Manipuler Git avec Rstudio

Annaig De-Walsche, Marina Gomtsyan, Mary Savino

Novembre 2022

1 Résumé

Dans cette séance, nous proposons de découvrir l'utilisation de Git via Rstudio pour profiter des nombreux intérêts qu'il offre, notamment la gestion d'un travail collaboratif. Nous apprendrons à utiliser les commandes basiques de Git, à communiquer avec un dépôt distant (Github) et à manipuler ces commandes directement depuis Rstudio. Dans un premier temps, nous verrons des illustrations simples des commandes Git à travers un tutoriel, puis chacun pourra mettre en pratique l'utilisation de Git et de Github depuis son ordinateur.

2 Introduction

Vous avez surement déjà entendu parler de "Git" ou "GitHub", mais finalement qu'est-ce que c'est ?

Liens utiles : `https://stateofther.netlify.app/post/decouverte-git/`
`https://thinkr.fr/travailler-avec-git-via-rstudio-et-versionner-son-code/`
`https://docs.github.com/en/get-started/quickstart/hello-world`
`https://learngitbranching.js.org`

2.1 Présentation de Git et Github

Git est un outil de gestion de version. Un gestionnaire de version garde toutes les versions successives d'un fichier et permet de savoir à tout moment qui a modifié un fichier, quand et pourquoi. Les principales fonctionnalités de Git sont :

— Revenir à une version antérieure
— Travailler à plusieurs
— Travailler en parallèle
— Simplifier la réconciliation entre deux versions différentes

Plus précisément, Git gère l'historique d'un ensemble de fichiers contenus dans un répertoire (et ses sous répertoires). On appelle ce répertoire un "dépôt". Chaque fichier déclaré par l'utilisateur est suivi, et à chaque "commit", les modifications apportées aux fichiers suivis sont enregistrées.

Github est une plate-forme d'hébergement de code permettant de stocker ses projets sur des serveurs distants. Elle fournit une interface web pour faciliter le travail en équipe : gérer les dépôts "git", publication des paquets, documentation...

2.2 Pré-requis

Télécharger et installer Git ici
Création d'un compte GitHub là

3 Travaux préliminaires sur les commandes `Git`

Tout d'abord, afin d'utiliser `Git` correctement, il faut comprendre l'architecture du *workflow* sous-jacent (Figure 1) et comment celui-ci s'utilise. Le dépôt local consiste en trois étapes maintenus par `Git` :

1. La première étape est le **dépôt de travail** (*working directory*) qui contient les fichiers réels,
2. la deuxième est **l'index** (*Index Stage*) qui agit comme une zone de transit,
3. et enfin la dernière étape est **le HEAD** qui pointe vers le dernier commit effectué.

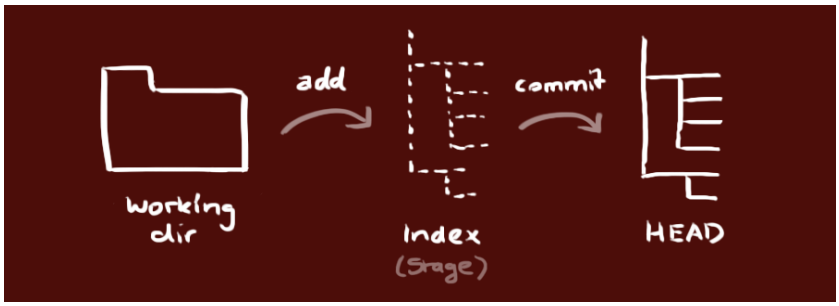

FIGURE 1 – Le *workflow* de `Git`

3.1 Les commandes principales

Dans cette section, nous présentons les commandes de `Git` les plus fréquemment utilisés.

— `git init`
Utilisé pour construire un nouveau dépôt.
Utilisation : `git init [nom du dépôt]`

— `git clone`
Utilisé pour obtenir une copie d'un dépôt existant (local ou distant).
Utilisation : `git clone [/chemin/vers/dépôt]` (local)
`git clone [username@host:/chemin/vers/dépôt]` (distant)

— `git add`
Utilisé pour ajouter un fichier à la zone de transit (Index).
Utilisation : `git add [nom du fichier]`

— `git commit`
Utilisé pour commiter dans le HEAD, mais pas encore dans le dépôt distant.
Utilisation : `git commit -m "Message de commit"`

— `git status`
Cette commande liste tous les fichiers qui sont en attente de commit.
Utilisation : `git status`

— `git rm`
Utilisé pour supprimer un fichier à la fois dans le *working directory* et dans Index. Pour

affecter cette suppression à HEAD, il faut par la suite commiter.
Utilisation : `git rm [nom du fichier]`

— `git pull`
Cette commande récupère et fusionne les changements du dépôt distant dans le *working directory*.
Utilisation : `git pull`

— `git push`
Cette commande met a jour le dépôt distant avec les changements du dépôt local.
Utilisation : `git push origin master`

3.2 Les branches `Git`

Une branche dans Git est simplement un pointeur léger et déplaçable vers un des commits. La branche par défaut dans Git s'appelle `master`. On utilise d'autres branches pour le développement et on les fusionne avec la branche `master` une fois le développement terminé. Les branches permettent un développement en parallèle, sans s'affecter entre elles. A la fin, les modifications des différentes branches peuvent être ajoutées à la branche `master`.

— `git branch`
Pour lister toutes les branches locales dans le dépôt actuel et pour créer une nouvelle branche
Utilisation : `git branch`
`git branch feature` (crée une nouvelle branche appelée `feature`)
`git branch -d feature` (pour supprimer la branche)

— `git checkout`
Cette commande est utilisée pour passer d'une branche à l'autre.
Utilisation : `git checkout [nom de la branche]`
`git checkout -b [nom de la branche]` (pour créer une nouvelle branche et y passe en même temps)

— `git merge`
Cette commande intègre l'historique de la branche spécifiée dans la branche courante.
Utilisation : `git merge [nom de la branche]`

— `git diff`
Cette commande montre les différences entre les fichiers qui ne sont pas encore dans Index.
Utilisation : `git diff`
`git diff [première branche] [seconde branche]` (pour voir les différences entre les deux branches)

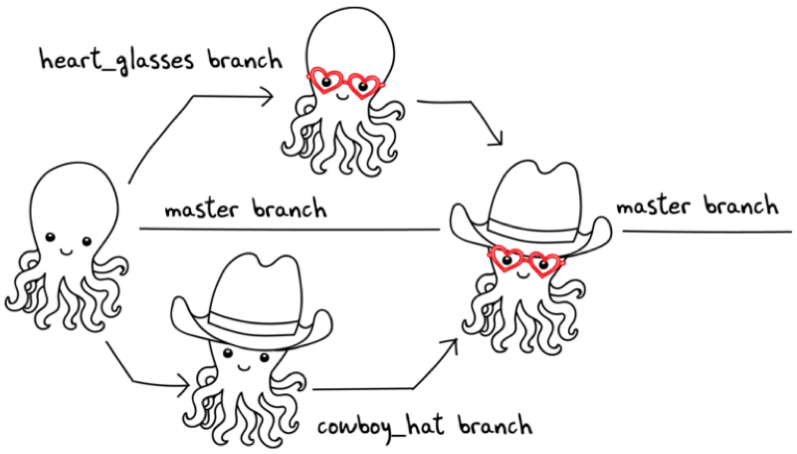

FIGURE 2 – Merging de deux branches `Git`

4 Mise en pratique et travaux sur le dépôt distant

4.1 Mise en place du dépôt distant

Les dépôts distants (remote repository en anglais) sont des copies de votre dépôt local sur un autre ordinateur. Vous pouvez donc accéder à ce dépôt via Internet, ce qui vous permet de transférer les commits effectués.

Les avantages sont nombreux :
Tout d'abord, le dépôt distant sert de sauvegarde. En effet, le dépôt local de Git a la capacité de restaurer des fichiers à un état précédent (par conservation des versions antérieures). Cela permet d'avoir une sauvegarde du dépôt Git sur d'autres ordinateurs et de récupérer les données qui peuvent être perdues. Enfin, le dépôt distant permet de réaliser du travail collaboratif et facilite la contribution des différents participants d'un même projet.

Pour bénéficier de ce dépôt distant, il faut se créer un compte GitHub (voir Prérequis dans la section 2.2). Afin de créer un nouveau dépôt à distance sur GitHub, vous pouvez opter pour deux options :

- Soit vous réalisez la commande suivante depuis votre terminal de commande `git init`
- Soit vous allez directement sur votre profil GitHub et dans l'onglet ***Repositories*** vous cliquez sur ***New***. Indiquez un nom pour ce dépôt. Vous avez la possibilité de choisir son statut : soit **public** soit **privé**. Ce statut peut être modifié en aval.
  Vous pouvez ajouter un fichier README en cochant la case ***Add a README file***. La création d'un fichier .gitignore est également possible lors de la création du dépôt sur GitHub et permet de demander à Git de ne pas prendre en compte certains fichiers présents en local et donc de ne pas les inclure dans le dépôt (par exemple des sorties de programme, code compilé, fichiers .Rproj, ...). Pour cela, il faut cocher la case ***Add .gitignore***.

Après avoir créé ce nouveau dépôt, on peut à présent le cloner en local et commencer à y ajouter des modifications.

4.2 Communiquer avec le dépôt distant

Il y a deux façons principales de communiquer avec GitHub : en HTTPS ou SSH.

4.2.1 En HTTPS

Créer un token

Dans un premier temps, vous devez créer une clé digitale, appelée **token**, afin de pouvoir vous identifier avec votre compte GitHub. Pour cela, lorsque vous être sur votre page d'accueil, cliquer en haut à droite sur votre profil puis allez dans ***Settings*** → ***Developer Settings*** (en bas de la liste) ***Personal Access Tokens*** → ***Tokens (classic)*** puis faites ***Generate new token*** → ***New personal access token (classic)***. Vous pouvez alors lui ajouter un label et une date d'expiration (obligatoire). Choisissez les accès permis par ce token en cliquant sur les catégories que vous voulez (cochez toutes les cases pour être tranquille) et faites *Generate token*. Pensez à bien **copier et conserver** votre token quelque part afin que vous puissiez l'utiliser.

Cloner le dépôt

Pour cela, il faut récupérer l'adresse HTTPS du dépôt qui s'affiche sur GitHub (pour l'obtenir, il faut cliquer sur ***Code*** en vert puis ***HTTPS***). Placez-vous dans le répertoire où vous voulez y intégrer votre dépôt local, exécutez `git clone` en précisant l'adresse HTTPS de votre dépôt. Une authentification sera nécessaire afin de finaliser le clonage. Pour ***Username***, précisez votre identifiant GitHub. **Attention !** Pour ***Password***, précisez **non pas** votre mot de passe de compte GitHub mais le token que vous avez créé préalablement (voir Section 4.2.1). On peut réaliser un `git config credential.helper store` suivi d'un push afin de garder en mémoire l'identification et de ne pas avoir à préciser le token à chaque fois que RStudio se connectera à Git .

4.2.2 En SSH

Création d'une paire de clés SSH

Il faut d'abord générer une paire de clés SSH (clé Privé/clé Publique) qui sera propre à votre ordinateur : `ssh-keygen`. Vous aurez alors les identifiants de vos deux clés contenus dans deux fichiers : id_clé pour la clé privée et id_clé.pub pour la clé publique. Sur GitHub, vous pouvez ainsi préciser la **clé publique** ainsi créée, directement dans ***Settings*** → ***SSH and GPG keys*** → ***New SSH key***. **Attention !** Il ne faut pas donner sa clé privée !

Cloner le dépôt

Ensuite, pour cloner un dépôt distant, vous devez cliquer sur Code puis SSH. Il faut alors copier l'adresse du dépôt, aller dans votre terminal de commande et exécuter git clone suivi de l'adresse. L'avantage du SSH est que vous n'aurez alors jamais besoin de vous identifier lorsque vous souhaiterez cloner vos dépôts distants sur votre ordinateur.

4.2.3 Sur Rstudio

Afin d'utiliser Git directement depuis Rstudio, il faut s'assurer que RStudio et `Git` soient bien connectés. Dans un premier temps, indiquez à RStudio où se trouve Git sur votre ordinateur :

Tools → ***Global Options...*** → ***Git/SVN***. Pour interagir avec Git, les deux protocoles cités précédemment sont possibles. Pour créer une clé SSH depuis Rstudio directement, il faut cliquer sur Create RSA key... et procéder comme décrit dans Section 4.2.2.

A présent sur Rstudio, aller dans ***File*** → ***New Project...*** → ***Version Control*** → ***Git***, renseignez l'URL/le SSH de votre repository que vous avez préalablement copié, le nom du projet R (le même que Git idéalement) et le dossier dans lequel le placer. Enfin cliquez sur ***Create Project***.

Dans le terminal de Rstudio, faire :

```
git remote add origin adress-https-ou-ssh-du-depot
git push -u origin master
```

afin de connecter le dépôt distant à notre projet R. L'état des différents fichiers que vous aurez créés, modifiés, supprimés... de votre répertoire git est précisé dans l'onglet Git de Rstudio comme le montre la figure 3. Les différents états possibles sont précisés dans la figure 4. Il faut éviter de push son `.gitignore`. Vous pouvez suivre l'historique en allant dans ***Commit*** puis ***History*** de Rstudio.

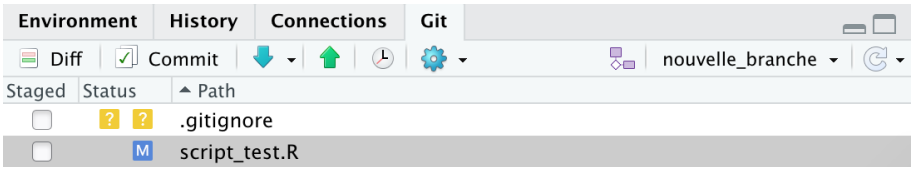

FIGURE 3 – Emplacement des différents fichiers ainsi que leur statut respectif

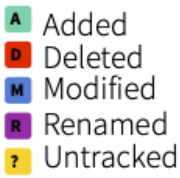

FIGURE 4 – Différents statuts des fichiers

4.3 Résolution des conflits

Modifier un fichier à plusieurs en même temps peut générer ce qu'on appelle des "conflits". Il arrive que les modifications effectuées par un collaborateur (que l'on récupère du dépôt distant avec un pull) contredisent les modifications que l'on a faites de notre côté en local. Cela provoque alors un conflit, et un message d'erreur apparaît au moment du push. Bien entendu, chaque conflit est unique et nécessite une solution individuelle. Cependant, nous allons essayer de donner quelques idées de base sur la façon dont nous pouvons commencer à traiter les conflits.

4.3.1 Quelques commandes utiles

— Pour remplacer le fichier modifié dans le dépôt de travail par le dernier contenu dans HEAD : `git checkout -- [nom du fichier]`

— `git reset`
Pour conserver un commit et, par conséquent, supprimer tous ceux qui le suivent :
`git reset [id du commit]`
Pour supprimer toutes les modifications et commits locaux, récupérez l'historique le plus récent du serveur distant et faire pointer la branche `master` locale vers lui :
`git fetch origin`
`git reset --hard origin/master`

— `git revert`
Pour supprimer un commit :
`git revert [id du commit]`

4.3.2 Sur RStudio

Ici, nous allons voir comment régler un conflit simple sur Rstudio. Dans la branche `master`, nous disposons d'un fichier script contenant du code (figure 5) dans lequel nous générons deux variables. Nous créons une branche appelée `nouvelle_branche`.

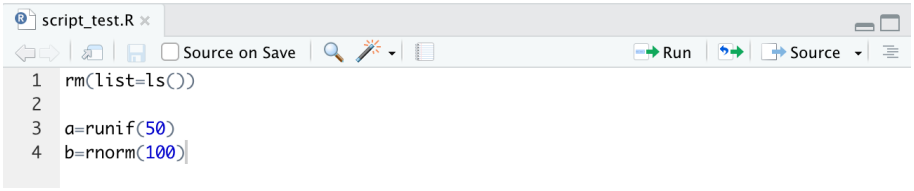

FIGURE 5 – Code initial

Dans le script de la `nouvelle_branche`, nous ajoutons la variable `c`. Parallèlement, dans le script de la branche `master`, nous ajoutons une autre variable `d`. En se positionnant sur la branche `master`, nous essayons d'effectuer un `merge` de la `nouvelle_branche` avec la branche `master` en exécutant :

```
git merge nouvelle_branche
```

La fusion des deux branches n'est pas possible car les deux versions du fichier se contredisent. Un message d'erreur comme indiqué dans la figure 6 apparaît, et un conflit est crée.

FIGURE 6 – Message de conflit

Dans le fichier, les parties délimitées par <<<< et >>>> indiquent les différences dans le script de `nouvelle_branche` et `master`, comme dans la figure 7. Il faut résoudre le conflit en choisissant manuellement les modifications que nous désirons retenir entre les 2 versions du fichiers

et en supprimant les <<<< et >>>>. Une fois la modification effectuée, nous pouvons à nouveau faire un commit.

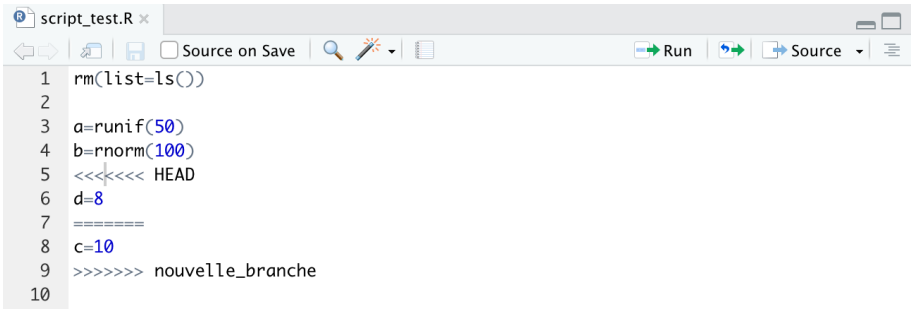

FIGURE 7 – Différences dans les fichiers