Git 简单入门---git在本地的入门应用

git是一种可以进行编辑、保存代码库，回溯版本库等多功能的工具，属于分布式版本控制系统。

github和gitlab等等网站都是基于git工具使用的

建议初学者按照流程**在电脑上**自己实践一遍，实践中的理解会更加深刻。

下载地址：https://git-scm.com/downloads

上述网址最终会跳转到github下载git。

部分机器可能对github访问卡顿，大部分原因是国内的dns污染比较严重，推荐切换首选dns服务器：8.8.8.8或8.8.4.4 （谷歌dns服务器）

本方法和其他方法详见 https://www.zhihu.com/question/472533878/answer/2683395248

玩过steam的话也可以使用uu加速器或steam++，自行百度。

注：安装过程中有一个默认文本编辑器选择的环节，我建议选择VSCode（但是需要你去官网下载VSCode），正常情况默认Vim编辑器。

一、git工具简介

git工具一般来说分工作区(Workspace)，暂存区(Index/Stage)、本地仓库(Repository)、远程仓库(Remote Directory) 四个部分。

工作区：当前工作目录及所属所有文件 (除隐藏文件.git) 。

暂存区：隐藏文件.git目录下文件，字面意思，相当于**进行最终提交修改版本之前**的缓冲区，待修改的文件需要先存到此处，再提交到版本库。

本地仓库：即.git文件。

远程仓库：如github,gitlab,gitee等远程托管平台。

在git中，文件分为四个属性：

Untracked(未追踪)：工作区中并没有进行提交，也没有暂存的新增文件。

Modified(已修改)：**已经提交的文件中**内容改变的文件。

UnModified(未修改)：**已经提交的文件中**内容未改变的文件。

Staged(已暂存)：已经提交到暂存区中的Modified或Untracked文件。

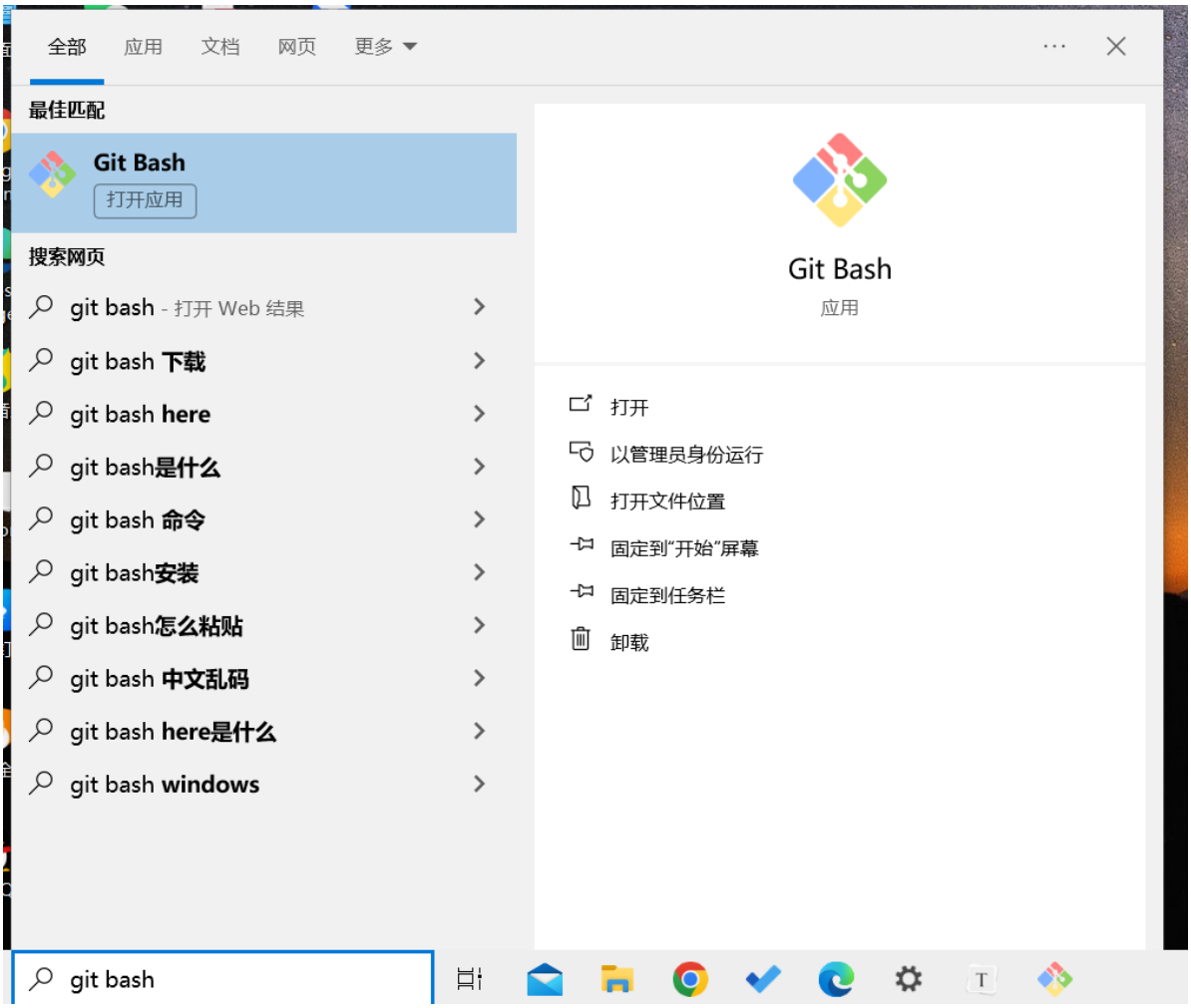
现在看不懂没关系

以下我会用实际例子来介绍每个命令和操作，在看完所有操作理解git控制逻辑后回头看就理解了。

安装完成后

二、设置账户

操作git工具使用git bash程序。



使用 git config 命令。 --global 表示对所有库。我们需要给予版本库一个账户名和email地址

```
YXY@DESKTOP-PSGQT5E MINGW64 ~
$ git config --global user.name "YXY"

YXY@DESKTOP-PSGQT5E MINGW64 ~
$ git config --global user.email "211238146@qq.com"
```

三、创建版本库

1. 选择创建目录

git bash 中 cd 命令和 cmd 中一致。

(cd 命令：同磁盘下：cd + D:\a\b\c 让命令行切换到上述目录，异磁盘 cd + E: 切换至磁盘E，默认目录在磁盘C)

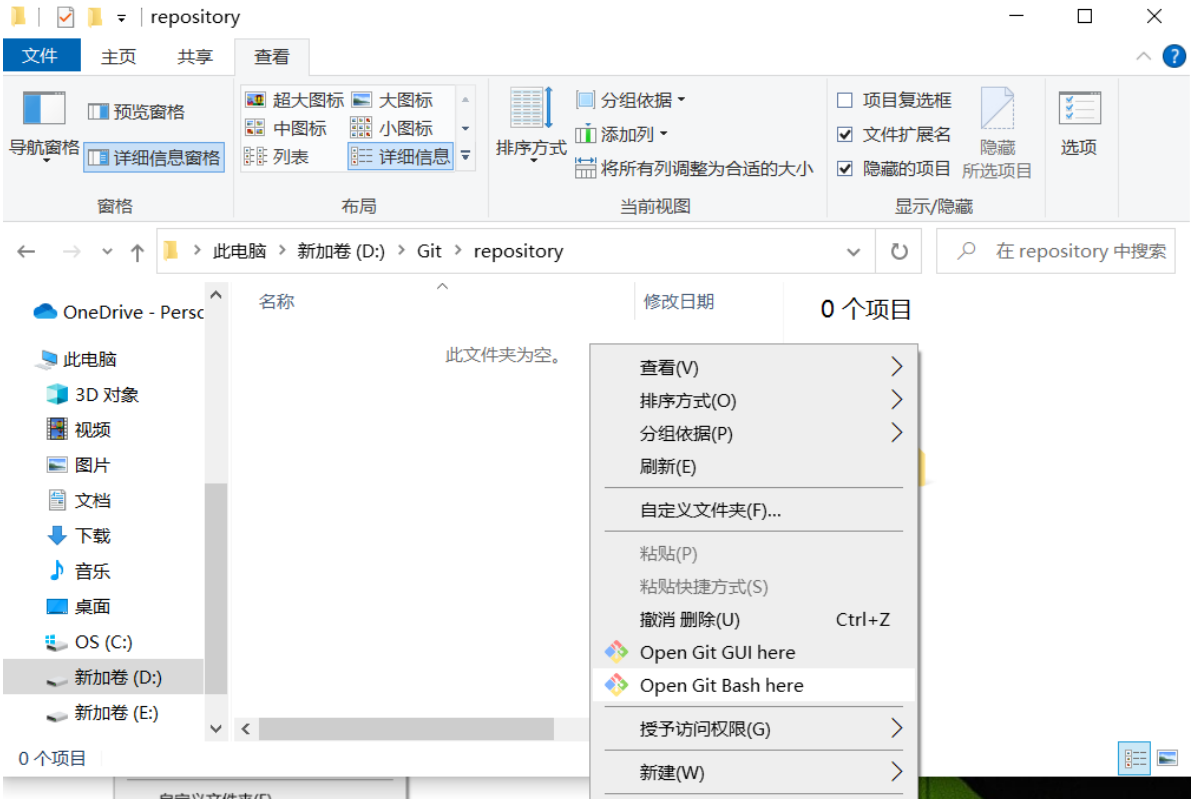
使用 cd 命令切换到要创建版本库的地址

```
YXY@DESKTOP-PSGQT5E MINGW64 ~
$ cd D:

YXY@DESKTOP-PSGQT5E MINGW64 /d
$ cd Git/repository

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ pwd
/d/Git/repository
```

可以使用pwd命令查看当前所在目录位置。

或者

在目的位置右击 -> 选择 Open Git Bash here

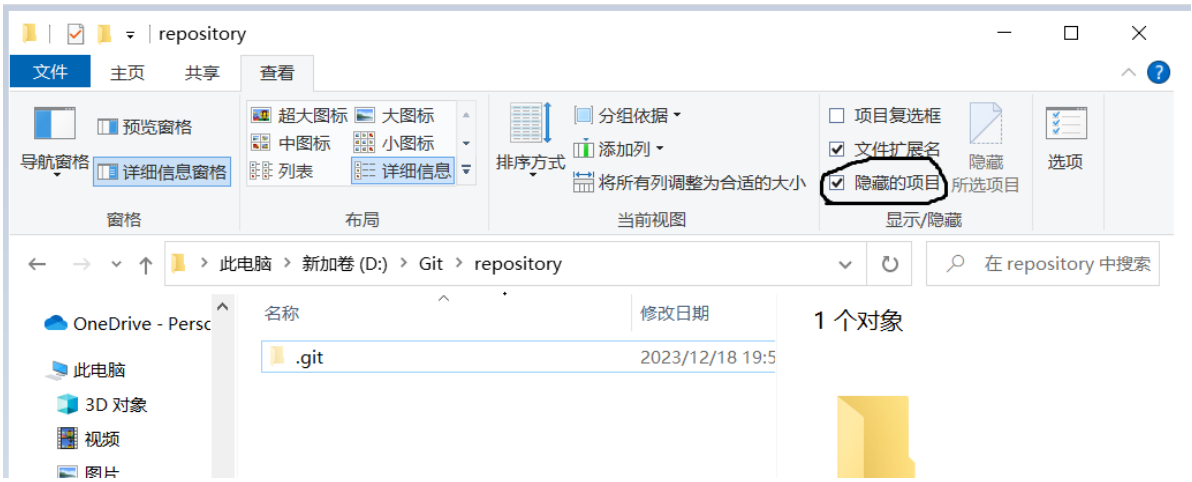


2.创建版本库

使用 git init 命令创建版本库。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository
$ git init
Initialized empty Git repository in D:/Git/repository/.git/
```

此时目录下产生新文件 .git （打开查看->显示隐藏项目），**这个文件包含git控制的所有逻辑，所以不要修改这个文件**



至此版本库创建成功。

四、修改/上传版本库

在工作区直接修改文件即可，比如我从IDEA上新建New Project名为Hi. （Hi 此时为 **Untracked** 状态）

使用git status命令查看当前已暂存和已修改的文件

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Hi/

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

使用git add 命令将Hi文件加入暂存区，第一次输入可能有warning，这是因为new project中回车有部分为linux下回车，但是存入git暂存区和库中需要统一为windows下换行符CRLF，再输入一次即可。没有反应表示成功。

另一个常用语句

`git add .` 表示将所有文件加入暂存区。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add Hi
warning: in the working copy of 'Hi/src/Main.java', LF will be replaced by CRLF the next time Git touches it

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add Hi
```

再次输入git status，发现Hi这个项目已经成功存入暂存区 （Hi 此时为 **Staged** 状态）

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Hi/.idea/.gitignore
        new file:   Hi/.idea/misc.xml
        new file:   Hi/.idea/modules.xml
        new file:   Hi/.idea/vcs.xml
        new file:   Hi/Hi.iml
        new file:   Hi/src/Main.java
```

输入 git commit -m "这里是备注，随便写一些什么，不可以不写。"

（只输入 `git commit` 并回车会打开git安装默认文本编辑器，此时可以输入多行备注。）

注意：如果你在安装的时候默认一路绿灯，那么你的默认文本编辑器是vim，vim文本编辑器很强大，但是它的命令非常复杂，退出vim编辑状态的步骤如下：按Esc，此时你的光标会出现在vim编辑框下方，输入:wq回车来退出（冒号也要输入）。这表示保存并退出的意思。
其他命令：
:q!表示强制退出，不保存最新更改。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git commit -m "Test"
[master 2523000] Test
 6 files changed, 39 insertions(+)
 create mode 100644 Hi/.idea/.gitignore
 create mode 100644 Hi/.idea/misc.xml
 create mode 100644 Hi/.idea/modules.xml
 create mode 100644 Hi/.idea/vcs.xml
 create mode 100644 Hi/Hi.iml
 create mode 100644 Hi/src/Main.java
```

这样我们实现了将Hi文件上传到版本库的操作。

版本库的修改和上传操作类似。

五、回溯版本库

git中的版本库按照树状逻辑存储，同时存在一个HEAD指针，指向目前工作区所对应的版本。

值得注意的是，已修改、暂存但未提交修改的文件不会被版本回溯影响。

我们用举例说明。

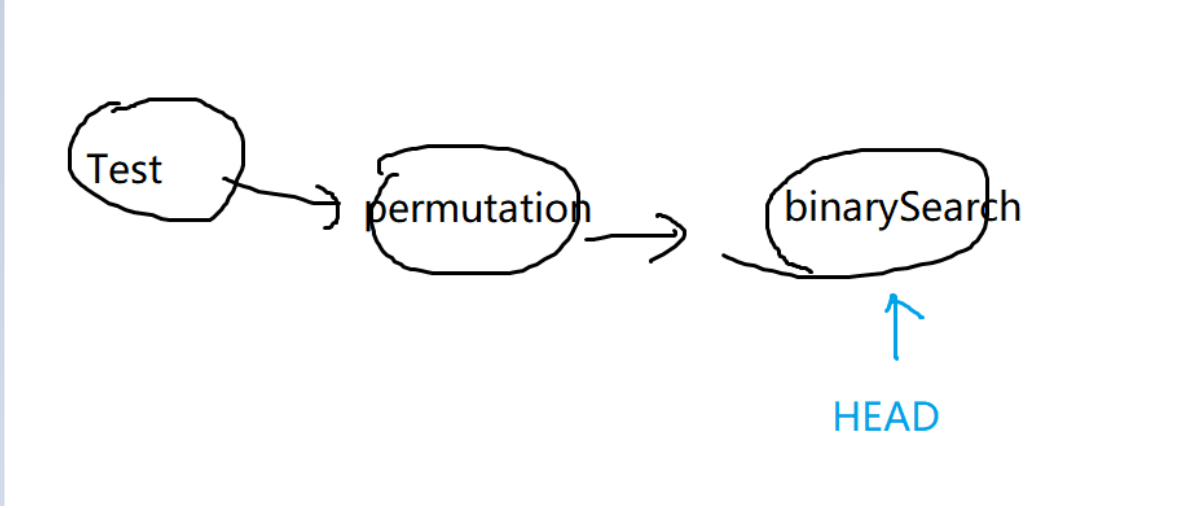
我先新建了一个permutaition.java并提交。

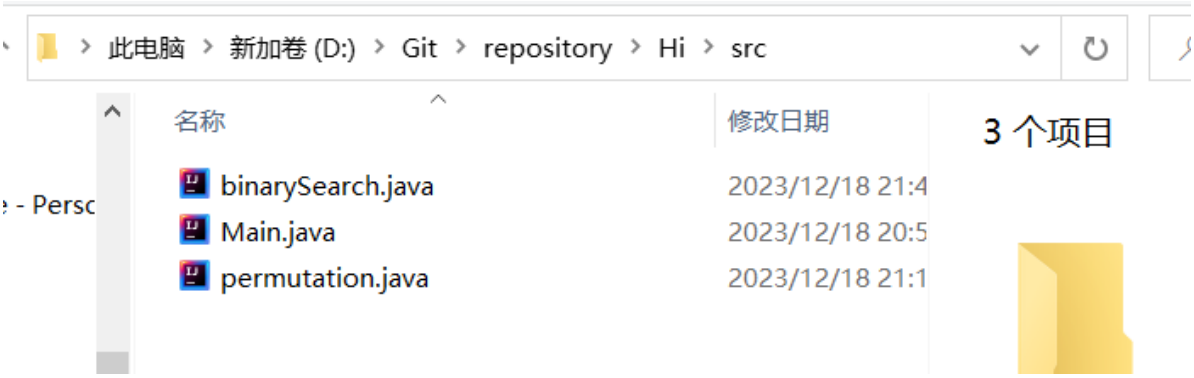
```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Hi/out/
        Hi/src/permutation.java

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

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add .

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git commit -m "permutation"
[master 7b73adc] permutation
 3 files changed, 31 insertions(+)
 create mode 100644 Hi/out/production/Hi/Main.class
 create mode 100644 Hi/out/production/Hi/permutation.class
 create mode 100644 Hi/src/permutation.java
```

我又新建了一个binarySearch.java并提交。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Hi/out/production/Hi/binarySearch.class
        Hi/src/binarySearch.java

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

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add .

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git commit -m "binarySearch"
[master 60c74d9] binarySearch
 2 files changed, 27 insertions(+)
 create mode 100644 Hi/out/production/Hi/binarySearch.class
 create mode 100644 Hi/src/binarySearch.java
```

此时**版本树形图**和**工作区src文件夹**内如下：

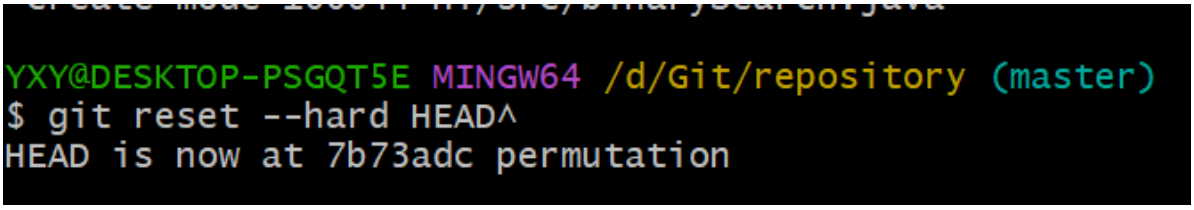


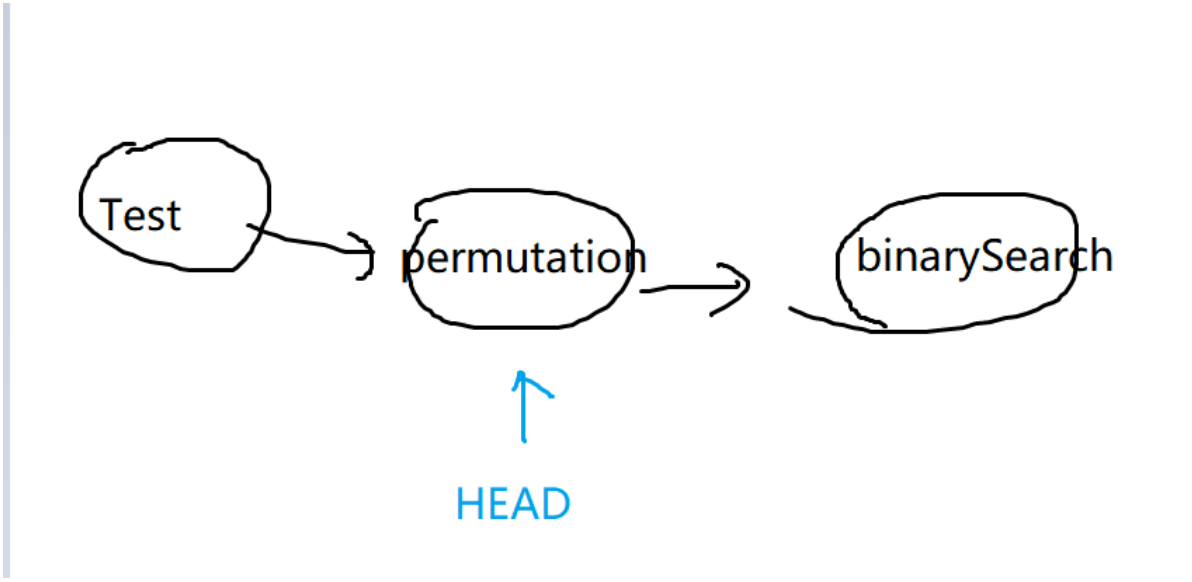


输入 git reset --hard HEAD^

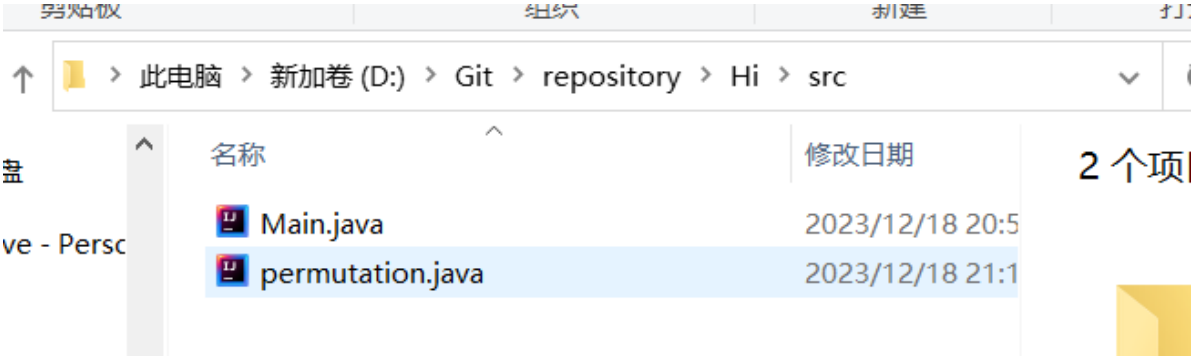
此时HEAD回退一位，回到上一版本

```
git reset --hard HEAD^^ 向上回两个版本
git reset --hard HEAD^^^ 向上回三个。几个箭头回几个版本。
git reset --hard HEAD~100 向上回100个版本。
```





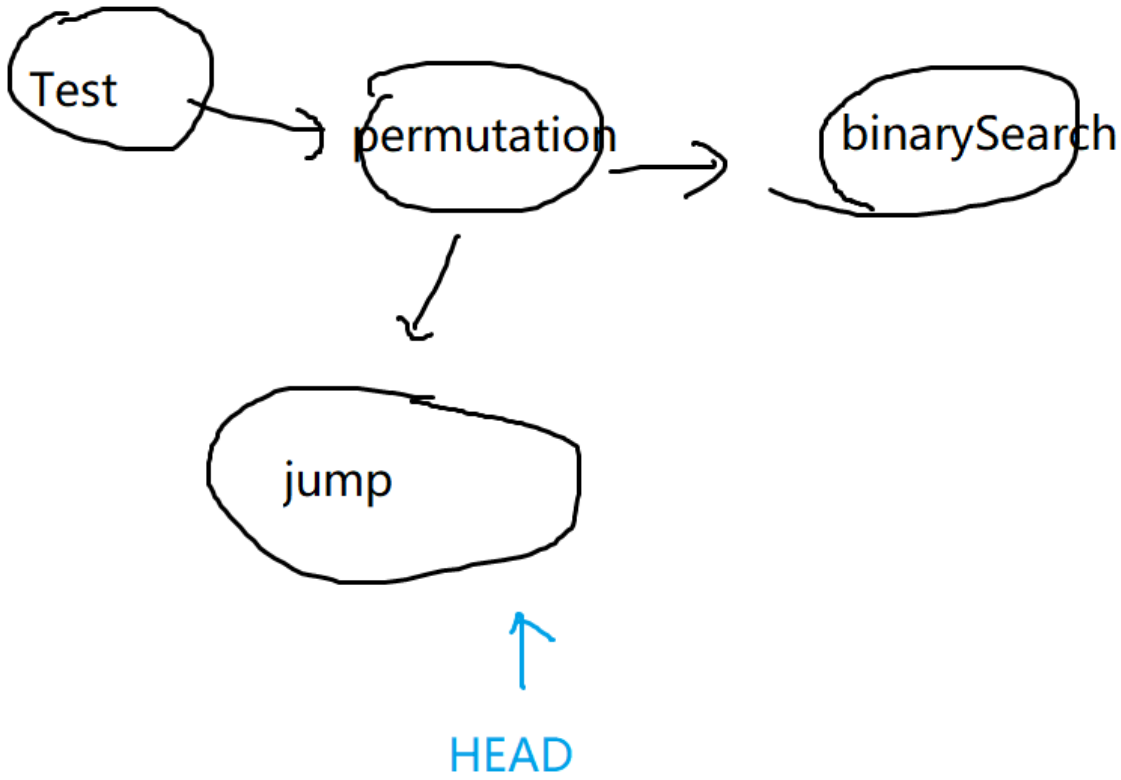
进入工作区，我们发现Hi这个项目下只有Main和permutation这两个java文件了。

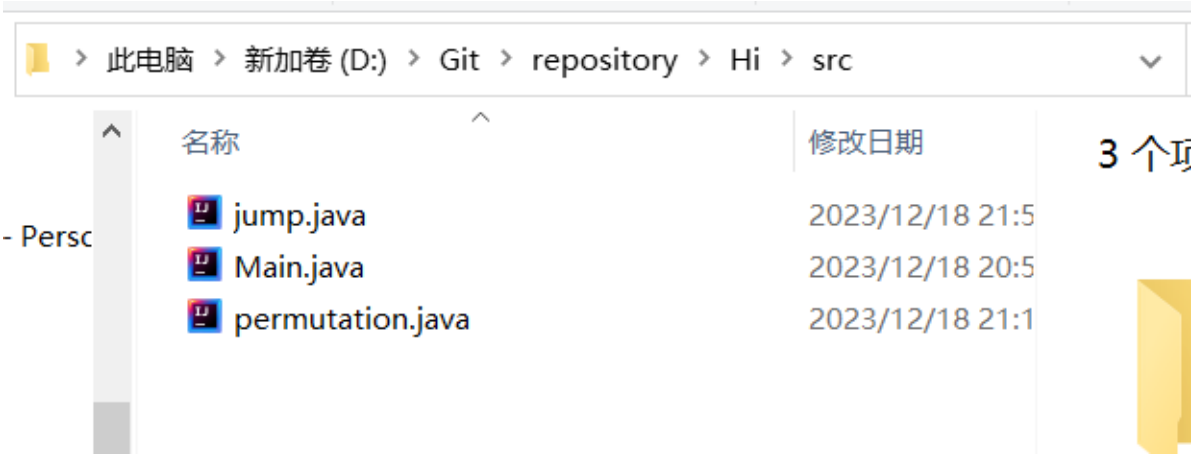


我们再次新建jump.java并上传提交到git版本库

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Hi/out/production/Hi/jump.class
        Hi/src/jump.java

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

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add Hi/out/production/Hi/jump.class

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git add Hi/src/jump.java

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git commit -m "jump"
[master 999197d] jump
 2 files changed, 5 insertions(+)
 create mode 100644 Hi/out/production/Hi/jump.class
 create mode 100644 Hi/src/jump.java
```

此时**版本树形图**和**工作区src文件夹**如下：





我们使用 git reflog 命令，可以查询所有与**HEAD指针变动（版本切换）相关**的操作

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git reflog
999197d (HEAD -> master) HEAD@{0}: reset: moving to 999197d
60c74d9 HEAD@{1}: reset: moving to 60c74d9
999197d (HEAD -> master) HEAD@{2}: commit: jump
7b73adc HEAD@{3}: reset: moving to HEAD^
60c74d9 HEAD@{4}: commit: binarySearch
7b73adc HEAD@{5}: commit: permutation
2523000 HEAD@{6}: commit: Test
```

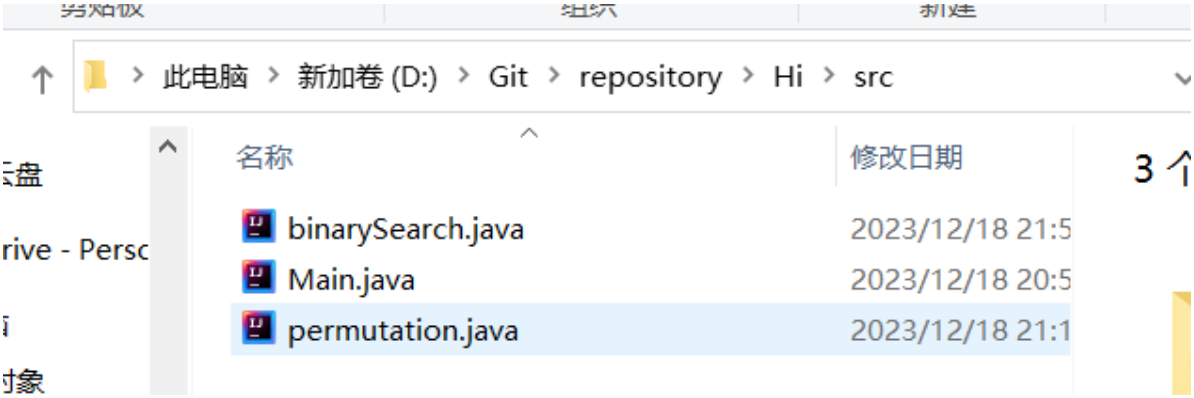
版本前面的一串编号代指这一版本。

我们使用 **git reset --hard 编号** 即可回到编号对应版本

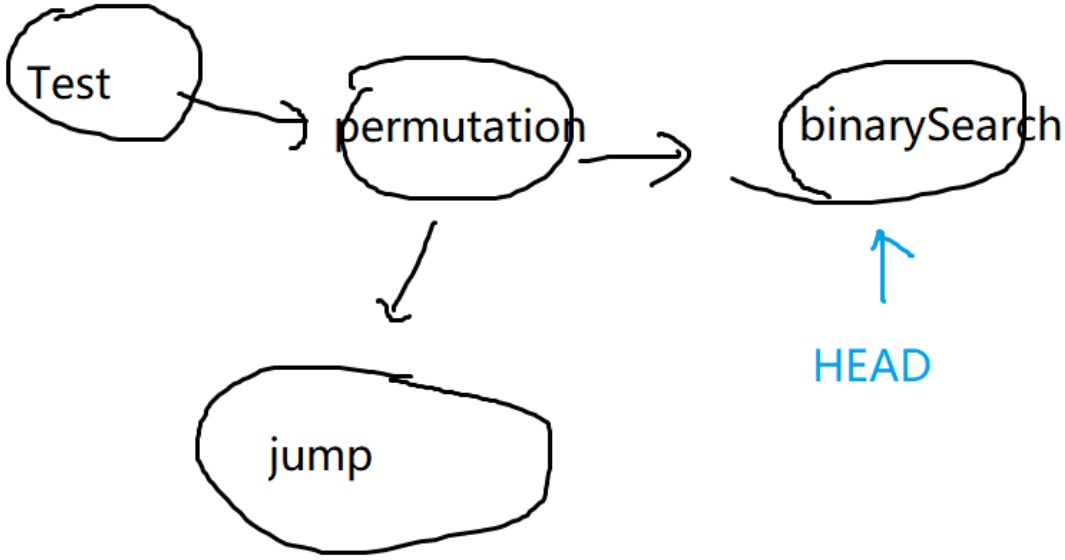
输入 git reset --hard 60c74d9

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git reset --hard 60c74d9
HEAD is now at 60c74d9 binarySearch
```

此时工作区



树状结构



以上是有关版本回溯的超简单介绍。

事实上，**由于我们只关心从头到HEAD的所有记录**，所以这种版本回溯实际上起到了删除某些commit操作的作用。

其他命令如 git log 可以展示所有从头到HEAD的commit记录，包含提交者，时间，版本备注

注意 当提交过多log一页显示不完时需要不断按空格进行显示，直到其显示(END)标识，表示显示结束，此时在英文输入下按q退出，git reflog同理。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/repository (master)
$ git log
commit 60c74d94b9100107184d46535de63ba129c41fd4 (HEAD -> master)
Author: YXY <211238146@qq.com>
Date:   Mon Dec 18 21:35:17 2023 +0800

    binarySearch

commit 7b73adc8ca0e8324d6c65edd52d2303df5a44789
Author: YXY <211238146@qq.com>
Date:   Mon Dec 18 21:23:39 2023 +0800

    permutation

commit 252300047f26445b5c79d3f9ebf8c01f04532847
Author: YXY <211238146@qq.com>
Date:   Mon Dec 18 21:12:16 2023 +0800

    Test
```

六、分支

~~(之前的库被我误删了，然后我重新写了一遍，所以commit时间可能对不上，但内容是一样的)~~

分支是git中重要的概念，分支概念基本实现了多人开发合作的功能。

如果你安装git的时候一路绿灯，那么你的默认分支（主分支）就是master分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
```

我们可以使用git branch命令查看当前库的所有分支，绿色表示当前所在分支

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git branch
* master
```

git内部存在一个HEAD指针指向当前分支。

输入git log

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git log
commit d5f866773bc29f8c588b5e096efb76f6090ff5c5 (HEAD -> master)
Author: YXY <211238146@qq.com>
Date:   Fri Dec 22 10:25:24 2023 +0800

    binarySearch

commit cdfc8ba27fd017dd509c98725a5f8bf0227aff7e
Author: YXY <211238146@qq.com>
Date:   Fri Dec 22 10:24:56 2023 +0800

    permutation

commit eb269f37d3638195bafc3cb0a82391f9b0447b62
Author: YXY <211238146@qq.com>
Date:   Fri Dec 22 10:24:20 2023 +0800

    Test
```

1.新建/删除分支

使用 git branch [branch_name] 来新建一个分支

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git branch develop_Version_1

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git branch develop_Version_2
```

git checkout -b [branch_name] 来新建一个分支并切换到该新建分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git checkout -b Chinese
Switched to a new branch 'Chinese'
```

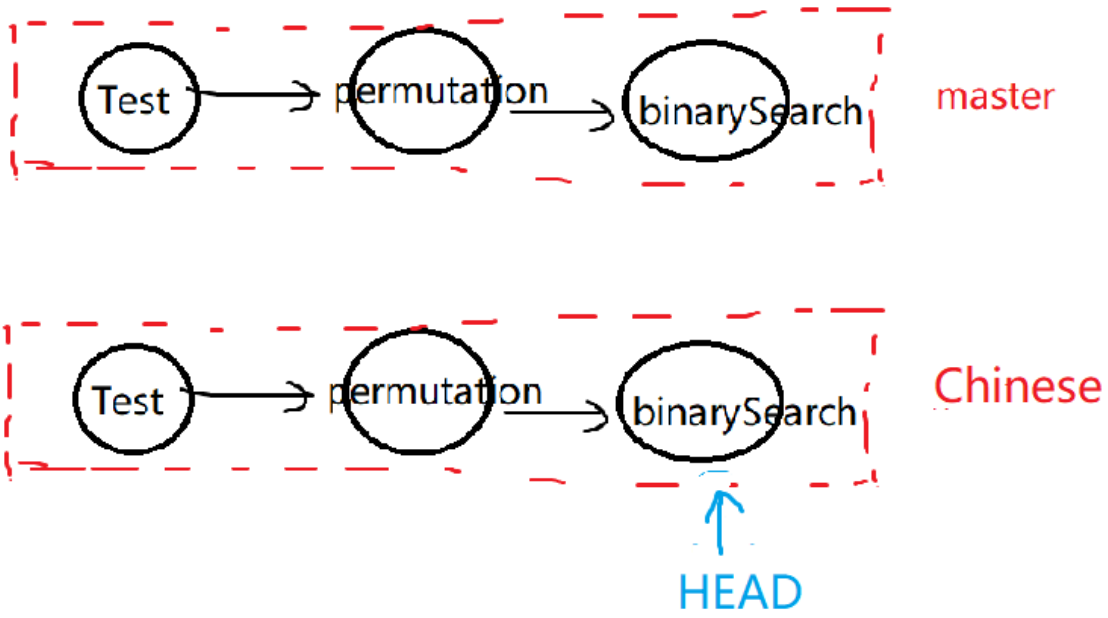
输入git branch查看分支

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git branch
* Chinese
  develop_Version_1
  develop_Version_2
  master
```

使用 git branch -d [branch_name] 来删除一个已有的分支。注意不能删除自己当前所在分支，比如这里删除Chinese分支就会报错。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git branch -d develop_Version_1
Deleted branch develop_Version_1 (was d5f8667).

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git branch -d develop_Version_2
Deleted branch develop_Version_2 (was d5f8667).

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git branch -d Chinese
error: cannot delete branch 'Chinese' used by worktree at 'D:/Git
/Repo'
```

当你新建一个分支时，git会copy原来从头到HEAD的版本到新分支中。（可以这样理解，但实际上并不是这么设计的）

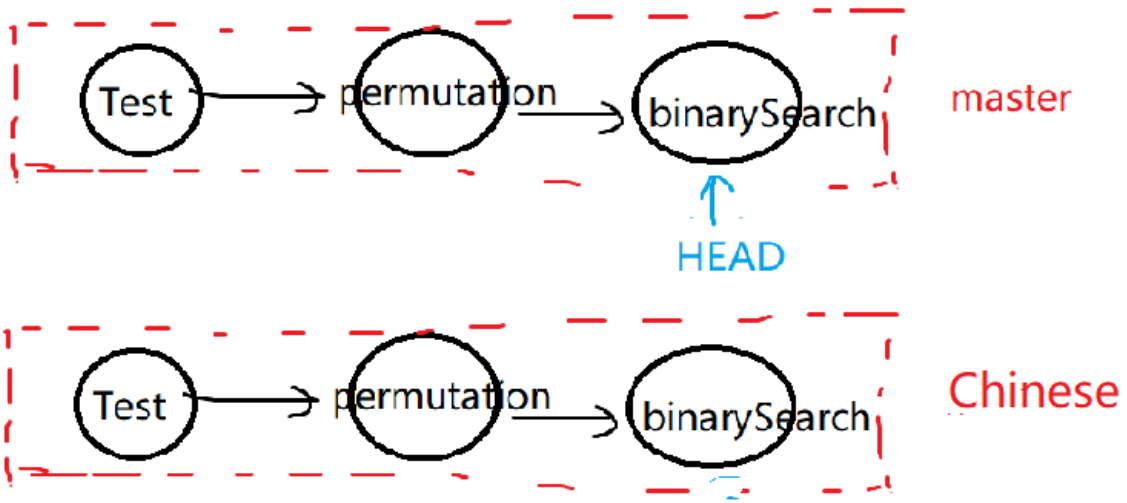


HEAD指针指向当前分支。

2.切换分支

使用 git checkout [branch_name] 来切换到已有分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git checkout master
Switched to branch 'master'

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git branch
  Chinese
* master
```



3.分支逻辑

我们新建English分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git checkout -b English
Switched to a new branch 'English'

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (English)
$ git branch
  Chinese
* English
  master
```

在工作区，我们发现文件和master分支完全一致，接下来我们新建English_fonts.txt文件并保存。

我们进行提交，由于当前处于English分支，因此这实现了English分支上的提交。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (English)
$ git status
On branch English
Untracked files:
  (use "git add <file>..." to include in what will be committ
        English_fonts.txt

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

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (English)
$ git add English_fonts.txt

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (English)
$ git commit -m "English_texts"
[English c2d046b] English_texts
 1 file changed, 1 insertion(+)
 create mode 100644 English_fonts.txt
```

切换到Chinese分支，进行类似操作

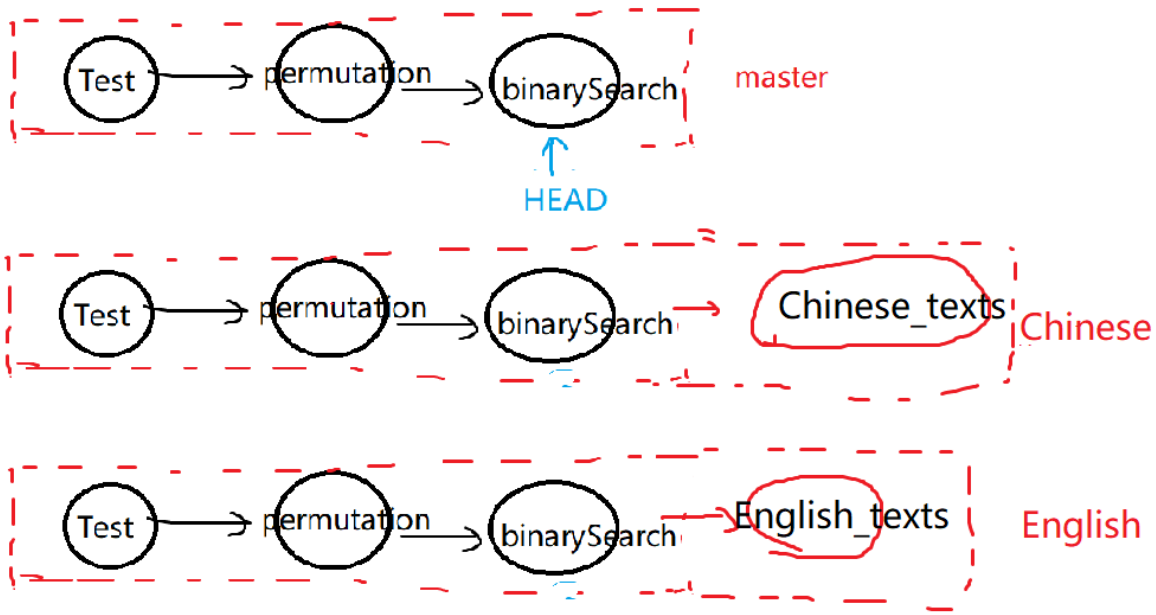
提交。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git status
On branch Chinese
Untracked files:
  (use "git add <file>..." to include in what will be
        Chinese_fonts.txt

nothing added to commit but untracked files present (u

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git add .

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git commit -m "Chinese_texts"
[Chinese cfd2047] Chinese_texts
 1 file changed, 1 insertion(+)
 create mode 100644 Chinese_fonts.txt
```

我们切换到master分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git checkout master
Switched to branch 'master'
```

此时逻辑图如下。



想象一下这些不同分支被交给不同电脑上的人，那么这实际上就实现了多人协作开发软件的不同功能，比如上面，Chinese分支可以被交给一些人来完成软件的中文版本，English分支可以被交给一些人来完成软件的英文版本。

接下来我们介绍分支的合并。将上述多人完成的不同操作进行综合整理。

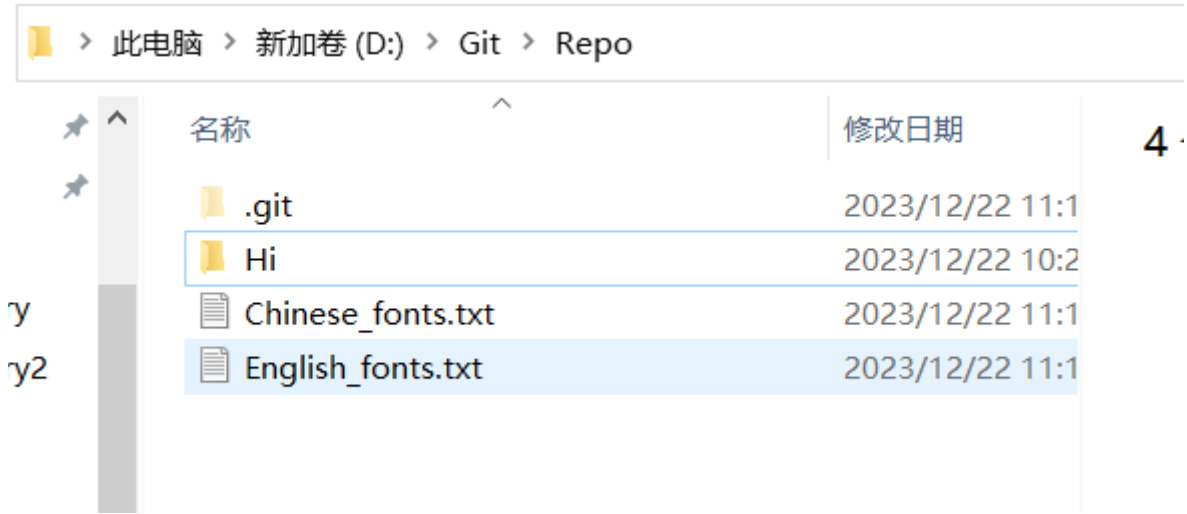
4.分支合并

我们使用git merge [Sourcebranch_name]命令实现[Sourcebranch_name]分支和当前分支的合并。

```
git merge -m [Sourcebranch_name] "添加合并信息和备注"

没有-m ""则会打开默认文本编辑器进行多行信息输入，和commit命令一样，比如下图中第二个merge命令会打开我的VSCode。
```

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git merge Chinese -m "merge_ChineseFonts"
Updating d5f8667..cfd2047
Fast-forward (no commit created; -m option ignored)
 Chinese_fonts.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Chinese_fonts.txt

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git merge English
Merge made by the 'ort' strategy.
 English_fonts.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 English_fonts.txt
```

查看工作区，ok，两个分支都合并好了。

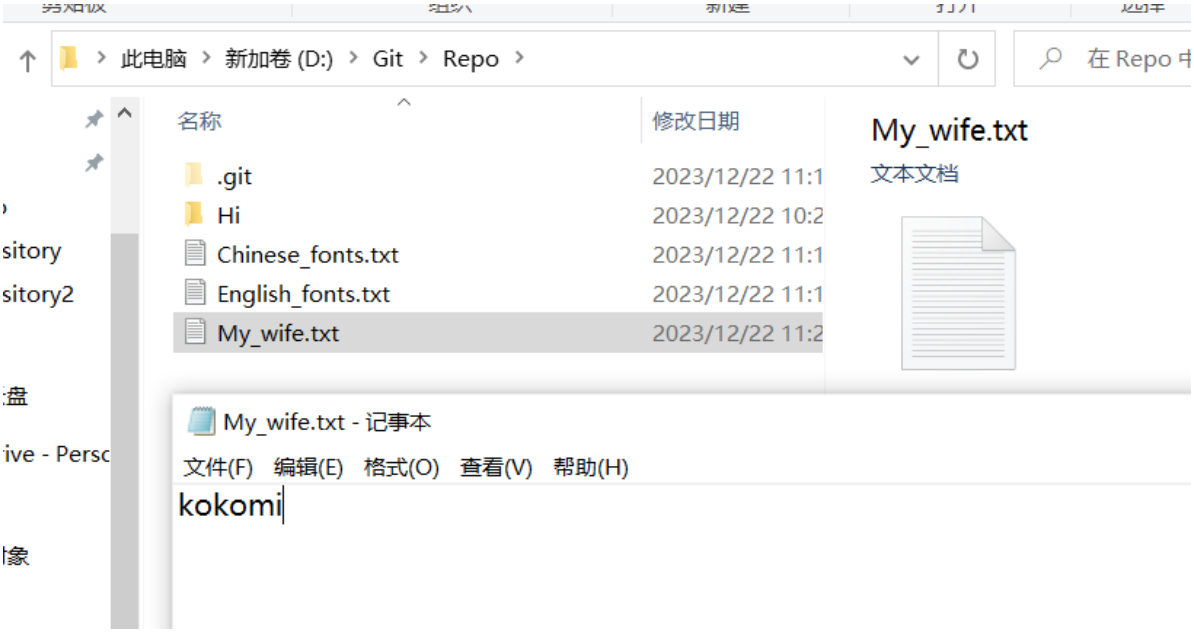


5.Conflict处理

对于目标分支的新建文件和新添加代码内容，git会自动识别并添加，不会出问题。

但是如果目标分支和当前分支修改了同一个文件的同一行，git就无法自动合并了，这个时候git会抛出CONFLICT，需要我们自行处理。

比如我们在master分支新建My_wife.txt文件，输入内容。



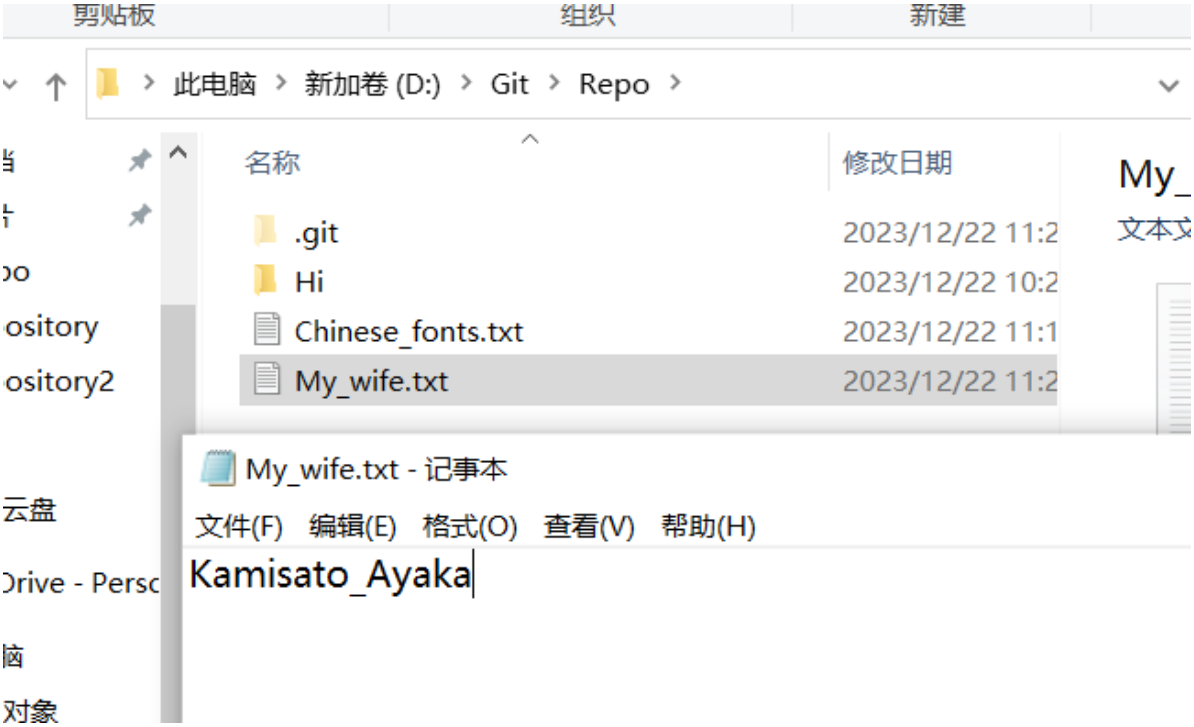
提交

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git add .

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git commit -m "kokomi"
[master 8505921] kokomi
 1 file changed, 1 insertion(+)
 create mode 100644 My_wife.txt
```

然后我们切换到Chinese分支。

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git checkout Chinese
Switched to branch 'Chinese'
```

新建同名文件，但是不同内容。



提交

```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git add .

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git commit -m "ayaka"
[Chinese 27ac574] ayaka
 1 file changed, 1 insertion(+)
 create mode 100644 My_wife.txt
```

切换到master分支，并执行合并命令。

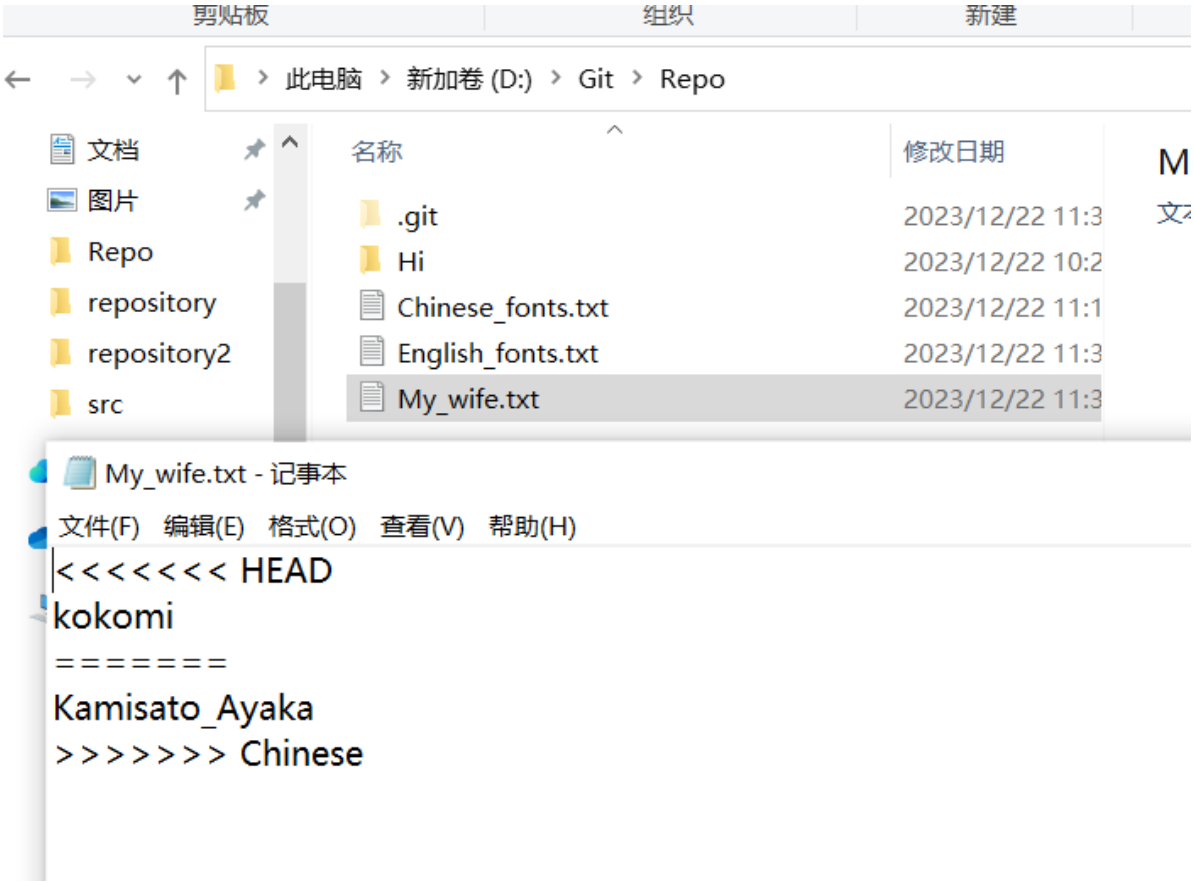
```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (Chinese)
$ git checkout master
Switched to branch 'master'

YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master)
$ git merge Chinese -m "wife"
Auto-merging My_wife.txt
CONFLICT (add/add): Merge conflict in My_wife.txt
Automatic merge failed; fix conflicts and then commit the result.
```

发现merge命令抛出了CONFLICT异常。

使用git status命令可以查看冲突文件名和具体路径。

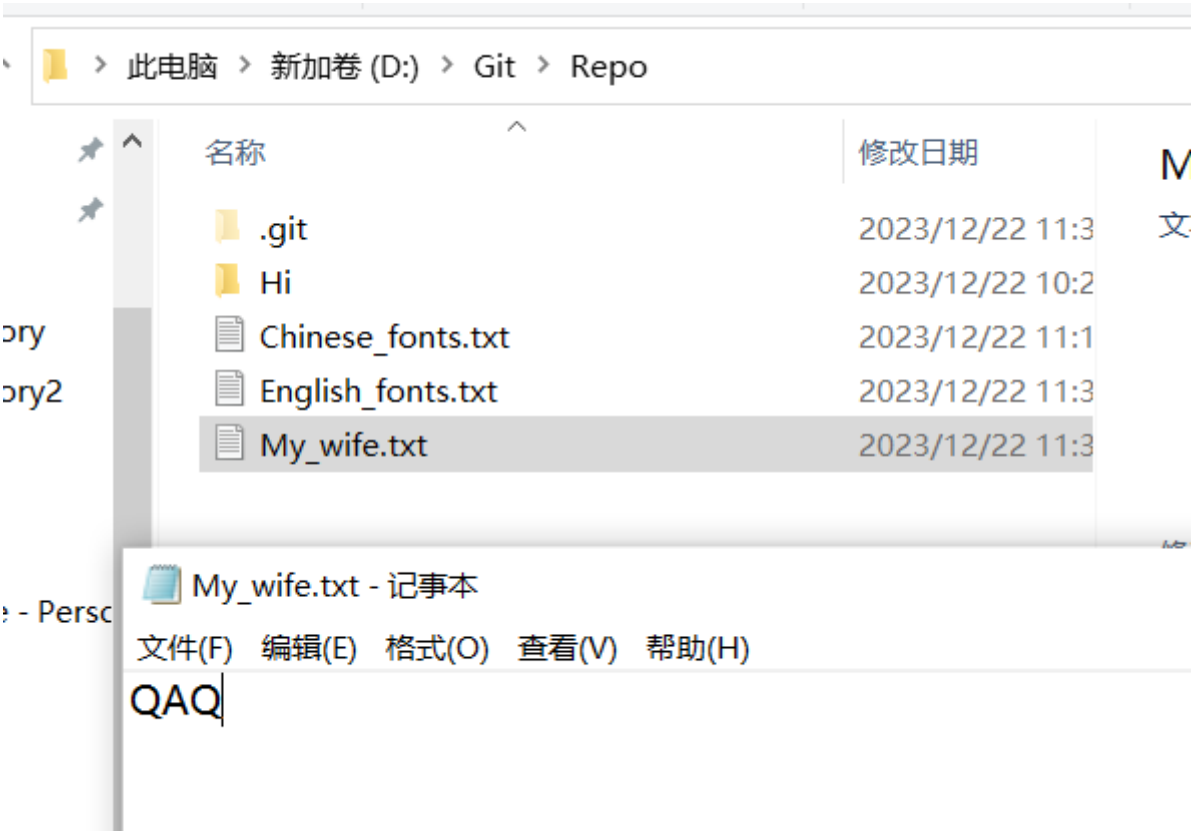
```
YXY@DESKTOP-PSGQT5E MINGW64 /d/Git/Repo (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both added:      My_wife.txt

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

我们打开工作区 (此时为master分支) 中冲突的文件 My_wife.txt

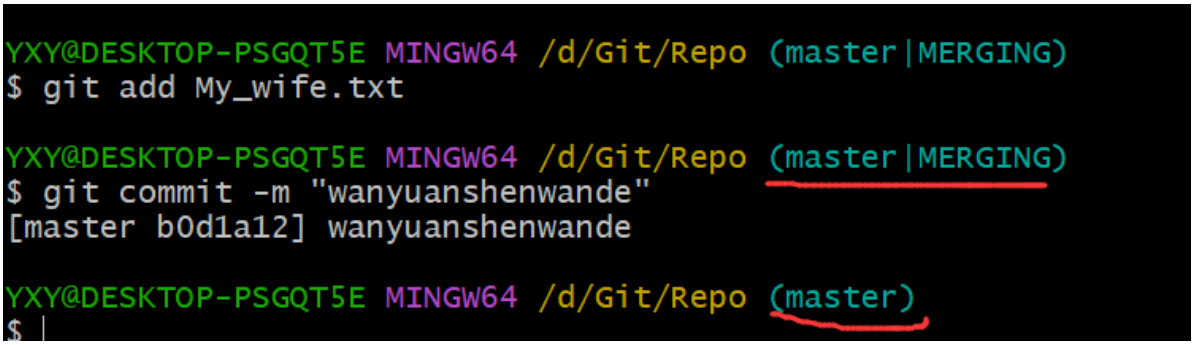


git对无法自动合并的文本进行了如上的区分，这个不用过多解释。

我们手动处理冲突部分。



手动修改完毕后，执行git add和git commit命令。



这样就成功处理Conflicts了，处理完所有的Conflicts后git会自动合并，可以看到红线处标识符改变了。

（以上是其中一类Conflict处理的介绍，想了解其它类型的Conflict请自行百度 >_<）

其实分支合并操作一般在远程库比较常见，远程库实际上自己本身就是一个分支，我们本地提交最新版本的时候也是提交的一个分支，实际上仍然是分支的合并。

以上是本地git处理版本库的简单介绍，需要注意到以上内容仅为实际常用用法的一部分，其他诸如IDEA集成Git开发、.gitignore文件等概念并没有阐述，大家可以在自己的电脑上按照入门markdown自己尝试一下，也可以自行搜索上面的概念，深入了解各种奇怪的科技，希望能帮上忙QAQ。