SCM avec Git

Salah Gontara
2022-2023

Source Code Management

- La gestion du code source (SCM) est utilisée pour suivre les modifications apportées à un référentiel de code source.
- SCM suit un historique des modifications apportées à une base de code et aide à résoudre les conflits lors de la fusion des mises à jour de plusieurs contributeurs.
- SCM est également synonyme de contrôle de version.

Codons à plusieurs !

Versionner son projet avec Git

Pourquoi utiliser git ?

Objectifs :

- **travailler à plusieurs sans se marcher dessus** : indispensable pour les projets en équipe
- **garder un historique propre de toutes les modifications** : on organise son travail sous forme de “commits” documentés

La théorie de git

3 zones, 3 ambiances

Les modifications sont sauvegardés 3 fois

Working directory	Index	Local repository
C’est la zone de travail : les fichiers tout juste modifiés sont ici	Zone qui permet de stocker les modifications séléctionées en vue d’être commitées	Code commité, prêt à être envoyé sur un serveur distant

Les commits

Commit : ensemble de modifications cohérentes du code

Un bon commit est un commit :

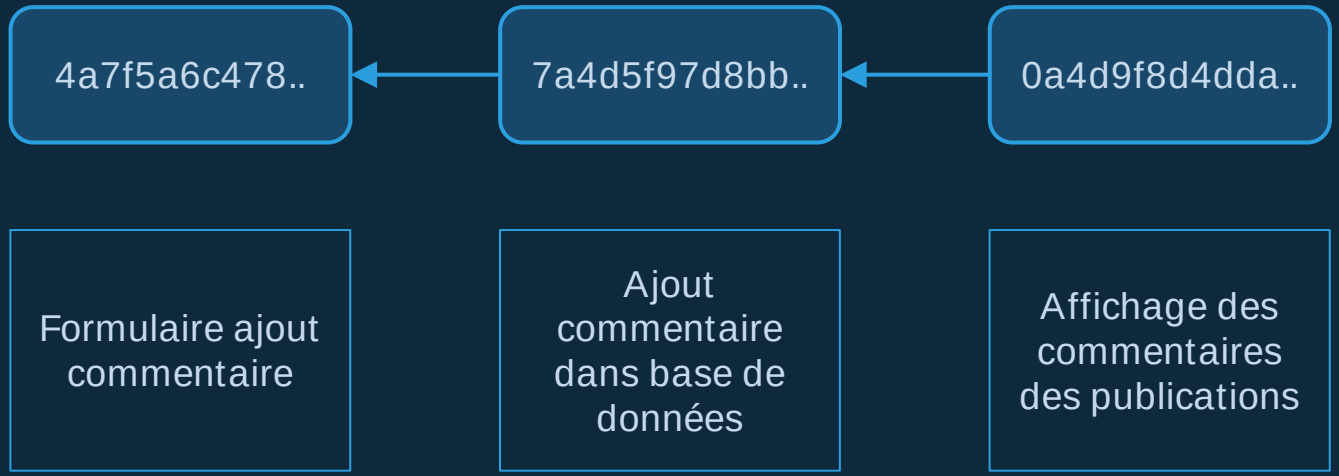
- qui ne concerne qu'une seule fonctionnalité du programme
- le plus petit possible tout en restant cohérent
- Idéalement qu'il compile seul

C'est quoi concrètement un commit ?

- une différence (ajout / suppression de lignes)
- des méta-données (titre, hash, auteur)

Les commits

Arbre de commits dans le git repository



Aller jusqu'au commit

3 zones, 3 ambiances

Les modifications sont sauvegardés 3 fois

Aller jusqu’au commit

Où j’en suis dans mes 3 zones ?

```
git status
```

```
On branch 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:   views/add_commentaire.php

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

Aller jusqu'au commit

Visualiser les différences entre le working directory et l'index

```
git diff
```

```
diff --git a/views/add_commentaire.php b/views/add_commentaire.php
index e69de29..8cff573 100644
--- a/views/add_commentaire.php
+++ b/views/add_commentaire.php
@@ -0,0 +1,6 @@
<?php include('header.php'); ?>

+<form action="/commentaire/ajouter" method="post">
+       <p>Pseudo : <input type="text" name="pseudo"/></p>
+       <p>Commentaire : <textarea name="commentaire"></textarea></p>
+</form>
```

Aller jusqu'au commit

Ajouter mes modifications à la zone de staging (index)

```
git add views/add_commentaire.php
git status
```

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

        modified:   views/add_commentaire.php
```

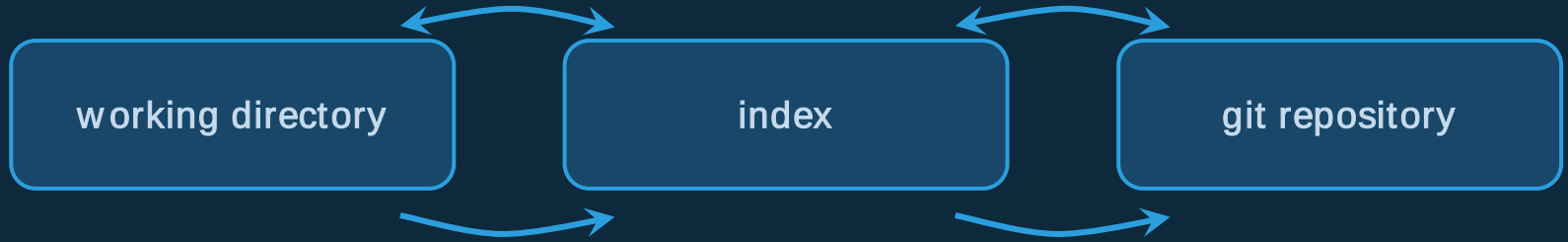
Aller jusqu'au commit

Récupitulatif des commandes

affichage différences

`git diff`

`git diff --staged`



ajouter des modifications

```
git add mon_fichier
git add -p (interactif)
git add -A (tout ajouter)
```

```
git commit -m “message”
```

Revenir au dernier commit

```
git status
```

```
On branch 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:   model/commentaires_model.php

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

Revenir au dernier commit

Enlever des modifications dans le working directory

```
git checkout -- monfichier
git status
```

```
On branch master
nothing to commit, working directory clean
```

Désindexer des fichiers

```
git status
```

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

        modified:   model/commentaires_model.php
```

Désindexer des fichiers

```
git reset HEAD monfichier
git status
```

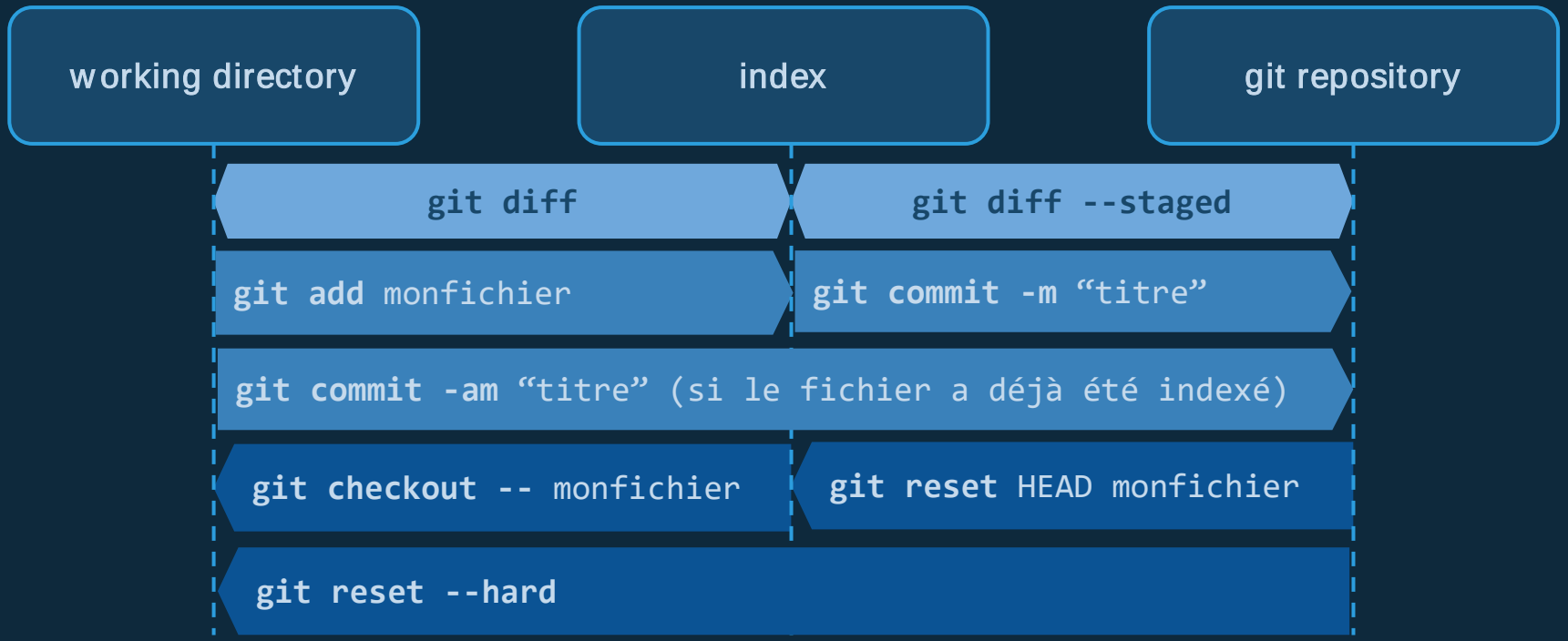
```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   model/commentaires_model.php

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

Aller jusqu'au commit

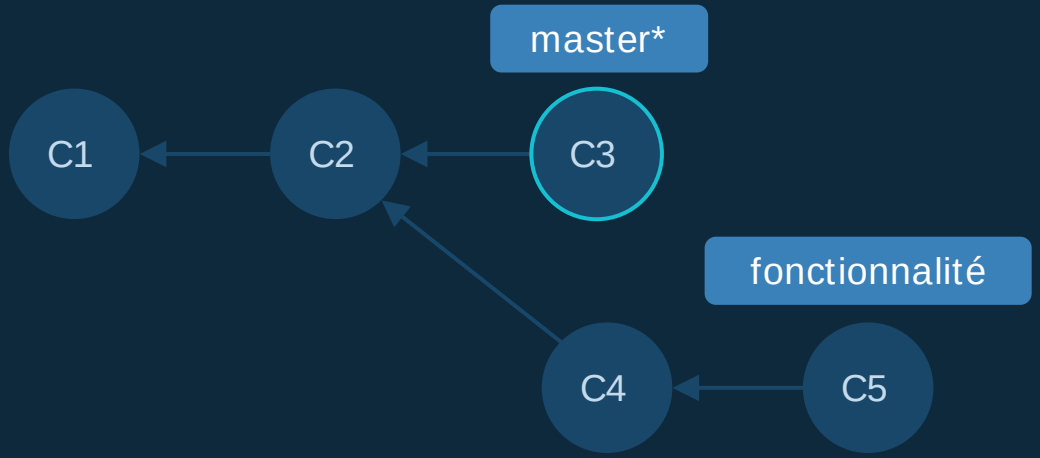
Récupitulatif des commandes



Les branches

- branche : *pointeur* vers un commit
- une branche principale : **master**
- Branche courante : **HEAD**
- en général, une branche par fonctionnalité en cours de développement

Les branches



Les branches et commits

Arbre de commits dans le git repository

Gestion des branches

Création et modification de branches

```
git branch : affichage des branches
```

```
git branch ma_branche : créer la branche ma_branche
git checkout ma_branche : déplace HEAD vers ma_branche
```

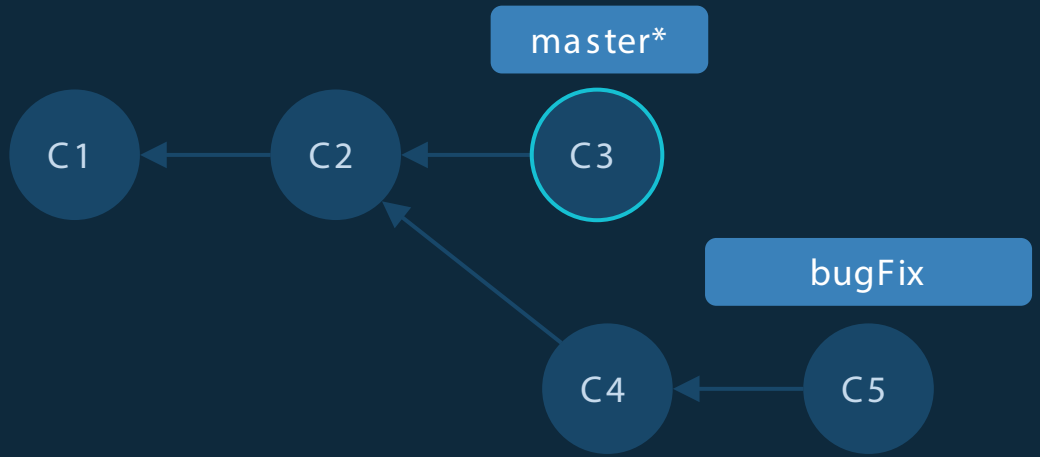
ou pour simplifier...

```
git checkout -b ma_branche : créer la branche
ma_branche et déplace HEAD dessus
```

Gestion des branches

Merge : intégration des modifications d'une branche dans la branche courante

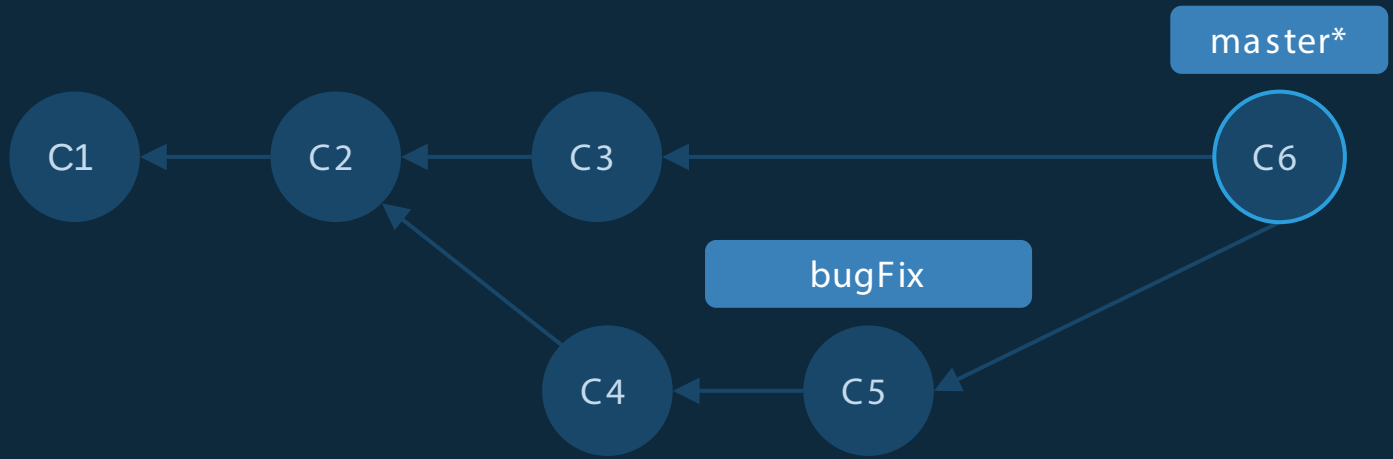
```
git merge ma_branche: merge ma branche dans la branche courante
```



Gestion des branches

Merge : intégration des modifications d'une branche dans la branche courante

```
git merge ma_branche: merge ma branche dans la branche courante
```



Les dépots distants

Centraliser les données sur un dépot git !

Les dépôts distants

- git directory sur un serveur distant pour le travail collaboratif

- Dépots distants :
  - github (pack étudiant)
  - **Gitlab (en cours)**

Pourquoi gitlab ?

- code review, merge request, interface web, ...

Voir et ajouter des dépots distants

Cloner un dépot distant : crée un dossier et récupère les fichiers

```
git clone <url>
```

Envoyer sur le dépot distant

```
git push
```

Envoie notre local repository sur le dépot distant

On ne touche plus aux commits pushés !

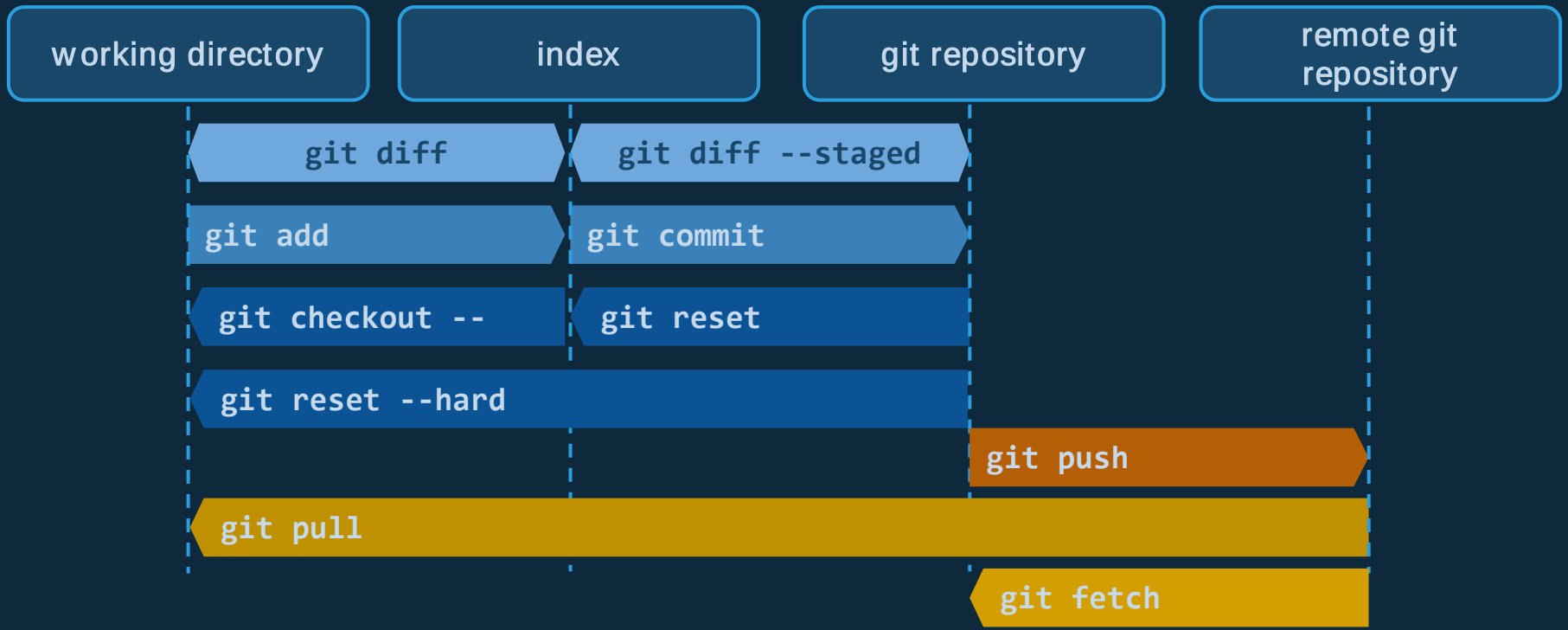
Recevoir depuis le dépot distant

```
git pull
```

Récupère les commits sur le dépot distant et met à jour le working directory

Le schéma de base de git

Récupitulatif des commandes



Installation de git

Sous windows :

- https://git-for-windows.github.io/
- http://babun.github.io/ (Emulateur de shell)

Sous OS X :

- http://sourceforge.net/projects/git-osx-installer

Sous Linux :

- <votre package manager> install git

(apt-get, rpm, ...)

Authentification par clé

Pour accéder aux dépots sur gitlab, il faut y ajouter sa clé

On génère une paire de clés :

```
ssh-keygen -t rsa -C " prenom.nom@polytechnicien.tn"
```

On affiche le contenu de la clé publique :

```
cat ~/.ssh/id_rsa.pub
```

On copie **tout** le contenu de la clé publique sur https://gitlab.com/eps_devops > mon profil > clés SSH

Configuration minimale

```
git config --global user.name "Prénom Nom"
git config --global user.email “prenom.nom@polytechnicien.tn”

En option mais c’est mieux :
git config --global color.ui true
git config --global color.diff.meta yellow
```

La zone de bordel : Stash

Le stash ça sert à :

- Sauvegarder les modifications du working directory dans une zone tampon pour rendre le working directory propre.
- Possibilité de rejouer les modifications stashées n'importe où
- Peut être vu comme une zone de brouillons

Stash : les commandes

On stash un ensemble de modifications

```
git stash
```

On récupères les modifications stashées

```
git stash apply
```

Pour plusieurs stashs :

```
git stash list
git stash apply stash@{id}
```

Effacer le contenu du stash

```
git stash clear
```

Pour aller plus loin avec git…

- **`git rebase`** ou **`git merge`** ?
- balader ses commits avec **`git cherry-pick`**
- afficher un commit : **`git show`** `<commit>`
- Engueulez vos amis : **`git blame`** `<fichier>`
- visualiser l'historique des commits : **`git log`**
- J'ai tout cassé ! **`git reflog`**