Programmation et projet encadré - L7TI005

Git : introduction

Yoann Dupont, Serge Fleury prenom.nom@sorbonne-nouvelle.fr
Pierre Magistry pierre.magistry@inalco.fr

2022-2023

Université Sorbonne-Nouvelle
INALCO
Université Paris-Nanterre

Les bases

Git et Github

Git et Github sont deux choses différentes :

- Git est un outil créé par Linus Torvald (créateur du noyau Linux)
- GitHub est un service web construit autour de Git créé par l'entreprise Github, Inc. (rachetée par Microsoft en 2018)
  - Offre un espace de stockage en ligne pour les dépôts Git
  - Mais aussi d'autres fonctionnalités absentes de Git seul

Git en bref

Git...

- ... est un **système de gestion de versions** ou SGV (en anglais *Version Control Software* ou *VCS*).

Git en bref

Git...

- ... est un **système de gestion de versions** ou SGV (en anglais *Version Control Software* ou *VCS*).
- ... permet de gérer les **modifications** effectuées sur un dossier données de manière **décentralisée**.

Git en bref

Git...

- ... est un **système de gestion de versions** ou SGV (en anglais *Version Control Software* ou *VCS*).
- ... permet de gérer les **modifications** effectuées sur un dossier données de manière **décentralisée**.
- ... ne versionne pas des fichiers, mais des ensembles ordonnés de modifications.

Git en bref

Git...

- ... est un **système de gestion de versions** ou SGV (en anglais *Version Control Software* ou *VCS*).
- ... permet de gérer les **modifications** effectuées sur un dossier données de manière **décentralisée**.
- ... ne versionne pas des fichiers, mais des ensembles ordonnés de modifications.

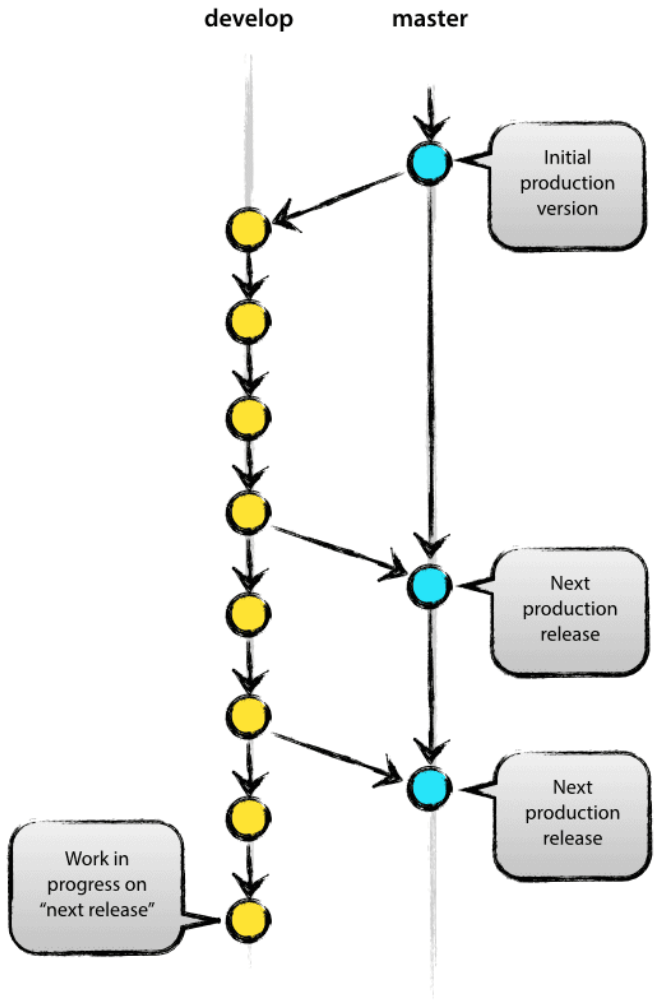

Figure 1: Une vision simplifiée d'un dépôt. À gauche, vos changements locaux, à droite, l'état de votre dépôt distant.

Les commandes git

Généralités

Git fournit un ensemble de commandes qui permettent de gérer les changements.

La syntaxe générale prend la forme suivante :

```
git <sous-commande> [-options...]  [arguments...]
```

Généralités

Git fournit un ensemble de commandes qui permettent de gérer les changements.

La syntaxe générale prend la forme suivante :

```
git <sous-commande> [-options...]  [arguments...]
```

Les commandes git sont paramétrables, on ne traitera ici que des cas de base.

En accord avec la philosophie Unix, les commandes git sont également très précises et font le moins de choses possibles.

git clone

Permet de créer une copie d'un dépôt existant sur sa machine.

Cette copie peut de suivre et de faire suivre les changements apportés au dépôt. On a donc un **clone** du dépôt distant.

```
git clone [-options...]  <URL>
```

Où `<URL>` est le lien vers un dossier git (généralement distant).

git clone

Permet de créer une copie d'un dépôt existant sur sa machine.

Cette copie peut de suivre et de faire suivre les changements apportés au dépôt. On a donc un **clone** du dépôt distant.

```
git clone [-options...]  <URL>
```

Où `<URL>` est le lien vers un dossier git (généralement distant).

Lier un dépôt de travail à son compte GitHub

En supposant que `USER` soit votre identifiant GitHub et `EMAIL` votre email :

```
git config user.name "USER"
git config user.email "EMAIL"
```

On peut mettre l'option `-global` pour ne pas faire ces deux lignes à chaque fois.

git pull

Permet de mettre-à-jour votre branche vers la bonne version (par défaut : la dernière version en ligne).

En termes git, on **tire** les changements du dépôt distant vers notre répertoire local.

```
git pull
```

git fetch

Permet de mettre-à-jour les métadonnées du dépôt sur votre répertoire local.

Le terme **fetch** fait sans doute référence à **play fetch** (jouer à la ramener balle).

```
git fetch
```

git fetch

Permet de mettre-à-jour les métadonnées du dépôt sur votre répertoire local.

Le terme **fetch** fait sans doute référence à **play fetch** (jouer à la ramener balle).

```
git fetch
```

`git fetch` a quelques avantages par rapport à pull :

- Fonctionne toujours (ajoute simplement plus de métadonnées).
- Ne modifie pas les fichiers du dépôt.
- Permet d'anticiper les conflits (changements incompatibles).

git add

Permet d'indiquer des fichiers dont on veut suivre les modifications avant validation.

En terminologie git, on **ajoute au suivi** des modifications faites sur des fichiers. On appelle cette étape la mise-en-place (*staging*).

```
git add <FILE...>
```

Où `<FILE...>` est un ou plusieurs fichiers dont on souhaite suivre les modifications

git add

Permet d'indiquer des fichiers dont on veut suivre les modifications avant validation.

En terminologie git, on **ajoute au suivi** des modifications faites sur des fichiers. On appelle cette étape la mise-en-place (*staging*).

```
git add <FILE...>
```

Où `<FILE...>` est un ou plusieurs fichiers dont on souhaite suivre les modifications

Attention :

`git add` suit les modifications **passées, mais pas futures**. Si vous modifiez un fichiers après un `add`, ces nouveaux changements ne seront pas suivis.

git rm

Permet de **retirer du suivi** (*remove*) un fichier. Cela équivaut à supprimer le fichier des futures versions.

```
git rm <FILE...>
```

Où `<FILE...>` est un ou plusieurs fichiers dont on souhaite supprimer du dépôt. Retirer un fichier via `git rm` met en place le changement automatiquement.

git rm

Permet de **retirer du suivi** (*remove*) un fichier. Cela équivaut à supprimer le fichier des futures versions.

```
git rm <FILE...>
```

Où `<FILE...>` est un ou plusieurs fichiers dont on souhaite supprimer du dépôt. Retirer un fichier via `git rm` met en place le changement automatiquement.

Attention :

- Un fichier retiré du dépôt peut être à nouveau ajouté par la suite.
- Un fichier retiré du dépôt ne disparaît pas totalement : il demeure accessible sur les versions précédentes.

git commit

Permet de valider les modifications des fichiers suivis.

Pourquoi *commit* ? De nouvelles modifications peuvent être faites après un commit, mais git **se tiendra** aux modifications validées.

```
git commit [-m message]
```

Où `message` est le message qui décrit les changements que vous faites.

git commit

Permet de valider les modifications des fichiers suivis.

Pourquoi *commit* ? De nouvelles modifications peuvent être faites après un commit, mais git **se tiendra** aux modifications validées.

```
git commit [-m message]
```

Où `message` est le message qui décrit les changements que vous faites.

Attention :

Si aucun message n'est donné, git ouvrira un éditeur de texte dans le terminal pour que vous puissiez écrire. Ça peut faire de mauvaises surprises.

git push

Envoie les modifications mises en place (ou "commitées") vers le dépôt distant.

On dit, en git, qu'on **pousse** les modifications.

```
git push
```

git status

Permet de voir les changements de votre dossier par rapport à la version du dépôt.

```
git status
```

Cela affichera les fichiers mis en place (*staged*) et les modifications non suivies.

git log

Permet de voir l'ensemble des *commits* ayant été effectués sur le dépôt du plus récent au plus ancien.

```
git log
```

Log vous indiquera notamment les auteur·ices, dates et messages des commits. Il faut appuyer sur la touche "q" pour sortir du log.

Mettre à jour un dépôt depuis votre ordinateur

Le travail sur un dépôt git suit une certaine routine. Le scénario le plus simple :

- `git pull`
- On fait des modifications...
- `git add` / `git rm`
- `git commit`
- `git push`

Et maintenant la pratique

Installer git

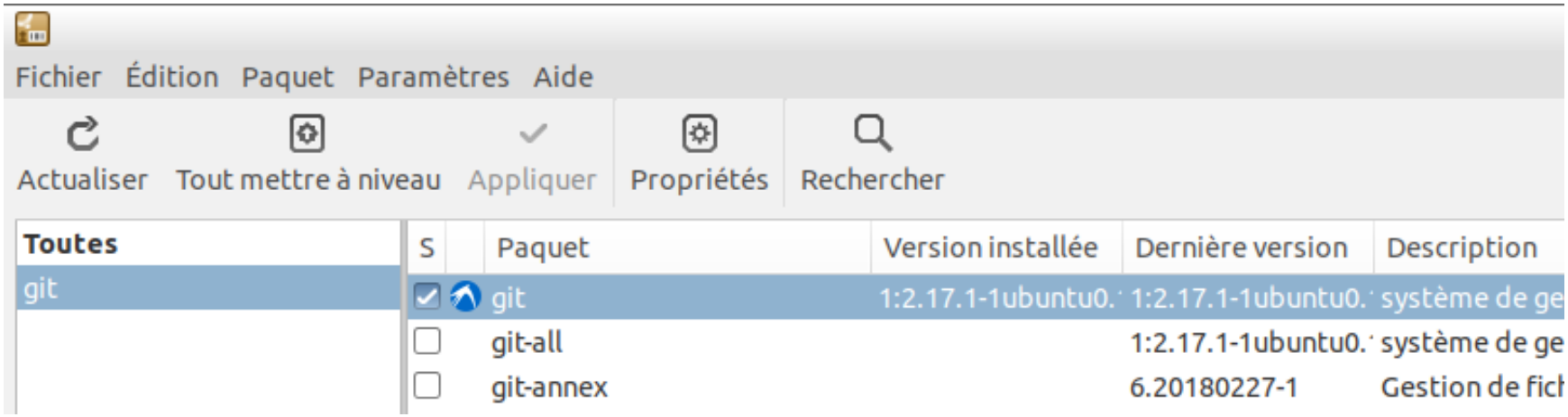

Figure 2: Allez dans le logiciel de gestion des programmes et sélectionnez tout simplement "git" pour installation.

git clone - Premier test

Petit test - récupérer les slides du cours

Créez-vous un dossier plurital quelque part **PAS SUR LE BUREAU**.

Allez dedans et tapez la commande suivante :

```
git clone https://github.com/YoannDupont/PPE.git
```

Inutile de lier vos identifiants à ce dépôt, vous ne pourrez pas pousser.