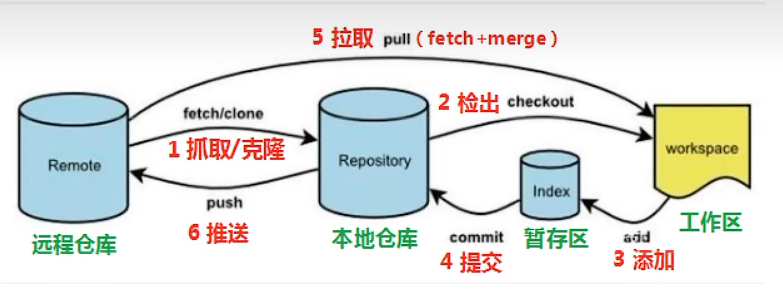
Git知识点

1. Git

5.Git工作流程图

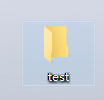


1. clone(克隆):从远程仓库中克隆代码到本地仓库
2. checkout(检出):从本地仓库中检出一个仓库分支进行修订
3. add(添加):在提交前先将代码提交到暂存区
4. commit(提交):提交到本地仓库。本地仓库中保存修改的各个历史版本
5. fetch(抓取):从远程仓库，抓取到本地仓库，不进行任何的合并操作，一般操作比较少
6. pull(拉取):从远程仓库拉到本地，自动进行合并(merge),然后放到工作区，相当于fetch+merge
7. push(推送)：修改完成后，需要和团队成员共享代码，将代码推送到远程仓库
8. Git使用的具体操作
9. 本地仓库的获取

如果需要使用Git对我们的代码进行一个版本控制，首先需要获得本地仓库，其具体步骤如下：

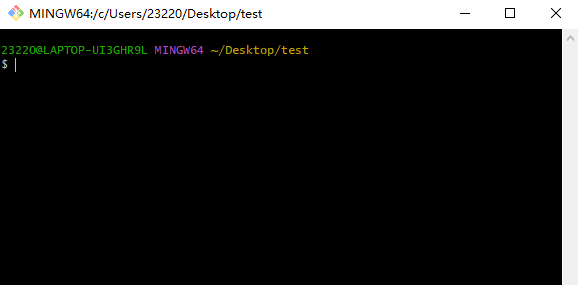
1. 在电脑的任意位置创建一个空目录来作为我们的本地Git仓库。

在桌面创建一个test文档，来作文本地Git仓库

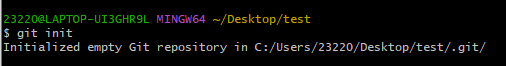


1. 进入这个目录中，点击右键打开Git base窗口

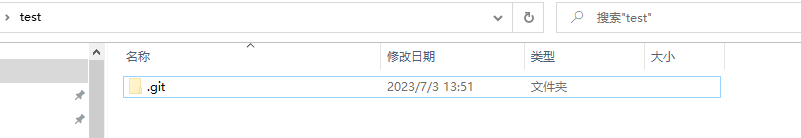




1. 执行命令git init

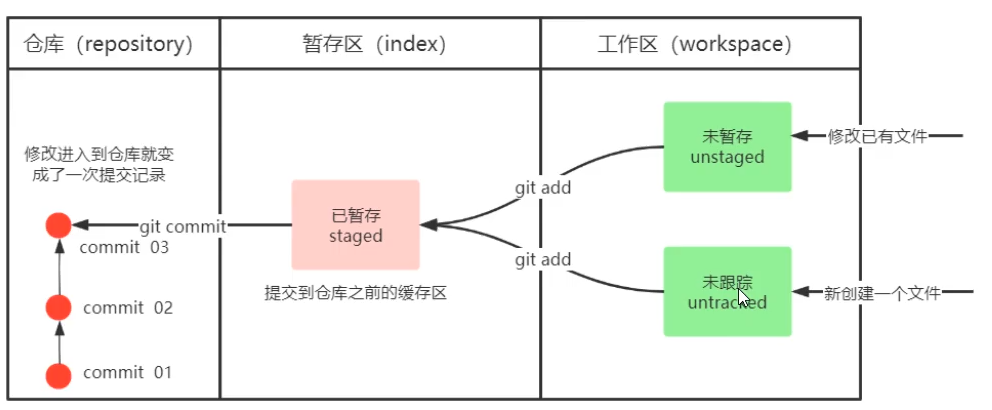


1. 如果3的命令执行成功后，我们可以在文件夹下看到一个隐藏的git目录。



1. Git的基本操作指令

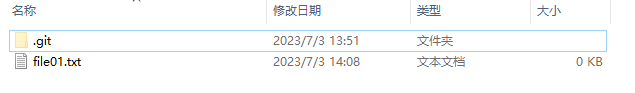
Git工作目录下（即有.git隐藏文件的目录）对于文件的修改（增加、删除、更新）会存在几个状态，这些修改的状态会随着我们执行Git的命令而发生变化，其具体的状态转换如下所示：



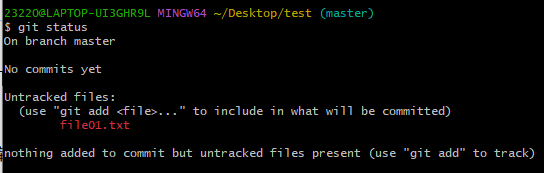
git add命令 (工作区 --> 暂存区)

我们先在git工作目录下创建一个文件





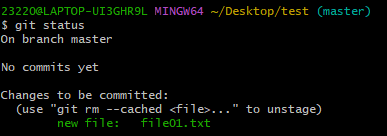
通过git status命令查看当前工作目录下的文件状态，可以看到有一个未跟踪的文件file01.txt



接着，我们可以通过git add file01.txt命令（或直接git . 使用.通配符，将当前工作目录中的所有的文件都加入到暂存区），将工作区的未跟踪的文件加入到暂存区

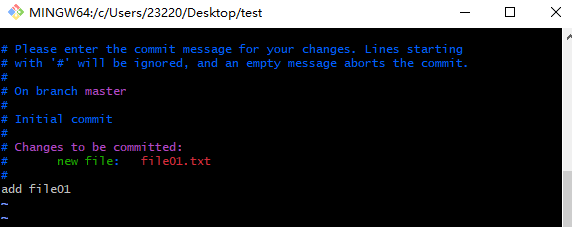


我们再通过git status命令查看一下文件状态，其显示该文件的状态为即将被提交，即已经将该文件从工作区内加入到了暂存区（即提交到仓库之前的缓存区）内



git commit命令 (暂存区 --> 本地仓库)

我们可以通过git commit命令将暂存区中的文件提交到本地仓库中，我们可以携带参数-m,并在后面跟上提交信息，如果不加-m参数，在输入git commit命令后，会进入vi命令界面再输入提交信息。我们这里不使用-m参数，并且在进入vi命令界面后，输入提交的信息



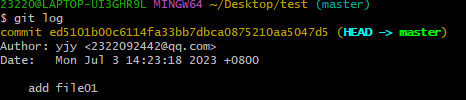
我们可以看到已经成功将暂存区中的文件提交到了本地仓库



我们在通过git status命令，查看一下所有文件的状态，可以看到此时已经没有需要提交的文件了，说明确实成功将暂存区中的文件提交到了本地仓库



我们可以通过git log命令去查看一下提交的历史记录,可以看档我们刚刚将那个文件提交了

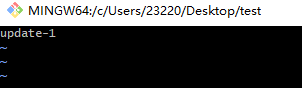


修改文件（仓库 -- > 工作区）

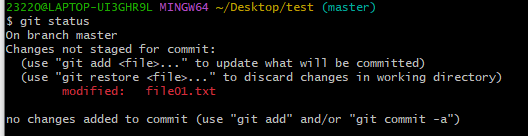
我们通过vi命令对file01文件进行修改



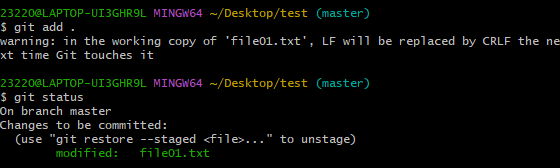
在当前文件中写入如下信息，表示第一次更新

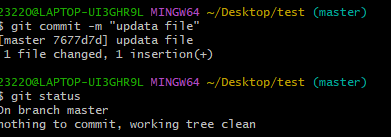


我们再次通过git status命令，查看一下当前所有文件的状态，可以看到之前提交的file01.txt文件，又回到了工作区，且状态为未暂存

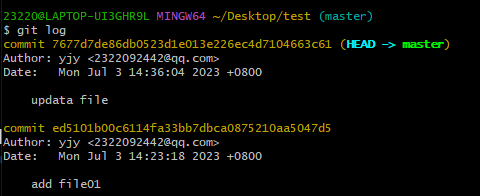


接着我们再次通过git add和git commit命令，将修改后的文件添加到仓库中





然后我们通过git log查看一下提交的历史记录，可以发现此时有两次提交，并且，有对应的描述信息，



具体的命令：

1. 查看修改的状态（status）

作用：查看修改的状态（暂存区、工作区）

命令形式：git status

1. 添加工作区到暂存区（add）

作用：添加工作区一个或多个文件的修改到暂存区

命令形式：git add单个文件名|通配符

将所有修改加入暂存区：git add

1. 添加暂存区到本地仓库（commit）

作用：提交暂存区内容到本地仓库的当前分支

命令形式：git commit -m “注释内容”

1. 查看提交日志（log）

作用：查看提交的记录

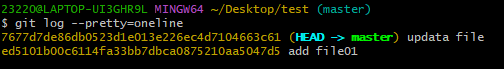
命令形式:git log [option]

options可以是如下

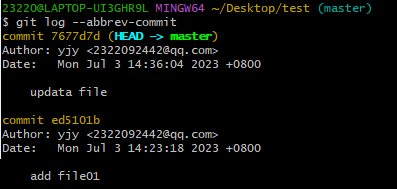
-- all

-- pretty=oneline

只显示Id和注释的内容



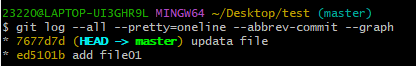
-- abbrev-commit 使得输出的commitId更简短，不用32位那么长，只需要前几位



--graph 以图的方式显示

使分支看起来更加直观

上面的参数可以一起使用

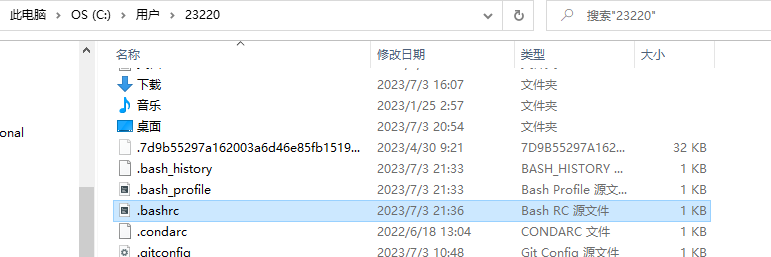


但由于其需要写的东西比较长，因此我们可以进一步配置

通过gitbash，以命令touch~/.bashrc创建bashrc文件，~表用用户的目录



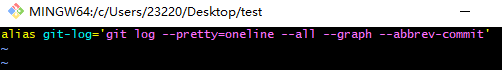
可以看到在用户目录下创建.bashrc文件成功



通过vi命令，进入.bashrc文件进行配置其中的内容



配置内容如图所示，通过alias去指定



通过source命令，将该配置文件生效



之后我们就可以知己通过git-log命令来执行上述很长的那一段命令了，两者是等价的



1. 版本回退

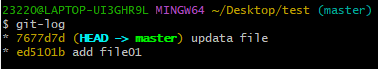
作用：版本切换

命令形式：git reset --hard commitID

（注意：上面的commitID可以通过git log进行查看到各个操作的唯一id值）

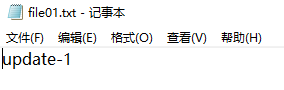
具体操作：

我们先看一下当前所有的提交记录



假设我们现在只需要回到添加文件的时候，而不需要修改文件

我们可以看到当前的文件中是有内容的（修改后的信息）



我们执行命令git reset --hard ed5101b，回到只进行添加文件提交的时候



我们通过git-log可以看到，此时只有一个提交了



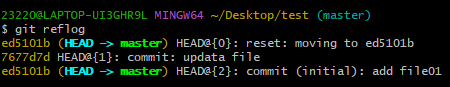
然后我们查看一下当前file01中的内容，可以看到里面没有内容，即已经回到了只提交添加该文件的时候



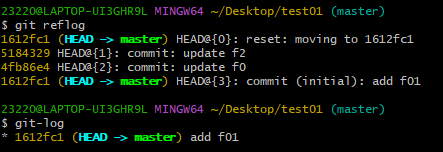
我们可以通过git reflog查看已经操作的记录，该指令可以帮助恢复误操作的数据，我们可以根据git reset --hard回到已经删除的记录时。

下面是通过git reflog命令，关于提交记录的回退显示详情：

三条提交记录的回退



四条提交记录的回退

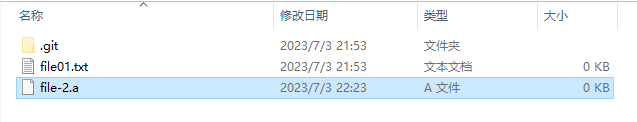


1. 指定不需要git管理的文件

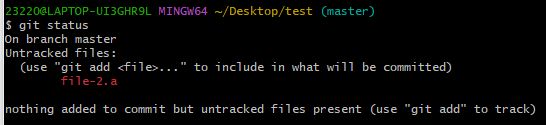
一般我们总会有有些文件无须纳入Git管理，也不希望他们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。在这种情况下，我们可以在工作目录中，创建一个名为.gitignore的文件（文件名称固定），在其中列出需要忽略的文件即可，具体操作如下所示：

我们创建一个以.a结尾的文件



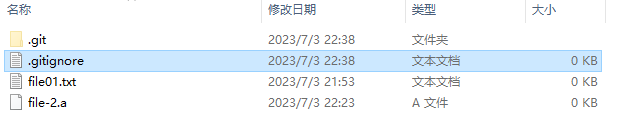


通过git status命令可以看到此时未跟踪的文件



如果我们不想让该后缀为.a的文件被git管理，我们可以在当前工作目录中通过touch命令创建一个.gitingore

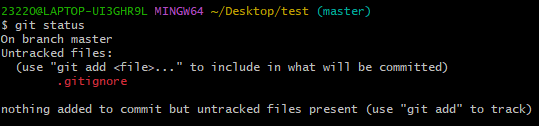




通过vi命令编辑.gitignore文件，里面写上\*.a,表示所有以.a文件结尾的文件都不参与git的管理



我们在通过git status命令可以看到此时git不再管理.a后缀的文件



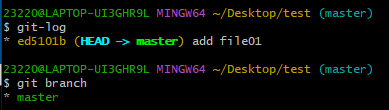
1. 分支

**分支概念：**几乎所有的版本控制系统都以某种形式支持分支，使用分支意味着你可以把你的工作从开发主线上分离开来去进行重大的Bug修改、开发新功能、以免影响开发主线。

1. 查看本地分支

命令：git branch

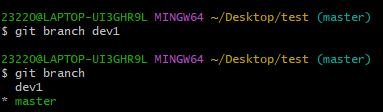
可以看到master就是一个分支



1. 创建本地分支

命令：git branch 分支名

可以看到此时又多出来一条分支dev1



我们可以通过git log查看一下各个分支提交的内容，可以看到此时两个分支都是和主线在同一位置，并且都只完成了一个添加文件file01的提交



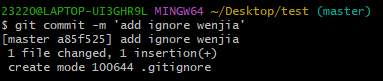
1. 切换分支（checkout）

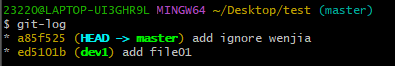
命令：git checkout 分支名

如果有多个分支，我们只能对一个分支进行操作（即在一个分支上进行任务的提交等操作）

我们将我们之前建立的.gitignore文件进行提交后，再次查看文件提交情况，我们可以看到，我们此时的提交时在master分支上的，对dev1分支没有任何影响（HEAD指向谁，那么当前使用的就是哪一个分支）

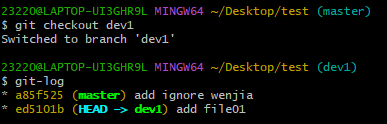


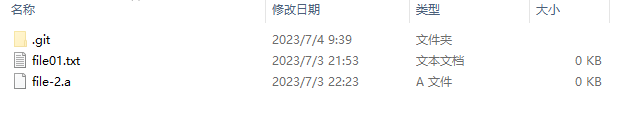






我们通过git checkout 分支名 命令，将分支切换到dev1，并且我们可以看到，此时工作目录下的.gitignore文件也消失了

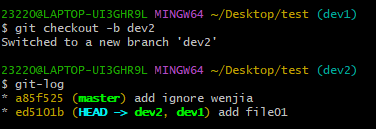




我们还可以直接切换到一个存在的分支（创建并切换）

命令：git checkout -b 分支名

并且新创建的分支的提交情况和当前使用的分支的提交情况是一样的



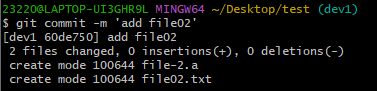
1. 合并分支

命令：git merge 分支名称（合并到当前正选择的分支上）

我们切换到dev1分支，并在该分支上进行一个创建文件file02的提交（由于没有master的.gitignore文件，在进行git add.命令后，将后缀为.a的文件也一起提交了，在后面忽略即可）









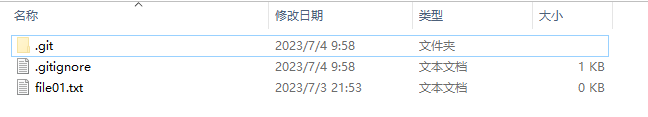
可以看到此时dev1此时和master作为两个分支，在dev1中可以看到file02文件，但看不到master分支中的ignore文件，但是在master分支中可以看到ignore文件，却看不到file02文件

Dev1分支对应的文件内容:

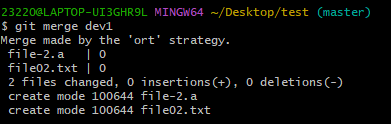


master分支对应的文件内容：

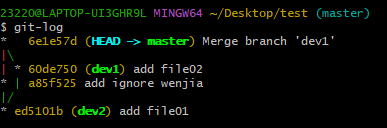




如果我们既想要有file02，又想要有.gitgnore，那么我们需要将master分支和dev1分支进行合并，一般我们合并的时候都是将其他分支合并到master分支上的



可以看档两个分支已经完成了合并



1. 删除分支

命令：

git branch -d 分支名 删除分支时，需要做各种检查

git branch -D 分支名 不做任何检查，强制删除

如对于master分支后面还有分支A，如果通过-d去删除分支A，可能会被系统认为是误操作，无法删除，因此需要-D去进行删除。

1. 解决冲突

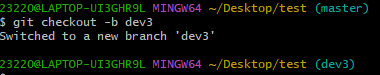
冲突的出现：当两个分支上对文件的修改可能会存在冲突，例如同时修改了同一个文件的同一行，这时需要手动去解决冲突，其解决冲突步骤如下：

1. 处理文件中冲突的地方
2. 将解决完冲突的文件加入暂存区（add）
3. 提交到仓库（commit）

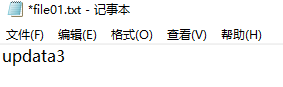
具体实例：

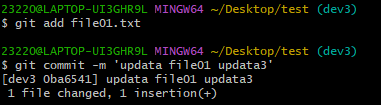
我们在master提交的情况下建立一个dev3分支

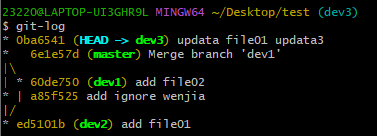




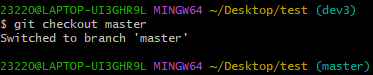
在dev3分支中，在file01中写入内容“updata3”，然后将file01文件提交到仓库

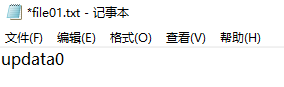


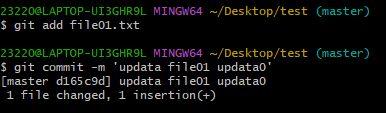


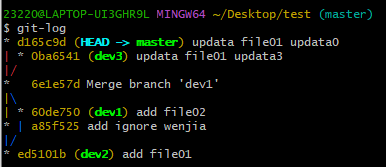


然后我们切换到分支master，也去修改file01中的内容，写入内容“updata0”，然后将file01文件提交到仓库

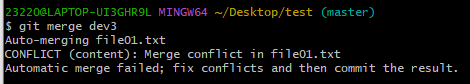




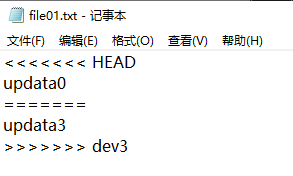




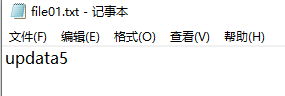
那么我们接下来，如果合并master和dev3的时候，就会出现问题，其提示在合并file01.txt文件的时候出现了冲突，自动合并失败，需要我们手动去解决矛盾然后进行提交到仓库

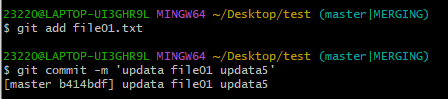


我们查看一下file01.txt中的显示内容，我们可以看到其在file01.txt中明确表示出了master分支（即当前HEAD）和dev3分支修改的冲突内容

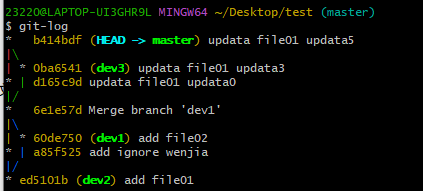


我们只需要暴力一点，将其统一一下即可，选择一个结果，如下图所示，将file01.txt文件中的内容修改为updata5（我们期望合并后的样子），然后对file01文件（发送冲突的文件）进行提交到仓库即可完成对冲突的解决





我们可以看到dev3和master合并了，并且冲突的文件统一为updata5



1. 开发中分支使用原则与流程

几乎所有的版本控制系统都以某种形式支持分支，使用分支意味着你可以把你的工作从开发主线上分离开来进行重大BUG的修复、发现新的功能、以免影响开发主线。

在开发中，一般有如下分支使用原则与流程：

* master（生产）分支：

线上分支、主分支、中小规模项目作为线上运行的应用对应的分支。

* develop（开发）分支：

是从master创建的分支，一般作为开发部门的主要开发分支，如果没有其他并行开发不同期上线要求，都可以在此版本中进行开发，阶段开发完成后，需要合并到master分支上，准备上线。该分支一般不删除，是一直存在的。

* feature/xxxx分支：

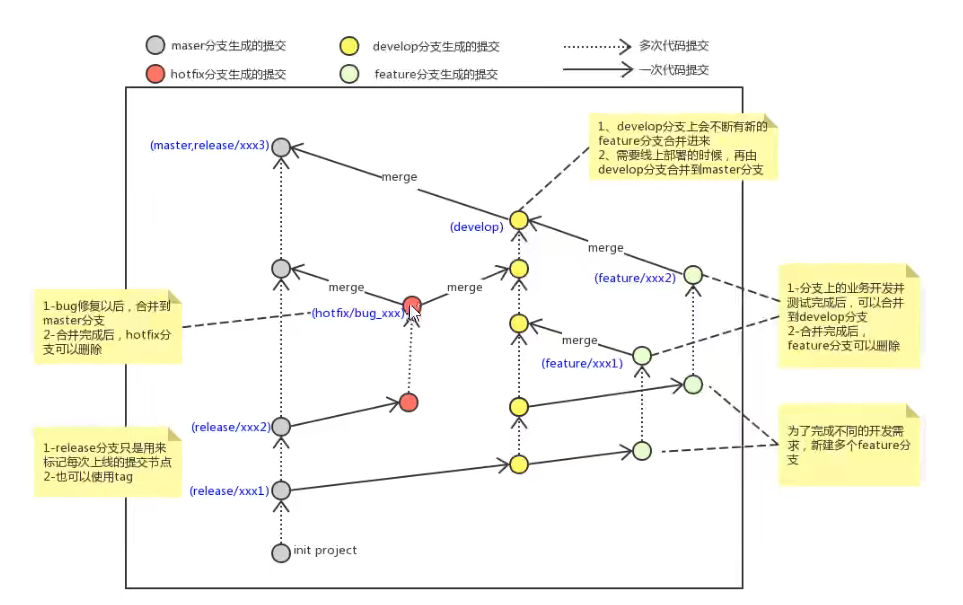
从develop创建的分支，一般是同期并行开发，但不同期上线时创建的分支，分支上的研发任务完成后合并到develop分支。

* hotfix/xxxx分支：

从master派生的分支，一般作为线上bug的修复使用，修复完成后需要合并到master、test、develop分支，作为同步。

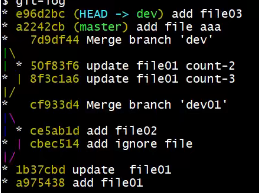
* 还有一些其他分支，在此不再赘述，例如test分支（用于代码测试）、pre分支（预上线分支）等等

具体的开服项目的版本管理图如下：



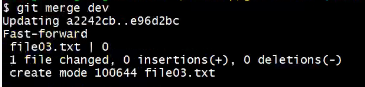
1. 合并的快进模式

如下这种情况，如果要将master合并到dev中，由于master分支比dev分支少了一次提交，并且使线性的，从下到上都是提交的累计，因此只需要将分支master移到dev分支所在的位置即可，这种合并也叫作合并的快进模式。

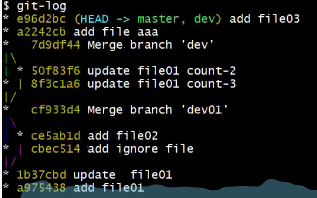


通过命令进行合并，提示此次合并为快进模式

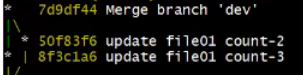




结果如下图所示：



如果是两个没有共同部分的分支的合并，需要生成一个新的版本



1. Git远程仓库
2. 常用的托管服务[远程仓库]

我们已经知道git中存在的两种类型的仓库，分别是本地仓库和远程仓库。那么我们如何搭建一个远程仓库？我们可以借助互联网上一些代码托管服务来实现，其中比较常用的有GitHub、码云、GitLab等。



1. 远程仓库码云的配置



点击注册，输入相关信息后，注册一个账号



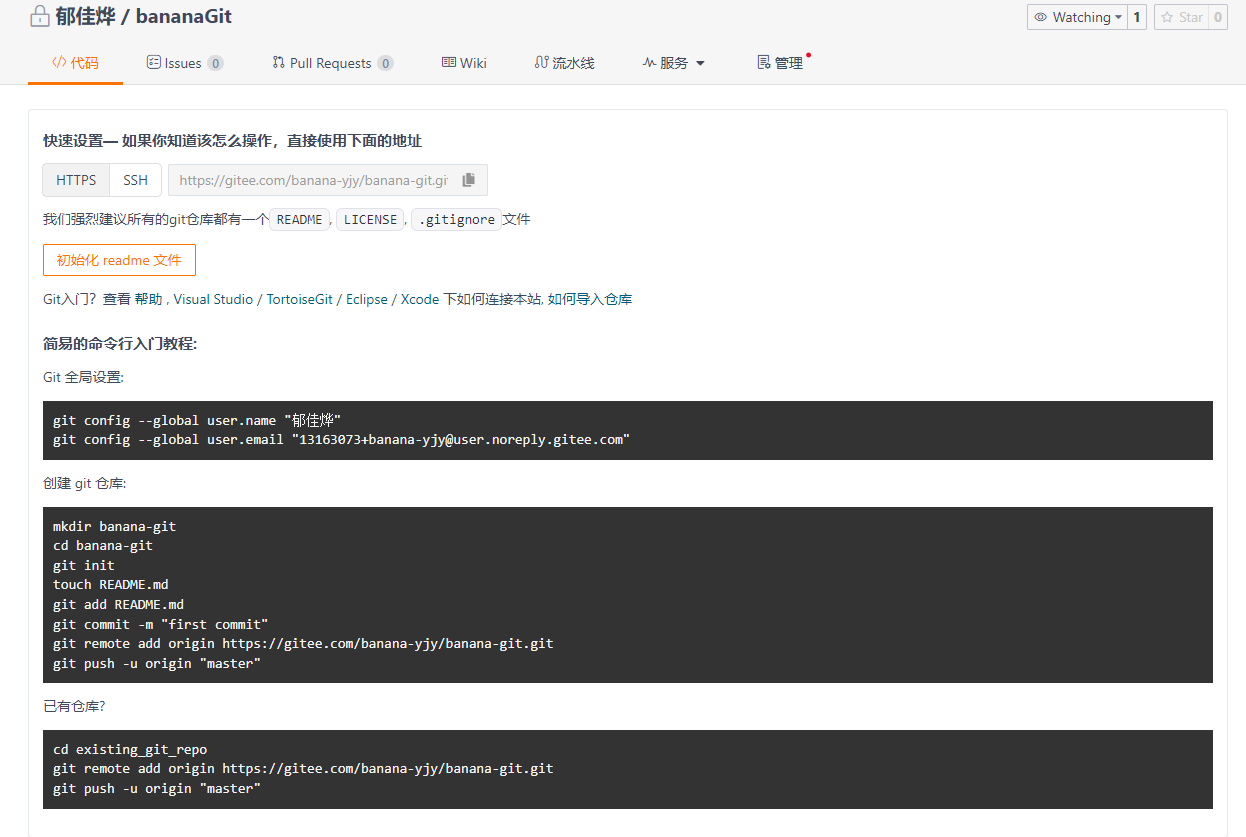
注册成功后，如下图所示



点击创建仓库，填入对应的仓库信息

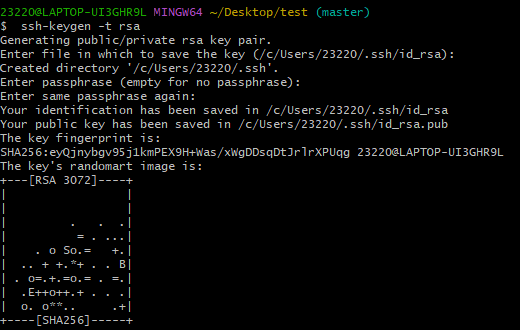


创建成功，如下图所示

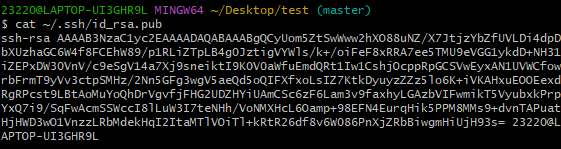


那么不是任何人都可以将本地仓库内容推到我们的远程仓库上的，因此我们还需要配置SSH公钥，来进行身份的验证（也可以通过输入码云的账号和密码进行验证）

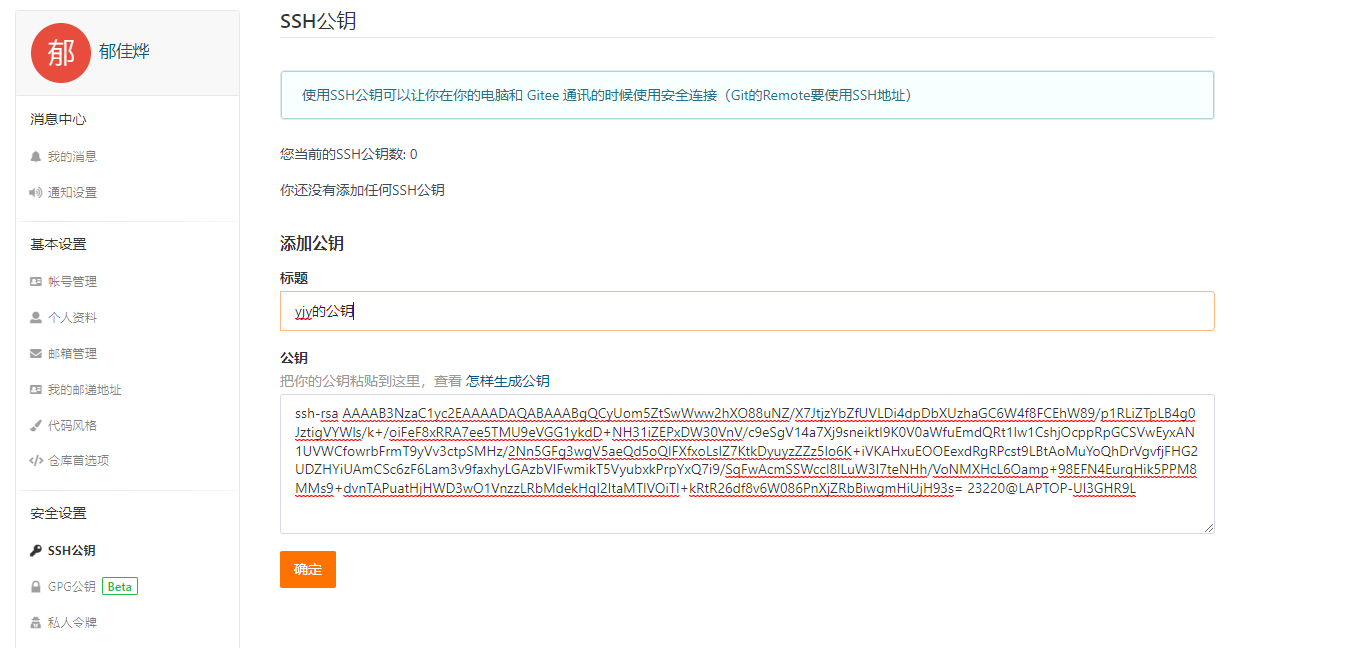
我们通过命令ssh-keygen -t rsa去生成ssh公钥，如果它提示你输入东西，就不断回车即可，如果公钥已经存在的话，会将其覆盖掉：



我们可以通过cat ~/.ssh/id\_rsa.pub去获取对应的公钥（其就是将公钥放在一个文件中，我们可以在该文件中进行查看公钥信息）

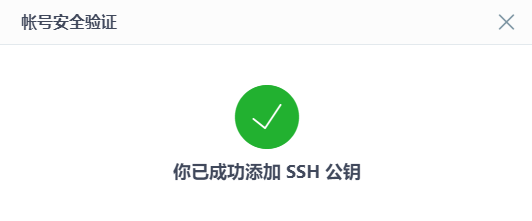


复制该公钥，去gitee中进行添加



输入码原密码即可完成添加





我们在gitbash中通过命令ssh -T [git@gitee.com去验证是否配置成功](mailto:git@gitee.com去验证是否配置成功)

如果是第一次，可能需要你输入各yes或no去确定是否进行连接

如下图所示，我们就成功连接了



1. 远程仓库添加（推送）

复制仓库的地址ssh



首先要告诉本地仓库，其对应的远程仓库是哪个（即绑定本地仓库和远程仓库）

通过命令：

git remote add <远程名称> <仓库路径>

远程名称：一般默认是origin，因为一个本地仓库可能对应多个远程仓库

仓库路径：从远端服务器获取此URL



我们可以通过命令：git remote查看添加的远程仓库是否成功，如果其显示对应的名称，则表示添加成功



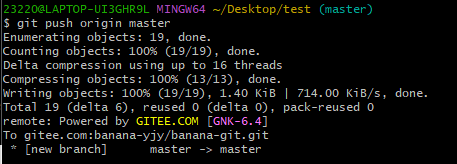
然后我们通过命令git push 远程仓库名称 分支名称 将某一分支推到远程仓库上

第一次会提示比较多的信息

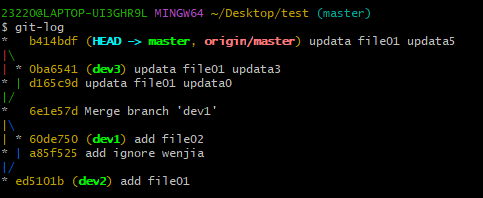
推送命令：

git push [-f] [--set-upstream] [远端名称 [本地分支名][:远端分支名]]

* 如果本地分支名和远端分支名是一样的，我们可以省略：远端分支名，那么此时的提交后的远端分支名就是本地分支名，但是此时两者并没有关联关系
* -f：表示强制覆盖（如果有冲突的话）
* --set-upstream：推送到远端的同时并且建立起和远端分支的关联关系（在第一次推送的时候应该加上）。
* 如果当前分支已经和远端分支关联，则可以省略分支名和远端名



由于省略了远端分支名，因此远端分支名和本地分支名是一样的，都是master



然后我们去gitee远程仓库中查看，可以看到我们本地仓库的内容已经成功推到了远程仓库中

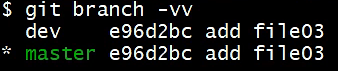


我们通过--set-upstream进行关联关系的建立

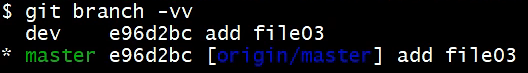


通过git branch -vv 查看关联关系建立情况

建立前



建立后



建立完关系后，我们就可以直接通过git push进行推送

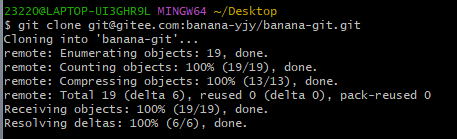


1. 远程仓库clone

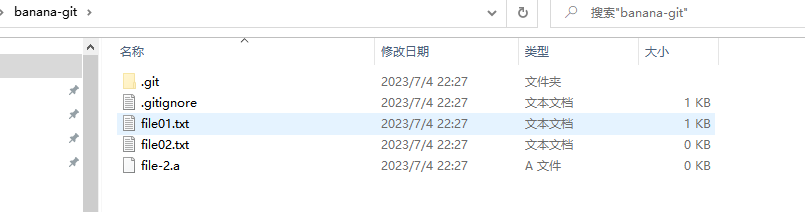
如果已经有一个远程仓库，我们可以直接clone到本地

命令：git clone <仓库名称> [本地目录]

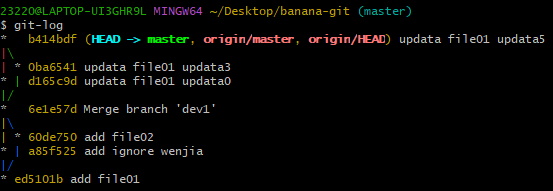
* 本地目录省略的话，会以仓库名称的末尾作为本地目录



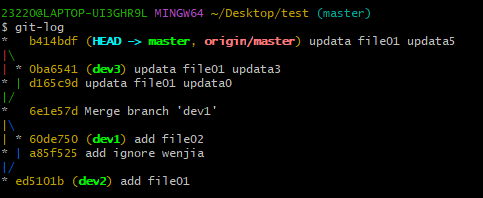




克隆过来文件的版本情况：



原文件的版本情况：



我们可以看到，除了没有推上去的dev1、2、3分支，其他基本上是一致的。

origin/HEAD就像一个指针，表示默认分支，在上图中指向远端分支origin/master，表示远端的默认分支是origin/master

1. 从远程仓库抓取和拉取

一般多人开发的时候，我们为了同步不会去选择clone，因为clone需要的工作量可能是很大的，比如A和B此时的版本是一样的，如果A此时进行了一次版本提交，那么B如果取远端仓库进行clone的话，又会将其已经有的所有代码又clone了一遍，一般对同一个仓库，我们的clone操作只使用一次。对于同步操作，我们使用的是抓取和拉取操作，

抓取：

命令：git fetch [remote name] [branch name]

* remote name：远端仓库名称
* branch name：远端仓库的哪一个分支
* 抓取指令就是将仓库里的更新都抓取到本地，不会进行合并。
* 如果不指定远端名称和分支名，则抓取所有分支

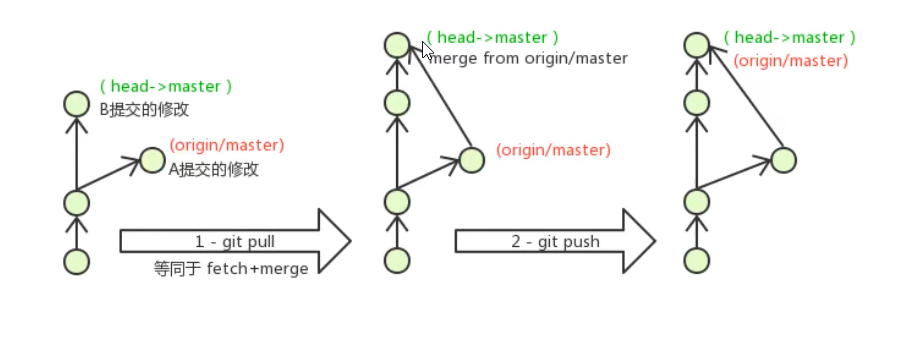
拉取：

命令：git pull [remote name] [branch name]

* 拉取指令就是将远端仓库的修改拉到本地并且自动进行合并，等同于fetch+merge
* 如果不指定远端名称和分支名，则抓取并更新当前分支

1. 解决合并冲突

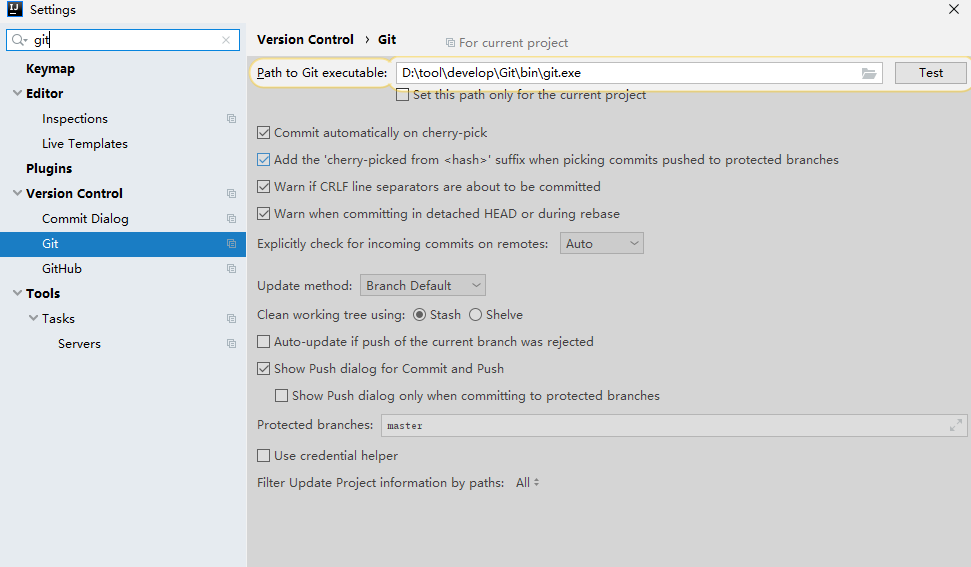
情景：在一段时间，A、B用户修改同一个文件，且修改了同一行位置的代码，则会发生合并冲突。如某一个时刻，A和B的版本是一样的，A用户在本地修改代码后优先推送到了远端仓库，此时B用户在本地修改好代码，提交到本地仓库后，也需要将本地仓库的代码推送到远端仓库，此时B的推送要晚于A，故需要先拉取仓库的提交（不然A提交的信息就丢失了，需要同步后，才可以提交到远端仓库），经过合并后才能够推送到远端分支，如下图所示：



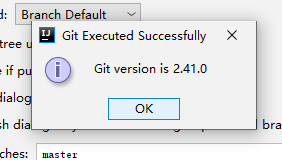
当B用户拉取代码的时候，因为A、B用户同一段时间修改了同一个文件相同的代码，那么这个时候如果合并的话就会发生冲突。（注意：远程分支也是分支，所以合并时冲突的解决方式和解决本地分支冲突是相同的，只需要将冲突的文件统一一下，将需要的内容确定下来即可）解决完冲突后，再将B的仓库push到远端仓库即可。

1. 在Idea中使用Git
2. 在idea中配置git

安装好idea后，如果git安装在默认路径下，那么idea会自动找到git位置，如果更改了git的安装位置，则需要手动配置下git路径，选择file-setting打开设置窗口，找到version control下的git选项，选择git的目录



然后点击test，如果实现git的版本号，则表示配置成功

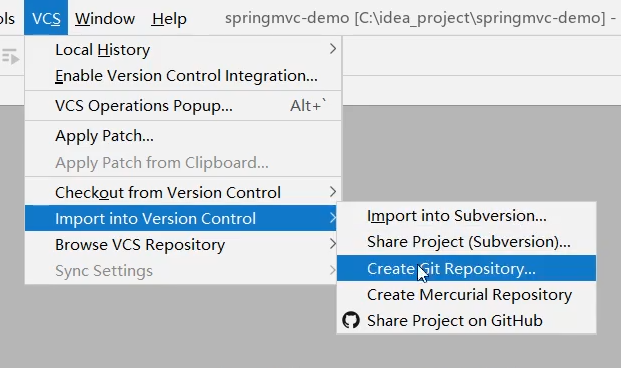


1. 创建项目远程仓库

和之前一样，在gitee上面创建一个远程的仓库即可。

1. 创建本地仓库

在Idea中，打开我们需要管理的项目，点击VCS-Import into Version Control-Create Git Repository即可创建一个当前项目的本地仓库

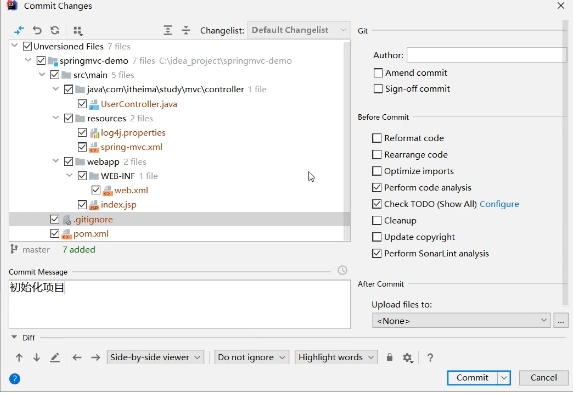


创建本地仓库完成后，会在idea中出现如下标志

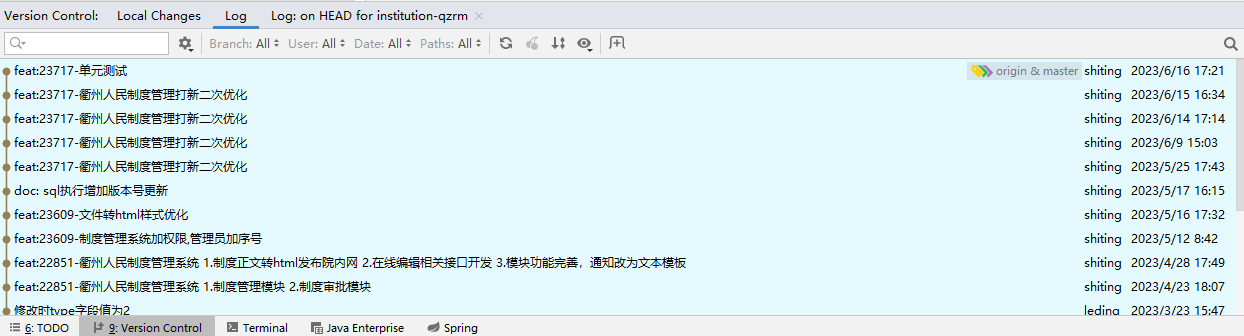


蓝色是pull命令

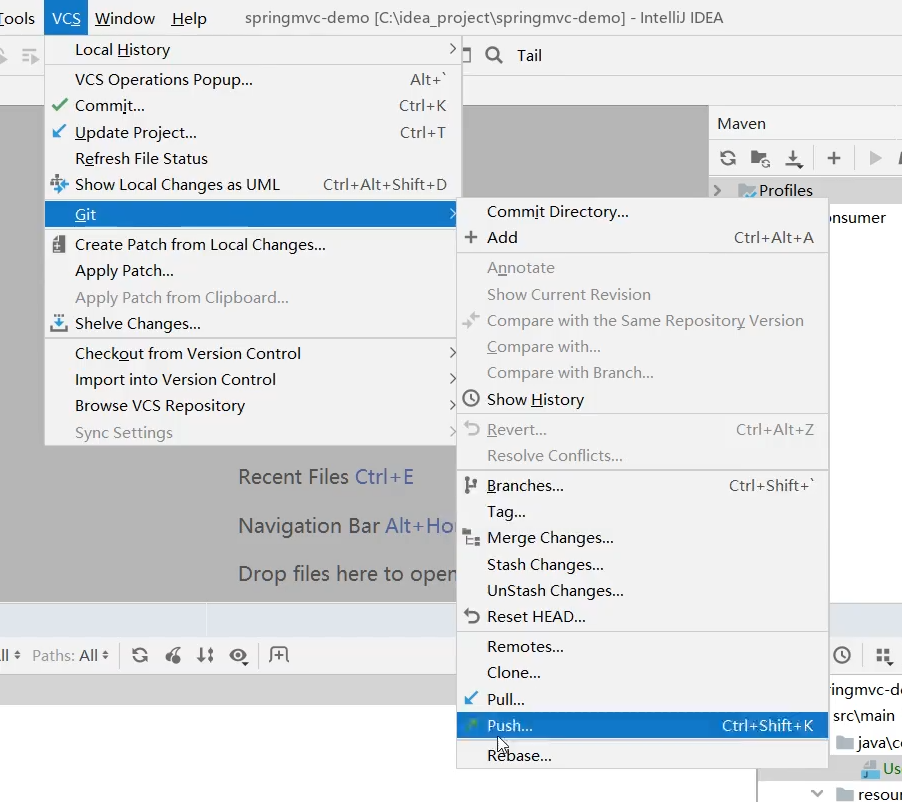
其中绿色的√就是提交的意思，其弱化了git中的工作区和暂存区的概念，我们只需要选择需要提交的文件，写入对应的commit Message，然后点击提交（commit）即可，如果我们已经配置了仓库，直接选择commitandpush即可发到远端仓库上。



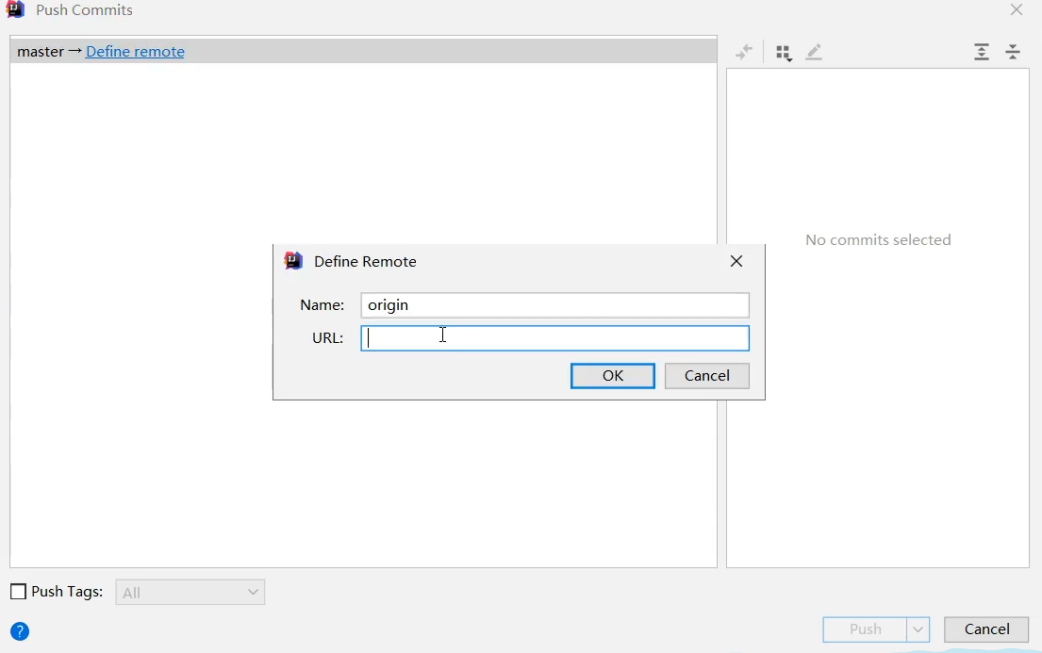
我们点击idea下面的version control-log，即可查看提交的记录



我们选择VCS-Git-Push，即可将我们的本地仓库推送到远端仓库



还要进行远端仓库的添加，输入我们gitee的地址，即可进行push

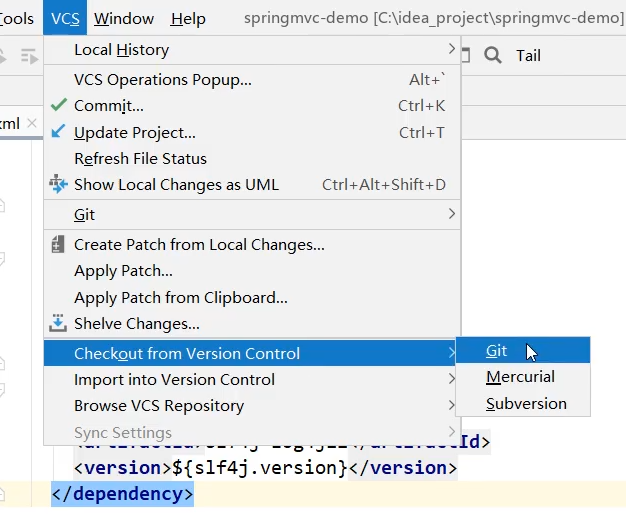


1. 克隆远端仓库项目

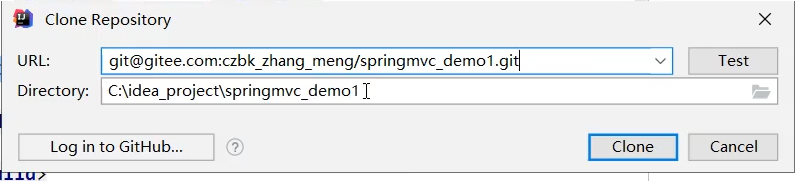
我们复制需要克隆的远端仓库ssh



选择VCS-Checkout from Version Control - Git

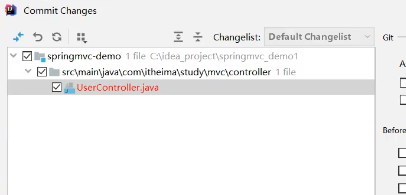


选择需要克隆的远端仓库版本和放到本地仓库的位置，然后接下去根据它的提示进行选择即可



如果在IDEA中，A和B协同开发，并且A和B都是对一个文件进行仓库，那么在A进行提交到远端仓库后，B需要先从远端仓库抓取最新版本下来（pull）后，再进行提交，如果在抓取的时候遇到冲突，那么就和上面的解决方法一样，确定唯一的内容，再次提交即可。

解决完冲突之后，需要提交的时候，发现冲突文件爆红

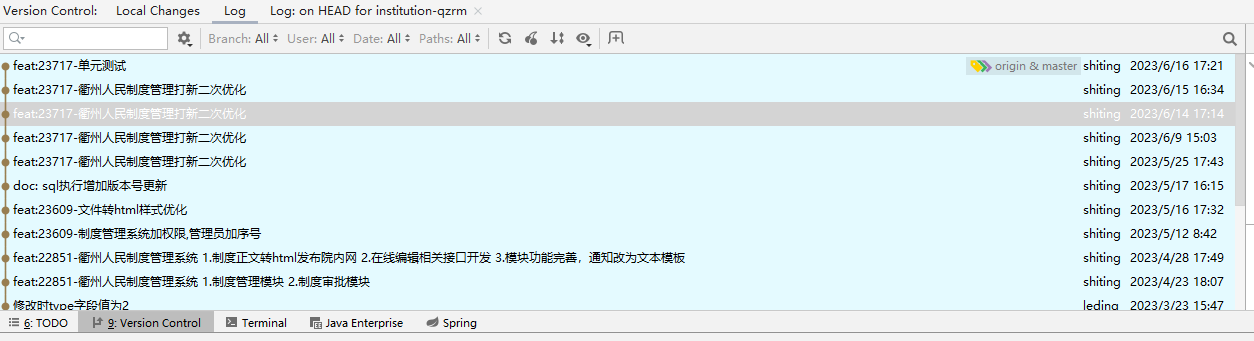


我们右键选中该冲突文件，选择git-add，之后再进行提交就可以了

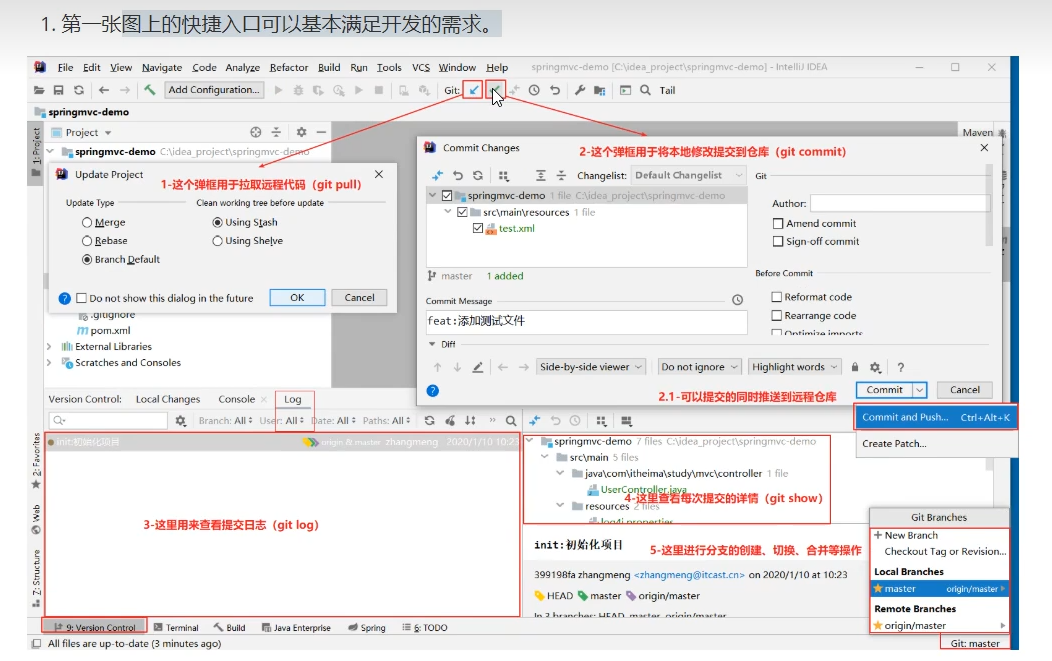
1. 分支的切换

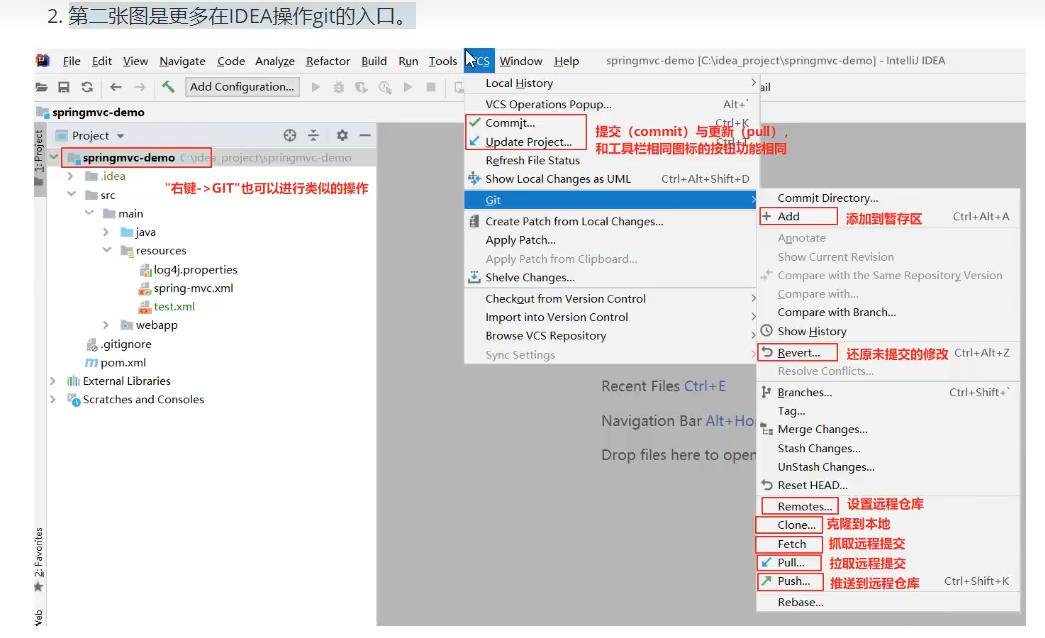
方法一：选择IDEA右下角可以进行分支的切换



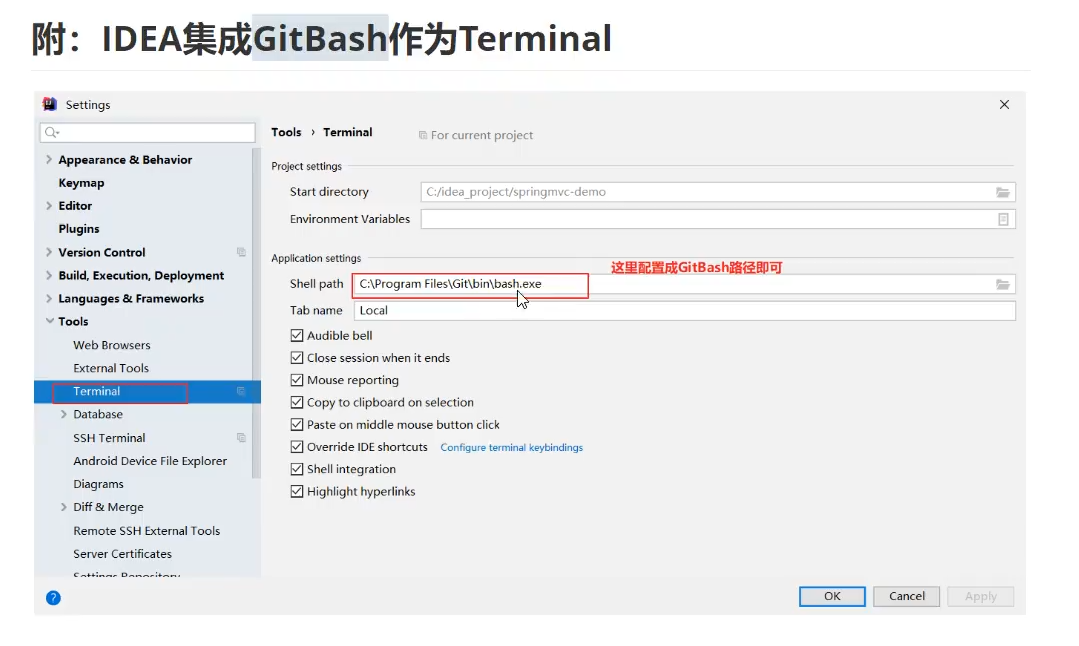
方法二：选择左下角的version Control-log，然后右击想要的版本，然后new添加一个新的分支，并为其命名

1. git操作入口





1. 其他
2. IDEA集成GitBash作为Terminal



1. 铁令

切换分支前先提前提交本地的修改

代码及时提交，提交了就不会丢

遇到任何问题都不要删除文件目录