# Design for ngc(not-a-git-clone)

### **Overview -**

This project aims to create a version-control tool, ngc. This tool should be able to keep track of various versions of the user’s software project, retrieve the specified version as when necessary, show the differences of modified files and have other relevant functions.

### **Design considerations -**

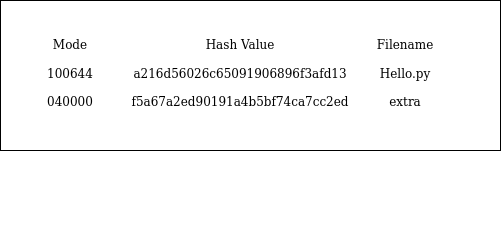
* Assumptions : The user is aware of how a version control works and of its basic functions
* System environment: The project is designed to work on most unix-like systems as well as Windows

### **Architecture -**

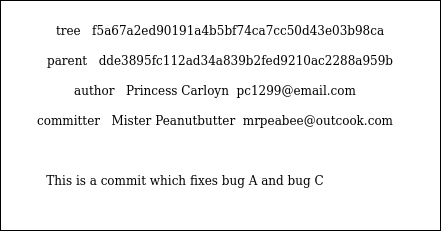
Ngc will store files and meta-data in a hidden subdirectory called ngc itself. All the committed files and the meta-data will be stored as compressed files.

There will be 3 main object files:

* Blob object -
  + Each file will be stored as a blob object
  + It will be present in the directory in a compressed form using any appropriate compression technique(preferably zlib)
  + Its format: HEADER + CONTENT
    - HEADER = “blob{space}{content.length}{null-character}”
    - CONTENT = contents of the files
  + For example consider a file with content “Wubba lubba dub dub!” then the blob content will be: “blob 20\0Wubba lubba dub dub!”
* Tree object -
  + Tree will represent the structure of the repository
  + It will list the files and other subdirectories in a table format
  + The headings of the table will consist of -
    - File mode
    - Hash value - value generated after running through any appropriate hash function(preferably SHA1)
    - File name
  + The tree object can reference another tree object as well
  + For example consider a file “Hello.py” with hash value “bc84202464fcf6eb0c6dab88bccc03c633959df1” and a subdirectory “extra” which is represented by a tree with hash “27a0818f3db1fb6a5c961e5e34f6db234d91bc6e” :



* Commit object -
  + It will contain hash references to its tree object and parent commit.
  + It will also contain details regarding the author of the repository and the committer who executed the commit
  + A message regarding the commit will be appended at the end
  + Example layout:



The objects will be stored in another subdirectory called “objects” with their filename as the hash value of their file contents.

### **Working -**

Below are various ngc commands and an explanation of how they will work.

#### ngc init -

* + It will create a hidden subdirectory “ngc” which will contain the repository’s metadata and object files
  + Execution flow -

*create hidden “ngc” subdirectory -> create empty “HEAD” file inside “ngc” -> create empty “objects” subdirectory inside “ngc”*

#### ngc commit -

* + This command will create a snapshot of the repository using the three main objects, ie, tree object, blob object and commit object
  + A message will be required from the committer to describe the commit
  + Execution flow -

*store files as blobs -> write trees -> write commit objects -> update HEAD file with the hash value of the current branch commit*

#### ngc status -

* + It will show a list of modified files and suggest commands accordingly
  + Checking for modification can be done by comparing modified time stat of the file with its blob object stat
  + Execution flow -

*Iterate through all files and subdirectories recursively and check for modification -> if file changed print its name -> at the end of iteration suggest command to commit*

#### ngc diff -

* + It will print the differences of the modified files
  + Differences can be shown by assigning marker legends to the modified file and its previous version and iteratively showing each modified line
  + Execution flow -

*Iterate through all files -> check differences in lines -> if differences exist print the modified line and previous version line*

#### ngc reset -

* + It will discard any changes made after last commit and re-create the repository as was in the last commit
  + Execution flow -

*Check for modified files and replace them the previous commit version -> check for new files created and delete them -> check for deleted files and replace those files from previous commit*

#### ngc log <commit-hash> -

* + Show that commit’s message as well as meta-data
  + Previous parent commits are also shown
  + Execution flow -

*Get current commit’s reference from HEAD file -> print metadata of that commit -> if parent exists print its metadata as well -> continue till no parent*

#### ngc checkout <commit-hash> -

* + Load the repository as was in the specified commit’s snapshot
  + This can delete any non-committed files so a warning will be necessary in that case
  + Execution flow -

*Delete all the files and subdirectories in the repository except the ngc subdirectory -> search for the given hash value to find the commit object in the objects subdirectory -> use the commit object to find the tree hash value -> use the tree object to load all the files and subdirectories for that commit*

### **Internals -**

Certain small but essential concepts to complete the project.

* Branching -
  + ngc initializes a repository with a branch named “master”
  + The details of which branch the repository is currently on resides in a file called “HEAD” which will be located in the hidden subdirectory “ngc”
  + The “HEAD” file will contain the hash value of the commit object which describes the commit in the current branch

### **Code Structure -**

#### ngc class -

* Constants
  + Repository path
* Methods
  + Init command
  + Status command
  + Diff command
  + Commit command
  + Reset command
  + Log command

#### Object class -

* Constants
  + Compression method
  + Hashing function
* Methods
  + Extract object
  + Compress object

#### Tree class extends Object class-

* Constants
  + File mode dictionary
* Methods
  + Read listing
  + Write listing
  + Delete listing
  + Create tree

#### Blob class extends Object class -

* Methods
  + Read header
  + Write header
  + Read content
  + Write content
  + Create blob

#### Commit class extends Object class -

* Methods
  + Read metadata
  + Write metadata
  + Create commit object

### **Unittests -**

#### Integration Tests for commands -

* Checking working of init command
  + After the command is executed, an empty HEAD file and an empty subdirectory “objects” should be created inside a hidden directory “ngc”
* Checking working of commit command
  + Consider a directory with one file filled with random content and one subdirectory containing another file with random content
  + Execute init and commit commands
  + Decompress blob files and match the content with their respective files
* Checking working of checkout command
  + Create two commits from a sample repository with evident changes
  + Match file and subdirectories for the respective commit when using the checkout command for the two commits
* Checking working of status and diff command
  + Create a commit on a sample repository
  + Make changes
  + Check if the status command is listing all the changes
  + Tally if the diff command is showing all the changed lines
* Checking working of log
  + Create a sample repository
  + Initialize ngc and create a commit
  + Create a series of commit and changes
  + Execute log and tally the commits and their metadata

#### Unit tests for objects -

* Test tree class methods
  + Create tree object
  + Execute read, write and delete methods and check their functioning
* Test blob class methods
  + Create a mock blob object
  + Write header and content with a sample file
  + Analyse and test the blob object
* Test commit class methods
  + Create a mock commit object from a sample repository
  + Analyse and check the resultant object

MORE

#### Unit tests for algorithms -

* Diff
* Compression/decompression
* Hashing
* Merge