Git指南Part2

通过本文档你将学到

Git指南Part2
- 三、版本更新与回滚
  - 3.1 查看修改记录与版本记录
  - 3.2 穿越时空
  - 3.3 工作区和版本库
    - 3.3.1 工作区 Working Directory
    - 3.3.2 版本库 Repository
    - 3.3.3 工作原理
    - 3.3.4 撤销修改
    - 3.3.5 删除与恢复删除

三、版本更新与回滚

3.1 查看修改记录与版本记录

当我们在提交了当前版本的文件，或是从其他人那里缓存了新版本的文件，之后我们做了新的修改。在**提交之前**，我们想知道我们修改了什么内容，这是我们就用到了`git diff`指令

还记得最开始我们创建并修改的那个helloworld.py，我们把它修改成以下这样：

```
for i range(5):
    print("helloworld")
```

然后我们在git命令窗输入

```
$ git diff helloworld.py
diff --git a/helloworld.py b/helloworld.py
index d9596d9..d2d0e2b 100644
--- a/helloworld.py
+++ b/helloworld.py
@@ -1 +1,2 @@
-print("helloworld")
\ No newline at end of file
+for i in range(5):
+    print("helloworld")
\ No newline at end of file
```

确认无误之后，我们就可以运用`add`和`commit`指令提交了

当当前目录下所有的文件都已经提交之后输入`status`指令

```
$ git status
On branch master
nothing to commit, working tree clean
```

Git会告诉我们没有文件是可提交的，工作目录是干净的

然后你就问了：我们改了那么多次，**我们怎么看所有提交的记录呢?**

问得好

这时我们只需用一个简简单单的`log`指令就可以了

```
$ git log
commit 18bd5035b757972d3ecObddbdc8cc017d6d56812 (HEAD -> master)
Author: whitemo<whitemozj@outlook.com>
Date:Wed Nov 17 07:51:46 2021 +0800

    nothing

commit e55d58bc7a839b3e40190e2176e69748428485 f0
'Author: whitemo<whitemozj@outlook.com>
Date:Wed Nov 17 07:48:19 2021 +0800

    add a circle

(commit 852003627f81d03e62e307bde503e2509d16b284
Author: whitemo<whitemozj@outlook.com>
Date:Tue Nov 16 17:59:19 2021 +0800

    new a helloworld python file
```

3.2 穿越时空

如果某一次修改中，我们把文件内容修改得乱七八糟，已经没有挽救的必要了，我们不得不进行版本回滚

我们要回到上一个版本，也就是`add a circle`那一版本

首先要让Git知道我们想要的那个版本，`HEAD`**表示当前版本**，也就是`nothing`，**上一个版本就是**`HEAD^`**，上上一个就是**`HEAD^^`，当然返回过早的版本用^肯定不想，如果我们要返回前100个版本，可以用`HEAD~100`

并且使用`git reset`**指令**

```
$ git reset --hard HEAD^
HEAD is mow at e55d58b add a circle
```

然后如果想**确定内容**，可以用`cat`，当然不是“猫”

```
$ cat helloworld.py
for i in range(10):
    print("helloworld")
```

其实这时已经**顺便把文件也同时改了**

那你可能又要问了：哎，我把版本回滚了，是不是现在这个老版本变成新版本了啊?

并不是，它只是在本地成为了一个最新版本，因为还尚未提交，所以在“服务器”上，各版本的对应关系还是不变

你说你不想？我用`git reflog`给你看一下，这个用来记录你的**每一次新提交，回溯，更新的命令**

```
$ git reflog
e55d58b (HEAD -> master) HEAD@{0}: reset moving to HEAD^
18bd503 HEAD@{1}: commit: nothing
e55d58b (HEAD -> master) HEAD@{2}: commit: add a circle
8520036 HEAD@{3}: commit (initial): new a helloworld python file
```

你有说你后悔了，你还是想要原来那个新的，怎么办？**使用`reset`指令**

```
$ git reset
```

没有任何反应是不是？那就对了，我们再来一遍`git reflog`

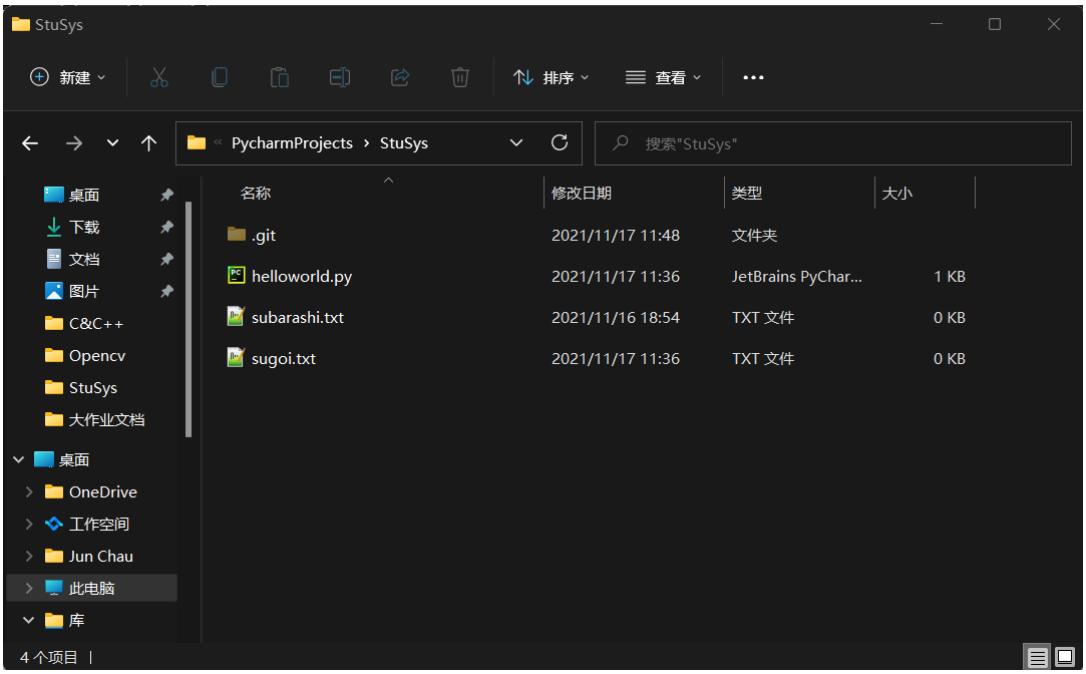
```
$ git reflog
e55d58b (HEAD -> master) HEAD@{0}: reset moving to HEAD

e55d58b (HEAD -> master) HEAD@{1}: reset moving to HEAD^
18bd503 HEAD@{2}: commit: nothing
e55d58b (HEAD -> master) HEAD@{3}: commit: add a circle
8520036 HEAD@{3}: commit (initial): new a helloworld python file
```

最前面多一行是不是？而且指向的就是`HEAD`

3.3 工作区和版本库

3.3.1 工作区 Working Directory

之前我们一直提到的文件夹，目录，这些存在本地计算机上的，就叫**工作区**



3.3.2 版本库 Repository

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

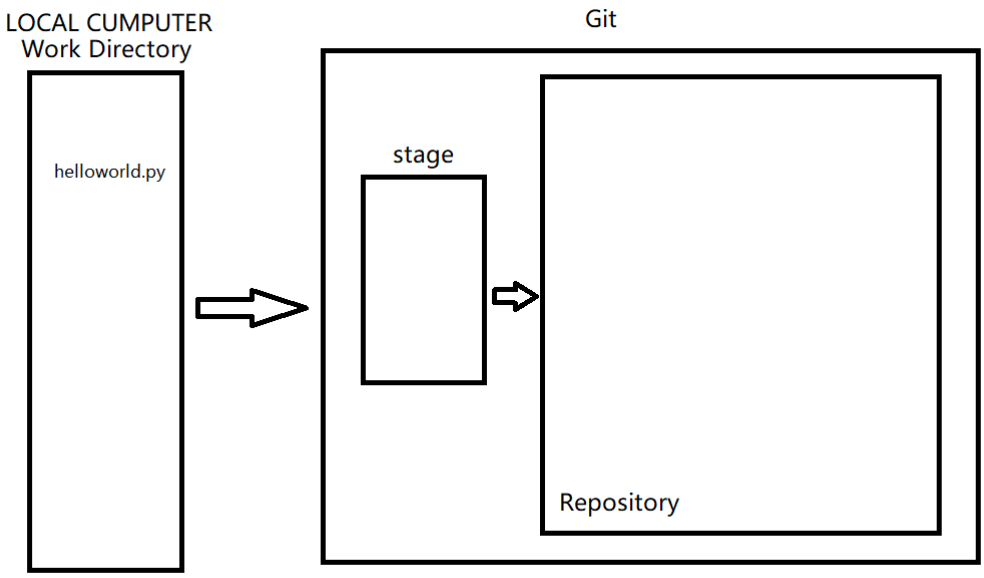
Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支`master`，以及指向`master`的一个指针叫`HEAD`（C党DNA动了）

3.3.3 工作原理

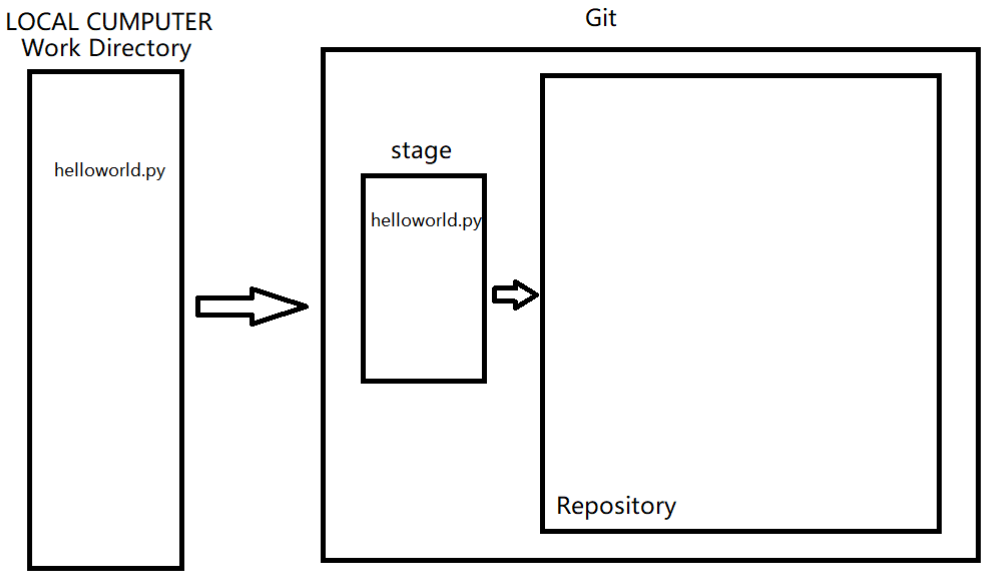
> 这部分如果没时间学可以简单看看，不要求理解

把大象放冰箱要三步，之前我们说过，把文件添加Git的版本库，只需两步，不再赘述了

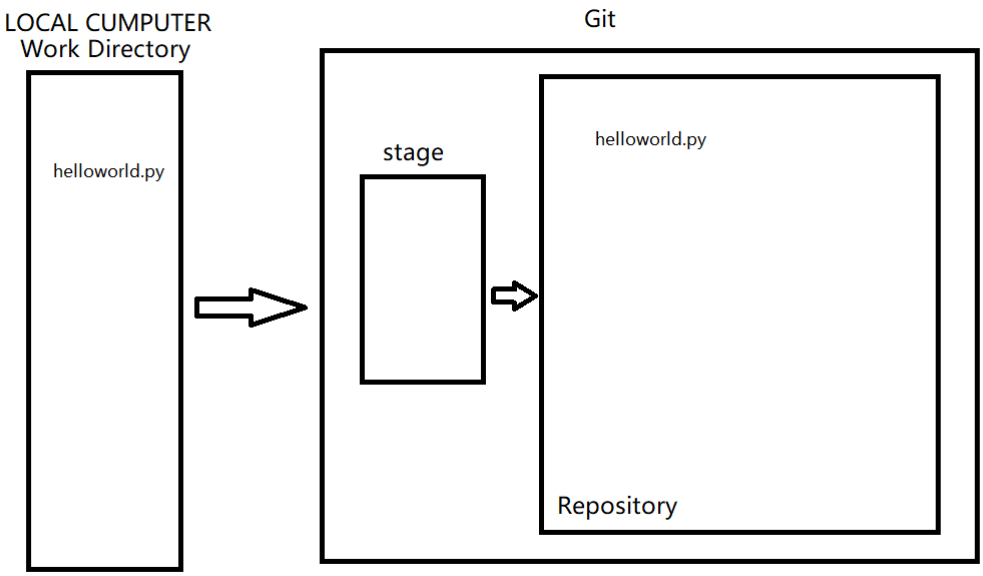
默认情况下，我们的文件只存在于我们的本地目录上



当我们输入`add`指令后，我们的文件会拷贝到**stage缓存区**



输入`commit`之后，stage内的文件就会传到**版本库**中



因为我们创建Git版本库时，Git自动为我们创建了唯一一个`master`分支，简单理解，`master`**是版本库的主分支。**所以，现在，`git commit`就是往`master`分支上提交更改

至于分支，我们后面再说

要说的是每次所`commit`文件都是`add`到暂存区的文件。所以，当你在修改了一部分之后使用了`add`指令，然后又修改了一部分之后使用了`commit`指令，则提交到版本库里的只是前面修改的那一部分，而后面的部分不会更新。如果要提交后面的一部分修改，则再使用一次`add`即可

3.3.4 撤销修改

如果有一次你在原有的基础上做了一些修改，而这些修改不满足你的预期，你就会想把它重写，但是你又不知道当初从何写起的了。

如果用`git status`看一下

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

    modified:   helloworld.py

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

你可以发现，Git会告诉你，`git checkout -- file`可以丢弃工作区的修改：

```
$ git checkout -- helloworld.py
```

命令`git checkout -- helloworld.py`意思就是，把`helloworld.py`文件在工作区的修改全部撤销，这里有两种情况：

一种是`helloworld.py`自修改后还**没有被放到暂存区**，现在，撤销修改就**回到和版本库一模一样的状态**；

一种是`helloworld.py`已经**添加到暂存区后，又作了修改**，现在，撤销修改就**回到添加到暂存区后的状态。**

总之，就是让这个文件回到最近一次`git commit`或`git add`时的状态。

`git checkout -- file`命令中的`--`**很重要**，没有`--`，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到`git checkout`命令。

现在假定你又写错了一些东西，还`git add`到暂存区了，之后没再修改

在`commit`之前，你发现了这个问题。用`git status`查看一下，修改只是添加到了暂存区，还没有提交：

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   helloworld.py
```

Git同样告诉我们，用命令`git reset HEAD <file>`可以把暂存区的修改撤销掉（unstage），重新放回工作区：

```
$ git reset HEAD helloworld.py
Unstaged changes after reset:
M   helloworld.py
```

再用`git status`查看一下，现在暂存区是干净的，工作区有修改：

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

    modified:   helloworld.py
```

如果要回到我们修改之前，按照之前说的撤销修改的方法就可以了

假设你不但改错了东西，还从暂存区提交到了版本库，怎么办呢？还记得**穿越时空**一节吗？可以回退到上一个版本。不过，这是有条件的，就是你还没有把自己的本地版本库推送到远程。还记得Git是分布式版本控制系统吗？我们后面会讲到远程版本库，一旦文件被提交推送到远程版本库，那我也没办法了，只能找管事的修改远程版本库了。

3.3.5 删除与恢复删除

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用`rm`命令删了：

```
$ rm helloworld.py
```

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，`git status`命令会立刻告诉你哪些文件被删除了：

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    helloworld.py

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

现在你有两个选择，一是**确实要从版本库中删除该文件**，那就用命令`git rm`删掉，并且`git commit`：

```
$ git rm helloworld.py
rm 'helloworld.py'

$ git commit -m "remove helloworld.py"
[master d46f35e] remove helloworld.py
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

现在，文件就从版本库中被删除了。

另一种情况是**删错了**，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

```
$ git checkout -- helloworld.py
```

`git checkout`其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“**一键还原**”。