TD3

Exercie 1

git --version

git config --global user.name "Thomas"  
git config --global user.email "[thomasni.2001@gmail.com](mailto:thomasni.2001@gmail.com)

git config --list

user.name=Thomas  
user.email=[thomasni.2001@gmail.com](mailto:thomasni.2001@gmail.com)

Exercise 2: Basic workflow with a single file

Create a git repository

mkdir test-repo  
cd test-repo  
git init

2.Check that git has correctly initialized a repository by displaying the files within your current folder  
ls -la

3.Check the current git status

git status

4Create a text file named "readme.md" whose content is "# Test repository"  
echo "# Test repository" > readme.md

5.Check the current git status

git status

6 check the file

git add readme.md

Check the current git status

git status

8 commit the file

git commit -m "Initial commit"

9.check the current git status  
git status

10Check the git logs  
git log

11 Which information is displayed?  
The git logs display the commit hash, author, date, and commit message. In this case, there should be one commit with the message "Initial commit".

Exercise 3: Basic workflow with multiple files treated separately

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

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

**Solution:**

1. Create three empty files named “requirements.txt”, “.gitignore” and “.private”

$ touch requirements.txt .gitignore .private

1. Check the current git status

$ git status

1. Stage all the files at once

$ git add .

1. Check the current git status

$ git status

1. Commit the current staged files

$ git commit -m "Add requirements.txt, .gitignore, and .private"

1. Check the current git status

$ git status

1. Check the git logs where each log is displayed on a single line.

$ git log --oneline

Exercise 5: Private files  
1Emulate a temporary empty file by creating a file named “temp.ipynb”

$ touch temp.ipynb

2Check the current git status  
$ git status

3 Add an instruction to .gitignore to prevent git from tracking this temp file  
$ echo "temp.ipynb" >> .gitignore

4.Check the current git status  
$ git status

5Create other temporary files named “temp.aux” and “temp.log”  
$ touch temp.aux temp.log

6.Check the current git status  
$ git status

7Change your instruction in .gitignore to prevent git from tracking any file which name starts with “temp”  
$ echo "temp\*" >> .gitignore

8.Check the current git status  
$ git status

9Now 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

$ echo ".private/" >> .git/info/exclude

Exercise 6: Difference between versions

1. Add a online description of your repository in the “readme.md” file  
   Open the readme.md file and add a description of your repository
2. Stage your “readme.md” file  
   $ git add readme.md
3. Display the changes in your root directory since the last commit (not just the current status)  
   $ git diff HEAD~
4. Commit your change  
   $ git commit -m "Add description to readme.md"
5. Display the changes since the last commit  
   $ git log --oneline

Find the commit hash for the previous commit (the one before "Add description to readme.md")  
For example, if the hash is "abcd1234":  
$ git diff abcd1234 HEAD

1. Display again the changes in your root directory since the last commit  
   $ git diff HEAD~
2. Change some words in the description of the “readme.md”  
   Open the readme.md file and make changes to the description
3. Display the changes since the last commit  
   $ git diff HEAD

Exercise 7: Undo

Suppress all your files.  
$ rm -rf \*

Use Git to restore your files.  
$ git checkout .

Backup your Git repository elsewhere (pretending a copy exists on another colleague’s computer or on a remote server).  
Copy the entire project directory to the desired backup location.

Suppress your root directory, create a new empty one and use your backup to restore everything.  
$ rm -rf \*  
$ mkdir my\_project  
$ cd my\_project  
$ cp -r /path/to/backup/\* .

Unstage your first file.  
$ git reset HEAD

Commit your two file changes directly, without staging them.  
$ git commit -a -m "Commit message for the changes"

Check your commit log history. Do you see your new commit?  
$ git log

Yes, the new commit should be visible in the log history.

Without affecting your Git repository, set your root directory state as of the snapshot of your first commit.  
$ git checkout .

Check your commit log history. You do not see all commits, do you? How can you see all of them?  
No, we do not see all the commits. To see all commits, we can use the following command:

$ git log --all

Return to the snapshot of your last commit.  
$ git checkout

Undo your second commit by adding a new commit that reverts it.  
$ git revert

Check the content of your root directory. Have your previous changes disappeared?  
No, the changes from the previous commits are still present.

Check your commit log history. Do you see your revert commit?  
$ git log

Yes, the revert commit should be visible in the log history.

Remove the last 2 commits from the history.  
$ git reset HEAD~2  
$ git push --force

Note: Be careful when using git reset with --force, as it can permanently remove commits.

Check the content of your root directory. Have your previous changes disappeared?  
Yes, the previous changes have disappeared as the last 2 commits have been removed.

Check your commit log history. Have you lost the last 2 commits?  
Yes, the last 2 commits are no longer visible in the log history.

Exercice 8  
Create a "s" alias for the git status command.  
$ git config --global alias.s "status"

Create a "co" alias for the git checkout command.  
$ git config --global alias.co "checkout"

Create a "b" alias for the git branch command.  
$ git config --global alias.b "branch"

Create a "ci" alias for the git commit command.  
$ git config --global alias.ci "commit"

Create a "dog" alias for the git log --all --decorate --oneline --graph command.  
$ git config --global alias.dog "log --all --decorate --oneline --graph"

Create a "dag" alias for the git log --all --decorate --graph command.  
$ git config --global alias.dag "log --all --decorate --graph"

Create a "list" alias for the git diff-tree --no-commit-id --name-only -r command.  
$ git config --global alias.list "diff-tree --no-commit-id --name-only -r"

Create a "unstage" alias for the git reset HEAD -- command.  
$ git config --global alias.unstage "reset HEAD --"

Create a "last" alias for the git log -1 HEAD command.  
$ git config --global alias.last "log -1 HEAD"

Exercice 9

**Create a root directory**

$ mkdir hashing  
$ cd hashing

**Create a text file with content "Hello World"**

$ echo "Hello World" > file.txt

**Check the size of the file**

$ ls -l file.txt  
-rw-r--r-- 1 user group 12 Mar 8 16:28 file.txt

**Display the file content**

$ cat file.txt  
Hello World

**Compute the SHA-1 hash of the file content**

$ sha1sum file.txt  
f572d396fae9206628714fb2ce00f72e94f2258f file.txt

**Compute the Git hash of the file content**

$ git hash-object file.txt  
ce013625030ba8dba906f756967f9e9ca394464a

**Create a second file with content that Git would use**

$ echo -e "blob (��−�<����.���)\0(cat file.txt)" > git\_file.txt

**Compute the SHA-1 hash of the Git file content**

$ sha1sum git\_file.txt  
ce013625030ba8dba906f756967f9e9ca394464a git\_file.txt

**Install QPDF**

$ sudo apt install qpdf

**Create a directory for the Git object**

$ mkdir .git/objects/f5

**Compress the second file using QPDF and save it in the Git object directory**

$ qpdf --compress-level=1 --object-streams=disable --file-inline-streams=disable git\_file.txt .git/objects/f5/2258fc7e807e0b0f7a97338b4c40b905d8b29e

**Check that Git recognizes the object**

$ git cat-file -p f52258fc7e807e0b0f7a97338b4c40b905d8b29e  
Hello World

Exercice 10

**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).**

cd ..  
rm -rf root\_directory/.git  
git init

**2. Check that Git is aware of your 2 files but does not track them yet.**

git status

**3. Check that no object is stored yet in the objects subdirectory of your Git repository.**

ls .git/objects

**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.**

hash=$(sha1sum git\_hello.txt | awk '{ print $1 }')  
mkdir -p .git/objects/${hash:0:2}

**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.**

sudo apt-get install qpdf

**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.**

zlib-flate -1 < git\_hello.txt > .git/objects/${hash:0:2}/${hash:2}  
ls -l .git/objects/${hash:0:2}

**7. Check that Git successfully considers this file as one of its inner objects.**

git cat-file -p ${hash}

**9. Stage your first file in Git and check that its name and content are identical to yours.**

git add hello.txt  
git cat-file -p HEAD:hello.txt

**10. Create another text file whose content is 100 lines of “Hello Mister i” (i varying from 1 to 100).**

for i in {1..100}; do echo "Hello Mister $i" >> hello$i.txt; done

**11. Stage this new file in Git and check that the compression ratio on this second example is better than on the first one.**

git add .  
git cat-file -s HEAD