Git Guide

Git Object Types

```
- Blob -> All types of files are stored as Blob
- Tree -> Information about Directories
- Commit -> Allows to store different versions of files of project
- Annotated Tag -> Persistant text pointer to a specifi commit
```

Git Low Level Commands:

- git hash-object
- git cat-file
- git mktree

Blob

Creating blobs from text:

```
$echo "Hello, Git" | git hash-object --stdin -w   (-w to write object to file
system)
$b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
```

Creating blobs from files:

To write file:

```
$ git hash-object filename -w
```

This will create fiel object inside .git-> objects -> b7 (starting 2 characters of Hashcode)

Git cat-file:

Provide content or type and size information for repository objects Flags: -p to print contects, -s to print size and -t for type of object

e.g:

```
git cat-file -p b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5eob
```

Git doesn't store object based on their name. Instead it uses it's hash calculated as:

> ***Git Object Name => Git Object Hash = Content + Object Type + Object Length = Hash***

Example:

```
Hello, Git => Blob 11\0Hello, Git
```

```
$ echo "blob 11\0Hello, Git" | shasum
5fa6d9c54f3d2c4b6bb2b0e6ed71059a84fa7463
```

*To tell Git that **\0** should be interpreted as null escape character (if you are getting different sha) add **-e***

```
$ echo -e "blob 11\0Hello, Git" | shasum
b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
```

It is Same as generated by git hash-object

```
$ echo -e "blob 30\0Second file in Git repository" | shasum
4400aae52a27341314f423095846b1f215a7cf08
```

Using Git hash-object:

$ echo "Second file in Git repository" | git hash-object --stdin 4400aae52a27341314f423095846b1f215a7cf08

Tree

A "*tree*" in Git is an object (a file, really) which contains a list of pointers to blobs or other trees. Each line in the tree object's file contains:
mode/permissions, the type, the hash, and the file name
e.g

```
  $ git cat-file -p ef34a15
  100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad    readme.md
```

Git Permission types:

Binary	Octal	Meaning
0100000000000000	(040000)	Directory
1000000110100100	(100644)	Regular non-executable file

Binary	Octal	Meaning
1000000110110100	(100664)	Regular non-executable group-writeable file
1000000111101101	(100755)	Regular executable file
1010000000000000	(120000)	Symbolic link
1110000000000000	(160000)	Gitlink

Creating a Git Tree Object:

We have two files now in Git objects. We can view them as:

```
$ find objects/ -type f
objects/44/00aae52a27341314f423095846b1f215a7cf08
objects/b7/aec520dec0a7516c18eb4c68b64ae1eb9b5a5e
```

We will create file for them and from that file we will make Git Tree Object.

Create a file with following contents:

```
100644 blob b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e    file1.txt
100644 blob 4400aae52a27341314f423095846b1f215a7cf08    file2.txt
```

Important: Only use tab between hash and file name space will not work. We will name it temp-tree.txt.

To create git Tree Object:

```
$ cat /d/temp-tree.txt | git mktree
3b95df0ac6365c72e9b0ff6c449645c87e6e1159

$ git cat-file -t 3b95
tree
```

To Move Files from Repository into Stagging Area:

```
$ git read-tree 3b95
```

To read contents of stagging area:

```
$ git ls-files
file1.txt
file2.txt
```

```
$ git ls-files -s
100644 b7aec520dec0a7516c18eb4c68b64ae1eb9b5a5e 0       file1.txt
100644 4400aae52a27341314f423095846b1f215a7cf08 0       file2.txt
```

0 represented after hash code shows file in stagging area as exactly same as one in repository and there are no changes.

To Move File to Working Area From Stagging:

```
$ git checkout-index -a
```

***-a** flag is used to checkout all files*