TP5 : Réseau et forges

Configuration de GitHub

Nous allons utiliser la plate-forme GitHub (`https://github.com`). Vous devez créer une compte dessus via le lien "Sign Up" en haut à droite de l'interface. Pour pouvoir communiquer via SSH, vous auriez aussi besoin de transmettre votre clé publique à GitHub, afin de permettre au service de vous authentifier. Mais durant ce TP, nous utiliserons exclusivement le protocole HTTPS, qui vous authentifie en vous demandant votre mot de passe GitHub. Aucune configuration supplémentaire ne sera donc nécessaire.

Fork et clonage

Plusieurs textes sont disponibles sur `https://github.com/cguziolo/Examples_git_medev.git`.

Pour commencer, créez un fork sous votre contrôle, en utilisant le bouton "Fork" situé en haut à droite de l'interface de GitHub. Lorsque c'est fait, vous devriez être redirigé automatiquement sur votre fork. Pour travailler dessus localement, vous allez ensuite devoir copier son contenu sur votre machine. Pour cela, vous avez besoin de son URL HTTPS, que vous pouvez obtenir de l'interface GitHub de la façon suivante :

— En cliquant sur le bouton vert "Clone or download".
— En cliquant sur l'option "Use HTTPS" si elle s'affiche.
— L'interface devrait maintenant afficher une URL commençant par "http ://".

Une fois l'URL obtenue, créez un clone local du fork de la façon suivante :

```
1 $ cd Bureau/
2 $ git clone <url de votre depot>
```

Connexion au dépôt officiel

Comme notre clone local dans le TP précédent, le clone du fork possède une remote pointant vers le dépôt distant qui permet d'échanger facilement des commits avec ce dernier :

```
1 $ cd Examples_git_medev
2 $ git status
3 $ git remote -v
```

Mais pour appliquer la méthode de travail basée sur les *forks* et les *pull requests* (appelée *GitHub flow)*, il faut aussi être capable d'échanger des commits avec le dépôt officiel, celui à partir duquel vous avez créé votre fork. En effet, des nouveautés vont arriver sur ce dernier au fur et à mesure du TP, et vous devrez vous synchroniser avec ces dernières.

Par convention, le dépôt officiel est généralement appelé *upstream* :

```
1 $ git remote add upstream https://github.com/cguziolo/Examples_git_medev.git
2 $ git fetch upstream
```

Si vous ouvrez maintenant une application pour visualiser l'état de git comme `gitg` (Linux) ou `gitKraken` (Mac Os, Linux, Windows), qui permettent de visualiser la structure des branches et l'historique de *commits*, vous pourrez constater que vous pouvez maintenant visualiser vos commits et branches locales, celles de votre fork, et celles du dépôt officiel. Tout est prêt !

Travail distribué avec le *GitHub flow*

Introduction

Le *GitHub flow* est une méthode de travail qui, malgré son nom, n'a rien de spécifique à la forge GitHub. Toutes les forges sociales modernes peuvent être utilisées de cette manière. Dans cette méthode, le travail s'organise autour de quatre grandes étapes :

1. Rester à jour par rapport au dépôt officiel
2. Préparer des changements dans une branche dédiée de son dépôt local
3. Publier ses changements sur son fork lorsqu'ils sont prêts
4. Les soumettre au dépôt officiel pour qu'ils y soient intégrés

Examinons ces quatre étapes en détail

Se synchroniser avec le dépôt officiel

Comme nous l'avons vu dans le TP précédent, il est possible de récupérer les changements du dépôt officiel avec git fetch :

```
1 $ git fetch upstream
```

Cependant, dans le cadre de cette méthode de travail, nous n'avons généralement pas besoin de nous intéresser à l'ensemble des branches de ce dépôt. Il suffit de tenir notre branche master à jour, ce que l'on peut faire avec `git pull` après avoir configuré upstream/master comme branche amont :

```
1 $ git checkout master/main
2 $ git branch --set-upstream-to=upstream/master/main
```

Par la suite, vous pourrez mettre à jour votre master via :

```
1 $ git checkout master/main
2 $ git pull --ff-only
```

Je conseille ici l'utilisation de `--ff-only`[1] pour vous aider à repérer une erreur classique. Si l'on applique correctement le GitHub flow, la branche master du dépôt officiel devrait avancer de façon monotone, et votre branche master locale devrait suivre son avancée. Par conséquent, si vous ne pouvez pas mettre en avance rapide, cela relève d'un comportement pathologique :

— Soit les mainteneurs du dépôt officiel ont fait diverger master. C'est parfois nécessaire, mais très rare, et un mainteneur responsable vous informera lorsqu'il doit en venir à de tels extrêmes.

— Soit, cas de loin le plus courant, vous avez vous-même créé une divergence, typiquement en ajoutant des commits sur votre master local.

Dans ce dernier cas, pour vous tirer d'affaire, vous pouvez :

— Si vous le souhaitez, sauvegarder vos changements dans une nouvelle branche avec `git branch <nom>`.

— Forcer votre branche master à revenir au même point que le master officiel avec la commande `git reset --hard upstream/master/main`.

Construire des changements dans une branche dédiée

Une fois que l'on a un master à jour, il est possible de construire des changements par-dessus ce dernier. Il suffit pour cela de créer une branche qui part de master et de commencer à créer des commits dessus :

1. Git mettra à jour votre brache si elle peut être *fast-forwarded* sans créer des nouveaux commits

```
$ git checkout master/main
$ git checkout -b changements
$ echo -e "\nCeci n'est pas un pied de page" >> texte
$ git add texte
$ git commit -m "Ajout d'un pied de page rebelle"
```

Mais pendant que vous effectuez vos modifications, le dépôt officiel continue d'évoluer, et il est possible que des nouveautés arrivent sur la branche master. Les sorties des commandes `git fetch` et `git pull` vous en informeront.

Si c'est le cas, vous pourrez intégrer ces nouveautés à votre branche de développe- ment avec les outils que nous avons vus précédemment. Un cycle complet de mise à jour de master et d'intégrations de ses nouveautés à votre branche pourrait ainsi ressembler à ceci :

```
$ git checkout master/main
$ git pull --ff-only
$ git checkout changements
$ git merge master/main
```

En cas de conflit de fusion, il vous appartiendra de vous adapter aux nouveautés de la branche master officielle qui sont incompatibles avec vos changements.

En règle général, plus une de vos branches a une durée de vie longue, plus il y a de chances que cela se produise. De plus, si vous utilisez `git rebase` plutôt que `git merge`, comme la politique de certains projets l'exige, il peut aussi arriver que vous rencontriez de conflits de fusion en cascade, où vous devrez faire face à des conflits non pas une seule fois, mais une fois par commit de votre branche !

Pour éviter cela et simplifier le passage en revue de vos changements par les mainteneurs, il est recommandé de privilégier des branches de durée de vie courte, aux objectifs précis et modestes, que l'on soumettra au dépôt officiel dès que possible. Nous allons maintenant étudier ce processus de soumission.

Mettre en ligne des changements sur son fork

Lorsque vous êtes satisfait de votre branche locale, vous pouvez la mettre en ligne sur votre fork avec la commande `git push`.

L'option `-set-upstream`, que nous avons évoquée précédemment, vous permettra de mémoriser l'association entre votre branche locale et la branche présente sur votre fork, et ainsi de pousser avec un simple git push par la suite. Note : pour exécuter cette commande il faut associer votre projet à un token ; suivez les instructions ici : `https://docs.github.com/es/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token`.

```
$ git push --set-upstream origin changements
```

Nous l'avons évoqué précédemment, dès lors que vous avez mis des changements en ligne sur un serveur, il faut être prudent avec les modifications d'historique et les `git push -f` associés pour ne pas "couper l'herbe sous le pied" aux gens qui travaillent par-dessus vos branches.

Mais l'un des avantages qu'il y a à utiliser un fork, c'est que ce dépôt n'appartient qu'à vous. Personne d'autre n'est donc censé baser son travail dessus, et c'est pourquoi la plupart des projets tolèrent l'utilisation libérale du `force-push` sur les forks, un confort très appréciable pour les amateurs d'édition d'historique.

Gardez cependant à l'esprit que cette règle de vie est à double tranchant. Si vous devez un jour baser votre travail sur le fork de quelqu'un d'autre, il vous appartiendra de vivre avec les suppressions de branches et autres modifications d'historique effectuées par cette personne sur son fork. Evitez donc si possible de baser du travail sur le fork d'autrui.

Soumission au dépôt officiel

Une fois que vous avez mis des changements en ligne sous la forme d'une branche sur votre *fork*, vous pouvez demander l'intégration de ces changements au dépôt officiel avec une fonctionnalité des forges git modernes, la pull request. Cette fonctionnalité est également appelée merge request par GitLab, mais le terme pull request étant plus répandu, et la forge GitHub que nous utilisons dans ce TP l'employant, c'est celui que nous utiliserons par la suite.

Cette possibilité vous sera généralement directement offerte par l'affichage d'une URL dans la console lorsque vous poussez la branche sur votre fork. Mais vous avez manqué ce raccourci, vous pouvez créer "manuellement" une pull request en allant dans l'interface web de votre fork, affichant la liste des branches, et cliquant sur le bouton "New pull request".

Comme son nom l'indique, une `pull request` invite le(s) mainteneur(s) du dépôt officiel à intégrer des changements au projet en fusionnant les nouveautés de votre branche dans un historique officiel (généralement la branche master). Vous êtes invité à donner un titre court à votre branche, ainsi qu'une description qui peut être plus étoffée et agrémentée de mise en page Markdown. Selon la forge que vous utilisez, d'autres fonctionnalités de gestion de projet peuvent vous être proposées lors de la création d'une `pull request`, nous vous invitons à consulter la documentation de votre forge pour en savoir plus.

Pour les mainteneurs, une `pull request` se présente sous la forme du titre et de la description que vous avez donnés, d'une visualisation des changements proposés, et d'une interface permettant de discuter desdits changements (pour proposer des modifications avant intégration par exemple).

Il est aussi possible pour le mainteneur de définir une batterie de tests automatisés que toute pull request doit passer avant d'être acceptée par le projet. On parle alors d'intégration continue.

Si des changements sont demandés par le mainteneur, vous pouvez ajouter ceux-ci à votre branche locale via de nouveaux commits et de les pousser sur votre fork avec un simple git push. La pull request associée sera alors mise à jour automatiquement

A vous de jouer !

A ce stade, vous en savez assez pour commencer à éditer les fichiers disponibles sur : `https://github.com/cguziolo/Examples_git_medev.git`.

Des exemples des modifications possibles sont la proposition d'une traduction vers une autre langue que l'anglais des textes en anglais. Aussi vous pouvez travailler avec le texte en français et proposer des nouveaux noms de personnages et villes, ainsi qu'une fin à cette histoire.

Avec chaque modification, n'hésitez pas à créer des commits en cours de route pour vous permettre de revenir facilement en arrière. Puis, quand vous aurez fini, poussez votre branche finale sur votre fork, et soumettez une `pull request` au dépôt officiel pour terminer ce TP.

Conclusion

Dans ce TP, nous avons vu comment utiliser git en réseau, avec une méthode de travail basée sur les forks et les pull requests. Si ce n'est pas la seule manière d'utiliser git, cette méthode est une des plus populaires aujourd'hui, car elle sécurise le dépôt officiel, facilite les contributions extérieures, et évite certains problèmes liés à l'utilisation naïve d'un dépôt git partagé (par exemple un "force-push" malencontrueux sur une branche du dépôt principal qui sème le chaos dans le projet).

Exercices

1. Partez de votre clone du dépôt `Examples_git_medev`.
2. Vérifiez que les URLs de vos deux remotes sont correctes : origin doit pointer sur votre fork, upstream doit pointer sur le dépôt officiel, et les deux remotes doivent utiliser une connexion HTTPS.

3. Créez une nouvelle branche, et ajoutez-y un commit.
4. Poussez cette branche sur votre fork.
5. Ajoutez deux nouveaux commits à la branche, puis poussez-les sur votre fork aussi.
6. Mettez à jour votre dépôt vis à vis des éventuelles nouveautés du dépôt officiel du projet.
7. Détruisez la version locale de votre nouvelle branche avec git branch -D, puis régénérez-la à partir de la version sauvegardée sur votre dépôt.

Informations complémentaires

Authentification SSH

Arrivé à la fin de ce TP, vous en avez probablement d'ores et déjà assez de taper votre mot de passe GitHub sans arrêt. Inutile, donc, de dire que vous ne vous imagineriez pas le faire tous les jours. Heureusement, des solutions existent. Une méthode à la fois élégante et sécurisée pour vous authentifier sans mot de passe sur un dépôt git est d'utiliser le protocole à clé publique de SSH. L'idée est de générer une paire clé privée / clé publique comme vous le feriez pour vous authentifier sur un serveur Unix distant en SSH, puis de configurer la clé publique associée dans votre forge logicielle. Une fois ceci fait, git se chargera ensuite d'envoyer à votre forge des commandes signées numériquement avec votre clé privée, et la forge pourra utiliser votre clé publique pour s'assurer que ces commandes viennent bien de vous et non d'un imposteur qui essaie de se faire passer pour vous. Pour se passer totalement de mot de passe, votre clé privée doit être stockée de façon non chiffrée (pas de "passphrase"). Un tel stockage offre une sécurité suffisante à condition que votre ordinateur remplisse les prérequis suivants :

— Votre dossier personnel ("home") est situé sur un stockage chiffré, et à l'abri du regard des autres utilisateurs si l'ordinateur est partagé.
— Vous n'exécutez que des logiciels de confiance, non susceptibles d'espionner votre clé privée et de la transmettre à des tiers malveillants.

La procédure de configuration exacte dépend de la forge logicielle utilisée, mais les étapes suivantes sont généralement nécessaires :

1. Vérifier que vous êtes autorisé à vous connecter à des serveurs SSH distants (certains administrateurs réseau configurent leurs pare-feux de façon si restrictive que c'est impossible, il faudra alors vous arranger avec eux).
2. Générer une paire de clés (privée et publique) avec ssh-keygen.
3. Saisir votre clé publique dans l'interface de configuration de la forge.
4. Configurer les remotes git pour utiliser une URL SSH (celles-ci sont générale- ment de la forme git@<serveur> :<utilisateur>/<dépôt>.git).