Programmation et projet encadré - L7TI005

Git : un peu plus loin

Yoann Dupont yoann.dupont@sorbonne-nouvelle.fr
Pierre Magistry pierre.magistry@inalco.fr

2022-2023

Université Sorbonne-Nouvelle
INALCO
Université Paris-Nanterre

Annonce

Inscription sur icampus

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=45893

clé d'inscription (autoinscription) : `PPE1@2324`

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`

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 défault N=1 représente le commit parent)
- `^[N]` : représente le n-ième parent du commit (non linéaire, par défault 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 défault N=1 représente le commit parent)
- `^[N]` : représente le n-ième parent du commit (non linéaire, par défault 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 lettres 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