

# LEPL1503/LSINC1503 - Cours 3

*O. Bonaventure, B. Legat, L. Metongnon*

☐ Full Width Mode    ☐ Present Mode

## ☰ Table of Contents

Stack overflow 🌟

Git

Merge

Markdown

## Rappel organisation du cours

.....

Rappel slides du premier cours

Permanence le vendredi de 13h à 15h les semaines où il n'y a pas de cours théorique

Le tuteur n'a pas le temps de résoudre les problèmes d'installation en séance

# Stack overflow

## Factorial

C (C17 + GNU extensions)

```
1 int factorial(int n) {
2     if (n <= 1)
3         return 1;
4     else
5         return n * factorial(n - 1);
6 }
7
8 int main () {
9     factorial(10);
10 }
```

Stack

main

Heap

C/C++ [details](#): none [default view] ▼

[Edit Code & Get AI Help](#)

→ line that just executed

→ next line to execute

< Prev

Next >

Step 1 of 41

Visualized with [pythontutor.com](#)

► Que se passe-t-il si on oublie `if (n <= 1) ?`

## Fibonacci

Cette implémentation de Fibonacci a une très mauvaise complexité mais elle illustre bien la dynamique de la stack qui grandit et rapetissit rapetissit au rythme des appels de fonctions.

C (C17 + GNU extensions)

Stack

Heap

```
1 int fibonacci(int n) {  
2     if (n <= 2)  
3         return 1;  
4     else  
5         return fibonacci(n - 1) + fib  
6 }  
7  
8 int main () {  
→ 9     fibonacci(6);  
10 }
```

main

C/C++ [details](#): none [default view] ▼

[Edit Code & Get AI Help](#)

→ line that just executed

→ next line to execute



< Prev



Next >

Step 1 of 76

Visualized with [pythontutor.com](https://pythontutor.com)






# Git

---

- Tutoriel  git disponible dans le syllabus
- Learn Git Branching interactively 

## Contexte

---

- Linus Torvalds crée  **git** en 2005
- A maintenant 87 % du market share les logiciels de gestion de versions
- Serveur open source  **GitLab** (utilisé par la forge)
  - GitLab CI permet de run les tests et build la documentation de façon **automatisé** et **reproductible** à *chaque nouveau commit*
- En 2014, Satya Nadella remplace Steve Ballmer en tant que CEO de  Microsoft
  - 2016: Release du logiciel open source VS Code 
  - 2016: Release de Windows Subsystem for Linux (WSL) → *Virtually* all OS are **UNIX** now!
  - En 2018, acquisition de 
    - Lancement de GitHub Actions (imitant GitLab CI) gratuit pour dépôts open source

## Typical workflow

---

Dans le projet, on vous demande d'utiliser le workflow Git classique (voir par exemple  **JUMP**):

- **Invariant** La branche `main` la dernière version
  - Elle doit passer tous les tests sans erreurs
  - Les changements incorporés dans `main` ne sont plus sujets à discussion
- Une branch correspond à **un changement**
  - Eviter de mettre plusieurs changements qui ne sont pas liés dans la même branch
  - Une branch n'est **pas** la dernière version du code d'un développeur

- Un développeur qui veut essayer un autre changement doit repartir de `main` et créer une nouvelle branche
- Un développeur qui veut aider au même changement peut commit sur la même branch.

### Warning:

Il faut faire la différence entre un *logiciel de gestion de versions* comme Git et les *versions de releases*.

- **Si l'invariant est respecté, tout nouveau commit dans `main` pourrait faire l'objet d'une nouvelle release mais**

## Le cycle de vie d'une branch

---

- Avant tout changement, on part de la dernière version de `main`:
  - `git checkout main`
  - `git pull origin main`
- Après tout changement, on crée une branche et on la push
  - `git checkout -b new_branch`
  - `git commit -am "Courte description"` ce message deviendra le titre du MR
  - `git push origin new_branch`
- Sur GitLab, on crée un merge request (MR) et on attend
  1. le résultat de GitLab CI
  2. les peer reviewing de nos pairs
- Pour résoudre les problèmes de CI ou les reviews, on ajoute des commits sur la branche
  - `git checkout new_branch` (plus besoin de `-b` car la branche existe déjà)
  - `git pull origin new_branch` au cas où d'autres l'ont changé
  - `git commit -am "Address review"` ce message a moins d'importance
  - `git push origin new_branch`
- Une fois que le CI est vert et que nos pairs on accepté, on merge le MR

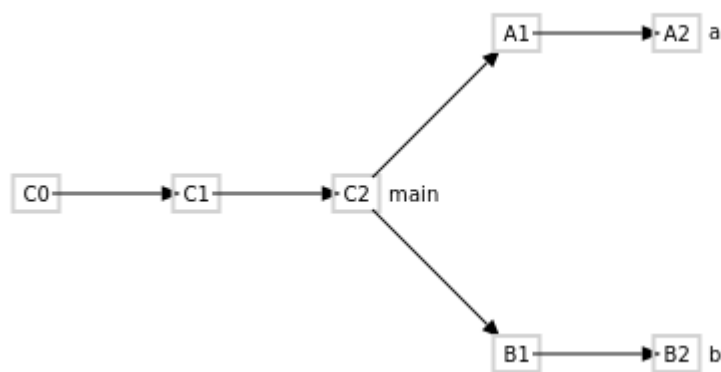
# Merge

---

Source des images

## Deux branches en cours...

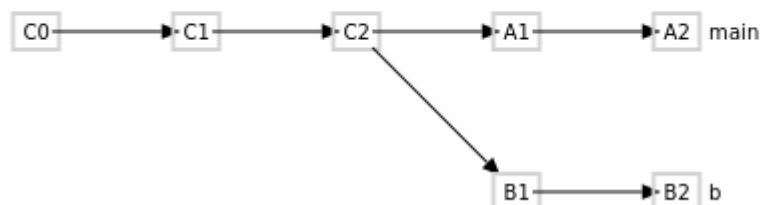
.....



## Une des deux est mergée en premier

.....

```
$ git checkout main  
$ git merge a
```



Comme main était un ancêtre de a, c'est un *fast forward* merge. On aurait pu faire

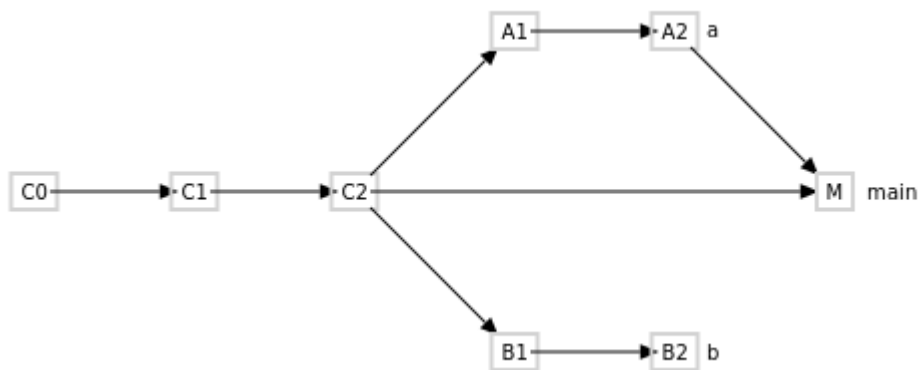
```
$ git merge --ff-only a
```

pour qu'il envoie une erreur si ce n'est pas fast forward.

## Un merge explicite

---

```
$ git checkout main  
$ git merge --no-ff a
```

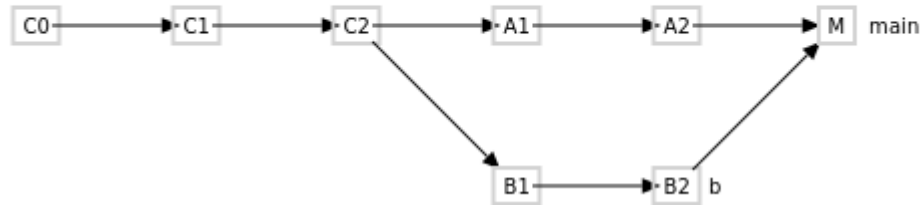


**Important** Comme on était sur la branche `main`, on ne modifie **que** `main`. Remarquez que `a` n'a pas bougé! Ceci est toujours vrai sur `git`, vous ne modifiez que la branche sur laquelle vous êtes `checkout`.

## Si l'autre branche est prête, on la merge

---

```
$ git checkout main  
$ git merge b
```



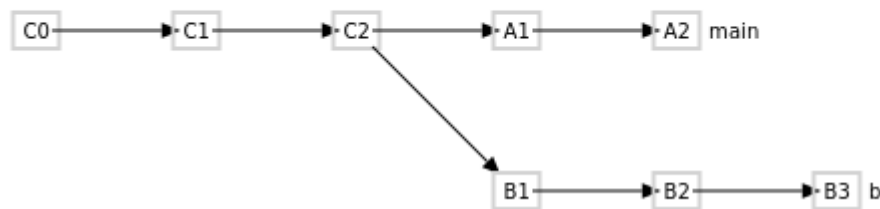
Ici, le merge ne peut pas être fast forward donc

```
$ git merge --ff-only b
```

enverrait une erreur.

## Si l'autre branche n'est pas prête, que faire ?

► Peut-on continuer à commit sur l'autre branche sans synchroniser ?

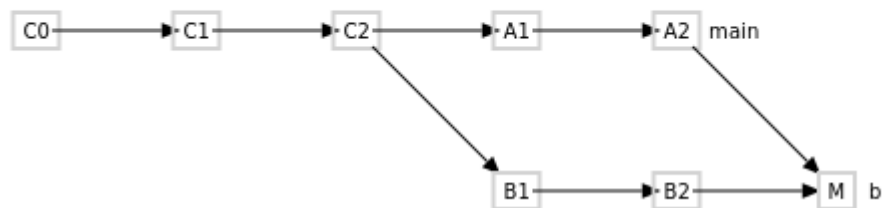




## Avant de continuer, on synchronise

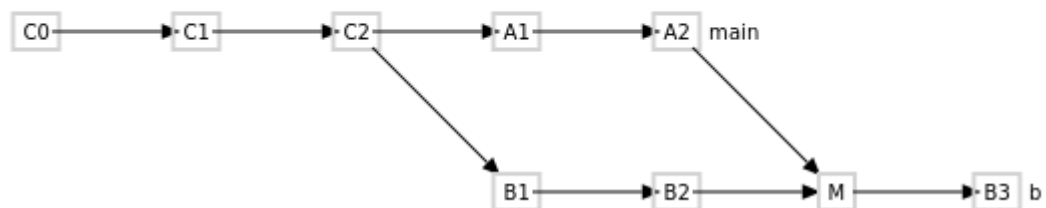
---

```
$ git checkout b  
$ git merge main
```



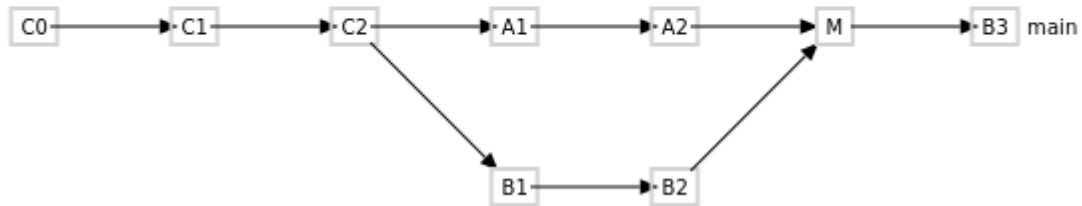
## On peut ensuite continuer

---



## Quand la seconde branche est finie

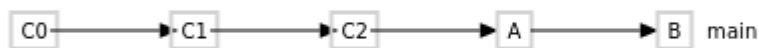
---



## Bouton merge sur GitHub/GitLab

---

La branche main du slides précédente contient des détails internes aux branches a et b . On aimerait plutôt avoir un commit par branche avec un lien vers le merge request correspondant pour voir plus de détails le cas échéant.



# Squash commits



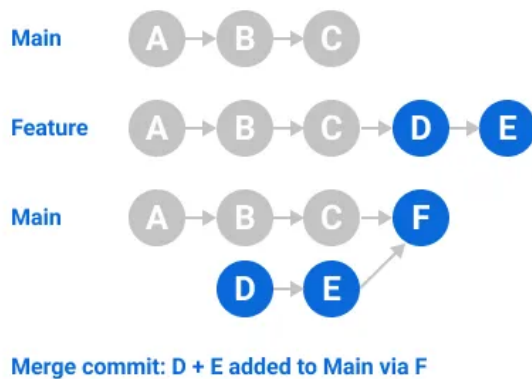
Ready to merge!

☐ Squash commits [?](#) ☐ Edit commit message

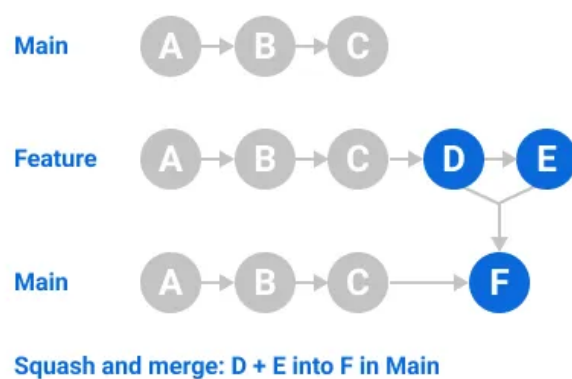
The source branch is [2 commits behind](#) the target branch. · 4 commits and 1 merge commit will be added to main.

Merge...

Sans Squash commits



Avec Squash commits



[Source des images](#)

## Reset

Si vous avez sans faire exprès commit sur main, faites

```
$ git checkout -b new_branch  
$ git checkout main
```

### Beware!

Vos changements sont maintenant sur `new_branch` donc vous pouvez les retirer de `main`. Mais il faut faire très attention car les commandes suivantes peuvent vous faire perdre des changements. Je conseille de faire `git push origin new_branch`, créer un merge request et

vérifier visuellement dans l'interface GitLab que vos changements sont là. Ensuite, 2 possibilités. Soit

```
$ git reset --hard origin/main # Dangereux command!
```

Soit en deux temps (en s'assurant qu'on est à la racine du dossier, sinon utiliser `..` ou `../..` etc... à la place de `.`)

```
$ git reset HEAD~1 # Annule le dernier commit, en supposant qu'il y en ai q  
u'un  
$ git checkout . # Dangereux, ça écrase aussi les changements qui n'ont pas  
été commit
```

## Stash changes

---

On est parfois dans la mauvaise branche et on veut appliquer nos changement dans une autre ou nouvelle branche. On doit alors faire `git checkout` mais `git` ne voudra pas si on a changé des fichiers qui différent (même si c'est à des lignes différentes). On `stash` alors les changements. Conceptuellement la même chose que `cherry-pick` 🚁 mais sans faire de commit.

```
$ git stash  
$ git checkout main  
$ git stash apply
```

# Markdown

---

- Lors du peer review, l'utilisation correcte du Markdown sera prise en compte dans la cotation
- Markdown est un *markup language* (comme LaTeX ou HTML) inventé par John Grubber en 2004,
- Il a aujourd'hui différentes variantes qui l'étende ou interprète différemment des ambiguïté dans sa définition de 2004.
- En 2014, CommonMark est publié pour but de standardiser ces extensions et clarifier les ambiguïtés.

## Exemple avec CommonMark

---

Aligné à gauche	Aligné à droite	Aligné au centre
Row 1	Column 2	<u>1</u>
Row 2	Row 2	Column

[1]:

Footnote

# Syntax highlighting

---

```
int i = 0;
```

produit

```
<code class="language-c">
int i = 0;
</code>
```

```
1 md"""
2 ```c
3 int i = 0;
4 ```
5 produit
6 ```html
7 <code class="language-c">
8 int i = 0;
9 </code>
10 ```
11 """
```

```
int i = 0;
```

```
1 html"""<code class="language-c">int i = 0;</code>"""
```

## Permalink

---

Un lien vers des lignes spécifiques du code doivent être liées à un commit spécifique car une branche évolue avec le temps. [GitHub affiche un code snippet](#) comme ci-dessous mais pas encore sur GitLab



octocat commented now

...

Let's update this trigger for the CodQL analysis workflow.

[octo-repo/.github/workflows/codeql-analysis.yml](#)

Lines 14 to 22 in 6860375

```
14   on:
15     push:
16       branches: [ main, add-emoji, branch-, branch-"#", enterprise-2.*-release, line
17     pull_request:
18       # The branches below must be a subset of the branches above
19       branches: [ main ]
20       reuse-previous-outcome: true
21     schedule:
22       - cron: '32 11 * * 5'
```



Activating project at '~/work/LEPL1503/LEPL1503/Lectures'

