

招商信诺 Git 学习文档

(V1.0.4)

销售渠道系统开发团队

修订记录:

版本号	修订人	修订日期	修订描述
V1.0.1	Widon.Han	2021-03-26	创建初稿
V1.0.2	Rodey Luo	2021-03-30	增加前端开发工具相关插件描述
V1.0.3	Widon.Han	2021-03-31	增加标签处理，代码评审机制
V1.0.4	Widon.Han	2021-04-01	idea 中 Terminal 窗口集成 Git

公司内部 GitLab 文档访问地址:

<http://10.142.146.37:8081/common/cmb-common-doc>

目 录

1 简介.....	5
1.1 目的.....	5
1.2 范围.....	5
2 安装 git 客户端.....	5
3 打开 git 命令行控制台.....	5
4 配置客户端.....	6
4.1 配置用户名邮箱.....	6
4.2 配置保存密码.....	7
5 克隆项目.....	7
5.1 获取项目地址.....	7
5.2 在指定的文件夹中克隆项目.....	8
6 修改项目提交并推送.....	8
6.1 git 三级结构.....	8
6.2 文件状态查看.....	9
6.3 撤销工作区修改.....	10
6.3.1 插件操作.....	10
6.3.2 命令行操作.....	10
6.4 把工作区修改提交至 staging area(暂存区).....	11
6.5 撤销暂存区修改.....	12
6.6 提交暂存区修改到本地仓库.....	13
6.7 查看当前版本提交日志记录.....	13
6.8 撤销回退本地仓库的版本.....	14
6.9 把本地修改推送到远程版本库.....	14
7 比较不同.....	14
8 本地仓库和项目更新.....	15
8.1 查看本地仓库对应的远程仓库信息.....	15
8.2 本地仓库更新-不会自动合并.....	15
8.3 本地仓库某个具体分支代码更新.....	16
9 解决合并冲突.....	17
10 分支相关.....	17
10.1 查看分支信息.....	17
10.1.1 插件操作.....	17
10.1.2 命令行操作.....	18
10.2 新建分支.....	18

10.2.1 基于本地分支新建.....	18
10.2.2 基于远程仓库分支新建.....	19
10.2.3 新建分支推送到远程仓库.....	19
10.3 删除分支.....	19
10.3.1 删除本地分支.....	19
10.3.2 删除远程分支.....	19
10.4 分支版本合并.....	19
10.5 分支版本切换.....	20
10.5.1 本地分支切换.....	20
10.5.2 基于本地当前分支新建分支并切换.....	20
10.5.3 基于远程仓库的分支新建分支并切换.....	20
11 本地已有项目 push 到远程仓库.....	20
11.1 向项目组长申请新项目接入.....	20
11.2 初始化本地仓库.....	22
11.3 把本地项目提交到本地仓库.....	22
11.4 本地仓库添加远程仓库地址.....	23
11.5 更新本地仓库信息.....	23
11.6 本地仓库信息推送远程仓库.....	23
12 如何理解分支开发.....	24
12.1 项目小组长准备工作.....	24
12.2 利用 Eclipse 插件切换.....	25
12.2.1 克隆下来默认是 master.....	25
12.2.2 右键项目-->team.....	26
12.2.3 选择新建分支的原分支.....	27
12.2.4 点击完成.....	28
12.3 利用命令行进行切换到开发分支.....	29
12.3.1 项目克隆下来后默认只有 master 分支.....	29
12.3.2 用命令基于远程分支新建本地开发分支.....	29
13 忽略文件.gitignore 文件配置.....	29
14 插件操作.....	30
14.1 Eclipse 导入 GitLib 项目-法一.....	30
14.2 Eclipse 导入 GitLib 项目-法二.....	31
14.3 插件操作-Eclipse.....	32
14.4 插件操作-Idea.....	32
14.5 是否需要提交更新标志.....	35
14.6 Visula Studio Code（目前主要前端开发人员使用）相关插件.....	35
15 IDEA 中 Terminal 集成 Git.....	37
15.1 设置中添加 git 命令执行路径.....	37
15.2 快速进入 Terminal 窗口.....	37

16 使用 Git 做分支管理.....	38
16.1 目前现状.....	38
16.2 最终目标.....	39
16.3 版本命名规范.....	40
16.4 版本修改举例说明.....	41
17 GitLab 中代码评审机制.....	41
17.1 设置用户权限.....	42
17.2 设置分支保护.....	42
17.3 基于 branch-dev 新建一个本地分支 feature-widon, 并推送到 Gitlab.....	43
17.4 在 GitLab 中新建 merge request.....	43
17.5 项目负责人进行合并操作.....	44
18 生产发布后版本标签 tag.....	45
18.1 创建 tag.....	45
18.2 查看 tag.....	45
18.3 删除 tag.....	45
18.4 检出 tag.....	46
19 gitlab 提交代码自动触发 jenkins 构建.....	46
20 项目托管到 github 操作流程(拓展).....	46

1 简介

1.1 目的

招商信诺销售渠道开发团队内部产品研发团队 Git 命令行学习文档

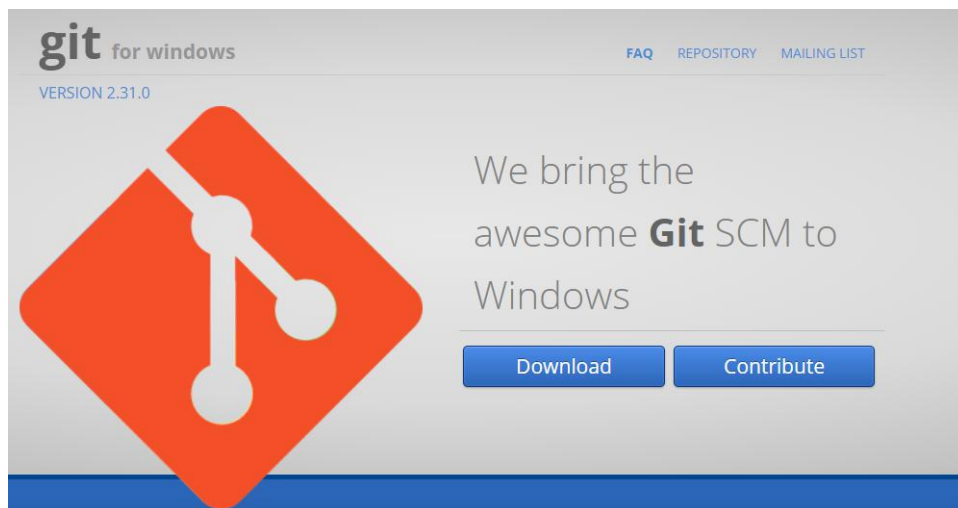
1.2 范围

招商信诺销售渠道开发团队所有人员

2 安装 git 客户端

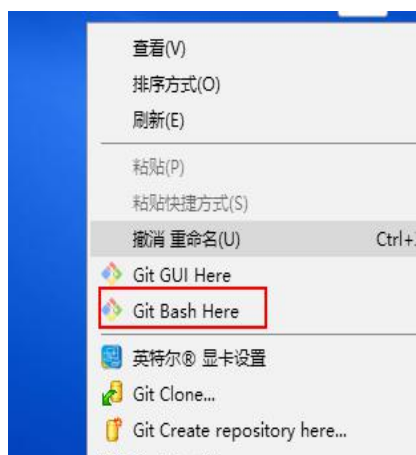
下载安装 git for windows, 安装步骤可按网上流传即可

<https://gitforwindows.org/>



3 打开 git 命令行控制台

在任何目录可以右键然后进入控制台,如果是操作某个项目,进入到 git 项目的文件夹,注: git 项目的一级目录有 `.git` 文件夹



4 配置客户端

4.1 配置用户名邮箱

该配置主要用于显示提交记录的的用户名等相关信息

设置用户名

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

设置用户的邮箱

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

A screenshot of a Windows command prompt window titled 'MINGW64:/d/widon'. The prompt shows a series of Git configuration commands and their outputs. The user is 'SZK190078@sziy2069' and the directory is '/d/widon'. The commands and outputs are:
\$ git config --global user.name widon
\$ git config --global user.email widon.han@cignacmb.com
\$ git config --global user.name
widon
\$ git config --global user.email
widon.han@cignacmb.com
\$ |

配置了以上用户信息后，修改代码提交后，提交日志里显示刚配置的用户名



注：我们为了学习使用快，采用的 gitLab 的 http 协议模式，没有用到 SSH 协议，所以不用生成 RSA 密码对以及上传到 gitlab，如果自己玩的话，GitLab 和开源中国的 OSgit 可以自己根据网上的教程自己配置。SSH 协议快，安全，不用每次都输入密码。

4.2 配置保存密码

使用 gitLab 的 http 协议，对于 clone,pull,push 操作需要提示输入密码，可以配置保存密码，不用每次都输入。

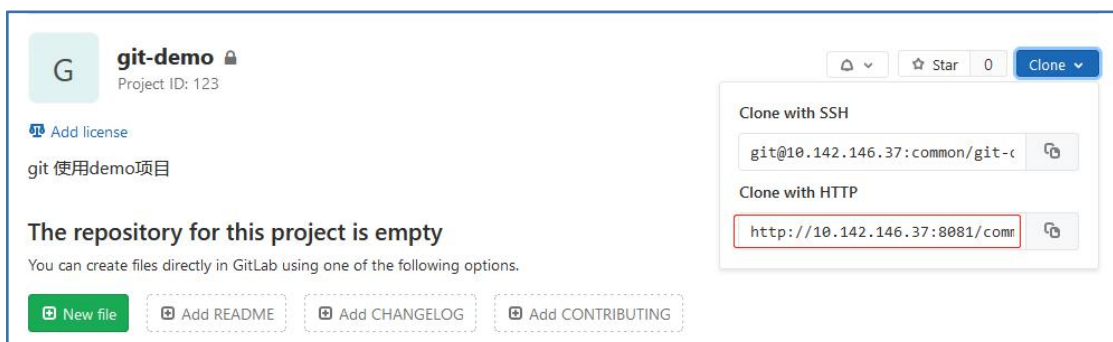
具体配置:当你配置好 git 后，在 C:\Documents and Settings\Administrator\ 目录下有一个 .gitconfig 的文件，里面会有你先前配好的 name 和 email，只需在下面加一行

```
[credential]  
helper = store
```

5 克隆项目

5.1 获取项目地址

当管理员给用户分配权限后，用户可以进入到该项目，可以查看仓库相关信息，特别是地址信息

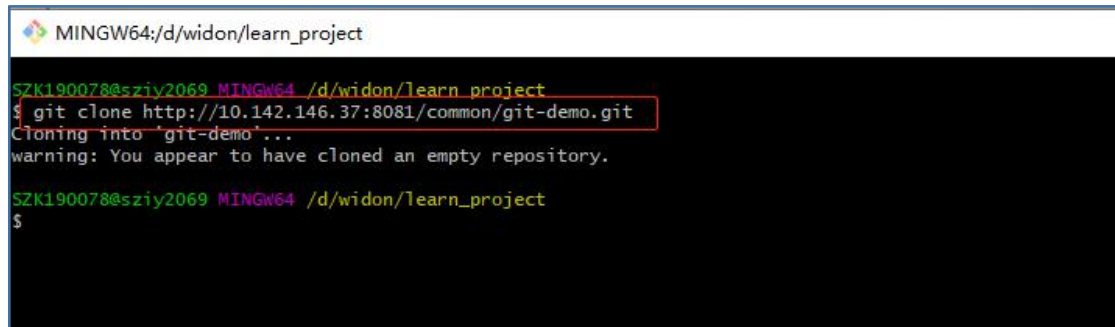


有了这个地址，就可以克隆了。

5.2 在指定的文件夹中克隆项目

在指定文件夹打开控制台，输入克隆命令

```
git clone http://10.142.146.37:8081/common/git-demo.git
```

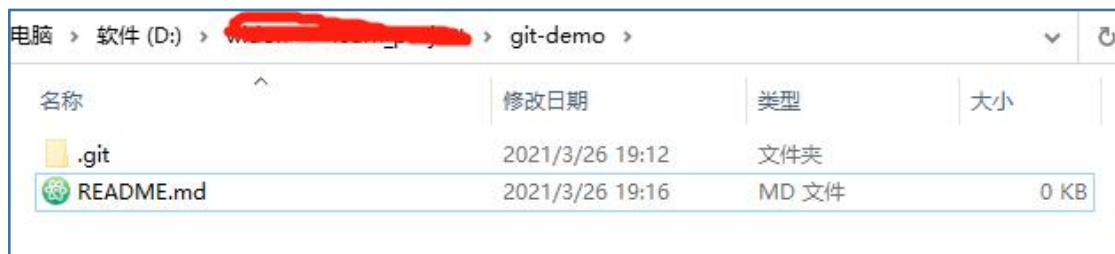


```
MINGW64:/d/widon/learn_project
$ git clone http://10.142.146.37:8081/common/git-demo.git
Cloning into 'git-demo'...
warning: You appear to have cloned an empty repository.
$
```

forest	2021/1/7 16:54	文件夹
git-demo	2021/3/26 19:07	文件夹
grpc-spring-boot-starter	2019/7/22 15:37	文件夹

如果提示输入密码，输入后继续

克隆下来是个空项目，新建一个空文件 README.md

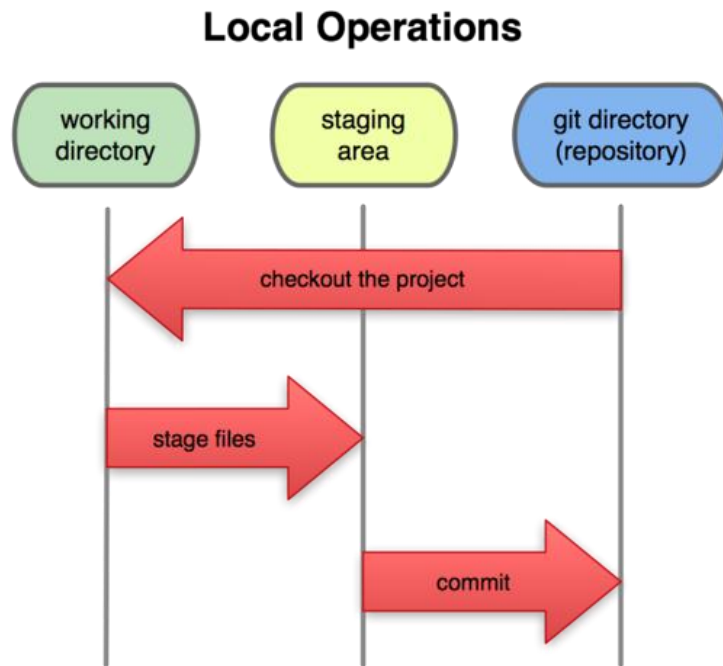


名称	修改日期	类型	大小
.git	2021/3/26 19:12	文件夹	
README.md	2021/3/26 19:16	MD 文件	0 KB

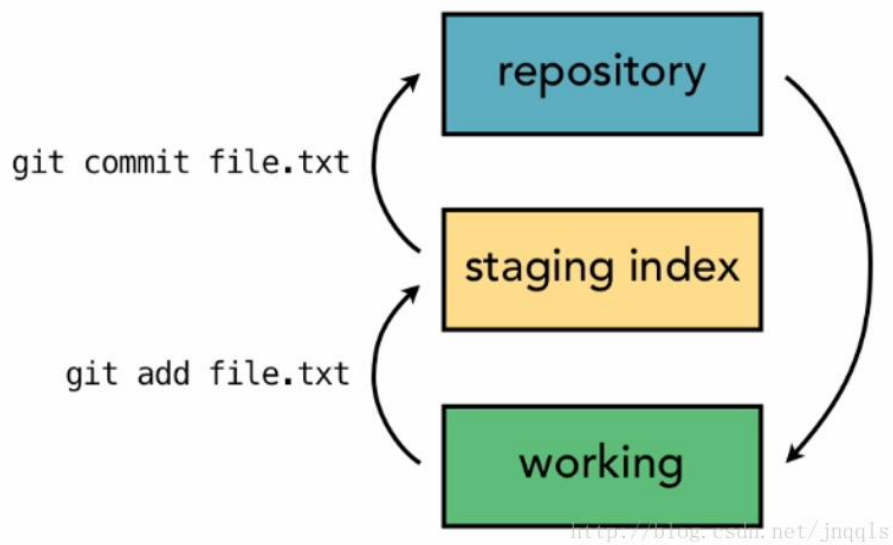
6 修改项目提交并推送

6.1 git 三级结构

工作区(本地编辑器)的修改先提交(git add)到暂存区，暂存区再提交(git commit)到本地仓库，本地仓库再推送到(git push)远程仓库



three-tree architecture



6.2 文件状态查看

文件修改后提交之前是红色，提交到暂存区后变成绿色了。这些文件状态可以通过命令查看

git status

6.3 撤销工作区修改

6.3.1 插件操作

Idea 插件无

6.3.2 命令行操作

查看状态，发现工作区文件已修改。

新建的文件，如果不需要可以直接删除掉

```
MINGW64:/d/widon/learn_project/git-demo

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

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

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ |
```

如果这个文件已经在本地仓库存在,现在只是在这个文件的基础上修改，可以用撤销命令撤销修改

```
MINGW64:/d/widon/learn_project/git-demo

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$
```

撤销工作区的修改命令

git checkout <filename> 对 filename 文件的修改进行撤销

git checkout . 对工作区中的所有修改进行撤销，注意有一点

```
SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git checkout .

SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean

SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$
```

6.4 把工作区修改提交至 staging area(暂存区)

git add <filename1> 把工作区修改的文件 1 的提交到暂存区

git add . 把工作区中所有的修改提交到暂存区

提交前:

```
MINGW64:/d/widon/learn_project/git-demo

SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

提交后:

```
MINGW64:/d/widon/learn_project/git-demo

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git add .

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

6.5 撤销暂存区修改

工作区的修改文件加入到暂存区后,查看状态是绿色,此时可以从暂存区撤销

```
SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
(use "git branch --unset-upstream" to fixup)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

用命令

git reset HEAD <file>

撤销命令

```
SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git reset HEAD README.md
Unstaged changes after reset:
M       README.md
```

查看文件状态,发现工作区中该文件仍然是修改状态,只是撤销了暂存区,没有撤销工作的修改

```

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

```

6.6 提交暂存区修改到本地仓库

git commit -m“此次提交注释”

提交暂存集结区的修改，并加上注释

git commit -a -m“此次提交注释”

可以把工作区文件的修改直接提交，不用先加入到暂存区，再提交。

注意：新加入的文件还是得一步一步的提交。

```

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git add .

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git commit -m "修改README.md文件"
[master 8964ed8] 修改README.md文件
1 file changed, 1 insertion(+)

```

6.7 查看当前版本提交日志记录

git log 查看版本的提交记录，以选择回退到之前哪个版本。可以得到提交 id
commit_id,用于版本定位

... **git log** 还有很多参数，命令很强大，可以自己学习，比如查看某个文件的修改历史

6.8 撤销回退本地仓库的版本

`git reset --hard HEAD^` 回退到前一个版本。

`git reset --hard commit_id` 回退到之前指定的一个版本。

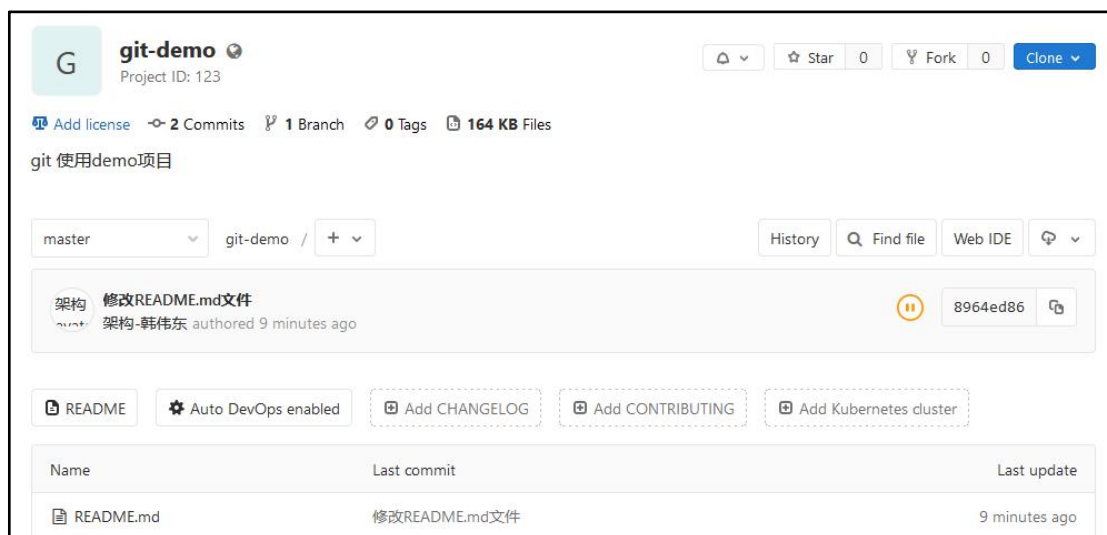
6.9 把本地修改推送到远程版本库

`git push origin <branch>` 把某个分支的修改推送到远程仓库

如图：把本地的 master 分支修改 推送到远程仓库对应的分支

```
MINGW64:/d/widon/learn_project/git-demo
SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git push origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 6 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 466 bytes | 155.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To http://10.142.146.37:8081/common/git-demo.git
* [new branch]      master -> master
```

推送后，仓库中有了文件信息



7 比较不同

另外在我们执行 `git diff` 操作时后面添加不同的参数，会得到不同的结果，这是因为比较的两个对象可能不相同，如：

`git diff` 不加参数，比较的是暂存区和工作区的数据。

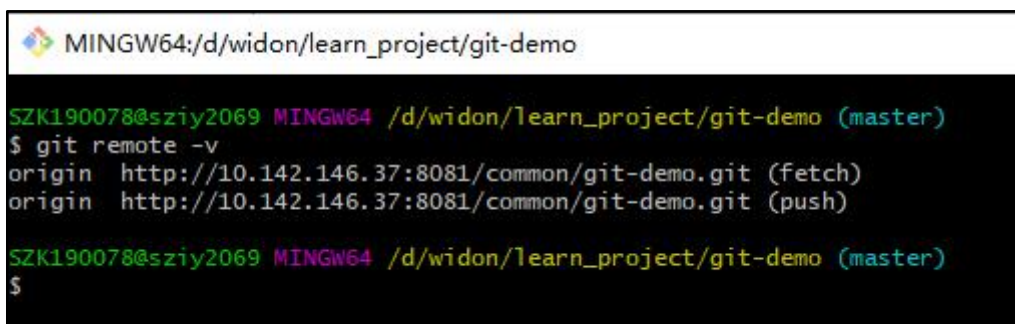
`git diff HEAD <file>` 比较的是版本库和工作区的数据

...其他很多选项

8 本地仓库和项目更新

8.1 查看本地仓库对应的远程仓库信息

`git remote -v`

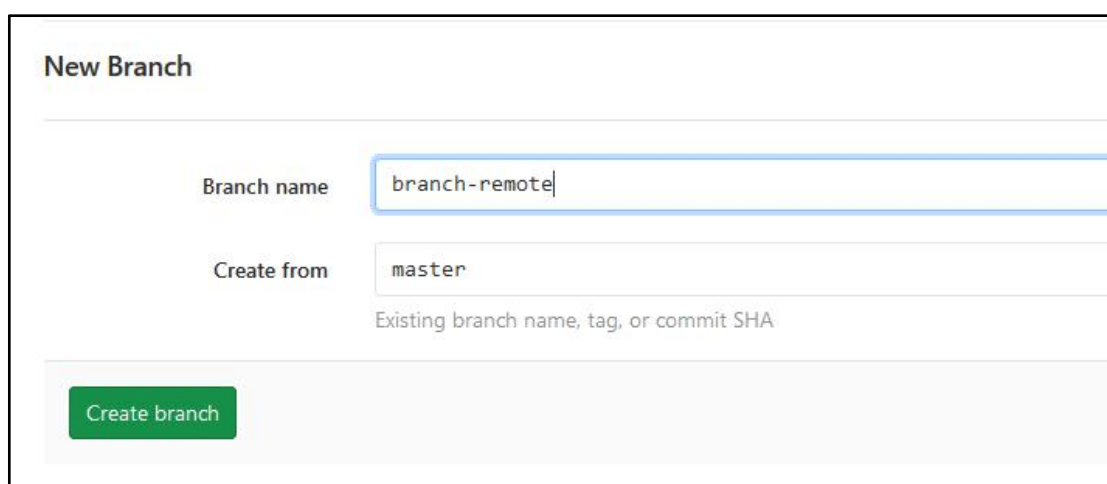
A terminal window with a black background and white text. The title bar shows 'MINGW64:/d/widon/learn_project/git-demo'. The prompt is 'SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)'. The command 'git remote -v' has been executed, showing two entries: 'origin http://10.142.146.37:8081/common/git-demo.git (fetch)' and 'origin http://10.142.146.37:8081/common/git-demo.git (push)'. The prompt returns to '\$'.

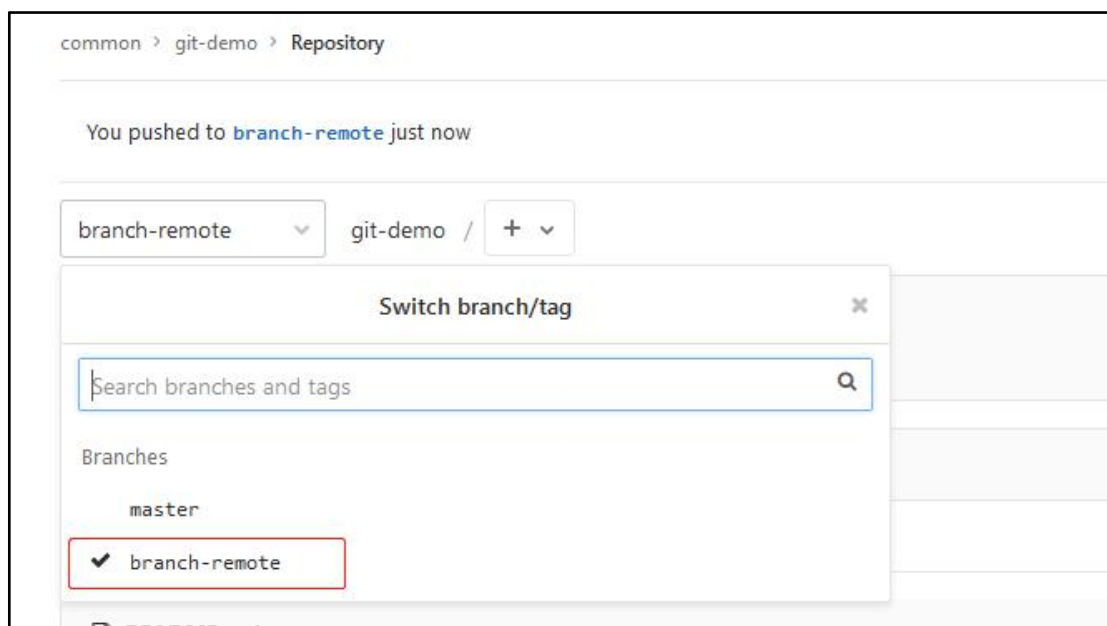
8.2 本地仓库更新-不会自动合并

`git fetch origin`

比如远程库 origin 增加了新的分支，这个信息本地仓库不知道
例：

假如现在远程仓库增加了分支，branch-remote

A web form titled 'New Branch'. It has two input fields: 'Branch name' with the value 'branch-remote' and 'Create from' with the value 'master'. Below the 'Create from' field is a small text label 'Existing branch name, tag, or commit SHA'. At the bottom left is a green button labeled 'Create branch'.



本地的仓库关于分支的信息还是克隆时候的状态，没有远程分支 `branch-remote` 的信息

```
SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch -a
* master
  remotes/origin/master
```

用 `git fetch` 命令获取最新的远程仓库信息到本地，有了这个远程分支的信息，可以建立本地分支。

```
SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git fetch origin
From http://10.142.146.37:8081/common/git-demo
* [new branch]      branch-remote -> origin/branch-remote

SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch -a
* master
  remotes/origin/branch-remote
  remotes/origin/master
```

8.3 本地仓库某个具体分支代码更新

当远程仓库的某个分支修改后，想把修改信息更新到本地对应的分支

```
git pull origin <branch>
```


9 解决合并冲突

当多人同时对一个文件的相同位置进行修改,其他人已经提交到远程仓库,自己执行更新操作,这个时候会产生冲突,冲突机制与 SVN 原理一致,至于什么情况下产生冲突,可以自行研究。

当出现冲突,文件中会显示类似于下面的信息

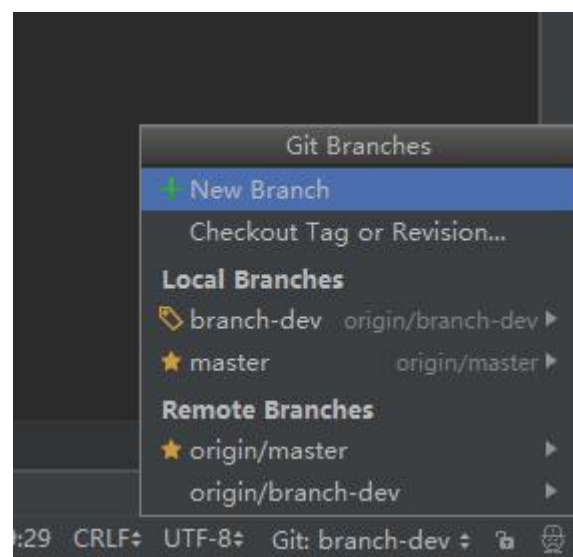
```
a123
<<<<<<< HEAD
b789
=====
b45678910
>>>>>>> 6853e5ff961e684d3a6c02d4d06183b5ff330dcc
cccccccccccccc
```

其中:冲突标记<<<<<<< HEAD (7个<)与=====之间的内容是自己的修改,=====与>>>>>>>之间的内容是别人的修改。当修改冲突时,最好和产生冲突的对方一起确定后,去掉错误的、过时的代码,保留正确的代码。然后增加到暂存区,提交到本地仓库,最后推送到远程仓库。

10 分支相关

10.1 查看分支信息

10.1.1 插件操作



10.1.2 命令行操作

git branch 查看本地分支

git branch -r 查看远程分支

git branch -a 查看所有的分支信息

```
SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch
* master

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch -r
  origin/branch-remote
  origin/master

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch -a
* master
  remotes/origin/branch-remote
  remotes/origin/master
```

10.2 新建分支

10.2.1 基于本地分支新建

在开发的过程中,想基于目前的开发分支新建一个分支用于开发新的特征或保存当前分支的状态,则在当前分支中,使用

git branch newBranchName

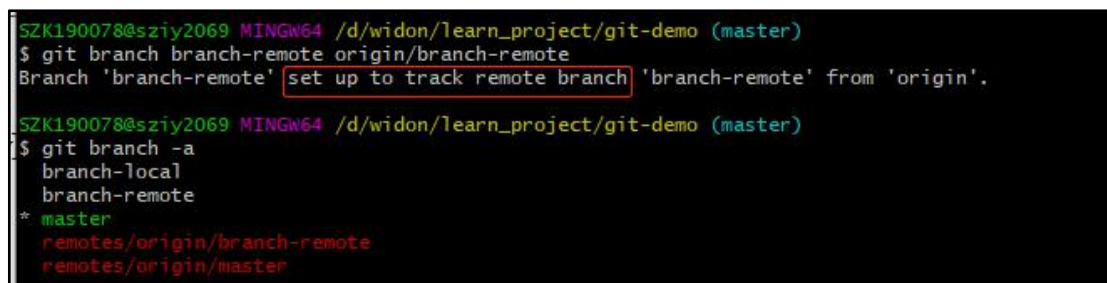
```
SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch branch-local

SZK190078@szy2069 MINGW64 /d/widon/learn_project/git-demo (master)
$ git branch
  branch-local
* master
```

注意: 当前工作空间仍在 master 分支上,注意分支名是绿色的,前面有星号。如果想让工作空间切换到新建的分支,可用 **git checkout** 命令,后面会讲。

10.2.2 基于远程仓库分支新建

```
git branch newBranchName origin/branch
```



A terminal window showing the execution of git commands. The first command is `git branch branch-remote origin/branch-remote`, which outputs `Branch 'branch-remote' set up to track remote branch 'branch-remote' from 'origin'.`. The second command is `git branch -a`, which lists the local branches (`branch-local`, `branch-remote`, `* master`) and the remote branches (`remotes/origin/branch-remote`, `remotes/origin/master`).

10.2.3 新建分支推送到远程仓库

```
git push orgin newBranch
```

10.3 删除分支

10.3.1 删除本地分支

```
git branch -d dev
```

10.3.2 删除远程分支

```
git push origin :branch
```

相当于推送一个空分支到远程的分支

10.4 分支版本合并

这个仓库一般有项目小组的组长进行操作，当组员开发的功能的功能完成后,此时合并到开发版本 brach-dev,然后上到测试,此时可能产生冲突，解决和上面的一致

```
git merge other-branch 常用
```

版本合并还有

```
git rebase other-branch
```

两者的区别自己可以研究。上面的那个较常用，下面的命令容易出错，非专业人士勿用。

10.5 分支版本切换

10.5.1 本地分支切换

```
git checkout target-branch //工作空间从当前分支切换到目的分支
```

10.5.2 基于本地当前分支新建分支并切换

```
git checkout -b new-branch //新建分支,工作空间切换到新建的分支
```

10.5.3 基于远程仓库的分支新建分支并切换

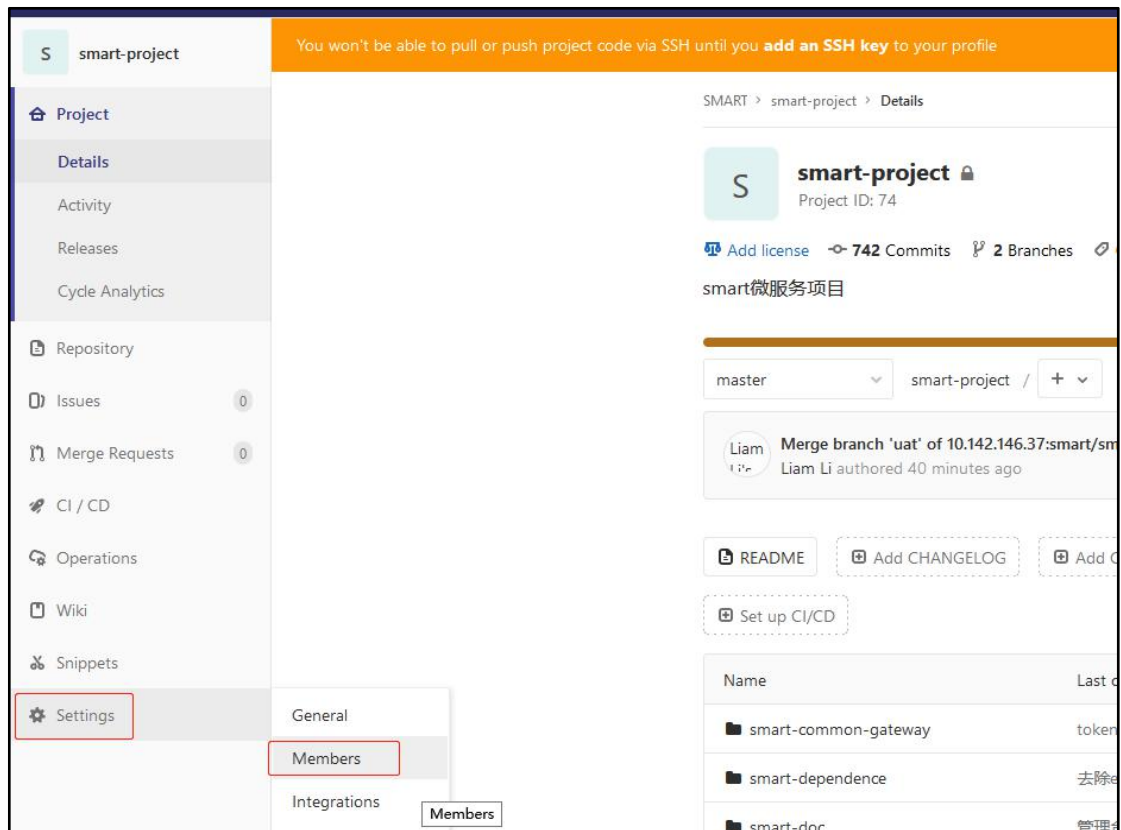
```
git checkout -b new-branch origin/branch
```

11 本地已有项目 push 到远程仓库

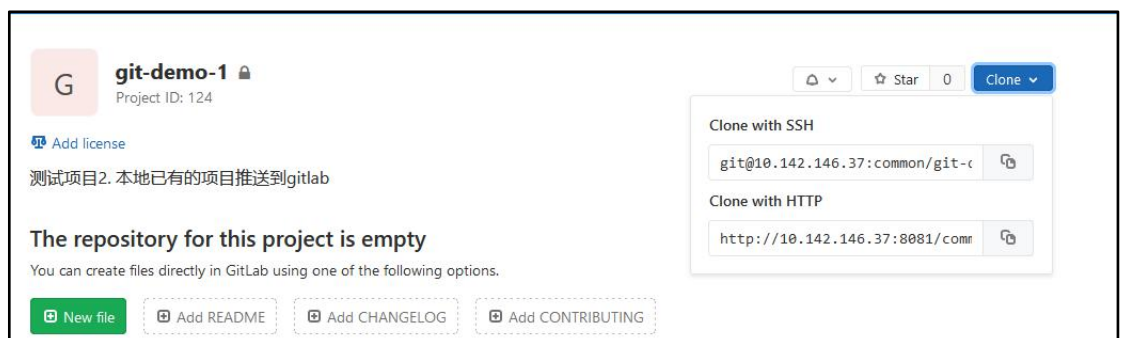
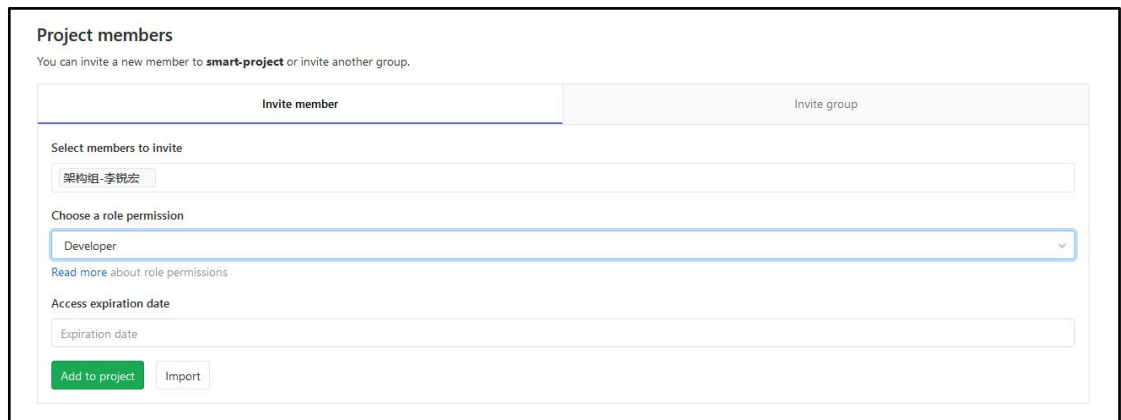
现在本地有正在开发的项目，想推送到远程对应的项目。

11.1 向项目组长申请新项目接入

项目组长在 gitlab 新建一个项目，把申请人加入项目中，如果申请者已经拥有小组权限，不用再重复添加。



把申请者加入项目中



11.2 初始化本地仓库

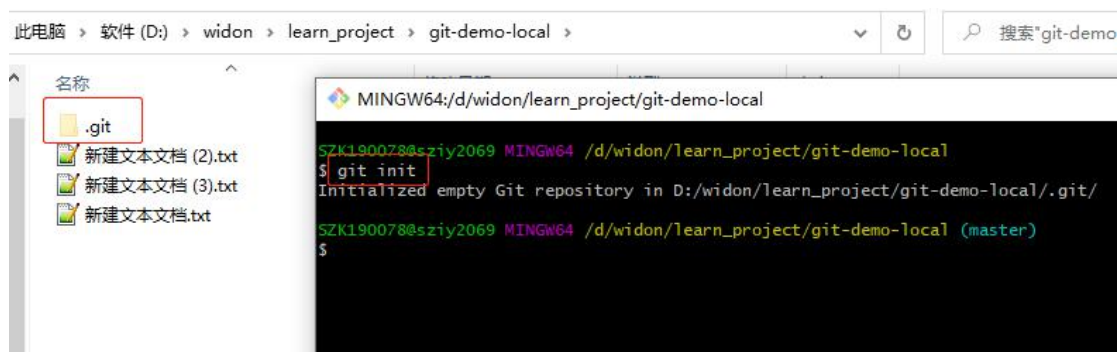
在要上传的项目目录中允许初始化命令，会生成一个 **.git** 文件夹

git init //初始化仓库命令

本地项目最初的状态



初始化后



11.3 把本地项目提交到本地仓库

第一步，是把所有的文件加入到暂存区(index)

git add .

第二步，把暂存区的所有文件提交到本地仓库

git commit -m”此时提交注释”

11.4 本地仓库添加远程仓库地址

```
git remote add origin http://10.142.146.37:8081/common/git-demo-1.git
```

11.5 更新本地仓库信息

因为管理员建立项目的时候，可能有其他的文件，此时要把这些信息先更新下来

```
git pull origin master
```

注：如果管理员建立的是空项目，此步可以跳过

11.6 本地仓库信息推送远程仓库

```
git push origin master
```

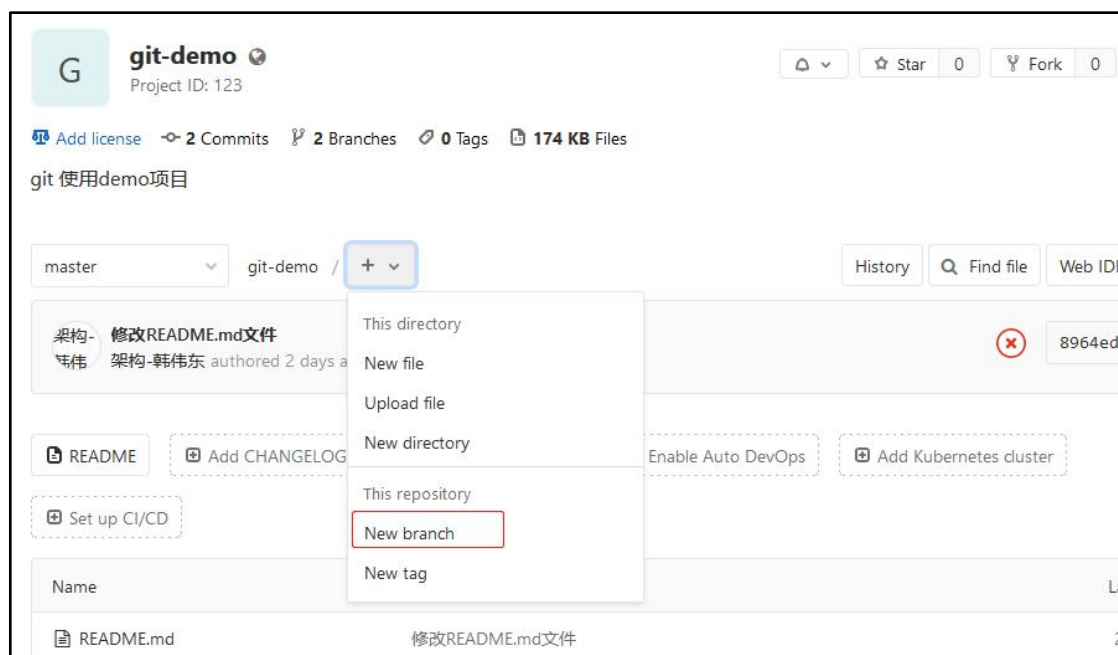
12 如何理解分支开发

此处说的分支开发主要是开发分支(branch-dev)开发，其他的分支开发类似，**项目克隆下来默认是 master 分支**，因为主版本(master)普通开发人员没有权限推送更新的。**注意！！**如果远程已经存在开发分支(origin/branch-dev),则基于远程的分支新建 branch-dev,而不是不是基于本地的 master 自己新建 branch-dev,如果远程不存在 branch-dev,则在本地基于 master 分支新建并推送到远程。**分支开发是之后版本管理的基础**，下面就**插件**和**命令行**详细讲解一遍

12.1 项目小组长准备工作

小组长最开始推送项目到 GitLab 时，不仅要推送 master 分支，也要基于 master 分支新建开发分支 branch-dev(本地新建分支，也可以直接在 GitLab 进行操作)这样其他的开发小组成员才能在这个开发分支 branch-dev 上开发。

注:在 GitLab 页面上新建远程分支，这样快些



New Branch

Branch name:

Create from: master → 基于master新建分支

Existing branch name, tag, or commit SHA

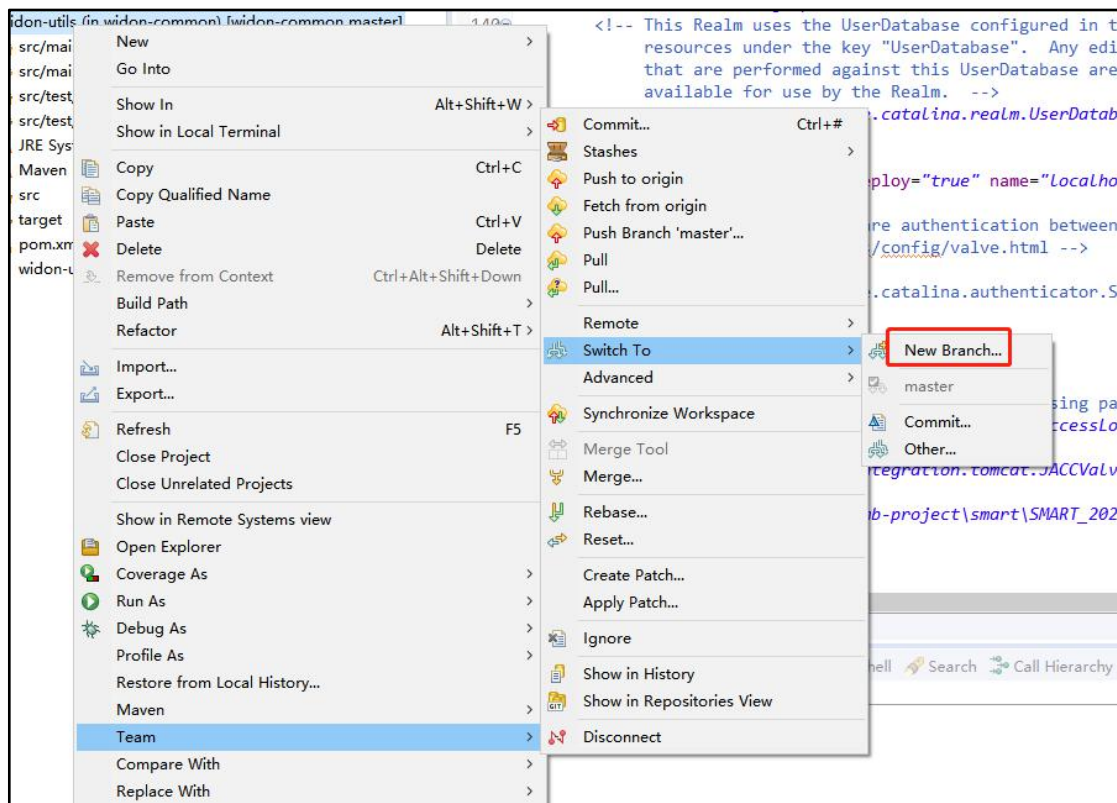
Create branch

12.2 利用 Eclipse 插件切换

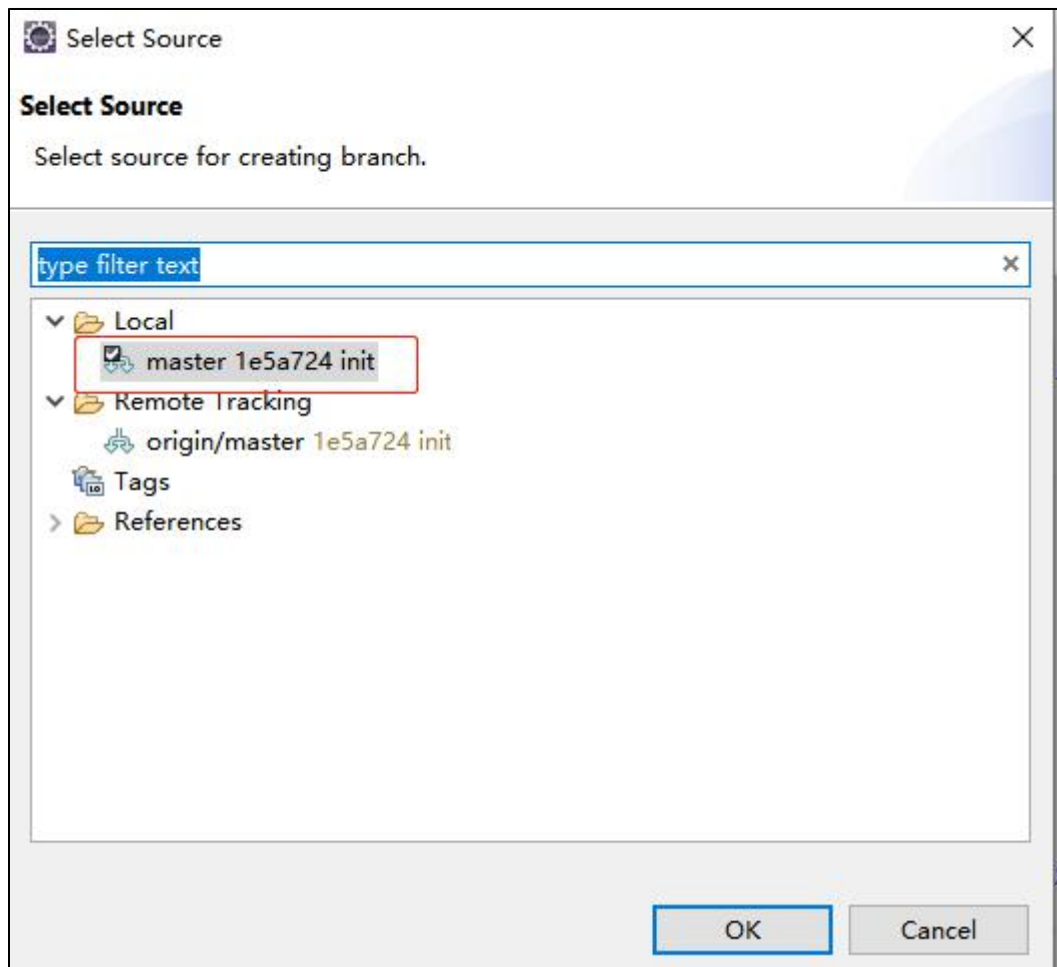
12.2.1 克隆下来默认是 master



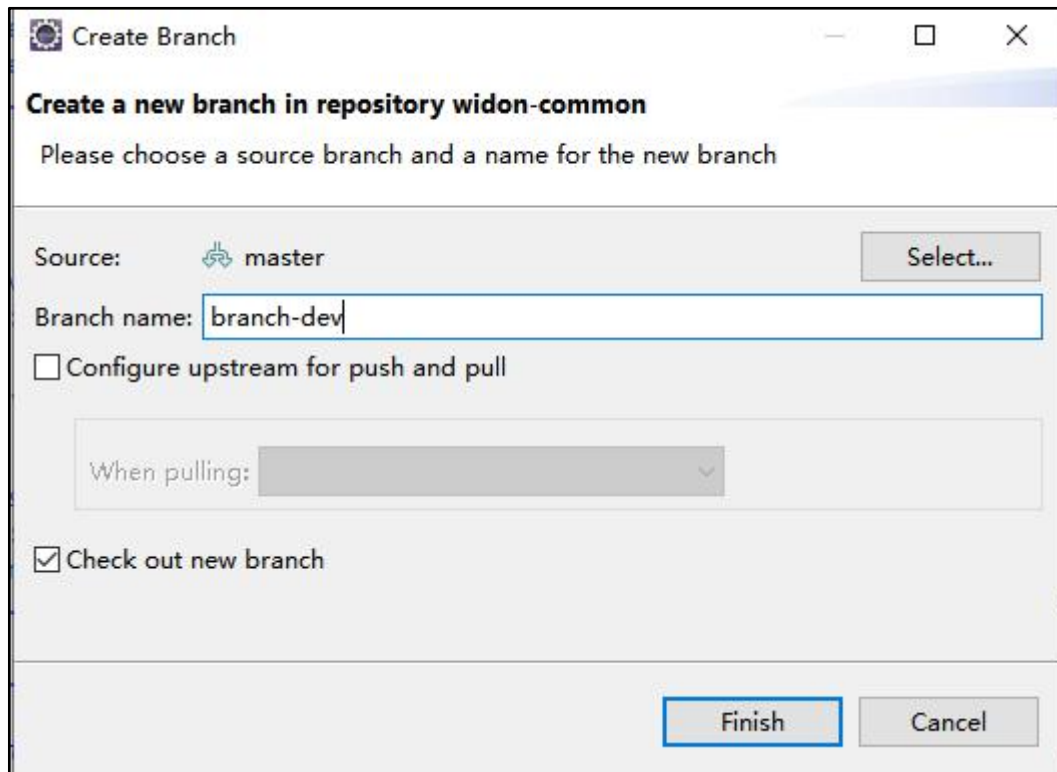
12.2.2 右键项目-->team



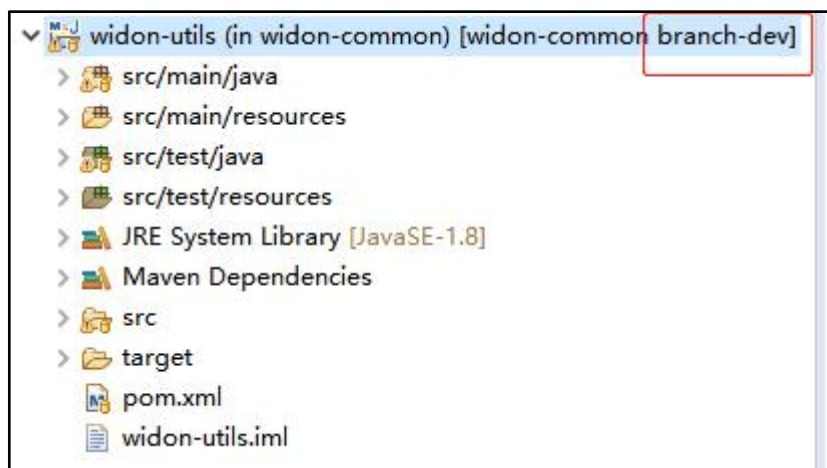
12.2.3 选择新建分支的原分支



12.2.4 点击完成



工作区已经切换到开发分支了(branch-dev)



12.3 利用命令行进行切换到开发分支

12.3.1 项目克隆下来后默认只有 master 分支

```
SZK190078@szziy2069 MINGW64 /d/widon/learn_project/widon-common (master)
$ git branch -a
* master
remotes/origin/branch-dev
remotes/origin/master
```

12.3.2 用命令基于远程分支新建本地开发分支

[10.2.2 基于远程仓库分支新建](#)

[10.5.3 基于远程仓库的分支新建分支并切换](#)

git branch newBranchName origin/branch

git checkout -b new-branch origin/branch

13 忽略文件.gitignore 文件配置

在项目根目录，也就是 .git 文件夹统计目录中，新建一个 .gitignore 文件，内容如

```
### IntelliJ IDEA ###
```

```
.idea
```

```
*.iws
```

```
*.iml
```

```
*.ipr
```

```
.mvn/
```

```
### Eclipse ###
```

```
/build/
```

```
target/
```

```
.classpath
```

```
.project
```

```
.settings/
```

```
.DS_Store
```

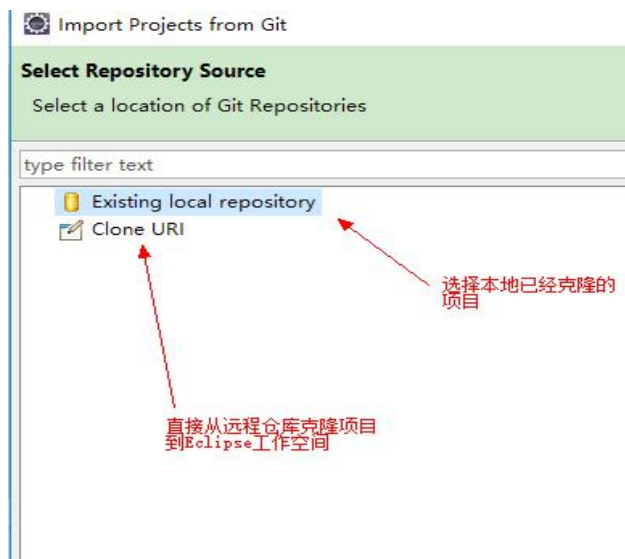
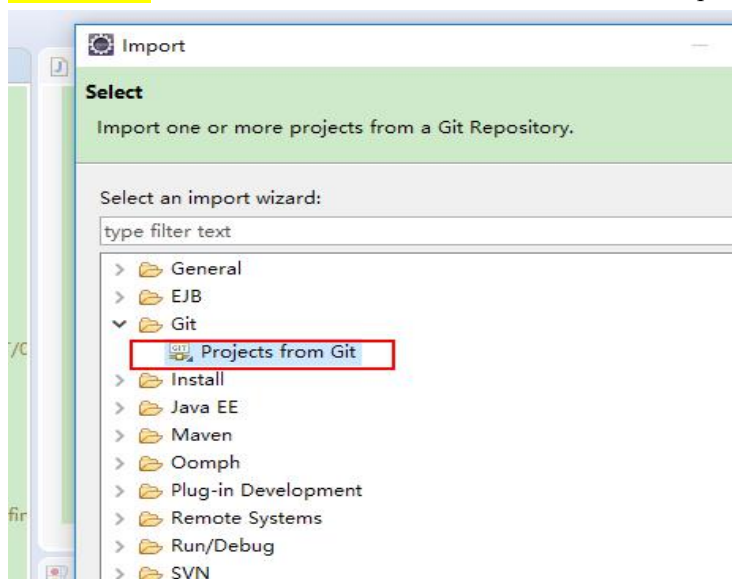
```
*~
```

14 插件操作

Eclipse 插件自动集成，只需要直接使用即可。目前插件是英文，如果觉得不方便，可以自己下载中文版本。

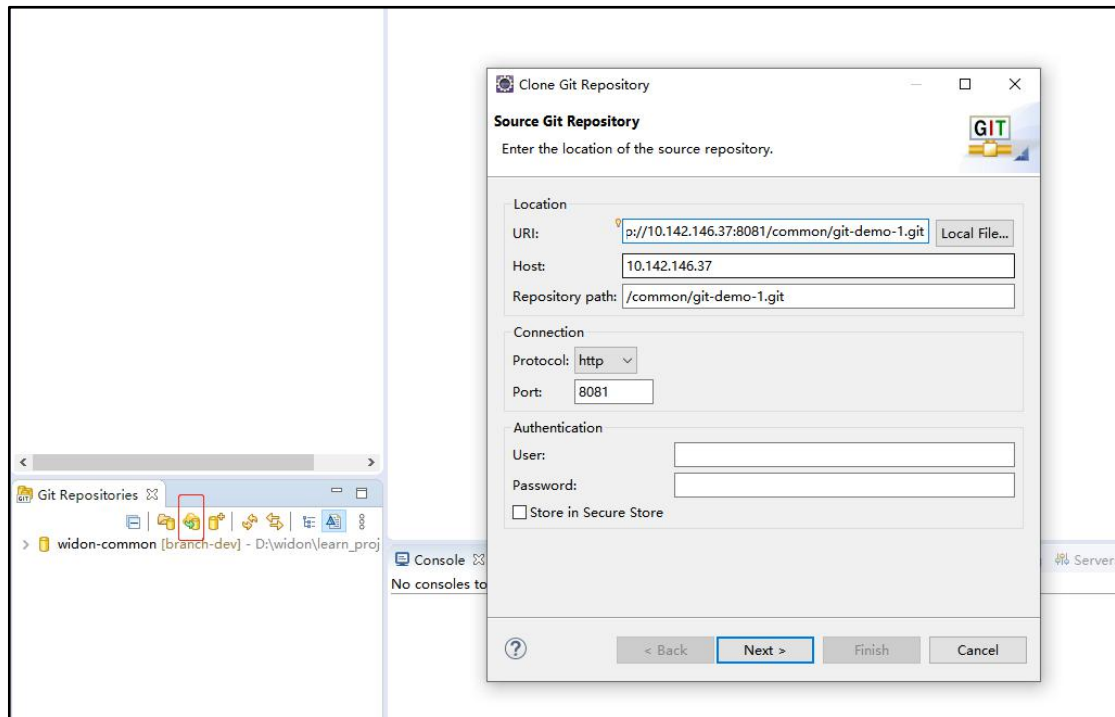
14.1 Eclipse 导入 GitLib 项目-法一

先用命令把项目 **clone** 到本地,然后用 Eclipse 导入,选择通过 **Git 方式** 导入项目或者通过 **maven 项目** 导入，通过 Maven 方式导入的话不能用进行 pull 等操作

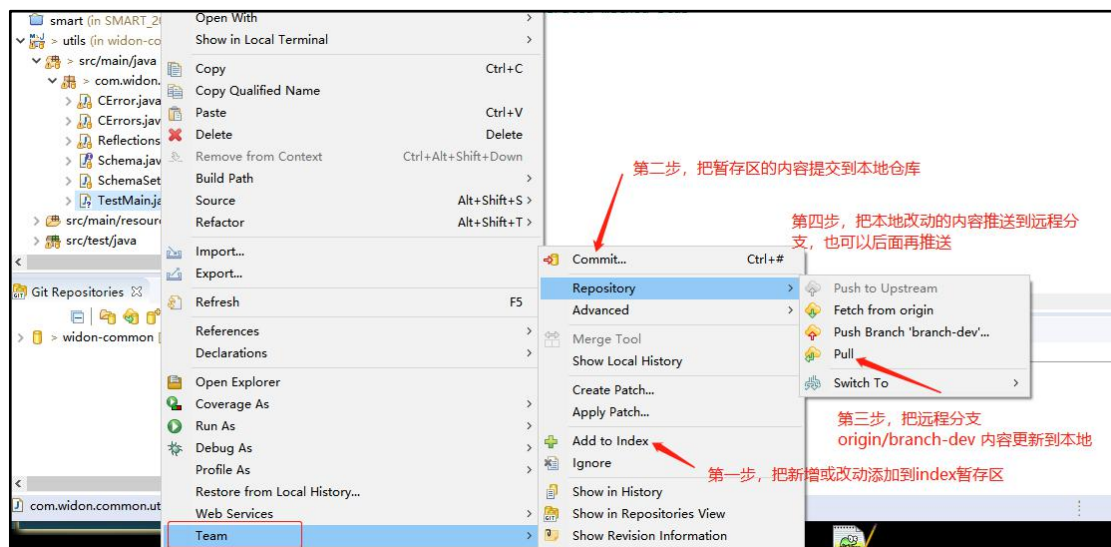


14.2 Eclipse 导入 GitLib 项目-法二

从 GitLib 中取得项目代码。菜单栏 Window -> Show View -> Git -> GitRepositories, 点击 OK; 在 Git Repositories 视图中, 点击图示按钮, 进入 Clone Git Repository 对话框; 填写 URI, User, Password, 点击 Next; 点击 Next; 选择要保存的路径, 点击 Finish。至此, 项目代码就被下载到指定目录下了, 需要使用的时候通过 Eclipse Import 进来即可。

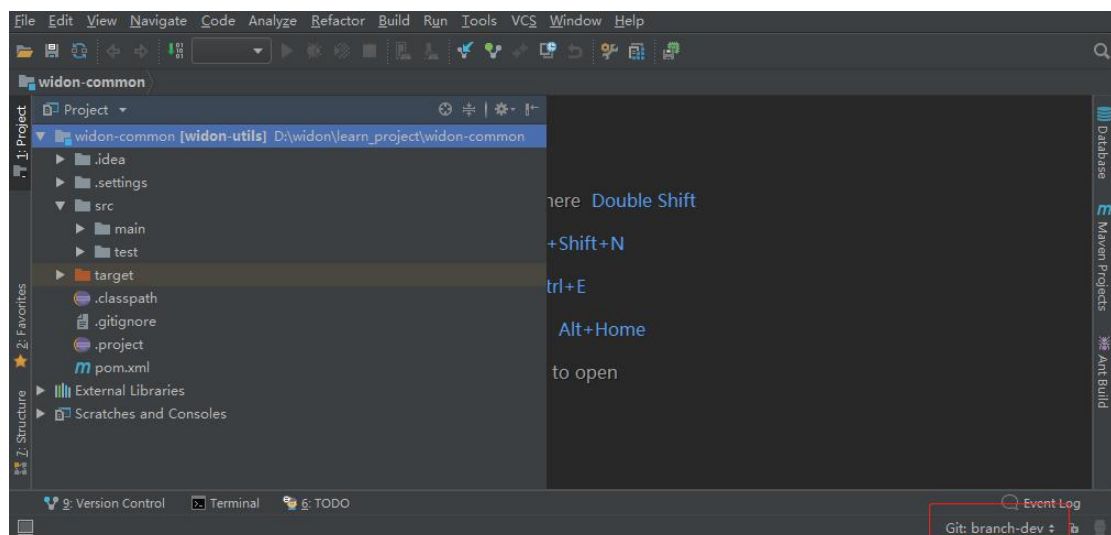


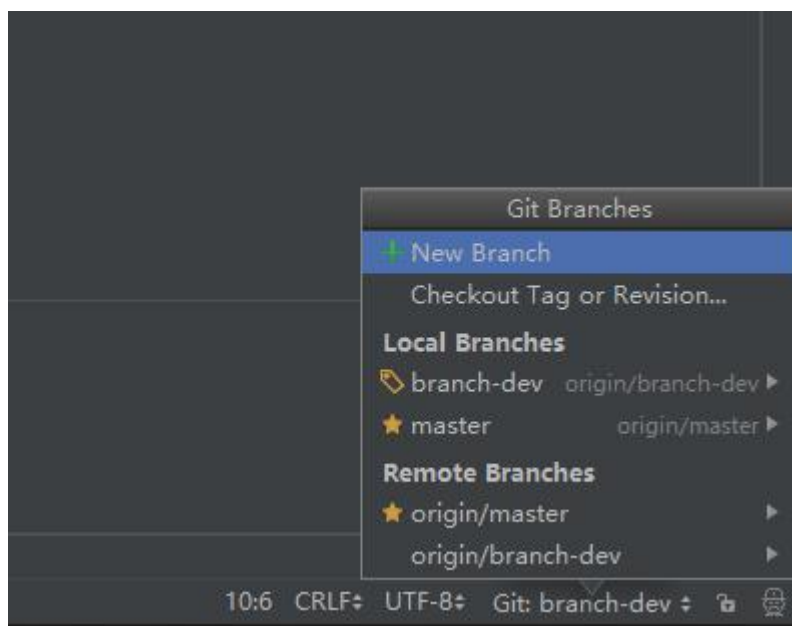
14.3 插件操作-Eclipse



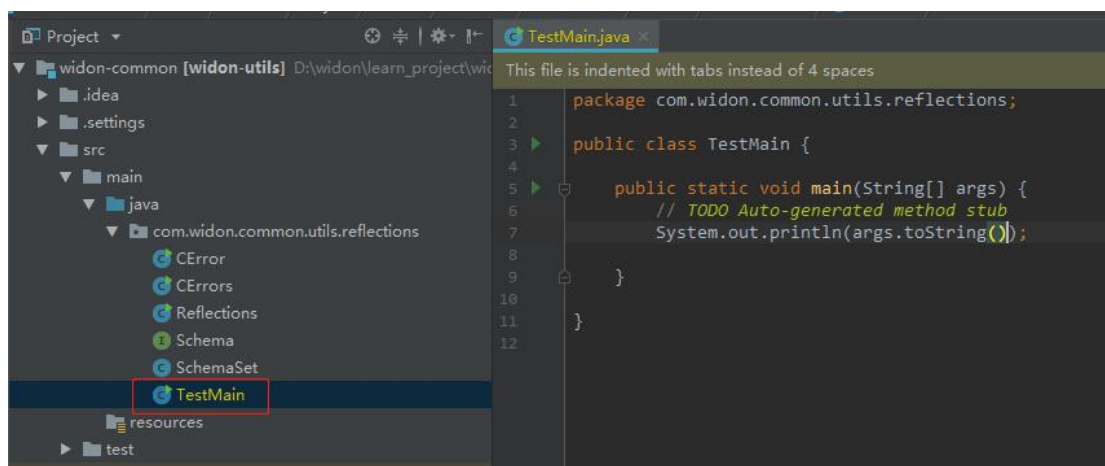
14.4 插件操作-Idea

查看分支

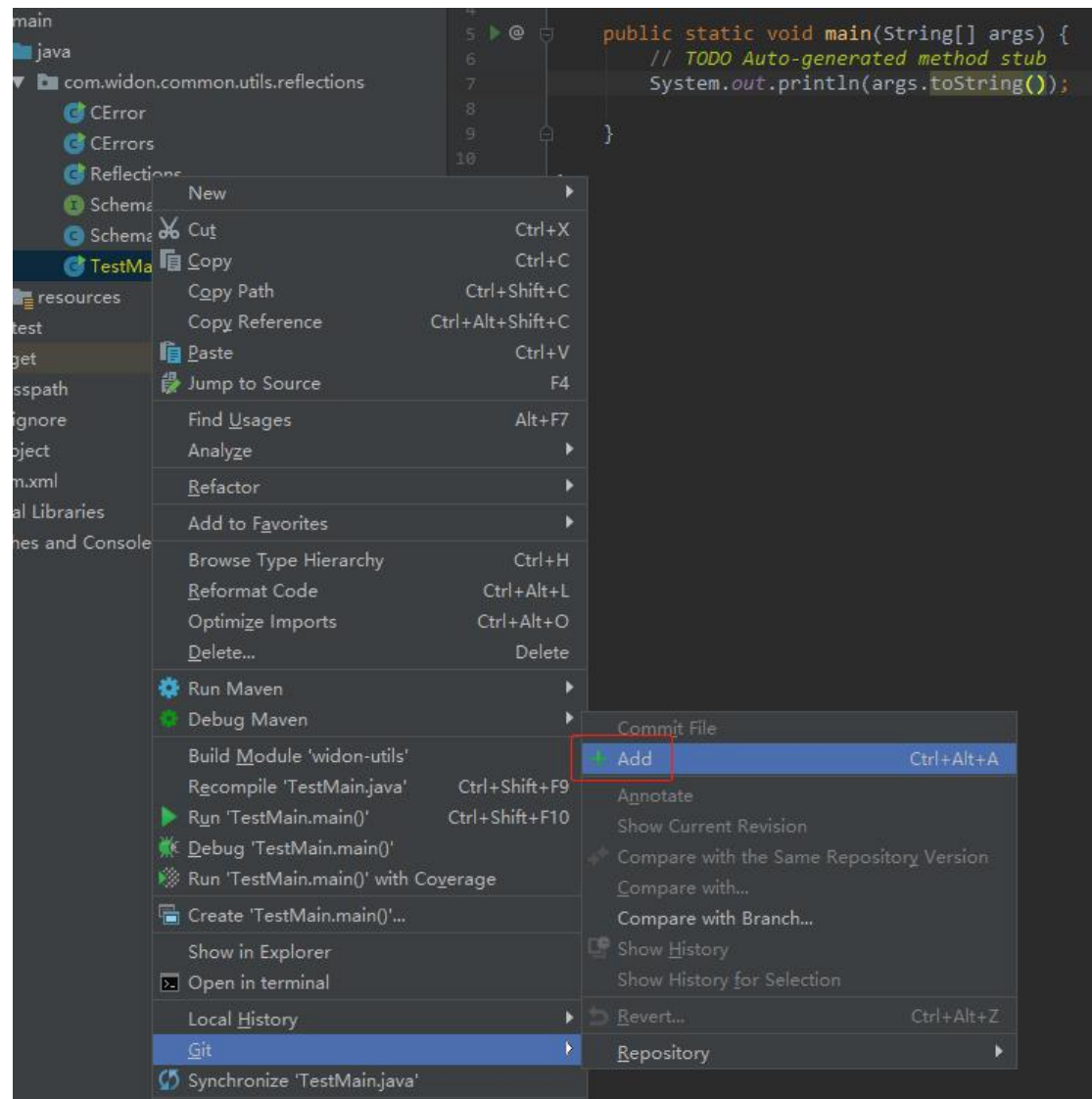




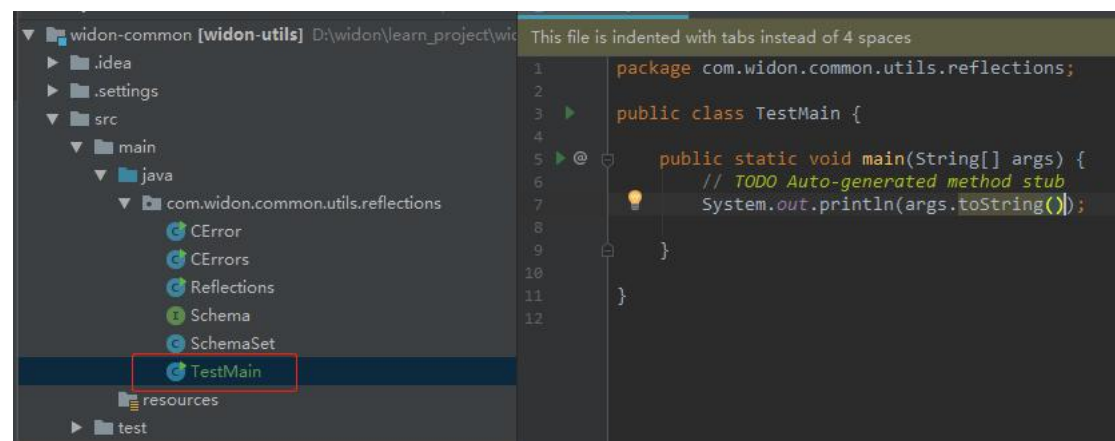
内容修改后 文件呈 亮黄色



文件修改后提交到缓存区



Add 添加到缓存区后呈 绿色

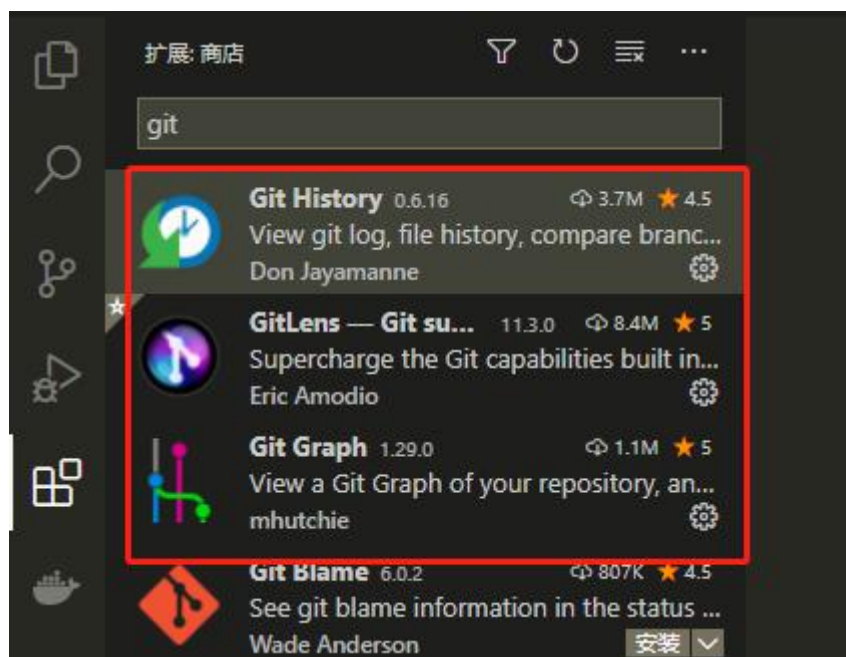


14.5 是否需要提交更新标志



14.6 Visula Studio Code（目前主要前端开发人员使用）相关插件

搜索 git 并安装一下插件：



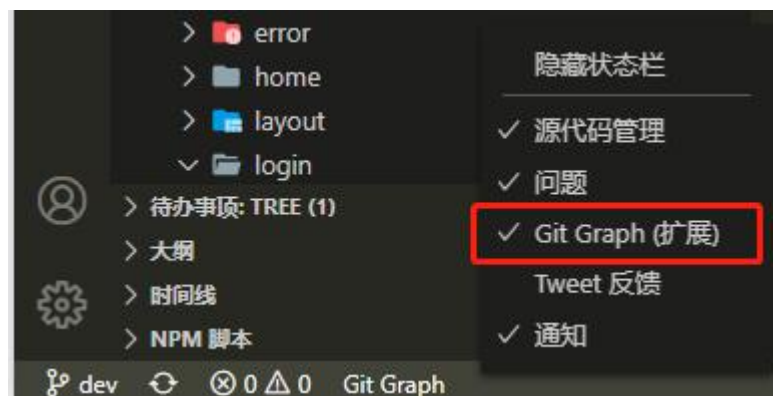
Git History: 主要用于查看 Git 历史记录，搜索和更多内容（包括 git log）

GitLens: 增强了 VSCode 的 Git 内置功能

Git Graph: 图形化展示 git 仓库，并包涵 git 相关操作

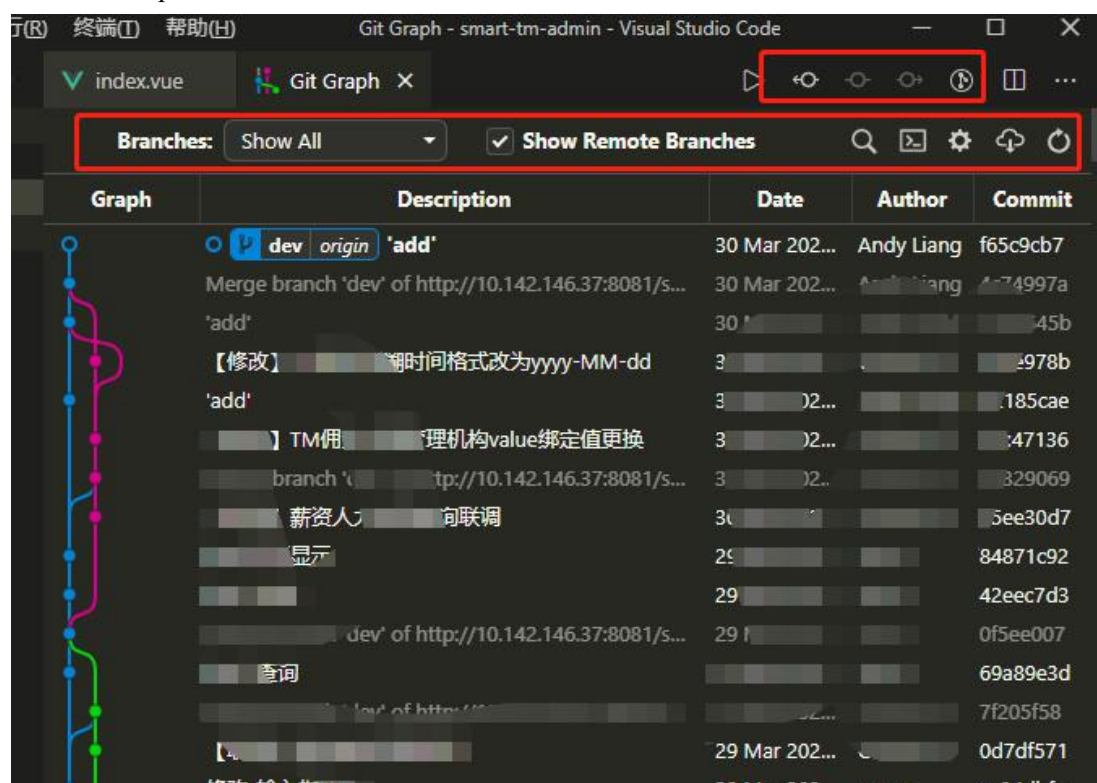
安装完成后在底部状态栏有相关显示





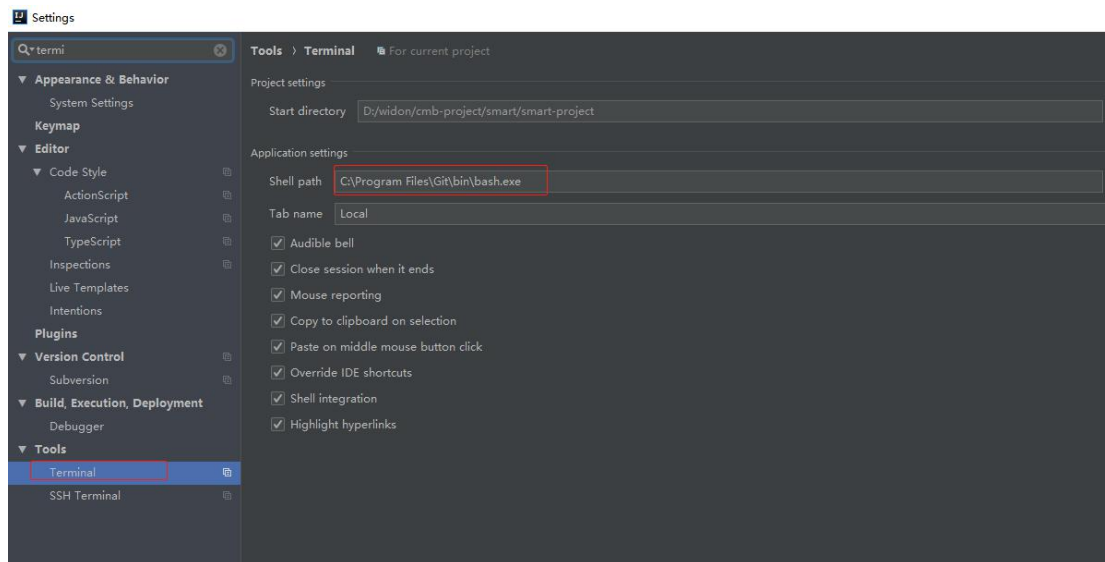
如果没有，请右键选择

点击 Git Graph，展示如下：



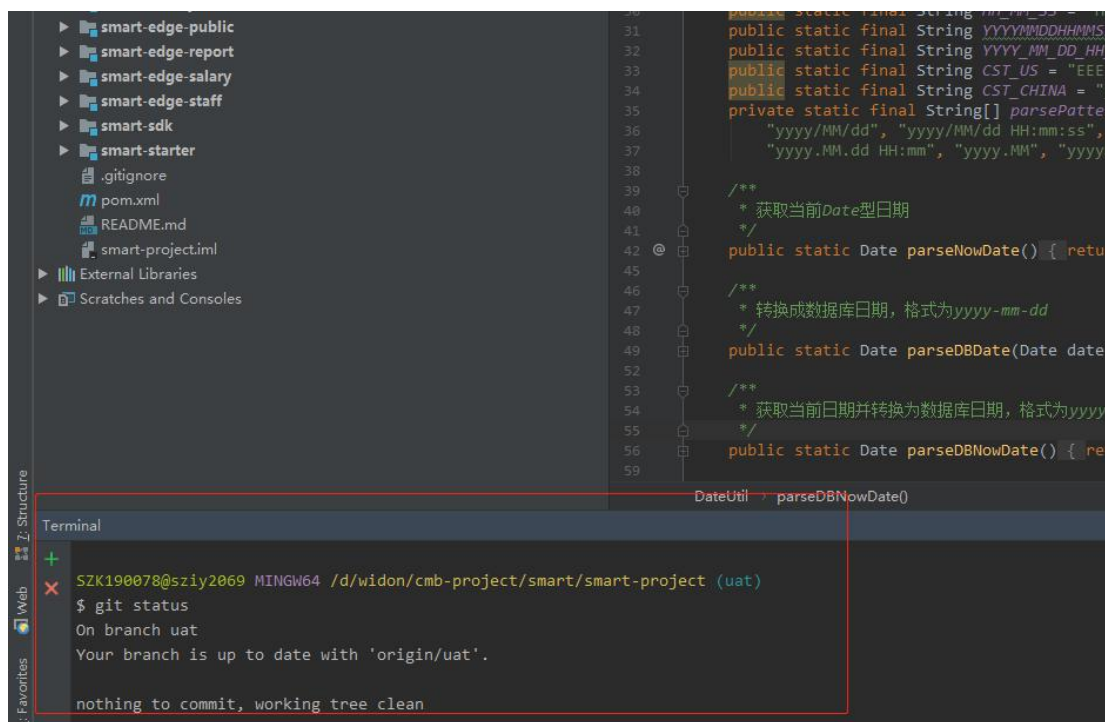
15 IDEA 中 Terminal 集成 Git

15.1 设置中添加 git 命令执行路径



15.2 快速进入 Terminal 窗口

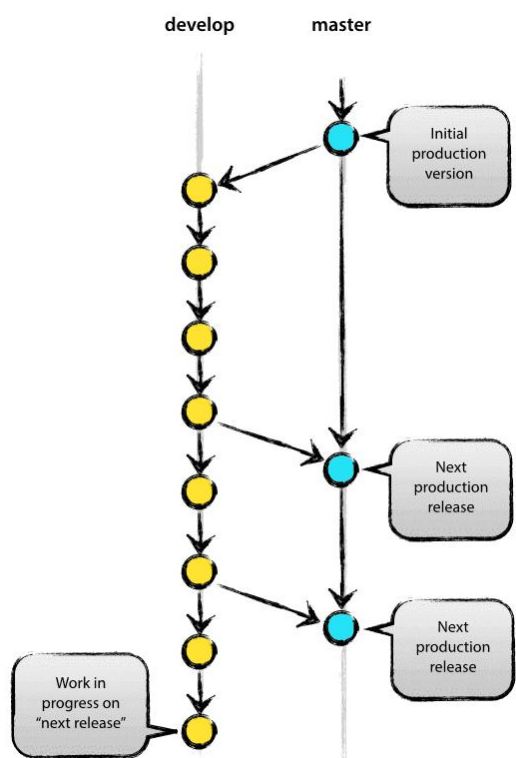
Alt+F12 快捷键，快速打开 Terminal 窗口，进入命令行 git 操作



16 使用 Git 做分支管理

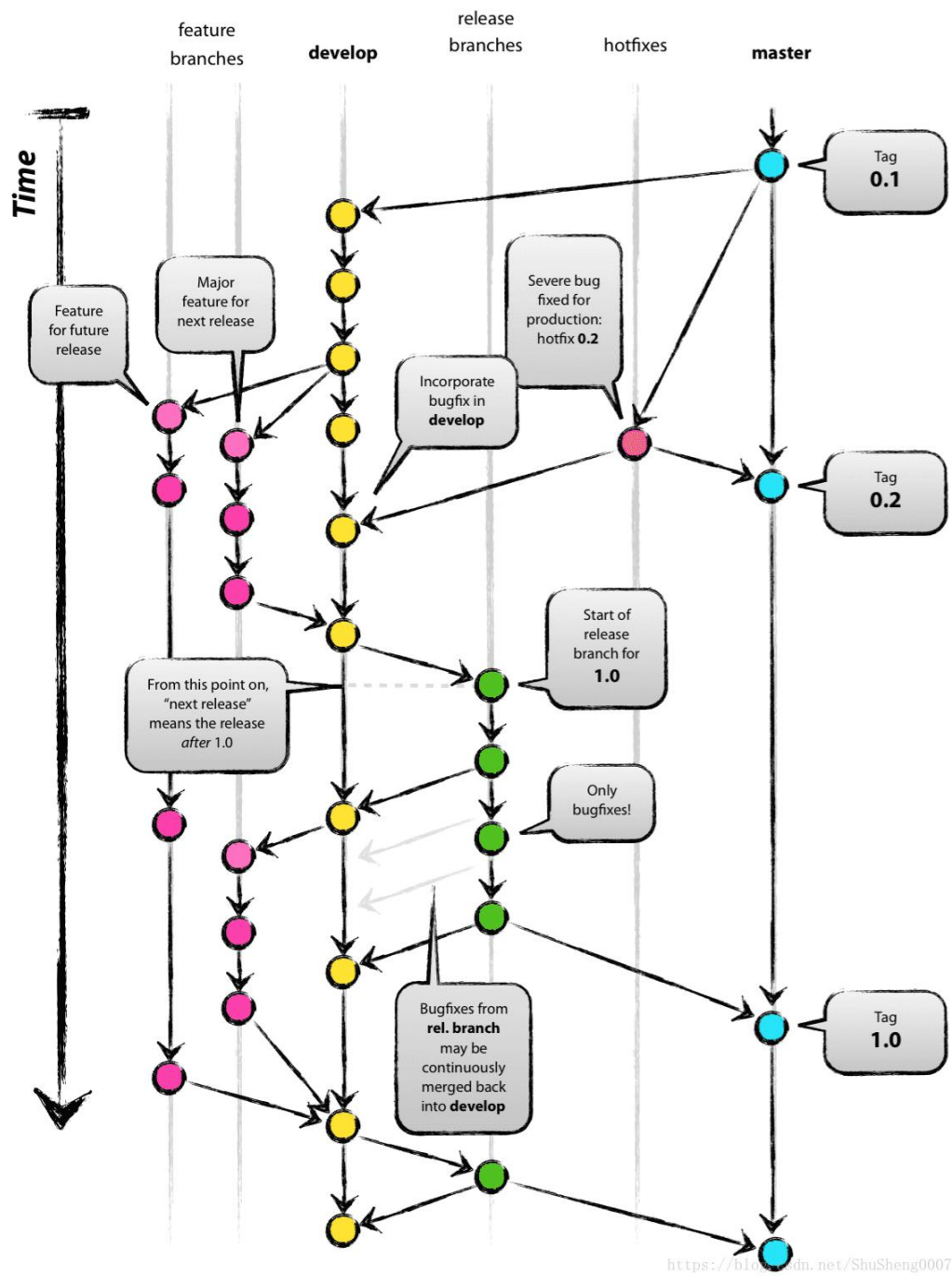
<https://blog.csdn.net/ShuSheng0007/article/details/80791849>

16.1 目前现状



<https://blog.csdn.net/ShuSheng0007>

16.2 最终目标



16.3 版本命名规范

原文链接：https://blog.csdn.net/weixin_42288219/article/details/112290998

软件版本号有四部分组成，第一部分为主版本号，第二部分为次版本号，第三部分为修订版本号，第四部分为日期版本号加希腊字母版本号，希腊字母版本号共有五种，分别为 **base**、**alpha**、**beta**、**RC**、**release**。

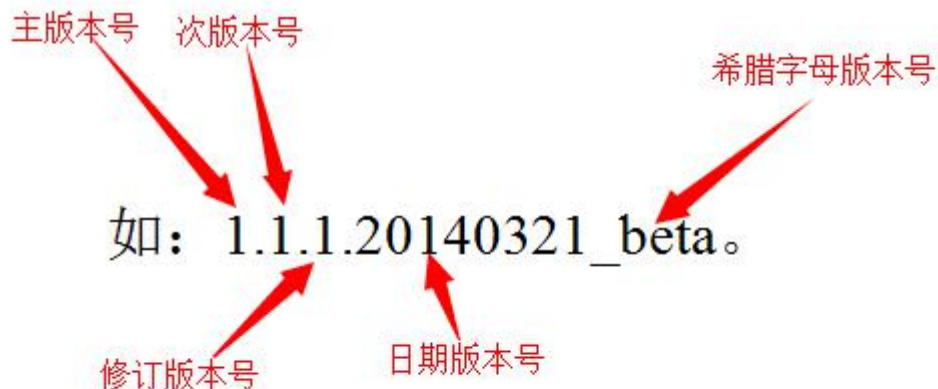
主版本号：当功能模块有较大的变动，比如增加模块或是整体架构发生变化。此版本号由项目决定是否修改。

次版本号：相对于主版本号而言，次版本号的升级对应的只是局部的变动，但该局部的变动造成程序和以前版本不能兼容，或者对该程序以前的协作关系产生了破坏，或者是功能上有大的改进或增强。此版本号由项目决定是否修改。

修订版本号：一般是 Bug 的修复或是一些小的变动或是一些功能的扩充，要经常发布修订版，修复一个严重 Bug 即可发布一个修订版。此版本号由项目经理决定是否修改。

日期版本号：用于记录修改项目的当前日期，每天对项目的修改都需要更改日期版本号。此版本号由开发人员决定是否修改。

希腊字母版本号：此版本号用于标注当前版本的软件处于哪个开发阶段，当软件进入到另一个阶段时需要修改此版本号。此版本号由项目决定是否修改。



https://blog.csdn.net/weixin_42288219

Base：此版本表示该软件仅仅是一个假页面链接，通常包括所有的功能和页面布局，但是页面中的功能都没有做完整的实现，只是做为整体网站的一个基础架构。

Alpha：软件的初级版本，表示该软件在此阶段以实现软件功能为主，通常只在软件开发者内部交流，一般而言，该版本软件的 Bug 较多，需要继续修改，是测试版本。测试人员提交 Bug 经开发人员修改确认之后，发布到测试网址让测试人员测试，此时可将软件版本标注为 alpha 版。

Beta：该版本相对于 Alpha 版已经有了很大的进步，消除了严重错误，但仍需要经过多次测试来进一步消除，此版本主要的修改对象是软件的 UI。修改的 Bug 经测试人员测试确认后可发布到外网上，此时可将软件版本标注为 beta

版。

RC：该版本已经相当成熟了，基本上不存在导致错误的 Bug，与即将发行的正式版本相差无几。

Release：该版本意味"最终版本"，在前面版本的一系列测试版之后，终归会有一个正式的版本，是最终交付用户使用的一个版本。该版本有时也称标准版。

16.4 版本修改举例说明

如此时版本号为：1.0.0.0321_alpha，此时为内部测试阶段。

开发人员修复了测试人员提交的 bug 并经测试人员测试验证关闭 bug 之后，发布到外网时，此时就进入了软件的下一个阶段，版本号可改为：1.0.0.0321_beta，如当前日期跟上一个版本号的日期不一样，版本号可改为：1.0.0.0322_beta。

如果修复了一些重大 Bug 并按照流程发布到外网时就可发布一个修订版，如 1.0.1.0322_beta，日期为发布的当前日期。

如果对软件进行了一些功能上的改进或增强，进行了一些局部变动的时候要修改次版本号，如：1.1.0.0322_beta（上一级有变动时，下级要归零）。

当功能模块有较大变动，增加模块或整体架构发生变化时要修改主版本号，如新增加了退款功能，则版本号要改为：2.0.0.0322_beta。

原文链接：https://blog.csdn.net/weixin_42288219/article/details/112290998

17 GitLab 中代码评审机制

基本思想：组员 develop 提交的代码需要 master 评审后，通过才可以合并到指定分支

17.1 设置用户权限

Expiration date

添加到项目

导入

现有的成员和群组

demo1的成员

按名称查找现有成员

名称, 升序排列

<div>Administrator @root · common</div> <div>4天前授权访问</div> <div>Owner</div>
<div>Jeck @Jeck · common</div> <div>1天前授权访问</div> <div>Maintainer</div>
<div>保险平台-张振军 @john.zhang · common</div> <div>1天前授权访问</div> <div>Maintainer</div>
<div>前端架构-罗勇 @rodey · common</div> <div>1天前授权访问</div> <div>Maintainer</div>
<div>架构组-李锐泰 @liam.li · common</div> <div>2天前授权访问</div> <div>Developer</div>
<div>架构-韩伟东 @widoon.han · common</div> <div>2天前授权访问</div> <div>Developer</div>
<div>团队-古仕磊 @aaron.gu · common</div> <div>1天前授权访问</div> <div>Maintainer</div>
<div>团队渠道组-田磊峰 @stephen.tian · common</div> <div>1天前授权访问</div> <div>Maintainer</div>

17.2 设置分支保护

Protect a branch

Branch:

Select branch or create wildcard

Wildcards such as `*-stable` or `production/*` are supported

Allowed to merge:

Select

Allowed to push:

Select

Protect

Protected branch (2)	Last commit	Allowed to merge	Allowed to push	
branch-dev	8964ed86 4天前	Maintainers	Maintainers	Unprotect
master default	8964ed86 4天前	Maintainers	Maintainers	Unprotect

17.3 基于 branch-dev 新建一个本地分支 feature-widon, 并推送到 Gitlab

```
SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-learn-demo/git-demo (branch-dev)
$ git checkout -b feature-widon-20210331
Switched to a new branch 'feature-widon-20210331'

SZK190078@szzy2069 MINGW64 /d/widon/learn_project/git-learn-demo/git-demo (feature-widon-20210331)
$ git push origin feature-widon-20210331
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 265 bytes | 132.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for feature-widon-20210331, visit:
remote: http://10.142.146.37:8081/common/demo1/merge_requests/new?merge_request%5Bsource_branch%5D=feature-widon-20210331
```

17.4 在 GitLab 中新建 merge request

common > demo1 > Details

You pushed to feature-widon-20210331 at common / demo1 just now

Create merge request

D demo1 Project ID: 137

0 Stars 0 Forks Clone

No license. All rights reserved 2 Commits 3 Branches 0 Tags 164 KB Files

demo1

master demo1 +

History Find file Web IDE

修改README.md文件 Commit pending 8964ed86

README Auto DevOps enabled

Name	Last commit	Last update
README	8964ed86	4 days ago

Title

承保功能开发完了

Start the title with `[WIP]` to prevent a Work In Progress merge request from being merged before it's ready.
Add description templates to help your contributors communicate effectively!

Description

WritePreview

承保功能1
承保功能2
承保功能3
承保功能4

请项目负责人XXX评审合并到branch-dev

Markdown and quick actions are supported

Attach a file

Assignee

Unassigned

Assign to me

Milestone

Milestone

Labels

Labels

Source branch

feature-widon-20210331

Target branch

branch-dev

Change branches

☐ Delete source branch when merge request is accepted.

☐ Squash commits when merge request is accepted.

Submit merge request

Cancel

17.5 项目负责人进行合并操作

demo1

项目

仓库

议题0

合并请求1

CI / CD

运维

Wiki

代码片段

设置

common > demo1 > Merge Requests > 11

OpenOpened 1分钟前 by架构-韩伟东EditClose 合并请求

承保功能开发完了

承保功能1 承保功能2 承保功能3 承保功能4

请项目负责人XXX评审合并到branch-dev

请求合并 feature-widon-20210331 入 branch-dev

在Web IDE中打开

检出分支

流水线 #255 等待中 使用提交 416a4a9c 于 feature-widon-20210331

当流水线成功时合并

删除源分支

1 次提交 和 1 个合并提交 将被源添加到 branch-dev. 修改合并提交

此合并请求可以手动合并, 请使用以下命令行

讨论0提交1流水线1变更1

显示所有活动

编辑预览

在此写评论或拖动您的文件到这里。

18 生产发布后版本标签 tag

tag 是 git 版本库的一个标记，指向某个 commit 的指针。

tag 主要用于发布版本的管理，一个版本发布之后，我们可以为 git 打上 v.1.0.1 v.1.0.2 ...这样的标签。

tag 感觉跟 branch 有点相似，但是本质上和分工上是不同的：

tag 对应某次 commit，是一个点，是不可移动的。

branch 对应一系列 commit，是很多点连成的一根线，有一个 HEAD 指针，是可以依靠 HEAD 指针移动的。

所以，两者的区别决定了使用方式，改动代码用 branch，不改动只查看用 tag。

18.1 创建 tag

```
git tag <tagName>           //创建本地 tag
```

```
git tag -a <tagName> -m "XXX..." //可以指定标签信息。
```

以上是基于本地当前分支的最后一个 commit 创建的 tag，但是如果不想以最后一个，只想以某一个特定的提交为 tag，也是可以的，只要你知道 commit 的 id。

```
git tag -a <tagName> <commitId>
```

创建 tag 是基于本地分支的 commit，而且与分支的推送是两回事，就是说分支已经推送到远程了，但是你的 tag 并没有，如果把 tag 推送到远程分支上，需要另外执行 tag 的推送命令。

```
git push origin <tagName> //推送到远程仓库
```

```
git push origin --tags //存在很多未推送的本地标签，你想一次全部推送的话
```

18.2 查看 tag

```
git show <tagName>           //查看本地某个 tag 的详细信息
```

```
git tag 或者 git tag -l       //查看本地所有 tag
```

```
git ls-remote --tags origin    //查看远程所有 tag
```

18.3 删除 tag

```
git tag -d <tagName>          //本地 tag 的删除
```

```
git push origin :<tagName>     //远程 tag 的删除
```

18.4 检出 tag

`git checkout -b <branchName> <tagName>`

因为 tag 本身指向的就是一个 commit，所以和根据 commit id 检出分支是一个道理

19 gitlab 提交代码自动触发 jenkins 构建

<https://blog.csdn.net/wanchaopeng/article/details/88233316>

<https://www.cnblogs.com/bugsbunny/p/7919993.html>

应用场景：

1. parent pom 修改后提交自动构建
2. 项目构建后还可以添加项目自动部署
3. 项目修改提交，触发自动化测试的脚本

20 项目托管到 github 操作流程(拓展)

