Versionning

Git

Ivan Canet · 28 janv. 2018 (28 janv. 2018)

Git est un outil nécessaire pour une grande partie des projets, nous allons présenter ici la majorité des commandes les plus utilisées. Ce document n'explique pas comment ces commandes fonctionnent, il est donc conseillé d'avoir une connaissance minimale de Git.

TABLE DES MATIÈRES

I. CRÉER UN DÉPÔT

1. Dans un dossier existant

```
git init
```

2. En tant que serveur (pas de code en local)

```
mkdir PROJET.git
cd PROJET.git
git init --bare
```

3. Récupérer un dépôt existant

Syntaxe

```
git clone <HTTP | GIT | SSH>
```

Exemples

```
git clone http://…/projet.git
git clone ssh://user@domain/p.git
```

II. MISES À JOUR

1. Local

1.1. Décider de ce qui va être pris en compte

1.1.1. Voir ce qui a été modifié

Fichiers modifiés

```
git status
```

Permet de voir les modifications de fichiers depuis le dernier commit : fichiers ajoutés, fichiers supprimés…

Les fichiers qui ne seront pas pris en compte sont affichés en rouge, les fichiers qui seront enregistrés sont affichés en vert.

Contenu des fichiers modifiés

```
git diff [FICHIER]
```

Permet de voir les modifications effectuées, lignes par lignes, aux fichiers. Si aucun fichier n'est spécifié, affiche les modifications de tous les fichiers du projet.

1.1.2. Prendre en compte une modification

Faire un commit ne prends en compte que les fichiers désignés ; on doit d'abord les « ajouter ».

```
git add FICHIER1 FICHIER2 …
```

On notera que la notation * fonctionne.

De la même manière, si un fichier a été supprimé, on utilise :

```
git rm FICHIER1 …
```

1.1.3. Ne pas prendre en compte une modification

Il est possible d'ignorer les modifications d'un (ou plusieurs) fichiers. On utilise :

```
git checkout -- FICHIER1 FICHIER2 …
```

Cette commande a pour effet de remplacer le·s fichier·s par leur dernière version enregistrée.

1.2. Enregistrer l'état actuel

```
git commit [-a] [-m 'Message'] [FICHIERS…]
```

Détaillons chaque option :

Tous les fichiers

L'option `-a` signifie que l'on veut prendre en compte tous les fichiers déjà enregistrés dans un commit précédent.

Spécifier un message

Si l'on utilise pas l'option `-m`, la commande va ouvrir un éditeur de texte pour demander ce qu'est la description du commit. Pour gagner du temps, on peut utiliser la syntaxe suivante pour spécifier la description directement :

```
git commit -m 'DESCRIPTION'
```

Spécifier des fichiers

Il est possible de spécifier directement quels fichiers seront pris en compte sans utiliser `add` ou `rm` ; en les listant simplement à la fin du message de commit.

Modifier le dernier commit

Il est aussi possible d'utiliser `--amend` pour modifier le dernier commit (faute de frappe, etc). Attention, ne jamais utiliser cette commande si le commit a déjà été envoyé (avec `push`) : dans ce cas elle risque de créer une divergence entre la version locale et distante.

1.3. Historique du projet

On utilise la commande `log` pour voir l'historique du projet :

```
git log
```

Celle-ci a de nombreuses options intéressantes :

Voir les lignes modifiées

L'option `-p` permet de voir quelles lignes ont été modifiées entre chaque commits (de manière similaire à `git diff`).

Voir les statistiques

L'option `--stat` permet de voir, pour chaque fichier, combien de lignes ont été ajoutées ou supprimées.

Sélectionner une époque

Les options `--since="PERIODE"` et `--before="PERIODE"` permettent de sélectionner une période. Elle peut être spécifiée comme une date (`"2017-03-12"`) ou sous forme relative à la date du jour (`"2 years 1 day 3 minutes ago"`).

Uniquement un certain auteur

On peut spécifier un auteur grâce à --author="AUTEUR". Il est possible d'utiliser cette option plusieurs fois (pour spécifier plusieurs auteurs).

Utiliser les expressions régulières

On peut utiliser une expression régulière pour rechercher des commits : --grep="REGEX". Cette option peut aussi être utilisée plusieurs fois.

Par défaut, si plusieurs REGEX sont spécifiés, la commande filtrera ceux qui valident n'importe lequel d'entre eux (OU logique) ; on peut utiliser --all-match pour filtrer uniquement ceux qui les valident tous (ET logique).

Rechercher les utilisations

Il est possible de rechercher les modifications d'un mot-clef ; par exemple :

```
git log -S 'test'
```

va chercher tous les commits dans lesquels le mot « test » a changé de nombre d'occurrences. Cette option peut être utile pour savoir quand une variable ou une fonction a été créée.

Afficher plus joliment

On peut s'ajouter un alias pour afficher l'historique plus joliment :

```
echo alias glog="git log --graph --oneline --decorate=short --branches='*'" >> .bashrc
```

On peut ensuite utiliser :

```
glog
```

2. Changer de version

2.1. Aller voir une ancienne version

```
git checkout ID
```

L'ID est le sha-1 précisé par log.

2.2. Créer une branche à partir d'une ancienne version

```
git checkout -b BRANCHE ID
```

Cf Branches, page 6.

3. Distant

3.1. Récupérer

Récupérer les données sans modifier la version locale

```
git fetch
```

Incorporer à la version locale

```
git up
```

Raccourci

```
git pull
```

Cette commande effectue les deux précédentes.

3.2. Envoyer

```
git push
```

Attention ; si quelqu'un d'autre a modifié la version distante, le `push` peut être refusé. Il faudra alors `pull` puis régler les éventuels conflits.

III. BRANCHES

1. Lister les branches

Locales

```
git branch
```

Distantes

```
git branch -r
```

Toutes

```
git branch -a
```

2. Créer et changer de branche

2.1. Créer

```
git branch NOM
```

Attention, cette commande ne change pas la branche actuelle.

2.2. Changer

Rapidement

```
git checkout NOM
```

Attention, toutes les modifications non-committées dans la branche en cours seront perdues.

Sans risques

Permet d'aller visiter une autre branche sans faire de commit et sans perdre la version actuelle.

```
git stash
git checkout AUTRE-BRANCHE
… modifications dans l'autre branche
git checkout PREMIERE
git stash apply
```

2.3. Fusionner deux branches

Imaginons que nous avons créé une branche `TEST` à partir de `MASTER` ; voici comment fusionner les commits de `TEST` :

```
git checkout MASTER
git merge TEST
```

2.4. Supprimer une branche

Sans pertes

La branche ne sera supprimée que si son continu a été fusionner dans la branche d'origine.

```
git branch -d BRANCHE
```

Abandonner une branche

La branche sera supprimée même si elle n'a pas été fusionnée.

```
git branch -D BRANCHE
```

IV. ÉTIQUETTES

On peut utiliser des étiquettes pour nommer certains commits (nommer une version, etc).

1. Opérations simples

Ajouter

```
git tag -a NOM -m DESCRIPTION [ID]
```

Ajoute une étiquette à un commit. L'`ID` est le code `sha-1` affiché par `log`. Si aucune `ID` n'est précisée, ajoute le tag au dernier commit.

Supprimer

```
git tag -d NOM
```

Rechercher

```
git tag -l NOM
```

2. Retourner à une étiquette

2.1. Voir le code

```
git checkout ETIQUETTE
```

2.2. Créer une branche à partir de l'étiquette

```
git checkout -b BRANCHE ETIQUETTE
```

3. Envoyer les étiquettes

```
git push --tags
```

V. IGNORER DES FICHIERS

VI. MODULES

VII. ANNEXES

1. Bibliographie

2. Index