

## Qualité de développement

### TP : Git

#### Clonage d'un dépôt distant

1. Créer un nouveau projet sur <https://gitlab.univ-lorraine.fr/> et le nommer **tp-git**. A la différence du cours et du td, laissez la case "Initialize repository with a README" cochée pour que le dépôt ne soit pas vide.
2. Nous allons maintenant **cloner** ce dépôt sur votre machine. Cloner signifie que l'on apporte une copie du projet avec le dépôt git correspondant au dernier commit. Ici, le projet est léger puisqu'il ne comporte que le fichier **README** et un seul commit. Pour cela, ouvrez un terminal et exécutez la commande suivante :

```
% git clone https://gitlab.univ-lorraine.fr/login/tp-git.git
Cloning into 'tp-git'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
% cd tp-git
% ls -a
.  ..  .git  README.md
% git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

#### Travail collaboratif et gestion de conflits

##### Mise en situation

Vous allez maintenant travailler en binôme. Dans le reste du sujet, l'un de vous prendra le rôle d'Alice et l'autre de Bob.

##### Actions à effectuer :

1. Alice invite Bob en tant que développeur sur son dépôt **tp-git**.
2. Bob crée un clone du dépôt **tp-git** d'Alice sur sa machine.

**Remarque :** si vous aimez être seul pour travailler, c'est possible. Dans ce cas, vous prendrez le rôle d'Alice et Bob. Pour cela, vous créerez deux clones de votre dépôt **tp-git** sur votre machine.

Nous allons maintenant tester différentes situations de travail collaboratif et de gestion de conflits.

## Situation 1 : Alice travaille et Bob ne fait rien

### Cas 1 : Bob fait bien les choses

1. Alice crée un fichier `alice.txt` dans le répertoire de travail et y écrit quelques lignes.
2. Alice ajoute ce fichier au dépôt et fait un commit.
3. Si Bob regarde son répertoire de travail, il ne voit pas le fichier `alice.txt` :

```
bob@bob:~/tp-git$ ls -a
.  ..  .git  README.md
```

**Explication :** les dépôts sont **locaux** et ne sont pas synchronisés en temps réel. Dans les faits, les deux dépôts locaux peuvent avoir un historique de commits complètement différent.

4. Si Bob regarde l'état de son dépôt, il ne voit pas qu'Alice a modifié son dépôt local :

```
bob@bob:~/tp-git$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

**Explication :** l'échange d'information entre les collaborateurs se fait par le biais du dépôt distant. Pour cela, il faut que les collaborateurs fassent des **mise à jour** du dépôt distant.

5. Alice pousse son commit sur le dépôt distant.
6. Si Bob regarde son répertoire de travail, il ne voit toujours pas le fichier `alice.txt` :

```
bob@bob:~/tp-git$ ls -a
.  ..  .git  README.md
```

**Explication :** quand un collaborateur fait une mise à jour du dépôt distant, cette mise à jour n'est pas automatiquement envoyée aux autres collaborateurs.

7. Si Bob regarde l'état de son dépôt, il voit qu'il est à jour par rapport au dépôt distant alors que ce n'est pas le cas :

```
bob@bob:~/tp-git$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

8. S'il veut savoir si son dépôt local est à jour par rapport au dépôt distant, Bob doit faire une **mise à jour** de son dépôt local avec la commande **pull**:

```
bob@bob:~/tp-git$ git pull
From https://gitlab.univ-lorraine.fr/alice/tp-git
   0d4b13e..365fb1f  main      -> origin/main
Updating 0d4b13e..365fb1f
Fast-forward
 alice.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 alice.txt
```

9. Si Bob regarde son répertoire de travail, il voit maintenant le fichier **alice.txt** :

```
bob@bob:~/tp-git$ ls -a
.  ..  .git  alice.txt  README.md
```

10. Si Bob regarde l'historique du dépôt, il voit que le commit de Alice est présent :

```
bob@bob:~/tp-git$ git log
Author: Alice <alice@alice.com>
Date:   Sat Feb 11 15:41:09 2023 +0100

    feat: ajout du fichier alice.txt

commit 0d4b13e7529e7c8c98991363a48f9089daf08467
Author: Alice <alice@ualice.com>
Date:   Sat Feb 11 14:37:52 2023 +0000

    Initial commit
```

11. Si Bob réessaye de faire un **pull**, il voit que son dépôt est à jour :

```
bob@bob:~/tp-git$ git pull
Already up to date.
```

## Situation 2 : Alice et Bob travaillent sur deux fichiers différents

### Cas 1

1. Alice modifie le fichier **alice.txt** et fait un commit et publie sur le dépôt distant.
2. Bob crée un fichier **bob.txt** mais ne fait pas de commit.
3. Bob fait un **git pull**

```
bob@bob:~/tp-git$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 297 bytes | 99.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/alice/tp-git
   365fb1f..21cc812  main       -> origin/main
Updating 365fb1f..21cc812
Fast-forward
   alice.txt | 1 +
   1 file changed, 1 insertion(+)
```

Il n'y a pas de problème car Bob n'a pas modifié le fichier `alice.txt` et il récupère donc les modifications d'Alice.

## Cas 2

1. Bob fait un commit en local mais ne le publie pas sur le dépôt distant.
2. Alice modifie à nouveau le fichier `alice.txt` et fait un commit et publie sur le dépôt distant.
3. Bob fait un `git pull` :

```
bob@bob:~/tp-git$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 302 bytes | 151.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/jozefowil/tp-git
   21cc812..7643bc4  main       -> origin/main
fatal: Need to specify how to reconcile divergent branches.
```

**Explication :** Bob a l'ancienne version du fichier `alice.txt` dans son dépôt local et Alice a modifié le fichier `alice.txt` sur le dépôt distant. Git ne sait pas comment fusionner les deux modifications. Il faut donc que Bob résolve le conflit.

**Remarque :** le problème est au niveau du dépôt local de Bob, le dépôt distant n'est donc pas impacté.

**Comportement :** tant que Bob n'a pas résolu le conflit, il ne peut pas faire de `pull` ou de `push`.

```
bob@bob:~/tp-git$ git push
To https://gitlab.univ-lorraine.fr/alice/tp-git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://gitlab.univ-lorraine.fr/jozefowil/tp-git'
```

Le problème est visible via la commande `git status` :

```
bob@bob:~/tp-git$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
```

4. Bob doit résoudre le conflit en fusionnant les deux versions du fichier `alice.txt` :

```
bob@bob:~/tp-git$ git merge origin/main
```

**Remarque** : un éditeur de texte s'ouvre pour enregistrer comment le conflit a été résolu. Vous pouvez laisser le message par défaut. Pour quitter l'éditeur, il faut faire `:wq` puis `entrée`.

5. Bob peut maintenant faire un `git push` pour publier ses modifications sur le dépôt distant :

```
bob@bob:~/tp-git$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 10 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 658 bytes | 658.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.univ-lorraine.fr/jozefowil/tp-git.git
7643bc4..223746c  main -> main
```

6. Alice peut maintenant faire un `git pull` pour récupérer les modifications de Bob.

**Etat actuel** : Alice et Bob ont chacun un dépôt local synchronisé avec le dépôt distant.

### Cas 3 (où Bob est un vilain garçon)

1. Alice modifie le fichier `alice.txt` et fait un commit et publie sur le dépôt distant.
2. Bob modifie le fichier 'bob.txt' et fait un commit local.
3. Bob fait un `git push` pour publier ses modifications sur le dépôt distant.

```
bob@bob:~/tp-git$ git push
To https://gitlab.univ-lorraine.fr/jozefowil/tp-git.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://gitlab.univ-lorraine.fr/jozefowil/tp-git.git'
```

**Explication** : Bob n'est pas synchronisé avec le dépôt distant. Il ne peut donc pas publier ses commits sur le dépôt distant.

**Règle d'or numéro une :** toujours faire un `git pull` avant de faire un `git push`.

**Remarque :** il peut être nécessaire de régler des conflits lors du `git pull`.

4. Bob fait un `git pull`, règle les conflits, et fait un `push`.
5. Alice fait un `git pull` pour récupérer les modifications de Bob.

**Etat actuel :** Alice et Bob ont chacun un dépôt local synchronisé avec le dépôt distant.

### Situation 3 : Alice et Bob travaillent sur le même fichier (et Bob fait une énorme bêtise)

Alice et Bob travaille sur le fichier `alice.txt`.

1. Alice ajoute à la première ligne du fichier `alice.txt` le texte `Alice was here`.
2. Alice fait un commit en local et publie sur le dépôt distant.
3. Bob modifie le fichier `alice.txt` en ajoutant à la première ligne le texte `Bob was here`.
4. Bob a appris de ses erreurs et fait un `git pull` avant de faire un `git push`.

```
bob@bob:~/tp-git$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 333 bytes | 111.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/alice/tp-git
   0ecbdc4..48b7cdc  main       -> origin/main
Updating 0ecbdc4..48b7cdc
error: Your local changes to the following files would be overwritten by merge:
    alice.txt
Please commit your changes or stash them before you merge.
Aborting
```

**Explication :** Bob a modifié le fichier `alice.txt` sans faire de commit. Il ne peut donc pas faire de `git pull`.

**Règle d'or numéro deux :** toujours faire un `git commit` avant de faire un `git pull`.

**Règle d'or :** toujours faire un `git commit` avant de faire un `git pull` avant de faire un `git push`.

5. Bob fait donc un `git commit`.
6. Bob fait un `git pull` (s'il y a un conflit, il faut le résoudre : `git merge origin/main`).
7. Bob obtient le message suivant :

```
bob@bob:~/tp-git$ git pull
Auto-merging alice.txt
CONFLICT (content): Merge conflict in alice.txt
Automatic merge failed; fix conflicts and then commit the result.
```

**Explication :** Alice et Bob ont modifié le même fichier. Leurs modifications respectives sont en conflits (ici on modifie la même ligne). Git ne peut pas savoir qui d'Alice ou de Bob a raison. Il faut donc que Bob résolve le conflit avant de pouvoir publier.

9. Bob fait un `git status` pour voir le fichier en conflit :

```
bob@bob:~/tp-git$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

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 modified:   alice.txt

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

8. Bob ouvre le fichier `alice.txt` et voit le conflit :

```
<<<<<< HEAD
Bob was here
=====
Alice was here
>>>>>> origin/main
```

**Explication :** Git a ajouté des marqueurs pour indiquer où commence le conflit. Le texte entre `<<<<<< HEAD` et `=====` est la version de Bob. Le texte entre `=====` et `>>>>>> origin/main` est la version présente sur le dépôt distant.

9. Bob modifie le fichier `alice.txt` pour résoudre le conflit en supprimant la partie qu'il ne veut pas garder et les balises. Ici, on va supposer que Bob a raison et qu'il veut garder sa version. Il laisse donc dans le fichier :

```
Bob was here
```

10. Bob fait un `git add alice.txt` pour indiquer que le conflit est résolu. S'il fait un `git status`, il voit que le fichier `alice.txt` est en vert, ce qui signifie que le conflit est résolu.

11. Bob fait un commit pour prendre en compte la nouvelle version du fichier `alice.txt`.

```
bob@bob:~/tp-git$ git commit -a -m "fix: résolution du conflit sur alice.txt par Bob -> on garde la version de Bob"
[main d160c52] fix: résolution du conflit sur alice.txt par Bob -> on garde la version de Bob
```

12. Bob qui a appris sa leçon, fait un **git pull** avant de faire un **git push** au cas où le dépôt distant aurait été modifié entre temps.
13. Alice fait un commit si nécessaire de son répertoire local avant de faire un **git pull** pour récupérer les modifications de Bob.

**Etat actuel :** Alice et Bob ont chacun un dépôt local synchronisé avec le dépôt distant.

## Conclusion

---

**COMMIT, PULL, PUSH :** toujours faire ces trois commandes dans cet ordre.

## Suite du travail

---

- Echanger avec votre binôme les rôles d'Alice et de Bob.
- Faites 2-3 tests de travail collaboratif, sur des fichiers différents ou des fichiers identiques, pour vous assurer que vous avez compris le mécanisme.

## Rendu

---

- Compléter votre fichier **compte\_rendu.txt** avec les nouvelles commandes que vous avez découvertes.
- Faire un tag pour indiquer qu'il s'agit de la fin de ce TD