Labo 03 – Git & GitHub

Exercices

Mathieu Lemieux © 2021

- Sur votre poste de travail, créer d'abord un dossier Labo03; c'est ici que nous créerons tous les dépôts (*repository*) à venir pour l'exercice de ce soir. Ouvrez un terminal et assurez-vous que le répertoire courant correspond bel et bien à ce dossier.

Partie I – Introduction aux fonctionnalités de Git

Initialisation

- **init**. Initialisez un nouveau dépôt repo01 avec la commande 'git init repo01'. Le dossier repo01 est créé pour vous (si le nom du dépôt est omis dans la commande, c'est le dossier courant qui est initialisé). Déplacez-vous dans ce dossier avec 'cd repo01'. À l'intérieur de ce dossier se trouve un dossier .git qui est normalement en mode caché; c'est l'indication que vous êtes dans le dossier racine d'un dépôt Git. Ne modifiez jamais directement ce dossier ou les fichiers à l'intérieur!

Suivi des fichiers

- **status**. Créez un nouveau fichier texte vide nommé readme.md et placez-le dans le dossier repo01. Vérifiez le statut du dépôt avec la commande 'git status'. Apparemment notre nouveau fichier n'est pas suivi (*untracked*); par défaut, Git ne fait le suivi des changements que sur les fichiers que l'on spécifie vouloir suivre.

- **add**. Pour amorcer le suivi des changements sur ce fichier, exécutez la commande 'git add readme.md'. Réaffichez le statut du dépôt; le fichier readme.md est maintenant ajouté à la liste des fichiers suivis (*new file : cela constitue en soi un changement pour Git*).

- **.gitignore**. Au lieu d'ajouter un à un les fichiers à suivre (ça pourrait être long avec des centaines de fichiers de code!), on peut tous les ajouter à la fois avec la commande 'git add .' On gère alors les exceptions en créant un fichier spécial nommé .gitignore. Ce fichier va contenir la liste des règles à appliquer pour le suivi des fichiers. Créez ce fichier et ajoutez-y une exception pour les fichiers d'extension .zip en inscrivant '*.zip' sur la première ligne du fichier. Sauvegarder. Faites un test en ajoutant un fichier test.zip dans le dossier repo01 et en exécutant 'git add .'.

- **ls-files**. Utilisez la commande 'git ls-files' pour afficher la liste des fichiers suivis par Git; le fichier test.zip ne devrait pas s'y trouver.

Choix des changements à intégrer (*staging*)

- **add**. Git permet d'établir des points de contrôle (aka *commits*) vers lesquels nous pouvons revenir en cas de besoin. Ceux-ci constituent en quelque sorte des portraits instantanés de l'état des fichiers échelonnés dans le temps. On peut choisir quels changements seront intégrés au prochain commit (aka *Cherry picking*) ou bien, heureusement pour nous, tous les ajouter du même coup. L'endroit où les récentes modifications attendent d'être ajoutés au prochain commit s'appelle le '*staging area*'. Nous pouvons ajouter les changements à l'aide de la même commande 'git add .' utilisée pour le suivi des fichiers; cette commande ajoute les fichiers non-suivi à la liste des fichiers à suivre tout en ajoutant les fichiers suivis avec changements à la liste des changements à intégrer au prochain commit. Écrivez quelques mots dans le fichier readme.md puis sauvegarder. Un coup d'œil au statut nous révèle que des changements ont été apportés; faites-en le *staging* avec 'git add .'.

- **reset**. Vous pouvez retirer un fichier du *staging area* avec la commande 'git reset <fichier>' ou tout retirer avec 'git reset'. Retirez le readme.md, consultez le statut, puis ajoutez-le de nouveau.

Intégration dans un point de contrôle (*commit*)

- **commit**. Chaque commit doit être accompagné d'un message descriptif; on utilise l'option -m pour écrire notre message de commit directement dans la ligne de commande. Exécutez la commande 'git commit -m 'Première ébauche de readme + .gitignore''. Faites ensuite d'autres changements dans le fichier readme.md. Cette fois, inutile d'exécuter préalablement 'git add .', nous feront les ajouts en même temps que le commit avec la commande 'git commit -am 'Deuxieme ebauche du readme''. L'option -a remplace 'git add .' pour le *staging* seulement; nous devons toujours ajouter manuellement le suivi des nouveaux fichiers. Nous utiliserons cette option pour la suite de l'exercice. Refaite encore au moins une fois le cycle changements/commits pour ajouter un peu d'historique à notre dépôt.

Revenir en arrière

- **log**. Regardons d'abord l'historique des *commits* avec la commande 'git log'.

- **Reset --hard**. Si vous regrettez vos nouveaux changement et désirez retourner en arrière, comment faites-vous? Voici quelques option :

  - Effacer tous les changements apportés depuis le plus récent commit : 'git reset --hard'
  - Remonter à un commit spécifique (et effacer tout ce qui venait après) : 'git reset –hard <SHA>'
  - Remonter un certain nombre de commits de façon relative : 'git reset –hard ~<nb>'

  Pour l'exercice, copiez l'identifiant SHA du commit 'Deuxième ébauche du readme' et utilisez la commande 'git reset –hard <SHA>' puis refaites un 'git log'.

  Attention! Il existe plusieurs subtilités lorsque l'on modifie l'historique comme on vient de le faire et certaines précautions sont de mise. En outre, vous ne devriez jamais effacer des commits que vous avez partagé avec d'autres utilisateurs!

Embranchements

- **branch**. Nous aimerions tester des modifications à notre fichier readme.md mais sans perdre l'état actuel du fichier. Vous avez sûrement déjà dupliqué des fichiers de code ou des documents Word pour conserver des anciennes versions tout en travaillant sur une nouvelle (ex. fichier.doc, fichierV2.doc, fichierV3.doc, etc.). Git a une solution qui s'appelle les embranchement (*branching*). Pour créer une nouvelle branche nommée image, exécutez 'git branch image'. Exécuter ensuit 'git branch' pour voir la liste des branches existantes.

- **checkout**. Par défaut, depuis le début, nous travaillons dans la branche master; pour quitter cette branche et nous diriger plutôt dans la nouvelle branche image, exécutez 'git checkout image'. Faites ensuite les opérations suivantes :

    1. Trouvez une image à votre goût sur le web, téléchargée-là dans le dossier repo01, puis modifiez le fichier readme.md en lui ajoutant une image à l'aide des fonctionnalités Markdown (Cheat sheet ici : https://packetlife.net/media/library/16/Markdown.pdf). Hint : '![<text>](./<filename>)'. Sauvegarder/ajout/commit.

    2. Regarder le contenu de repo01. Entrez la commande 'git checkout master' pour revenir à la branche principale puis regarder à nouveau le contenu du dossier.

    3. Toujours sur la branche master, télécharger une nouvelle image dans repo01 et remplacez tout le contenu du readme.md en ajoutant la nouvelle image en Markdown. Sauvegarder/ajout/commit.

Fusion et gestion des conflits

- **merge**. Maintenant on aimerait fusionner les changements présents dans la branche image avec ceux de la branche master, qui a subis ses propres changements. Toujours à partir de la branche master, exécutez 'git merge image'. Regardez le message qui vous est retourné…

    - Si aucun conflit n'avait été détecté, la fusion aurait eu lieu sans étapes additionnelles. Dans notre cas, Git a identifié des conflits que nous devrons régler avant de poursuivre. Il est indiqué que ces conflits se trouvent dans le fichier readme.md; ouvrez-le avec un éditeur de texte. Que voyez-vous?

    - La ligne '=======' sépare la version locale (en haut) des changements que l'on veut y intégrer (en bas). Décidez de la version finale que vous désirez conserver (gardons seulement l'affichage des 2 images, sur 2 lignes séparées) puis enlevez complètement les lignes '=======', '<<<<<<< HEAD' et '>>>>>>> image'. Sauvegarder/ajout/commit.

Dépôts distants

- **remote**. Par sécurité, on aimerait bien garder une copie en ligne de notre dépôt local. On va se connecter à notre compte GitHub et créer un nouveau dépôt public repo01, qu'on va laisser vide pour l'instant. Copiez l'url https dans le presse-papier. Si vous avez installé GitHub CLI, vous pouvez créer ce dépôt distant à l'aide de la commande 'gh repo create repo01'. De retour dans notre terminal, on ajoute une association avec un serveur distant à l'aide de la commande 'git remote add origin <url>' (origin est le nom conventionnel pour la référence au dépôt distant principal mais vous pouvez utiliser un autre nom). La commande 'git remote -v' permet de faire la liste des dépôts distants associés à ce dépôt; vous pourriez en définir plusieurs.

- **push**. Assurez-vous d'abord que les derniers changements ont été commis en vérifiant le statut; faites un commit au besoin. Mettez ensuite à jour le dépôt distant à l'aide de la commande 'git push origin master'. Allez sur votre compte GitHub et vérifiez que les changements ont bel et bien été apportés. Notez que vous avez pu faire un *push* sur origin car vous détenez les droit d'accès en écriture à ce dépôt distant. Dans le cas contraire, on fait plutôt un *pull request* pour demander au responsable de réviser notre code et de l'intégrer au code en ligne; on verra peut-être ça plus tard dans la session si ça vous intéresse.

- **clone**. On peut aussi recopier localement le contenu d'un dépôt distant. Allez sur le dépôt GitHub des labos du cours pour obtenir son URL https; nous allons nous en servir pour cloner localement ce dépôt. Dans le terminal, assurez-vous d'être revenu dans le dossier labo03 avec la commande 'cd ..'. Exécutez la commande 'git clone <url>'. Vous devriez voir apparaître un nouveau dossier dans labo03. À l'intérieur, vous avez une copie du dépôt. Vous pouvez renommer ce dossier sans affecter quoi que ce soit.

- **pull**. Déplacez-vous dans ce dépôt avec le terminal. De mon côté, je vais effectuer des modifications sur mon poste de travail puis faire un push vers le dépôt en ligne. À mon signal, mettez à jour votre copie locale avec la commande 'git pull'. Notez qu'une référence à un dépôt distant, nommée origin, a été ajouté automatiquement au moment du clonage; vous pouvez vérifier avec 'git remote -v'. Vous avez maintenant les derniers changements que j'ai apporté au fichier readme.md.

- **GitHub fork**. Lorsque l'on veut collaborer à un projet *open source*, GitHub propose une fonctionnalité de *forking*; au lieu de cloner directement le dépôt public d'intérêt, vous en faites d'abord une copie sur GitHub (à l'aide de l'option *fork*). Vous faites ensuite un clone local de ce fork, et vous créez une nouvelle branche pour les changements suggérés. Vous pouvez alors mettre régulièrement à jour votre copie sur GitHub et lorsque votre code est prêt, vous faites un *pull request* afin que votre branche soit intégrée à la branche principale sur le dépôt d'origine. Nous ne pratiquerons pas ensemble cette fonctionnalité pendant le laboratoire mais c'est bon de savoir que ça existe…

Partie II – Exercice de travail collaboratif

Voici les commandes que nous allons utiliser pour cet exercice :

```
git init <name>
git add .
git commit -am '<message>'
git remote add origin <url>
git push origin master
git clone <url>
git pull
git status
```

- Sur votre poste de travail, à l'aide d'un terminal dans Labo03, initialisez un nouveau dépôt nommé repo02 puis déplacez-vous dans le nouveau dossier ainsi créé. Ajoutez-y un fichier code.py et mettez-y un peu de contenu sur quelques lignes : écrivez une fonction avec un bloc conditionnel (ou une expression conditionnelle!) qui affiche un message quelconque; n'oubliez pas l'appel de cette fonction pour que l'on puisse exécuter le fichier. Sauvegarder/ajout/commit.

- Allez sur GitHub et créez un nouveau dépôt privé vide nommé repo02. Vous allez m'ajouter comme collaborateur (mathieulemieux) à ce dépôt (Dans l'onglet *Settings*, puis *Manage access* sur la gauche). Puisqu'on utilise une version gratuite de GitHub, vous ne pourrez pas, je crois, limiter mes accès mais c'est sans importance pour cet exercice. Copiez l'adresse https.

- De retour sur votre poste de travail, ajoutez la référence au dépôt distant puis mettez celui-ci à jour avec vos changements locaux. Faites-moi signe quand c'est fait!

- De mon côté, pendant ce temps, j'ai fait un clone local de chacun de vos dépôts. À votre signal, je vais faire une mise-à-jour avec vos changements, modifier un peu le code et remettre à jour le dépôt en ligne. Attendez mon signal avant de passer à la prochaine étape.

- Dans votre copie locale du fichier code.py, modifiez le nom de la fonction (ainsi que l'appel). Sauvegarder/commit. Tentez de mettre à jour le dépôt distant; la tentative a échoué car l'état d'avancement de la branche master est plus avancée sur le dépôt distant que sur le dépôt local!

  - Faites donc d'abord un 'git pull' pour mettre à jour votre copie locale avec les changements distants. Surprise : *merge conflict*! Comme vu précédemment avec la fusion de deux branches distinctes, la fusion de deux versions distinctes (commits) de la même branche peut aussi générer des conflits. Ne soyez pas découragé, le travail collaboratif ne génère pas toujours des conflits, là c'est un peu fait exprès…

  - Réglez le conflit comme on a vu avec la fusion des branches; si vous avez de la difficulté c'est le moment de demander de l'aide! Vous pouvez ensuite faire un commit et réessayer le push tenté initialement, tout devrait fonctionner. Allez voir sur GitHub l'état du dépôt. Vous pouvez entre-autres consulter l'historique des commits avec les messages et les auteurs, visualiser le contenu des fichiers avec les ajouts et les suppressions, et bien plus encore!

Partie III – Exercice avec GitHub Pages

- Cette dernière partie peut facilement être faite seul de votre côté si on n'a pas le temps ensemble pendant le labo. Créez un dépôt public vide sur GitHub. Faites-vous un clone local, puis ajoutez un dossier nommé docs. Dans ce dossier, créez un fichier index.html. Ajoutez-y le code suivant :

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h1>Ma première page web!</h1>
    </body>
</html>
```

- Sauvegarder/ajout/commit/push.

- Sur GitHub, dans l'onglet *Settings* de votre dépôt, restez dans les options et descendez jusqu'à la section GitHub Pages. Comme source, choisissez la branche master puis le dossier docs, puis sauvegardez. Pendant la sauvegarde, vous avez été ramené au haut de la page; revenez à la section GitHub page, puis visitez l'URL indiquée. Il faut parfois quelques minutes avant que les changements ne fassent effet; revenez-y plus tard au besoin. Cette page web est maintenant accessible à tous. Félicitation, vous venez de publier votre première (ou pas) page web!