

《Git对象模型》

在Git中有2个重要的概念：**对象集合**（object database）和**暂存区**（current directory cache）。

对象集合

在Git中，我们通常会面临文件的修改变更，协同开发中也需要对文件内容进行跟踪对比。那么就需要一种机制来跟踪和管理文件内容的变化。所谓对象集合，就是以文件内容为唯一标识来查询跟踪的对象集合。当然，我们不可能总是查询全部的文件内容，而是通过使用SHA1这种摘要算法来标识——也就是根据文件内容生成摘要信息，相同内容摘要信息相同，反之内容发生变化则摘要信息不同。我们Git产生的提交id其实就是一个摘要信息，是根据一次提交的新内容产生的唯一的信息摘要。

Git中，根据文件内容，使用SHA1算法计算的出该文件的哈希值。一个对象(文件，文件夹)总是有以下三个属性构成：

- `tag`: 表示对象类型
- `size`: 内容长度
- `\0`: ascii中的nul字符
- `对象内容`: 字节序列，长度为 `size`

将以上内容拼接 `<ascii tag without space> + <space> + <ascii decimal size> + <byte\0> + <binary data>`，在经过zlib压缩之后得到一个对象，压缩后在计算得出摘要值，便是该对象的名称。

对象类型

对象类型总共分三类：`blob`、`tree`、`changeset`。

blob

即普通文件类型，假设我们有一个叫 `hello.txt` 的文件，内容是 `hello world`。转换成 `blob` 之后的结构如下 `blob 11\0helloworld`，对应的文件名则是 `ede06c6ad21dd15a253f577717792731a320c404`。



tree

即目录类型，其中维护一个列表，列表中每一项都有可能是一个blob对象或tree对象。对于每一项都包含四个部分：

- `mode`: 权限和类型
- `file name`: 对象路径
- `\0`: ascii中的nul字符
- `sha1`: 对象的信息摘要用作的名称

将这四部分拼接一起 `<mode> + <space> + <file name> + <byte\0> + <sha1>`，得到一个tree完整对象。

```
1 | .
2 | └─ foo
3 |   └─ bar.txt
4 | └─ hello.txt
```

假设有如上目录结构，该目录对于的tree对象列表如下：

1	040000	tree	69ccc062670532506442789b4340dea98c73afd8	foo
2	100644	blob	ede06c6ad21dd15a253f577717792731a320c404	hello.txt

有两项内容，一项对应 foo 目录，这也是一个 tree 对象；另一项对应 hello.txt，这是一个 blob。而 foo 对应的 tree 对象的内容为：

1	100644	blob	0974d14c8dadb9ad9833dea721c63384b1448415	bar.txt
---	--------	------	--	---------

commit

一个commit对象信息如下，包含 tree、parent、author 和 commiter 字段。

```
tree 612ef8e50a094025e840ee951bce3036b030b888
parent 5cf777191a2da68ee90b0825162930e0f4a30032
author dustin dustin.wei@quectel.com Sat Jun 29 18:20:36 2022
committer dustin dustin.wei@quectel.com Sat Jun 29 18:20:36 2022

foo
```

commit对象对应一次提交，commit 对象首先通过 **tree 字段**指向一个 **tree 对象**，也就是一个**版本**。然后会有零个或多个parent，parent指向的是前一次的提交，这就是git检索历史提交的原理。而没有parent的提交，我们称之为root commit。

暂存区

前文讲到，一个 commit 对象对应一个 tree 对象，一个 tree 对象对应多个 tree 对象或者 blob 对象。这些 object 都是根据对象的**内容**命名的。我们修改了某个文件之后，需要对比改动的状态和内容。这就需要查看该文件被修改之前的版本。这需要找到当前 commit 对应的 tree，然后根据被修改文件的**路径**该 tree 做**深度优先搜索**才能找到对应的 blob。如果文件目录的层级很深，每一次 git diff 都会触发大量的磁盘操作，势必拖慢速度。

这个问题的根源是 tree 对象没有存储 blob 对象的**完整路径**，一个 blob 对象的路径信息被分散存储到了不同的 tree 对象中。例如前面提到的 `./foo/bar.txt`，路径 `foo` 被存储到了 `tree:612ef`，文件名 `bar.txt` 则被存储到了 `tree:69ccc` 中。解决思路也很简单，就是**加缓存**。

暂存区是一个文件，路径为 `.git/index`。Git 使用了 mmap 将文件映射到内存，可以像内存一样操作文件内容。文件的内容是一组所谓的 entry，每个 entry 对应一个 blob 对象，并且存储了 blob 对象的**完整路径**和其他一些状态信息。所有的 entry 是按照 blob 对象的文件路径**升序排列**的。这样，对于给定路径，Git 可以使用**二分查找**快速找到对应的 blob 对象。

所以，暂存区是 working directory 和 object database 纽带。