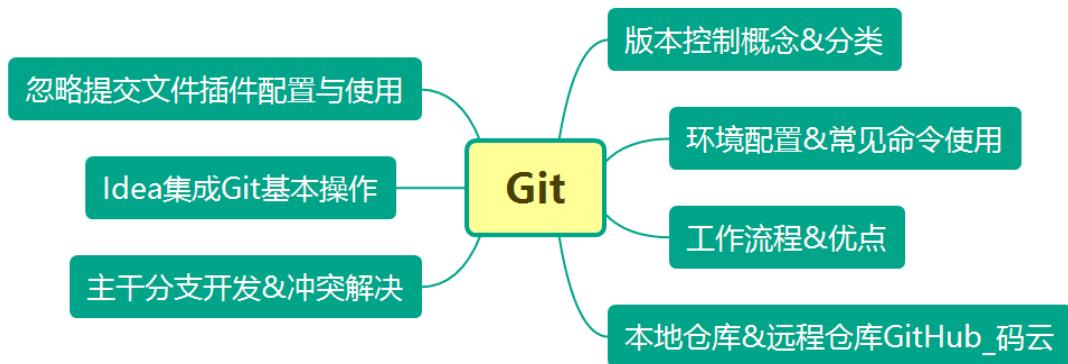


Git

1. 学习目标

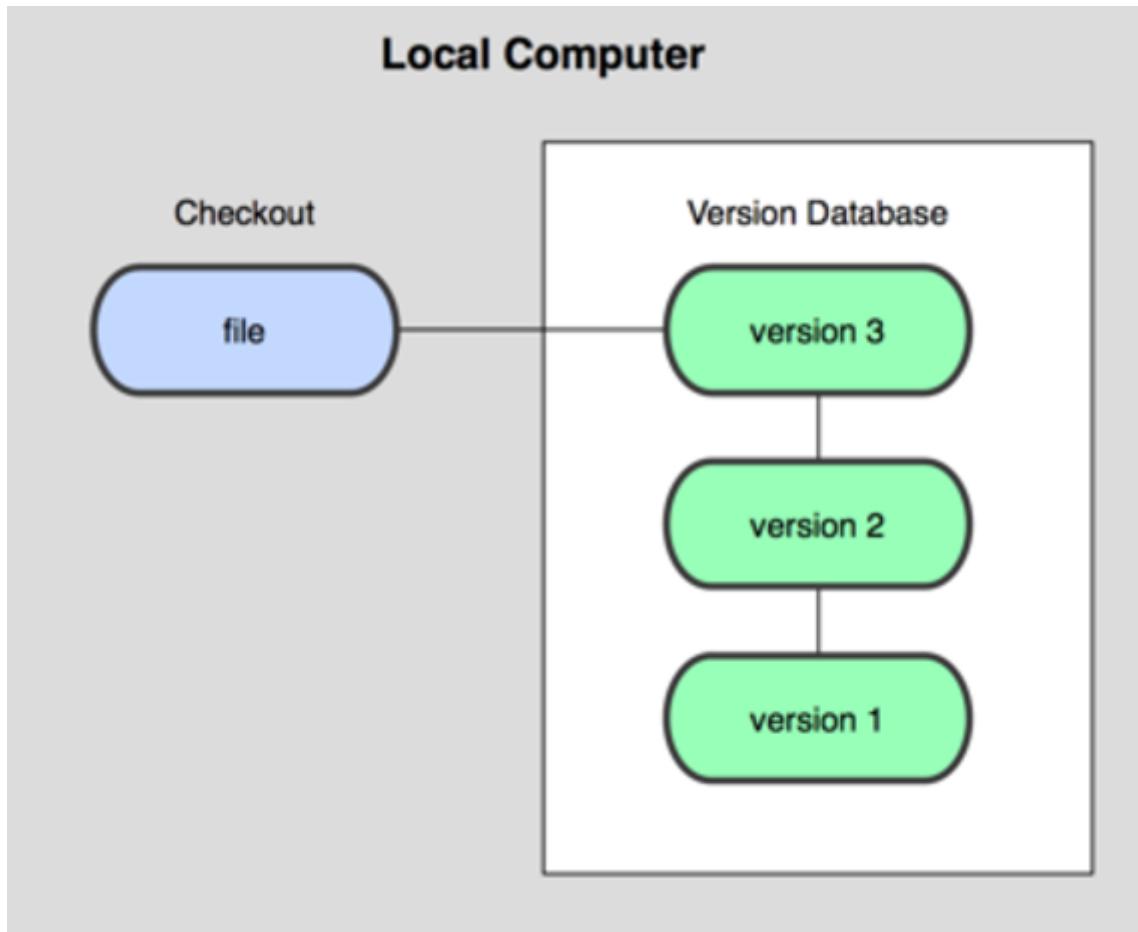


2. 版本控制

版本控制是一种记录若干文件内容变化，以便将来查阅特定版本修订情况的系统。简单讲就是备份和记录。接下来我们要了解三种不同版本控制的发展历程。

2.1. 本地版本控制系统

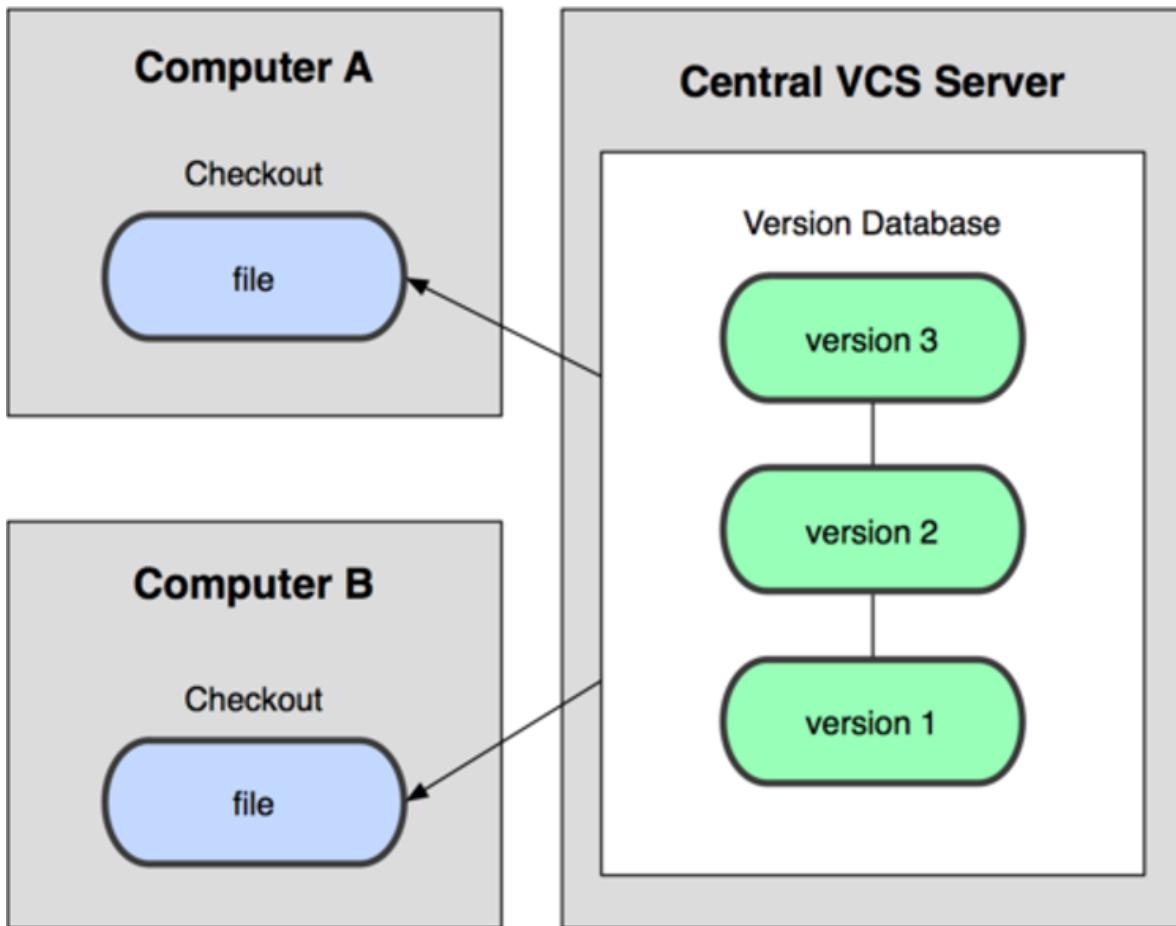
人们把项目拷贝到本地磁盘上进行备份，然后以命名方式来区分。这种做法好处是简单，但坏处也不少比如备份比较多或许就会混淆不同版本之间的区别。那为了解决这个问题，有人就开发了一个本地版本的管理系统，它的结构图如下：



本地版本管理就是把版本号存入数据库来记录文件的历次更新差异。

2.2. 集中化版本控制系统

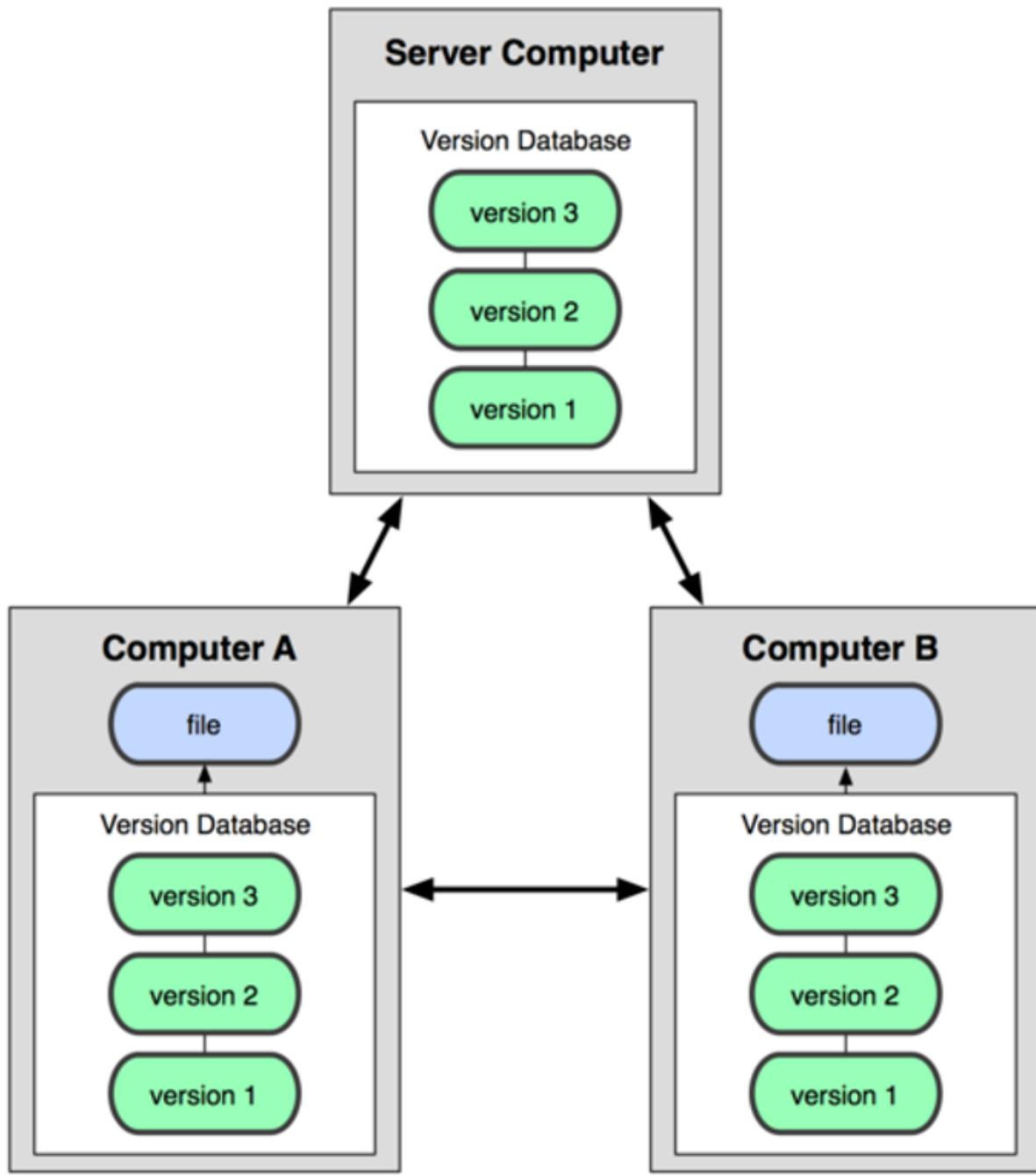
本地版本控制系统能够将不同版本的文档保存下来并且借助版本记录可以很方便定位相关文件但又引入了新的问题，如何让在不同系统上的开发者协同工作？于是，集中化的版本控制系统（Centralized Version Control Systems，简称 CVCS）应运而生。这类系统，诸如 CVS，Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这已成为版本控制系统的标准做法。



这样做的好处是解决了人们开发协同的问题,但是把所有的代码提交到同一台服务器上有一个很明显的问题就是单点故障,如果这台服务器宕机了,那所有人都不能提交代码,还有如果这台服务器如果磁盘发生故障,碰巧没做备份,或者备份不够及时,就还是会有丢失数据的风险。最坏的情况是彻底丢失整个项目的所有历史更改记录,而被客户端提取出来的某些快照数据除外,但这样的话依然是个问题,你不能保证所有的数据都已经有人事先完整提取出来过。本地版本控制系统也存在类似问题,只要整个项目的历史记录被保存在单一位置,就有丢失所有历史更新记录的风险。

2.3. 分布式版本控制系统

为了解决集中化版本管理所带来的问题分布式版本管理控制系统(Distributed Version Control System, 简称 DVCS)就应运而生了. 在这类系统中, 像 Git, Mercurial, Bazaar 以及 Darcs 等, 客户端不只是提取出最新版的文件快照, 而是把最原始的代码仓库镜像到本地. 这样一来, 任何一处协同工作用的服务器发生故障, 事后都可以用任何一个镜像出来的本地仓库恢复。因为每一次的提取操作, 实际上都是一次对代码仓库的完整备份。



所以综上来看的集中化版本控制系统是对本地版本控制系统的一次升级,因为它加入了协同操作,分布式版本控制系统是对集中化控制系统的一次补充,使之更加完善。

3. Windows上安装Git

最早Git是在Linux上开发的,很长一段时间内, Git也只能在Linux和Unix系统上跑。不过,慢慢地有人把它移植到了Windows上。现在, Git可以在Linux、 Unix、 Mac和Windows这几大平台上正常运行了。

在Windows上使用Git,先从Git官网直接[下载安装程序](#),选择指定系统下载,然后按默认选项安装即可。安装完成后,在开始菜单里找到“Git”->“Git Bash”,显示出类似命令行的窗口,说明Git安装成功!

在窗口内输入 git --version 查看git 版本信息如下:

 MINGW64:/d/git

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git --version
git version 2.24.0.windows.2
```

在使用用Git工作之前，我们需要做个一次性的配置。方便后续Git能跟踪到谁做了修改，我们需要设置对应的用户名与邮箱地址。

```
1 git config --global user.name "your_username"
2 git config --global user.email your_email@domain.com
3 git config --list 查看所有配置
```

注意`git config`命令的`--global`参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git config --global user.name "zsyue"

Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git config --global user.email [REDACTED]

Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git config --list
http.sslcainfo=./software/teach/Git/mingw64/ss1/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager
user.email=[REDACTED]
user.name=zsyue

Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$
```

4. 理解Git文件的三种状态与工作模式

使用Git操作文件时，文件的状态有以下三种：

状态	描述
已提交 (committed)	已提交表示数据已经安全的保存在本地数据库中。
已修改 (modified)	已修改表示修改了文件，但还没保存到数据库中。
已暂存(staged)	已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

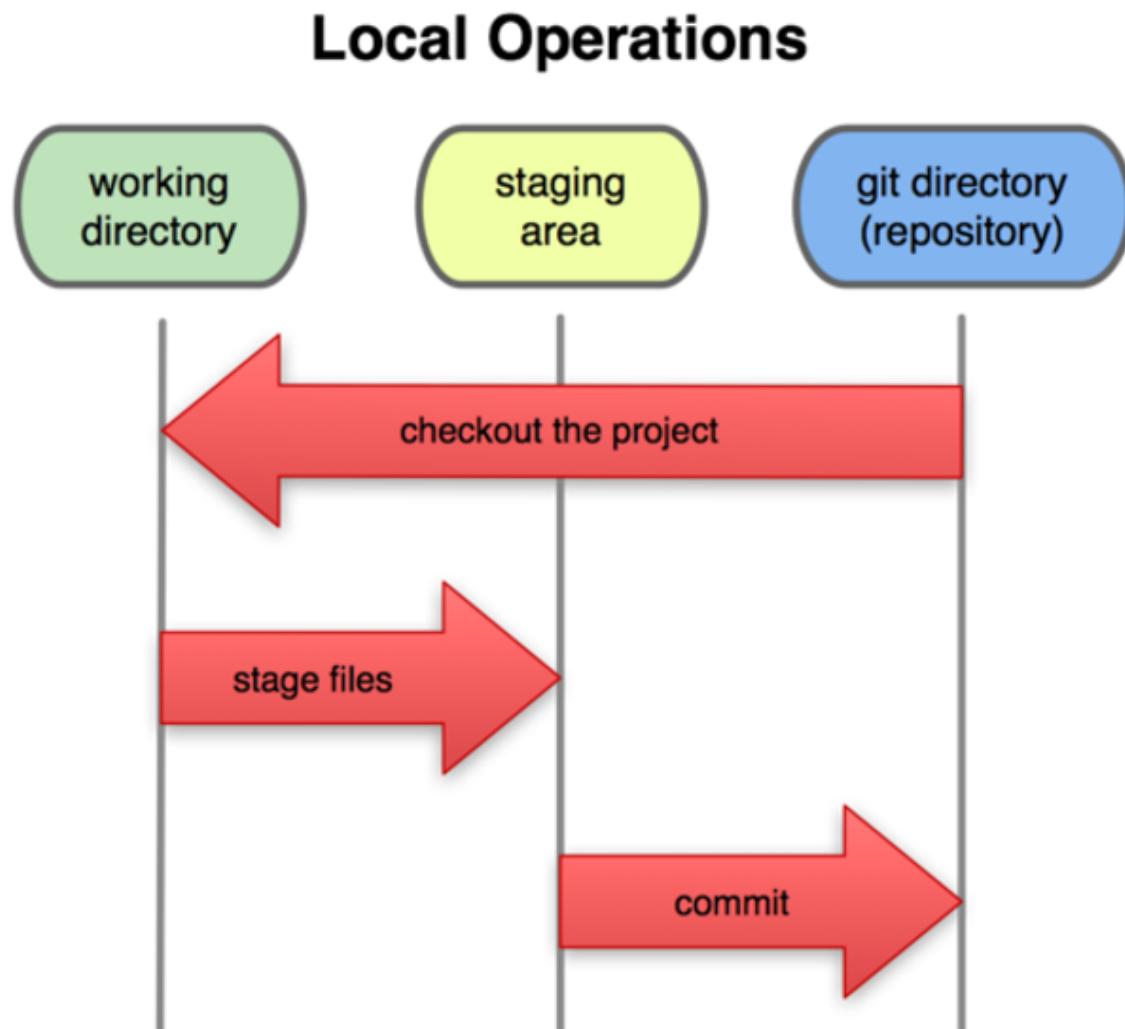
针对Git文件的三种状态，这里需要了解Git项目的三个工作区域：工作区、暂存区和Git仓库。

分类	描述
工作区	简单的理解为在电脑里能看到的目录，比如自己创建的本地项目目录
暂存区	Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git自动创建的第一个分支master，以及指向master的一个指针叫HEAD。
Git仓库	工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

基本的Git 工作流程描述如下：

- 在工作区中修改某些文件。
- 对修改后的文件进行快照，然后添加到暂存区。
- 提交更新，将保存在暂存区域的文件快照永久转储到 Git 仓库中。

流程图如下：



5. 创建版本库并提交文件

版本库又名仓库，可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。理解了Git文件状态与三种工作区域之后，通过一个例子来体验Git对于文件的基本操作。

编写一个文本文件并将文件提交到git仓库

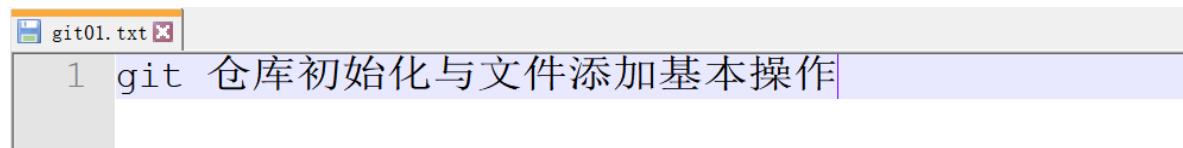
- 初始化git本地仓库

通过执行`git init`命令在本地初始化一个本地仓库，执行该命令后会在本地初始化一个没有任何文件的空仓库。

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git init
Initialized empty Git repository in D:/git/.git/
```

- 新建文本文件`git01.txt`并添加到暂存区

文本内容如下：



```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    git01.txt

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

在`.git`同级目录下添加`git01.txt`文件后，使用`git status`查看工作目录与暂存区文件状态

1 `git status`命令用于显示工作目录和暂存区的状态。使用此命令能看到那些修改被暂存到了，哪些没有，哪些文件没有被Git tracked到。

执行`git add`命令添加文件到暂存区

- 1 `git add path`通常是通过`git add <path>`的形式把`<path>`添加到索引库中，`<path>`可以是文件也可以是目录。
- 2 `git`不仅能判断出`<path>`中，修改(不包括已删除)的文件，还能判断出新添的文件，并把它们的信息添加到索引库中。

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   git01.txt
```

此时可以看到有一个git已tracked到新文件git01.txt，文件被成功存放到了暂存区

- 提交文件到本地版本库

文件被添加到暂存区后，执行`git commit`命令提交暂存区文件到本地版本库中。

1 | `git commit` 命令用于将更改记录(提交)到存储库。将索引的当前内容与描述更改的用户和日志消息一起存储在新的提交中。通常在执行提交时 在`git commit` 命令后跟上`-m` 属性 加入本次提交的记录说明 方便后续查看提交或改动记录。

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git commit -m 'git 版本库初始化与文件提交'
[master (root-commit) 79eda75] git 版本库初始化与文件提交
 1 file changed, 1 insertion(+)
 create mode 100644 git01.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

`git log`：命令用于显示提交日志信息。(比较常用,后续讲到时光穿梭时会经常使用该命令)。

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log
commit 79eda75dda912c48aaca514a5a682fc06b3d32e9 (HEAD -> master)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:47:50 2020 +0800

  git 版本库初始化与文件提交
```

6. 时光穿梭机

企业在多人的项目开发环境下，使用Git版本控制工具对项目版本进行管理时，通常会对项目不同版本的文件进行查看，项目历史版本，未来版本的切换操作，对于一个项目开发人员，此时对于Git的这些基本命令操作就成为了一项基本技能。

6.1. 修改文件与文件提交

修改后内容如下：

```
git01.txt
1 git 仓库初始化与文件添加基本操作
2 git 开启时光穿梭机的故事
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   git01.txt

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

此时当文件修改后 使用 `git status` 命令可以看到git 检测到文件被修改，git 版本库给出的下一步操作是添加修改的文件到暂存区 此时执行添加操作命令

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git add git01.txt
```

执行提交

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git commit -m '第一次修改git01.txt文本'
[master a278212] 第一次修改git01.txt文本
 1 file changed, 2 insertions(+), 1 deletion(-)
```

`git log` 命令查看操作日志记录

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log
commit a278212ed8ec7ceedbe261be57732019f7b4d785 (HEAD -> master)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:50:57 2020 +0800

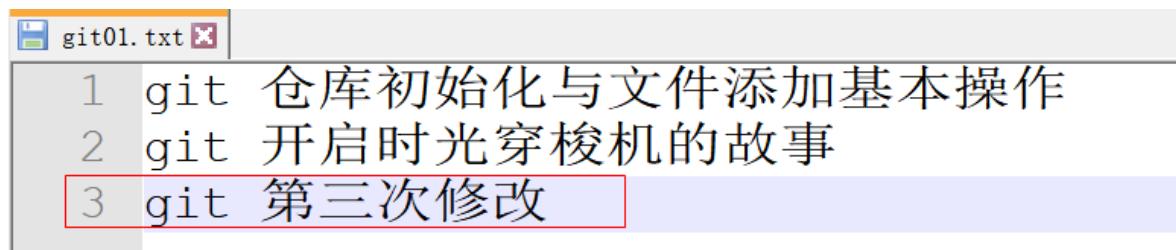
    第一次修改git01.txt文本

commit 79eda75dda912c48aaca514a5a682fc06b3d32e9
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:47:50 2020 +0800

    git 版本库初始化与文件提交
```

修改需要注意的问题

下面再次修改git01.txt 然后执行提交操作



执行提交操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git commit -m '第二次修改操作'
On branch master
Changes not staged for commit:
  modified:   git01.txt

no changes added to commit
```

此时 执行 `git diff HEAD -- git01.txt` 与版本库内容进行比较结果如下:

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git diff HEAD -- git01.txt
diff --git a/git01.txt b/git01.txt
index ad795e8..2436476 100644
--- a/git01.txt
+++ b/git01.txt
@@ -1,2 +1,3 @@
    git 仓库初始化与文件添加基本操作
-git 开启时光穿梭机的故事
\ No newline at end of file
+git 开启时光穿梭机的故事
+git 第三次修改
\ No newline at end of file
```

- 1 差异比较说明
- 2 `---`：表示变动前的文件
- 3 `+++`：表示变动后的文件
- 4 变动的位置用两个@作为起首和结束
- 5 @@ -1,2 +1,3 @@: 减号表示第一个文件，"1"表示第1行，"2"表示连续2行。同样的，"+1,3"表示变动后，成为第二个文件从第1行开始的连续3行。

可以看出：文本中第三行内容并没有提交到版本库中 原因在于修改后的git01.txt 并没有添加到暂存区，所有执行提交操作并不会发生改变。

暂存区文件提交与撤销

当发现因失误而将文件添加到暂存区时，git 支持文件的撤销操作 执行命令 `git reset HEAD` 文件 操作如下：

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git add test.txt
```

查看版本库状态并执行撤销操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   test.txt
```

再次查看版本库状态 test.txt 成为未追踪文件

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt

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

6.2. 版本回退

当文件修改后被提交的次数很多时，对于版本库中存放的文件就会出现不同的版本，在多人开发的项目环境中，通常会对不同版本文件进行查看甚至回退的情况（比如某些游戏中所提供的状态保存功能，能够在某一时刻保存整个游戏场景状态以方便后续继续在该状态下进行游戏进行而不是从头开始）。值得庆幸的是 Git 也提供了同样的功能，能够让开发者在不同版本的项目中进行切换，达到时空穿梭自如的目的！

对于上面操作的git01.txt 文件已有几个版本，对于历史版本的查看 使用 `git log` 命令：

- 1 `git log`: 命令用于显示提交日志信息

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log
commit c51926c8cf867dbd947c2faef970b75147598ccd (HEAD -> master)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:57:22 2020 +0800

    提交

commit a278212ed8ec7ceedbe261be57732019f7b4d785
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:50:57 2020 +0800

    第一次修改git01.txt文本

commit 79eda75dda912c48aaca514a5a682fc06b3d32e9
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:47:50 2020 +0800

    git 版本库初始化与文件提交
```

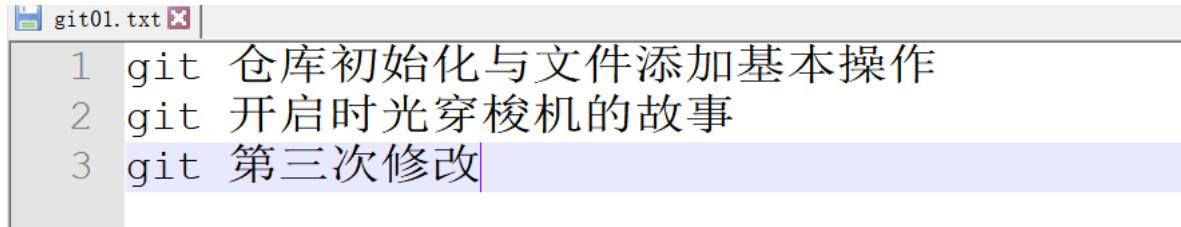
列表显示的结果按提交时间倒叙排序，其中第一条中 `HEAD -> master` 代表当前指针指向Git 版本库中master 主干分支，每次提交 Git 内部均会生成一个唯一的SHA 算法构建的字符串来唯一标识当前版本

此时如果想要执行版本版本回退操作使用命令 `git reset`

1 `git reset` 命令用于将当前HEAD复位到指定状态。一般用于撤消之前的一些操作(如: `git add,git commit`等)。

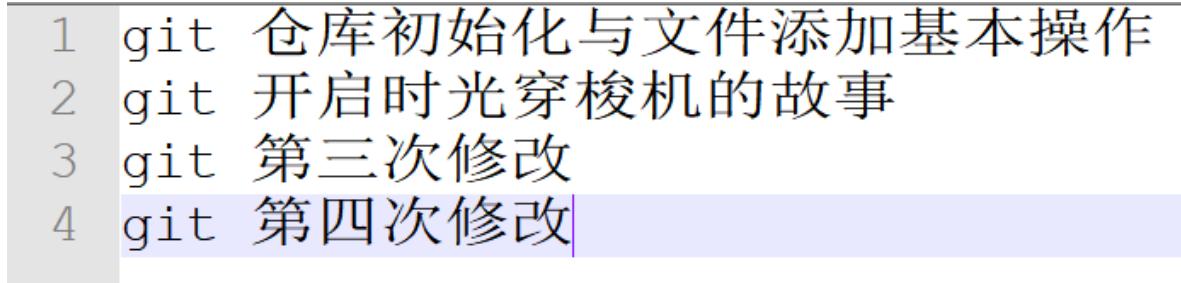
回滚前在执行两次提交操作 方便文件不同版本间的切换

第三次提交 修改后的git01.txt 内容如下:



```
git01.txt | 1 git 仓库初始化与文件添加基本操作  
           2 git 开启时光穿梭机的故事  
           3 git 第三次修改
```

第四次提交 修改后的git01.txt 内容如下:



```
git01.txt | 1 git 仓库初始化与文件添加基本操作  
           2 git 开启时光穿梭机的故事  
           3 git 第三次修改  
           4 git 第四次修改
```

`git log` 查看提交历史记录如下

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log
commit a4a44dc0d972e50caf3c369082b6cd7031890598 (HEAD -> master)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:59:52 2020 +0800

    第四次提交操作

commit c51926c8cf867dbd947c2faef970b75147598cccd
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:57:22 2020 +0800

    提交

commit a278212ed8ec7ceedbe261be57732019f7b4d785
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:50:57 2020 +0800

    第一次修改git01.txt文本

commit 79eda75dda912c48aaca514a5a682fc06b3d32e9
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:47:50 2020 +0800

    git 版本库初始化与文件提交
```

当然，如果提交历史记录较多可以加入数字控制显示的版本记录数 并且使用`--pretty=oneline` 简化输出 如下：

显示最近三次提交

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log -3 --pretty=oneline
a4a44dc0d972e50caf3c369082b6cd7031890598 (HEAD -> master) 第四次提交操作
c51926c8cf867dbd947c2faef970b75147598cccd 提交
a278212ed8ec7ceedbe261be57732019f7b4d785 第一次修改git01.txt文本
```

切换版本前 git01.txt 内容如下：

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git    第三次修改
git    第四次修改
```

回退到上一版本

执行 `git reset --hard HEAD^`

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git reset --hard HEAD^
HEAD is now at c51926c 提交
```

- 1 `HEAD^`: 将指针指向一个版本，如果是上一个就是 `HEAD^^`，上上一个 `HEAD^^^`，但这样记就比较麻烦，如果回退版本较多，简写为 `HEAD~100` 往前回退100个版本 ~后跟数字即可

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log
commit c51926c8cf867dbd947c2faef970b75147598ccd (HEAD -> master)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:57:22 2020 +0800

    提交

commit a278212ed8ec7ceedbe261be57732019f7b4d785
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:50:57 2020 +0800

    第一次修改git01.txt文本

commit 79eda75dda912c48aaca514a5a682fc06b3d32e9
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:47:50 2020 +0800

    git 版本库初始化与文件提交

```

查看回退后的git01.txt内容如下:

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git      第三次修改

```

回退操作已经完成，但此时如果想要回到未来的版本即最新的版本怎么办呢？其实这里也比较简单，前面说到针对提交后的版本库，每个版本均会有一个唯一标识，这里找到对应版本标识即可完成回到未来版本的操作如下

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log -3 --pretty=oneline
c51926c8cf867dbd947c2faef970b75147598ccd (HEAD -> master) 提交
a278212ed8ec7ceedbe261be57732019f7b4d785 第一次修改git01.txt文本
79eda75dda912c48aaca514a5a682fc06b3d32e9 git 版本库初始化与文件提交

```

执行git reset 命令如下:

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git reset --hard a4a44
HEAD is now at a4a44dc 第四次提交操作

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git log --pretty=oneline
a4a44dc0d972e50caf3c369082b6cd7031890598 (HEAD -> master) 第四次提交操作
c51926c8cf867dbd947c2faef970b75147598ccd 提交
a278212ed8ec7ceedbe261be57732019f7b4d785 第一次修改git01.txt文本
79eda75dda912c48aaca514a5a682fc06b3d32e9 git 版本库初始化与文件提交

```

如果此时回到某一个版本后直接关闭了当前git 命令窗口 怎么样才能回到未来版本呢？因为此时未来版本的唯一标识id 在窗口中看不到了！

值得庆幸的是，Git早已为你想到了这种情况，Git提供了一个命令 `git reflog` 用来记录用户操作的每一次命令，效果如下:

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git reflog
a4a44dc (HEAD -> master) HEAD@{0}: reset: moving to a4a44
c51926c HEAD@{1}: reset: moving to HEAD^
a4a44dc (HEAD -> master) HEAD@{2}: commit: 第四次提交操作
c51926c HEAD@{3}: reset: moving to HEAD
c51926c HEAD@{4}: commit: 提交
a278212 HEAD@{5}: reset: moving to HEAD
a278212 HEAD@{6}: commit: 第一次修改git01.txt文本
79eda75 HEAD@{7}: commit (initial): git 版本库初始化与文件提交

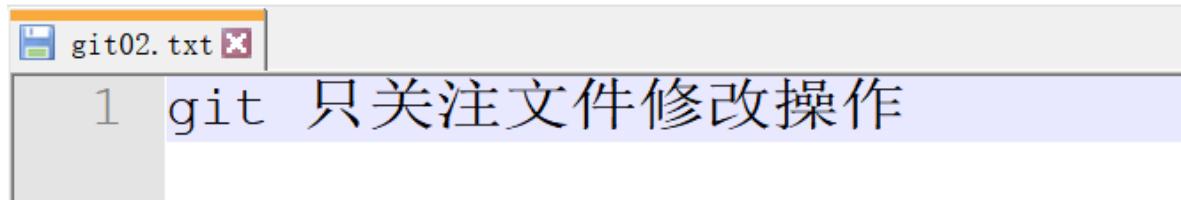
```

1 | git reflog: 查看记录在本地的HEAD和分支引用在过去指向的位置。

6.3. 文件删除

在Git中，删除文件同样是一个修改操作，即在Git世界中，Git仅仅关注文件是否被修改(文件添加，更新，删除)

在工作区添加新文件git02.txt 内容如下：



将文件添加到版本库

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    git02.txt

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

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git add git02.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   git02.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git commit -m '添加git02.txt文本文件'
[master 0540906] 添加git02.txt文本文件
 1 file changed, 1 insertion(+)
 create mode 100644 git02.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$
```

文件提交到版本库后，在工作目录执行手动删除操作后执行`git status`命令可以看到Git能够追踪到文件被删除情况，注意此时版本库中文件并没有被删除，只是工作目录中文件被删除！

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:   git02.txt

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

如果此时发现文件是被误删除呢，不用担心，这里可以将误删除的文件重新从版本库中检出，执行命令：

```
1 | git checkout -- 文件名
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git checkout -- git02.txt
```

如果确定是要执行删除操作 执行git rm 命令即可

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git rm git02.txt
rm 'git02.txt'
```

7. 远程仓库

Git是一个分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。截止目前，并没有看到分布式的环境，因为以上的操作都在在本地发生的，对于Git,除了前面提到的本地版本库外，Git 支持远程仓库的托管服务即使用者可以将本地版本库中的文件托管到远程服务器进行存储，这样就极大的方便开发，无论你走到哪，只要你的机器能够联网，就可以通过远程的仓库地址得到一份相同的项目库文件，并且下载到本地的文件版本记录与远程文件版本保持一致，并且可以很方便的实现多人协同开发操作。

对于Git 远程仓库，GitHub(<https://github.com/>)是比较知名的一个，目前已已被微软收购，而国内比较知名的当属码云(<https://gitee.com/>)了，当然除了这些远程仓库外，在公司，有的公司出于安全考虑，可能会自己搭建一套Git服务区来自Git的远程仓库，对于内部仓库会有专门人员来进行维护操作。这里以当下比较流行的GitHub仓库来介绍Git 远程仓库基本操作与使用。

7.1. 克隆远程项目到本地

初次接触GitHub的话，在没有账号的情况下，也可以很容易得到一些比较好的开源项目，即通过克隆的方式将远程项目下载到本地 比如现在Java12 都已经发布，想要找一些Java12 相关的项目入门，这里就可以借助GitHub来发现你想要的相关项目

The screenshot shows the GitHub search interface. The search bar at the top contains the query 'java12'. Below the search bar, there are navigation links for 'Why GitHub?', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. On the right side of the search bar, there are 'Sign in' and 'Sign up' buttons. To the left of the search results, there is a sidebar with links for 'Repositories' (181), 'Code' (426), 'Commits' (112), 'Issues' (0), 'Marketplace' (1), 'Topics' (1), 'Wikis' (4), and 'Users' (32). Below this sidebar is another section titled 'Languages' with a link to 'Java' (101) and 'HTML' (3). The main search results area displays 181 repository results. The first result is 'jacoco/jacoco', described as 'JaCoCo - Java Code Coverage Library', with a Java icon, a star rating of 1.6k, and a 'Sort: Best match' dropdown. The second result is 'loiane/java12-examples', described as 'Java 12 Examples', with a Java icon and a star rating of 10. The third result is 'fajfelek/zad12', with a Java icon. At the bottom of the results, it says 'MIT license Updated on 26 Mar'.

克隆项目到本地执行 git clone 命令

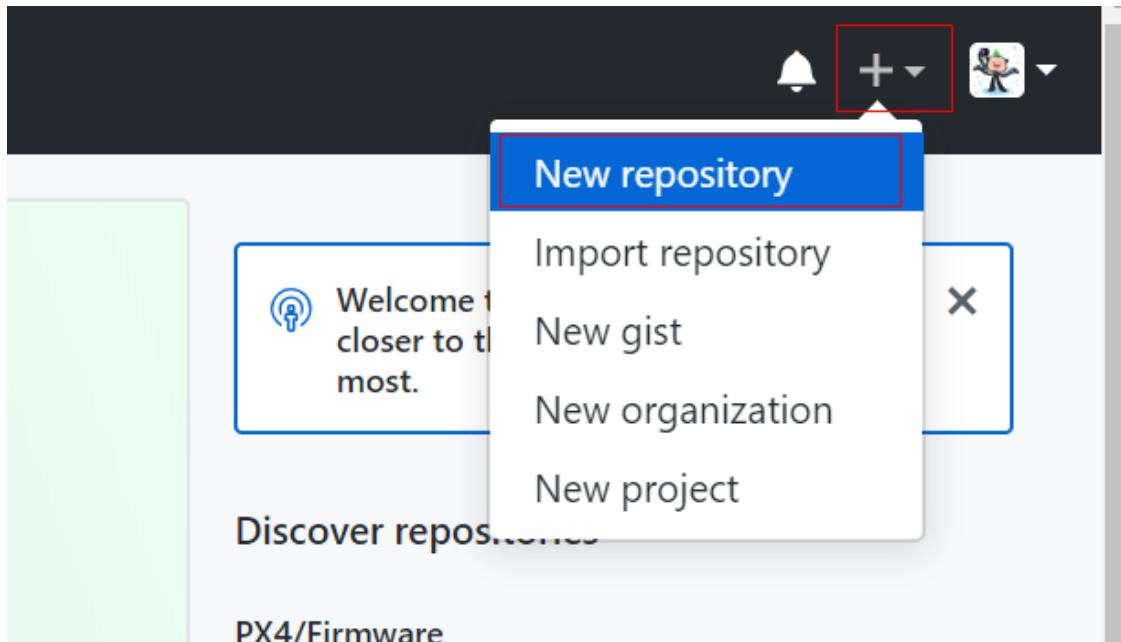
在本地磁盘指定目录下右键 `git bash here` 操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git
$ git clone https://github.com/loiane/java12-examples.git
Cloning into 'java12-examples'...
remote: Enumerating objects: 11, done.
remote: Total 11 (delta 0), reused 0 (delta 0), pack-reused 11
Unpacking objects: 100% (11/11), done.
```

此时在本地就得到了远程的相关项目。

7.2. 将本地库推送到远程

- 创建本地版本库并提交文件到本地库(这里以前面创建版本库为主，这里不再赘述)
- 使用GitHub创建远程库git01 如下



Create a new repository

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

Owner

zsyue ▾

Repository name *

/ git01 ✓

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

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer.

Add .gitignore: None ▾

Add a license: None ▾



[Create repository](#)

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH <https://github.com/zsyue/git01.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a README, LICENSE, and .gitignore.

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

```
echo "# git01" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/zsyue/git01.git
git push -u origin main
```



- 推送本地Git版本库文件到远程仓库git01

推送本地库文件到远程有两种方式

使用Https

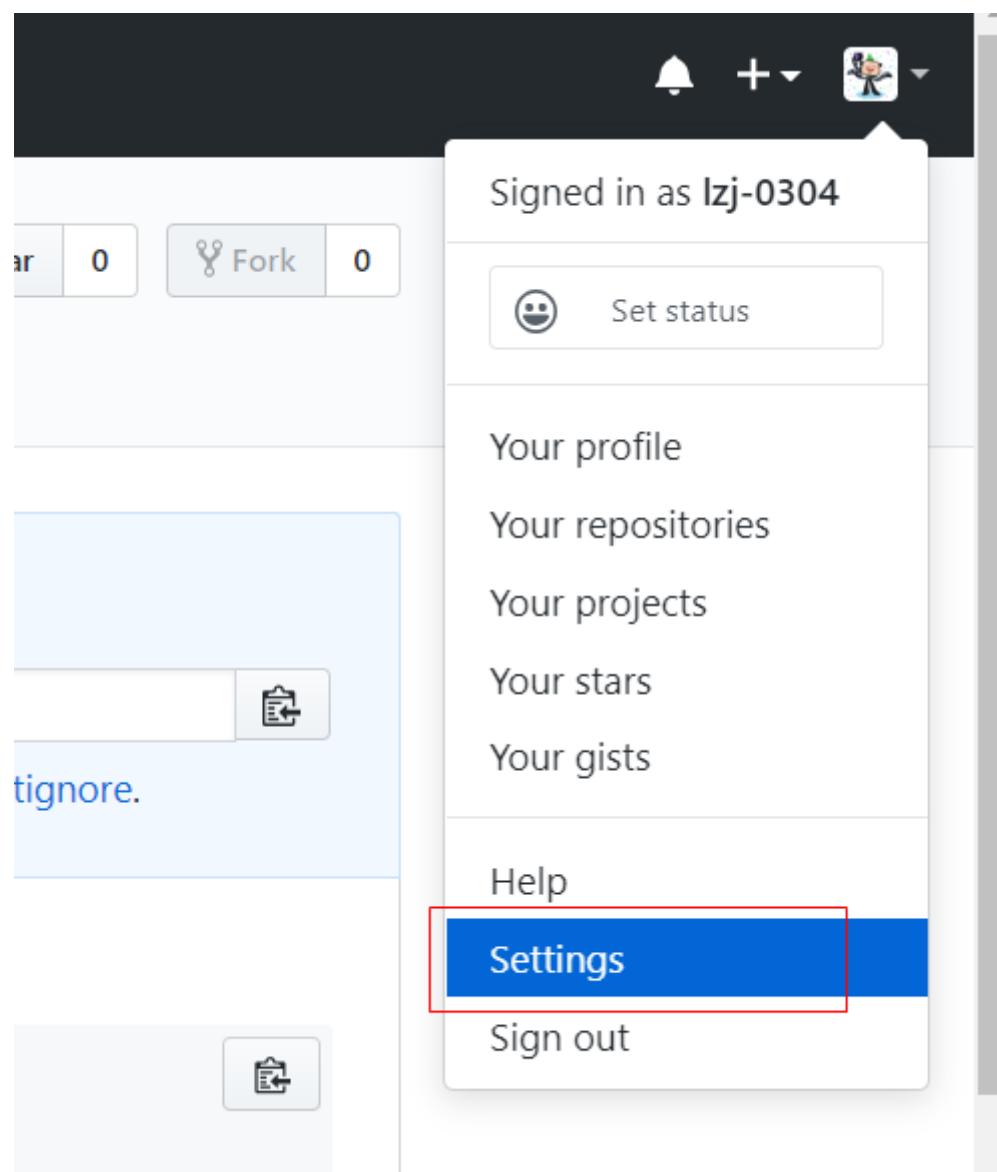
使用SSH

使用Https 比较简单，使用SSH 加密方式是Git 建议的一种推送，时间上与响应上效率都更高，这里介绍SSH推送方式配置

1. 使用本地Git客户端生成SSH公钥与私钥 执行命令 `ssh-keygen -t rsa -C "GitHub账户邮箱"`

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ ssh-keygen -t rsa -C "244173220@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa):
/c/Users/Administrator/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa.
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:83UU2wPhXr1YOWnEEeUzNkDMI2nKRWs7WKdyZf9c6Bw 244173220@qq.com
The key's randomart image is:
+---[RSA 3072]---+
| ..==+=|=|
| =0++^+| |
| . ++.=+#o| |
| o+ B.B.B| |
| So =.oEo.| |
| oo..+ oo| |
| . o o| |
+---[SHA256]---+
```

2. GitHub 公钥配置



Personal settings

- Profile
- Account
- Emails
- Notifications
- Billing
- SSH and GPG keys**
- Security
- Sessions

SSH keys

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot common SSH Problems.

GPG keys

There are no GPG keys associated with your account.

Learn how to [generate a GPG key](#) and add it to your account.

New SSH key

New GPG key

设置公钥标题与公钥串内容

SSH keys / Add new

Title

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQgQC+lsTqk0rEsjDY1FWXrDtn6bypL8blynwxSGcskJ6g5OThsj/zW4VJpuJytz6l59
eWCfvkbUytVMSN1K4SqUcvFDv7gGiNMXbOPPFdMmny8jL9P79ipPcXUMbtICjqWmC8FA/A4gRyxV012lohlMA73G
MAtnKzOLkB1If/ocoA3nHb0ilHIEM0XZ8gUByC6KSA6hPwHYZKfsSB8+oY3YlhVacOJ1z8M76NnhRLYgAvq+WMvH63
STEEoYYT3UUa6N67ih8AGzgnlo7FeEkneY8nC9CEUg5PyRxbPeramNDGtcUUrcBD3uzyk1Zgc11EHvnV/RxHLwX2fgTE4
LOPYf92ta48j2HWK2fLBodFko4S7ZW52MGVKkrvUV8vXUJSicEDaF16KRi4nRnb4/mOl8s8tZOLYya149LZhlpAP3Ciz71
Ni3A5QLfr3Q5dcfgsxWyejxXuD1hvCPnb0eXE2wlccvPIZdcO3/Ay+aGY9uFsb1s92xcrl+mFvHbU=
244173220@qq.com
```

Add SSH key

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



git test

12:1b:4e:9d:86:d9:bd:33:b7:d5:d7:07:98:59:d5:68

SSH

Added on 31 Jan 2020

Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot common SSH Problems.

3. 检查测试链接 执行命令 `ssh -T git@github.com`

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ ssh -T git@github.com
Hi zsyue! You've successfully authenticated, but GitHub does not provide shell access.
```

4. 使用SSH执行远程推送操作

Quick setup — if you've done this kind of thing before

Set up in Desktop

or

HTTPS

SSH

git@github.com:zsyue/git01.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a `README`, `LICENSE`, and `.gitignore`.

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

```
echo "# git01" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:zsyue/git01.git
git push -u origin main
```

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

```
git remote add origin git@github.com:zsyue/git01.git
git branch -M main
git push -u origin main
```

根据GitHub说明 执行SSH 推送 首先绑定远程地址到本地执行命令

```
git remote add origin git@github.com:zsyue/git01.git
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git remote add origin git@github.com:zsyue/git01.git
```

执行远程推送操作 将本地库推送到远程master 主分支 执行命令

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

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git push -u origin master
Warning: Permanently added the RSA host key for IP address '13.250.177.223' to the list of known hosts.
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (15/15), 1.36 KiB | 462.00 KiB/s, done.
Total 15 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To github.com:zsyue/git01.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

查看远程仓库 到此,使用SSH推送本地版本库文件到GitHub远程库操作成功完成!

The screenshot shows the GitHub repository page for 'zsyue/git01'. At the top, there's a header with the repository name and some stats: 5 commits, 1 branch, 0 packages, 0 releases, and a 'Fetching contributors' button. Below the header, there's a list of commits. The most recent commit is by 'zsyue' and is titled '添加git02.txt文本文件'. This commit was made '1 hour ago'. There are also other commits listed, such as 'git01.txt' and 'git02.txt'. At the bottom of the page, there's a section for adding a README, with a 'Add a README' button.

8. Git 分支操作

开发企业项目中在使用Git 或者其他类似版本控制软件对项目版本进行管理时，多人合作的项目在开发时通常不会直接在主干master 上进行操作，而是重新开辟新的分支，在新的分支上进行开发 调试 等操作，当项目调试通过时才会将分支项目的代码合并到主干中，这是在实战中比较好的一种策略，特别是多人协同开发一个项目的情况下尤其明显。Git 对于分支操作提供了一下基本命令：

命令	描述
git checkout branch	切换到指定分支
git checkout -b new_branch	新建分支并切换到新建分支
git branch -d branch	删除指定分支
git branch	查看所有分支, 并且*号标记当前所在分支
git merge branch	合并分支
git branch -m -M oldbranch newbranch	重命名分支, 如果newbranch名字分支已经存在, 则需要使用-M强制重命名, 否则, 使用-m进行重命名。

8.1. 本地分支创建、合并、重命名与删除

以前面git版本库为例。

- 创建本地分支、并查看分支

默认Git 版本库所在分支为master 通常称为项目的主干, 开发中通常会在主干上创建新的分支类进行本地开发工作 使用命令 `git checkout -b new_branch` 创建分支

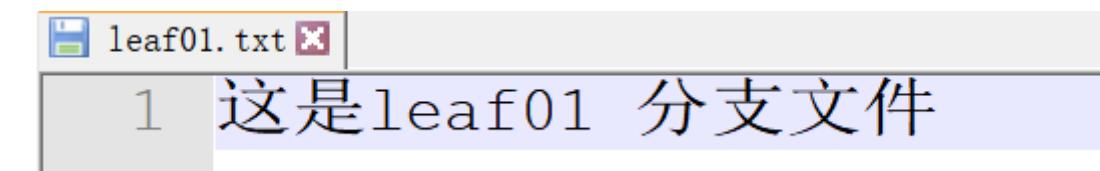
```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git checkout -b leaf01
Switched to a new branch 'leaf01'
```

分支创建完毕 该命令会自动切换到新建分支上 如上图

使用命令 `git branch` 查看分支列表 * 号标记为当前git 所在分支

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git branch
* Leaf01
  master
```

- 分支上添加文件leaf01.txt 添加操作同主干添加命令 leaf01.txt 文件内容如下:



```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git add leaf01.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git commit -m '添加leaf01.txt 文件到分支leaf01'
[leaf01 228c204] 添加leaf01.txt 文件到分支leaf01
 2 files changed, 1 insertion(+), 1 deletion(-)
  delete mode 100644 git02.txt
  create mode 100644 leaf01.txt
```

- 切换到主干master 并执行合并操作

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git merge leaf01
Updating 0540906..228c204
Fast-forward
  git02.txt | 1 -
  leaf01.txt | 1 +
  2 files changed, 1 insertion(+), 1 deletion(-)
  delete mode 100644 git02.txt
  create mode 100644 leaf01.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ cat leaf01.txt
这是leaf01 分支文件

```

- 重命名分支leaf01->leaf02

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git checkout leaf01
Switched to branch 'leaf01'

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git branch
* Leaf01
  master

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git branch -m leaf01 leaf02

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf02)
$ git branch
* Leaf02
  master

```

- 删除分支(不能再待删除的分支上执行删除当前分支操作!!!)

```

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf02)
$ git branch -d leaf02
error: Cannot delete branch 'leaf02' checked out at 'D:/git'

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf02)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git branch -d leaf02
Deleted branch leaf02 (was 228c204).

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git branch
* master

```

8.2. 分支Push与Pull操作

相关命令操作

命令	描述
git branch -a	查看本地与远程分支
git push origin branch_name	推送本地分支到远程
git push origin :remote_branch	删除远程分支(本地分支还在保留)
git checkout -b local_branch origin/remote_branch	拉取远程指定分支并在本地创建分支

- 查看远程仓库 此时远程仓库只有主干master

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git branch -a
* master
  remotes/origin/master
```

- 新建分支leaf01 并执行分支推送操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01.txt)
$ git checkout -b leaf01
Switched to a new branch 'leaf01'
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git push origin leaf01
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'leaf01' on GitHub by visiting:
remote:     https://github.com/zsyue/git01/pull/new/leaf01
remote:
To github.com:zsyue/git01.git
 * [new branch]      leaf01 -> leaf01

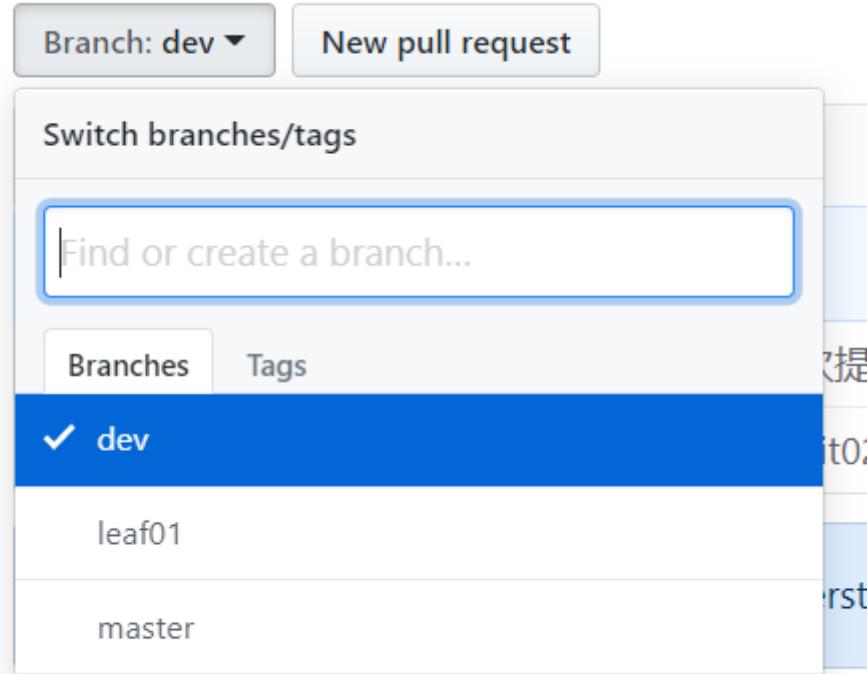
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ git branch -a
* leaf01
  leaf01.txt
  master
  remotes/origin/leaf01
  remotes/origin/master
```

远程查看 此时本地分支远程推送完成(注:推送远程分支名称可改)。

The screenshot shows a GitHub repository page for 'zsyue / git01'. At the top, there are navigation links for Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. Below the header, it says 'No description, website, or topics provided.' and has an 'Edit' button. A summary bar shows 5 commits, 2 branches, 0 packages, 0 releases, and 1 contributor. Under 'Your recently pushed branches:', the 'leaf01' branch is listed with a timestamp of '1 minute ago'. A red box highlights this entry. To the right of the branch list is a green 'Compare & pull request' button. Below the summary bar, there's a dropdown for 'Branch: master' and a 'New pull request' button. A sidebar on the left allows switching between branches and tags, with 'master' selected and 'leaf01' highlighted with a red box. The main content area shows a commit history with the latest commit being '0540906 1 hour ago' and a message about adding a README file. There are buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'.

- 远程创建dev分支并拉取分支到本地

GitHub支持远程分支在线创建，这里dev开发分支创建后分支文件内容与主干master一致



本地执行克隆 并拉取dev 分支到本地

执行`git clone` 命令

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/dev
$ git clone git@github.com:zsyue/git01.git
Cloning into 'git01'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 18 (delta 2), reused 18 (delta 2), pack-reused 0
Receiving objects: 100% (18/18), done.
Resolving deltas: 100% (2/2), done.
```

查看本地项目并执行命令 `git checkout -b dev origin/dev` 拉取dev 开发分支

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git branch -a
* master
  remotes/origin/leaf01
  remotes/origin/master

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git fetch
From github.com:zsyue/git01
 * [new branch]      dev      -> origin/dev

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git checkout -b dev origin/dev
Switched to a new branch 'dev'.
Branch 'dev' set up to track remote branch 'dev' from 'origin'.

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ git branch
* dev
  master
```

8.3. 分支操作冲突出现与解决

开发中对不同分支下同一文件进行修改后执行合并时就会出现文件修改冲突情况，这里说明一种比较常见的冲突问题 以master 和leaf01 两个分支进行演示说明。

8.3.1. 本地分支操作冲突

- 修改master与leaf01分支前git01.txt文件内容状态

master 主干文件内容:

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
```

leaf01 分支文件内容:

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
```

- 分支leaf01 下修改给git01.txt 并执行提交操作 效果如下

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (leaf01)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
分支leaf01 添加测试内容
```

- 主干master 下修改git01.txt 并执行提交操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
主干中添加测试内容
```

- 执行合并操作 此时发现git 在合并中产生冲突

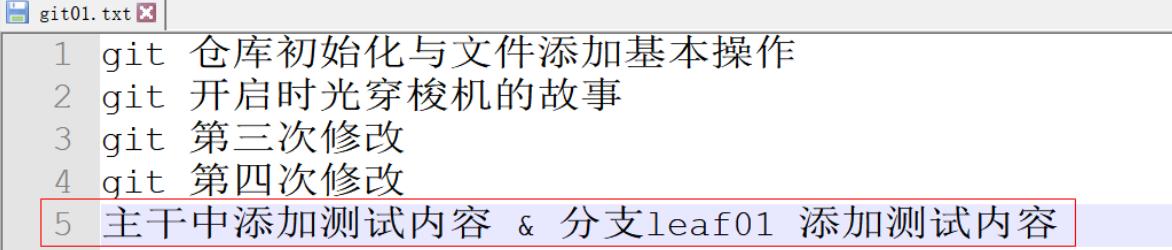
```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master)
$ git merge leaf01
Auto-merging git01.txt
CONFLICT (content): Merge conflict in git01.txt
Automatic merge failed; fix conflicts and then commit the result.
```

执行 `cat git01.txt` 可以看出冲突文件内容

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master|MERGING)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
<<<<< HEAD
主干中添加测试内容
=====
分支leaf01 添加测试内容
>>>>> leaf01
```

```
1 Git用<<<<<, =====, >>>>>标记出不同分支的内容
2 <<<<< HEAD 当前git 指向分支 这里指的为master
3 ===== 分离不同分支修改的内容
4 >>>>> Leaf01 Leaf01 与 master 在git01.txt 同一行同时出现了修改操作 这里git
是不允许发生的
```

此时出现冲突后 这里对git01.txt 内容进行修改 (实际开发视情况而定 这里将内容合并为一行)



```
1 git 仓库初始化与文件添加基本操作
2 git 开启时光穿梭机的故事
3 git 第三次修改
4 git 第四次修改
5 主干中添加测试内容 & 分支leaf01 添加测试内容
```

修改完毕master下执行提交即可 执行命令查看分支合并图 `git log --graph --`

`pretty=oneline`

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (master|MERGING)
$ git log --graph --pretty=oneline
* 414113bea9f4e6402cbfab3c0f9263d602473b6a (HEAD -> master) 主干中添加测试内容
* 228c204a421dc4f226606044d44f996c00ef7a6f (origin/leaf01) 添加leaf01.txt 文件到分支leaf01
* 05409065165399d8c127de2ff5b152fe03f32db0 (origin/master, origin/dev, dev) 添加git02.txt文本
文件
* a4a44dc0d972e50caf3c369082b6cd7031890598 第四次提交操作
* c51926c8cf867dbd947c2faef970b75147598cccd 提交
* a278212ed8ec7ceedbe261be57732019f7b4d785 第一次修改git01.txt文本
* 79eda75dda912c48aaca514a5a682fc06b3d32e9 git 版本库初始化与文件提交
```

8.3.2. 多人协同操作冲突

拉取远程库dev 并在本地创建dev开发库,执行命令 `git checkout -b dev origin/dev` 这里以同台机器不同窗口来模拟两个用户操作同一分支同一文件(实际开发时多人操作统一文件冲突情况比较常见)

这里两个客户端以c1与c2来描述

c1 客户端本地修改dev 分支git01.txt 文件并在本地执行提交操作 效果如下

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
C1修改了第五行内容
```

执行远程推送 将本地C1客户端提交的git01.txt 推送到远程dev分支

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 337 bytes | 337.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:zsyue/git01.git
  0540906..ccc4b46 dev -> dev
```

而此时C2 客户端在本地同样修改了dev分支git01.txt 文件并在本地执行提交

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ git add git01.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ git commit -m 'C2也修改了第五行内容'
[dev a5741b9] C2也修改了第五行内容
 1 file changed, 1 insertion(+)

Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
C2也修改了第五行内容
```

执行推送操作 此时冲突出现 原因是另外一个用户推送的文件与当前客户端推送内容存在冲突:

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git (dev)
$ git push
To github.com:zsyue/git01.git
 ! [rejected]          dev -> dev (fetch first)
error: failed to push some refs to 'git@github.com:zsyue/git01.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

此时解决方式Git 已有对应提示 Push 之前先执行Pull 操作 将远程文件拉取到本地 解决完冲突后再次执行Push 操作

冲突解决

先执行Pull 拉取操作

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:zsyue/git01
  1e99ff9..9436896  dev      -> origin/dev
Auto-merging git01.txt
CONFLICT (content): Merge conflict in git01.txt
Automatic merge failed; fix conflicts and then commit the result.
```

查看冲突文件内容

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev|MERGING)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
<<<<< HEAD
C2修改了第五行内容
=====
C1修改了第五行内容
>>>>> 943689660ba63d812ccabbd6eb630324a5a62378
```

这里在本地先处理冲突 将文本进行合并 然后提交 在 push 操作即可

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev|MERGING)
$ git add git01.txt

Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev|MERGING)
$ git commit -m '解决冲突'
[dev 9e58b31] 解决冲突

Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev)
$ cat git01.txt
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
C2修改了第五行内容
C1修改了第五行内容

Administrator@DESKTOP-AI9I3US MINGW64 /d/dev/git01 (dev)
$ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 576 bytes | 288.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:zsyue/git01.git
  9436896..9e58b31 dev -> dev
```

远程仓库内容如下:

The screenshot shows a GitHub repository page for 'zsyue / git01'. The 'Code' tab is selected. A dropdown menu 'Branch: dev' is open, and 'git01 / git01.txt' is highlighted. The file content is displayed below:

```
git 仓库初始化与文件添加基本操作
git 开启时光穿梭机的故事
git 第三次修改
git 第四次修改
C2修改了第五行内容
C1修改了第五行内容
```

The last two lines of the file content ('C2修改了第五行内容' and 'C1修改了第五行内容') are highlighted with a red box.

9. 标签管理

标签操作基本命令 git tag

命令	描述
git tag tag_name	新建标签 默认为HEAD
git tag -a tag_name -m 'xxx'	添加标签并指定标签描述信息
git tag	查看所有标签
git tag -d tag_name	删除一个本地标签
git push origin tag_name	推送本地标签到远程
git push origin --tags	推送全部未推送过的本地标签到远程
git push origin :refs/tags/tag_name	删除一个远程标签

同大多数 VCS 一样，Git 也可以对某一时间点上的版本打上标签。开发中在发布某个软件版本（比如 v1.0 等等）的时候，通常使用版本库软件命令来对某一版本打上一个标签，以方便标识。

- 添加本地标签

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag v1.0

Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag
v1.0
```

标签默认会打到最近的一次提交记录上

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git log --pretty=oneline --abbrev-commit
efbc692 (HEAD -> master, tag: v1.0, origin/master, origin/HEAD) C1修改了内容
0540906 添加git02.txt文本文件
a4a44dc 第四次提交操作
c51926c 提交
a278212 第一次修改git01.txt文本
79eda75 git 版本库初始化与文件提交
```

也可以对某个指定的提交标识进行标记

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag v0.1 0540906

Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag
v0.1
v1.0
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git log --pretty=oneline --abbrev-commit
efbc692 (HEAD -> master, tag: v1.0, origin/master, origin/HEAD) C1修改了内容
0540906 (tag: v0.1) 添加git02.txt文本文件
a4a44dc 第四次提交操作
c51926c 提交
a278212 第一次修改git01.txt文本
79eda75 git 版本库初始化与文件提交
```

打标签时给标签添加说明信息 `git tag -a v0.0.1 -m '软件基础版本' a4a44dc`

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag -a v0.0.1 -m '软件基础版本' a4a44dc
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag
v0.0.1
v0.1
v1.0
```

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git show v0.0.1
tag v0.0.1
Tagger: zsyue <244173220@qq.com>
Date:   Fri Jan 31 19:32:13 2020 +0800
```

软件基础版本

```
commit a4a44dc0d972e50caf3c369082b6cd7031890598 (tag: v0.0.1)
Author: zsyue <244173220@qq.com>
Date:   Fri Jan 31 16:59:52 2020 +0800
```

第四次提交操作

推送本地标签到远程

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
git push origin v0.0.1
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 179 bytes | 179.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:zsyue/git01.git
 * [new tag]           v0.0.1 -> v0.0.1
```

远程查看

No description, website, or topics provided.

Manage topics

6 commits 3 branches 0 packages 1 release 1 contributor

Your recently pushed branches:

- leaf01 (about 1 hour ago)
- dev (10 minutes ago)

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

zsyue C1修改了内容 Latest commit efbc692 21 minutes ago

git01.txt C1修改了内容 21 minutes ago

git02.txt 添加git02.txt文本文件 2 hours ago

Add a README

The screenshot shows a GitHub repository page for 'zsyue / git01'. The 'Tags' tab is selected. A single tag 'v0.0.1' is listed, along with its commit hash 'a4a44dc', a 'zip' file, and a 'tar.gz' file.

- 删除本地标签

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag
v0.0.1
v0.1
v1.0

Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag -d v0.0.1
Deleted tag 'v0.0.1' (was d44bdf2)

Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git tag
v0.1
v1.0
```

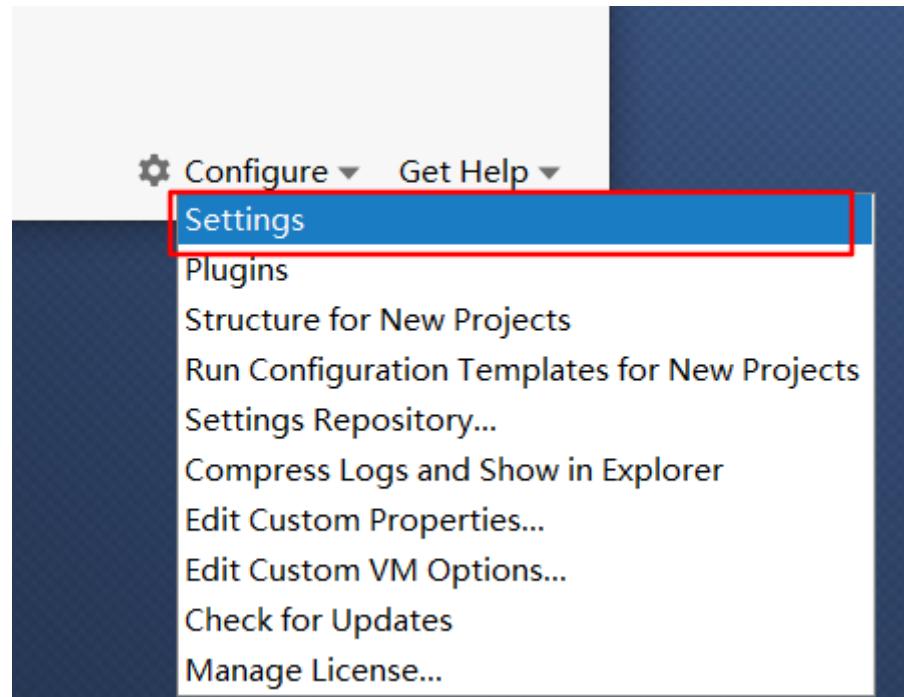
- 删除远程标签

```
Administrator@DESKTOP-AI9I3US MINGW64 /d/git/git01 (master)
$ git push origin :refs/tags/v0.0.1
To github.com:zsyue/git01.git
 - [deleted]           v0.0.1
```

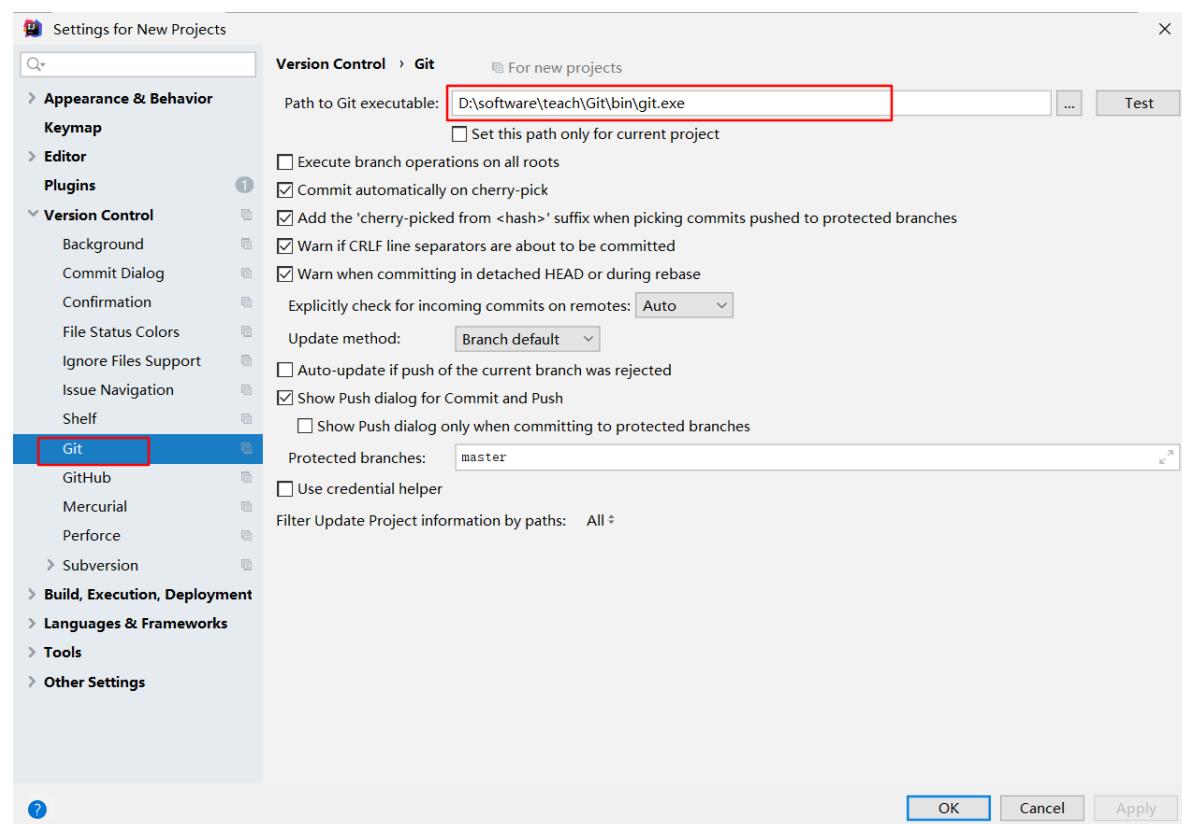
10. Idea下Git基本操作

10.1. 环境集成配置

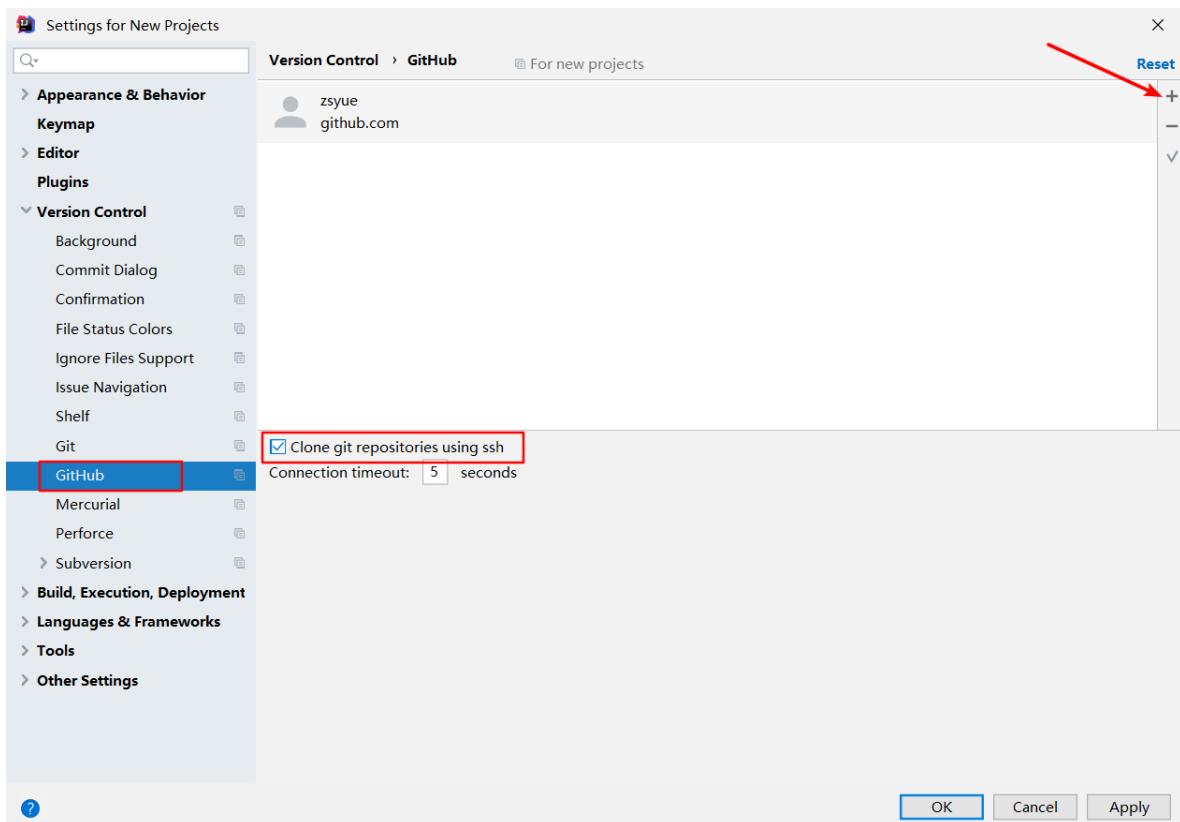
全局setting 下 git 环境指定



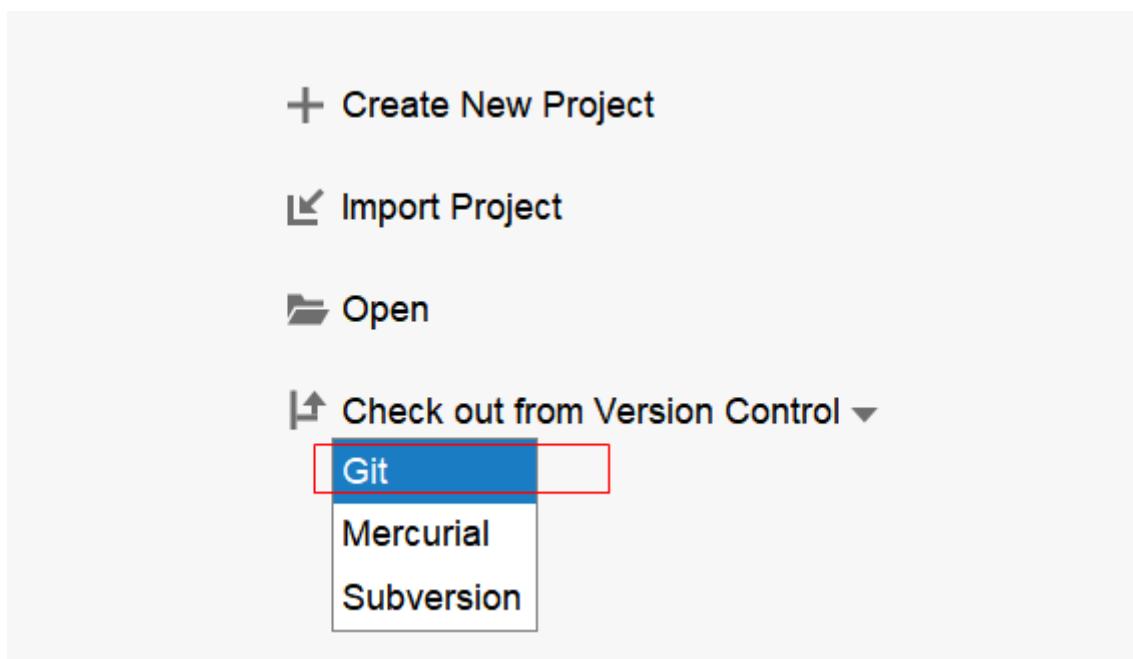
指定git.exe 文件路径



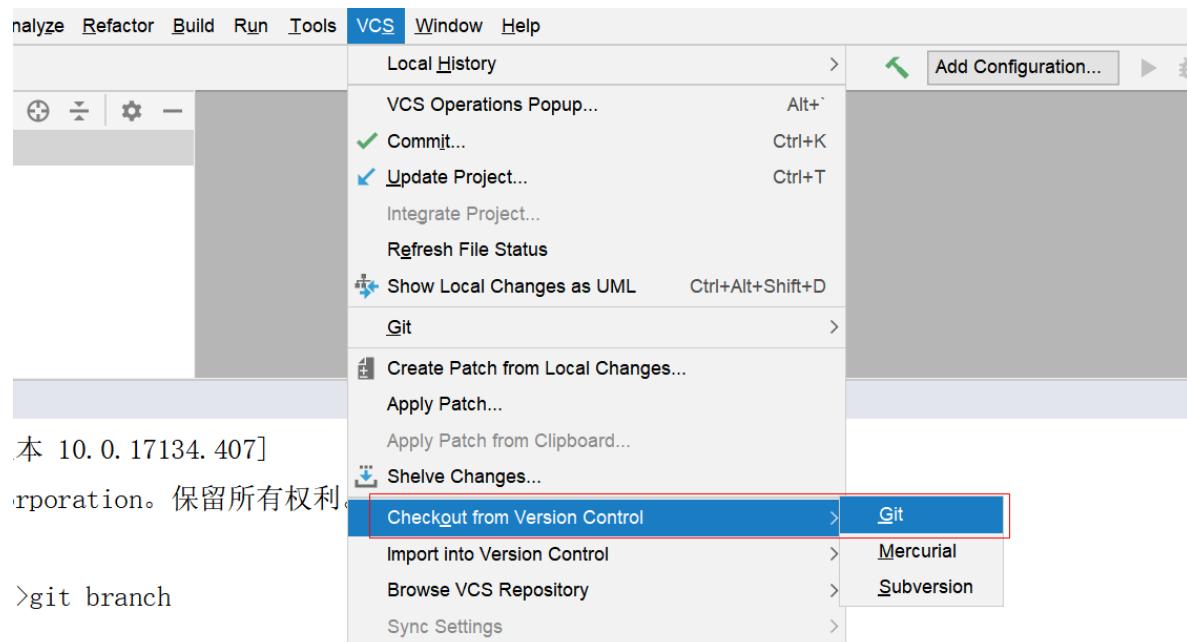
远程GitHub 配置



10.2. 克隆远程仓库到本地



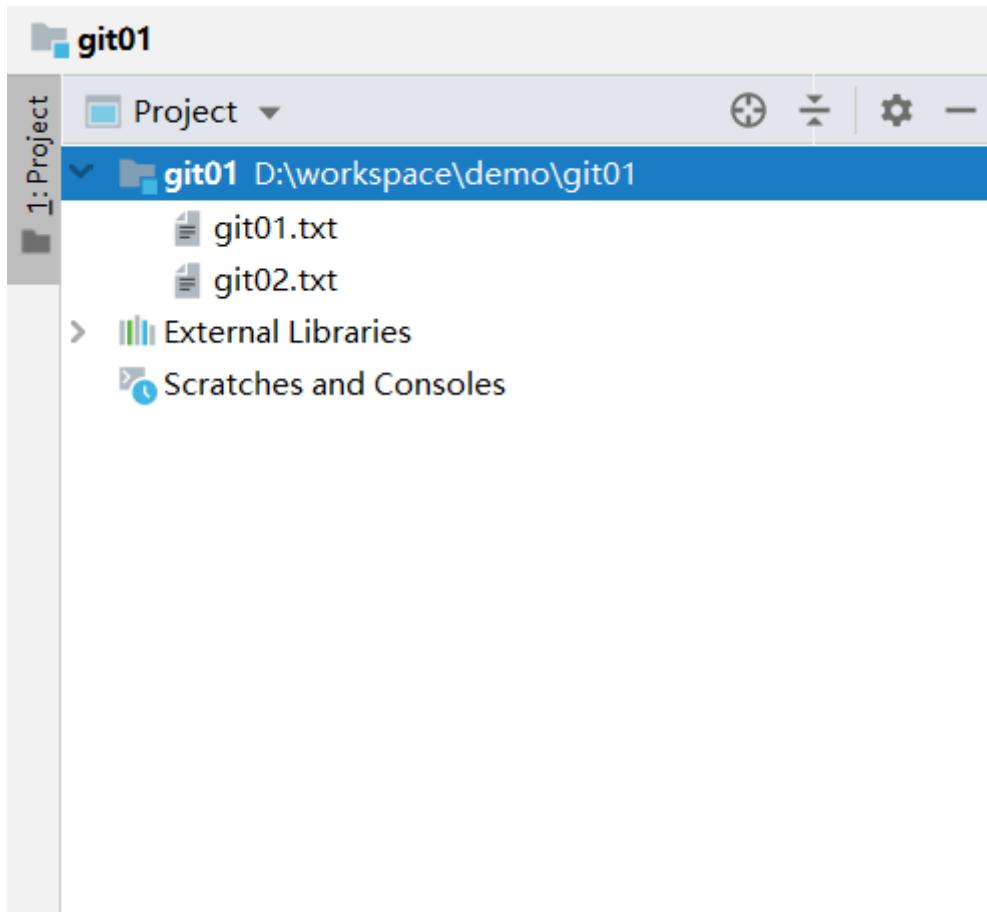
或者在Idea 打开情况下



设置远程url 地址(这里使用ssh) 并指定本地项目路径



克隆到本地 idea 打开效果:



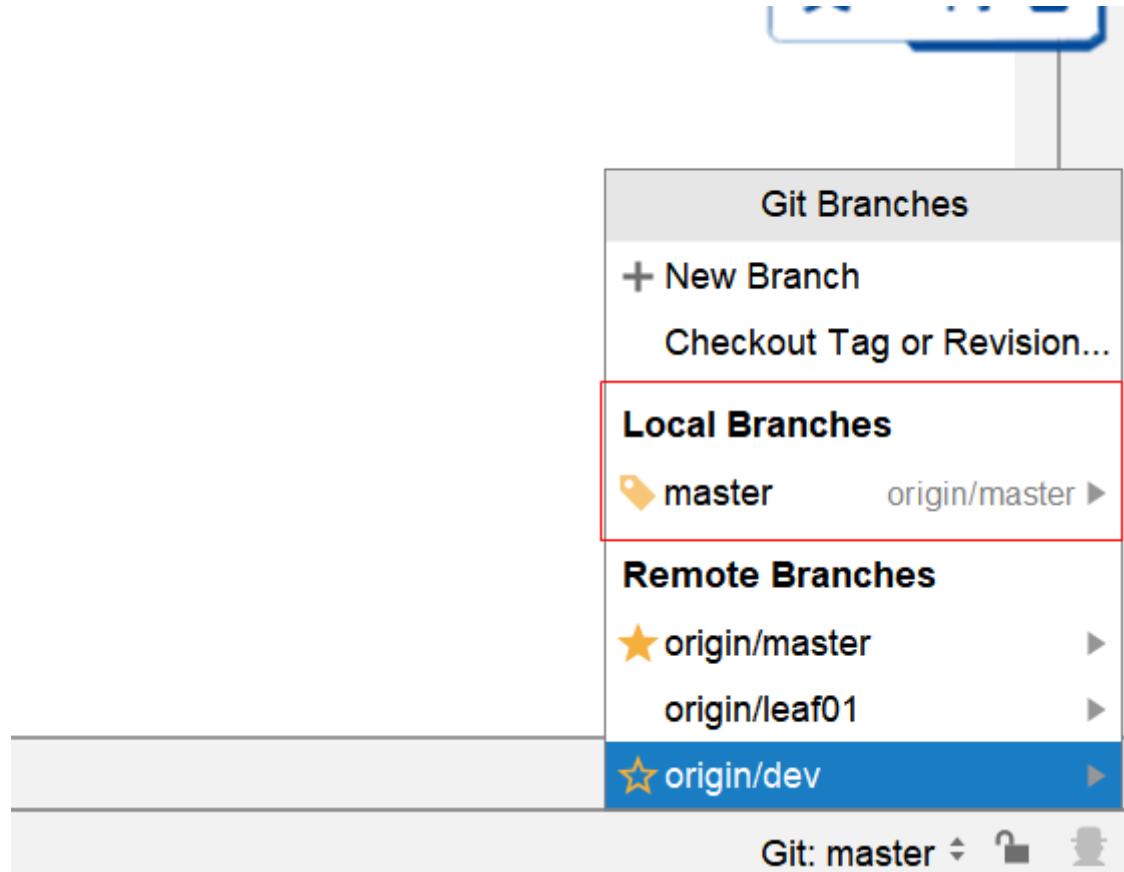
10.3. Idea 下分支拉取与远程推送

10.3.1. 分支拉取

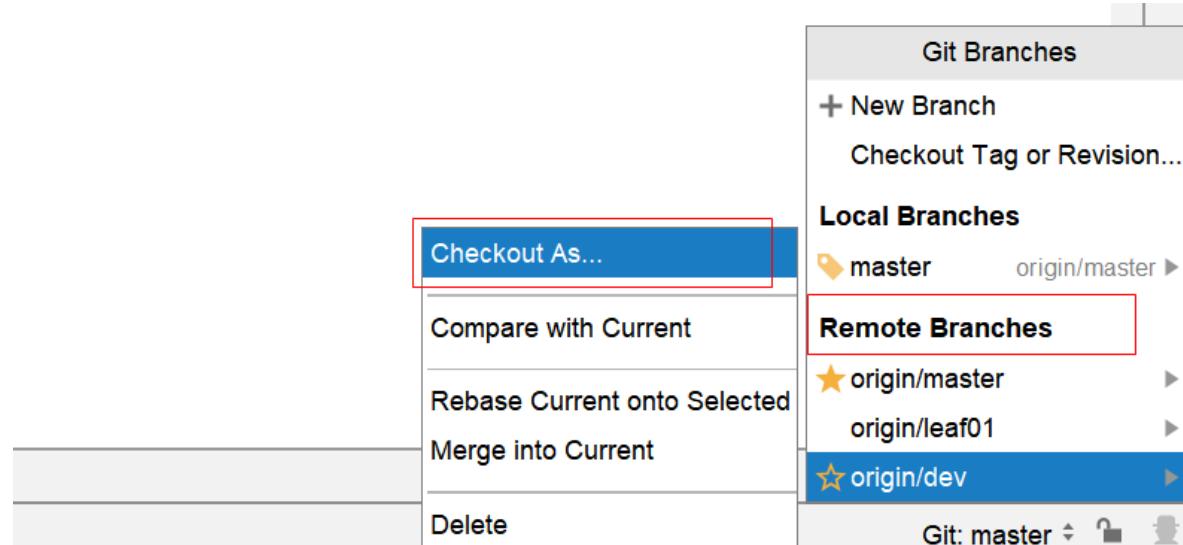
默认情况下，使用Idea检出远程仓库master文件到本地，这里点在idea中可以方便查看(与使用命令方式一致)

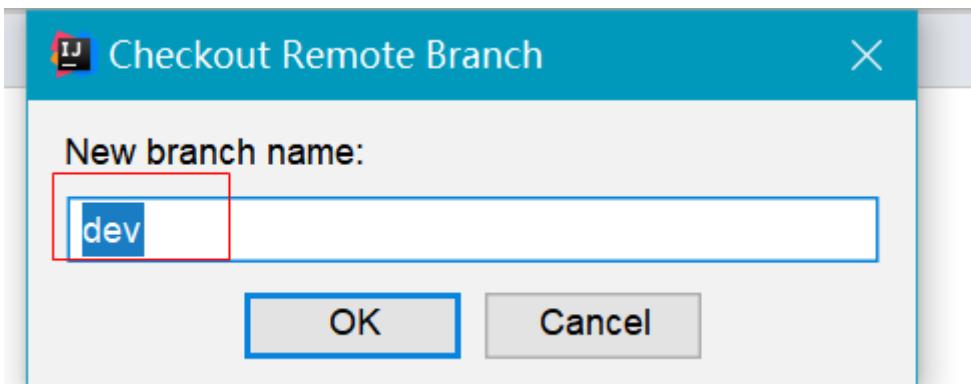
```
Terminal: Local +  
Microsoft Windows [版本 10.0.17134.407]  
(c) 2018 Microsoft Corporation。保留所有权利。  
  
C:\java\lotbyte\git01>git branch  
* master  
  
C:\java\lotbyte\git01>
```

或者 idea 右下角 查看如下:

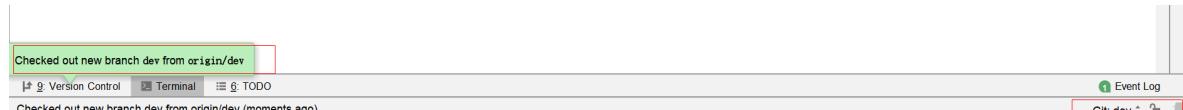


检出远程库dev 分支到本地 这里可以使用前面的命令 `git checkout -b local_branch origin/remote_branch` 或者使用图形化界面进行操作 这里介绍第二种 图形化操作 在idea 右下角拉取远程dev 到本地





检出成功后提示信息如下:

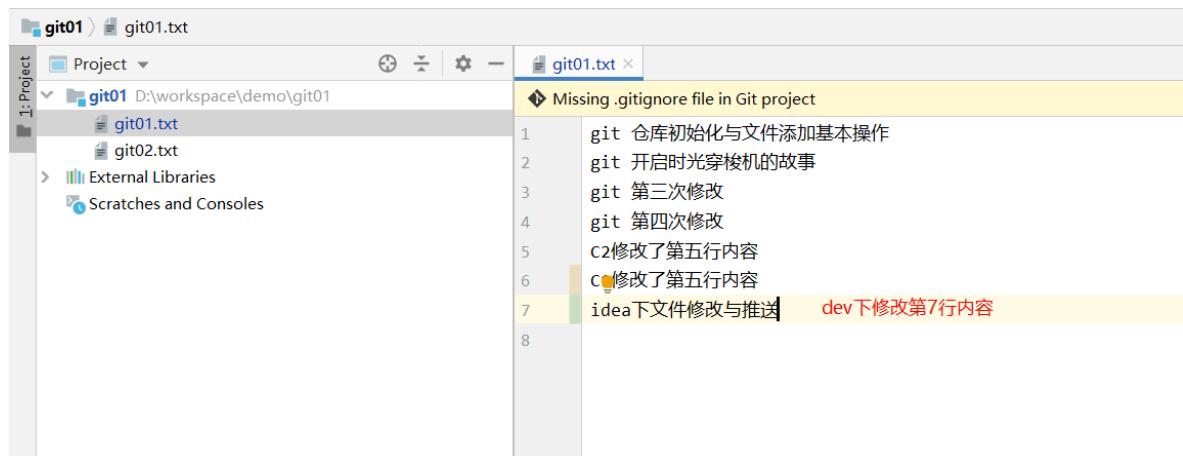


命令查看分支情况

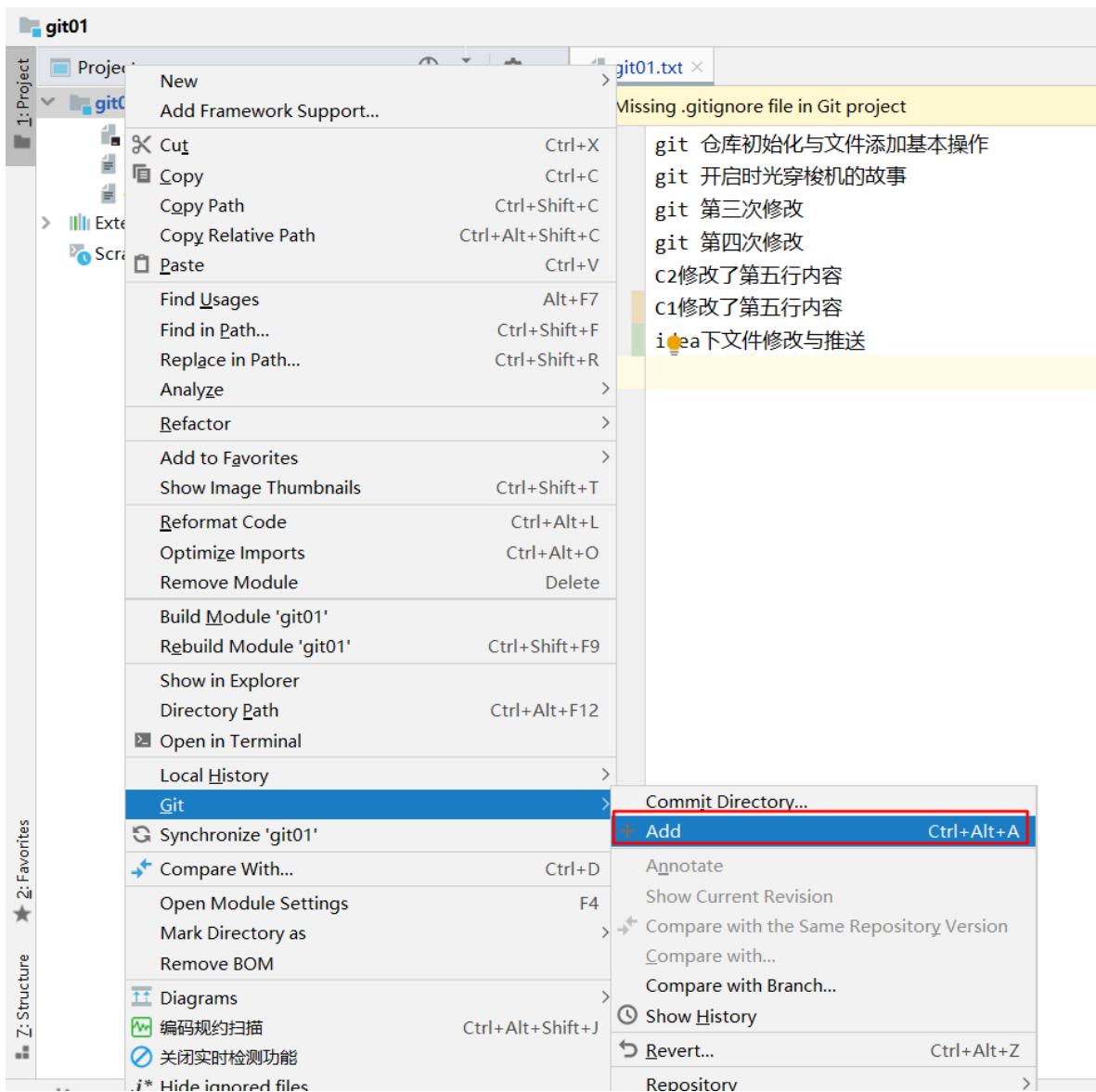
```
D:\workspace\demo\git01>git branch
* dev
  master
```

10.3.2. 远程推送

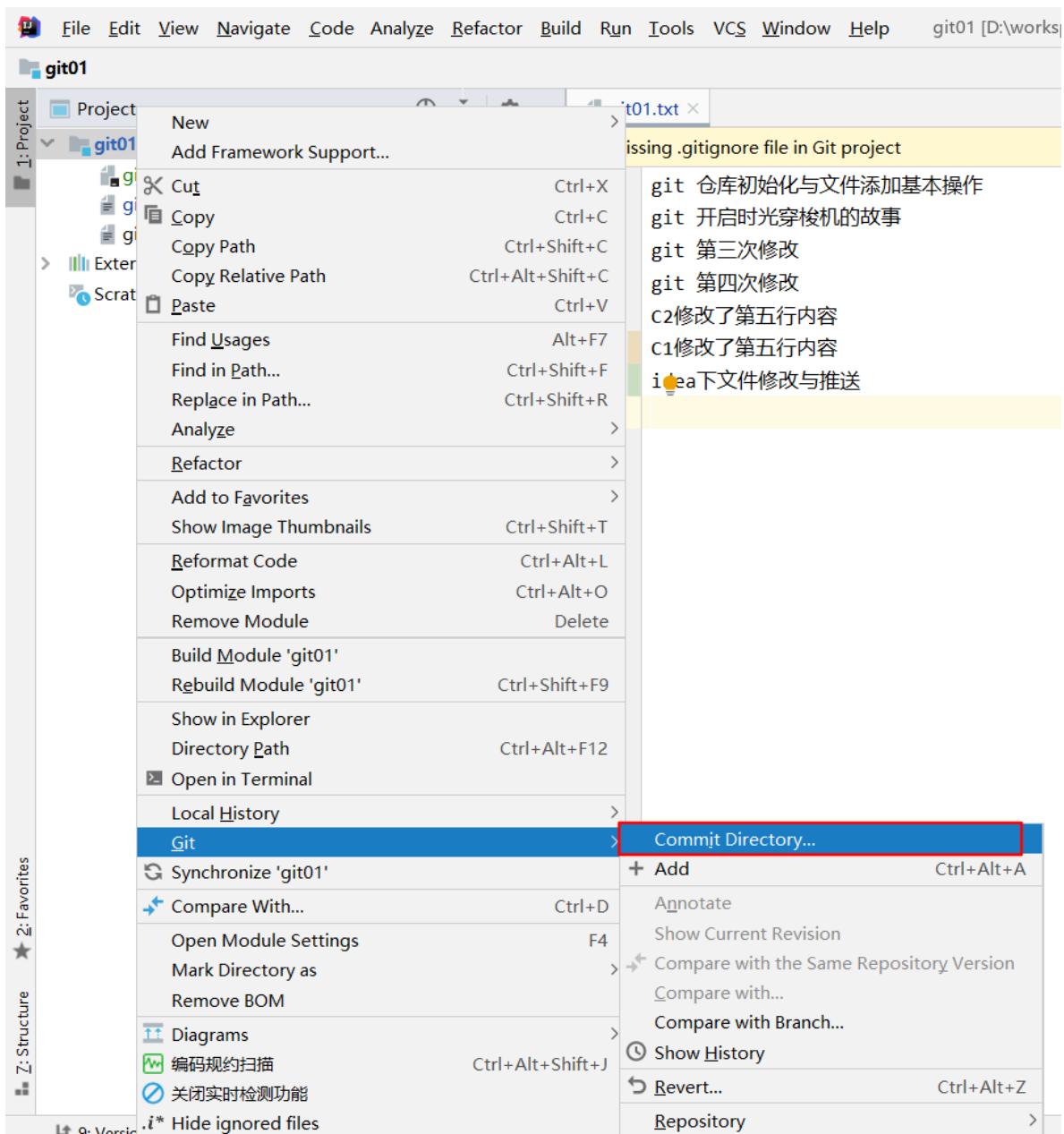
在以上拉取到本地的dev 分支下修改文件git01.txt 然后执行推送操作



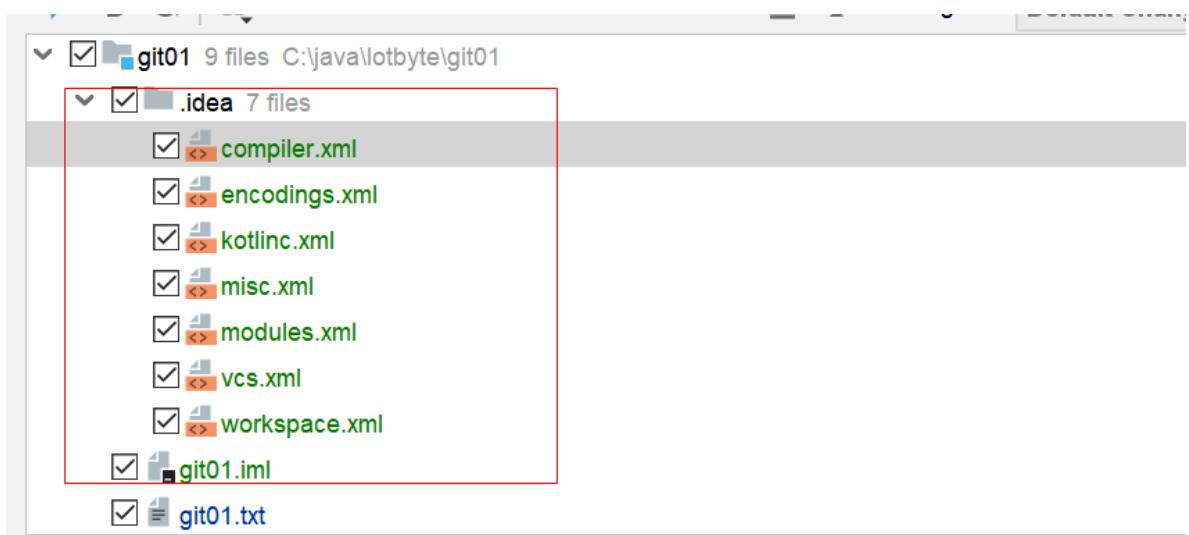
修改为文件添加到暂存区



执行本地提交操作



此时提交出现提交窗口



这里发现仅仅修改了git01.txt文件为什么会出现这么多新增的文件？这些文件也不是从远程拉取到本地的，没有必要进行提交操作。如果提交会影响他人本地环境!!! 此时的解决方案比较简单，将暂存区的无用文件转移到工作区即可。执行命令 `git reset HEAD 文件名` 即可达到效果（取消提交操作，这里需要借助git命令进行操作，后续会介绍一种简单办法）

执行命令 `git reset HEAD .idea/* git reset HEAD git01.iml` 将暂存区无需提交文件转移到工作区

执行 `git status` 查看暂存区文件，即将提交到本地版本库文件，发现待提交的文件只有git01.txt

```
C:\java\lotbyte\git01>git status
On branch dev
Your branch is up to date with 'origin/dev'.

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

    modified:   git01.txt

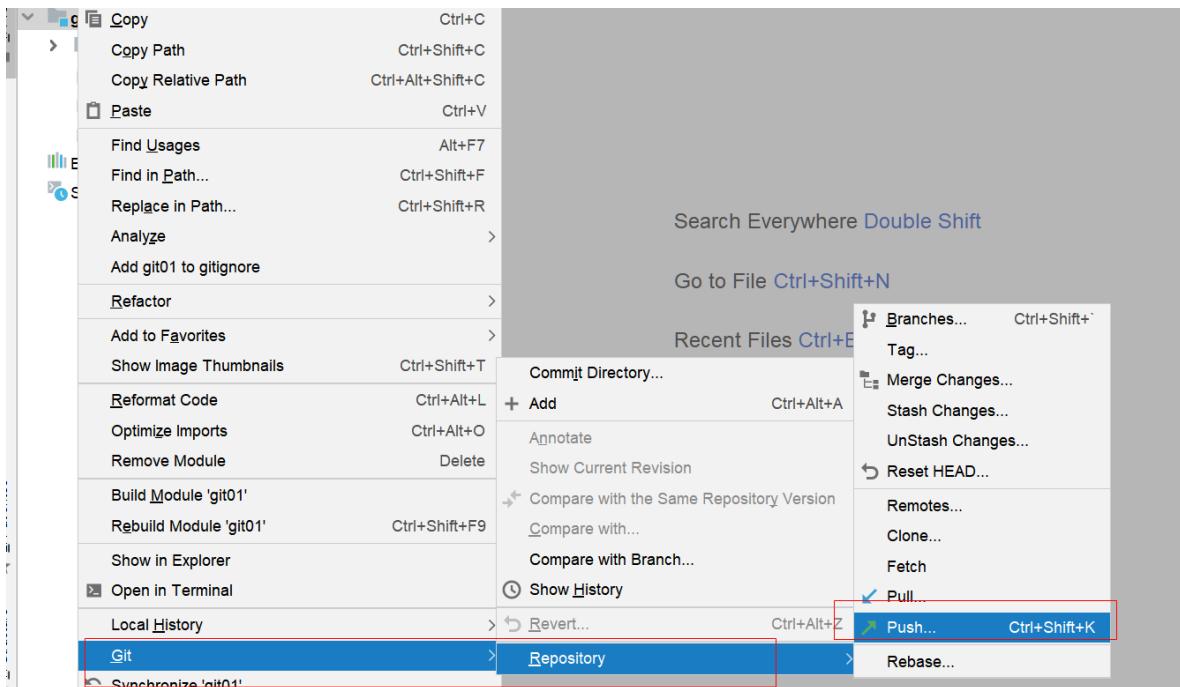
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .idea/
    git01.iml
```

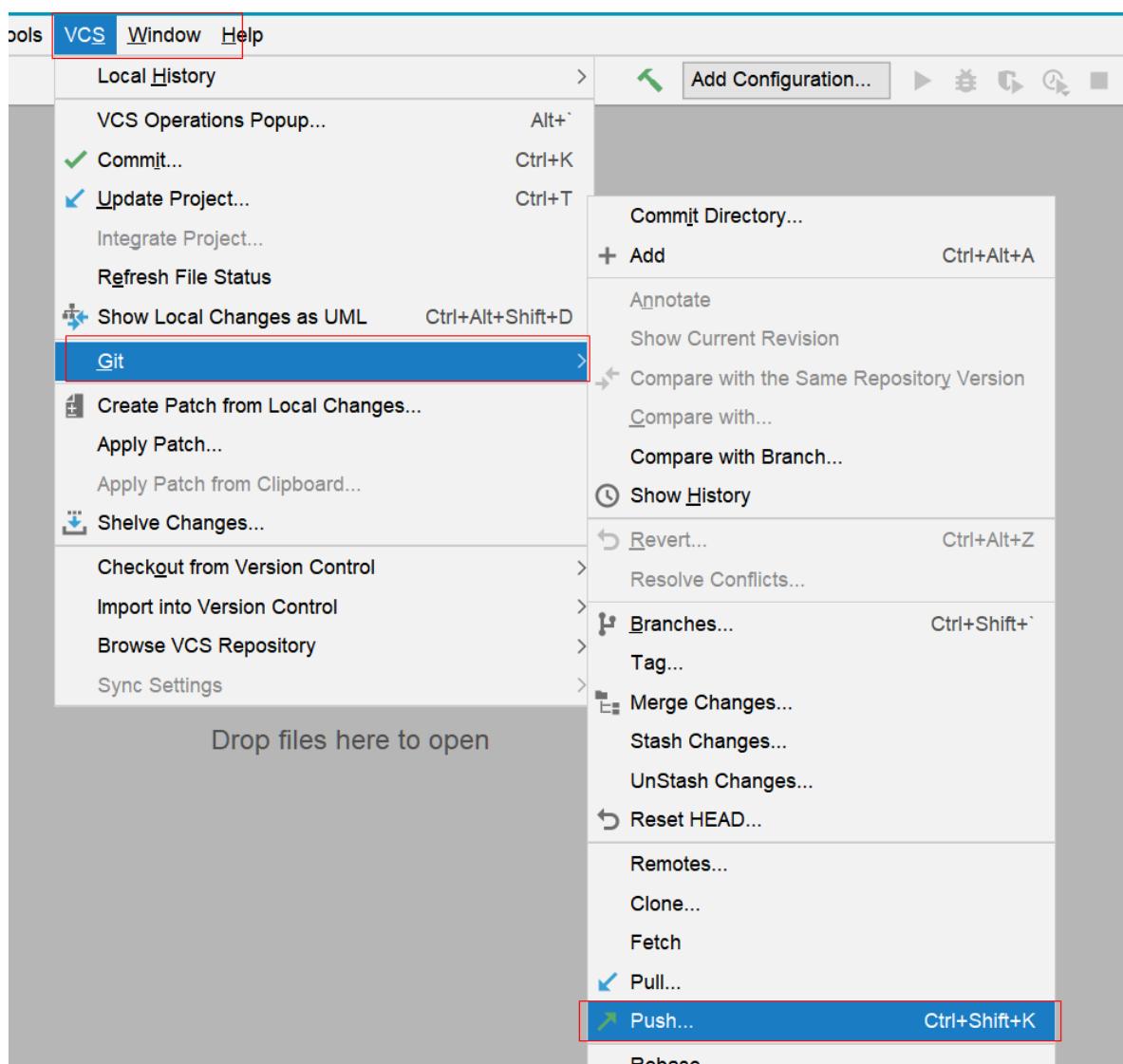
执行提交操作



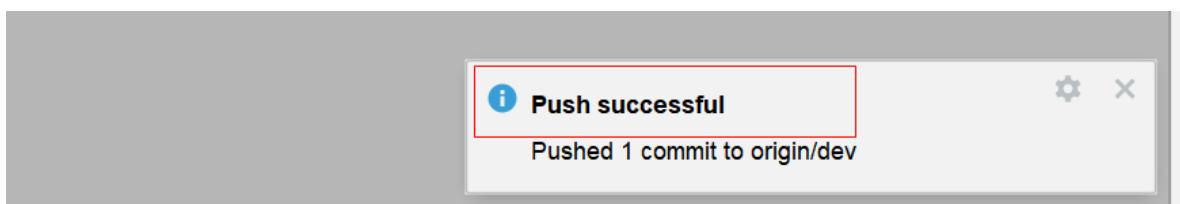
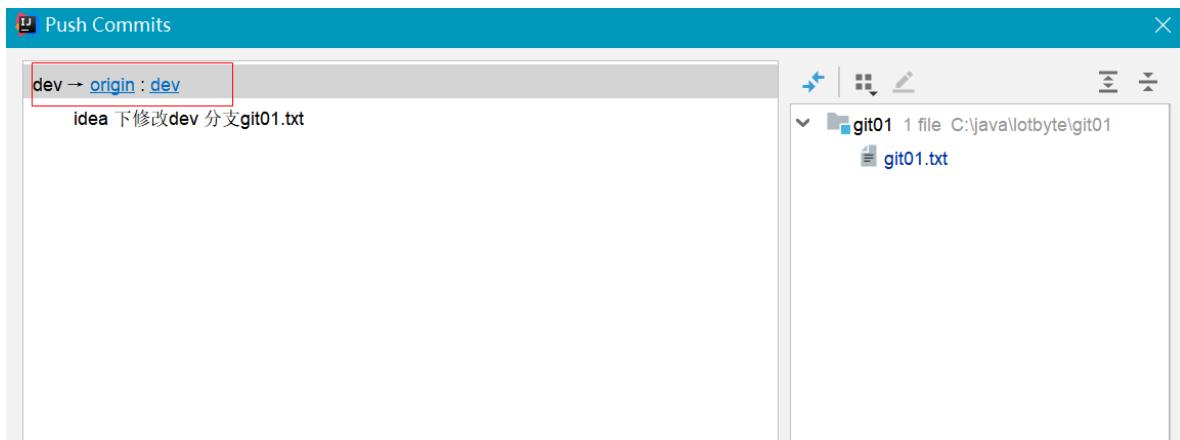
执行远程推送操作



或者



提交分支确认



远程查看

A screenshot of a GitHub repository page for "zsyue / git01". The "Code" tab is selected. A commit titled "idea 下修改dev 分支git01.txt" is shown, pushed on Jan 31, 2020. The commit message content is displayed, with line 7, "idea下文件修改与推送", highlighted with a red box.

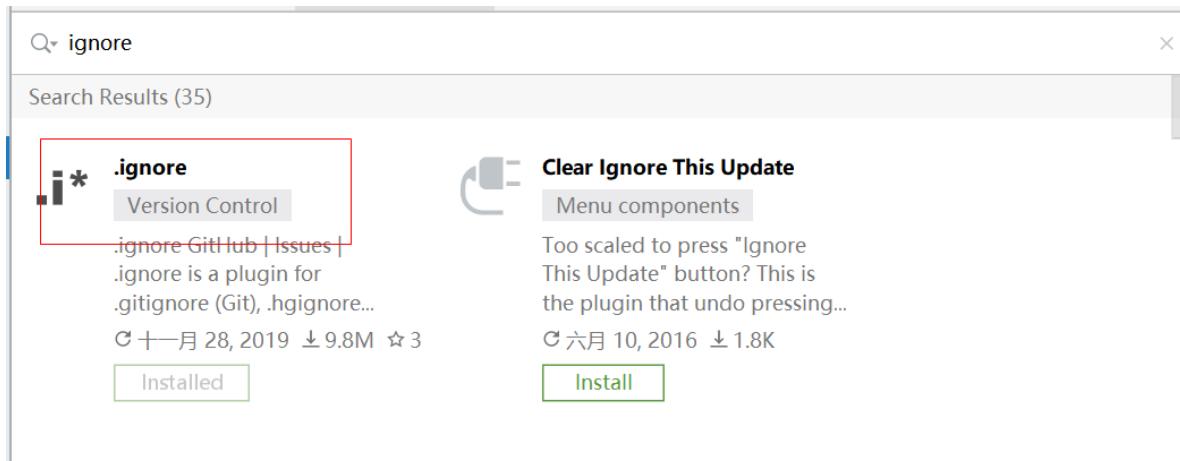
10.4. ignore插件集成

前面讲到dev 分支拉取与文件修改推送操作，遇到一个小的问题-在修改完git01.txt 文件后添加文件到暂存区后会将idea 本地环境相关配置文件一并提交，而这些文件没有必要提交 如果提交对于另外一方更新到本地后反而会影响项目运行 前面解决的方式是借助命令方式将暂存区文件转移到工作区来解决。

命令方式虽然能够解决，但操作麻烦，每次提交不可避免会遇到同样问题，这里介绍一个插件-gitignore,使用gitignore 插件可以在执行文件添加前将没有必要提交的文件让git给自动忽略掉这样开发中就只需要关注修改的文件即可，想想是不是很美！

10.4.1. 1) 安装插件ignore

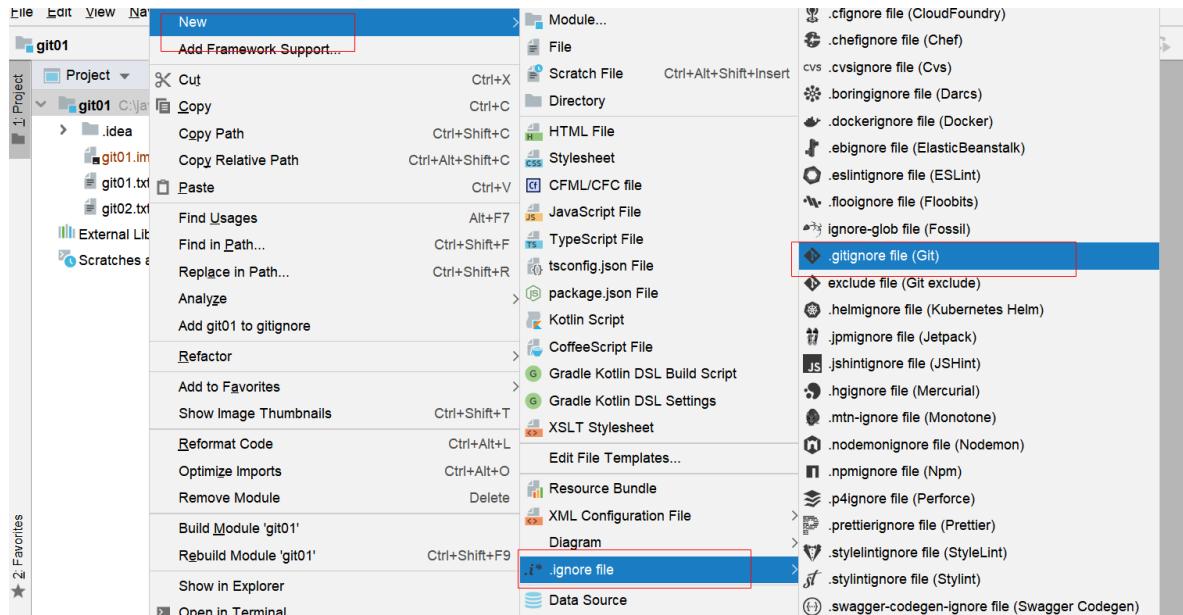
settings->plugins->Marketplace 搜索ignore插件 如果之前没有使用过该插件 点击install按钮，idea会自动下载并执行安装操作，当插件安装成功后重启idea即可。



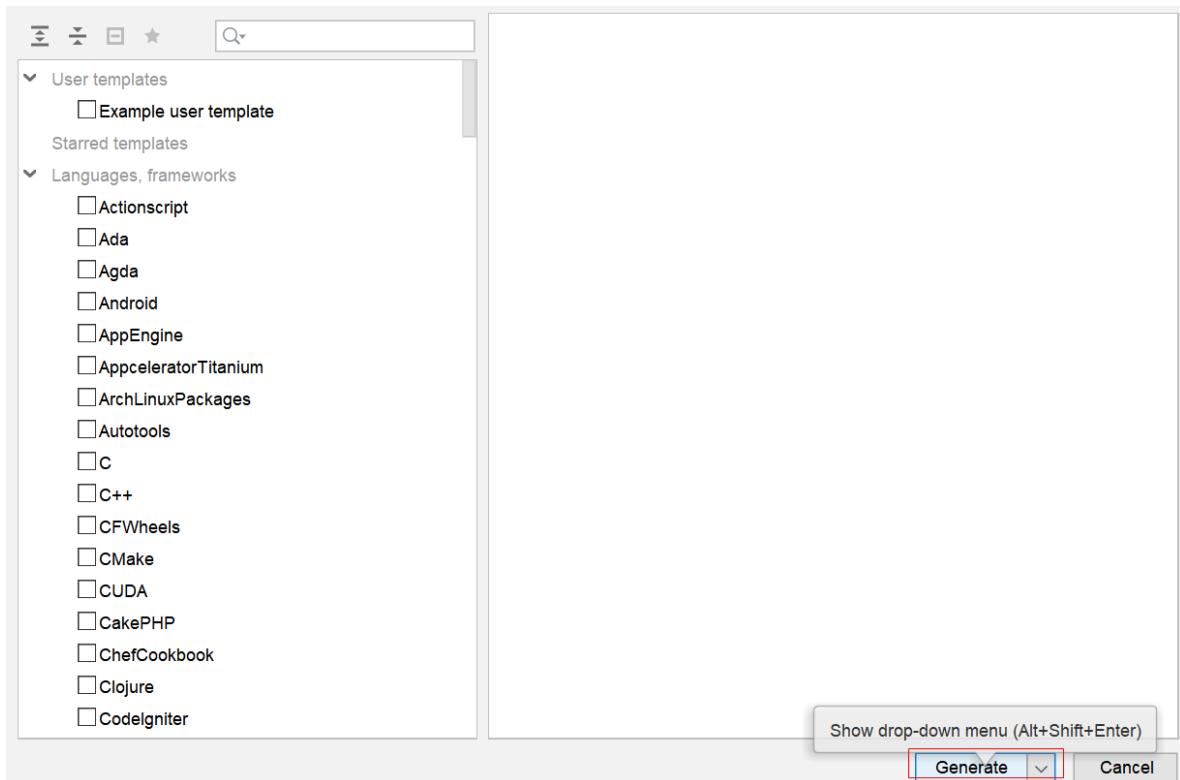
10.4.2. 2) .ignore忽略文件使用

插件安装成功重启idea后，ignore插件即生效，接下来就可以使用插件来忽略待添加的相关文件。

10.4.2.1. 新建.ignore忽略文件



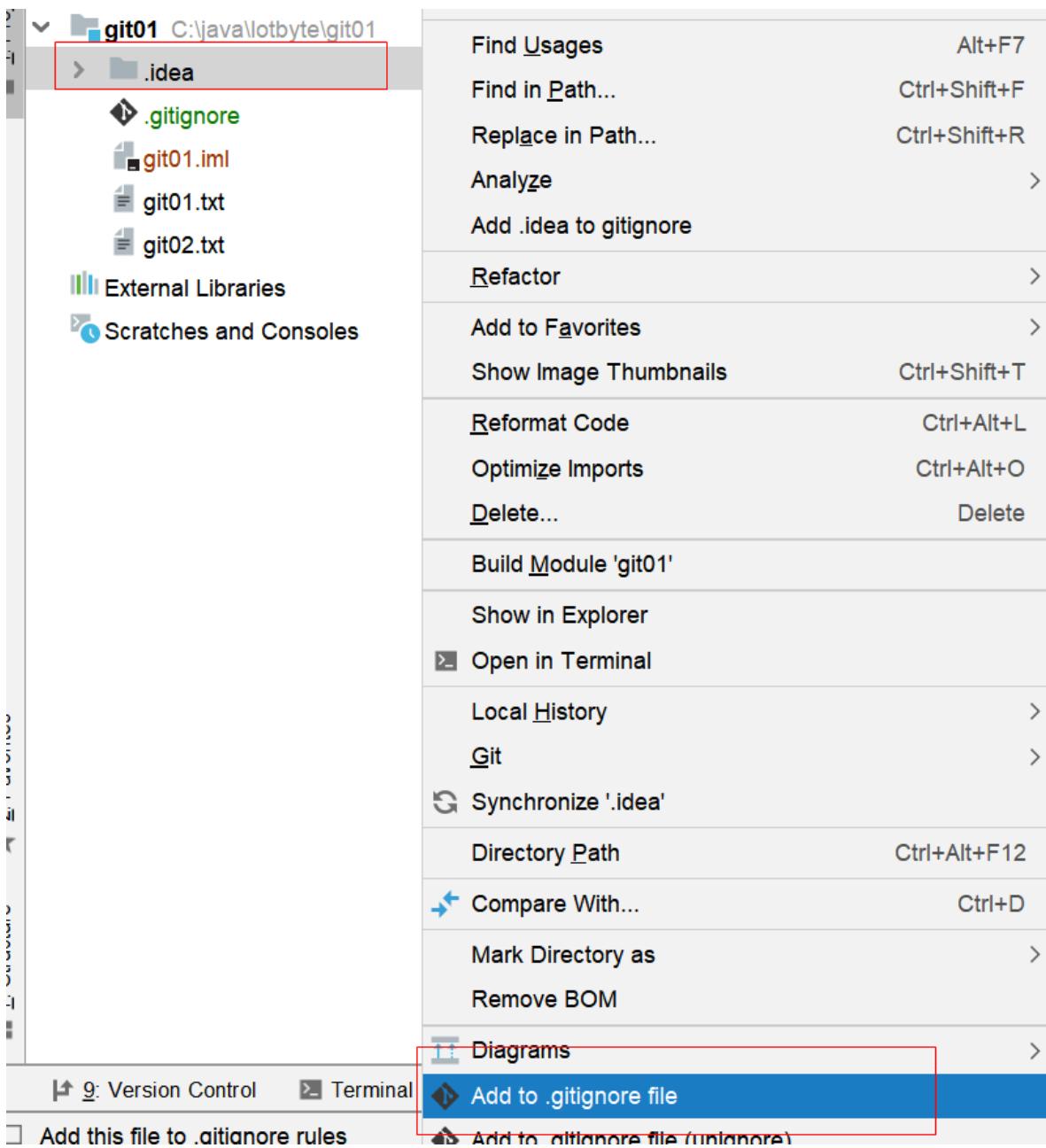
点击generate 新建文件即可



文件创建成功后会放在项目根目录下 这里将新创建的.ignore文件加入暂存区(会有相应提示 点击 add to git 即可)

10.4.2.2. 配置忽略文件

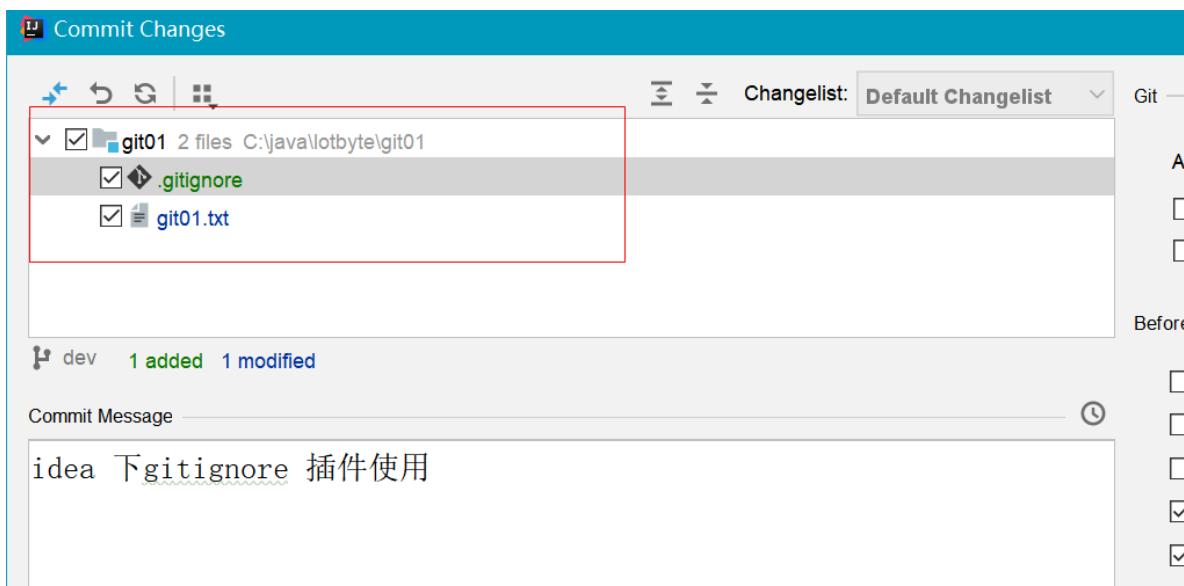
.ignore 忽略文件创建成功后，即可配置待忽略文件，配置方式相当简单 在待忽略的文件上右键添加到忽略文件即可 如下：



自动产生的文本内容如下:

1	# Created by .ignore support plugin (hsz.mobi)
2	/ .idea/
3	/git01.iml

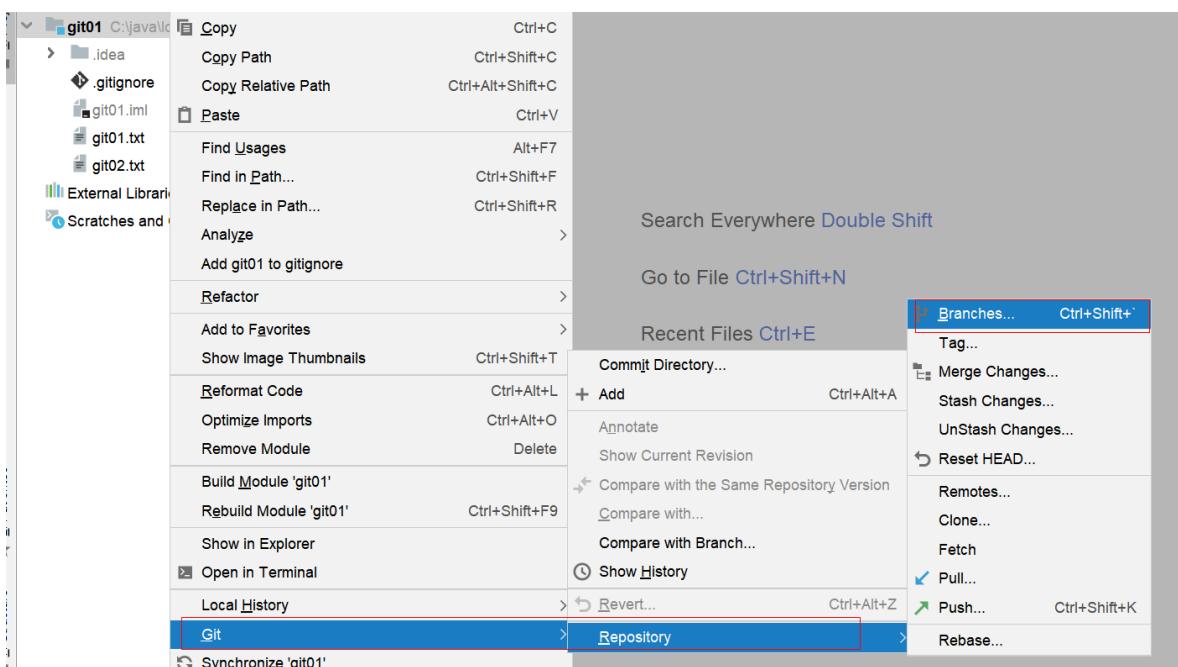
修改git01.txt文件后执行添加并提交 此时未受版本控制的文件列表已消失 执行提交即可(也可以提交同时进行推送一步到位!)。



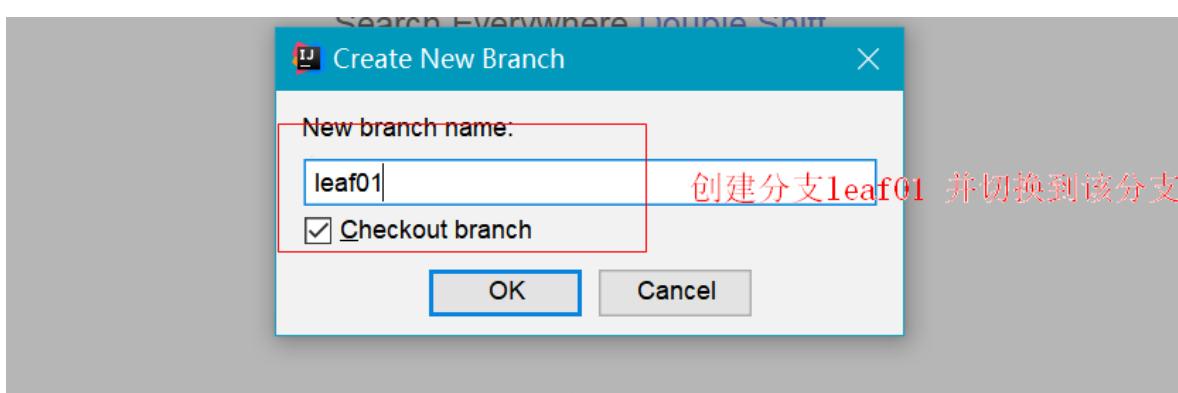
10.5. 分支操作与冲突处理

idea 下对于分支的操作使用起来相比命令操作是比较简单的，开发环境下idea分支操作比较常见

10.5.1. 本地分支创建与切换



设置本地分支名称leaf01



idea下分支切换比较简单 Git->Repository->Branches 选择指定分支 执行checkout即可。

10.5.2. 冲突出现与解决

多用户操作同一文件同一行内容提交冲突

多用户同时修改某个文件同一行内容，并能够提交，此时便会出现冲突，对同一版本文件不能出现不同内容的文件

文件提交前状态模拟

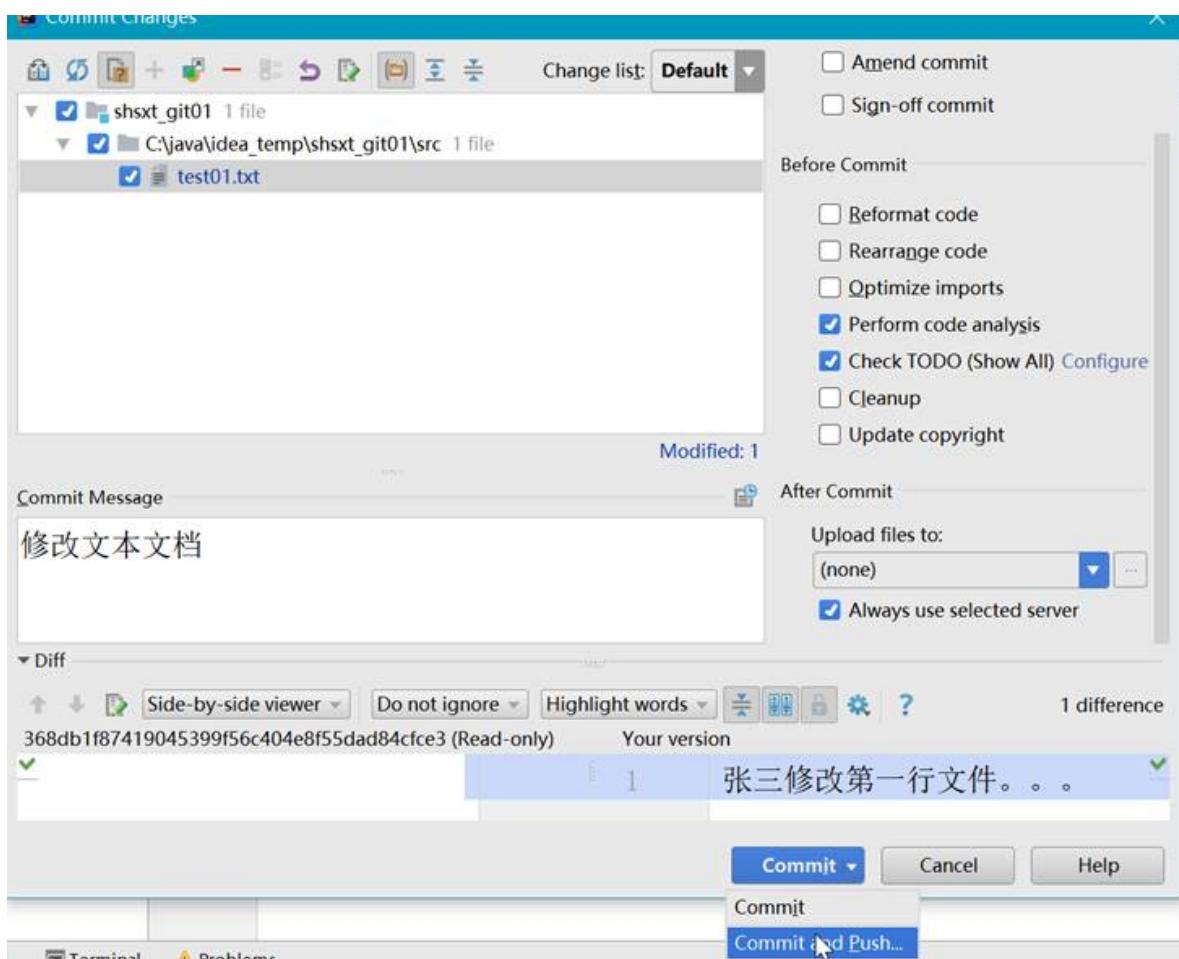
用户:张三



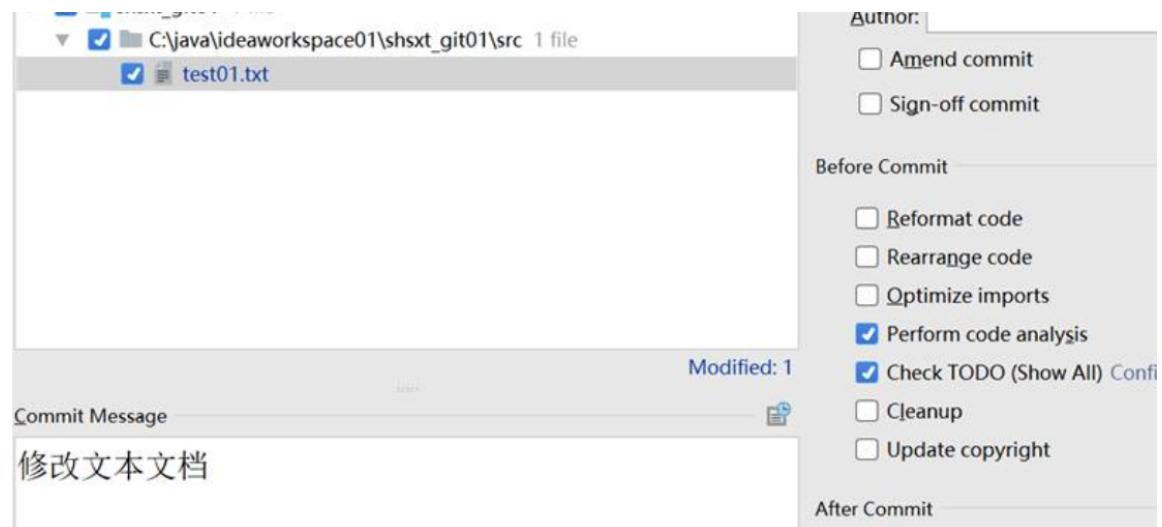
用户:王五



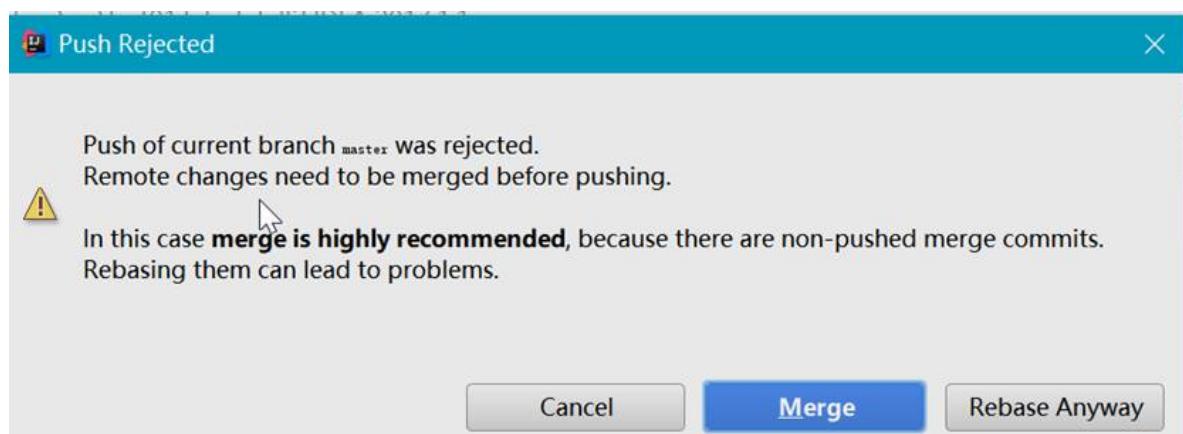
此时 假如张三抢先一步进行提交操作



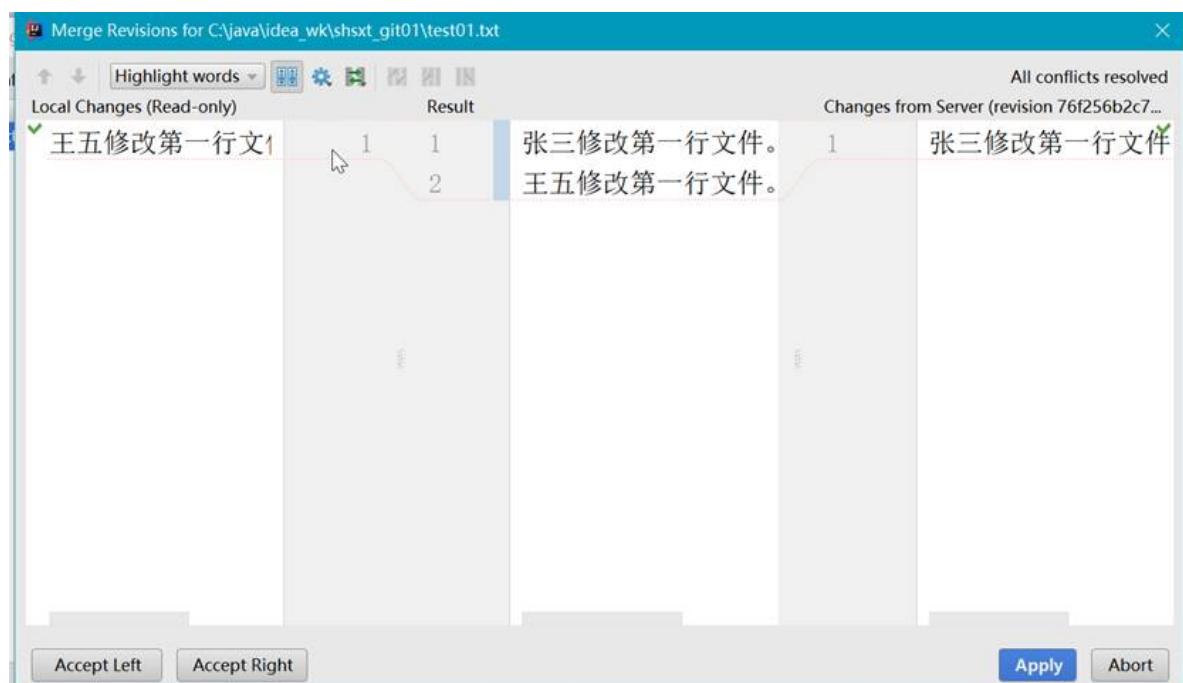
此时，对于张三来说，本地提交并推送没有问题，回到王五这端，此时假如王五在不知道张三对同一文件进行修改的情况下也进行了提交操作，此时，王五这端提交结果如下：



此时，在对文件进行提交时，系统提示建议先进行合并操作，意思就是将服务器端最新内容先合并到本地后再进行提交操作。



执行merge合并操作



保留两者文件内容 执行合并

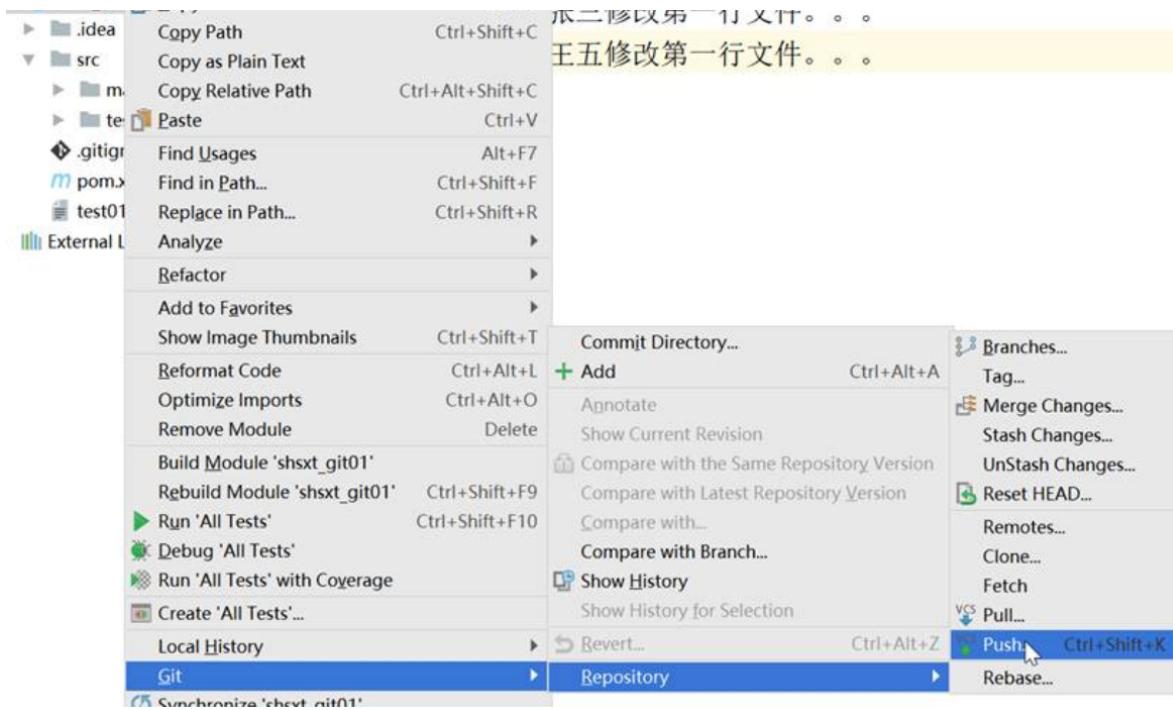
test01.txt

```
1 张三修改第一行文件。。
2 王五修改第一行文件。。
|
```

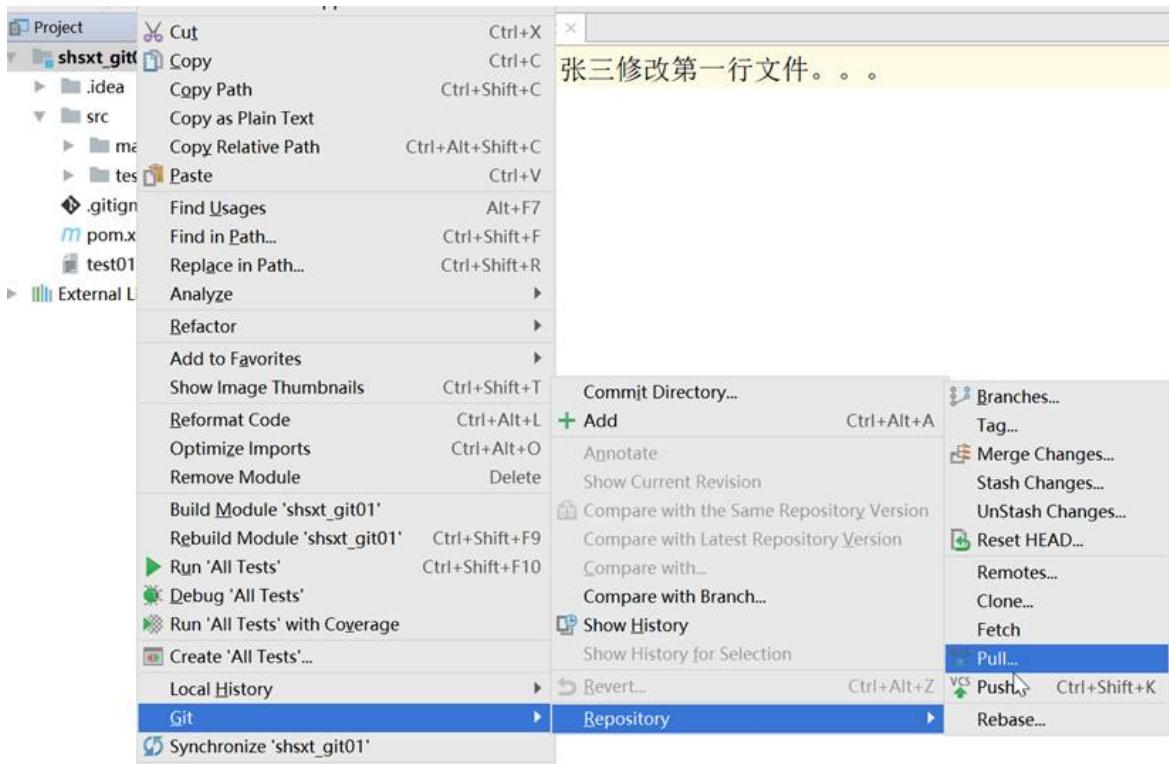
S 中 ⌂ ⌂ ⌂ ⌂

Push rejected
Push has been cancelled, because there were
conflicts during update....

最后提交 王五这端内容。



张三 这端执行拉取操作



分支操作内容冲突

在使用git对文件进行版本控制时，在不同分支同一文件同一行进行修改的情况下，此时对分支文件进行合并时，同样会有出现冲突的可能。

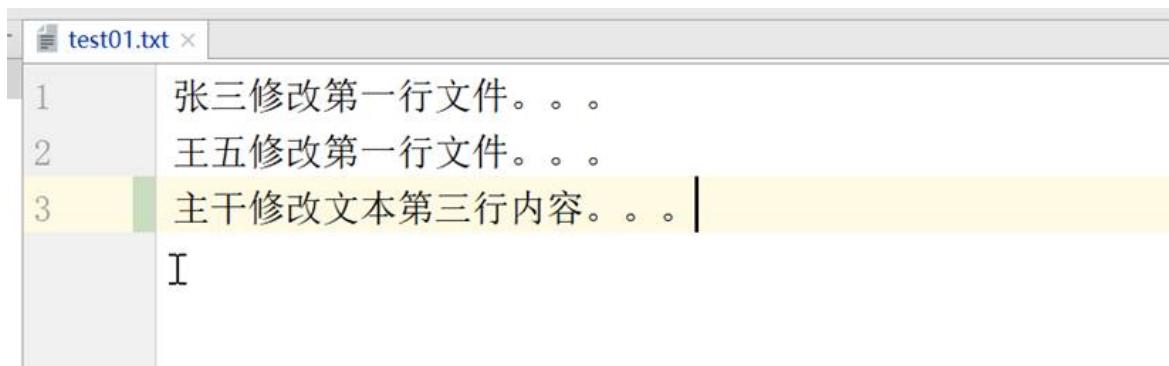
在master下创建新的分支leaf01



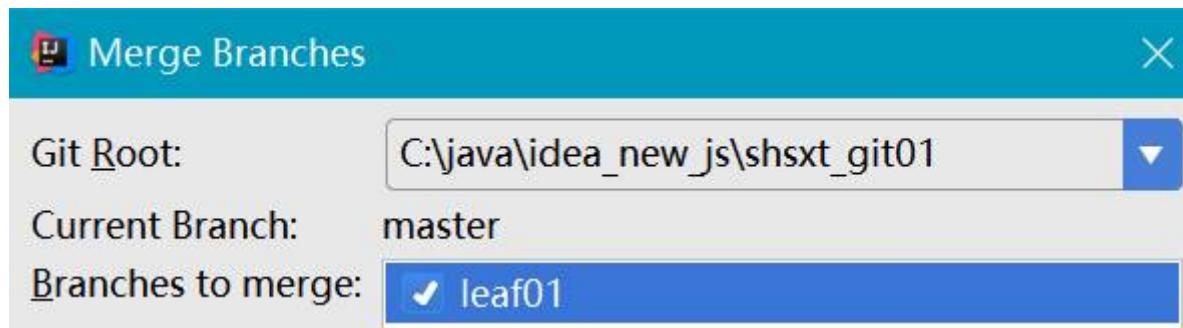
分支下修改文本第三行内容：

此时在分支leaf01上执行提交与push操作此时是没有问题的(假如其它用户没有在当前分支对应文本同一行修改与提交操作)

回到主干上修改主干文本第三行内容



主干上执行提交与push 操作此时主干提交没有问题(假如其它用户没有在当前主干对应文本同一行修改与提交操作) 然后在主干上执行合并操作 选择待合并的leaf01 分支:

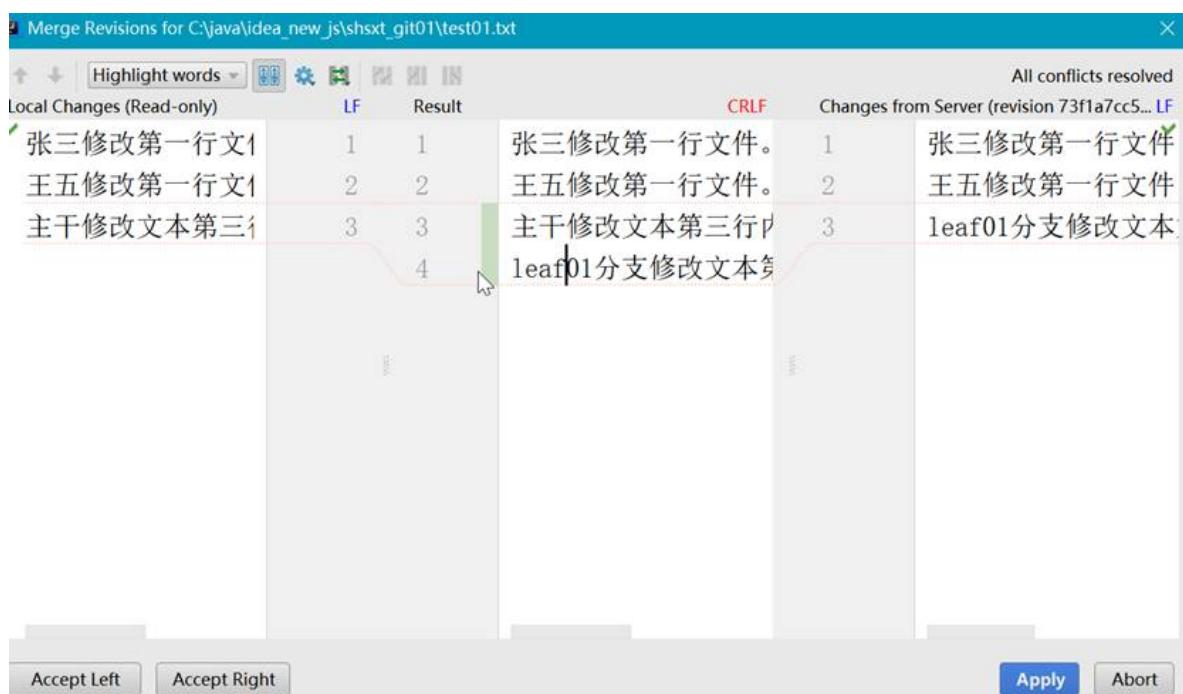


此时出现冲突场景

```
test01.txt
```

1	张三修改第一行文件。。。
2	王五修改第一行文件。。。
3	<<<<< HEAD
4	主干修改文本第三行内容。。。
5	=====
6	leaf01分支修改文本第三行内容。。。
7	>>>>> leaf01

冲突之所以出现的原因在于:主干与分支在统一位置同时进行了修改操作, 此时执行合并操作是不被允许的 因为git 此时再合并时不能区分是执行替换操作 还是执行追加新内容到主干此时需要用户来进行冲突解决(合并分支内容到主干或者进行替换处理)这里执行追加处理(注意:在企业中具体场景具体分析 添加还是做替换操作)



追加完毕后主干上执行提交与push 操作即可 此时冲突解决。

