TP d'initiation à Git

Février 2018

APERCU

Dans ce TP vous serez amenés à travailler avec un herbergeur Git, à installer la console Git Bash pour utiliser les commandes Git, puis à tester les différentes commandes permettant l'utilisation de Git.

OBJECTIFS

1. Créer et cloner un repository
2. Faire des modifications et les partager
3. Gérer les conflits

1. Hebergeur Git

1.1 Créer un compte GitHub

Chaque membre de l'équipe doit créer un compte GitHub, en se connectant sur le site GitHub (https://github.com/). Suivez les instructions d'inscription:

1. Sur la page d'accueil, remplissez le formulaire d'inscription
2. Choisissez le compte gratuit
3. Répondez au questionnaire et/ou terminez votre inscription
4. Activez votre compte avec le mail d'activation dans votre boite mail.

1.2 Création d'un repository

UN SEUL membre du groupe crée le nouveau repository. Il le partagera ensuite avec les autres membres. Ce membre s'appellera le « project owner ».

Pour cela, cliquez sur le lien "Start a project" après la création de votre compte.

Nommez votre repository “ECE-ProjetGit” ou autre et n’oubliez pas d’activer “Initialize this repository with a README”. (**Attention**: Si vous oubliez de cocher cette option vous allez devoir créer la branche principale **master** manuellement !). Cliquer sur “Create repository” pour terminer la création.

1.3 Ajout des membres

Pour ajouter des membres au projet, le “project owner” doit inviter les différents membres..

Une fois dans le panel du repository, cliquez sur “*Settings*” puis “*Collaborator*” et chercher les différents membres avec leur nom d’utilisateur ou adresse e-mail. Une fois tous les membres ajoutés, chacun d’entre eux devra confirmer l’invitation qui a été envoyé par mail. (Attention à bien vérifiez vos courriers indésirables.)

Une fois les membres ajoutés, vous pouvez procéder à l’installation de Git Bash pour effectuer les premiers commits.

2. Installation de Git Bash

Vous allez désormais installer un outil Git permettant de rapidement avoir une console configurer avec Git. Rendez-vous sur le site https://git-scm.com/downloads puis télécharger l'installateur selon votre système d’exploitation.

Une fois le téléchargement fini, exécutez l'installateur et suivez les instructions en laissant les paramètres par défaut. Une fois l’installation terminée, vous pouvez passer à la partie suivante.

3. Mise en place du Git Flow

Pour la mise en place de ce projet Git, vous créerez un petit projet en C (ou un autre langage de votre choix) avec l'IDE de votre choix.

Les commandes sont à effectuer dans la console *Git Bash* pour les systèmes Windows et dans le terminal pour les systèmes MacOS et Linux. (**Attention:** les caractères "$>" ne sont pas à prendre en compte dans votre console, ils symbolisent simplement l'entrée clavier console.)

Pour ouvrir le terminal sur MacOS, faites la combinaison de touche CMD+SPACE puis écrivez "terminal". Confirmez la saisie avec la touche ENTER.

3.1 Récupération du repository

Avant de pouvoir faire le premier commit sur le repository, il faut récupérer le repository distant et créer le repository local. Pour cela, effectuez la commande:

```
$> git clone https://github.com/owner_username/nom_du_repo
```

Astuce: l'adresse URL de la commande correspond à l'adresse du repository dans votre navigateur Web !

Par la suite n'oubliez pas de vous positionner dans votre dossier Git avec la commande:

```
$> cd nom_du_repo
```

N'oubliez pas d'ouvrir votre explorateur de fichier à l'adresse du repository afin de pouvoir y ajouter des fichiers plus tard. Pour connaitre la position du repository local dans votre système, utilisez la commande:

```
$> pwd
```

La commande vous indiquera la position du fichier sur votre disque dur.
Exemple: "\c\Utilisateurs\quentin\ECE-ProjetGit"

3.2 Configuration du compte GitHub

Avant d'effectuer des opérations sur votre repository distant, il va falloir configurer votre repository local pour pouvoir correctement vous connecter à l'hébergeur GitHub. Pour cela, effectuez la commande:

```
$> git config --global user.email votre.mail@edu.ece.fr
```

Exemple: "git config --global user.email quentin.cabanes@edu.ece.fr"

Pour vérifier que votre configuration est bien enregistrée, vous pouvez utilisez la commande:

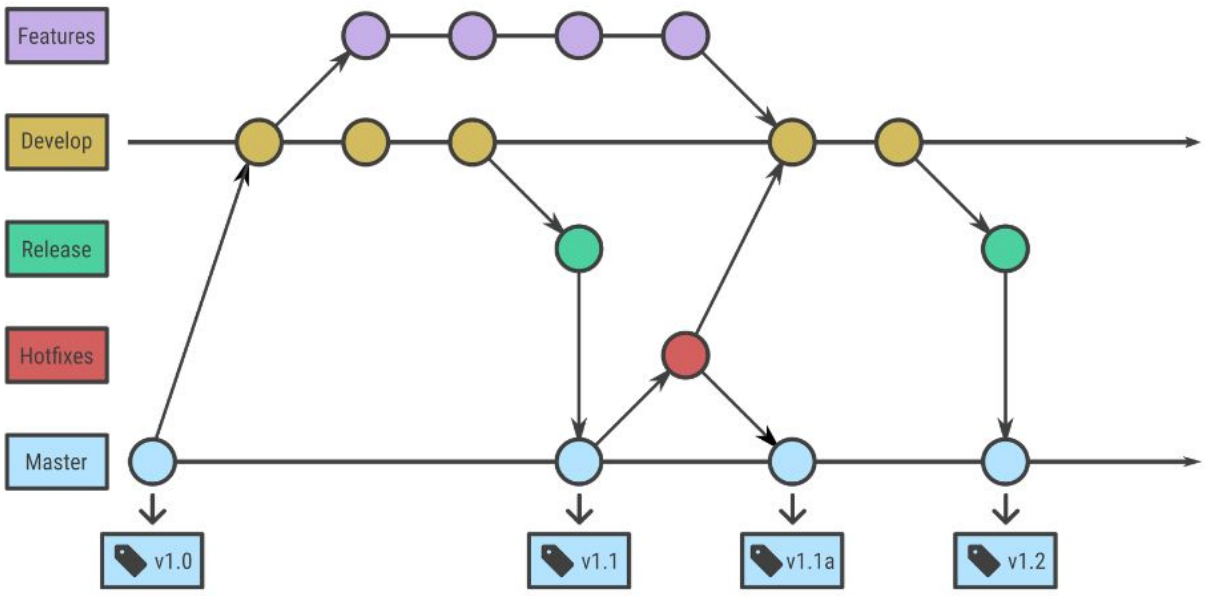
```
$> git config --list --global
```

Puis vérifiez que l'entrée **user.email** est présente. Si le commande **commit** vous indique encore une erreur, renseigner aussi **user.name** avec la même commande:

```
$> git config --global user.name nom_utilisateur_github
```

3.3 Création des branches

Vous allez désormais mettre en place le GitFlow vu plus tôt.



Pour cela vous allez créer les branches les plus importantes, à savoir **develop** et les différentes branches de **features**. Pour créer les branches, il faut utiliser la commande:

```
$> git checkout -b nom_de_la_branche
```

Attention: les noms de branche doivent être écrit en **minuscule** ! (Exemple: "master")

Un membre du groupe devra créer la branche **develop**, puis chaque membre devra créer sa propre branche **features**. Pour cela, chaque membre créera une branche préfixé par **ft_** suivi du nom de la features. (Exemple: **ft_helloworld**)

La branche ne sera créé sur le repository distant qu'à partir du moment où un commit puis une mise à jour du repository distant seront effectués dessus. De même, la branche ne sera pas sauvegarder sur le repository local tant qu'aucun commit n'aura été effectué dans la branche.

Pour voir les branches disponibles dans votre repository, vous pouvez utiliser la commande:

```
$> git branch -a
```

3.4 Création des fichiers et modifications

Pour chaque branche **features** créé, chaque membre devra ajouter dedans un fichier correspondant à la feature que vous souhaitez développer. Ajoutez ce fichier dans le repository local avec la commande:

```
$> git add nom_du_fichier
```

Attention: la commande "add" ne crée pas de fichier pour vous ! C'est à vous de créer un fichier et de l'ajouter au prochain commit grâce à cette commande. (Exemple: "git add main.c")

Après l'ajout du fichier, vous pouvez vérifier qu'il a été pris en compte avec la commande:

```
$> git status
```

Puis effectuez le commit avec un message explicatif (Exemple: "Création de HelloWord").

```
$> git commit -m "Votre message"
```

Après un commit, vous pouvez vérifier qu'il a bien été pris en compte avec la commande:

```
$> git log
```

Par la suite, vous allez créer un fichier dans la branche **develop** qui correspondra à la fonction principale de votre programme, puis vous l'ajouterez au repository local et effectuerez le commit. N'oubliez pas de faire appel aux fonctions développées dans les branches **features** dans votre programme**.**

Une fois tous les fichiers ajoutés et commit, mettez à jour le repository distant avec la commande:

```
$> git push
```

Dans le cas d'un repository privé, votre username/e-mail et votre mot de passe seront demandés.

Attention: Si la branche n'existe pas sur le repository distant, utilisez plutôt la commande:

```
git push --set-upstream origin nom_de_la_branche
```

Attention: il n'est pas possible de **push** si votre repository local n'a pas la même version de base que le repository distant, il faut donc utiliser la commande **pull** dans ces cas la.

3.5 Fusion des branches

Vous allez fusionner les branches de **features** avec la branche **develop**. Tout d'abord assurez vous que chaque branche est à jour avec la commande:

```
$> git fetch
```

Ensuite rendez-vous dans la branche dites "*réceptrice*", c'est à dire la branche qui accueillera le merge, donc la branche **develop**:

```
$> git checkout develop
```

Vous allez désormais fusionner la branche que vous souhaitez avec la branche **develop** grâce à la commande:

```
$> git merge nom_de_la_branche -m "Votre message"
```

Si la fusion c'est bien passée, vous pouvez désormais envoyez vos modifications au repository distant (Cf 3.4). Vous pouvez aussi supprimer la branche de **features** maintenant inutile avec la commande:

```
$> git branch -d nom_de_la_branche
```

La branche est désormais supprimé en local. Pour mettre à jour la suppression sur le repository distant, il faut exécuter la commande suivante:

```
$> git push --delete nom_remote nom_de_la_branche
```

Le nom de la remote devra normalement être **origin**. Pour vérifier le nom du remote, utilisez la commande:

```
$> git remote show
```

3.6 Gestion de conflits

Vous allez maintenant créer un conflit. Pour rappel, Git détecte un conflit quand deux personnes essaient de modifier le même fichier et qu'il est impossible de fusionner sans impacter un des deux commits. (Exemple: Modifier la même ligne d'un fichier)

Deux membres de l'équipe vont se placer sur la branche **develop** et vont modifier la même ligne dans un même fichier puis effectueront un commit. La première personne qui mettra à jour le repository distant n'aura pas de conflit, c'est le deuxième membre qui devra résoudre le conflit.

La console devrait vous montrer les fichiers qui ont rencontrés des conflits lors de la fusion et Git ajoutera des caractères spéciaux à l'endroit où le conflit aura eu lieux. Cela devrait ressembler à l'exemple ci-dessous:

```
<<<<<<< HEAD
Here is the original change.
=======
Here is the modified change.
>>>>>>> origin/branch
```

La première partie qui est entre "<<<<<<<" et "=======", correspond normalement aux changements locaux, c'est à dire les changements que vous avez effectués. La deuxième partie qui est entre "======" et ">>>>>>>" correspond aux changements distants. Le **HEAD** correspond la version de votre branche, et le **origin/branch** correspond à la branche sur laquelle les changements distants ont été effectués.

Pour résoudre le conflit, vous devez désormais supprimer le changement que vous pensez inutile puis supprimer le reste des caractères spéciaux. Par exemple:

```
Here is the modified change.
```

Une fois les fichiers en conflits modifiés, vous pouvez commit votre résolution de conflits et mettre à jour le repository distant.

3.7 Tag de version

Vous allez désormais mettre en place un tag sur la branche **master**. Tout d'abord, fusionnez la branche **develop** avec la branche **master** (Cf 3.5). Une fois la fusion terminée, vous pouvez ajouter le tag grâce à la commande:

```
$> git tag -a v0.0 -m "Votre message"
```

Vous pouvez désormais observez vos tags:

```
$> git tag
```

Ou bien affichez les modifications effectuées pour un tag:

```
$> git show nom_du_tag
```

Attention: Pour mettre à jour vos tags sur le repository distant, il faut spécifiquement demander à mettre à jour les tags avec la commande:

```
$> git push nom_remote --tags
```

Le nom de la remote devra normalement être **origin**. (Cf 3.5)