

Programmation et projet encadré - L7TI005

Git : un peu plus loin

Yoann Dupont, Serge Fleury prenom.nom@sorbonne-nouvelle.fr

Pierre Magistry pierre.magistry@inalco.fr

2022-2023

Université Sorbonne-Nouvelle
INALCO
Université Paris-Nanterre

Annonce



Si vous avez effectué votre inscription complémentaire à Sorbonne Nouvelle, vous avez un mail @sorbonne-nouvelle.fr et un accès à icampus.

Lien du cours :

<https://icampus.univ-paris3.fr/course/view.php?id=36409>

clé d'inscription (autoinscription) : PPE1@plurital

Addendum au cours précédent

Permet de voir les **différences** entre deux versions (typiquement votre machine).

```
git diff [FILES...]
```

Où [FILES...] est un ensemble de fichiers à comparer (facultatif). Si on ne donne rien, on compare tout le dossier.

- Les *issues* (problème ou question) permettent de discuter avec les mainteneurs d'un projet.

- Les *issues* (problème ou question) permettent de discuter avec les mainteneurs d'un projet.
- Crée un fil de conversation afin de résoudre le problème.

- Les *issues* (problème ou question) permettent de discuter avec les mainteneurs d'un projet.
- Crée un fil de conversation afin de résoudre le problème.
- Permet de garder une trace des bugs, propositions, futurs développements, etc.

- Les *issues* (problème ou question) permettent de discuter avec les mainteneurs d'un projet.
- Crée un fil de conversation afin de résoudre le problème.
- Permet de garder une trace des bugs, propositions, futurs développements, etc.
- Afin d'être sûr de ne pas louper votre dépôt, créez une issue dont le texte sera le lien vers votre dépôt sur le git du cours :
<https://github.com/YoannDupont/PPE>

- Les *issues* (problème ou question) permettent de discuter avec les mainteneurs d'un projet.
- Crée un fil de conversation afin de résoudre le problème.
- Permet de garder une trace des bugs, propositions, futurs développements, etc.
- Afin d'être surs de ne pas louper votre dépôt, créez une issue dont le texte sera le lien vers votre dépôt sur le git du cours :
<https://github.com/YoannDupont/PPE>
- onglet *issues* → *new issue* → entrer titre et contenu → cliquer sur *Submit New Issue*

GitHub et sécurité : la cryptographie asymétrique

GitHub est une plateforme collaborative.

- Du code, parfois très utilisé, circule
- Besoin de sécuriser le contenu des dépôts (usurpation, vandalisme, etc.)
 - Sécuriser des informations privées (chiffrer)
 - Authentification de l'auteur-ice d'un message (signer)
- Une solution est la cryptographie asymétrique

Aussi appelée Cryptographie à Clé Publique (*Public Key Cryptography*).

Pourquoi asymétrique ?

- Vous avez deux clés : une **publique** (que tout le monde peut voir) et une **privée** (qui doit n'être accessible qu'à vous).
- Deux usages :
 1. Chiffrez un message avec votre clé publique, déchiffrez avec votre clé privée
 2. Chiffrez un message avec votre clé privée, sera authentifié avec votre clé publique

Cryptographie asymétrique : illustration

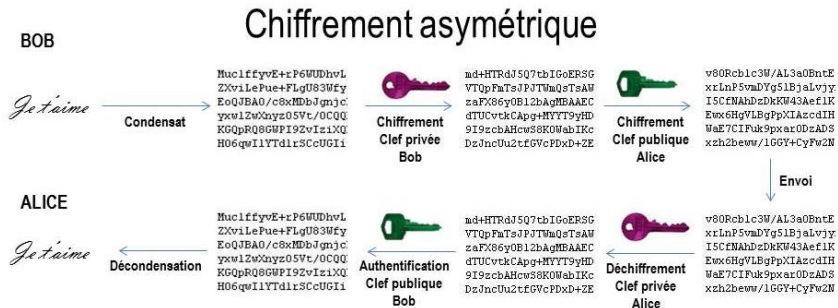


Figure 1: Source : [Wikipédia](#). Auteur : Jp.simon.manz (CC BY-SA)

Utiliser la cryptographie asymétrique dans GitHub

GitHub devient de plus en plus strict avec la sécurité :

- Vous aurez sans doute besoin d'une clé pour pousser vos changements.
- Type de chiffrement recommandé : Ed25519 ("historiquement" : RSA).
 - Ed25519 est en temps constant \implies timing pas exploitable
 - Ed25519 résiste mieux aux collisions \implies plus difficile de falsifier une clé donnant le même résultat

Utiliser la cryptographie asymétrique dans GitHub

GitHub devient de plus en plus strict avec la sécurité :

- Vous aurez sans doute besoin d'une clé pour pousser vos changements.
- Type de chiffrement recommandé : Ed25519 ("historiquement" : RSA).
 - Ed25519 est en temps constant \implies timing pas exploitable
 - Ed25519 résiste mieux aux collisions \implies plus difficile de falsifier une clé donnant le même résultat

Comment utiliser ces clés sur GitHub ?

1. Créer la paire clé publique/privée sur votre ordinateur
2. Donner la clé **publique** à GitHub


```
ssh-keygen -t ed25519 -C "<github email>"
```

Où <github email> est le mail que vous utilisez sur GitHub. Suggestion de github, mais vous pouvez mettre autre chose → -C n'est qu'un commentaire.

source : <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

```
ssh-keygen -t ed25519 -C "<github email>"
```

Où <github email> est le mail que vous utilisez sur GitHub. Suggestion de github, mais vous pouvez mettre autre chose → -C n'est qu'un commentaire.

La commande vous demandera alors :

1. Où sauvegarder la clé (par défaut la clé privée dans ~/.ssh/id_ed25519 et la publique dans ~/.ssh/id_ed25519.pub)
2. Un mot de passe pour la clé (qu'il faudra entrer à chaque fois)
 - 2.1 On peut ne rien donner, mais pas recommandé

source : <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

Ajouter la clé à votre agent SSH

Vérifier que votre agent SSH tourne :

```
$ eval "$(ssh-agent -s)"
```

source : <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

Ajouter la clé à votre agent SSH

Vérifier que votre agent SSH tourne :

```
$ eval "$(ssh-agent -s)"
```

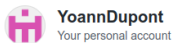
Affichera un message type "Agent pid XXXX". Si tel est le cas, on ajoute la clé à l'agent (en supposant que ~/.ssh/id_ed25519 est votre clé privée) :

```
ssh-add ~/.ssh/id_ed25519
```

source : <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

Ajouter la clé à GitHub

Finalement, on ajoute la clé **publique** (termine par .pub) à GitHub via les *settings*.



Public profile

Account

Appearance

Accessibility

Notifications

Access

Billing and plans

Emails

Password and authentication

SSH and GPG keys

Organizations


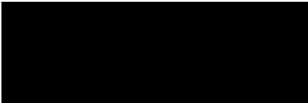

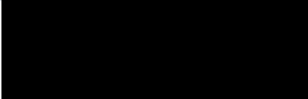
Moderation

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication Keys

 SSH		Delete
 SSH		Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

source : <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account>

GitHub : Les étiquettes

Quoi et pourquoi

Dans la longue chaîne des modifications, il peut être difficile de savoir si des commits sont plus intéressants que d'autres.

- Les étiquettes (*tag*) permettent de marquer un commit particulier.
- La valeur et la signification du tag dépend uniquement des mainteneurs.
- GitHub transforme cependant les tags en "*release*", ce qui est un parti pris.

source : <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

Quoi et pourquoi

Dans la longue chaîne des modifications, il peut être difficile de savoir si des commits sont plus intéressants que d'autres.

- Les étiquettes (*tag*) permettent de marquer un commit particulier.
- La valeur et la signification du tag dépend uniquement des mainteneurs.
- GitHub transforme cependant les tags en "*release*", ce qui est un parti pris.

Utilisation des tags dans ce cours :

Vous utiliserez les tags pour nous indiquer les versions finies de vos travaux pour que nous allions les vérifier.

source : <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

On peut ajouter un tag à un commit :

```
git tag [-a] [-m message] <tagname> [commit]
```

- `tagname` est le nom du tag (seul élément non-optionnel).
- `[commit]` indique le commit qu'on veut tagger (sinon, le commit courant).
- `-a` permet d'annoter un tag avec un message donné par `-m`.

source : <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

On peut ajouter un tag à un commit :

```
git tag [-a] [-m message] <tagname> [commit]
```

- `tagname` est le nom du tag (seul élément non-optionnel).
- `[commit]` indique le commit qu'on veut tagger (sinon, le commit courant).
- `-a` permet d'annoter un tag avec un message donné par `-m`.

Pousser un tag vers GitHub :

```
git push origin <tagname>
```

source : <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

GitHub : Corriger des erreurs

Quels moyens pour quelles erreurs.

Il est normal de faire des erreurs en git. En fonction de différents cas, on va vous apprendre à les corriger.

Quels moyens pour quelles erreurs.

Il est normal de faire des erreurs en git. En fonction de différents cas, on va vous apprendre à les corriger.

Nous allons utiliser les commandes pour cela :

- `git reset`
- `git revert`
- `git commit`

Un peu de syntaxe avant

Quelques éléments à savoir avant de continuer :

- HEAD : représente le commit sur lequel vous êtes en train de travailler
- <tag> : représente le commit sur lequel on a placé l'étiquette
- ~ [N] : représente l'ascendance directe de votre commit (linéaire, par défaut N=1 représente le commit parent)
- ^ [N] : représente le n-ième parent du commit (non linéaire, par défaut N=1 représente le commit parent)

source : <https://git-scm.com/docs/git-rev-parse>

Un peu de syntaxe avant

Quelques éléments à savoir avant de continuer :

- HEAD : représente le commit sur lequel vous êtes en train de travailler
- <tag> : représente le commit sur lequel on a placé l'étiquette
- ~[N] : représente l'ascendance directe de votre commit (linéaire, par défaut N=1 représente le commit parent)
- ^[N] : représente le n-ième parent du commit (non linéaire, par défaut N=1 représente le commit parent)

On peut faire des choses très précises, on se contentera de travailler ici dans l'ascendance directe.

source : <https://git-scm.com/docs/git-rev-parse>

Défaire jusqu'à des commits non poussés (gentil)

```
git reset HEAD~
```

Revient à dernière la version du dépôt et annule la mise-en-place (*staging*).

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Défaire jusqu'à des commits non poussés (gentil)

```
git reset HEAD~
```

Revient à dernière la version du dépôt et annule la mise-en-place (*staging*).

```
git reset --soft HEAD~
```

Revient à dernière la version du dépôt mais n'annule la mise-en-place (*staging*).

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Défaire jusqu'à des commits non poussés (gentil)

```
git reset HEAD~
```

Revient à dernière la version du dépôt et annule la mise-en-place (*staging*).

```
git reset --soft HEAD~
```

Revient à dernière la version du dépôt mais n'annule la mise-en-place (*staging*).

Attention :

`git reset` fonctionne sur des commits entiers, pas sur des fichiers spécifiques.

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Défaire jusqu'à un commit non poussé (méchant)

```
git reset --hard
```

Revient à la version HEAD. Vous perdrez tous les changements que vous avez fait.

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Revenir à un commits spécifique

```
git reset <commit>
```

Où <commit> peut être :

- l'identifiant SHA du commit (longue chaîne de lettre et nombres).
- un tag

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Revenir à un commits spécifique

```
git reset <commit>
```

Où <commit> peut être :

- l'identifiant SHA du commit (longue chaîne de lettre et nombres).
- un tag

Les options `soft` et `hard` s'appliquent comme précédemment.

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>

Défaire un commit déjà poussé

```
git revert <commit>
```

Où <commit> peut être :

- l'identifiant SHA du commit (longue chaîne de lettre et nombres).
- un tag

Crée un nouveau commit où les changements sont annulés.

source : <https://til.bhupesh.me/git/how-to-undo-anything-in-git>