Rolland Julia **Rapport GitHub** 09/10/2019

INFO 1 LO43

GitHub est une plateforme d’hébergement de code. Elle est centralisée et possède un historique. Cette plateforme est utilisée dans le but de collaborer puisque tout le monde a accès au code une fois celui-ci déposé dans un repository (dépôt) en ligne. De plus, l’historique sur GitHub permet de voir où en est rendu le projet ainsi que les modifications de chacun. C'est-à-dire qu’une fois que nous modifions le dépôt, la plateforme enregistre après un commit, les changements (personne qui a modifié, dans quels fichiers, quand les changements ont eu lieu). Le fait de laisser un historique permet de pouvoir toujours revenir en arrière si un bug apparait.

Dans ce compte-rendu, nous allons voir quelques fonctions principales, utilisées sur GitHub :

**Git config :**

Les commandes liées à git config sont utilisées afin de configurer les informations des utilisateurs pour tous les dépôts locaux. On peut configurer le nom, l’email que nous voulons associer au projet ainsi que le colorisation de la sortie en ligne de commande.

Voici la manière dont ces commandes doivent être écrites :

$ git config --global user.name "[nom]" //association nom

$ git config --global user.email "[adresse email]" //association email

$ git config --global color.ui auto //colorisation de la sortie

**Git init :**

Cette commande permet de créer un nouveau dépôt local. On peut le retrouver dans l’espace repository. Dès que l’on utilise cette commande, un historique du fichier se crée et se met à jour à chaque fois que l’on modifie un élément du dépôt. Un fichier, contenant la structure interne permettant le contrôle du projet, est caché à l’intérieur de ce dépôt.

Généralement, on doit préciser à la suite de cette commande le nom du projet. Voilà un exemple :

$ git init monProjet

**Git status :**

Git status fait partie des commandes qui permettent d’effectuer des changements. Son rôle est d’afficher une liste de tous les fichiers (nouveaux ou modifiés) à commiter. Cette commande est très pratique puisqu’elle nous permet de suivre notre avancement lorsque l’on apporte des modifications au projet.

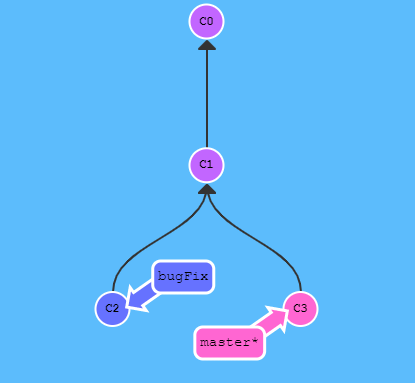
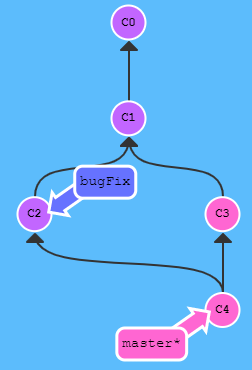
$ git status

**Git merge :**

La commande git merge permet de grouper des changements. Elle combine dans la branche courante le nom de la branche mis en paramètre. Le ‘merge’ crée un commit spécial qui a deux parents

Cette commande s’écrit de cette manière : $ git merge [nom-de-branche]

Ici, nous voulons combiner les branches master et bugFix. Voici la commande à écrire : $ git merge bugFix

 🡪 git merge 🡪 

**Git diff :**

Tout comme Git status, cette commande fait partie de celles qui permettent d’effectuer des changements dans le dépôt. Selon la façon dont elle est écrite, elle peut effectuer plusieurs actions.

$ git diff

Lorsque rien n’est spécifié derrière, elle montre les modifications de fichier qui ne sont pas encore indexées.

$ git diff --staged

Lorsqu’on rajoute les deux tirets, elle montre les différences de fichier entre la version indexée et la dernière version.

$ git diff [premiere-branche]...[deuxieme-branche]

Dans ce cas-là, elle permet de faire une comparaison entre les deux branches spécifiées.

**Git blame :**

La commande ‘blame’ montre la dernière personne qui a effectué des modifications ainsi que l’emplacement de toutes ces modifications dans le code du projet. En fonction du paramètre renseigné à la suite, plusieurs informations sont ajoutées dans la description fournie par ‘blame’. Il est également possible de renseigner plusieurs paramètres dans un seul ‘git blame’ afin d’avoir une description complète.

Voici deux exemples de paramètre :

$ git blame --root //ne considère pas les commits de racine comme des limites

$ git blame --show-stats //inclut des statistiques supplémentaires à la fin de la commande

Et encore bien d’autres.