Introduction à Git

Programmation séquentielle en C, 2022-2023

Orestis Malaspinas (A401) et un tout petit peu Michaël El Kharroubi

2022-11-08

Informatique et Systèmes de Communication, HEPIA

Qu'est-ce que Git?

- Git est un outil de gestion de versions (dév. par L. Torvalds).
  - Cela évite d'avoir à gérer les fichiers d'un projet comme:
    - fichier.c
    - fichier_10_3_2020.c
    - fichier_10_3_2020_16h.c
    - fichier_10_3_2020_16h_Malaspinas.c
    - fichier_10_3_2020_16h_Albuquerque.c
  - L'historique est accessible à tout moment.
  - Difficile d'écraser le mauvais fichier lors d'une synchronisation.
- Possibilité de découpler le développement dans un projet.
  - Fusionne les modifications non-conflictuelles automatiquement.
  - Un projet peut avoir différentes *branches* de développement (on peut développer une nouvelle version et faire des corrections de bug en parallèle).
- **Permet le travail de plusieurs développeurs sur le même projet!**

Principe de fonctionnement de Git (1/3)

Git est un outil décentralisé...

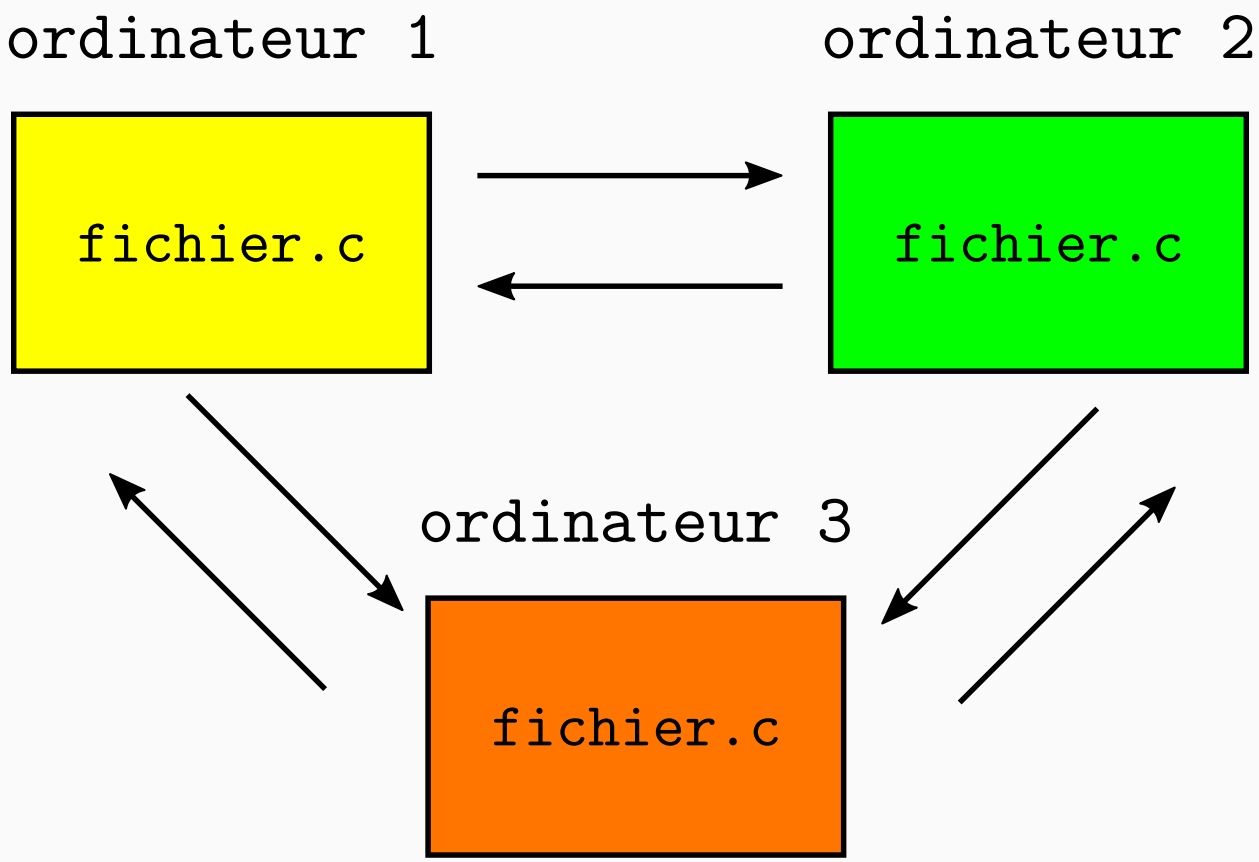
Principe de fonctionnement de Git (1/3)

Git est un outil décentralisé...

Mais, typiquement un projet git possède un serveur “officiel” (centralisé):

- Un · e développeur · euse peut faire une copie (clone) de tout le projet (sur son ordinateur).
- Modifier localement le projet et publier (push) ses propres modifications (sur son ordinateur).
- Demander au gestionnaire du projet de fusionner (merge) ses modifications avec le serveur “officiel” (pull/merge request):
  - L’administrateur récupère le projet depuis le serveur du développeur.
  - Fusionne le projet officiel avec celui modifié (merge).
  - Publie les modifications sur le serveur officiel (push).

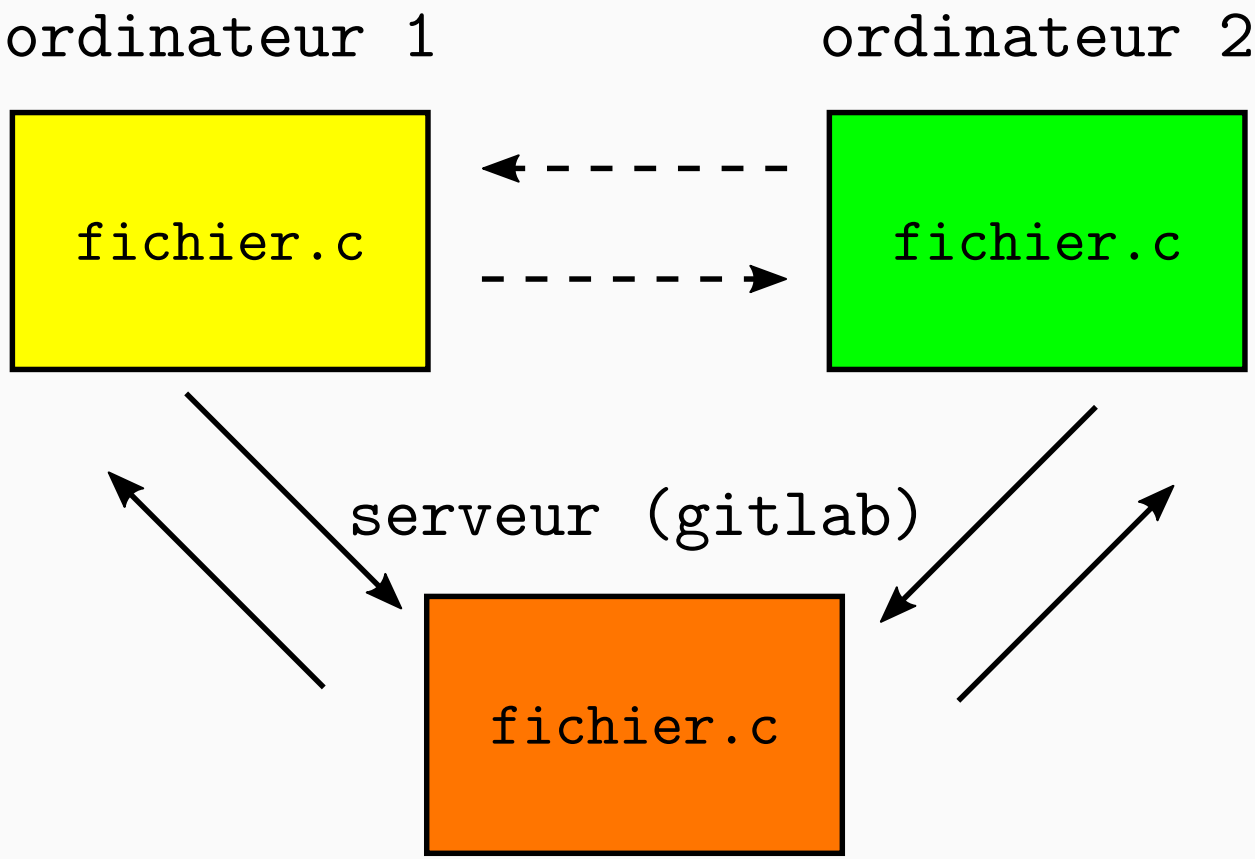
Principe de fonctionnement de Git (2/3)



Une copie complète du projet par ordinateur

Figure 1: Décentralisation complète

Principe de fonctionnement de Git (3/3)



Une copie complète du projet par ordinateur

Figure 2: Avec un serveur central

Exemple de fonctionnement

Création du dépôt et clone

1. Création d'un dépôt *tutorial* git sur https://gitedu.hesge.ch.
2. Clone du dépôt.

```
$ git clone ssh://git@ssh.hesge.ch:10572/orestis.malaspin/tutorial.git
Cloning into 'tutorial'...
warning: You appear to have cloned an empty repository.
$ cd tutorial
[tutorial]$
```

3. Et voilà vous êtes dans votre dépôt git.

Ajout de fichiers à l'historique (1/4)

Commandes: `git add`, `git status`, `git commit`, `git push`

1. Création du fichier `premierfichier.c`.
2. Ajout du `premierfichier.c` aux fichiers suivis par git.
3. *Commit* du fichier ajouté à l'historique des modifications.
4. *Push* de l'état de l'historique sur le serveur.

```
[tutorial]$ echo Hello World > premierfichier.c
[tutorial]$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    premierfichier.c

nothing added to commit but untracked files present (use "git add" to
    track)
```

Ajout de fichiers à l'historique (2/4)

Commandes: `git add`, `git status`, `git commit`, `git push`

```
[tutorial]$ git add premierfichier.c
[tutorial]$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   premierfichier.c

[tutorial]$ git commit -m "mon premier commit"
[master (root-commit) a4f2052] mon premier commit
 1 file changed, 1 insertion(+)
 create mode 100644 premierfichier.c
```

Ajout de fichiers à l'historique (3/4)

Commandes: `git add`, `git status`, `git commit`, `git push`

```
[tutorial]$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
[tutorial]$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 238 bytes | 238.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://ssh.hesge.ch:10572/orestis.malaspin/tutorial.git
 * [new branch]      master -> master
```

Ajout de fichiers à l'historique (4/4)

Recommandations

- Faire des *commits* réguliers (ne pas attendre d'avoir un projet qui fonctionne compèlètement).
- Mettre des messages de *commit* qui font du sens.
- Éviter d'ajouter de fichiers binaires (prennent de la place).
  - Les fichiers binaires sont générables par l'utilisateur du projet.
- Éviter de faire `git add .`
- Utiliser les fichiers `.gitignore` pour se protéger.

Modification de fichiers dans l'historique (1/3)

Commandes: `git diff`, `git log`

1. Modification du fichier `premierfichier.c`.
2. Ajout/commit/push des modifictations.

```
[tutorial]$ echo Wild World > premierfichier.c
[tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
       directory)

    modified:   premierfichier.c

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

Modification de fichiers dans l'historique (2/3)

Commandes: `git diff`, `git log`

```
[tutorial]$ git diff
diff --git a/premierfichier.c b/premierfichier.c
index 557db03..9622e40 100644
--- a/premierfichier.c
+++ b/premierfichier.c
@@ -1 +1 @@
-Hello World
+Wild World
[tutorial]$  git commit -am "nouvelles modifications"
[master f9ab3ec] nouvelles modifications
 1 file changed, 1 insertion(+), 1 deletion(-)
[tutorial]$ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 274 bytes | 274.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://ssh.hesge.ch:10572/orestis.malaspin/tutorial.git
   a4f2052..f9ab3ec  master -> master
```

Modification de fichiers dans l'historique (3/3)

Commandes: `git diff`, `git log`

```
[tutorial]$ git log
commit f9ab3ec4a00c46a12d7a45f133295acc5fb5cd20 (HEAD -> master, origin/
    master)
Author: Orestis Malaspinas <orestis.malaspinas@hesge.ch>
Date:   Sun Mar 4 22:48:21 2018 +0100

    nouvelles modifications

commit a4f2052147a752a8c12641f4f3352c5aa1802559
Author: Orestis Malaspinas <orestis.malaspinas@hesge.ch>
Date:   Sun Mar 4 22:25:24 2018 +0100

    mon premier commit
```

Revenir en arrière dans l'historique (1/6)

Commandes: `git checkout`, `git reset`

1. Faire une modification dans un fichier par erreur.
2. Faire un `git add` par erreur.
3. Faire un `git commit` par erreur.

```
[tutorial]$ echo Oh no! An awful modification! > premierfichier.c
[tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
       directory)

    modified:   premierfichier.c

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

Revenir en arrière dans l'historique (2/6)

Commandes: `git checkout`, `git reset`

Une modification dans un fichier par erreur

```
[tutorial]$ git checkout premierfichier.c
[tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Revenir en arrière dans l'historique (3/6)

Commandes: `git checkout`, `git reset`

Faire un `git add` par erreur (1/2)

```
[tutorial]$ echo Oh no! An awful modification! > premierfichier.c
[tutorial]$ git add premierfichier.c
[tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   premierfichier.c
[tutorial]$ git reset HEAD
Unstaged changes after reset:
M   premierfichier.c
```

Revenir en arrière dans l'historique (4/6)

Commandes: `git checkout`, `git reset`

Faire un `git add` par erreur (2/2)

```
[tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
       directory)

    modified:   premierfichier.c

no changes added to commit (use "git add" and/or "git commit -a")
[tutorial]$ git diff
diff --git a/premierfichier.c b/premierfichier.c
index 9622e40..cfd5469 100644
--- a/premierfichier.c
+++ b/premierfichier.c
@@ -1 +1 @@
-Wild World
+Oh no! An awful modification!
```

Revenir en arrière dans l'historique (5/6)

Commandes: `git checkout`, `git reset`

Faire un `git commit` par erreur (1/2)

```
[tutorial]$ git commit -am "troisieme commit"
[master 0563c02] troisieme commit
 1 file changed, 1 insertion(+), 1 deletion(-)
[tutorial]$ git push
[tutorial]$ git reset f9ab3ec4a00c46a12d7a45f133295acc5fb5cd20
[tutorial]$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-
    forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
    modified:   premierfichier.c

no changes added to commit
[tutorial]$ git checkout premierfichier.c
```

Revenir en arrière dans l'historique (6/6)

Commandes: `git checkout`, `git reset`

Faire un `git commit` par erreur (2/2)

```
[tutorial]$ echo Wonderful World > premierfichier.c
[tutorial]$ git commit -am "la bonne troisieme modification"
[master 1b42970] la bonne troisieme modification
 1 file changed, 1 insertion(+), 1 deletion(-)
[tutorial]$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
[tutorial]$ git pull
Auto-merging premierfichier.c
CONFLICT (content): Merge conflict in premierfichier.c
Automatic merge failed; fix conflicts and then commit the result.
```

Il ne reste qu'à corriger le conflit et refaire un `git commit`, `git push`

Un `git push` par erreur

- Un `git push` est très difficile à “effacer”.
- Cela revient à *réécrire* l’historique de votre projet.
  - Cela est *dangereux*, surtout quand on travail à plusieurs.
- Le plus simple est de revenir à une version antérieure et faire un nouveau commit.
- Il existe des techniques *violentes* qu’on verra pas ici.

Retirer un fichier du contrôle de version (1/3)

Commande: `git rm`

- Il n'est plus nécessaire de suivre un fichier.
- **Attention : le fichier ne disparaît pas de l'historique.**

```
[orestis@perka tutorial]$ git rm premierfichier.c
rm 'premierfichier.c'
[orestis@perka tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    premierfichier.c
[orestis@perka tutorial]$ git commit -am "efface donc ce fichier"
[master 8f76d90] efface donc ce fichier
 1 file changed, 1 deletion(-)
 delete mode 100644 premierfichier.c
```

Retirer un fichier du contrôle de version (2/3)

Commande: `git rm` (1/2)

```
[orestis@perka tutorial]$ ls -ltr
total 0
[orestis@perka tutorial]$ git reset
    bbb151324289dc2f85468f5721ec1021692dd216
Unstaged changes after reset:
D   premierfichier.c
[orestis@perka tutorial]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
      directory)

    deleted:    premierfichier.c

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

Retirer un fichier du contrôle de version (3/3)

Commande: `git rm` (2/2)

On peut retrouver le fichier dans l'historique.

```
[orestis@perka tutorial]$ ls -ltr
total 0
[orestis@perka tutorial]$ git checkout premierfichier.c
[orestis@perka tutorial]$ ls -ltr
total 4
-rw-r--r-- 1 orestis orestis 17  5 mar 11:13 premierfichier.c
```

Commandes et concept un peu plus avancés

Il existe une **grande quantité** de fonctionnalités non discutées ici:

1. `git branch`
2. `git merge`
3. `git tag`
4. `git rebase`

ET SURTOUT:

5. `git trois-lignes-de-commandes-incompréhensibles-que-seul stackoverflow-peut-vous-permettre-d'écrire`

Le fichier `.gitignore`

L'état des fichiers

Git voit les fichiers dans trois états possibles:

1. *tracked*, un fichier qui a été `add` (`staged`) ou `commit` (dans la terminologie git).
2. *untracked*, un fichier qui n'a pas été `add` ou `commit`.
3. *ignored*, un fichier qui est explicitement ignoré par git.

Certains fichiers ne doivent pas être `addables`

- Ils doivent *explicitement* être ignorés.

Quels fichiers ignorer

On ignore typiquement:

- Les fichiers binaires: exécutables, images, ...
- Les produits de compilation: `*.o`, `*.pyc`, ...
- Les produits d'exécutions: logs, ...
- Les fichiers de configuration d'un IDE: .vscode, ...
- Les fichiers système.

Comment ignorer des fichiers?

- Créer un fichier texte nommé `.gitignore`.
- L'ajouter au répo git et le "commit".
- Y ajouter les règles à suivre pour ignorer les fichiers.

Exemple: [1]

```
biden       # ignore le fichier biden
*.o         # ignore tous les fichier `.o`
!trump.o    # mais PAS trump.o
sanders     # ignore le répertoire sanders
**/sanders # ignore tous les répertoires sanders
```

[1] Pour une liste plus exhaustive voir le site https://bit.ly/2HTZJyQ par exemple.

Des références

Il existe énormément de très bons documents et tutoriels en ligne:

- https://git-scm.com/
- https://try.github.io/

Des tas de repo en ligne:

- Githepia
- Github
- Gitlab

Et des GUI assez utiles:

- GitExtensions
- GitKraken

Des questions?

Figure 3: Internet wisdom.