**Bloc 01 – Git**

**1er laboratoire avec Git : l’exploration**

Vous avez tout le reste de la période, et s’il faut vous devez continuer en dehors des heures de classes. À faire individuellement. À remettre pour le début du prochain cours.

**1ère partie : installation de Git**

1. Rendez-vous au site officiel de Git et téléchargez l’installateur : <https://git-scm.com/downloads>.
2. Démarrez l’installateur et sélectionnez les options par défaut tout au long du processus.
3. Une fois l’installation terminée, vous verrez «Git BASH». Nous utiliserons cette interface plutôt que l’interface GUI.

**2ième partie : se familiariser avec Git BASH**

Git Bash est une application de type console. L’utilisation des consoles est très fréquente en développement Web. Nous verrons que nous allons l’utiliser souvent. Même si cela a l’air rebutant au début, il reste néanmoins que cette façon de faire nous permet une plus grande productivité et une plus grande souplesse.

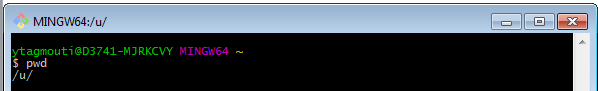
Git Bash est ni plus ni moins qu’un émulateur de BASH, le très populaire interpréteur de commande UNIX/LINUX. En soi, cela ressemble au DOS, mais plus évolué.

Lorsque vous démarrez Git bash, vous verrez quelque chose comme cela :



Ceci est l’invite de commande. Autrement dit, Git Bash attend que vous rentriez une commande. Essayons quelque chose.

Tapez « **PWD** » suivi de **enter**.

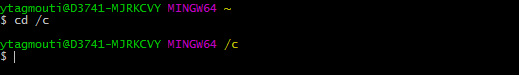


Que se passe-t-il? L’interpréteur de commande Git Bash vous indique dans quel répertoire vous êtes. PWD veut dire **P**RINT **W**ORKING **D**IRECTORY. Pour moi, c’est /u. C’est mon répertoire courant (working directory).

Il faut comprendre que les répertoires sont séparés par des **/** et non des **\**.

Maintenant, explorons un peu le système de fichier!

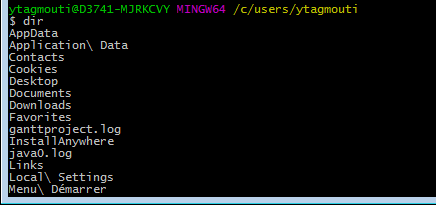
Tapez : «cd /c» (suivi de ENTER. Mais à partir de maintenant, je ne vous dirai plus de le faire, car vous avez compris qu’il faut appuyer sur ENTER pour que la commande s’exécute».



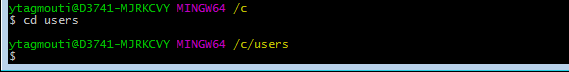
La commande **CD** indique à Git Bash qu’on veut changer de répertoire. CD veut dire «Change Directory». Remarquez ce qu’il y **a** en jaune, dans l’invite de commande. C’est le répertoire en cours.

*Note : Les lecteurs Windows (C:, D :, etc) sont traduits en répertoires avec Git Bash!*

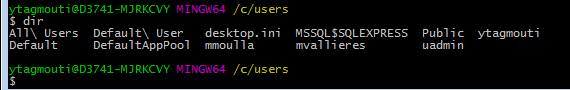
Pour visionner le contenu d’un répertoire, on effectue soit la commande «**dir**», «**ls**», ou «**ls -l**». Allez-y, faites ces trois commandes, et vérifiez les différences entre les trois! Voici un exemple de «**dir -l**»,



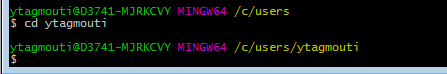
Ensuite, nous allons nous déplacer. Faites «**cd Users**» (remarquez que cd Users et cd users (sans le U majuscule) revient au même).



La commande CD ne fait que changer de répertoire. Remarquez en jaune ce qui a changé!!! Pour afficher le contenu de ce répertoire, il faut lire le contenu

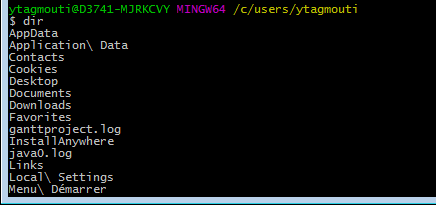


Vous vous y reconnaissez? C’est le répertoire des comptes utilisateurs de Windows! Votre nom de compte devrait s’y retrouver. Pour moi, c’est ytagmouti. Je vais donc aller dans mon répertoire. Mais pour vous, c’est votre nom. Alors vous devrez remplacer ytagmouti par votre nom de compte.

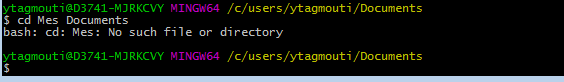


Remarquez en jaune encore une fois. Je me trouve dans , cela s’appelle le **chemin d’accès absolu** du répertoire courant (working directory).

Hey oui! Lorsqu’on se déplace dans un répertoire, il est d’usage d’afficher son contenu. Encore une fois, faisons un DIR :



Nous y sommes presque. Nous allons maintenant allez dans «Mes Documents». (Se seront vos documents). Que devriez-vous taper? «cd Mes Documents» ? Si vous faites cela, bash va vous donner une erreur :



Pour comprendre cela, il faut s’imaginer que Git Bash utilise un certain langage, un peu à la manière d’un langage de programmation. Pour Git Bash, les espaces sont très importants. Par exemple, l’espace entre «CD» et «Mes» permet à Git Bash de dire : «Ah! Il veut que je fasse un cd». S’il n’y avait pas d’espace, Git Bash n’aurait pas compris cela.

En fait, les espaces subséquents servent à délimiter les paramètres que nous envoyons à Git bash. Ainsi, Git Bash interprète notre commande comme un «cd Mes». Comme «Mes» n’est pas un répertoire, alors il nous retourne une erreur. C’est aussi bête et simple que cela.

Vous allez me dire : «Mais comment vais-je faire pour me déplacer dans un répertoire contenant des espaces dans le nom?», et votre question sera très légitime! Pour cela, il y a deux façons de faire : une façon longue, et une courte.

Façon longue : Remplacer les espaces par des «\ » (backslash suivi d’un espace). Le backslash sera à «échapper» (du verbe *escape* en anglais) la séquence. Donc, on peut rentrer cd Mes\ documents.

Façon courte : Tapez simplement «cd Mes», mais cette fois-ci sans appuyer sur enter. Ensuite, tapez sur TAB (touche tabulation). Que se passes-t-il? Git Bash fait ce qu’on appelle une déduction par auto-complétion. Comme il n’y a qu’un seul répertoire qui commence par les lettres «Mes», alors il me propose de compléter l’entrée avec celle qui existe déjà.

Je ne sais pas pour vous, mais moi j’adore cette façon de faire!

Note : S’il y avait plus d’une entrée correspondant au terme de la recherche, Git Bash nous aurait affiché les répertoires trouvés. Essayez avec une seule lettre. Allez-y, vous n’avez rien à perdre et tout à gagner!

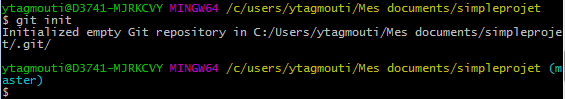
Maintenant, vous devriez être dans votre répertoire «Mes Documents». Votre invite Git Bash devrait ressembler à ceci :



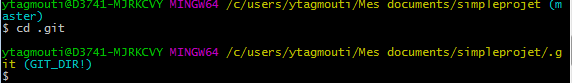
**3ième partie : Création de notre premier projet et dépôt Git**

Nous allons maintenant créer un petit projet web tout simple dans le but d’en activer le contrôle de version avec Git.

1. Créez un répertoire nommé «simpleprojet» dans le répertoire «Mes Documents». (Vous pouvez soit utiliser l’explorateur de fichiers Windows ou soit utiliser la commande MKDIR dans Git Bash».
2. Faites «**cd simpleprojet**»
3. Et, «**git init**».

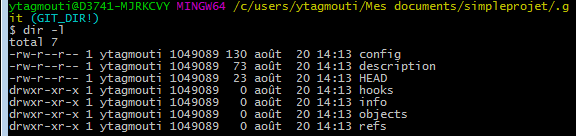


1. Allez dans le répertoire .git :



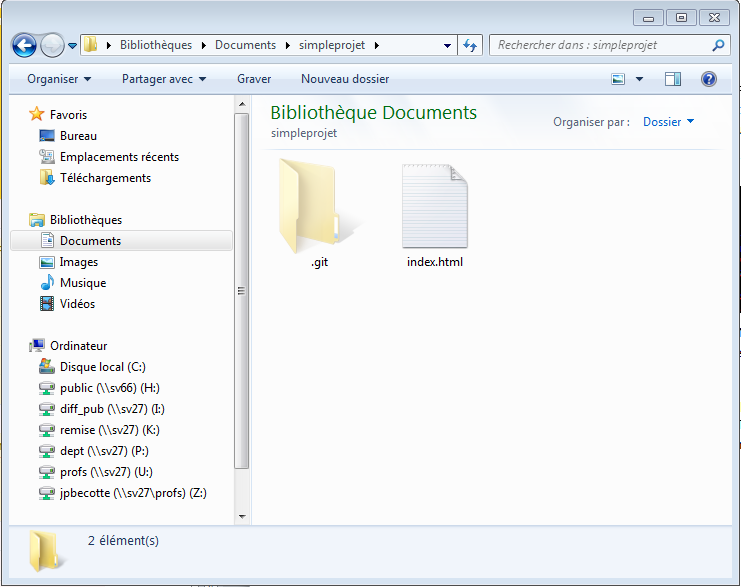
Notez comment Git vous avertis au sujet du répertoir .git (GIT\_DIR!). C’est parce que normalement nous ne devrions pas nous retrouver dans ce répertoire, nous ne sommes pas supposé changer les fichiers ni la structure du répertoire .git. Il faut donc être prudent!

Que se passe-t-il si nous regardons la structure du répertoire .git?



Il n’est pas utile de détailler chacun de ces fichiers ni répertoires. Notons simplement que nous pouvons configurer les options de Git dans le fichier config; et que la base de donnée se trouve dans le répertoire «objects».

1. Créons maintenant une simple page web HTML dans le répertoire simpleprojet. (Vous devrez utiliser l’explorateur de fichiers Windows). ATTENTION! Le fichier ne doit pas se trouver dans le répertoire .git! !!Le fichier doit s’appeler «index.html».



Notez que si vous avez spécifié vos options de dossier correctement, vous pourrez voir le dossier .git! !!

1. Remplissez le contenu du fichier index.html avec votre éditeur préféré avec les lignes suivantes:

<!DOCTYPE html>

<html>

<body>

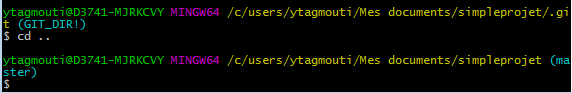
<h1>Voici une simple page web</h1>

<p>Bonjour!</p>

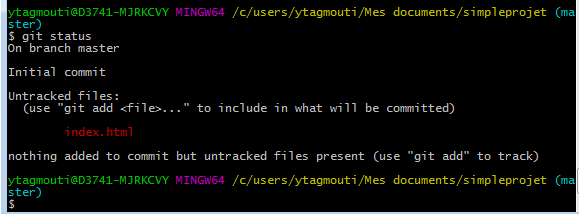
</body>

</html>

1. Voyons ce que Git a à dire (vous devrez revenir en arrière dans la hiérarchie de vos répertoires. La commande **cd ..** permet cela):



Tapez ensuite la commande **git statuts** qui permet d’afficher l’état d’un fichier, rappelez-vous des sections principales d’un projet Git.



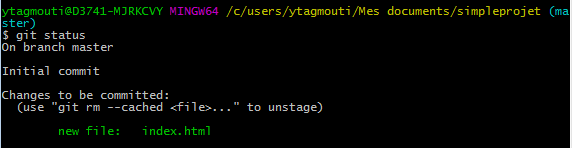
Ici, Git nous indique que index.html est «untrack». Cela veut dire que le fichier ne se trouve même pas dans la base de données.

1. Plaçons le fichier sur le «staging» qui se résume à la zone d’index. Tapez la commande «**dit add . »**



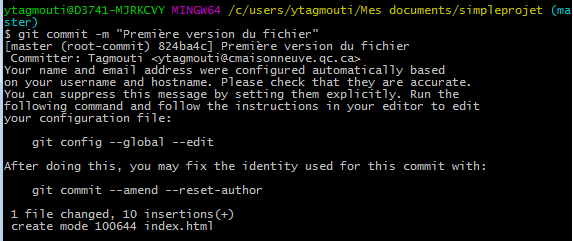
«git add» introduit une confusion. Ici, «Add» veut dire «ajoute les fichiers et répertoires suivants dans le staging c'est-à-dire la zone d’index» et non «ajoute le fichier dans la base donnée». Bien entendu, le fait d’ajouter dans le staging ajoutera le fichier à la base de données éventuellement, mais c’est une nuance importante à comprendre. (Car l’opération inverse de «git add» est «git rm»… git rm n’efface pas le fichier du répertoire de travail… seulement du staging.)

1. Que se passes-t-il maintenant si on fait un «git status»?



Git nous informe que notre fichier est bel et bien sur le staging!

1. Une fois notre *staging* prêt, on peut faire un commit! Avec la commande **git commit,** l’option –m permet d’ajouter un commentaire à notre commit

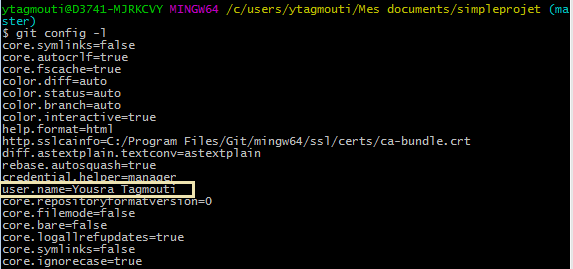


WoW! Git est capable de mettre par défaut deux paramètres importants de Git : le nom de la personne qui fait des modifications ainsi que son email. Si vous êtes sur le réseau du Cégep, cela se produira. Sinon, il faudra manuellement mettre ces informations.

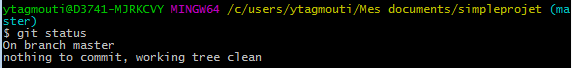
1. J’aimerais tout de même que mon nom apparaisse. Voici comment :



1. J’aime bien m’assurer que Git a bien pris mes changements en compte. Voici comment :

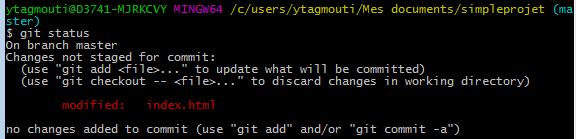


1. Et maintenant, vérifions avec un autre «git status» :



Qu’est-ce que cela signifie? En fait, Git nous dit que notre répertoire de travail est «clean» donc exempt de modifications! Modifions un peu les choses.

1. Dans le fichier index.html, ajoutez le nom de votre animal préféré dans la balise <p>.
2. Ensuite, faites un «**git status**» :

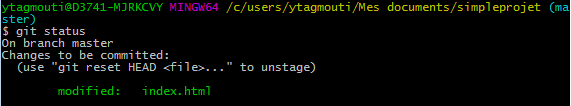


Maintenant, Git nous indique que notre fichier a été modifié. Nous devons le placer sur le staging. Quelle commande allons-nous utiliser? Notez aussi la nuance du «git add». Add veut dire «ajouter au staging».

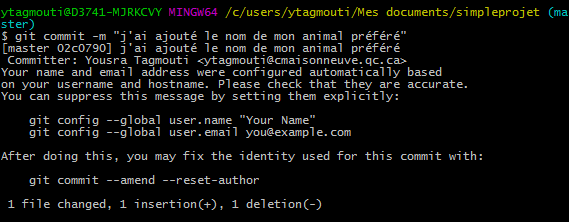
1. Faites **git add**!



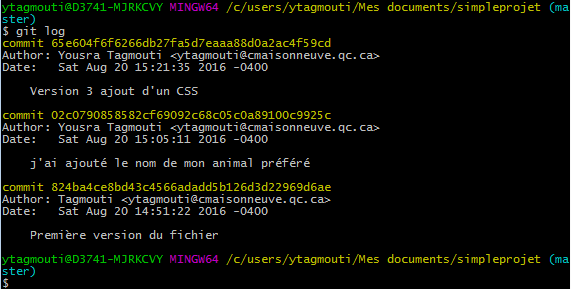
1. Et ensuite encore un **git status** :



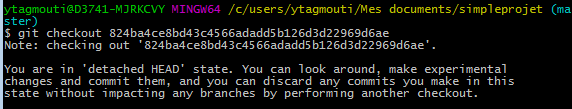
1. Voilà qui est bien. On peut maintenant ajouter les changements à la base de données.



1. Nous avons effectué deux versions! Il est maintenant le temps d’en faire une troisième. Ajoutez un dossier CSS dans lequel vous placerez une feuille de style. Votre page doit s’afficher sur fond bleu grâce à cette feuille de style.
2. Ensuite, ajoutez tout cela avec git. Cette fois-ci, je vous laisse aller!
3. Maintenant, voyons l’historique! Tapez la commande **git log**



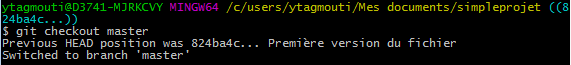
1. Vu que Git conserve un instantané des fichiers à chaque commit, on peut même revenir en arrière! Pour cela, on utilise «**git checkout** SHA1» où SHA1 représente la clé de hashage du commit. Voici un exemple :



1. De quoi est fait le contenu HTML du fichier index.html?

Répondez la partie 3 de la question #25

1. Revenons à la dernière version.



1. Répondez aux questions suivantes :
2. La différence entre «**dir**», «**ls**», «**ls –l**»
3. Où Git place-t-il les différentes versions? Sur un serveur? Ou ailleurs?

|  |
| --- |
|  |

1. Insérez ici un copier-coller (résultat de l’opération) de l’étape #20. Commentez brièvement vos actions.

|  |
| --- |
|  |

1. Insérer le contenu du fichier HTML de l’étape #23
2. Avec vos propres mots, Quelle est la différence entre le répertoire de travail, le staging (index), et le repo ?

|  |
| --- |
|  |

1. Avec vos propres mots, que fait «**git init**»?
2. Avec vos propres mots, que fait «**git add**»?

|  |
| --- |
|  |

1. Avec vos propres mots, que fait «**git commit**»?

|  |
| --- |
|  |