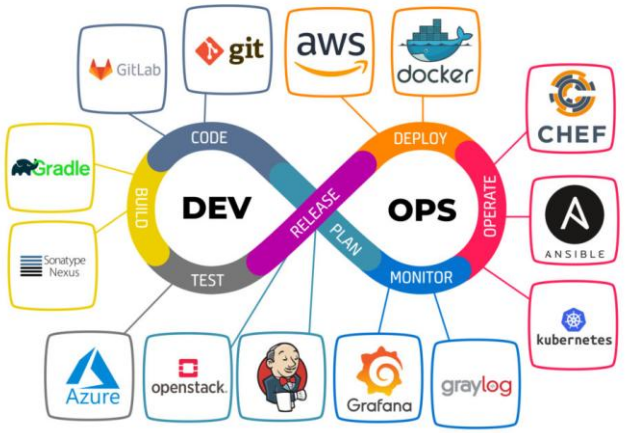
TEK-UP
Ecole Supérieure Privée
Technologie & Ingénierie



Git & GitHub

CORRECTION DES TPS (4, 5 ET 6)

GIT MERGE

TP4 Git branch

1. Tapez la commande « git branch », Quel est le résultat obtenu ?
2. Créez une nouvelle branche « Branche1 » avec la commande « git branch " nom-dela-branche" ».
3. Vérifiez la création de la nouvelle branche.

```
user@DESKTOP-RRVCC4I MINGW64 ~
$ mkdir TP3

user@DESKTOP-RRVCC4I MINGW64 ~
$ cd TP3

user@DESKTOP-RRVCC4I MINGW64 ~/TP3
$ git init
Initialized empty Git repository in C:/Users/user/TP3/.git/

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ touch file1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git add .

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git commit -m "file1 commit"
[master (root-commit) b74e4a8] file1 commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch
* master
```

TP4 Git branch

4. Switchez vers la branche créée avec la commande « git checkout " nom-de-la-branche" ».

5. Vérifiez l'emplacement actuel du pointeur.

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch
* master

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch branch1

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch
  branch1
* master

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git checkout branch1
Switched to branch 'branch1'

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git branch
* branch1
  master
```

TP4 Git branch

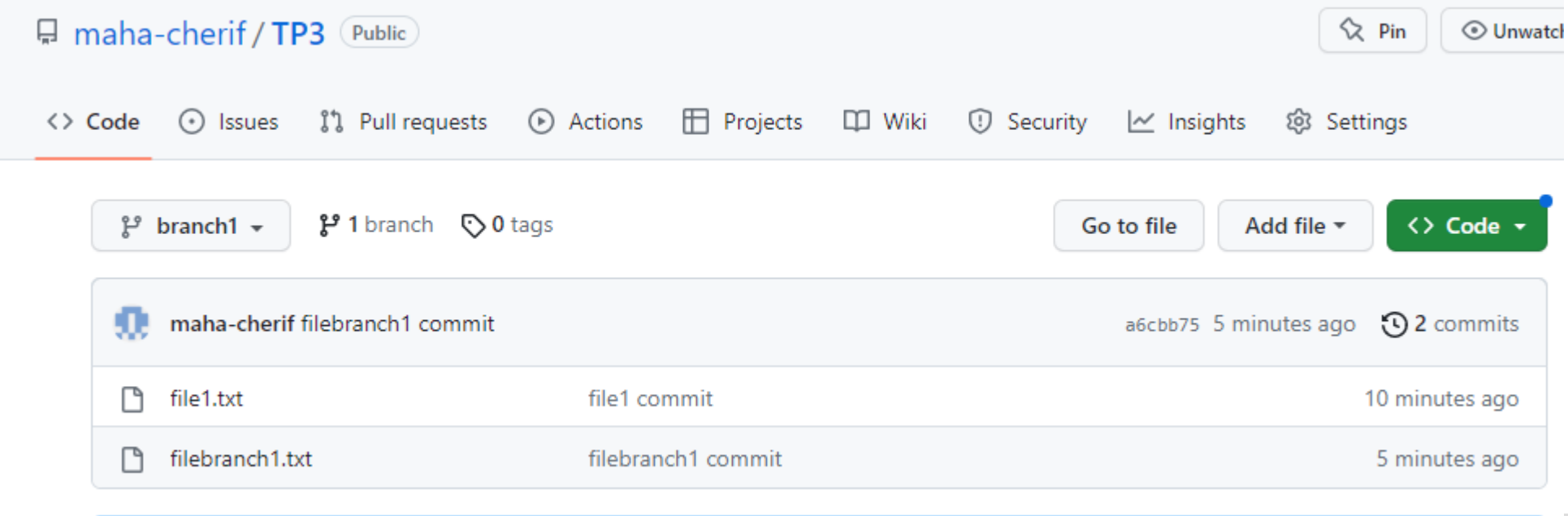
6. Switchez vers la branche créée, ajouter un fichier dedans (filebranch1.txt) par la suite commiter le, Switchez vers main et consultez tous les commits. Que remarquez-vous ?

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ touch filebranch1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git add .

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git commit -m "filebranch1 commit"
[branch1 a6cbb75] filebranch1 commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 filebranch1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git checkout master
Switched to branch 'master'

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git branch
  branch1
* master

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git log --oneline
b74e4a8 (HEAD -> master) file1 commit
```

TP4 Git branch

7. Uploader la nouvelle branche dans serveur distant avec la commande « gi push origin Nom de branche » que remarquez-vous ?

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git remote add origin git@github.com:maha-cherif/TP3.git

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (master)
$ git push origin branch1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 419 bytes | 209.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:maha-cherif/TP3.git
 * [new branch]      branch1 -> branch1
```



TP4 Git branch

8. Modifiez le contenu du fichier et consulter les modifications apportées avec la commande « git diff NomFichier »
9. Modifiez les deux derniers commits et vérifier la différence entre ces deux derniers avec la commande « git diff "id de 1er commit" "id de 2eme commit " »
10. Consulter la différence entre les deux derniers commits et sur quels fichiers ils ont été effectués avec la commande « git diff "id de 1er commit" "id de 2eme commit " -- name oneline »

Remarque :

Si --name oneline ne sera pas accepté

Utiliser --name only

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git add filebranch1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git status
On branch branch1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   filebranch1.txt


user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git commit -m "branch1 commit"
[branch1 bdd20b9] branch1 commit
 1 file changed, 1 insertion(+), 1 deletion(-)

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git diff filebranch1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git diff HEAD

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git status
On branch branch1
nothing to commit, working tree clean

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git diff "29e5a11" "bb87209"
diff --git a/filebranch1.txt b/filebranch1.txt
index 41430c8..9dcf873 100644
--- a/filebranch1.txt
+++ b/filebranch1.txt
@@ -1 +1 @@
-création branch1
\ No newline at end of file
+modifier branch1
\ No newline at end of file
```

TP4 Git branch

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git diff "29e5a11" "bb87209" --nameoneline
usage: git diff [<options>] [<commit>] [--] [<path>...]
   or: git diff [<options>] --cached [--merge-base] [<commit>] [--] [<path>...]
   or: git diff [<options>] [--merge-base] <commit> [<commit>...] <commit> [--] [<path>...]
   or: git diff [<options>] <commit>...<commit> [--] [<path>...]
   or: git diff [<options>] <blob> <blob>
   or: git diff [<options>] --no-index [--] <path> <path>

common diff options:
  -z            output diff-raw with lines terminated with NUL.
  -p            output patch format.
  -u            synonym for -p.
  --patch-with-raw
                output both a patch and the diff-raw format.
  --stat        show diffstat instead of patch.
  --numstat     show numeric diffstat instead of patch.
  --patch-with-stat
                output a patch and prepend its diffstat.
  --name-only   show only names of changed files.
  --name-status show names and status of changed files.
  --full-index  show full object name on index lines.
  --abbrev=<n>  abbreviate object names in diff-tree header and diff-raw.
  -R            swap input file pairs.
  -B            detect complete rewrites.
  -M            detect renames.
  -C            detect copies.
  --find-copies-harder
                try unchanged files as candidate for copy detection.
  -l<n>         limit rename attempts up to <n> paths.
  -O<file>      reorder diffs according to the <file>.
  -S<string>    find filepair whose only one side contains the string.
  --pickaxe-all
                show all files diff when -S is used and hit is found.
  -a  --text    treat all files as text.


user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git diff "29e5a11" "bb87209" --name-only
filebranch1.txt
```

TP4 Git branch

11. Créez un fichier, après modifiez le et faites un roolback avec la commande « git checkout -- Nom de fichier » (On peut retourner à l'état initial de tous les fichiers au niveau d'un dossier courant avec la commande « git checkout -- »

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ touch file2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git add file2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git commit -m "next commit"
[branch1 66b62c5] next commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git add file2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git commit -m "fin de commit"
[branch1 0a1c770] fin de commit
 1 file changed, 1 insertion(+)

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git checkout file2.txt
Updated 0 paths from the index

user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git log --oneline
0a1c770 (HEAD -> branch1) fin de commit
66b62c5 next commit
bdd20b9 branch1 commit
bb87209 fichier modifié
29e5a11 modification filebranch1
a6cbb75 (origin/branch1) filebranch1 commit
b74e4a8 (master) file1 commit
```

TP4 Git branch

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 (branch1)
$ git checkout 66b62c5
Note: switching to '66b62c5'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 66b62c5 next commit
```

TP4 Git branch

Si la commande git diff nom_fichier ne sera pas accepté

On peut utiliser git diff ID de fichier

Ou bien git log –p

➔Pour voir la diff entre les commits

```
$ git log -p
commit 66b62c58b3f79b8465bbf693ba5a93b9d67adbb4 (HEAD)
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 20:10:33 2023 +0200

    next commit

diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..e69de29

commit bdd20b98e2c4c5d49f01b4fcee4db62685f97352
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 19:50:28 2023 +0200

    branch1 commit

diff --git a/filebranch1.txt b/filebranch1.txt
index 9dcf873..ce7c77a 100644
--- a/filebranch1.txt
+++ b/filebranch1.txt
@@ -1 +1 @@
-modifier branch1
\ No newline at end of file
+modif. branch1
\ No newline at end of file

commit bb872097890ebeb09438715168dcd585459d8ec7
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 19:46:04 2023 +0200

    fichier modifié

diff --git a/filebranch1.txt b/filebranch1.txt
index 41430c8..9dcf873 100644
--- a/filebranch1.txt
+++ b/filebranch1.txt
@@ -1 +1 @@
-création branch1
\ No newline at end of file
+modifier branch1
\ No newline at end of file

commit 29e5a11004de2c9d54dd0f3b85acd85917b73116
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 19:40:42 2023 +0200

    modification filebranch1

diff --git a/filebranch1.txt b/filebranch1.txt
```

TP4 Git branch

On peut utiliser git log -p -2

➔ Diff entre les deux derniers commit

```
user@DESKTOP-RRVCC4I MINGW64 ~/TP3 ((66b62c5...))
$ git log -p -2
commit 66b62c58b3f79b8465bbf693ba5a93b9d67adbb4 (HEAD)
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 20:10:33 2023 +0200

    next commit

diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..e69de29

commit bdd20b98e2c4c5d49f01b4fcee4db62685f97352
Author: maha-cherif <maha.cherif@gmail.com>
Date:   Sun Apr 23 19:50:28 2023 +0200

    branch1 commit

diff --git a/filebranch1.txt b/filebranch1.txt
index 9dcf873..ce7c77a 100644
--- a/filebranch1.txt
+++ b/filebranch1.txt
@@ -1 +1 @@
-modifier branch1
\ No newline at end of file
+modif. branch1
\ No newline at end of file
```

TP5 Git Merge

1. Créer une nouvelle branche « hotfix », positionnez vous au niveau de cette nouvelle branche.

2. Créez et remplissez deux fichiers «HotfixFile1.txt» et «HotfixFile2.txt » et faites un commit pour chacun.

```
user@DESKTOP-RRVCC4I MINGW64 ~
$ mkdir merge

user@DESKTOP-RRVCC4I MINGW64 ~
$ cd merge

user@DESKTOP-RRVCC4I MINGW64 ~/merge
$ git init
Initialized empty Git repository in C:/Users/user/merge/.git/

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ touch file1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git add file1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "version file1"
[master (root-commit) e08d540] version file1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git checkout -b hotfix
Switched to a new branch 'hotfix'

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ touch HotfixFile1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ git add HotfixFile1.txt
```

TP5 Git Merge

3. Déplacez-vous au niveau de la branche principale « master » et vérifiez la position du pointeur HEAD.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ git commit -m "version hotfix1"
[hotfix b2a36e5] version hotfix1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 HotfixFile1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ touch HotfixFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ git add HotfixFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ git commit -m "version hotfix2"
[hotfix 97d0b1d] version hotfix2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 HotfixFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (hotfix)
$ git checkout master
Switched to branch 'master'
```

TP5 Git Merge

4. Créez deux fichiers «MasterFile1.txt» et «MasterFile2.txt» en les remplissant et faites un commit pour chacun à part.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
e08d540 (HEAD -> master) version file1

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ touch MasterFile1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git add MasterFile1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "version Master1"
[master 8650448] version Master1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 MasterFile1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ touch MasterFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git add MasterFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "version Master2"
[master 9e916a3] version Master2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 MasterFile2.txt
```

TP5 Git Merge

5. Vérifiez que vous vous positionnez dans le main et fusionner (Merge) la branche hotfix dans la branche principale avec la commande «git merge hotfix» .

6. Lister l'ensemble des fichiers pour vérifier que les fichiers créés au niveau de la branche « hotfix » ont été bien rajoutés dans la branche principale.

```
MINGW64:/c/Users/user/merge
Merge branch 'hotfix'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
.git/MERGE_MSG [unix] (13:01 01/05/2023)                    1,1 All
"~/merge/.git/MERGE_MSG" [unix] 6L, 248B
```

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git merge hotfix
Merge made by the 'ort' strategy.
 HotfixFile1.txt | 0
 HotfixFile2.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 HotfixFile1.txt
 create mode 100644 HotfixFile2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
f927805 (HEAD -> master) Merge branch 'hotfix'
9e916a3 version Master2
8650448 version Master1
97d0b1d (hotfix) version hotfix2
b2a36e5 version hotfix1
e08d540 version file1
```

TP5 Git Merge

7. Tapez la commande « git log --oneline --graph --decorate » pour visualiser le merge effectué.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
f927805 (HEAD -> master) Merge branch 'hotfix'
9e916a3 version Master2
8650448 version Master1
97d0b1d (hotfix) version hotfix2
b2a36e5 version hotfix1
e08d540 version file1

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline --graph --decorate
*   f927805 (HEAD -> master) Merge branch 'hotfix'
|\
| * 97d0b1d (hotfix) version hotfix2
| * b2a36e5 version hotfix1
* | 9e916a3 version Master2
* | 8650448 version Master1
|/
* e08d540 version file1
```

TP5 Git Merge

8. Créez une nouvelle branche « squashBranch », créer un nouveau fichier dedans « squashtest1.txt » et faites un commit.

9. Retournez vers la branche principale, créez un nouveau fichier « MainSquashtest1.txt » et faites un commit.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git checkout -b squashBranch
Switched to a new branch 'squashBranch'

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ touch squashtest1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git add squashtest1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git commit -m "version squashtest1"
[squashBranch c23f33a] version squashtest1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 squashtest1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git checkout master
Switched to branch 'master'

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ touch MainSquashtest1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git add MainSquashtest1.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "version MainSquashtest1"
[master 8df8188] version MainSquashtest1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 MainSquashtest1.txt
```

TP5 Git Merge

10. Pointez de nouveau sur la branche « SquashBranch », créer un 2ème fichier dedans « squashtest2.txt » et faites un commit.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git checkout squashBranch
Switched to branch 'squashBranch'

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ touch squashtest2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git add squashtest2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git commit -m "version squashtest2"
[squashBranch b46d9bd] version squashtest2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 squashtest2.txt
```

TP5 Git Merge

11. Retournez vers le main et faites un merge avec squash. Vérifiez le log. Que remarquez-vous ?

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (squashBranch)
$ git checkout master
Switched to branch 'master'

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git merge squashBranch --squash
Automatic merge went well; stopped before committing as requested
Squash commit -- not updating HEAD

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
8df8188 (HEAD -> master) version MainSquashtest1
f927805 Merge branch 'hotfix'
9e916a3 version Master2
8650448 version Master1
97d0b1d (hotfix) version hotfix2
b2a36e5 version hotfix1
e08d540 version file1

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   squashtest1.txt
        new file:   squashtest2.txt
```

TP5 Git Merge

12. Faites un nouveau commit et nommez le « Merge squash into main ».

13. Vérifiez le log de nouveau.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "Merge squash into master"
[master dc88588] Merge squash into master
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 squashtest1.txt
 create mode 100644 squashtest2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
dc88588 (HEAD -> master) Merge squash into master
8df8188 version MainSquashtest1
f927805 Merge branch 'hotfix'
9e916a3 version Master2
8650448 version Master1
97d0b1d (hotfix) version hotfix2
b2a36e5 version hotfix1
e08d540 version file1
```

TP5 Git Merge

12. Faites un nouveau commit et nommez le « Merge squash into main ».

13. Vérifiez le log de nouveau.

```
user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git commit -m "Merge squash into master"
[master dc88588] Merge squash into master
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 squashtest1.txt
 create mode 100644 squashtest2.txt

user@DESKTOP-RRVCC4I MINGW64 ~/merge (master)
$ git log --oneline
dc88588 (HEAD -> master) Merge squash into master
8df8188 version MainSquashtest1
f927805 Merge branch 'hotfix'
9e916a3 version Master2
8650448 version Master1
97d0b1d (hotfix) version hotfix2
b2a36e5 version hotfix1
e08d540 version file1
```

TP6 Conflit de merge

1. Créez une nouvelle branche « feature_db » en se positionnant dedans.

```
user@DESKTOP-RRVCC4I MINGW64 ~
$ mkdir conflit

user@DESKTOP-RRVCC4I MINGW64 ~
$ cd conflit

user@DESKTOP-RRVCC4I MINGW64 ~/conflit
$ git init
Initialized empty Git repository in C:/Users/user/conflit/.git/

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ touch file.txt

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git add file.txt

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git commit -m "version 0"
[master (root-commit) ac2425a] version 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file.txt

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git checkout -b feature_db
Switched to a new branch 'feature_db'
```

TP6 Conflit de merge

2. Créez un fichier « data_base.db, remplissez le par ce contenu « this is feature_db branch source code » et commiter-le.

```
user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git checkout -b feature_db
Switched to a new branch 'feature_db'

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (feature_db)
$ echo "this is feature_db branch source code" > feature.db

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (feature_db)
$ cat feature.db
this is feature_db branch source code

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (feature_db)
$ git add feature.db
warning: in the working copy of 'feature.db', LF will be replaced by CRLF the ne
xt time Git touches it

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (feature_db)
$ git commit -a -m "version feature_db1"
[feature_db 4410f36] version feature_db1
 1 file changed, 1 insertion(+)
 create mode 100644 feature.db
```

TP6 Conflit de merge

3. Pointez le pointeur sur la branche principale, créez un nouveau fichier avec le même nom « data_base.db », remplissez-le par ce contenu « this is main branch source code » et commiter-le.
4. Fusionner la branche « feature_db » dans le master. Que se passe il ?

```
user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git add feature.db
warning: in the working copy of 'feature.db', LF will be replaced by CRLF the ne
xt time Git touches it

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git commit -a -m "version main_db1"
[master f876cac] version main_db1
 1 file changed, 1 insertion(+)
 create mode 100644 feature.db

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git merge feature_db
Auto-merging feature.db
CONFLICT (add/add): Merge conflict in feature.db
Automatic merge failed; fix conflicts and then commit the result.
```

TP6 Conflit de merge

4. Fusionner la branche « feature_db » dans le master. Que se passe il ?
5. Accédez au fichier et gardez le contenu approprié.

feature - Bloc-notes

Fichier Edition Format Affichage Aide

```
<<<<<<< HEAD
this is main branch source code
=======
this is feature_db branch source code
>>>>>>> feature_db
```

*feature - Bloc-notes

Fichier Edition Format Affichage Aide

```
this is main branch source code

this is feature_db branch source code
```

```
user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master)
$ git merge feature_db
Auto-merging feature.db
CONFLICT (add/add): Merge conflict in feature.db
Automatic merge failed; fix conflicts and then commit the result.

user@DESKTOP-RRVCC4I MINGW64 ~/conflit (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both added:      feature.db

no changes added to commit (use "git add" and/or "git commit -a")
```

TP6 Conflit de merge

6. Maintenant, on suppose qu'un développeur est en train de travailler sur un feature et une tache urgente s'est avérée, il voulait sauvegarder son travail mais sans faire le commit (faute d'organisation). Il est possible de sauvegarder son état actuel grace à la commande « git stash »

a. git stash : permet de sauvegarder seulement l'état des fichiers existants dans le staging area
b. git stash -a : permet de sauvegarder l'état des fichiers existants dans le staging area et l'untracked area (mm un nouveau fichier non encore suivi par le git)
c. git stash list : pour lister les différentes modifications non commitées
d. git stash apply id_stash: pour retourner à un état spécifique en fournissant son id (à partir du stash list) : Exemple : git stash apply stash@{0}
e. git stash pop id_stash: permet de retourner à un état spécifique en vidant la cache ou on a sauvegardé la liste des stashs (modifications).