Cours Inf3523 – Architecture et Génie des Logiciels

Lab_1 : Bases de GIT

Git a rapidement évolué pour devenir l'outil de contrôle de version de code source de facto. C'est le deuxième enfant célèbre de Linus Torvalds, qui, après avoir créé le noyau Linux, a forgé ce logiciel de contrôle de version pour suivre ses millions de lignes de code.

Introduction

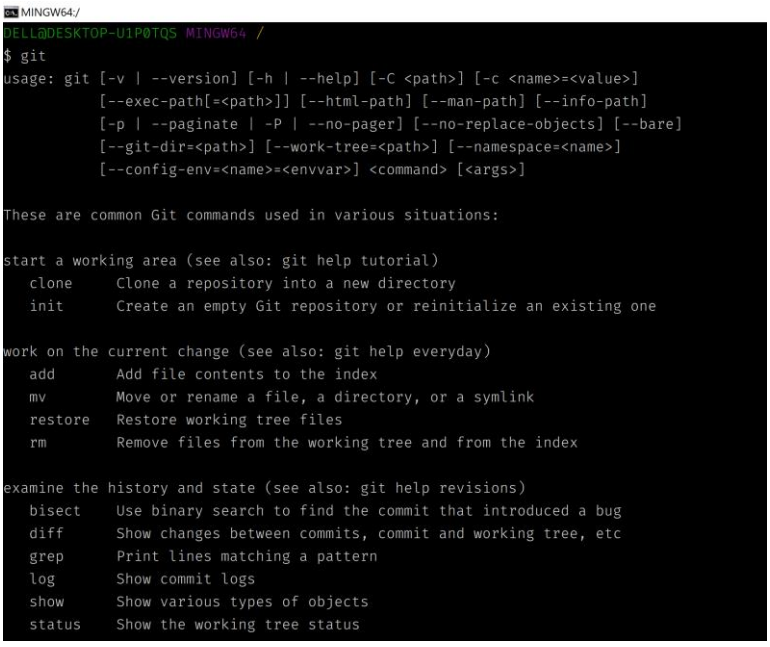
Que vous soyez un développeur professionnel ou amateur, vous avez probablement entendu parler du concept de contrôle de version. Vous savez peut-être qu'ajouter une nouvelle fonctionnalité, corriger une erreur ou revenir à une version précédente est une routine quotidienne.

Cela nécessite l'utilisation d'un outil puissant qui peut vous aider à prendre soin de votre travail, vous permettant de naviguer rapidement dans votre projet.

Il existe de nombreux outils pour cela sur le marché, propriétaires ou open source. Généralement, vous trouverez des systèmes de contrôle de version (VCS) et des systèmes de contrôle de version distribués (DVCS). Des exemples d'outils centralisés sont : Concurrent Version System (CVS), Subversion (SVN), etc. En DVCS, vous pouvez trouver Mercurial et Git, entre autres. La principale différence entre les deux familles est la contrainte - dans le système centralisé - d'avoir un serveur distant à partir duquel récupérer et dans lequel placer vos fichiers; inutile de dire que si le réseau est en panne, vous êtes en difficulté. En DVCS, en revanche, vous pouvez avoir ou ne pas avoir de serveur distant (même plus d'un), mais vous pouvez également travailler hors ligne. Toutes vos modifications sont enregistrées localement afin que vous puissiez les synchroniser à un autre moment. Aujourd'hui, Git est le DVCS qui a gagné plus de faveur publique que d'autres, passant rapidement d'un outil de niche à la norme.

Premier pas avec GIT

Si Git a été installé correctement, taper "**git**" sans spécifier autre chose donnera une courte page d'aide, avec une liste de commandes courantes (sinon, essayez de réinstaller Git).



Faire les présentations

Git a besoin de savoir qui vous êtes. C'est parce que dans Git, chaque modification que vous faites dans un dépôt doit être signée avec le nom et l'e-mail de l'auteur. Donc, avant de faire quoi que ce soit d'autre, nous devons dire à Git cette information. Tapez ces deux commandes (veillez à mettre votre nom et votre mail) :

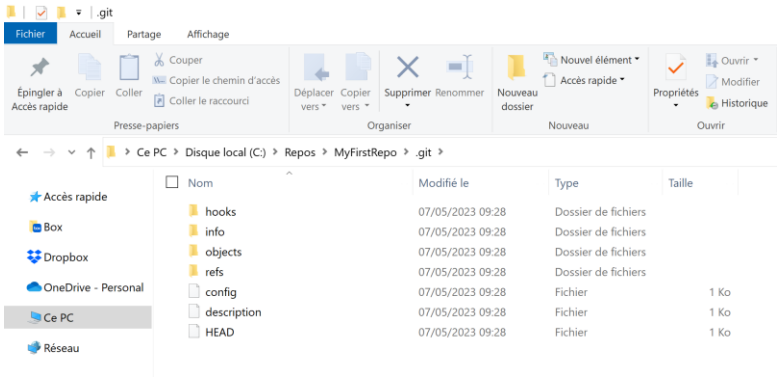
En utilisant la commande **git config**, nous avons configuré deux variables de configuration - **user.name** et **user.email**. À partir de maintenant, Git les utilisera pour signer vos commits dans tous vos dépôts.

Création d'un nouveau dépôt

La première étape consiste à configurer un nouveau dépôt. Un dépôt est un conteneur pour votre projet entier ; chaque fichier ou sous-dossier à l'intérieur appartient à ce dépôt, de manière cohérente. Physiquement, un dépôt n'est rien d'autre qu'un dossier qui contient un dossier spécial : **.git**.

Essayons de créer notre premier dépôt. Choisissez un dossier que vous aimez (par exemple, C:\Repos\MyFirstRepo), et tapez la commande **git init**, comme illustré ici:

Git a créé un sous-dossier : **.git**. Ce sous-dossier (normalement caché sous Windows) contient d'autres fichiers et dossiers, comme illustré dans la capture d'écran suivante:



À ce stade, il n'est pas important pour nous de comprendre ce qui se trouve à l'intérieur de ce dossier. La seule chose que vous devez savoir, c'est que vous ne devez jamais le toucher ! Si vous le supprimez ou si vous modifiez les fichiers à l'intérieur à la main, vous pourriez avoir des problèmes.

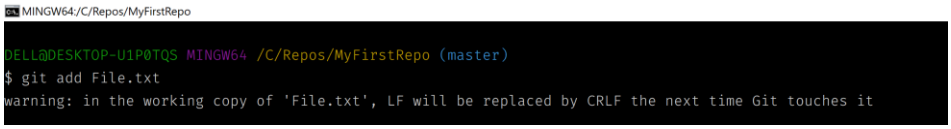
Maintenant que nous avons un dépôt, nous pouvons commencer à y mettre des fichiers. Git peut tracer l'historique de n'importe quel type de fichier, qu'il soit à base de texte ou binaire, petit ou grand, avec la même efficacité (plus ou moins ; les fichiers volumineux posent toujours un problème).

Ajout d'un fichier

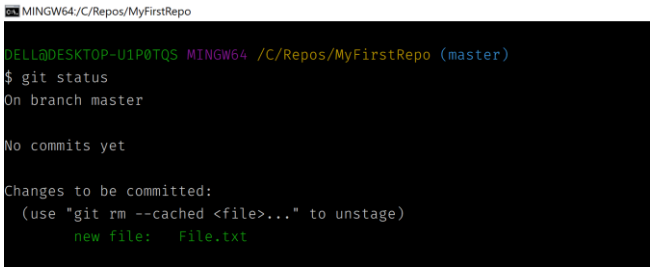
Créons un fichier texte, juste pour essayer :

Nous devons dire à Git de mettre explicitement ce fichier dans votre dépôt. ***Git ne fait rien que vous ne voulez pas qu'il fasse***. Si vous avez des fichiers de rechange ou temporaires dans votre dépôt, Git ne s'en occupera pas, mais vous rappellera simplement qu'il y a des fichiers dans votre dépôt qui ne sont pas sous contrôle de version (plus tard, nous verrons comment instruire Git pour les ignorer lorsque cela est nécessaire).

Je veux que file.txt soit sous le contrôle de Git, il me suffit de mettre la commande **git add** qui dit à Git que nous voulons qu'il prenne soin de ce fichier et qu'il le vérifie pour les modifications futures.



En utilisant la commande **git status**, nous pouvons vérifier l'état du dépôt, comme illustré dans cette capture d'écran :



Comme nous pouvons le voir, Git a accompli son travail comme prévu. Dans cette image, nous pouvons lire des mots tels que "**branch**", "**master**", "**commit**" et "**unstage**". Nous allons les étudier plus tard.

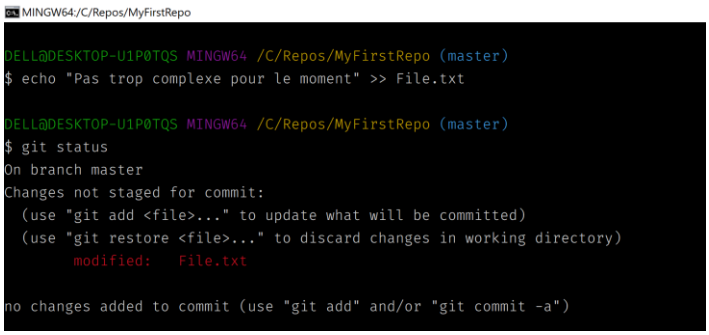
Commit du fichier ajouté

À ce stade, Git est au courant de l'existence de file.txt, mais nous devons effectuer une autre étape pour fixer l'instantané de son contenu. Nous devons le commit en utilisant la commande appropriée **git commit**. Cette fois-ci, nous ajouterons un peu de contenu à notre commande en utilisant le sous-commande **--message (ou -m)**, comme le montre l'exemple ci-dessous :

Avec la validation de file.txt, nous avons enfin lancé notre dépôt. Ayant effectué le premier commit (également appelé **root-commit**, comme vous pouvez le voir dans la capture d'écran), le dépôt a maintenant une branche principale avec un commit à l'intérieur. Nous parlerons des branches plus tard. Pour l'instant, prenez-le comme un chemin de notre dépôt, et gardez à l'esprit qu'un dépôt peut avoir plusieurs chemins qui se croisent souvent.

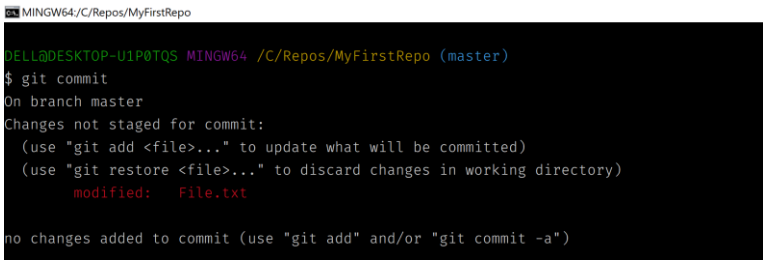
Modification du fichier commité

Maintenant, nous pouvons essayer d'apporter des modifications au fichier et voir comment y faire face, comme indiqué dans la capture d'écran suivante :



Comme vous pouvez le voir, le shell Bash nous avertit qu'il y a des modifications en peignant le nom des fichiers modifiés en rouge. Ici, la commande **git status** nous informe qu'il y a un fichier avec des modifications et que nous devons le commettre si nous voulons sauvegarder cette étape de modification dans l'historique du dépôt.

Cependant, que signifie "**no changes added to commit**"? En effet, Git vous fait prendre une seconde fois ce que vous voulez inclure dans le prochain commit. Si vous avez modifié deux fichiers mais que vous voulez commettre seulement un, vous pouvez n'ajouter que celui-là. Si vous essayez de commettre en sautant l'étape d'ajout, rien ne se passera (voir la capture d'écran suivante).



Alors, ajoutons à nouveau le fichier dans le but de préparer les choses pour le prochain commit:

Ensuite, faisons un autre commit cette fois-ci en omettant la sous-commande **--message**. Tapez **git commit** et appuyez sur la touche Entrée :

Qu'est-ce que c'est ? C'est **Vim (Vi IMproved)**, un éditeur de texte ancien et puissant, utilisé même aujourd'hui par des millions de personnes. Vous pouvez configurer Git pour utiliser votre propre éditeur préféré, mais si vous ne le faites pas, c'est avec ça que vous devez faire face. **Vim** est puissant, mais pour les nouveaux venus, cela peut être difficile à utiliser. Il a une manière étrange de traiter le texte. Pour commencer à taper, vous devez appuyer sur **i** pour insérer du texte, comme indiqué dans la capture d'écran suivante :

Une fois que vous avez tapé votre message de commit, vous pouvez appuyer sur Échap pour sortir du mode édition. Ensuite, vous pouvez taper la commande **:w** pour enregistrer les modifications et la commande **:q** pour quitter. Vous pouvez également taper les commandes par paires comme **:wq**, comme nous le faisons dans cette capture d'écran, ou utiliser la commande équivalente **:x** :

Après cela, appuyez sur Entrée et un autre commit est alors effectué :

Résumé

Une fois que vous avez choisi un répertoire à inclure dans un dépôt Git, vous pouvez voir que l'initialisation d'un nouveau dépôt Git est aussi simple que d'exécuter une commande **git init**. Pour l'instant, ne vous inquiétez pas de le sauvegarder sur un serveur distant, etc. Ce n'est pas obligatoire de le faire; vous pouvez le faire quand vous en avez besoin, en préservant l'intégralité de l'historique de votre dépôt. C'est une fonctionnalité essentielle de Git et des DVCS en général. Vous pouvez travailler confortablement hors ligne et pousser votre travail vers un emplacement distant lorsque le réseau est disponible, sans aucun tracas.

Enfin, nous avons découvert l'une des caractéristiques les plus importantes de Git : il ne fera rien si vous ne le mentionnez pas explicitement. Vous avez également appris un peu sur la commande add. Nous avons été obligés d'exécuter une commande **git add** pour un fichier lorsque nous l'avons commité pour la toute première fois. Ensuite, nous avons utilisé une autre commande lorsque nous l'avons modifié. Cela s'explique par le fait que si vous modifiez un fichier, Git ne s'attend pas à ce que vous vouliez qu'il soit automatiquement ajouté au prochain commit. Dans le prochain lab, nous examinerons certains fondamentaux de Git.