


Introduction à git


Après les deux TP que nous passerons sur git, vous devrez rédiger une fiche de synthèse qui vous servira plus tard à vous rappeler des commandes git les plus importantes. Vous rendrez cette fiche avant le vendredi 11 février. 1 ou 2 pages maximum, dans votre format favoris. (mais le pdf est le format le plus lisible si vous devez un jour consulter votre fiche depuis votre téléphone.)

Un site internet de référence est maintenu par la communauté, il contient des tutoriels avancés et un livre entier gratuit sur git. (disponible en français)

Git

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local

 <https://git-scm.com/>



Si vous avez un doute sur une commande ou une syntaxe, vous pouvez aller consulter le manuel <https://git-scm.com/docs>. (en anglais seulement)

Installation et Configuration

Avant tout il faudrait installer git sur votre machine, git est déjà installé sur les machines de la salle mais si vous voulez utiliser git depuis chez vous, il vous faudra l'installer.

Comme c'est la première fois que vous allez lancer git sur la machine, il va falloir configurer votre nom et votre email (Prévoyez d'utilisez la même email que pour votre futur compte sur une plateforme de stockage de dépôt git en ligne et un nom d'utilisateur type PrenomNOM). Cette configuration peut se faire depuis n'importe quel dossier, ouvrez un git bash quelque part et saisissez

```
git config --global user.name "Votre Nom"
git config --global user.email "Votre email"
```

Votre premier dépôt git

Nous allons créer un projet à partir de rien. Créer un dossier dans votre répertoire personnel et accédez y depuis un git Bash. Lancez alors la commande

```
git init
```

Vous avez créé un dépôt git dans ce dossier, à partir de maintenant, git va faire attention à ce qu'il s'y passe.

git parle en anglais, dans la suite, "repository" sera traduit par dépôt

Un dossier caché ?!

A priori, il ne s'est rien passé dans le répertoire mais remarquez que votre git bash précise maintenant ([master](#)), nous reviendrons plus tard sur ce point.

Commençons par observer ce que git a fait : lancez la commande que vous connaissez bien

```
ls -a
```

Que remarquez-vous ? Si vous entrez dans le dossier, votre git Bash annonce ([GIT-DIR!](#)) Allez un peu explorer les fichiers et dossiers contenus dans ce dossier caché. C'est dans ce dossier que git va stocker les modifications apportées à votre projet.

En général, nous n'irons pas voir ce qu'il s'y passe et nous laisserons git gérer ce dossier.

Une commande pour connaître la situation

Entrez la commande

```
git status
```

Cette commande vous montre 3 choses :

1. Vous vous trouvez sur la branche 'master'. C'est votre branche principale, par défaut, toutes vos branches principales s'appelleront 'master'. *C'est pour cela que git bash indique (master).*
2. Vous n'avez pour le moment aucun commit. *Les commits sont les points de sauvegarde dont on a parlé dans le cm.*
3. Votre répertoire de travail est propre, toutes les modifications sont enregistrées dans le dépôt. (il n'y en a aucune pour le moment)

Cette commande nous sera très utile ! Dès que nous aurons un doute sur l'état de notre dépôt et de nos fichiers, nous lancerons un `git status` pour avoir un résumé de la situation.

Commençons à travailler

Le plupart des éditeurs "intelligents" vont reconnaître qu'un dépôt git est initialisé dans le dossier. Visual Studio indique le nom de la branche courante en bas à droite.

Créez dans le dossier un fichier nommé `index.html` et copiez ce contenu

```
<!DOCTYPE html>
<html>
<body>

<h1>Je m'appelle -Votre Nom- </h1>
<p>Les mathématiques sont ma passion, et j'apprends git !</p>

</body>
</html>
```

Pourquoi un fichier html ?

A la fin du tp, nous hébergerons notre projet sur la plateforme github qui possède une fonctionnalité appelée github pages. Elle vous permet d'héberger votre propre page sous une url de la forme "username.github.io".

Lorsque vous candidateriez à des emplois, une page à votre nom qui décrit votre projet sera un plus pour votre profil. En plus, cela montrera à l'employeur que vous êtes familiers avec git ! Profiter de ce tp pour en construire les fondations, vous pourrez y revenir plus tard.

Tout ce que nous allons faire fonctionne aussi pour des projets en c++, java, .sql, tout ce qui fonctionne à base de ligne de caractères.

Enregistrer le fichier et relancer la commande `git status`, que dit-elle ?

1. Nous nous trouvons toujours sur la branche 'master'.
2. Nous n'avons toujours aucun commit.
3. Il n'y a des fichiers qui ne sont pas suivis par git: git voit que le fichier index.html a été crée mais qu'il n'est pas suivi.

La commande qui permet de dire à git qu'il faut prendre en compte les modifications apportées au fichier est

```
git add index.html
```

Le fichier est alors considéré dans la zone de préparation (*staged zone en anglais.*)

Relancer encore une fois la commande `git status`, que dit-elle ?

1. Toujours sur la branche master
2. Toujours aucun commit.
3. Toutes les modifications ont été ajoutées à la zone de préparation, git est prêt à créer un point de sauvegarde avec un commit.

Votre premier commit

Un commit doit toujours être accompagné d'un message. Ce message a pour but de décrire ce qu'on a modifié depuis le dernier commit. Pour notre premier commit, vous pouvez saisir

```
git commit -m "Première version du projet"
```

En réponse à ce commit, git nous dit combien de fichiers ont été modifié et le nombre de lignes qui ont été insérée. Il dirait aussi le nombre de lignes supprimées le cas échéant.

Relançons encore une fois un git status, que répond-elle ?

1. On est toujours sur la branche 'master'
2. L'arbre de travail est propre, les fichiers sont comme ils l'étaient au dernier commit (*celui que l'on vient de faire.*)

Faisons un deuxième commit, modifiez le texte entre les balises <p> pour ajouter: “ ... apprends git, mais j'ai déjà fait un commit !</p>”

Ajoutez également un deuxième paragraphe sur votre saé par exemple

```
<p>J'ai participé à un projet qui visait à programmer une IA capable de gagner dans un jeu de morpion.</p>
```

- Ajoutez ces modifications à la zone de préparation ainsi qu'un nouveau fichier vide intitulé 'sae.html' et faites un deuxième commit avec le message “ajout de précisions, création de sae.html”

git annonce 2 nouvelles lignes, 1 ligne en moins. Comprenez-vous pourquoi ?

git n'a pas l'étiquette “ligne modifiée”. Si une ligne a été modifiée même d'un seul caractère, pour git, c'est comme si on avait effacé toute la ligne et qu'on l'avait réécrite en entier.

Pour ajouter plusieurs fichiers à la zone de préparation, pas obligé de les faire un par un, on peut lancer la commande `git add .`

Cette commande va ajouter tous les fichiers modifiés du dossier à la zone de préparation, mais il peut y avoir certains fichiers ou dossiers qu'on ne veut pas prendre en compte, pour cela, on crée un fichier intitulé .gitignore et on liste à l'intérieur le nom des fichiers ou dossiers qu'on ne veut pas suivre.

La syntaxe est la suivante: chaque ligne de ce fichier correspond à un fichier/dossier à ignorer. Pour ignorer un fichier, on écrit NomDuFichier et on passe à la ligne. Pour ignorer un dossier, on écrit /NomDuDossier et on passe à la ligne.

Par exemple, vous voudrez ignorer les fichiers compilés en .class ou les a.exe par exemple, puisqu'il vont être source de conflit à chaque compilation. Les dossiers où sont installés des bibliothèques comme node_modules également

Voir l'historique des commits

On peut voir tous les points de sauvegarde du projet avec la commande

```
git log
```

Cette commande liste les derniers commits effectués sur la branche en cours. Chaque commit possède un identifiant unique, qui est une suite de caractère hexadécimaux. Ces caractères sont obtenus à l'aide d'une fonction de hachage.

*Une **fonction de hachage** est un énorme programme composés principalement de ET et de OU arrangés de manière extrêmement compliquée. Elle peut prendre en entrée une chaîne de caractère, un fichier texte, une image, même plusieurs fichiers et des centaines d'images. Elle passe tous ces fichiers dans ses portes logiques extrêmement compliquées et elle renvoie une chaîne de caractère hexadécimaux.*

Naviguer dans les commits

On peut revenir facilement au commit précédent avec la commande (*attention cette commande va ouvrir un éditeur de texte*)

```
git revert HEAD~1
```

Cette commande va restaurer les fichiers dans leur état du dernier commit et créer un nouveau commit pour sauvegarder cet état. Ce commit nécessite un message, la commande va ouvrir un éditeur de texte avec le message par défaut 'Revert "message du dernier commit" '. Si vous êtes d'accord, vous pouvez fermer l'éditeur et le commit sera créé.

Vous pouvez alors constater que le fichier 'sae.html' dans votre explorateur a disparu et que le fichier index.html est revenu à son état d'origine.

La commande `git log` affiche bien le commit supplémentaire avec le message.

- Effectuer un nouveau git revert pour annuler le commit qui avait annulé le dernier commit.

On peut revenir plusieurs commits en arrière, à la place de HEAD:

1. HEAD~2 revient de 2 commits en arrière
2. HEAD~3 revient de 3 commits en arrière
3. etc ...

Il est également possible de revenir dans un état du projet bien antérieur en précisant le début du hash du commit auquel on veut revenir avec la commande `git reset` mais nous ne nous attarderons pas plus ici.

Une autre commande que nous aimerions évoquer est la commande `git rebase`. Si l'historique de votre git était un arbre, git rebase permet de tronçonner une branche pour la recoller depuis un autre commit de la branche principale. La fusion devient alors beaucoup plus lisible et cela évite les commits de fusion qui polluent un peu l'historique des commits. (un article complet en français : <https://www.miximum.fr/blog/git-rebase/>)

Votre première Branche

Une branche est une version indépendante du projet sur laquelle on travaille en général pour conserver une version fonctionnelle du projet. Lorsqu'on a atteint l'objectif, on fusionne la nouvelle branche avec la branche principale.

Nous allons créer une nouvelle branche pour ajouter nos coordonnées sur la page web que nous sommes en train de développer.

Pour créer une nouvelle branche qui s'appelle coordonnees, on saisit

```
git branch coordonnees
```

Nous avons créé une branche mais nous sommes toujours sur la branche principale. Pour aller sur la nouvelle branche, on utilise

```
git checkout coordonnees
```

Pour avoir une vision des branches existantes, on peut utiliser la commande

```
git branch
```

(on peut créer une branche et y basculer directement en utilisant l'option -b sur la commande checkout)

Apportez vos modifications au fichier html pour ajouter vos coordonnées:

- Votre statut: “étudiant au BUT Informatique, Nevers” par exemple
- Votre email (ne le laissez pas sous forme cliquable, sinon attention aux bots : utilisez par exemple ‘prenom . nom AT u-bourgogne.fr’ avec le AT pour remplacer l’arobase.

Ajouter le fichier à la zone de préparation puis faite un commit.

La fusion des branches

Lorsqu’on a terminé de développer la nouvelle fonctionnalité, on peut fusionner la nouvelle branche avec la branche principale.

Replacer vous sur la branche master avec

```
git checkout master
```

Modifier le fichier index.html à la ligne où vous annoncez votre nom pour ajouter “et voici ma page personnelle”. Ajouter cette modification à git avec un nouveau commit.

Même si nous avons travaillé sur le même fichier dans deux branches différentes, la fusion va bien se passer parce que nous n’avons pas travaillé sur les mêmes lignes du fichier.

Fusionner les deux branches depuis la branche principale avec

```
git merge coordonnees
```


Si vous lancez un git branch, vous remarquerez que la branche coordonnees est toujours présente. Quand on fait une fusion, on ne modifie que la branche sur laquelle on fusionne (en général, c'est la branche principale)

pour supprimer la branche, on peut faire

```
git branch -d coordonnees
```

Votre premier conflit

Créez une nouvelle branche et modifier plusieurs lignes du fichier index.html dans cette branche.

Retournez sur la branche principale et modifiez les mêmes lignes du fichier.

Tentez de fusionner les deux branches. Git va vous annoncer qu'il y a eu un conflit. la fusion automatique n'a pas fonctionnée, vous allez devoir régler le conflit à la main.

Comme on l'a vu en cm git a modifié le fichier à l'endroit du conflit pour afficher les deux versions. Si vous avez un éditeur intelligent (visual studio par exemple) l'éditeur vous propose des options toutes prêtes: Accept incoming change, accept both, ... Si vous éditez vos fichiers au bloc notes, vous devrez modifier à la main, retirer ce que git a ajouté et régler le conflit.

Lorsque vous avez terminé, vous pouvez enregistrer le fichier, l'ajouter à la zone de préparation puis faire un commit pour créer le point de sauvegarde.

Votre premier stockage en ligne

Rendez vous sur github.com et créez un compte (avec la même adresse mail que vous avez renseigné à git au début de ce TP et le même nom d'utilisateur.)

Créez un nouveau dépôt git qui a pour nom username.github.io

- L'option public fait que tout le monde a accès au code
- Avec l'option Private, vous seul avez accès.

Pour synchroniser votre dépôt local avec le dépôt en ligne, tout se fait depuis votre git bash local. Tout d'abord allez copier l'URL du dépôt (trouvable sur la page du

dépôt) puis saisissez dans le git bash

```
git remote add origin URLDeVotreDepotEnLigne
```

Par défaut, le dépôt en ligne est qualifié avec le nom 'origin'.

Il faut maintenant uploader votre code en ligne, pour cela saisissez

```
git push --set-upstream origin master
```

En général, la commande `git push` permet de téléverser son code depuis le dépôt local vers le dépôt en ligne.

Comme votre dépôt en ligne est protégé, il faudra que vous prouviez votre identité pour assurer à github que vous possédez les droits de modifier le dépôt. Github va donc demander votre nom d'utilisateur et votre mot de passe.

En général, on ne saisit pas son mot de passe à chaque fois qu'on veut synchroniser son dépôt en ligne. On met en place des clés SSH, nous expliquerons le principe au prochain TP.

La commande `git pull` permet de télécharger son code depuis le dépôt en ligne vers le dépôt local. Cette commande sert uniquement si plusieurs personnes travaillent sur le même dépôt en ligne, nous en verrons un exemple au prochain TP.

Vous pouvez paramétrer la page username.github.io pour qu'elle soit disponible en ligne et vous pouvez ainsi la partager avec vos futurs CV. Vous n'avez pas d'accès serveur, vous ne pourrez pas envoyer de mail depuis cette page, mais vous pouvez modifier le côté client comme vous voulez en ajoutant d'autres fichiers html, des fichiers javascript, css, des images

La première partie de ce tp s'arrête ici. Dans le second TP, nous verrons comment travailler sur un code partagé sur une plateforme git avec `git clone` et `git fork`.

Si vous avez terminé, commencez à prévoir ce que vous allez placer dans votre fiche de synthèse.