前言

上一节主要是对Git进行安装并且配置了一些信息，Git是用来存放代码所以可以说Git本质就是数据库。这一节主要是对Git中**区域和对象**这个概念进行讲述。

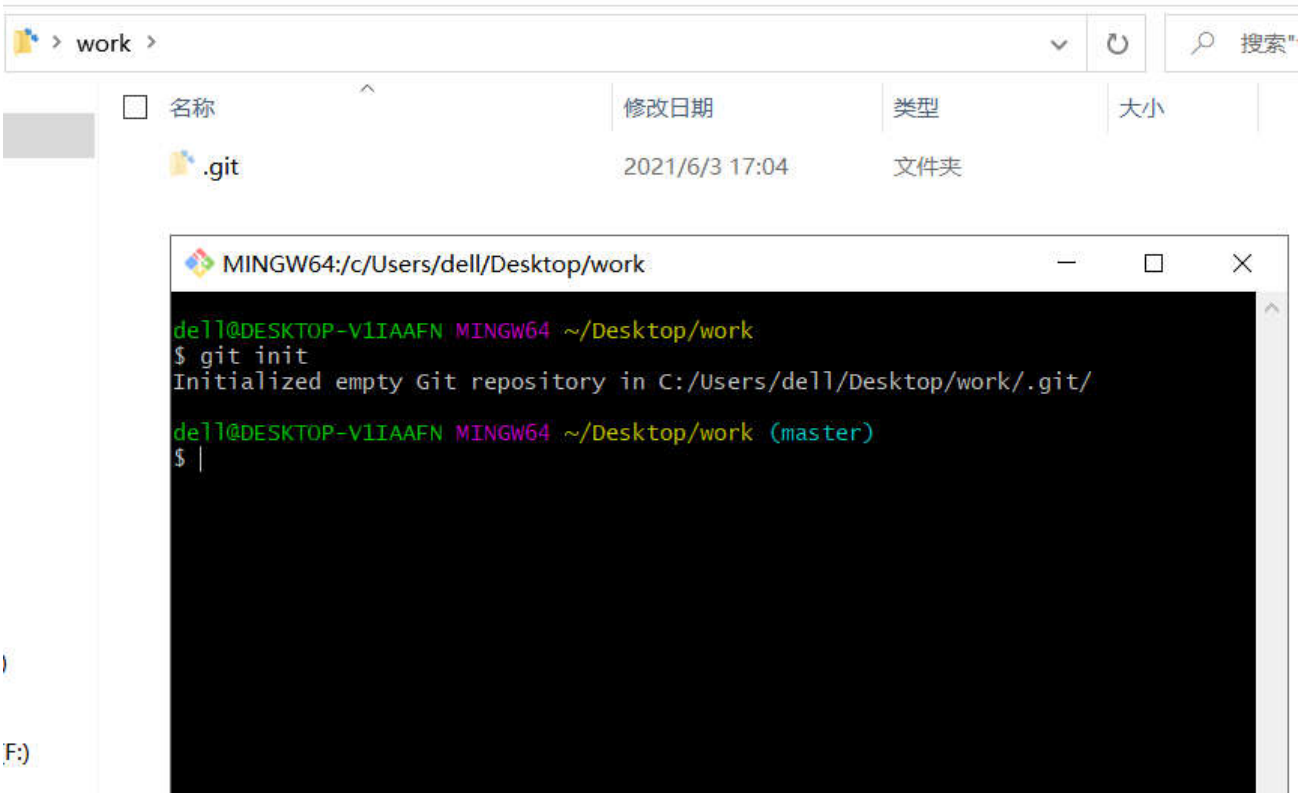
一、区域

工作区

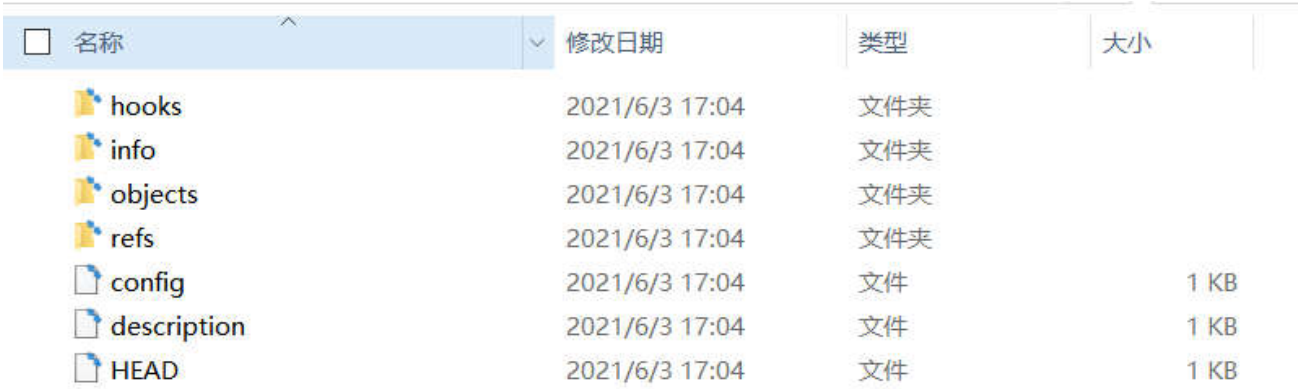
工作区从名字中进行分析可以理解为工作时候的区域,也就是我们编写代码的那个页面。工作区那肯定是想怎么玩就怎么玩，所以处于工作区的文件Git是不会进行管理的。那桌面创建一个文件夹那就是工作区咯，但是还不是Git仓库与Git是没有联系的。

暂存区

暂存区可以理解为暂时存储东西的区域，那为啥需要暂存区呢？其实道理很简单，我的项目分多个模块其中一个模块完成了也不需要修改，那个此时就可以将其放入暂存区。暂存区所存放的内容是项目内容的一部分，所以暂存区文件Git是进行管理的。打开刚刚创建的文件夹，在里面右击鼠标选择**Git Bash Here**然后在命令行界面输入 **git init**初始Git仓库。(PS:如果你看不见.git文件夹有可能是没有勾选显示隐藏文件)



我们继续打开.git文件夹看看里面有什么奥秘。



hooks: 目录包含客户端或者是服务端的钩子(应用场景应该是在提交代码之前进行一些操作，比如检测代码格式是否规范等一些操作)
info: 包含一个全局性排除文件(将不需要Git管理的文件放入info里面)
objects: 存放所有的数据内容。
refs: 存放数据(分支)的提交对象的指针。(PS:后面讨论分支的时候会在次说明)

config: 里面记录了项目的一些配置信息。
description: 对仓库的描述信息。
HEAD: 表明目前被检出的分支。

文件夹中的内容都介绍完了，SO~那个是暂存区？因为没有向暂存区中存放东西所以目前没有暂存区。 - -！如果有暂存区那个他的名字应该是**index**！！

版本库

版本库从名字进行分析可以理解为项目的雏形，那么版本库就是暂存区中多个文件的合集。那版本库也必然是Git进行管理的。上图中 **.git文件夹**就是版本库

所以按照我们理解应该是工作区的文件先进入暂存区，然后暂存区的文件那进入版本库，但是具体还需要进行验证。

二、对象

Git对象

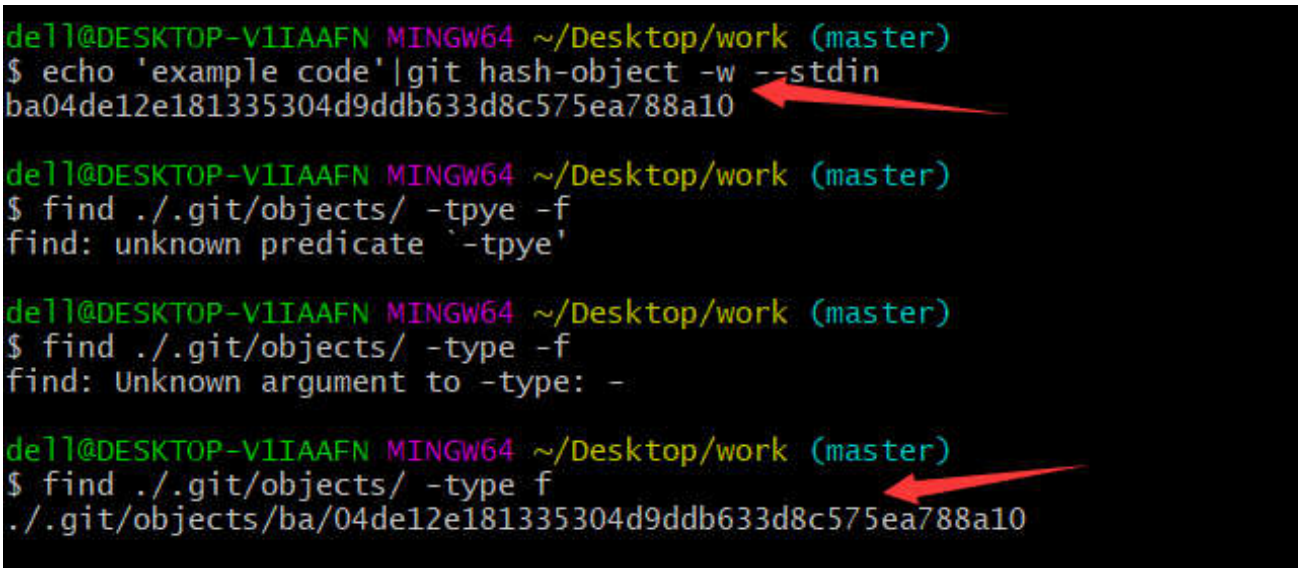
Git的核心部分是一个简单的键值对数据库（key-value data store）。你可以向Git仓库中插入任意类型的内容，它会返回一个唯一的键，通过该键可以在任意时刻再次取回该内容。
命令:

echo 'example code'|git hash-object -w --stdin

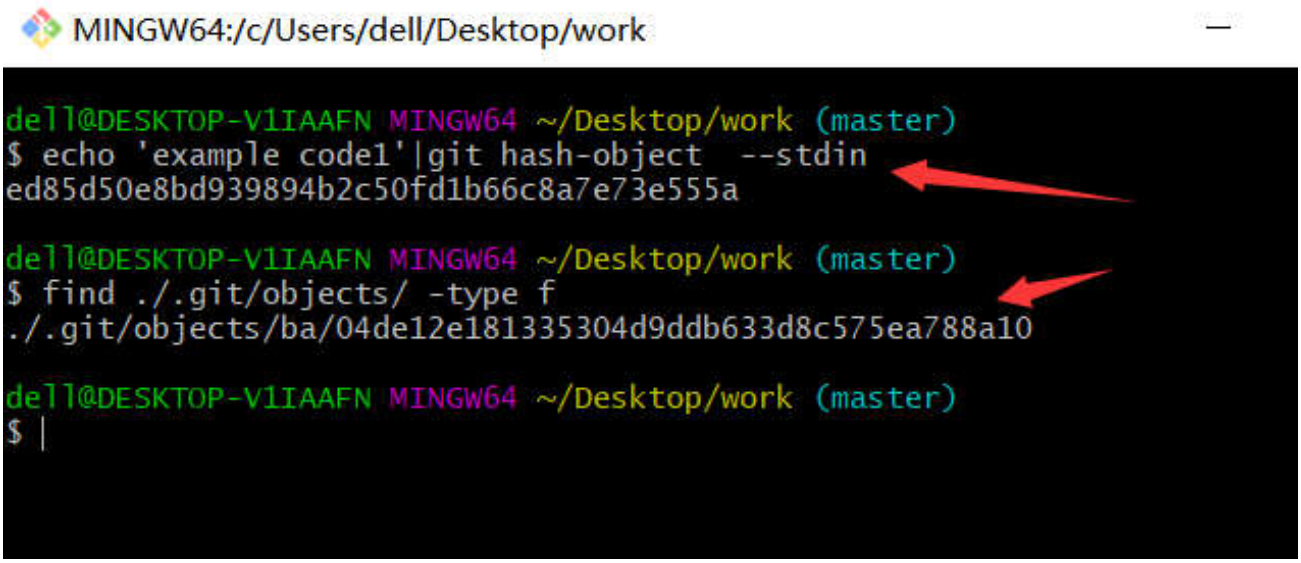
-w 选项指示 hash-object 命令存储数据对象；若没有-w那么只单纯的返回键值(键值是一个长度40个字符的校验和。是一个SHA-1哈希值)只有加上-w 才能存入数据库(版本库)中。

--stdin 选项指示该命令从标准输入读取内容，如果不指定需要在命令结尾给出带写入文件的路径。

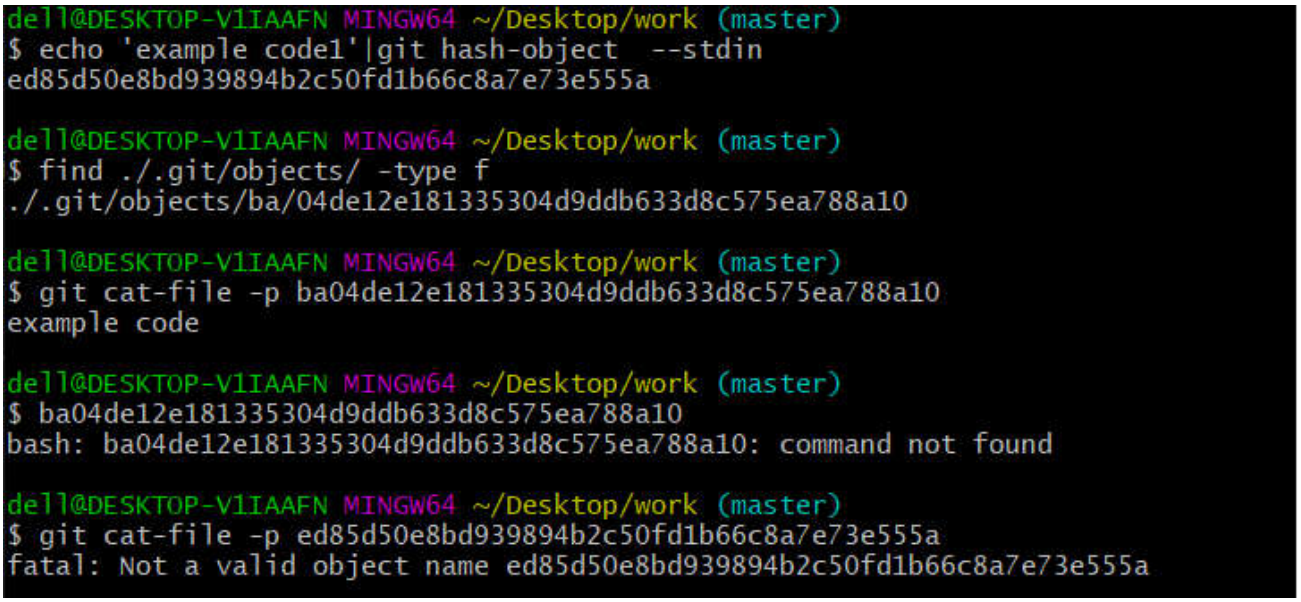
git hash-object -w 文件路径



因为Objects是Git存放所有内容的地方，所以通过看这里就能看见是否将其存入进去。可以看见返回的哈希值的前两位作为文件夹的名字，后面的位数作为文件名字，内容不同那么键值就不同。

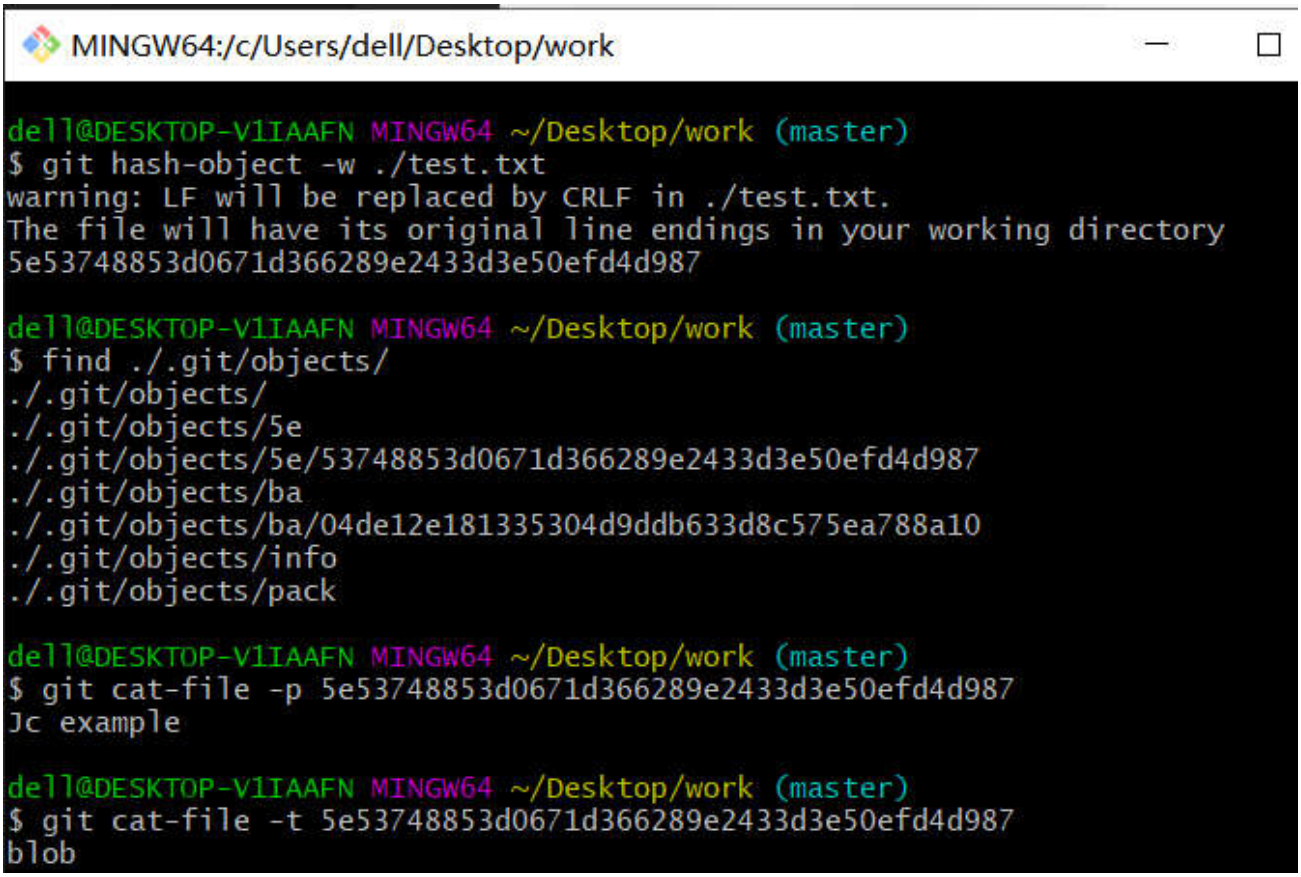


我们也可以根据键值来查看内容,通过git cat-file -p 键值 命令来获得其内容。其中 **-p** 可以自动判断文件的类型让其显示友好的格式。



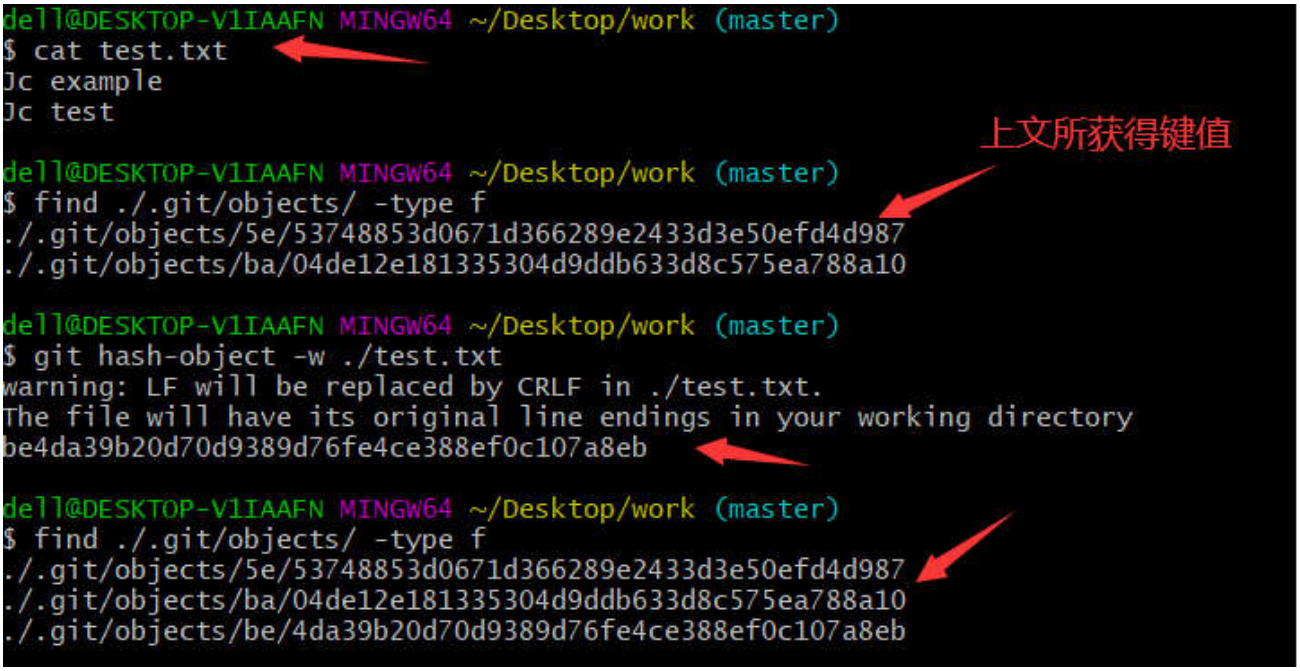
没有加 **-w** 所得到的键值是没有办法查看的，因为他根本就没有存入Git版本库的呀！！

我们日常都是对文件进行存储和控制而不是从标准输入获得内容，我给一个大致的演示。



)

当然我们也可以对刚才存入版本库中的text.txt中的内容进行修改，但是修改之后的内容并不会由Git管理，需要再次用 git hash-object -w ./test.txt 将其提交给Git。



Git对象总结： Git对象就是(key:val)组成的键值对。也就是说将我们所需要的内容做成键值对，键是我的内容所对应的哈希，值则是内容,这就是 **Git对象** ,通过命令git cat-file -t 键值(要自己实践哦！) 可以看见是什么对象,通过命令可以看见键值对是blob类型。

问题： 一个Git对象只能代表一个文件的版本不能代表项目的一个版本，而且Git对象只保存了文件的内容没有保存文件名。那这个解决方案就是通过 **树对象** 进行解决

注意： 目前操作都是在本地数据库进行的不涉及暂存器，直接将数据内容放进版本库里面。

树对象

树对象，可以解决文件名保存的问题，允许多个文件组织到一起。Git以类似于Unix文件系统的方式存储内容。所有的内容均已树对象和数据对象(Git对象)的形式存储，其中的树对象对应Unix中的目录项，数据对象大致对应文件的内容。一个树对象可以包含一条或者多条记录(每条记录含有一个指向git对象或者数对象的哈希指针，以及相应的模式、类型、文件名信息)。一个树对象可以包含另外一个树对象。

构建数对象:

我们通过updata-index;write-tree;read-tree等命令来构建树对象并放

进缓存区。

通过 **updata-index 命令** 给test.txt文件的首个版本创建一个暂存区，通过 **write-tree** 命令生成数对象。

命令:

git updata-index-add --cacheinfo 100644 哈希值 test.txt

git write tree

100644——是一个普通文件；100755——可执行文件；120000——一个符号链接。

--add选项:因为此前文件并不在缓存区，第一次需要加上--add选项

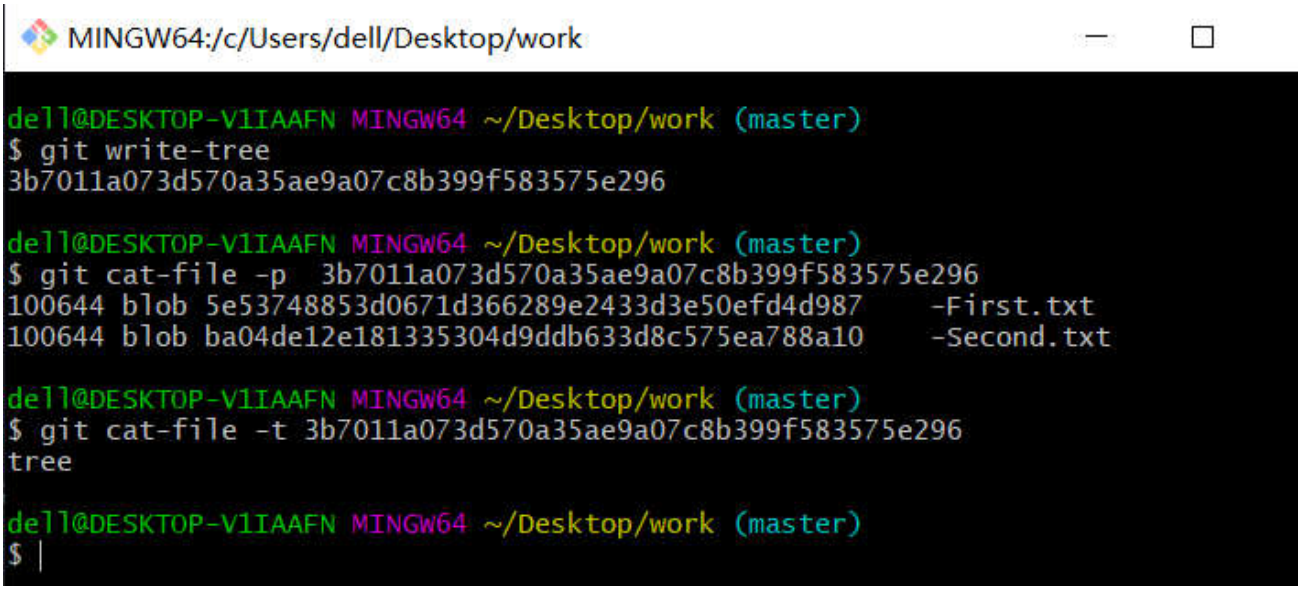
--cacheinfo选项:需要将所添加文件放置在Git数据库中，而不是当前目录。

前面在看.git文件夹的时候说暂存区没有用到所以文件夹中并没有显示出来,我们现在可以利用 updata-index来创建一个暂存区看看咯，验证我说的对不对哈哈。

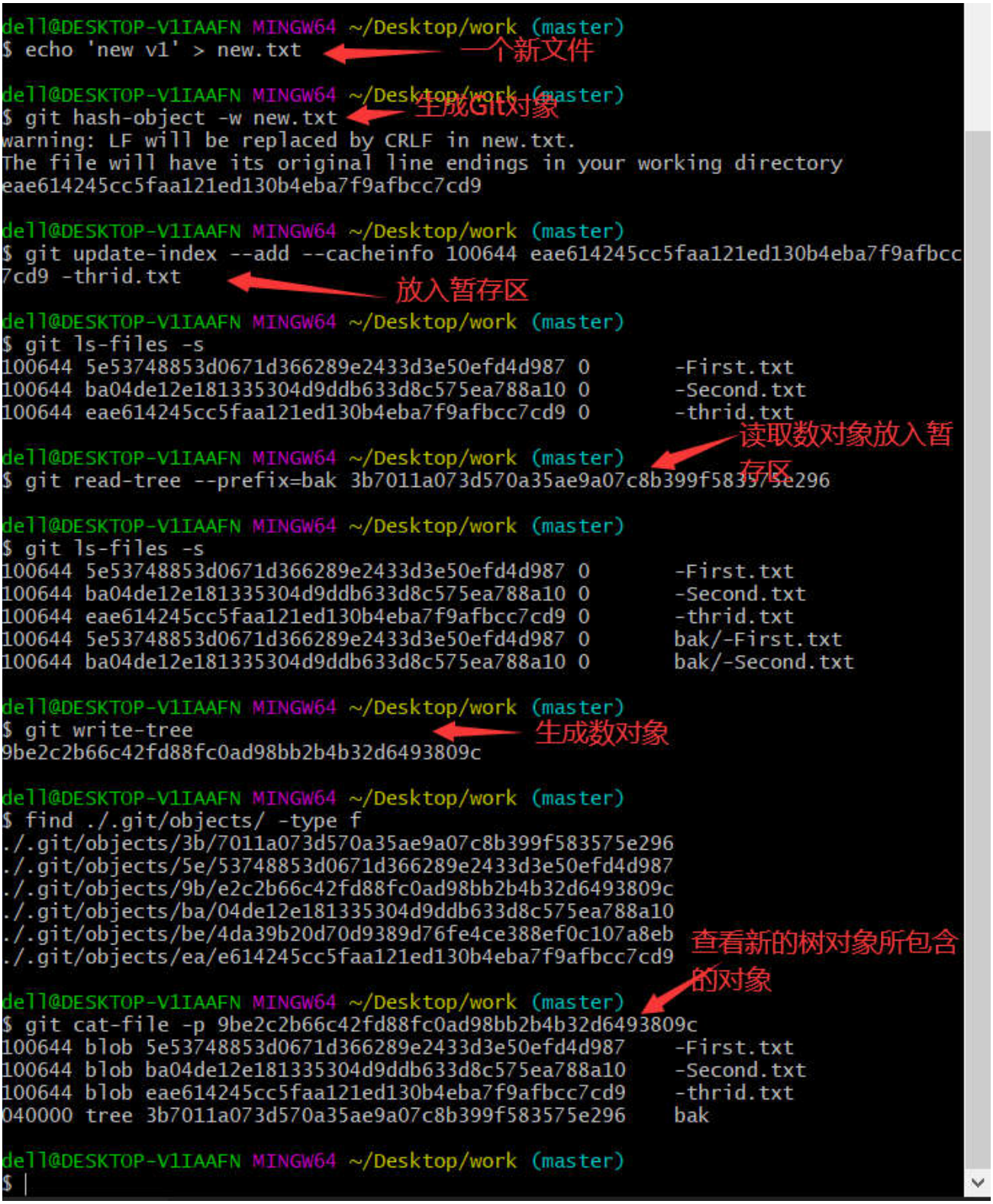
```
//首次创建text.txt里面的内容
$ git cat-file -p 5e53748853d0671d366289e2433d3e50efd4d987
Jc example

//修改text.txt之后里面的内容
$ git cat-file -p be4da39b20d70d9389d76fe4ce388ef0c107a8eb
Jc example
Jc test

//第一次从标准输入写入的内容
$ git cat-file -p ba04de12e181335304d9ddb633d8c575ea788a10
example code

//当我们执行完这个条命令以后 .git文件夹就出现了index文件,此命令并没有往
$ git update-index --add --cacheinfo 100644 5e53748853d0671d36628

$ git update-index --add --cacheinfo 100644 ba04de12e181335304d9d

//通过下面的命令可以看暂存区里面的内容
$ git ls-files -s
```

```
100644 5e53748853d0671d366289e2433d3e50efd4d987 0       -First.tx
100644 ba04de12e181335304d9ddb633d8c575ea788a10 0       -Second.t

//通过查看发现版本库里面的内容并没有增加,验证了上面的说法
$ find ./.git/objects/ -type f
./.git/objects/5e/53748853d0671d366289e2433d3e50efd4d987
./.git/objects/ba/04de12e181335304d9ddb633d8c575ea788a10
./.git/objects/be/4da39b20d70d9389d76fe4ce388ef0c107a8eb
```

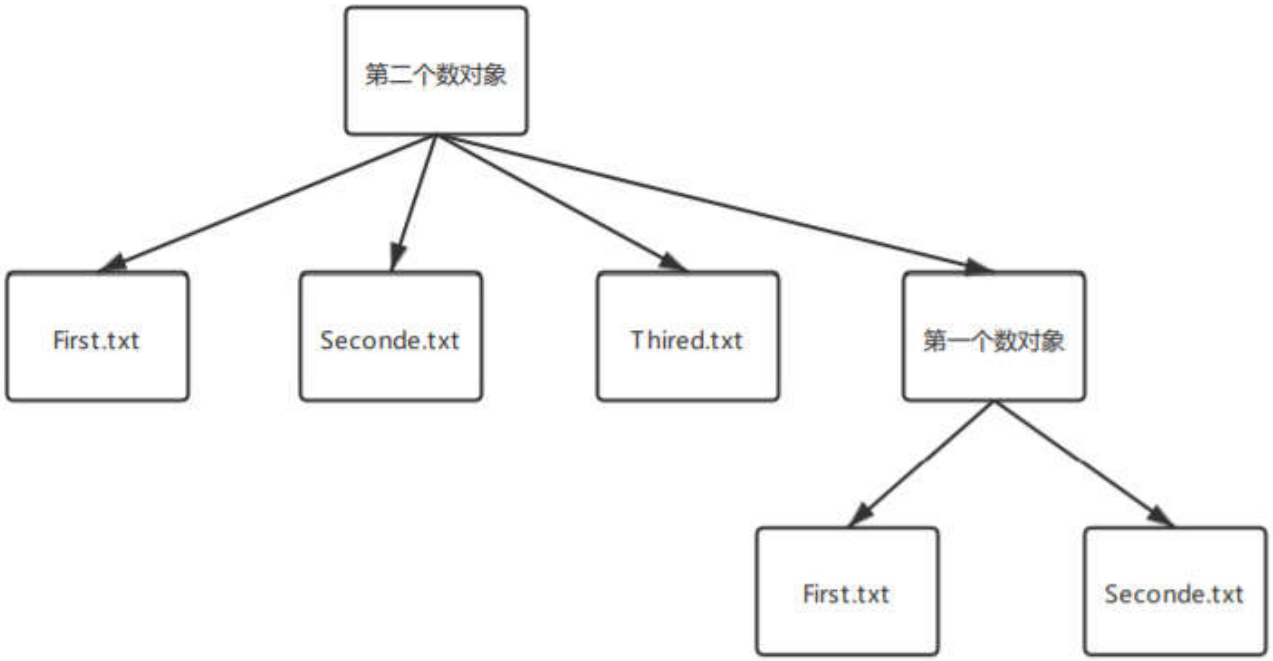
现在暂存区里面有两个文件了，分别是First.txt和Second.txt，假设这两个文件可以完成项目的第一个版本，那么我们应该生成一个树对象啦。通过 **git write tree**生成以一个树对象,下图可以看到相关信息。



当然我们也可以让一个数对象包含另一个数对象咯，通过git read-tree --prefix=bak 数对象的哈希值，读取版本库中的数对象放在暂存区。我们先创建一个文件将其写入缓存区，此文件和一个数对象组成一个新的数对象。



目前关系有点复杂，那就看看下面的图吧！！看完图肯定就可以理解咯。



树对象总结： 也就是说树对象是项目的版本，Git对象是文件的版本。一个项目的版本最少要有两个对象(一夫多妻？？？都是贤内助吗？)。

问题： 树对象虽然解决了版本的问题但是想用数对象那需要记住哈希值，这不就是扯淡吗。。并且也不知道是谁保存这个快照(可以理解为项目的版本),以及该快照的作用是啥子。这些问题就交给 **提交对象** 进行解决。

提交对象

可以通过commit-tree命令来创建提交对象，需要给定一个树对象的哈希值，以及该提交对象的父对象(如果没有的话，第一次将暂存区做快照就没有父对象（git commit-tree 数对象哈希值 -p 提交对象的父对象的哈希值））。一图胜千言，直接上图啦,图中可以看到我们在第一节笔记中所配置的信息。

```
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ echo 'First commit' | git commit-tree 3b7011a073d570a35ae9a07c8b399f583575e296
6d306822c766e63318a57d7623419284b7590773

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ git cat-file -p 6d306822c766e63318a57d7623419284b7590773
tree 3b7011a073d570a35ae9a07c8b399f583575e296
author Jc <Jc@example.com> 1622797339 +0800
committer Jc <Jc@example.com> 1622797339 +0800

First commit

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ echo 'Second Commit' | git commit-tree 9be2c2b66c42fd88fc0ad98bb2b4b32d6493809
c -p 6d306822c766e63318a57d7623419284b7590773
12a89339119bce670cb666f7186fdd7bf59f86d6

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ git cat-file -p 12a89339119bce670cb666f7186fdd7bf59f86d6
tree 9be2c2b66c42fd88fc0ad98bb2b4b32d6493809c
parent 6d306822c766e63318a57d7623419284b7590773
author Jc <Jc@example.com> 1622797495 +0800
committer Jc <Jc@example.com> 1622797495 +0800

Second Commit

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$
```

提交对象总结： 所以树对象是项目的版本，提交对象只是给树对象一些注释信息，但是提交对象是链表结构可以将其串联起来。所以上一节说Git所存放的是项目的快照而不是项目的增量，所以想回退版本那么只需要知道提交对象的哈希值就OK啦!

三、 结尾

至此呢Git的底层命令基本完毕，如果感觉有用可以点个赞的哦！我会持续更新，如果有错误还请指出来,感谢观众老爷的赏脸。

若想获得上述内容的PDF版本移步到GitHub下载。

地址： Git 学习笔记专区。

-----缱绻