# **Git基础使用**

## 2.1 Git基础概念

Git是一个免费的，开源的分布式版本控制软件系统，学习Git软件的具体操作前，我们需要对一些基础的概念和名词进行解释

### 2.1.1 版本控制

一般情况下，一份文件，无论是DOC办公文档，还是编程源码文件，我们都会对文件进行大量的修改和变更。但是我们无法保证每一次的修改和变更都是正确并有效的，往往有的时候需要追溯历史操作，而版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

没有进行版本控制或者版本控制本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并发性、软件源代码的安全性，以及软件的整合等问题。

### 2.1.2 分布式

在Git中，每个版本库都是一样重要得。所以就不存在像集中式版本控制软件中以谁为主得问题。任何一个库都可以当成主库。

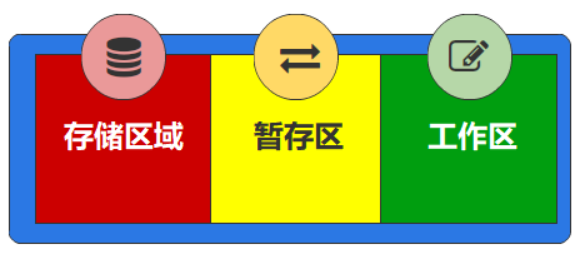
这种方式可以更大限度地保证项目资源得安全。

### 2.1.3 系统

一般软件系统指的是可以独立运行的软件应用程序。而Git软件就是专门用于对代码文件进行版本控制得应用程序。同时也提供客户端对系统所管理得资源进行访问。

### 2.1.4 区域

Git软件为了更方便地对文件进行版本控制，根据功能得不同划分了三个区域



* 存储区域：Git软件用于存储资源得区域。一般指得就是.git文件夹
* 工作区域：Git软件对外提供资源得区域，此区域可人工对资源进行处理。
* 暂存区：Git用于比对存储区域和工作区域得区域。Git根据对比得结果，可以对不同状态得文件执行操作。

## 2.2 Git基础指令

Git软件是免费、开源的。最初Git软件是为辅助 Linux 内核开发的一套软件，所以在使用时，简单常用的linux系统操作指令是可以直接使用的

### 2.2.1 linux系统操作指令

|  |  |  |
| --- | --- | --- |
| **指令** | **含义** | **说明** |
| cd 目录 | change directory | 改变操作目录 |
| cd .. |  | 退回到上一级目录 |
| pwd | Print work directory | 打印工作目录 |
| ls | list directory contents | 显示当前目录的文件及子文件目录 |
| ll | ls -l 简化版本 | 更详细地显示当前目录的文件及子文件目录 |
| mkdir 文件夹名称 | make directory | 新建一个文件夹 |
| rm 文件 | remove | 删除文件 |
| rm -r 文件夹 | Remove | 删除文件目录 |
| touch 文件 |  | 如果创建的文件不存在，那么创建一个空文件 |
| reset |  | 清屏 |
| clear |  | 清屏 |
| exit |  | 退出终端窗口 |

### 2.2.2 Git软件指令

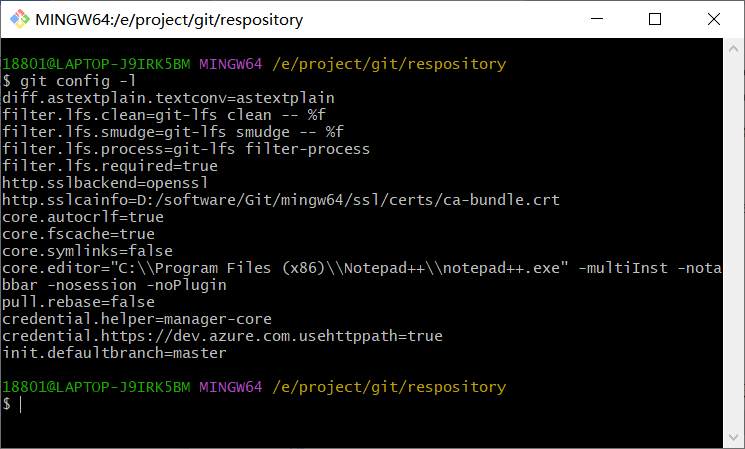
#### 2.2.2.1 配置信息

作为一个工具软件来讲，一般都会有默认的配置文件来保存基础的配置信息，Git软件的配置文件位置为：**Git安装路径/etc/gitconfig**



默认情况下，我们可以通过指令获取软件的配置信息：

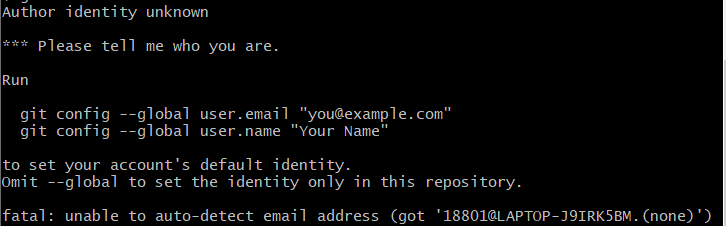
git config -l



#### 2.2.2.2 名称和邮箱

如果你是第一回使用Git软件，需要告诉Git软件你的名称和邮箱，否则是无法将文件纳入到版本库中进行版本管理的。这是因为在多人协作时，不同的用户可能对同一个文件进行操作，所以Git软件必须区分不同用户的操作，区分的方式就是名称和邮箱。

当然了，你可能会说我就用本地库就行了，不需要进行多人协作，是不是就可以不用配置呢。这是不行的，因为Git软件的设计初衷本身就是针对于linux系统的分布式开发协同工作，所以它天生就是用于分布式协同工作的，这里无论你是否使用这个功能，它本身就是这么设计的。所以是一定要配置的，否则就会出现如下提示：



当然了，配置的过程并不复杂，输入相关指令即可

git config --global user.name test

git config --global user.email test@atguigu.com

这里的--global表示全局配置，后续的所有文件操作都会使用该用户名称及邮箱。此时在操作系统的用户目录，会产生新的配置文件



文件中就包含了刚刚增加的配置信息

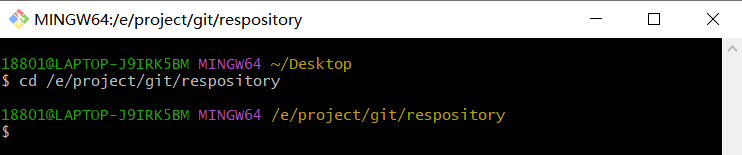


#### 2.2.2.3 初始化版本库

Git软件主要用于管理文件的版本信息，但它只是一个软件，不可能安装后就直接将系统中所有的文件全部纳入到它的管理范畴中。并且，软件管理版本信息的主要目就是管理文件的修改和变更，如果将系统中所有文件都进行管理其实意义是不大的。所以一般情况下，我们需要指定某一个文件目录作为软件的管理目录。因为这个目录主要就作为Git软件的管理文件的版本变化信息，所以这个目录也称之为Git软件的版本仓库目录。

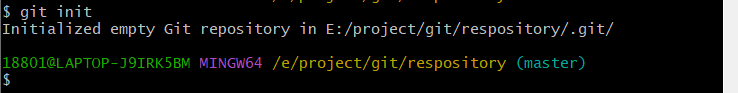
具体操作过程如下：

* 我们首先通过指令进入到指定文件目录

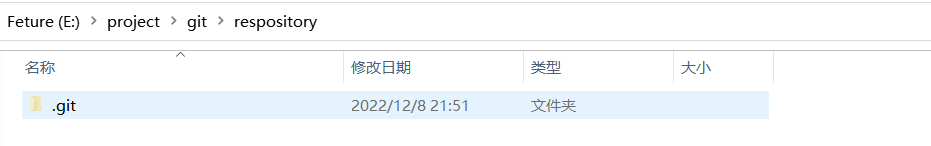


* 执行指定的指令，创建文件版本库

git init



* 版本库创建成功后，会在目录中创建.git目录，用于管理当前版本库。



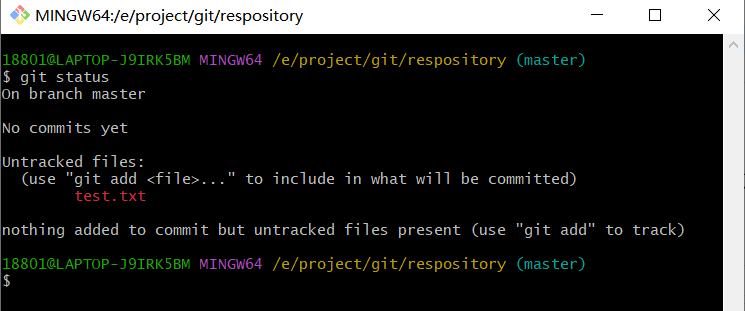
#### 2.2.2.4 向版本库中添加文件

虽然创建了版本库，但是现在版本库中还没有任何的文件，所以这里我们先手动创建文件：test.txt



因为文件已经放置在版本库中了。所以可以通过软件的指令查看版本库状态

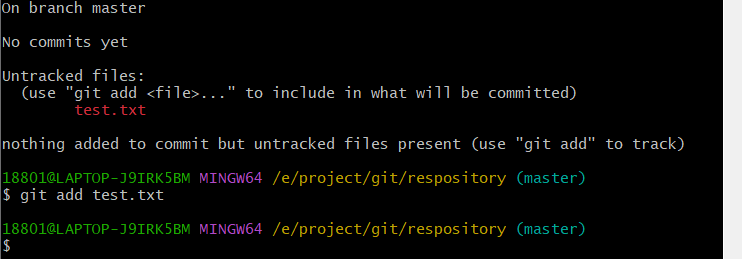
git status



此时会发现，test.txt文件属于untracked files（未追踪文件）,这里表示当前的test.txt文件虽然放置到了版本库的文件目录中，被Git软件识别到了，但是未纳入到版本库管理中。所以属于未追踪文件。通过这个现象可以认为，系统文件夹物理目录和版本库管理目录的含义是不一样的。只有文件被纳入到版本库管理后，Git软件才能对文件修改后的不同版本内容进行追踪处理，也就是所谓的tracked files了。那么如何将文件纳入到版本库的管理呢，这就需要我们执行以下命令了：

#这里的文件是需要提供扩展名的

git add test.txt



此时你再查看版本库状态，就会发现文件状态的变化。



你会发现，此时文件状态为cached file，这是什么意思呢？其实这也是Git管理文件时的一种状态：暂存状态。就是我们生活中常说的草稿状态。也就是说对于Git来讲，它认为此时文件只是一种临时草稿状态，随时可能会进行修改或删除，并不算真正的操作完成状态。所以并不会把文件纳入到版本库管理中。

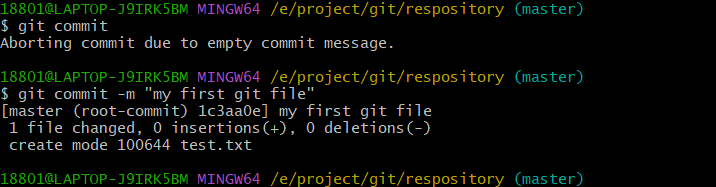
为什么会这样呢？其实这就涉及到版本的作用。生活中，我们学习时，一般会写学习笔记，虽然写完后不一定会看，但是该写的还是要写的。然后给这些笔记文件起名时，一般就会带着当天的时间或数字。比如【Java学习笔记\_20220101.md】，或者【Java学习笔记\_Ver1.1.md】，这里的时间或数字主要作用就是用于区分同一份笔记在不同时间节点记录的内容，这里的数字或时间我们就称之为版本。

那如果你只是随便写写，或写到一半，还没有写完的话，会专门给文件改个名称吗？应该不会，对不对，因为对于你来讲，这个笔记文件并没有记录完成，对吗。但是你非得说，你今天不想继续学习了，然后给文件改了一个名称，也不是不可以。对于Git软件来讲，道理是一样的。如果确定要把文件放置在版本库中，那么就需要执行确定提交指令

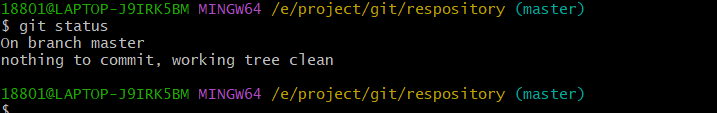
# commit表示真正地纳入到版本库中

# -m 表示提交时的信息（message），是必须输入的。用于描述不同版本之间的差别信息

git commit -m "my first git file"

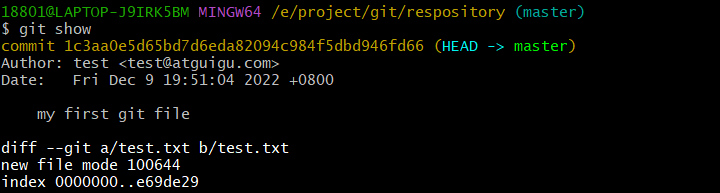


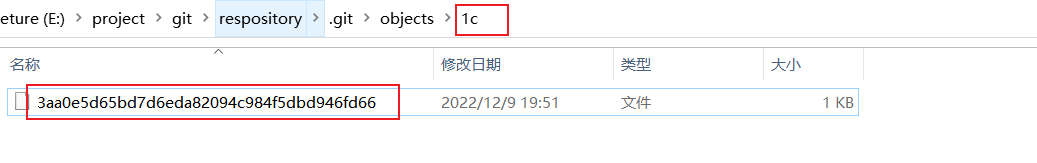
再查看Git状态



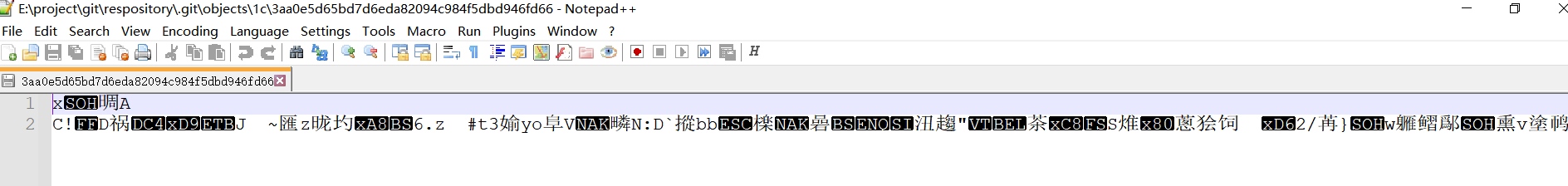
提交后，Git会对当前的操作进行Hash计算，通过计算后的值将数据保存下来，保存的位置为版本库.git文件目录的objects中，我们可以通过指令查看当前提交

git show



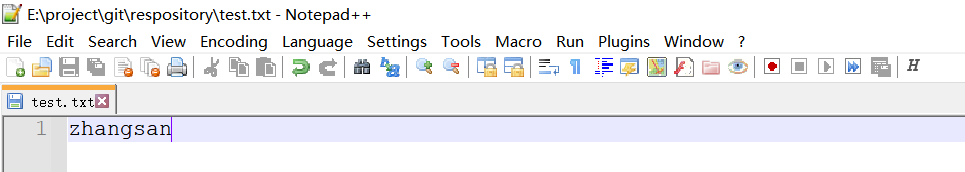


由于文件内容进行了转换处理，直接打开你是看不懂的

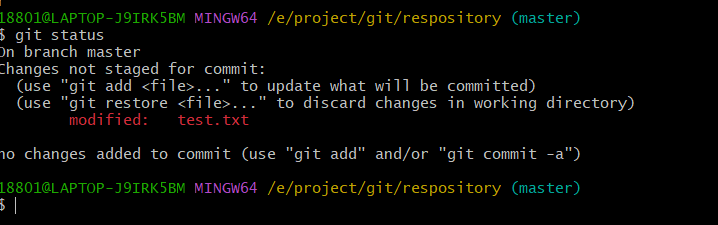


#### 2.2.2.5 修改版本库文件

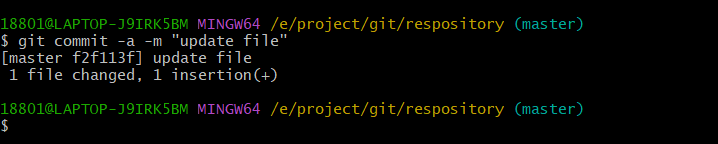
现在文件已经被纳入到版本库中，因为咱们的文件是空的，所以这里我们增加一些内容



此时，Git版本库中的文件和本地的文件就有了不同。我们可以查看状态



**modified**表示文件已经修改了，我们可以把这一次的修改提交到版本库中



原则上来讲，这里的操作顺序依然应该是

# 先增加，再提交

git add test.txt

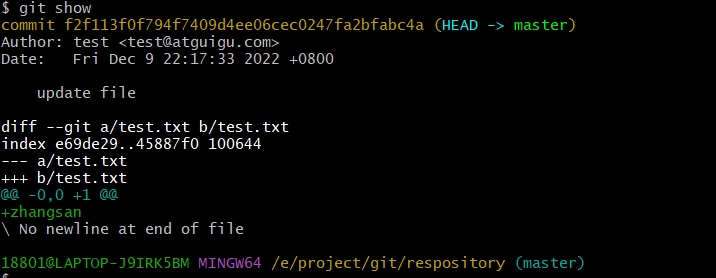
git commit

但是这里我们简化了一下操作

git commit -a -m "update file"

这个指令操作中多了一个**-a**的参数，等同于将增加，提交两步操作融合成了一步。

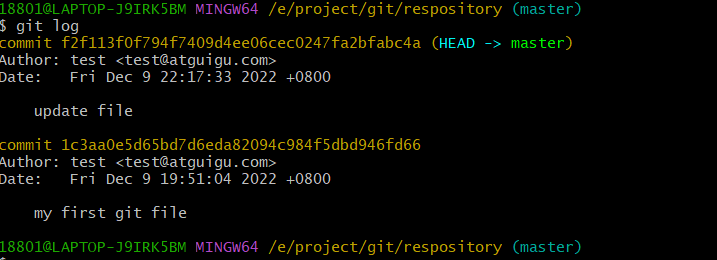
提交成功后，我们来展示以下当前Git软件库



#### 2.2.2.6 查看版本库文件历史

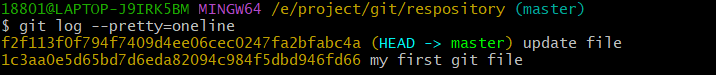
版本库中的文件我们已经修改并提交了，那么文件的版本信息就会发生变化，那我们如何来查看这个变化呢？这里我们可以采用log指令进行查看

git log



如果感觉看着不舒服，也可以美化一下显示方式:

git log --pretty=oneline



也可以使用简单方式查看

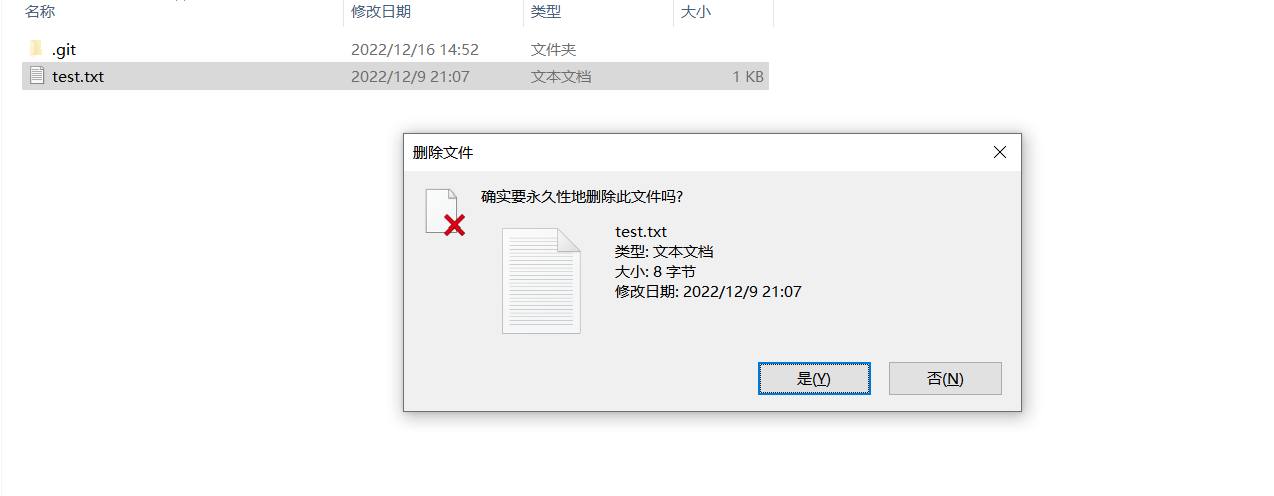
git log --oneline



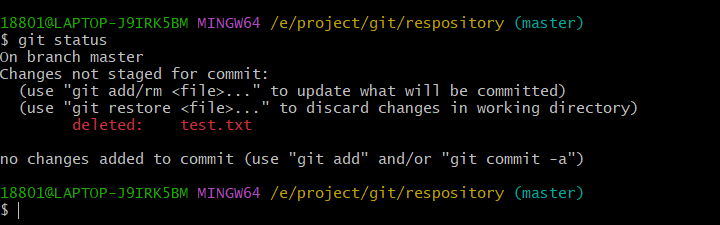
#### 2.2.2.7 删除文件

一般情况下，Git软件就是用于管理文件的版本变更，但是在一些特殊的场景中，文件可能作废或不再使用，那么就需要从版本库中删除，记住，这里说的并不是从物理文件目录中删除，而是从版本库中删除。

* 将本地文件从目录中删除



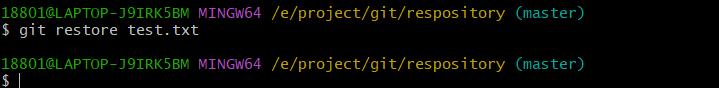
* 查看Git版本库状态信息



此时Git软件会识别出来，版本库中有一份文件和当前用于临时操作文件的暂存区内的文件状态不一致：版本库中文件还在，但是操作区内的文件已经没有了。所以软件提供了两个选择：一个是将版本库中的文件也进行（提交）删除操作。另外一个就是从版本库中恢复文件。

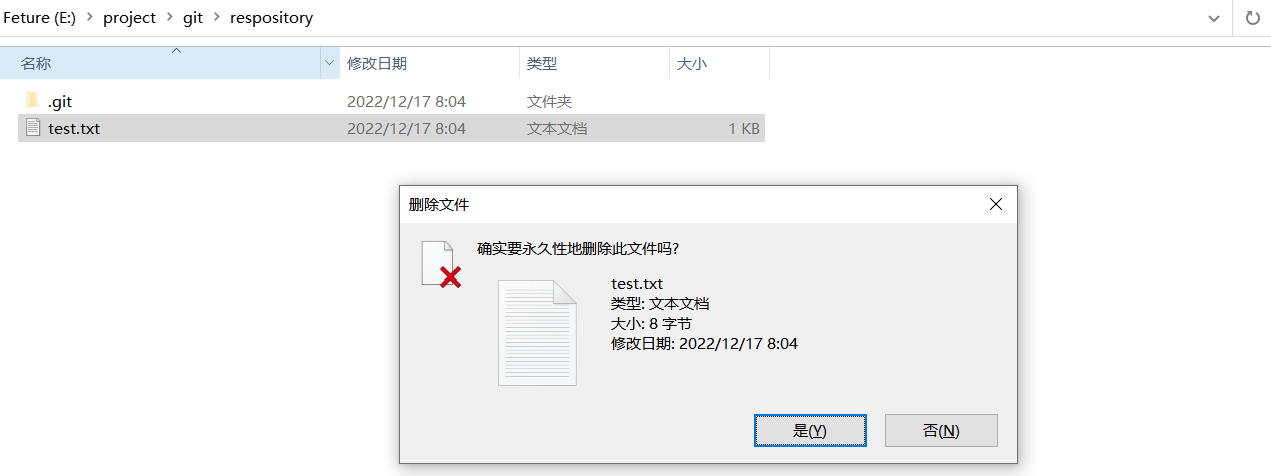
* 使用指令从版本库中恢复文件

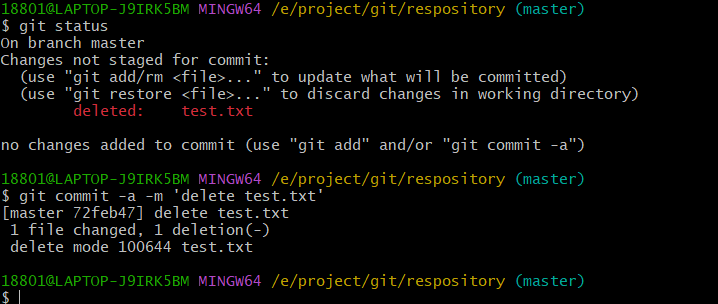
git restore test.txt



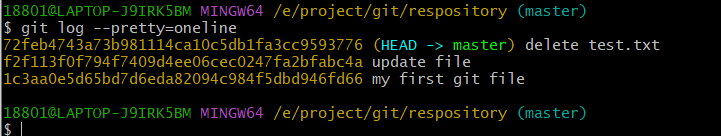


* 如果想要真正删除文件，那么也要将版本库中同时删除





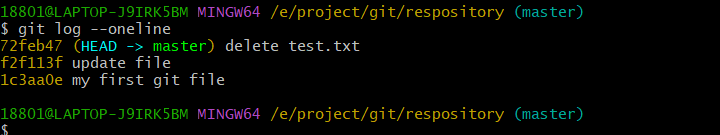
此时查看Git日志



#### 2.2.2.8 恢复历史文件

如果版本库中一份文件中已经被删除了，那么此时这份文件还能找回来吗？其实原则上来讲，已经不行了，因为文件删除本身也是一种变更操作，也算是版本库管理的一部分。所以想要将已经删除的那份文件从版本库中取出来，已经是不可能了。但是，要注意的是，版本库管理的是文件不同版本的变更操作，这个不同版本的概念还是非常重要的。也就是说，最后的那个删除的文件版本已经没有了，但是之前版本的文件其实还是存在的。所以如果我们能将文件恢复到某一个版本，那么那个版本的文件就依然存在。

* 查看版本库信息



* 将版本库文件重置到某一个版本

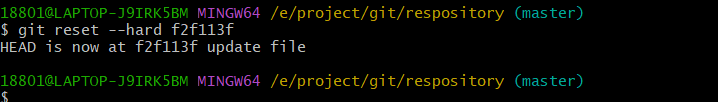
# 这里的f2f113f就是版本Hash值，用于唯一确定版本库中此版本的标记

# 当然了这是一个简短版，完整的比较长

# 如果不记得具体的版本值，版本值也可以使用HEAD值，比如最新的上一个版本：HEAD^

# 如果后退更多的版本,可以使用 HEAD~N

git reset --hard f2f113f



* 被删除的文件回来了

