

Manipulation de GIT.

version 2

Complément de Programmation

1 Les rappels

Git est un logiciel de gestion de version décentralisé. Il permet d'assurer des communications entre les différents développeurs en passant par un serveur ou directement (lors de phase de maintenance en particulier).

Dans le cadre de ce TP, nous allons voir le fonctionnement de base du logiciel TortoiseGit qui fournit une interface intégrée à l'explorateur Windows (le logiciel de navigation du système de fichier par défaut). Si vous utilisez un autre système d'exploitation je vous laisse adapter l'énoncé à votre logiciel. Comme vu en cours, il existe beaucoup de logiciel permettant de se passer de la console Git (initialement prévue) au profit d'une interface avec un contrôle souris (GitKraken, Visual Studio Git, EclipseGit ...).

Pour accéder à l'interface TortoiseGit, il suffit de faire un clique-droit sur un répertoire/fichier depuis l'explorateur Windows. Vous avez une section dédiée aux commandes Git. L'ensemble des commandes disponibles dépendent du contexte d'utilisation. En effet, si TortoiseGit ne reconnaît pas le dépôt, il vous propose d'en créer un, si le fichier n'est pas connu dans le dépôt il vous propose de l'ajouter, ou enfin si le fichier est modifié il vous propose de le sauvegarder.

Enfin, TortoiseGit possède plusieurs façons pour gérer un dépôt, via le menu contextuel, via de grandes fenêtres de contrôle. ...L'objectif est de privilégier la méthode des menus contextuels (clique-droit sur l'élément) pour rester très proche de l'utilisation classique du système de fichier Windows.

2 Préparation de notre dépôt

Lors de cette séance, nous allons travailler de manière locale à un ordinateur : votre dépôt Git et vos répertoires de travaux seront présent localement à l'ordinateur. Ainsi, vous pourrez travailler seul pour mieux comprendre les commandes de bases à votre rythme.

La première étape est de réaliser le dépôt Git de votre projet, pour cela veuillez suivre les étapes suivantes :

1. Allez dans votre espace personnel : `Z:\>`.
2. Créez un répertoire nommé : `CompProg`.
3. Double-cliquez sur ce répertoire pour "y entrer/aller".
4. Créez un répertoire nommé : `Depot.git`. Attention, ne mettez pas d'accent ou d'espace et ajoutez bien le suffixe qui indique que le répertoire sera le dépôt Git (c'est une convention).
5. Faites un clique-droit sur le répertoire de dépôt et choisissez "Git create repository here...".
6. Une fenêtre de confirmation apparaît pour vous confirmer le "Make it Bare"¹. Assurez-vous que la case est cochée et cliquez sur OK.
7. Normalement, une fenêtre d'informatique indique la création d'un dépôt vide. Cliquez sur OK pour la fermer.

Votre dépôt Git est, à présent, correctement créé et vous n'avez plus à y toucher. Si votre curiosité est trop forte, vous pouvez l'ouvrir et voir qu'il y a pleins de répertoires et de fichiers liés à la gestion Git, en particulier les copies des fichiers et des informations sur le dépôt.

3 Création de nos répertoires de travail

A présent, que le dépôt est créé nous allons créer le répertoire de travail. C'est-à-dire, le répertoire contenant notre copie que vous modifierez avant l'envoi vers les autres développeurs.

Dans le jargon Git, un répertoire de travail s'obtient en clonant le dépôt. Pour cela, réalisez les étapes suivantes :

1. Retournez dans le répertoire `Z:\CompProg`
2. Créez un répertoire nommé : `TravailA` qui sera votre première copie de travail.
3. Faites un clique droit sur le répertoire de travail et choisissez l'option : `Git clone...`
4. Dans la nouvelle fenêtre :

1. Ce mode interdit la création d'un répertoire de travail dans le répertoire de dépôt. Il est donc inutile de travailler directement dessus.

- (a) Dans le champ URL mettez l'adresse du dépôt : `Z:\CompProg\Depot.git`
- (b) Dans le champ Directory, modifiez `Z:\CompProg\TravailA\Depot` en `Z:\CompProg\TravailA`
- (c) Si vous oubliez l'étape précédente, vous aurez un répertoire inutile et il faudra adapter vos chemins dans la suite².
- (d) Vérifiez que tout est décoché, sauf l'option **Recursive**.
- (e) Cliquez sur **OK**.

Vous venez de préparer votre répertoire de travail contenant la copie de votre dépôt. Si vous lisez en détail la fenêtre d'information (fenêtre de log) vous verrez un message indiquant que le dépôt est vide ce qui peut limiter certains traitements.

4 Création d'une première ressource

A présent, nous nous plaçons dans le répertoire de travail et nous pouvons commencer à le modifier. Généralement, le premier fichier d'un dépôt est un fichier `Readme.txt` contenant les informations sur le projet en cours. En utilisant votre éditeur préféré, créez un fichier texte contenant le descriptif suivant : "*Premier projet sur la manipulation Git*".

Maintenant que notre fichier est prêt pour l'envoi sur le dépôt, vous devez réaliser les étapes suivantes :

1. Sélectionner le fichier `Readme.txt` (normalement par un simple clic gauche dessus).
2. Cliquez droit sur votre sélection.
3. Choisissez TortoiseGit et dans le sous-menu cliquez sur **Add**.
4. Dans la nouvelle fenêtre, vous pouvez soit la fermer (en cliquant sur **OK**) pour continuer à faire d'autre modification. Soit cliquer sur le bouton **commit** pour envoyer dès à présent vos modifications sur le dépôt. Nous décidons de valider nos modifications donc cliquez sur **commit**.
5. Si vous n'avez jamais configuré TortoiseGit, une fenêtre d'erreur va apparaître pour que vous fournissiez deux informations importantes pour la communication sur le serveur. En effet, cela permettra de garder une trace de l'auteur ayant fait la modification. Cliquez sur "Oui" et réalisez les traitements suivants dans la nouvelle fenêtre :
 - (a) Indiquez un nom d'utilisateur (ce nom sera affiché lors des demandes d'historique d'une ressource).
 - (b) Indiquez une adresse électronique (utilisez pour communiquer avec l'auteur de la modification) : indiquez votre adresse `univ-poitiers.fr`.
 - (c) Confirmez votre modification.
6. Une nouvelle fenêtre s'ouvre : la fenêtre de **commit**. Elle répertorie dans la partie inférieure l'ensemble des fichiers du dépôt et leur statut. Normalement, il doit y figurer le fichier `Readme.txt` dont le statut est **Added**.
7. La partie supérieure regroupe une zone de texte, demandant au développeur de saisir un message clair. Faites le.
8. Une fois le message indiqué, le bouton en bas **Commit & Push** devient utilisable (utilisez la flèche du bas sur le bouton pour choisir cette option si cette option n'est pas proposée par défaut). Cliquez sur ce bouton.
9. Une nouvelle fenêtre apparaît : la fenêtre de progression. Elle indique l'avancement de votre opération (ici l'envoi d'une ressource).
10. Une fois l'opération terminée et lu les informations fournies sur le bon fonctionnement de l'envoi, vous pouvez fermer la fenêtre.

Votre fichier ressource est correctement envoyé sur le dépôt et les autres développeurs pourront le récupérer pour pouvoir en profiter.

En vous inspirant de votre première expérience, allez chercher une image sur internet ou de votre répertoire et ajoutez la à votre dépôt.

5 Se mettre dans la peau d'un autre développeur

Avant d'attaquer les aspects dans la manipulation des dépôts Git, nous allons commencer à créer un second répertoire de travail.

Pour cela, en vous inspirant de la section précédente : cloner votre dépôt dans le répertoire `Z:\CompProg\TravailB`. Il s'agit d'un second répertoire de travail. Dans une utilisation normale, on peut considérer ce répertoire comme celui d'un autre développeur ou d'une seconde machine de travail (par exemple, pour expérimenter le logiciel sur une configuration très spécifique).

2. Vous remarquerez que le nom du dépôt est automatique renommer pour retirer le `.git` et obtenir le nom du projet en cours.

Lorsque vous aurez réussi votre clonage, vous remarquerez que le répertoire `TravailB` est une copie de `TravailA` avec les mêmes fichiers : votre fichier texte `Readme.txt` et votre image.

Maintenant toujours dans votre répertoire `TravailB`, éditez votre fichier texte et remplacez le point final par la phrase : “avec `TortoiseGit`.”.

Une fois la modification réalisée, sélectionnez le fichier `Readme.txt`, faites un clic droit dessus et choisissez `Git commit` — > `master`. Vous remarquerez que le menu contextuel s’adapte avec votre sélection. Indiquez un message de suivi et validez le `Commit & Push`.

A présent, retournez dans le répertoire `TravailA`. Nous allons rapatrier la modification depuis le serveur, procédez comme suit :

1. Retournez dans le répertoire `Z:\CompProg`.
2. Sélectionnez le répertoire `TravailA`.
3. Faites un clic droit et dans le sous-menu de `TortoiseGit`, choisissez la commande `Pull`.
4. Dans la nouvelle fenêtre vous n’avez rien à faire (pour l’instant). Cliquez sur `OK`.
5. Dans la fenêtre résumant la mise-à-jour, vous verrez un ensemble d’information synthétisant les fichiers modifiés et leur modification.
6. Fermez la nouvelle fenêtre.
7. Vous pouvez vérifier que la modification s’est bien correctement appliquée sur le fichier texte.

Pratiquez les mécanismes d’ajout de ressources, d’envoi sur le serveur et de récupération des ressources, en faisant bien toutes les étapes du début à la fin :

- Dans le répertoire `TravailA` : ajoutez quelques lignes décrivant votre première expérience avec `Git`.
- Envoyez les modifications sur le dépôt. Et mettez à jour le répertoire `TravailB`.
- Dans le répertoire `TravailB` : modifiez l’image (modifiez le fond ou ajoutez des traits de couleurs).
- Envoyez les modifications sur le dépôt. Et mettez à jour le répertoire `TravailA`.
- Dans le répertoire `TravailA` : réalisez plusieurs ajouts de lignes dans le fichier `Readme.txt`, en faisant plusieurs `commit` (inutile de mettre à jour `TravailB` entre-temps). Réalisez au moins 5 `commits`.
- Mettez-à-jour le répertoire `TravailB` d’un seul coup. Normalement vous ne devriez avoir aucun souci et récupérer directement la dernière version.

La mécanique que nous avons fait montre que lorsqu’une équipe de développeurs travaillent à tour de rôle, il n’y a aucun souci dans l’échange de fichier.

Pour résumer, les commandes vues jusqu’à présent sont :

- La création d’un dépôt (équivalent : `git init -bare`)
- L’ajout d’une ressource dans le gestionnaire (équivalent : `git add <file>`)
- L’envoi effectif vers le dépôt : (équivalent : `git commit` et `git push`)
- La récupération de la dernière version depuis le dépôt : (équivalent : `git pull`)

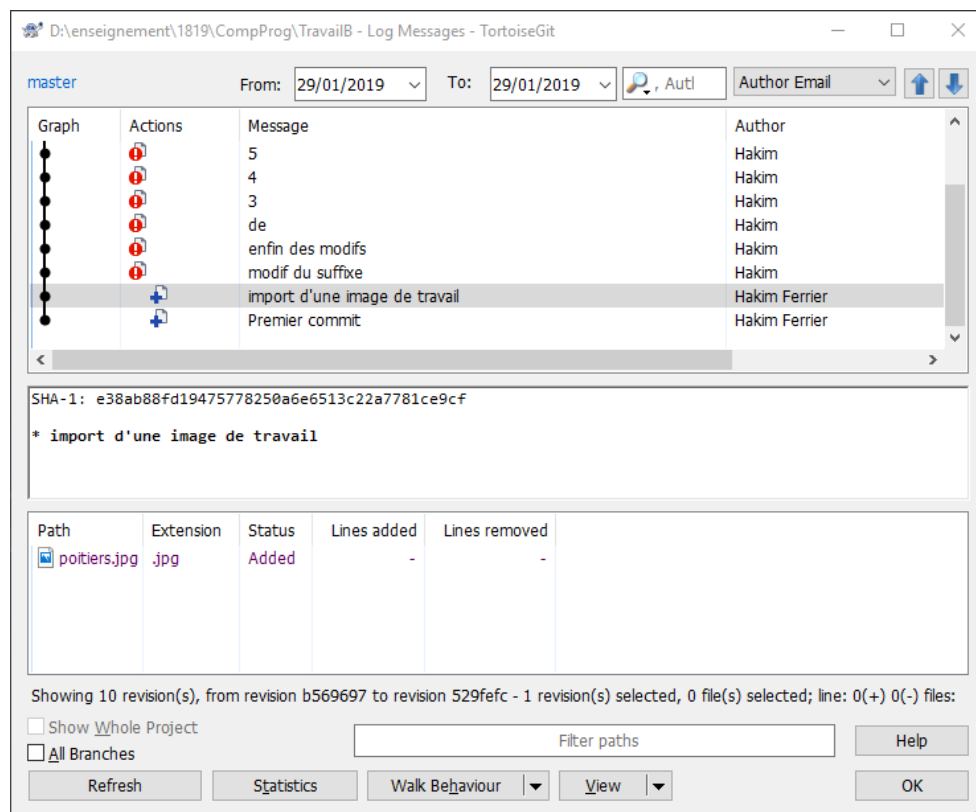
6 Revenir dans l’historique des versions

Jusqu’à présent, nous avons enregistré pleins d’informations sur le dépôt. L’un des avantages est de pouvoir visualiser l’historique des modifications de nos fichiers. C’est dans cette visualisation que les outils graphiques tirent leur avantage, car là, les modes consoles prennent énormément de temps pour retrouver toutes les informations et plusieurs commandes peuvent se succéder avant de réussir à faire ce que l’on souhaite.

Pour cela il suffit d’aller dans les commandes `TortoiseGit` et choisir la commande `Show log`. Cette commande s’adapte en fonction de la ressource sélectionnée : si le répertoire principal est choisi alors l’historique complet du dépôt est affiché, si un fichier est choisi alors uniquement l’historique en rapport avec la ressource est fourni. Vous devez obtenir une fenêtre séparée en quatre grandes parties comme sur l’image ci-dessous :

1. La partie supérieure indique l’historique générale des modifications subies par le début entre deux dates, sous la forme d’une version par ligne. Vous pouvez cibler précisément l’historique d’un auteur ou une période particulière pour filtrer la quantité d’informations fournies.
2. La partie centrale redonne le message associé au `commit` pour la version sélectionnée.
3. La troisième partie donne les détails de la version sélectionnée et le statut du fichier au moment du `commit`.

4. La quatrième partie est l'ensemble des boutons pour lancer des actions plus avancées.
5. Vous pouvez fermer toutes les fenêtres pour passer à un exemple d'utilisation.



A présent, on se propose de récupérer une ancienne version de nos fichiers. Pour cela, on vous propose de retrouver la première version de votre image (sans les modifications qui ont été apportées) :

1. Allez dans le répertoire TravailA (histoire de bien voir que l'historique ne dépend plus du répertoire ayant subi la modification).
2. Sélectionnez l'image et faites un clic droit dessus et dans le sous menu de TortoiseGit, cliquez sur **Show log**
3. La fenêtre indique l'historique du fichier image.
4. Dans la première zone, choisissez la version la plus ancienne.
5. Dans la troisième zone, sélectionnez l'image et faites un clic droit dessus.
6. Dans le sous-menu choisissez **Compare with working tree**. Une nouvelle fenêtre montre deux images la version choisie et celle du répertoire de travail.
7. Fermez la fenêtre de comparaison.
8. Dans le sous-menu choisissez **Revert to this version** pour revenir à la version demandée. Fermez la petite fenêtre de confirmation.
9. Attention! Vous venez de modifier votre répertoire de travail avec cette nouvelle version (même si c'est une copie d'une ancienne version).
10. Mettez-à-jour le dépôt en faisant un **commit-push**.
11. Mettez-à-jour le répertoire TravailB pour bien visualiser la nouvelle version de l'image.

A présent à vous de jouer : consulter différentes versions du fichier texte sans forcément modifier à chaque fois le dépôt (en bref, vos modifications se feront sur le répertoire TravailB et vous ne ferez aucun commit).

Normalement, vous ne savez plus quelle version est en cours dans vos fichiers. Vous avez donc plusieurs solutions :

- Avec vos connaissances, vous pouvez retourner dans l'historique et choisir la dernière version présente dans le dépôt.
- Une autre possibilité est d'abandonner toutes les modifications de répertoire TravailB.

Nous allons procéder avec la seconde méthode en abandonnant toutes les modifications (on dit qu'on veut faire un reset) avec les étapes suivantes :

1. Fermez toutes les fenêtres TortoiseGit qui sont ouvertes.
2. Rendez-vous dans le répertoire `Z:\CompProg`
3. Sélectionnez le répertoire TravailB.
4. Cliquez avec le bouton droit sur la sélection et dans le sous-menu TortoiseGit, choisissez **Revert...**
5. Dans la nouvelle fenêtre vous voyez l'ensemble des modifications en attente.
6. Cochez l'ensemble des ressources.
7. Cliquez sur le bouton OK.
8. Une nouvelle fenêtre vous indique la progression des modifications (normalement tout doit bien se passer).
9. Fermez la fenêtre lorsque le traitement s'est terminé.

Si vous avez bien respecté toutes les étapes les deux répertoires TravailA et TravailB doivent contenir les mêmes fichiers (et les mêmes versions).

Pour conclure cette section, nous synthétisons les mécanismes Git vus ici :

- Fenêtre pour consulter l'historique du dépôt ou d'un fichier.
- La navigation dans l'historique d'une ressource.
- La comparaison sans modification du répertoire de travail d'une ressource.
- La récupération d'une ancienne version et l'utiliser dans le répertoire de travail.
- La récupération implique une modification dans le répertoire de travail et nécessite l'envoi vers le dépôt de cette nouvelle version.
- Le mécanisme d'abandon des modifications d'un répertoire de travail.

7 Gestion des conflits

Jusqu'à présent, nous supposons que les développeurs travaillent à tour de rôle, ce qui est assez rare surtout dans les grandes équipes. Dans la réalité, les développeurs travaillent sur des parties différentes pour éviter des conflits. Cependant, lors d'étape de maintenance ou de petites évolutions, il se peut que deux développeurs fassent une modification sur un même fichier. Il se produit alors un conflit. Il faut distinguer deux origines d'un conflit :

- Vous avez fait des modifications sur votre travail et vous souhaitez les envoyer sur le dépôt. Cependant, ce dernier a subi d'autre modification. Il faut donc que vous récupériez la nouvelle version et que vous gériez le conflit avant de pouvoir envoyer vos modifications.
- Vous tentez de récupérer la dernière version depuis le dépôt, mais vous aviez des modifications en suspens que vous ne vouliez pas envoyer pour l'instant (par exemple, le code n'étant toujours pas fini).

Si vous regardez bien, les deux causes se traitent de la même façon : on règle les conflits chez nous avant d'envoyer une version "propre" vers le dépôt.

Attention

Jusqu'à présent, lorsque nous envoyons sur le dépôt, nos ressources nous faisons toujours un "Commit & Push". En réalité, nous réalisons deux opérations distinctes :

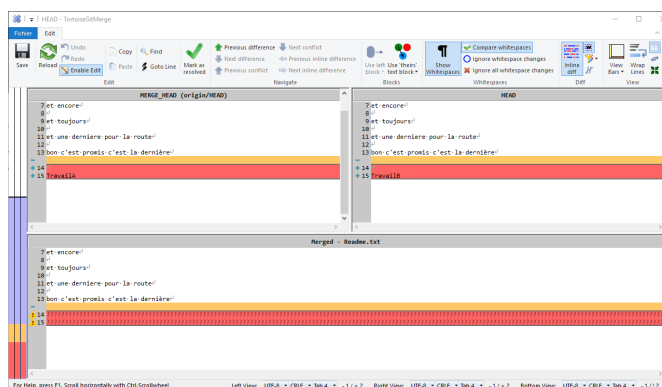
- **commit** : le commit envoie les fichiers du répertoire de travail vers votre copie du dépôt (contenu dans un répertoire caché nommé `.git`).
- **push** : le push consiste à synchroniser le dépôt distant (celui que nous avons appelé dépôt depuis le début) et le dépôt local (le fameux répertoire caché).

Dans le cadre de ce cours, nous avons décidé de fonctionner en faisant le commit suivi du push, pour simplifier votre compréhension et votre expérience. Cependant, lors des conflits vous aurez parfois le commit qui fonctionnera et pas le push il faudra donc peut-être synchroniser les dépôts avant de pouvoir gérer un conflit.

A présent, nous allons provoquer un conflit dans le fichier `Readme.txt`, procédez comme suit :

1. Ajoutez une nouvelle ligne dans le fichier `Readme.txt` qui indique le répertoire de travail dont il provient.
2. Envoyer votre modification du répertoire TravailA, comme nous avons vu précédemment. Il n'y a aucun conflit à ce moment.
3. Maintenant dans votre répertoire TravailB, tentez d'envoyer vos modifications.

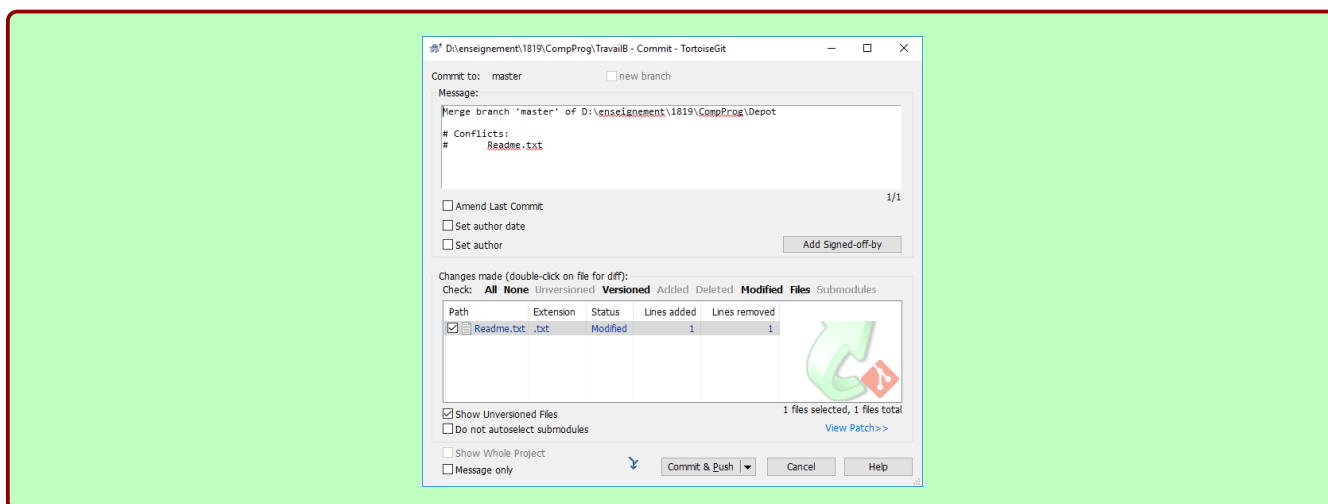
4. Vous obtenez un message d'erreur vous expliquant qu'un autre développeur a fait un push avec succès avant vous et donc que votre répertoire n'est plus à jour.
5. Vous devez rapatrier les nouvelles modifications en faisant un pull. Normalement TortoiseGit vous propose dans la partie inférieure un raccourci pour lancer le pull immédiatement.
6. Lors de la mise-à-jour, vous verrez apparaître des messages de conflits sur le fichier Readme.txt.
7. **La philosophie de Git est de traiter les conflits le plus tôt possible.** En bas de la fenêtre de progression, TortoiseGit vous propose de résoudre le conflit immédiatement, mais ce bouton est ambigu. En effet, si vous cliquez dessus vous allez commiter (conserver) le fichier résultant de la fusion de l'ancienne et la nouvelle version avec l'ensemble des différences.
8. Il faut donc fermer la fenêtre pour gérer correctement le conflit.
9. Une petite fenêtre apparaît lors de la fermeture pour afficher les différences. Il faut cliquer sur oui. Si vous ratez cette fenêtre, ne paniquez pas ! Il suffit d'aller dans l'interface "Show log" du dépôt. Dans la partie supérieure, il faut alors choisir le répertoire de travail. Alors la troisième fenêtre (celle des détails) affiche les fichiers en conflit. Il suffit alors sélectionner un fichier en conflit (ici Readme.txt) et faites un clic droit dessus et choisir "Edit conflict".
10. La fenêtre d'édition des conflits se divise en trois parties :
 - La zone MERGE_HEAD qui correspond à la version du fichier sur le dépôt distant (par défaut à gauche).
 - La zone HEAD qui correspond à la version présente sur le répertoire de travail (par défaut à droite).
 - La zone Merged, qui correspond à la version finale où la résolution du conflit est en cours.



11. Les zones rouges indiquent les zones en conflits que vous devez résoudre : pour cela vous avez plusieurs façons de faire :
 - (a) Vous pouvez utiliser la souris dans les lignes rouges et faire un clic droit dessus pour choisir quelle version vous souhaitez copier.
 - (b) Vous pouvez taper directement du texte en plus dans la version Merged pour résoudre le conflit.
12. Testez les deux grands mécanismes pour résoudre les conflits, les zones rouges deviennent vertes au fur et à mesure des résolutions.
13. Lorsque vous avez fini, cliquez sur le bouton de sauvegarde.
14. TortoiseGit vous propose de marquer le fichier comme résolu. Cliquez dessus.
15. Si vous retourner voir la fenêtre "Show log" vous verrez l'aspect conflit a disparu pour reprendre le statut modifié.
16. Envoyez vos modifications sur le dépôt. Dans la fenêtre de commit vous verrez un changement.

Attention

En Git, la résolution des conflits engendrent une branche automatiquement pour indiquer la fusion de ressources. Ainsi, le message d'information est lui rédigé de manière automatique par Git lui-même. Vous pouvez le modifier mais ce n'est pas nécessaire dans la situation des conflits.



Félicitations, vous venez de gérer un conflit avec Git. Dans une situation réelle vous risquez d'avoir plusieurs fichiers en conflits qu'il faudra gérer manuellement. Il est donc important d'adopter des bonnes pratiques pour les détecter le plus tôt possible dans le développement d'une application. Ainsi il est conseillé de :

- Lors de votre arrivée, il faut mettre à jour votre répertoire de travail (malgré que vous soyez sûr qu'il n'y ait aucune modification entre temps).
- En fin de séance de travail, il faut impérativement envoyer tout son travail sur le dépôt et ce pour tous les développeurs. Ainsi, le dernier à envoyer ses ressources doit résoudre les conflits.

Pour conclure cette section, nous synthétisons les mécanismes Git vus ici :

- Les développeurs doivent dans la mesure du possible travailler sur des fichiers différents.
- La gestion des conflits peut désynchroniser le commit et le push.
- Les conflits doivent être traités le plus tôt possible.
- L'interface "Show log" permet la gestion des conflits simplement.
- La résolution d'un conflit engendre automatiquement la création d'une branche et d'un message automatique au prochain commit.

8 Mécanismes avancés

8.1 La suppression de fichiers

La suppression d'un fichier dans le répertoire de travail risque de vous engendrer beaucoup d'erreur (de type "missing..."). En effet, malgré l'intégration de TortoiseGit dans l'explorateur Windows la suppression standard ne peut pas être intercepter par Git. Si cela se produit il vous suffit d'annuler vos modifications avec la commande "Revert". Pour supprimer un fichier, il faut donc passer par la commande "Delete" dans le menu TortoiseGit et de commiter votre modification pour la propager aux autres développeurs.

8.2 Renommage de fichiers

Le renommage suit les mêmes problèmes que la suppression d'un fichier. La solution est aussi similaire, il suffit d'utiliser la commande "Rename" dans le menu TortoiseGit. Ainsi, le renommage conserve l'historique du fichier.

8.3 Ignoré des fichiers

Certains fichiers peuvent ne pas avoir d'intérêt pour un projet, mais on souhaiterait quand même le conserver dans notre répertoire de travail (par exemple, un mémo personnel sur des fonctionnalités qui nous incombent). Dans ce cas, Git offre un mécanisme pour ignorer des fichiers. Nous vous proposons d'expérimenter ce mécanisme sur un exemple dans votre répertoire de travail **TravailB** :

1. Créez un fichier Word (ou libreWriter), que nous noterons **madoc.doc** dans la suite.

2. Faites un clique droit dessus.
3. Dans le sous-menu de TortoiseGit, vous avez l'option "Add to ignore list".
4. Dans le sous-menu précédent, choisissez "madoc.doc".
5. Une nouvelle fenêtre apparaît, lisez bien les options (les choix par défaut nous conviennent pour cette première expérimentation).
6. Cliquez sur OK.
7. Un nouveau fichier `.gitignore` est apparu. Le nom de fichier est conventionné en Git.
8. A présent, tentez de faire un commit sur votre dépôt.
9. Vous devez visualiser la présence du fichier `.gitignore` uniquement. Ainsi, on voit bien que le fichier `madoc.doc` n'est pas dans la liste des fichiers à ajouter.
10. Si vous ajoutez le fichier `.gitignore` au dépôt, tous les autres développeurs pourront ignorer les mêmes fichiers que vous.

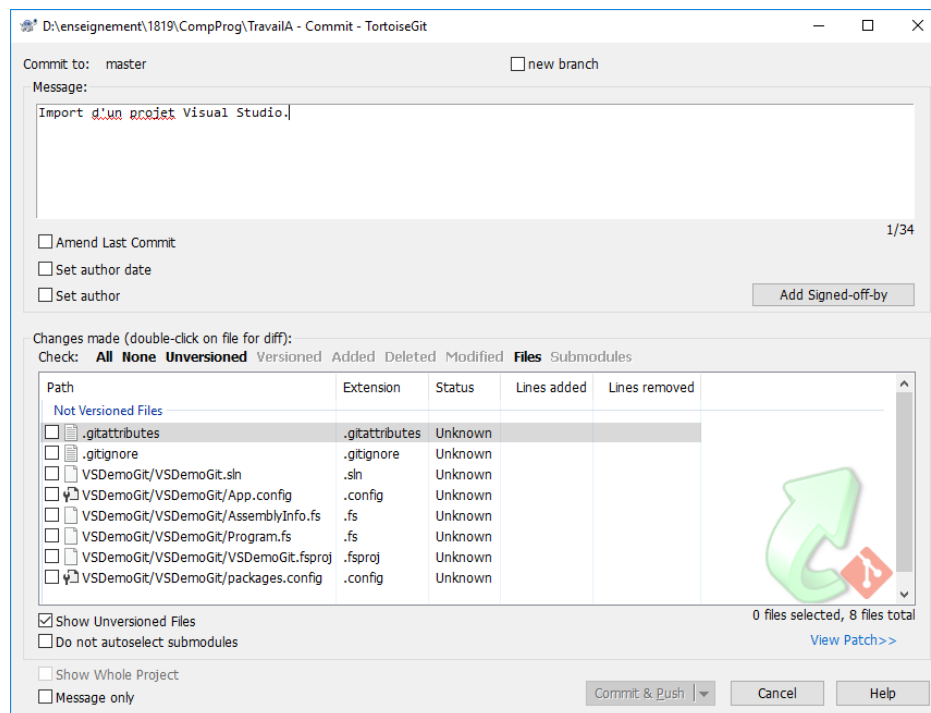
Pour conclure, le fait d'ignorer des fichiers permet d'assurer un certain rangement sur le dépôt. Il est particulièrement utile pour éviter d'ajouter aux dépôts des notes personnelles ou alors des fichiers, qui sont générés à partir d'autre fichier (par exemple, un exécutable est produit à partir des fichiers sources).

9 Ajout d'une solution Visual Studio dans un dépôt

Avant de créer votre projet, vous devez nettoyer votre répertoire de travail. En effet, cette action se fait très tôt en raison de certaines configuration. Pour cela, il faut que vous supprimiez complètement (du dépôt et des répertoires de travail) le fichier `.gitignore` que nous avons créé précédemment.

A présent, vous allez créer un nouveau projet dans le répertoire **TravailA**. Pour cela suivez les étapes suivantes :

1. Créez un nouveau projet console .NET standard, nommé **VSDemoGit** comme vu en cours.
2. Mentionnez bien le répertoire **TravailA**, comme emplacement de la solution.
3. (Pour rappel, il ne faut pas cliquer sur la case à cocher "Créer un dépôt Git" dans l'interface Visual Studio.)
4. Normalement, Visual Studio détecte que vous êtes dans un répertoire de travail et ajoute automatique plusieurs fichiers, en plus des fichiers du projet (attention certains sont des fichiers cachés).



5. Si vous avez d'autres fichiers, votre Visual Studio est sans doute mal calibré et il faut ajouter (avant de faire un commit), les deux fichiers présents sur le updago `.gitignore` et `.gitattributes` pour éviter l'ajout de fichiers inutiles. Fermez et relancez le commit.

6. Cochez tous les fichiers non-versionnés et envoyez le tout sur votre dépôt.
7. C'est tout !
8. Si vous avez ajouté les fichiers depuis updago. Il faudra que vous fassiez attention aux fichiers ajoutés. En effet, on évite de mettre des ressources étant le fruit d'une génération (par exemple, les exécutables, les bibliothèques ...). Il faudra donc que vous gériez ça, à la main.

La démocratisation de Git, a permis une très bonne intégration des éditeurs avancés. Ainsi, le développeur se focalise directement sur son projet au lieu de perdre du temps à démultiplier des tâches de maintenance sans forcément de lien avec sa productivité.

A présent, profitez de la fin de la séance pour manipuler Visual Studio et Git afin de maîtriser leurs manipulations de base.