# TD6/9: Git Filesystem & Commits

All these exercises should be done using only the command-line in your Linux shell

## Exercise 1: Configure Git

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: Basic workflow with a single file

1. Create a git repository
2. Check that git has correctly initiliazed a repository by displaying the files wihtin your current folder
3. Check the current git status
4. Create a text file named "readme.md" whose content is "# Test repository"
5. Check the current git status
6. Stage the file
7. Check the current git status
8. Commit the file
9. Check the current git status
10. Check the git logs
11. Which informations are displayed ?

## Exercise 3: Basic workflow with multiple files treated separately

1. Create two empty python files named "main.py" and "functions.py"
2. Check the current git status
3. Stage only the file "main.py"
4. Check the current git status
5. Commit the file with an appropriate message
6. Check the current git status
7. Now stage and commit the file "functions.py"
8. Check the current git status
9. Check the git logs

## Exercise 4: Basic workflow with multiple files treated all at once

1. Create three empty files named “requirements.txt”, “.gitignore” and “.private”
2. Check the current git status
3. Stage all the files at once
4. Check the current git status
5. Commit the current staged files
6. Check the current git status
7. Check the git logs where each log is displayed on a single line

## Exercise 5: Private files

Files can be private in two ways :
— being a temporary file (like an open Excel would produce or Python Jupyter Notebook would produce). This would happen to anyone using your project.
— being a personal file (personal notes, etc.)

1. Emulate a temporary empty file by creating a file named “temp.ipynb”
2. Check the current git status
3. Add an instruction to .gitignore to prevent git from tracking this temp file
4. Check the current git status
5. Create other temporary files named “temp.aux” and “temp.log”
6. Check the current git status
7. Change your instruction in .gitignore to prevent git from tracking any file which name starts with “temp”
8. Check the current git status
9. Now let’s consider your personal notes will be added to the “.private” folder. Use the “exclude” git file to prevent git from tracking this “.private” folder

## Exercise 6: Difference between versions

1. Add a online description of your repository in the “readme.md” file
2. Stage your “readme.md” file
3. Display the **changes** in your root directory since the last commit (not just the current status)
4. Commit your change
5. Display the changes since the last commit

6. Display again the changes in your root directory since the last commit
7. Change some words in the description of the “readme.md”
8. Display the changes since the last commit

## Exercise 7: Undo

1. Suppress all your files.
2. Use Git to restore your files.
3. Backup your Git repository elsewhere
   (pretending a copy exists on another colleague's computer or on a remote server).
4. Suppress your root directory, create a new empty one and use your backup to restore everything.
5. Unstage your first file
6. Commit your two file changes directly, without staging them.
7. Check your commit log history. Do you see your new commit ?
8. Without affecting your Git repository, set your root directory state as of the snapshot of your first commit.
9. Check your commit log history. You do not see all commits, do you ? How can you see all of them ?
10. Return to the snapshot of your your last commit.
11. Undo your second commit by adding a new commit that reverts it.
12. Check the content of your root directory. Have your previous changes disappeard ?
13. Check your commit log history. Do you see your revert commit ?
14. Remove the last 2 commits from the history.
15. Check the content of your root directory. Have your previous changes disappeard ?
16. Check your commit log history. Have you lost the last 2 commits ?

## Exercise 8: Aliases

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.

## Exercise 9: Hashing

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 10: Compressing

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 your Git repository.
4. Create a directory inside the **objects** subdirectory of your 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.