Tutoriel de découverte de Git

Eric Adjakossa, Tâm Le Minh

02/12/2021

`http://git-scm.com/book/fr/v2`

Prérequis :
— Avoir un compte GitHub (`https://github.com`)
— Avoir installé Git (`https://git-scm.com/book/fr/v2/Démarrage-rapide-Installation-de-Git`)

On utilisera Git en lignes de commande via le terminal sur Linux/Mac, avec Git Bash sur Windows.

1 Commencer

On va commencer par créer un dépôt via GitHub. Il est à noter que pour créer un nouveau dépôt à partir de zéro, il est aussi possible de le faire en local avec la commande `git init`.

Allez sur la page de votre profil GitHub et dans l'onglet Repositories, cliquez sur New. Indiquez `tp-git-happyr` pour le nom du dépôt. Vous pouvez choisir de rendre ce dépôt public ou privé (modifiable par la suite, mais cela n'a pas d'importance pour le TP). Cochez les case "Add a README file" et "Add .gitignore" (template R par exemple).

Le répertoire est désormais créé sur GitHub. Le fichier README peut être modifié pour fournir une présentation du dépôt qui est affichée sur GitHub quand on y accède. Le fichier .gitignore permet de demander à Git de ne pas prendre en compte certains fichiers présents en local mais qu'on ne veut pas inclure dans le dépôt (par exemple : sorties de programme, code compilé, fichiers .Rproj, etc.).

Pour récupérer le dépôt sur son ordinateur, il faut cloner le dépôt. Cliquez sur le bouton Code. Il y a deux manières principales de communiquer avec GitHub : en HTTPS ou SSH. Essayons d'abord en HTTPS. Cliquer sur HTTPS puis copiez l'adresse qui s'affiche. Sur votre terminal, placez-vous là où vous voulez mettre votre dépôt local, puis exécutez `git clone` suivi de l'adresse que vous avez copié.

```
cd ~\workspace
git clone https://github.com/tam-leminh/tp-git-happyr.git
```

Uniquement si HTTPS ne marche pas Certains OS n'autorisent pas la connexion en HTTPS. Si cette manipulation ne marche pas, il faut se connecter par SSH. Si ce n'est pas fait, il faut créer une clé SSH sur son ordi (`https://www.atlassian.com/git/tutorials/git-ssh`) et l'ajouter sur GitHub.

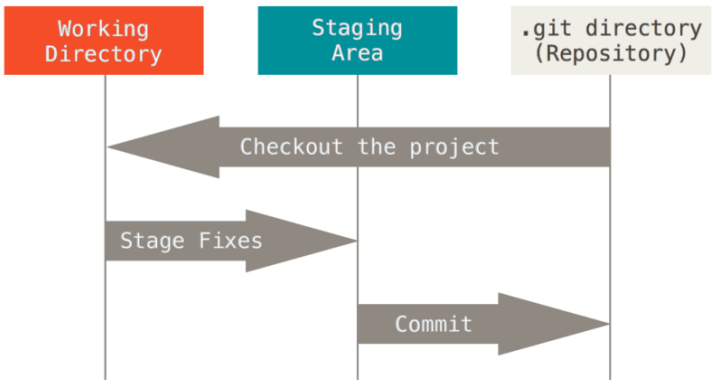
```
cd ~\workspace
git clone git@github.com:tam-leminh/tp-git-happyr.git
```

Un répertoire `tp-git-happyr` se retrouve maintenant là où vous avez fait le `git clone`. On peut explorer ce répertoire :

```
cd tp-git-happyr
ls
ls -al
```

On voit que ce répertoire contient le fichier README du dépôt, mais également un répertoire caché `.git`. Ce répertoire caché contient tous les informations du dépôt pour que Git le reconnaisse (il contient aussi notamment l'index et l'histoirique des commits, ce que nous verrons plus tard).

2 Explorer les états des fichiers



Exécutez `git status` une première fois pour voir que le dépôt est propre. Maintenant, commençons à manipuler le dépôt local. Créez un fichier vide nommé `helloworld.R`, puis vérifiez `git status`.

```
git status
touch helloworld.R
git status
```

Le status nous indique que le fichier `helloworld.R` n'est pas encore indexé. On va donc l'indexer, c'est-à-dire l'envoyer dans le staging area, avec `git add`.

```
git add helloworld.R
git status
```

Maintenant, nous validons les modifications de l'index en réalisant un commit. L'état de l'index validé devient la nouvelle référence pour `git status` : les changements validés n'apparaissent donc plus.

```
git commit -m "ADD empty helloworld.R file"
git status
```

Il est temps d'écrire quelque chose dans le fichier `helloworld.R`. Mettez dans `helloworld.R` une ligne simple : `print("helloworld")`. Pour ce faire, vous pourriez l'ouvrir avec votre éditeur de texte préféré, mais vous pouvez aussi utiliser la commande `echo` comme indiqué ci-dessous.

```
echo "print(\"helloworld\")" > helloworld.R
cat helloworld.R
git status
git add helloworld.R
git status
```

La modification du fichier est indexée grâce à la commande `git add`. Vous remarquez que `git status` vous donne des indications au cas où vous voudriez désindexer le changement avec la commande `git reset`.

```
git reset -- helloworld.R
git status
```

Vous remarquez donc que `git reset` est la commande "inverse" de `git add`.

3 Rétablir des modifs

Rajoutez donc la modification dans l'index et regardons ce qui se passe si on supprime le fichier indexé dans le working directory.

```
git add helloworld.R
rm helloworld.R
git status
```

`git status` indique 2 états du fichier : la modification du fichier qui est indexée et la suppression du fichier, mais cette dernière n'est pas encore indexée. Il vous suggère aussi des commandes en fonction de ce que vous voulez faire :

— si vous voulez restaurer le fichier de votre working directory avec la version indexée, vous utiliserez `git checkout`,

— si vous voulez au contraire indexer la suppression du fichier, il faut utiliser `git add`.

Essayons les deux...

```
git checkout -- helloworld.R
git status
rm helloworld.R
git add helloworld.R
git status
```

Après le `git checkout`, le fichier `helloworld.R` était bien restaurée. Après `rm` puis `git add`, la suppression du fichier est indexée.

À présent, si on voulait restaurer helloworld.R, on ne pourrait plus récupérer la version modifiée qui était précédemment indexée : en effet, la modification vient d'être écrasée par la suppression du fichier dans l'index. Le mieux qu'on puisse faire est de récupérer la version du dernier commit. Pour cela, on réinitialise l'index avec `git reset`, puis on fait `git checkout`.

```
git reset -- helloworld.R
git status
git checkout -- helloworld.R
git status
```

On en est donc revenu à la situation de notre dernier commit, c'est-à-dire avec un fichier `helloworld.R` vide.

4 Communiquer avec le dépôt distant

Afin d'interagir avec le dépôt distant sur GitHub, il deux commandes à connaître :
— `git pull` pour récupérer les derniers changements du dépôt distant,
— `git push` pour mettre à jour le dépôt distant avec les changements du dépôt local.
Poussons notre dépôt sur GitHub.

```
git push origin main
```

Règle d'or : Quand on travaille sur plusieurs ordis/avec des collaborateurs, etc., il faut toujours faire `git pull` avant de faire `git push` pour minimiser les erreurs.

5 Branches

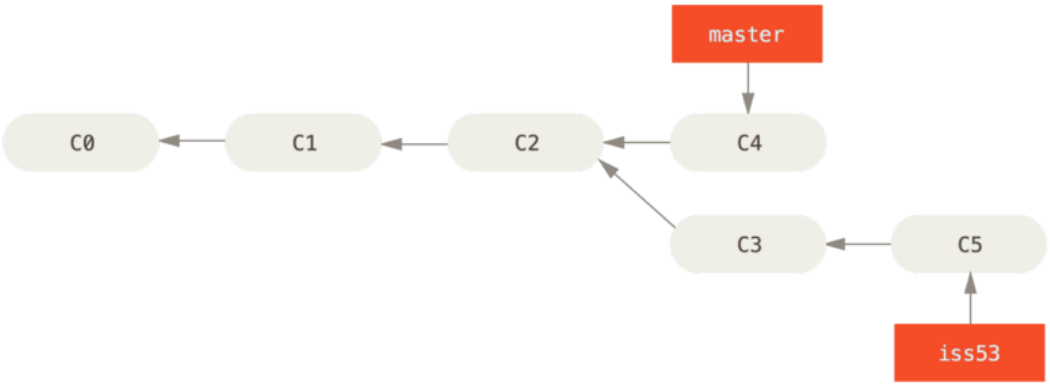
On peut créer une branche avec `git checkout -b` (il est aussi possible d'en créer une sur GitHub, puis faire `git pull` pour la récupérer en local). `git branch` donne la liste des branches qui existent en local.

```
git checkout -b feature/newfunction
git branch
```

Pour switcher entre les branches, on pourra utiliser `git checkout`.

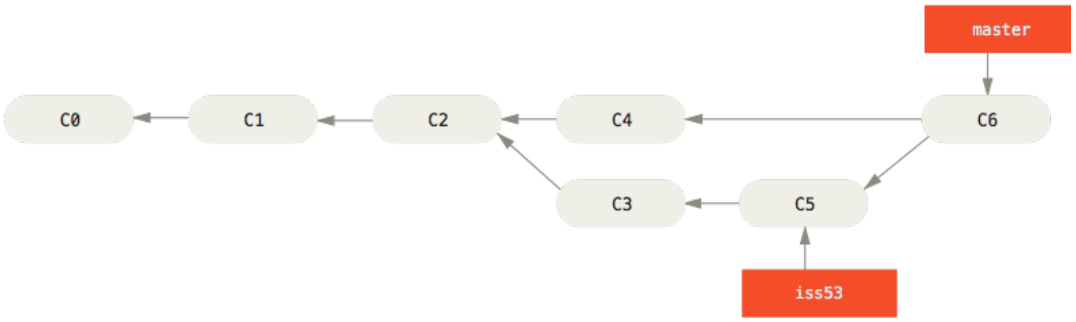
```
git checkout main
git checkout feature/newfunction
```

À présent, vous disposez de deux branches qui peuvent être développées séparément, puis refusionnées.



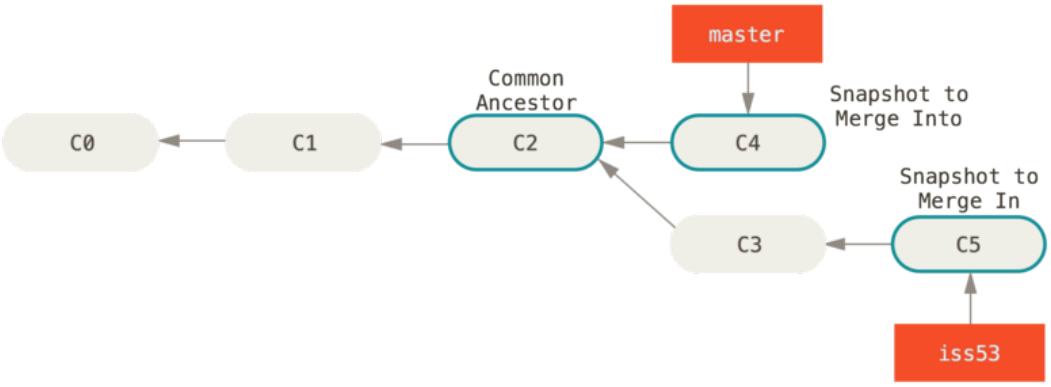
Dans la figure ci-dessus, les branches `master` et `iss53` ont divergé. Une fusion de branche se fait dans un sens : par exemple, supposons qu'on veuille fusionner `iss53` dans la branche `master`. Alors il faudra se placer dans la branche `master` puis utiliser la commande `git merge iss53`. On obtiendrait alors l'arbre ci-dessous.

Lorsqu'on fait un `git pull` d'un dépôt distant, Git récupère le dépôt distant sous forme de branche et réalise implicitement une fusion avec la branche locale. Avant de voir un autre exemple pratique de fusion, parlons des conflits.



6 Conflits

Un conflit survient suite à une fusion de branches que Git ne sait pas résoudre. En effet, pour réaliser une fusion, Git utilise 3 commits : les derniers commit des deux branches impliquées par la fusion ainsi que le dernier commit qu'elles ont en commun. Si un fichier est présent sous une forme différente dans les 3 commits, alors un conflit existe.



Créons artificiellement un conflit. On se place sur la branche `feature/newfunction` et on modifie le fichier helloworld.R (actuellement vide). Écrivons encore `print("helloworld")` dans ce fichier, mais cette fois faisons un commit.

```
git checkout feature/newfunction
echo "print(\"helloworld\")" > helloworld.R
git add helloworld.R
git commit -m "ADD a print command in helloworld.R"
```

À présent, changeons de branche. On remarque que dans la branche `main`, le fichier `helloworld.R` est bien vide. Écrivons une autre commande dans ce fichier, par exemple `rm(list=ls())`. Validons avec un commit.

```
git checkout main
echo "rm(list=ls())" > helloworld.R
git add helloworld.R
git commit -m "ADD a clean command in helloworld.R"
```

Maintenant nous allons créer un conflit en fusionnant les deux branches. En effet, on a 3 versions différentes du fichier `helloworld.R` dans les 3 commits que Git utilise pour réaliser la fusion :

— fichier contenant `print("helloworld")` dans la branche `feature/newfunction`,
— fichier contenant `rm(list=ls())` dans la branche `main`,
— fichier vide dans l'ancêtre commun.

Demandons à Git de fusionner `feature/newfunction` dans `main`, et regardons comment on peut résoudre le conflit.

```
git merge feature/newfunction
git status
```

Normalement, un message doit vous avertir du conflit. Ouvrez le fichier `helloworld.R` avec un éditeur de texte pour voir ce qu'il s'y passe. Il suffit d'y apporter les modifications qu'on veut pour résoudre le conflit. On pourra alors réaliser le commit de merge et finaliser la fusion.

Après avoir résolu les différences dans le fichier `helloworld.R` :

```
git add helloworld.R
git status
```

Après la fusion, la branche `feature/newfunction` devient inutile, il est alors conseillé de la supprimer avec la commande `git branch -d`.

```
git branch -d feature/newfunction
```