2 TD2/5: Git Filesystem & Commits

Exercise 1 Configure Git

Using only command-line in your Linux shell,

1. Check that Git is installed on your environment.
2. Configure your name and e-mail globally.
3. Check that Git has correctly recorded these two pieces of information.
   hint: All Git commands have a ***-h*** *flag to display the corresponding help. Look there for the option of the* ***git config*** *command that lists all Git configuration.*

Exercise 2 Hashing

Using only command-line in your Linux shell and without creating a Git repository for the moment,

1. Create a root directory.
2. Create a text file inside whose content is "Hello World".
3. What is the size of the file ?
4. Display the file content on the screen.
5. Compute the SHA-1 hash of the file content.
   hint: You can use the GNU core utilities ***sha1sum*** *command.*
6. What hash would Git compute on this file ?
   hint: You can use the ***git hash-object*** *plumbing command (no need to create a Git repository for the moment).*

   They are different aren't they ?
   Actually Git prepends 2 properties to the file content before hashing, compressing and saving it:

   (a) the Git object type followed by a space character
   (b) the file size followed by a null character (\0)
7. Create a second file whose content is what Git would really consider when saving your first file.
   hint: The ***echo*** *command has a* ***-e*** *flag to interpret backslash escapes.*
8. Compute the SHA-1 hash of this second file and check it is equal to the Git hash of your first file.

Exercise 3 Compressing

Using only command-line in your Linux shell,

1. Create an empty Git repository in your root directory (if you have accidentally already created a Git repository in your root directory, delete it before).
2. Check that Git is aware of your 2 files but does not track them yet.
3. Check that no object is stored yet in the **objects** subdirectory of you Git repository.
4. Create a directory inside the **objects** subdirectory of you Git repository, whose name is the first two characters of the SHA-1 hash computed in the previous exercise.
5. Install the QPDF free command-line program.
   Part of this program is the **zlib-flate** command that compress and uncompress files using the **deflate** algorithm.
6. Create a file inside the directory that you have just created, whose content is the deflate compression (level 1) of your second file and whose name is the last 38 characters of the SHA-1 hash computed in the previous exercise.
7. Check that Git successfully considers this file as one of its inner object.
   *hint: You can use the different flags of the **git cat-file** plumbing command.*
8. Backup your Git repository and create a new one.
9. Stage your first file in Git and check that its name and content are identical to yours.
10. Create another text file whose content is 100 lines of "Hello Mister i" (i varying from 1 to 100).
11. Stage this new file in Git and check that the compression ratio on this second example is better than on the first one.

Exercise 4 Local and remote backup

Using only command-line in your Linux shell,

1. Commit the two files that must be in your staging area.
2. Check that more objects have been saved in your Git repository.
3. Check that a commit exists in your Git repository.
   *hint: You can use the **git log** porcelain command.*

4. Walk Git inner directed acyclic graph (DAG) from your commit to your two file contents.
   *hint: You can use several times the **git cat-file** plumbing command.*
   *hint: If you are a teaser you can try combining **awk**, **print**, **xargs** and **head** commands.*

5. Suppress your two files.

6. Use Git to restore your files.

7. Backup you Git repository elsewhere
   (pretending a copy exists on another colleague's computer or on a remote server).

8. Suppress your root directory, create a new empty one and use your backup to restore everything.

Exercise 5 Diff

Using only command-line in your Linux shell,

1. Add one line to your first "Hello World" file.

2. Delete the last 10 lines of your second "Hello Mister i" file. *hint: You can use the **sed** Unix command.*

3. Check that Git is aware of your changes

4. Display the changes in your root directory since the last commit. You should see your two changes. *hint: You can use the **git diff** Unix command.*

5. Stage your first file only.

6. Check that Git has correctly staged only one file.

7. Display again the changes in your root directory since the last commit. Which file changes do you see ?

8. Display the other file changes.

Exercise 6 Undo

Using only command-line in your Linux shell,

1. Unstage your first file

2. Commit your two file changes directly, without staging them.

3. Check your commit log history. Do you see your new commit ?

4. Without affecting your Git repository, set your root directory state as of the snapshot of your first commit.
5. Check your commit log history. You do not see all commits, do you ? How can you see all of them ?
6. Return to the snapshot of your your last commit.
7. Undo your second commit by adding a new commit that reverts it.
8. Check the content of your root directory. Have your previous changes disappeard ?
9. Check your commit log history. Do you see your revert commit ?
10. Remove the last 2 commits from the history.
11. Check the content of your root directory. Have your previous changes disappeard ?
12. Check your commit log history. Have you lost the last 2 commits ?

Exercise 7 Aliases

Using only command-line in your Linux shell,

1. Create a "**s**" alias for the **git status** command.
2. Create a "**co**" alias for the **git checkout** command.
3. Create a "**b**" alias for the **git branch** command.
4. Create a "**ci**" alias for the **git commit** command.
5. Create a "**dog**" alias for the **git log –all –decorate –oneline –graph** command.
6. Create a "**dag**" alias for the **git log –all –decorate –graph** command.
7. Create a "**list**" alias for the **git diff-tree –no-commit-id –name-only -r** command.
8. Create a "**unstage**" alias for the **git reset HEAD** – command.
9. Create a "**last**" alias for the **git log -1 HEAD** command.