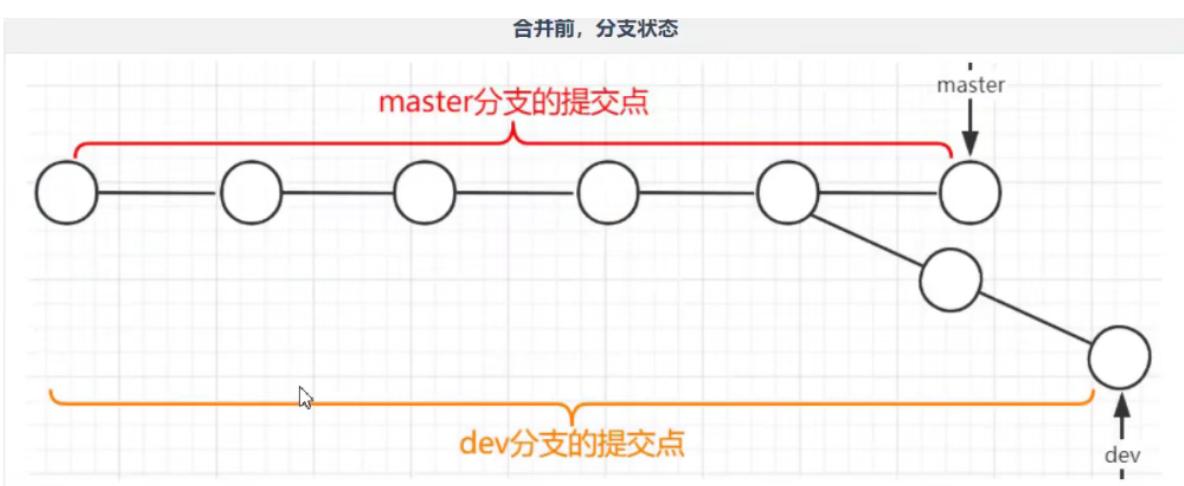
### 8.4.1 快速合并

如果分支A当前的修改，是完全基于分支B的修改而来，则B分支合并A分支，就是移动指针即可。

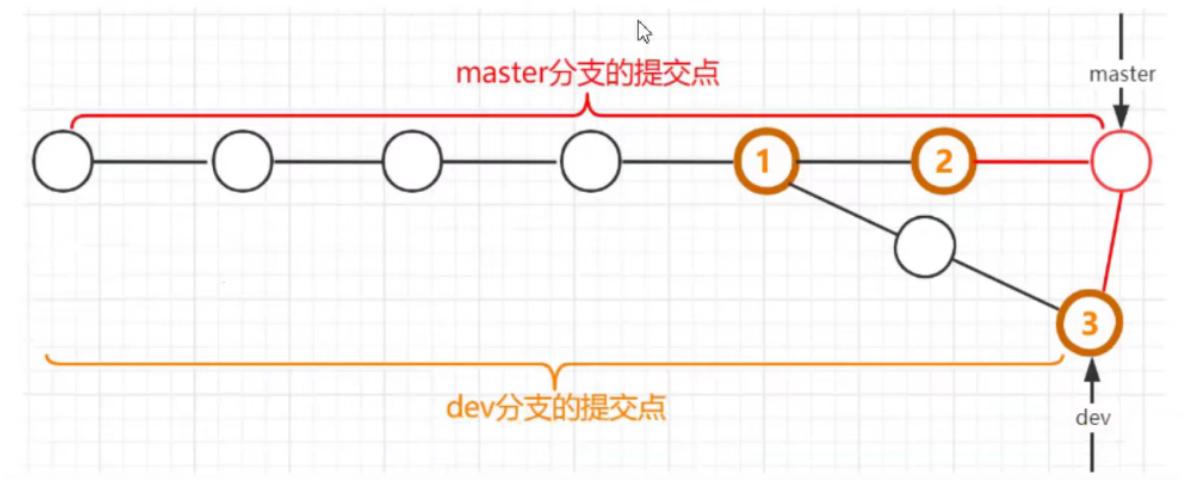


#### 8.4.2 三方合并

在不具备快速合并的条件下，会采用三方合并。



三方合并，将 2 和 3 的更改都累加在 1 上，形成新的提交点



### 8.4.3 合并冲突

两个分支进行合并，但它们含有对同一个文件的修改，则在合并时出现冲突，git无法决断该保留改文件哪个分支的修改。

#### 8.4.3.1 冲突演示

场景模拟如下：

master分支修改hig.txt文件

```
D:\> D盘 (D:) > repo1 >
名称
  .git
  abc.txt
  def.txt
  hig.txt

C:\Windows\System32\cmd.exe
6185b10 (HEAD -> dev, master) second commit
762f568 first commit
D:\repo1>git checkout master
Switched to branch "master"
M      hig.txt
D:\repo1>git add .
D:\repo1>git commit -m "master update hig.txt"
[master 5060d49] master update hig.txt
 1 file changed, 1 insertion(+)

D:\repo1>
```

D:\repo1\hig.txt - Notepad++

文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R) 插

1 hello master update

dev分支修改hig.txt

```

    道 > D 盘 (D:) > repo1 >
    名称
    .git
    abc.txt
    def.txt
    hig.txt

    C:\Windows\System32\cmd.exe
    D:\repo1>git checkout dev
    Switched to branch 'dev'
    D:\repo1>git add .
    D:\repo1>git commit -m "dev update hig.txt"
    [dev d103217] dev update hig.txt
    1 file changed, 1 insertion(+)

    D:\repo1>

    D:\repo1\hig.txt - Notepad++
    文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(...
    hig.txt
    1 world dev update
  
```

在master分支 合并 dev分支

合并dev分支

```

    C:\Windows\System32\cmd.exe
    合并
    D:\repo1>git merge dev
    Auto-merging hig.txt
    重现冲突
    CONFLICT (content): Merge conflict in hig.txt
    Automatic merge failed; fix conflicts and then commit the result.

    D:\repo1>
  
```

此时，打开hig.txt文件：

冲突后，git会将两个分支的内容都展示在文件中

```

    D:\repo1\hig.txt - Notepad++
    文件(E) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置
    hig.txt
    1 <<<<< HEAD 当前分支内容
    2 hello master update
    3 =====
    4 world dev update 被合并分支内容
    5 >>>>> dev
    6

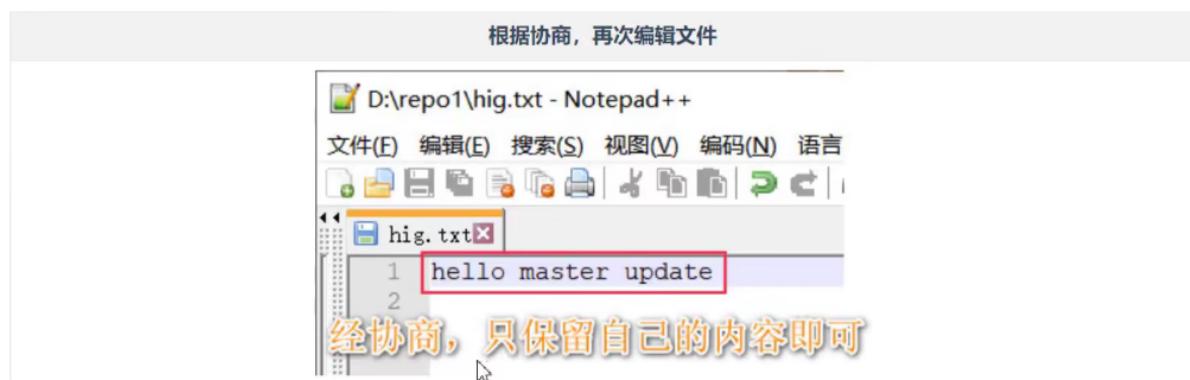
    两个分支的内容用 <<< ==>>> 做了分隔
  
```

### 8.4.3.2 冲突解决

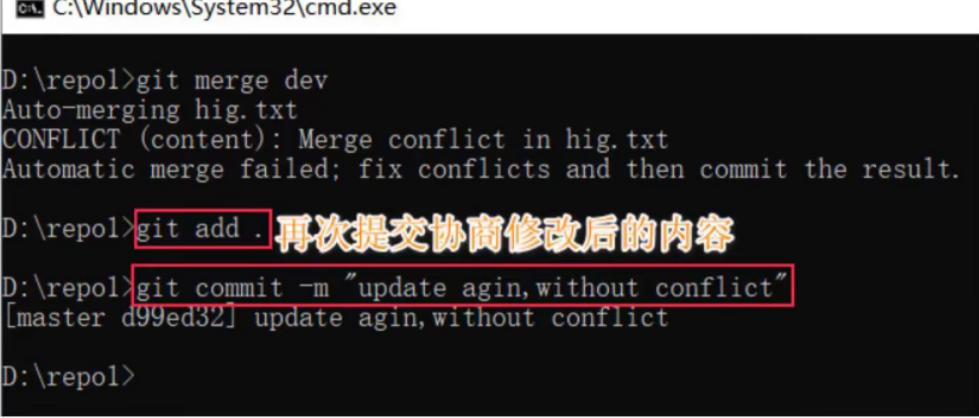
出现冲突后，如要由两个开发人员当面协商，该如何取舍，为冲突文件定义最终内容。

解决方案：

1. 保留某一方的，删除另一方的
2. 保留双方的
3. 但无论如何，要记得删除 <<< ==>>> 这些
4. 本质是两人协商为冲突的内容，定制出合理的内容。



提交再次编辑后的文件



```
D:\repo1>git merge dev
Auto-merging hig.txt
CONFLICT (content): Merge conflict in hig.txt
Automatic merge failed; fix conflicts and then commit the result.

D:\repo1>git add .再次提交协商修改后的内容
D:\repo1>git commit -m "update again, without conflict"
[master d99ed32] update again, without conflict

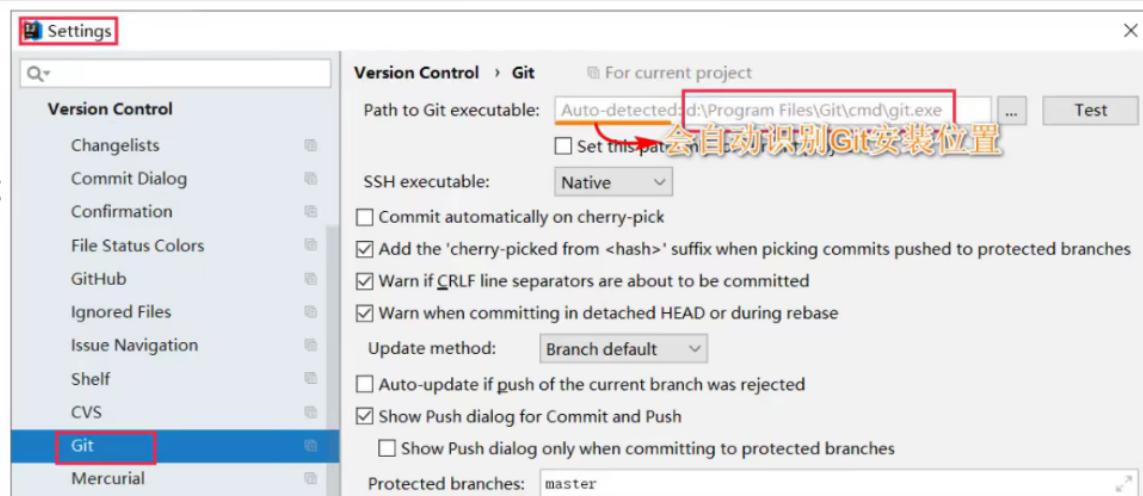
D:\repo1>
```

## 九、Idea关联Git

### 9.1 关联git

File > Settings 关联过程是自动的

此处关联是Idea可以自动完成的



## 9.2 创建仓库

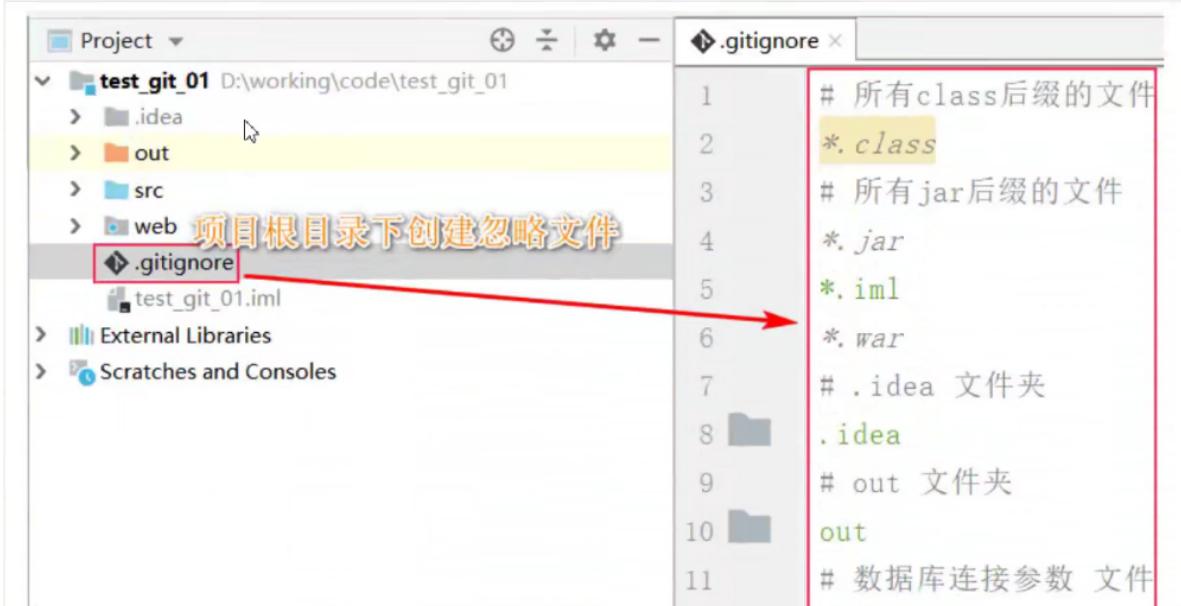
新建项目后，将项目目录创建为git仓库

不过，注意！！要先设置 忽略文件 ".gitignore"！

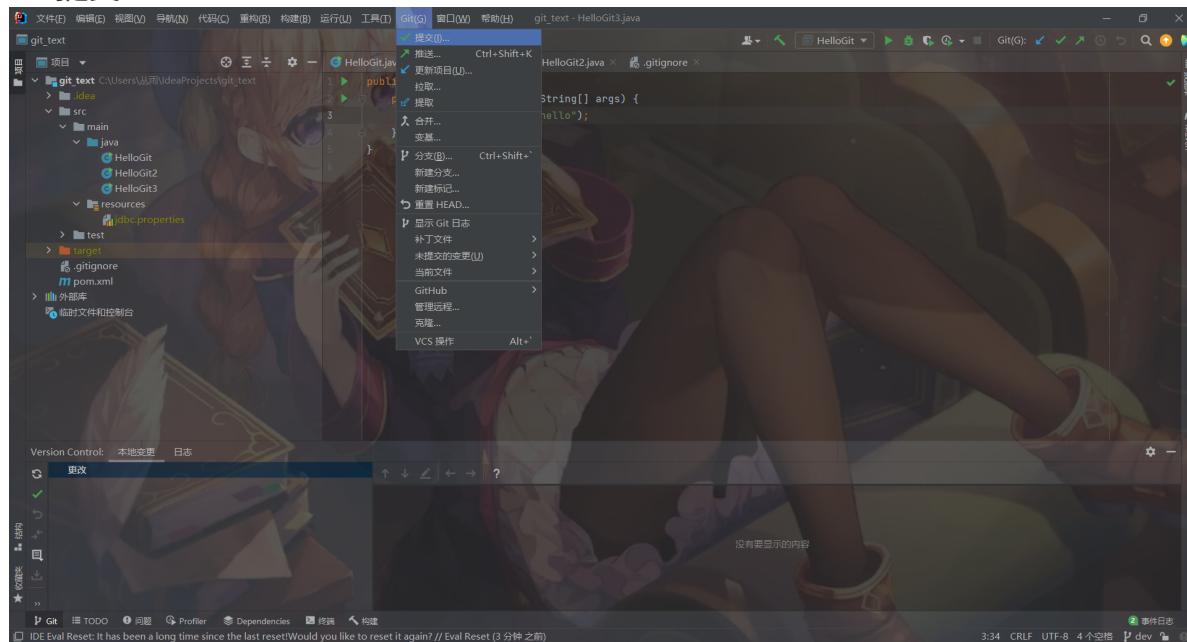
作用：被忽略的文件会被版本记录忽略，版本中不包含它们。

范围：不需要和其他开发共享的文件，具体见下图。

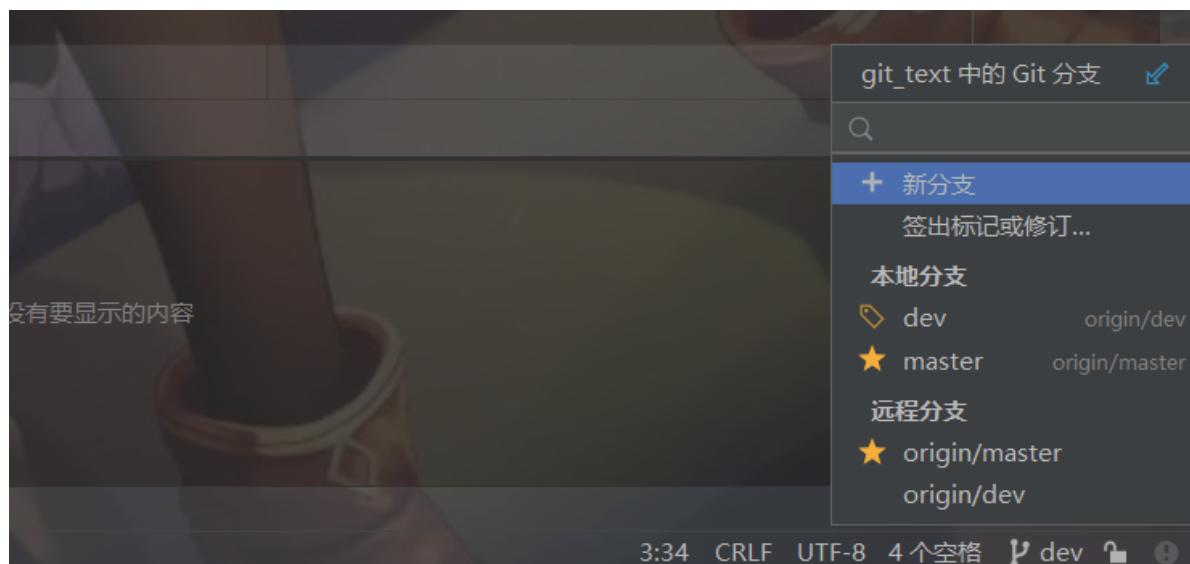
创建仓库前，先添加忽略文件



## 9.3 提交-commit

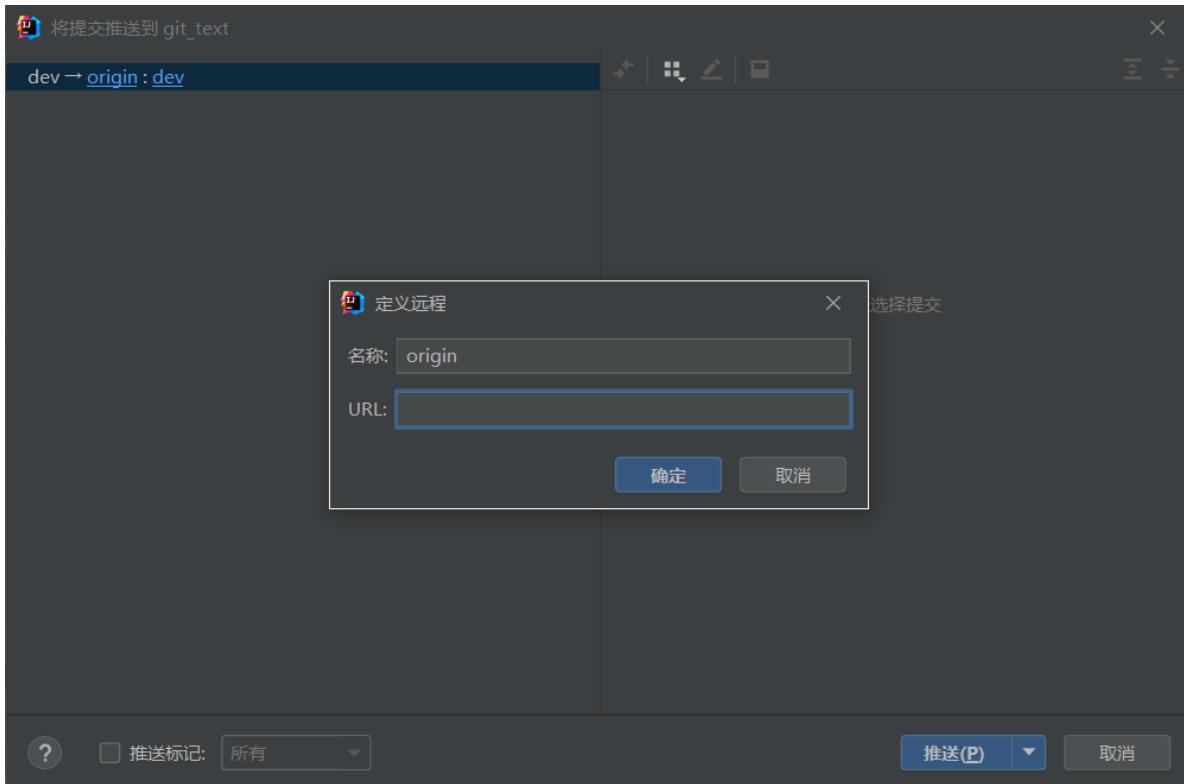


## 9.4 创建分支



## 9.5 上传到远程仓库

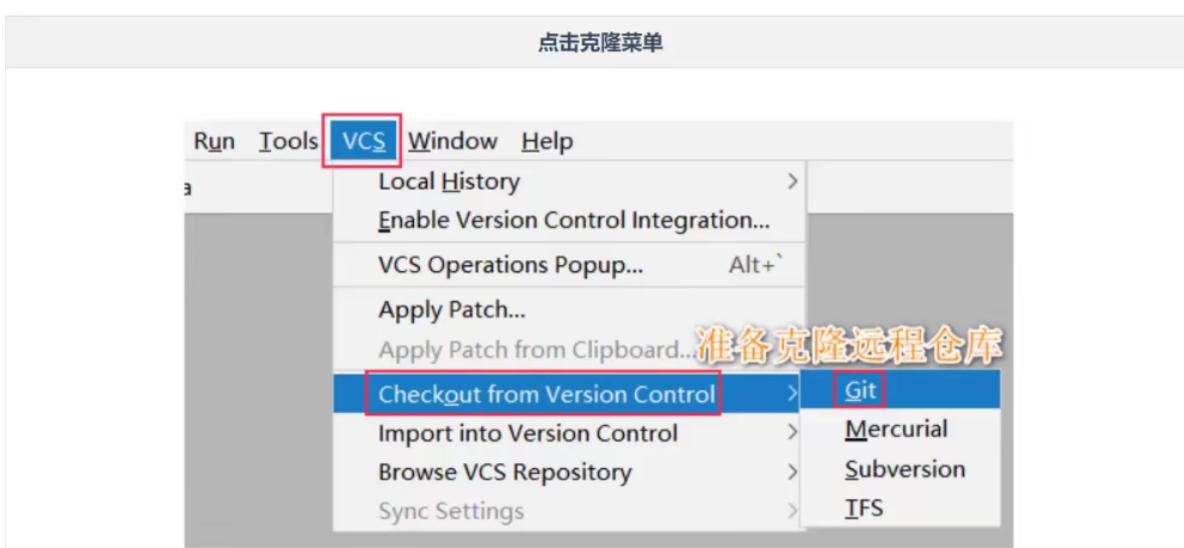
URL就是远程仓库地址



## 9.6 复制到本地仓库

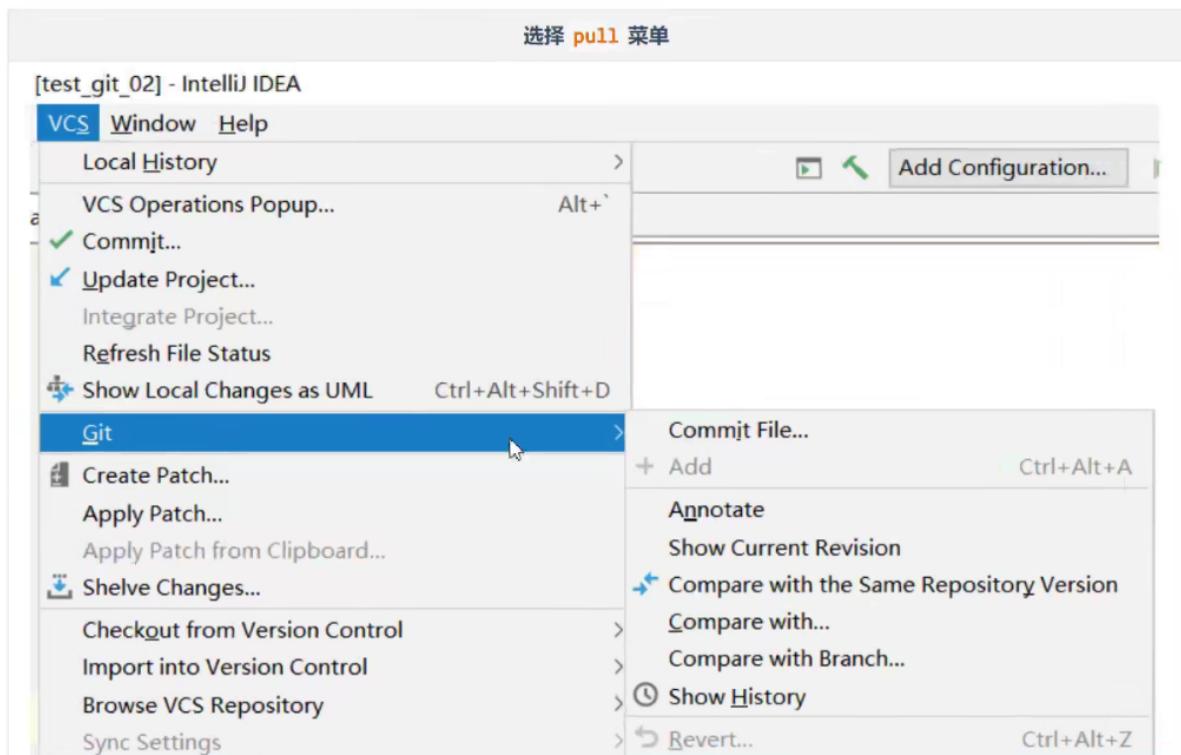
如果有建好的远程仓库，比如公司内已经在用的仓库，或者github，码云上的一些公开仓库。

可以用将其复制到本地使用



## 9.7 更新本地项目

如果远程仓库有更新，则你的本地项目也需要一起更新。



## 9.8 冲突解决

合并分支时，如果出现冲突，则需要解决冲突。



## 十、多人协同开发

多人开发协同，git操作

### 10.1 项目管理员(项目经理)

1> 由管理员负责创建一个远程库，初始的库中什么也没有，为裸库。库的名称建议和项目同名

2> 管理员会在idea中创建一个初始项目,其中包含 `.gitignore`文件。

并在项目根目录下 建立本地库。并建立 `dev`分支。

3> 管理员将本地库上传到远程库

4> 将其他开发人员拉入远程库的 `开发成员列表中`，使得其他开发人员可以访问该远程库。

流程如下：

The screenshot shows the 'Invite User' section of a repository settings page. It includes a note about adding up to 5 members to a private repository. A red box highlights the 'Direct Addition' button. Another red box highlights the 'Developer' permission level dropdown and the input field for the developer's username. A third red box highlights the 'Add' button.

The screenshot shows a confirmation dialog for inviting a user. It displays the user's email and nickname, their personal space address, repository permission (Developer), account status (Cloud Account), and the 'Submit' button. A red box highlights the 'Submit' button.

序号	用户邮箱	用户昵称	个人空间地址	仓库权限(双击可变更)	用户状态	操作
1	****	126	126	开发者	码云用户	移除

查看已添加的开发成员

仓库设置

仓库成员管理

所有

管理员

**开发者**

观察者

报告者

为什么我不能添加仓库成员（[协作者列表](#)）  
个人私有仓库最多支持 5 人协作（如个人拥有多个私有仓库，所有协作人数总计不得超过 5 人）。企业仓库 / 私有仓库协作开发，更适合使用[码云企业版](#)，[了解区别](#)

**开发者 (1) [?](#)**

Z 126 z@126.com

权限说明：开发者能推送代码，新建和删除分支，创建Issue, Pull Request, Wiki 等

开发者 [移出仓库](#)

5> master分支设置为 [protected分支](#)，只有管理员有权限将代码合并到其中。dev分支设置为 [常规分支](#) 所有开发人员都可以其中合并代码

进入分支设置

暂无描述

4 次提交

**2 个分支**

0 个标签

0 个发行版

从 1 位贡献者

分支保护是为了防止相关成员推送代码到重要的分支（例如 master 分支）。便于仓库的分支管理，点击前往保护分支规则设置。

- 常规分支：仓库成员（开发者权限及以上）可推送分支
- 保护分支：可自定义保护策略，默认仓库管理员才能管理（推送）被保护的分支
- 只读分支：任何人都无法推送代码（包括管理员和所有者），需要推送代码时应设为“常规”或“保护”分支

设置保护分支，让master分支不能被随意更改

所有分支

+ 新建分支

分支名	更新信息	状态	操作
dev <a href="#">默认分支</a>	Z a60ddaa update src/com/zhihu/e...	23分钟前	常规分支 <a href="#">设置保护分支</a>
master	Z a60ddaa update src/com/zhihu/e...	23分钟前	保护分支 <a href="#">设置保护分支</a>

## 10.2 开发人员

- 1> 初始化：在idea中 [clone](#) 远程库，获得项目。会建立本地库
- 2> 后续的开发中，都要在dev分支上进行。开发完一个功能并测试通过后就 [commit](#) 提交到本地的dev分支中，然后上传([push](#))到远程dev分支中。
- 3> 需要更新项目内容时，通过 [pull](#) 从远程仓库拉取内容。
- 4> 注意：多人协同时，每次在 [push](#) 到远程库前，都先做一次 [pull](#)，一来是把远程最新内容合并到本地，二来是核实本地内容是否和远程内容有冲突。

## 十一、经典问题

在使用https协议做push时，如果曾经使用过码云，但密码有过改动，此时会报错

### 使用https协议报错

```
D:\repol>git push origin master
remote: You do not have permission to push to the repository via HTTPS
fatal: Authentication failed for 'https://gitee.com/shine29/git_shine.git/'
```

解决方案：控制面板 → 凭据管理器 → 删除对应凭证，再次使用时会提示重新输入密码。

### 删除之前的码云凭证，然后重新push即可

控制面板 > 所有控制面板项 > 凭据管理器

/ 工具(T)

#### 管理你的凭据

查看并删除你为网站、已连接的应用程序和网络保存的登录信息。



Web凭据



Windows凭据

备份凭据(B) 还原凭据(R)

#### Windows凭据

添加 Windows凭据

无 Windows凭据。

#### 基于证书的凭据

添加基于证书的凭据

无证书。

#### 普通凭据

添加普通凭据

git:https://gitee.com

修改时间: 今天



Internet 地址或网络地址: git:https://gitee.com

用户名: shine\_1229@sina.cn

密码: **删除之前的凭证**

永久性: 本地计算机

编辑

**删除**