Présentation Git

Développeur Web

Signes & Formations

Promo 2023-2024

Introduction

GIT

GIT est le gestionnaire de version le plus populaire, utilisé dans de nombreux projets open-source comme AngularJS, NodeJS, Spring Framework, Bootstrap, Gradle, …

Il a été créé par Linus Torvalds (auteur du noyau Linux) en 2005. Sa particularité est d'être décentralisé, contrairement à SVN (Subversion).

Caractéristiques de GIT

- Dépôt décentralisé
- Travail en local possible
- Très rapide
- Bonne gestion des branches et conflits
- Protection des branches
- Syntaxe complexe

Installation

Installation et configuration

- Télécharger Git depuis : https://git-scm.com/downloads

- Lors de votre première utilisation de GIT deux paramètres doivent être configurés : votre nom et adresse email. Ceci peut être fait via les commandes suivantes avec le terminal (Git Bash sur Windows) :

```
git config --global user.name "<NOM prenom>"
git config --global user.email <mail>
```

- Afin d'éviter quelques conflits que nous pouvons rencontrer quand nous sommes amenés à travailler entre les OS différents

```
git config --global core.autocrlf false
```

Note : Ces configurations sont stockés dans un fichier **.gitconfig**

Installation et configuration

- Afin d'éviter les conflits fréquents que nous pouvons rencontrer quand nous sommes amenés à travailler entre plusieurs OS différents (Windows, Linux ou Mac)

```
git config --global core.autocrlf false
```

- Par défaut, l'éditeur de texte sur Git Bash est vi ou vim selon l'OS. Pour le modifier :

```
git config --global core.editor <executable editeur>
```

Premiers pas

Créer un dépôt

Chaque commande commence par le mot clé **git**

Ces commandes sont exécutées dans le répertoire où se trouve l'application.

Pour initialiser un dépôt, la commande utilisée est : **`git init`**

Lors de l'exécution de cette commande, un dossier caché **.git** est créé dans le répertoire courant. Il contient toutes les informations sur les versions de notre projet.

À partir de ce répertoire, d'autres commandes pourront être utilisées pour gérer nos différentes modifications.

Statut du dépôt

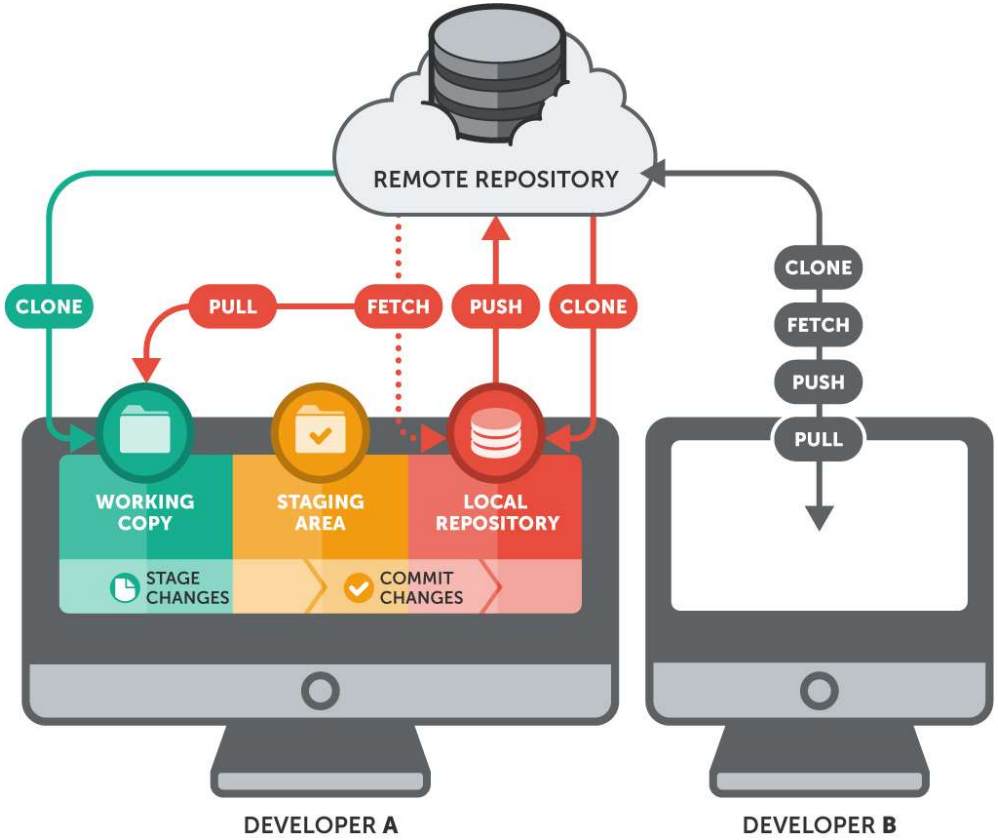
La commande **`git status`** permet de voir l'état global du dépôt GIT. Voici le résultat de cette commande après l'initialisation d'un dépôt :

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Cet exemple indique que nous nous trouvons sur la branche *master* et qu'aucune modification n'a été effectuée sur notre répertoire de travail.

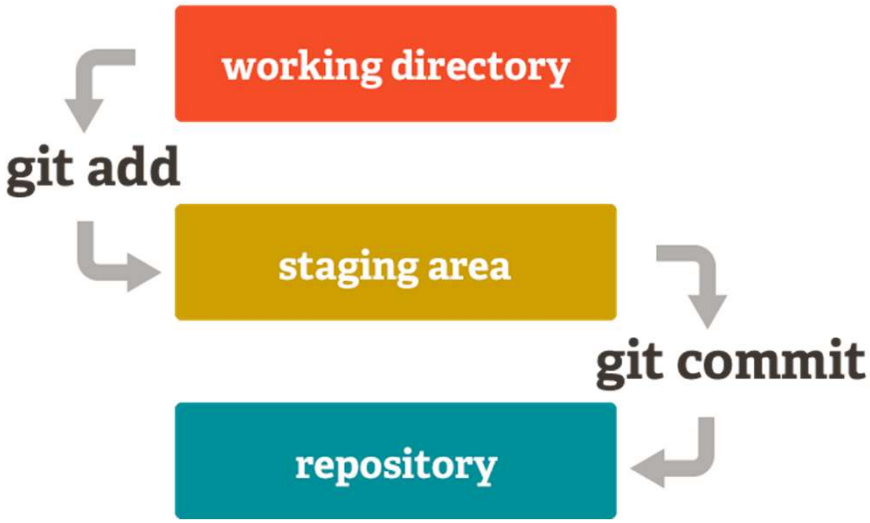
Nous verrons plus tard la notion de branches.

Fonctionnement de GIT



Source : https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction#start

Zoom sur la zone de travail



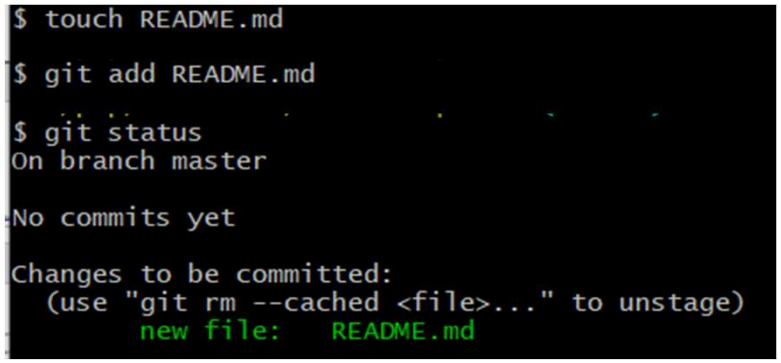
L'utilisation standard de Git se passe comme suit :

- vous modifiez des fichiers dans votre répertoire de travail
- vous indexez les fichiers modifiés, ce qui ajoute des instantanés de ces fichiers dans la zone d'index
- vous validez, ce qui a pour effet de basculer les instantanés des fichiers de l'index dans la base de données du répertoire Git.

Indexer des fichiers

Pour indexer un changement sur un fichier, la commande est : **`git add <file>`**

Un changement peut aussi désigner la création, la modification ou la suppression d'un fichier.
Exemple :



Pour désindexer un fichier nous pouvons utiliser la commande : **`git reset HEAD <file>`**

Valider une modification

Une fois les modifications placées dans la zone d'index, elles peuvent être validées grâce à la commande **`git commit`**. Elles sont alors stockées dans le dépôt local.

Les options conseillées sont les suivantes :

- L'option **`-m`** : Entrer un message de commit (si cette option n'est pas renseignée un éditeur de texte sera ouvert par défaut pour entrer le message).
- L'option **`-a`** : Indexer toutes les modifications (attention : cela n'indexe pas automatiquement les nouveaux fichiers créés).

Exemple :

```
$ git commit -am "add README"
[master (root-commit) d297def] add README
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

Chaque commit possède un identifiant (somme de contrôle SHA-1) unique. L'historique des commits peut être visualisé grâce à la commande : **`git log`**

Sauvegarder temporairement ses modifications

La commande **`git stash`** permet de sauvegarder temporairement nos modifications sans pour autant les commiter.

Cela permet de passer rapidement sur une autre branche/un autre commit puis revenir plus tard sur ces modifications. Même les fichiers indexés seront sauvegardés.

Exemple :

```
$ git stash
Saved working directory and index state WIP on master: d297def add README
```

Pour obtenir la liste des modifications enregistrées : **`git stash list`**

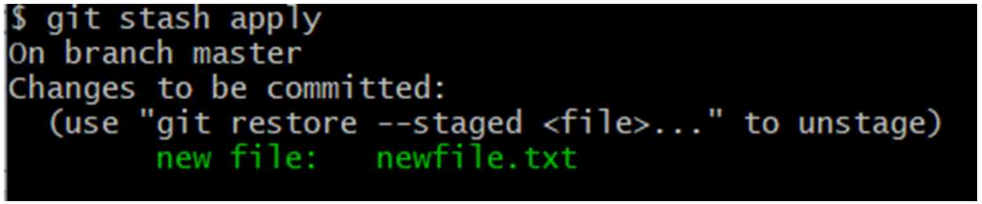
Exemple :

```
$ git stash list
stash@{0}: WIP on master: d297def add README
```

Récupérer ses modifications temporaires

Pour récupérer les modifications enregistrées avec la commande *git stash*, on peut utiliser le paramètre *apply* de la commande stash : **`git stash apply`**

Exemple



Gestion des branches

Gestion des branches

Une des particularités de GIT est sa facilité à gérer des branches de développements.

Habituellement, le workflow de développement est la suivante :

- Pour chaque nouvelle évolution, quelque soit sa taille, une branche locale est créée.
- Une fois l'évolution terminée, cette branche est fusionnée à la branche de développement principale.
- On s'assure ensuite que la branche reste stable après la fusion.

Git permet de changer de branche rapidement, ce qui est pratique pour « expérimenter » ou travailler sur plusieurs évolutions

Grace à GIT, la répartition des tâches lors du développement d'une application se fait aisément.

En effet chaque fonctionnalité est isolée dans une branche spécifique. On s'assure simplement que la branche principale reste stable après l'intégration des différentes fonctionnalités.

Gestion des branches

Une branche est une « ligne de développement », ou un ensemble de commits.

Imaginons le scénario suivant:

Je suis en train d'implémenter une nouvelle fonctionnalité sur la branche master (qui est la branche par défaut sous GIT). Mon n+1 me demande de corriger un bug urgent sur un précédent commit. Malheureusement, je suis obligé de terminer mes modifications avant de corriger ce bug…

La solution aurait été de séparer la nouvelle fonctionnalité sur une autre branche (par exemple « myFeature ») et de revenir sur la branche master afin de corriger le bug. Plus tard, je pourrais revenir sur ma nouvelle fonctionnalité et la fusionner avec la branche master pour l'implémenter dans l'application.

Gestion des branches

Exemple d'architecture de branches pour un projet :

- La branche **master** correspond aux versions stables de l'application.
- La branche **develop** est la branche sur laquelle sont faits les développements.
- Chaque **feature** est développée dans une branche indépendante pour ne pas impacter l'appli en développement.

Les commandes

- Une branche peut être créée avec l'une des 2 commandes :

```
git checkout -b <branche>
git branch <branche>
```

- La branche doit être déplacée dans le dépôt distant :

```
git push --set-upstream origin <branche>
```

- Pour se déplacer sur une branche, utilisez la commande:

```
git checkout <branche>
```

- Pour se déplacer sur un commit spécifique, utilisez la commande:

```
git checkout <id commit>
```

Les commandes

- Pour fusionner une branche avec la branche courante, utilisez

  `git merge <branche à fusionner>`

Attention : cette action peut générer des conflits… (voir diapo «Conflits de fusion »)

- Pour lister toutes les branches, utilisez `git branch`.

L'option `–v` permet de visualiser les dernières modifications sur chaque branche.

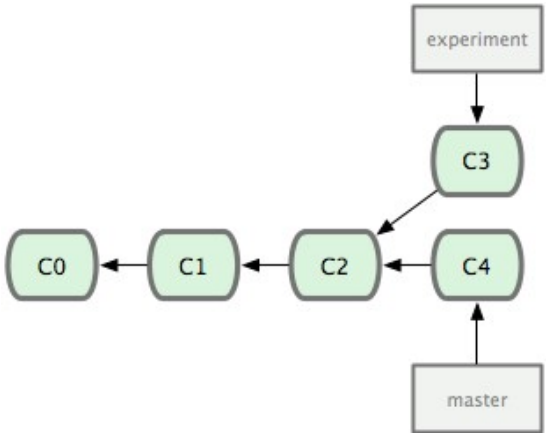
- Pour supprimer une branche, utilisez `git branch –d <nom branche>`

Ne pas utiliser `git branch –D` (suppression forcée) car vous risquez de perdre des commits !

Déplacer la base d’une branche

Il est souvent utile de déplacer la base d’une branche. Par exemple, cela peut servir à réécrire notre historique suite à une mauvaise manip, ou à éviter de fusionner deux branches alors qu’en déplacer une suffirai.

Prenons un exemple concret avec l’arbre suivant :



Déplacer la base d'une branche

Le moyen le plus simple de fusionner *experiment* et master serait de fusionner ces deux branches avec `git merge` :

Ça fonctionne, mais un commit de fusion viens s'ajouter à la liste des commits…

Déplacer la base d'une branche

Git permet de déplacer simplement notre branche *experiment* (et tous ses commits) à la suite de la branche *master* :

Et voila : nous avons un historique bien propre ! Toutes les modifications apparaissent en série même si elles ont eu lieu en parallèle…

Déplacer la base d’une branche

Voici la commande permettant de déplacer une branche :

```
git rebase <upstream> <branch>
```

Attention : ne rebasez jamais des *commits* qui ont déjà été poussés sur un dépôt public.

Cela modifierai l’historique de tous vos collègues…

Collaborer avec son équipe

Ajouter un dépôt distant

Pour collaborer avec notre équipe, nous avons besoin d'un référentiel commun sur lequel partager nos ressources. Nous utilisons pour cela un dépôt distant, que nous lions à notre dépôt local.

Nous pouvons alors envoyer des modifications sur le dépôt distant ou récupérer celles d'autres personnes.

- Pour ajouter un dépôt distant nous utilisons la commande :

```
git remote add <remote> <url>
```

Par défaut, le nom du dépôt distant *remote* est `origin`

- La commande `git remote -v` liste tous les dépôts distants avec leur URL.

- Pour supprimer un remote nous utilisons la commande :`git remote rm <remote name>`

Git clone

Il est possible de créer un dépôt local à partir d'un dépôt distant grâce à la commande :

```
git clone <url>
```

Cette commande crée un dépôt local dans un nouveau répertoire, contenant une copie du dépôt distant.

Le remote **origin** est automatiquement ajouté au dépôt local, faisant le lien avec le dépôt distant.

Envoyer des modifications

Une fois notre dépôt distant ajouté, nous pouvons lui envoyer les nouvelles modifications effectuées en local, pour les partager avec nos collègues.

L'ensemble des commits effectués en local peut être publié sur le dépôt distant avec la commande :

```
git push <remote> <branche>
```

Exemple : `git push origin master`

```
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 213 bytes | 106.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/DeafDam/testGit.git
 * [new branch]      master -> master
```

Récupérer des modifications

Nous aurons aussi besoin de récupérer en local les modifications publiées par d'autres personnes sur le dépôt distant.

Ceci peut être fait avec la commande `git fetch <remote> <branche>`

Exemple : `git fetch origin master`

Fonctionnement git pull

Il est aussi possible de récupérer puis fusionner automatiquement les modifications avec une commande très similaire :

```
git pull <remote name> <branch name>
```

Cette commande lance `git fetch` puis tente de fusionner automatiquement les nouvelles modifications avec nos modifications locales.

Exemple : `git pull origin master`

Lorsque nous avons effectué des modifications en local, il vaut mieux utiliser les commandes `git fetch` puis `git rebase`. Elles sont suffisantes dans la majorité des cas et évitent de surcharger l'historique avec des commits de fusion.

Récupérer des modifications (1/2)

Pour récupérer dans son dépôt local les modifications effectuées par son équipe sur le dépôt distant, on utilise la commande **`fetch`**.

La syntaxe est la suivante :

```
git fetch <remote> [<branche>]
```

Exemple : **`git fetch origin`**

Attention : Cela ne met pas à jour votre environnement de travail.

Pour fusionner ses modifications avec celles des autres, il faut faire un **`merge`**

Exemple :

```
git checkout version/03.01.00.00
git merge origin/version/03.01.00.00
```

Récupérer des modifications (2/2)

Il est aussi possible de récupérer puis fusionner automatiquement les modifications avec la commande :

`git pull <remote name>`

Exemple : `git pull origin`

Cela correspond à lancer `git fetch` puis `git merge`.

Remarque : peut générer des conflits

L'avantage du `fetch` est de pouvoir faire une différence entre branche avant de fusionner.

Exemple : `git diff version/03.01.00.00 origin/version/03.01.00.00`

Envoyer des modifications

Pour envoyer ses modifications vers le dépôt distant, on utilise la commande **`git push`**

Cela aura pour effet d'envoyer tous les commits locaux d'une branche vers le dépôt distant. La syntaxe est la suivante :

```
git push <remote> [<branche>]
```

Exemple : **`git push origin master`**

Outils graphique

Heureusement des logiciels permettent d'utiliser GIT avec une interface graphique pour une meilleure visibilité. Par exemple :

- **GITK** : l'outil par défaut fourni avec GIT.
- **TortoiseGIT** : Très populaire et facile d'utilisation.
- **SourceTree** : Très ergonomique.
- Etc…

Outils graphique côté serveur

Des sites proposent des outils graphiques et aussi de créer et d'héberger un serveur distant pour gérer les dépôts. Les plus connus sont :

- **GitLab**
- **GitHub**
- **Bitbucket**

GitLab

Bitbucket

GitHub

Tous ces sites sont des forges offrant un service de gestionnaire Git

Ce sont aussi des sites publics, **ne publier jamais des sources client sur ces sites !**

Gestion des conflits

Les conflits

Lorsque plusieurs personnes modifient simultanément le même fichier, des conflits peuvent survenir lorsque ses modification sont validées.

Il existe alors deux versions différentes du même fichier que GIT ne saura pas comment fusionner proprement.

Exemple :

*J'ai modifié le fichier index dans la branche **prob53**, que je veux fusionner avec la branche **master**. Problème : entre temps quelqu'un à modifié ce même fichier dans la branche master.*

➡ *CONFLIT DE FUSION !*

Conflits de fusion (1)

Voyons un exemple de conflit lors de la fusion de ces deux branches :

```
$ git merge prob53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

GIT n'a pas automatiquement créé le commit de fusion. Il a arrêté le processus le temps que vous résolviez le conflit. Lancez **`git status`** pour voir quels fichiers n'ont pas été fusionnés :

```
[master*]$ git status
index.html: needs merge
# 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)
#
#   unmerged:   index.html
#
```

Conflits de fusion (2/2)

Tout fichier qui comporte des conflits de fusion et n'a pas été résolu est listé comme **unmerged**. GIT ajoute des marques de conflits dans les fichiers à l'endroit de chaque conflit. Votre fichier contient alors des sections qui ressemblent à ceci :

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> prob53:index.html
```

Pour résoudre le conflit :

- Corriger le fichier, en prenant l'une ou l'autre des modifications ou en les fusionnant manuellement
- **`git add <fichier>`**
- **`git commit –m "un commentaire"`**

Bonnes pratiques

A adpater selon le workflow de votre entreprise

Recommandations

- Un commit représente un état stable de l'application : terminez vos modifications essentielles avant de les commiter
- Il vaut mieux beaucoup de *petits* commits contenant peu d'informations plutôt qu'un *gros* commit incompréhensible
- Utiliser `git fetch` avant `git push` permet de vérifier que notre dépôt est bien à jour avant d'envoyer nos modifications
- Ce n'est pas du temps perdu que de vérifier les modifications de chaque fichier avant de les commiter.
- Ne pas utiliser `git merge` quand `git rebase` suffit permet de garder un historique propre

Recommandations

- Rassembler les branches par groupes en utilisant le pattern <groupe>/<objet> au moment de leur création
  - Exemple :

Pour créer une branche évolutive

```
git checkout -b evo/add_user
```

Ou pour créer une branche version

```
git checkout -b version/03.01.00.00
```

Ou pour la résolution d'un bug en production :

```
git checkout -b fix/qc_235
```

Annexes

Plus d’info

Quelques sites sympa :

- https://try.github.io/ (ressources pour apprendre git) 
- http://rogerdudler.github.io/git-guide/ (Guide de démarrage) 
- https://www.git-tower.com/learn/git/ebook/ (Guide très complet et clair) 
- https://git-scm.com/documentation (Documentation officielle) 
- https://git-scm.com/download/gui/windows (liste des clients graphiques de git) 
- https://git-scm.com/book/fr/v2 (ebook officielle) 
- https://ndpsoftware.com/git-cheatsheet.html (Guide visuel des états et de leurs commandes)
- https://learngitbranching.js.org/ (jeu interactif et très visuel)

Merci pour votre attention