

Qualité de développement

Travail collaboratif via Git

Remarque : selon le serveur git, la version de git, l'OS ... certaines commandes peuvent se comporter légèrement différemment.

Principe

L'idée est que plusieurs personnes travaillent sur un même projet. Chacun travaille sur sa propre copie du projet. La synchronisation des copies se fait via un serveur distant. Git permet d'incorporer les modifications de chacun dans le projet commun et aussi de détecter les éventuels conflits.

On va voir dans ce cours quelque cas d'utilisation classique de Git. On va supposer que deux personnes (Alice et Bob) travaillent sur un même projet. Le projet est hébergé sur un serveur distant.

Remarque : il faut que le projet soit public pour que les deux personnes puissent y accéder ou qu'elles aient les droits d'accès au projet.

Cloner un projet

Si on ne possède pas une copie locale du projet, la première chose à faire est de cloner le projet, c'est-à-dire faire une copie locale du projet sur son ordinateur. Pour cela, on utilise la commande `git clone` :

```
git clone https://gitlab.univ-lorraine.fr/jozefowil/cours.git
Cloning into 'cours'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 25 (delta 4), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (25/25), done.
Resolving deltas: 100% (4/4), done.
```

Scénario 1 : Alice modifie le projet et Bob récupère les modifications

Etape 1 : Alice modifie le projet

Alice modifie le projet (création d'un fichier `alice.txt`) et souhaite que Bob puisse récupérer ses modifications. On suppose que le dépôt local d'Alice est à jour avec le dépôt distant. Pour cela, elle va d'abord ajouter ses modifications à son dépôt local :

```
alice % echo "alice" > alice.txt
alice % git add alice.txt
```

```
alice % git commit -m "ajout du fichier alice.txt"
[main 047054e] ajout du fichier alice.txt
1 file changed, 1 insertion(+)
create mode 100644 alice.txt
```

Important : Alice est prête à envoyer ses modifications sur le dépôt distant. Cependant, elle ne sait pas si le dépôt distant n'a pas été modifié entre temps. Il est donc nécessaire de récupérer les modifications du dépôt distant avant d'envoyer ses modifications. Si on ne le fait pas, on peut arriver à des conflits difficiles à résoudre (vous aurez l'occasion de voir cela en TD).

```
alice % git pull
Already up to date.
```

Ici, il n'y avait pas de modification. Alice peut donc envoyer ses modifications sur le dépôt distant :

```
alice % git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 279 bytes | 279.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Etape 2 : Bob récupère les modifications

On suppose que Bob n'a pas modifié son projet local.

```
git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

On remarque deux choses :

1. il n'y a pas de modification locale
2. la commande `status` n'indique pas que le dépôt local est en retard sur le dépôt distant

Pour savoir s'il y a une dé-synchronisation et mettre à jour son dépôt local, Bob doit faire un `pull` du dépôt distant :

```
bob % git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
```

```
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 259 bytes | 86.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/jozefowi1/cours
    df2e229..047054e  main      -> origin/main
Updating df2e229..047054e
Fast-forward
 alice.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 alice.txt
```

Son dépôt local contient bien le commit fait par Alice :

```
bob % git status
commit 047054e6fea4eb5e3a195626bfded2cb044bb79d (HEAD -> main, origin/main,
origin/HEAD)
Author: Alice <alice@thing.com>
Date:   Tue Dec 5 08:20:01 2023 +0100

    ajout du fichier alice.txt

commit df2e2298607ab5c74ed7448434445a31a2c4b5c7
Author: John Doe <john@doe.com>
Date:   Fri Nov 24 11:39:25 2023 +0100

    modification locale
...
```

Scénario 2 : Alice et Bob modifient le projet en même temps mais en travaillant sur des fichiers différents

Remarque : les rôles d'Alice et Bob sont interchangeables.

Etape 1 : Alice modifie le projet

```
alice % echo "ligne 2" >> alice.txt
```

Etape 2 : Bob modifie le projet

```
bob % echo "bob" >> bob.txt
bob % git add bob.txt
```

Etape 3 : Alice publie ses modifications sur le serveur distant

Comme elle a bien écouté le cours, Alice sait qu'elle doit faire un **commit** pour prendre en compte les modifications locales et ensuite un **pull** pour récupérer les éventuelles modifications du serveur distant avant de faire un **push** pour publier ses modifications.

```
alice % git commit -a -m "modification Alice scenario 2"
[main c30d201] modification Alice scenario 2
1 file changed, 1 insertion(+)
alice % git pull
Already up to date.
alice % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes | 278.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Etape 4 : Bob publie ses modifications sur le serveur distant

Bob est plus distrait. Il a déjà oublié qu'il faut faire un commit pour prendre en compte les modifications et qu'il faut faire un **pull** pour se synchroniser avec le serveur distant avant de faire un **push** pour publier ses modifications.

Il essaie directement de publier ses modifications :

```
bob % git push
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://gitlab.univ-
lorraine.fr/jozefowil/cours.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

ici git a détecté que le serveur distant contient des modifications qui ne sont pas présentes dans le dépôt local. Il faut donc récupérer les modifications du serveur distant avant de pouvoir publier ses modifications :

```
bob % 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 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 258 bytes | 86.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/jozefowil/cours
047054e..c30d201  main      -> origin/main
Updating 047054e..c30d201
```

```
Fast-forward
alice.txt | 1 +
1 file changed, 1 insertion(+)
```

Les modifications de Alice sont maintenant présentes dans le dépôt local de Bob.

```
bob % cat alice.txt
alice
ligne 2
```

Bob se dit qu'il peut maintenant publier ses modifications :

```
bob % git push
Everything up-to-date
```

On lui rappelle que les modifications qui sont prises en compte sont celles qui ont été ajoutées au dépôt local via **commit**. Or, Bob n'a pas encore ajouté ses modifications au dépôt local. Il faut donc faire un **commit** :

```
bob % git commit -m "modification Bob scenario 2"
[main f1407f3] modification Bob scenario 2
1 file changed, 1 insertion(+)
create mode 100644 bob.txt
```

Maintenant, Bob peut publier ses modifications :

```
bob % git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 277 bytes | 277.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Bilan

Bob commence à comprendre que pour publier ses modifications, il faut :

1. faire un **commit** pour prendre en compte les modifications locales
2. faire un **pull** pour récupérer les modifications du serveur distant
3. faire un **push** pour publier ses modifications

Scénario 3 : Alice et Bob modifient le projet en même temps en travaillant sur les mêmes fichiers mais sur des zones différentes

Etape 1 : Alice modifie le projet

Alice modifie la première ligne du fichier `alice.txt` qui devient localement :

```
alice scenario 3  
ligne 2
```

Etape 2 : Bob modifie le projet

Bob ajoute une ligne à la fin du fichier `alice.txt` qui devient localement :

```
alice  
ligne 2  
modification de bob scenario 3
```

Etape 3 : Alice publie ses modifications sur le serveur distant

Alice fait un `commit` pour prendre en compte ses modifications locales :

```
alice % git commit -a -m "modification Alice scenario 3"  
[main 3a14993] modification Alice scenario 3  
1 file changed, 1 insertion(+), 1 deletion(-)
```

Elle récupère les modifications du serveur distant (ici il lui manque notamment la modification de Bob faite lors du scénario 2) :

```
alice % git pull  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 257 bytes | 85.00 KiB/s, done.  
From https://gitlab.univ-lorraine.fr/jozefowil/cours  
c30d201..f1407f3  main      -> origin/main
```

Alice a bien récupéré le fichier `Bob.txt` :

```
alice % ls
README.txt  alice.txt  bob.txt      deno.jsonc  src
```

Elle peut donc publier ses modifications :

```
alice % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 293 bytes | 293.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Etape 4 : Bob publie ses modifications sur le serveur distant

Bob fait un **commit** pour prendre en compte ses modifications locales :

```
bob % git commit -a -m "modification Bob scenario 3"
[main 41a19e5] modification Bob scenario 3
1 file changed, 1 insertion(+)
```

Il récupère les modifications du serveur distant (ici il lui manque notamment la modification de Alice faite lors du scénario 3) :

```
bob % 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 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 273 bytes | 91.00 KiB/s, done.
```

Bob peut publier ses modifications :

```
bob % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Bilan

Le serveur distant a bien pris en compte les modifications de Alice et Bob. Bob commence à se dire que la règle de **commit, pull, push** est une bonne règle et qu'il ne faut pas l'oublier.

Remarque : Le dépôt local d'Alice n'est pas à jour, elle décide donc de synchroniser son dépôt local avec le dépôt distant.

Consciencieuse, elle décide d'appliquer la règle **commit, pull, push** :

```
alice % git commit -a -m "synchronisation avec le serveur distant fin scénario 3"

On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Elle n'a pas fait de modification par apport à son dernier commit local :

```
alice % git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Dans ce cas-là, il n'est donc pas nécessaire de faire le commit.

Elle récupère les modifications du serveur distant :

```
alice % git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 294 bytes | 98.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/jozefowi1/cours
   ce733cc..e837053  main      -> origin/main
Updating ce733cc..e837053
Fast-forward
 alice.txt | 1 +
 1 file changed, 1 insertion(+)
```

Elle peut maintenant publier ses modifications :

```
alice % git push
Everything up-to-date
```


Exception à la règle : si on souhaite juste récupérer les modifications du serveur distant et qu'il n'y pas eu de modification locale depuis le dernier commit, on peut faire un **pull** sans faire de **commit**. De même, si tous les commits locaux sont déjà sur le serveur distant, on peut se passer de faire le **push**.

Scénario 5 : Alice et Bob modifient le projet en même temps en travaillant sur les mêmes fichiers et sur les mêmes zones

Etape 1 : Alice modifie le projet local

Alice crée un fichier **conflit.txt** qui contient :

```
Alice scenario 5
```

Etape 2 : Bob modifie le projet local

Bob crée un fichier **conflit.txt** qui contient :

```
Bob scenario 5
```

Etape 3 : Alice publie ses modifications sur le serveur distant

Alice suit la règle **commit, pull, push** :

```
alice % git add conflit.txt
alice % git commit -m "modification Alice scenario 5"
[main ea4b73b] modification Alice scenario 5
 1 file changed, 1 insertion(+)
 create mode 100644 conflit.txt
alice % git pull
Already up to date.
alice % git pushEnumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 295.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Etape 4 : Bob publie ses modifications sur le serveur distant

Bob suit la règle **commit, pull, push** :

```
bob % git add conflit.txt
bob % git commit -m "modification Bob scenario 5"
```

```
[main d544d74] modification Bob scenario 5
1 file changed, 1 insertion(+)
create mode 100644 conflit.txt
bob % git pull
Auto-merging conflit.txt
CONFLICT (add/add): Merge conflict in conflit.txt
error: could not apply d544d74... modification Bob scenario 5
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply d544d74... modification Bob scenario 5
```

Il y a un problème car git ne sait pas intégrer les modifications de Bob dans le fichier `conflit.txt` avec celle d'Alice. Regardons l'état du fichier `conflit.txt` dans le dépôt local de Bob :

```
<<<<<<< HEAD
Alice scenario 5
=====
Bob scenario 5
>>>>>>> d544d74 (modification Bob scenario 5)
```

On voit que git a ajouté des marqueurs pour indiquer les zones de conflit. La première zone est celle du dépôt distant (HEAD) et la seconde est celle du dépôt local.

Il faut donc que Bob résolve le conflit manuellement. Il peut par exemple décider de garder les deux modifications :

```
Alice scenario 5
Bob scenario 5
```

ou de garder uniquement la modification de Bob :

```
Bob scenario 5
```

ou de garder uniquement la modification de Alice :

```
Alice scenario 5
```

Dans tous les cas, il s'agit de retirer les marqueurs de conflit. Une fois le conflit résolu, il faut faire un `add` pour indiquer à git que le conflit est résolu :

```
bob % git add conflit.txt
```

Il faut ensuite faire un **commit** pour prendre en compte la résolution du conflit :

```
bob % git commit -m "résolution du conflit"
[HEAD 42b2b45] résolution du conflit
1 file changed, 1 insertion(+), 1 deletion(-)
```

Dans un projet impliquant de nombreuses personnes, il faudrait faire un **pull** pour être sûr de récupérer d'éventuelles modification du serveur distant. Ici, on sait qu'Alice n'a rien fait depuis. On peut donc directement faire un **push** :

```
bob % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 277 bytes | 277.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
```

Alice peut maintenant récupérer les modifications de Bob (en faisant un commit pour prendre en compte les modifications locales s'il y en a et en faisant un push à la fin pour publier ses modifications si elle en a fait) :

```
alice % git pull
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
Unpacking objects: 100% (3/3), 406 bytes | 135.00 KiB/s, done.
From https://gitlab.univ-lorraine.fr/jozefowil/coursgit
   ea4b73b..42b2b45  main      -> origin/main
Updating ea4b73b..42b2b45
Fast-forward
   conflit.txt | 2 +-
   1 file changed, 1 insertion(+), 1 deletion(-)
alice % cat conflit.txt
Bob scenario 5
```

Important : la règle **commit**, **pull**, **push**

Pour résumer, la règle **commit**, **pull**, **push** est la suivante :

1. faire un **commit** pour prendre en compte les modifications locales
2. faire un **pull** pour récupérer les modifications du serveur distant (régler les conflits si besoin)

3. faire un **push** pour publier ses modifications

Si vous ne respectez pas cette règle, vous risquez d'avoir des conflits difficiles à résoudre et rendre dépôt local mais aussi distant inutilisable.