**Git Tutorial: Creating Blobs Tree and Commits**

[**RAJESH KUMAR**](https://www.devopsschool.com/blog/author/rajeshkumar/)**JULY 24, 2022**[**LEAVE A COMMENT**](https://www.devopsschool.com/blog/git-tutorial-creating-blobs-tree-and-commits/#respond)

Creating Blobs  
Creating a blob in your Git repository and getting a SHA back is pretty easy. The git hash-object command is all you’ll need. To create a blob object from an existing file, just run it with the ‘-w’ option (which tells it to write the blob, not just compute the SHA).

$ git hash-object -w myfile.txt  
6ff87c4664981e4397625791c8ea3bbb5f2279a3

$ git hash-object -w myfile2.txt  
3bb0e8592a41ae3185ee32266c860714980dbed7  
The STDOUT output of the command will the the SHA of the blob that was created.

Creating Trees  
Now lets say you want to create a tree from your new objects. The git mktree command makes it pretty simple to generate new tree objects from git ls-tree formatted output. For example, if you write the following to a file named ‘/tmp/tree.txt’ :

100644 blob 6ff87c4664981e4397625791c8ea3bbb5f2279a3 file1  
100644 blob 3bb0e8592a41ae3185ee32266c860714980dbed7 file2  
and then piped that through the git mktree command, Git will write a new tree to the object database and give you back the new sha of that tree.

$ cat /tmp/tree.txt | git mk-tree  
f66a66ab6a7bfe86d52a66516ace212efa00fe1f  
Then, we can take that and make it a subdirectory of yet another tree, and so on. If we wanted to create a new tree with that one as a subtree, we just create a new file (/tmp/newtree.txt) with our new SHA as a tree in it:

100644 blob 6ff87c4664981e4397625791c8ea3bbb5f2279a3 file1-copy  
040000 tree f66a66ab6a7bfe86d52a66516ace212efa00fe1f our\_files  
and then use git mk-tree again:

$ cat /tmp/newtree.txt | git mk-tree  
5bac6559179bd543a024d6d187692343e2d8ae83  
And we now have an artificial directory structure in Git that looks like this:

|– file1-copy  
-- our\_files |-- file1 — file2

1 directory, 3 files  
without that structure ever having actually existed on disk. Plus, we have a SHA (5bac6559) that points to it.

Rearranging Trees  
We can also do tree manipulation by combining trees into new structures using the index file. As a simple example, let’s take the tree we just created and make a new tree that has two copies of our 5bac6559 tree in it using a temporary index file. (You can do this by resetting the GIT\_INDEX\_FILE environment variable or on the command line)

First, we read the tree into our index file under a new prefix using the git read-tree command, and then write the index contents as a tree using the git write-tree command:

$ export GIT\_INDEX\_FILE=/tmp/index  
$ git read-tree –prefix=copy1/ 5bac6559  
$ git read-tree –prefix=copy2/ 5bac6559  
$ git write-tree  
bb2fa6de7625322322382215d9ea78cfe76508c1

$>git ls-tree bb2fa  
040000 tree 5bac6559179bd543a024d6d187692343e2d8ae83 copy1  
040000 tree 5bac6559179bd543a024d6d187692343e2d8ae83 copy2  
So now we can see that we’ve created a new tree just from index manipulation. You can also do interesting merge operations and such in a temporary index this way – see the git read-tree docs for more information.

Creating Commits  
Now that we have a tree SHA, we can create a commit object that points to it. We can do this using the git commit-tree command. Most of the data that goes into the commit has to be set as environment variables, so you’ll want to set the following:

GIT\_AUTHOR\_NAME  
GIT\_AUTHOR\_EMAIL  
GIT\_AUTHOR\_DATE  
GIT\_COMMITTER\_NAME  
GIT\_COMMITTER\_EMAIL  
GIT\_COMMITTER\_DATE  
Then you will need to write your commit message to a file or somehow pipe it into the command through STDIN. Then, you can create your commit object based on the tree sha we have.

$ git commit-tree bb2fa < /tmp/message  
a5f85ba5875917319471dfd98dfc636c1dc65650  
If you want to specify one or more parent commits, simply add the shas on the command line with a ‘-p’ option before each. The SHA of the new commit object will be returned via STDOUT.