1. Create a new directory which will be the working directory for a new git project. Give it a name of your choice. This can be done on Linux using **mkdir <name>.** Enter the directory (e.g. using **cd <name>**) and initialize a git project there using **git init**.

2. Verify that git has initialized a new, empty repository inside the working directory. This can be done by checking that git created a .git directory. Use **ls** to accomplish this. Remember that files and directories which start with a dot are hidden by default. In order to list them, you have to use ls with the **-a** flag. Which subdirectories are found inside the .git directory?

3. Choose one of your favorite programming languages, like JavaScript/Node.js, Python, Bash, or anything else. In the root (top level) of the working directory, create a source file which contains a function called sayHi or similar which does nothing, and which invokes said function. For example:

JavaScript	**Bash**	**Python**

`#!/usr/bin/node`
 	
`function sayHi() {`
`}`
 	
`sayHi();`
	`#!/bin/bash`
	
	`say_hi() {`
	`}`
	
	`say_hi`
		`#!/usr/bin/python`
		
		`def say_hi():`
		`  pass`
		
		`say_hi()`

You can use your favorite graphical text editor for this (e.g. Atom, Sublime, Visual Studio Code or similar), or you can do it from the terminal using a terminal-only text editor like **nano**. Remember to save the file inside the working directory of the Git project, and give it a name appropriate to the programming language you chose (e.g. `index.js` for JavaScript, `index.py` for Python or `hi` for Bash).

4. Find git's hash of the file by running **git hash-object <filename>.** What is the hash?

5. Compare the hash in 4 to the result of running **sha1sum <filename>**. What is this hash? Why is this hash different from the one in 4, although git uses SHA-1 to hash its objects?

6. Add the file you created in 3 to Git's object store by adding it to the index (**git add <filename>**)

7. Locate the file inside Git's object storage using your result from 4. What is the file system path to the file which contains the blob corresponding to the file? Verify that you can look the file up by hash by printing it using **git cat-file -p <object hash>**.

8. Write the tree-object corresponding to the current index by running **git write-tree**. What is the tree's hash? Can you locate it in the object store? List its content using **git cat-file -p <tree hash>**. What does the tree contain (e.g. which blobs does it reference)?

9. Create a sub-directory in the project's root directory called "old". Copy the source code file there using **cp**. You should now have a directory structure which looks similar to

```
.
├── index.js
└── old
    └── index.js
```

Note: your source file and its copy might be named differently from the above (index.js). Modify the original source file so that it fulfills its function, e.g.

JavaScript	**Bash**	**Python**

`#!/usr/bin/node`
 	
`function sayHi() {`
`  console.log(“Hi!”);`
`}`
 	
`sayHi();`
	`#!/bin/bash`
	
	`say_hi() {`
	`  echo “Hi!”`
	`}`
	
	`say_hi`
		`#!/usr/bin/python`
		
		`def say_hi():`
		`  print(“Hi”)`
		
		`say_hi()`

Add both the updated source file and its copy to the index using **git add**.

- What is the hash of the new and changed source file?
- Find it in the object store, i.e. on disk inside the .git directory.
- Save the tree of the current index using **git write-tree**.
- Inspect this tree using **git cat-file -p <tree hash>**.
- This tree now references another tree by its hash. Do you recognize this tree/hash from execise 8? Explain what git has done.

10. List the files inside .git/objects. Run Git’s garbage collection by issuing **git gc.** Note: this is something that you would ordinarily never do manually, and is a task git itself invokes at regular intervals. After garbage collection, what has happened to .git/objects? Where are the blobs stored now?