

Utilisation de Git avec Visual Studio 2015

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Commentaire** | **Auteur** | **Société** |
| 0.1 | 23/06/2017 | Création du document. | Vincent JACQUOT | EID |
| 0.2 | 08/09/2017 | Changement du périmètre du document. | Vincent JACQUOT | EID |
|  |  |  |  |  |

Table des matières

[1 Introduction 3](#_Toc492645368)

[*2* La fenêtre *Team Explorer* 3](#_Toc492645369)

[3 Sélectionner la branche de travail 3](#_Toc492645370)

[4 Consulter l’historique des modifications 4](#_Toc492645371)

[4.1 Consulter l’historique d’une branche 5](#_Toc492645372)

[4.2 Consulter l’historique d’un fichier 3](#_Toc492645373)

[5 Archiver des modifications 4](#_Toc492645374)

[5.1 Archiver toutes les modifications 4](#_Toc492645375)

[5.2 Archiver une partie des modifications 5](#_Toc492645376)

[5.3 Annuler l’archivage de modifications 6](#_Toc492645377)

[6 Récupérer les dernières modifications 7](#_Toc492645378)

[6.1 Récupérer les modifications avec une opération pull 7](#_Toc492645379)

[6.2 Récupérer les modifications avec une opération rebase 8](#_Toc492645380)

[7 Publier des modifications 9](#_Toc492645381)

[8 Annuler des modifications 10](#_Toc492645382)

[8.1 Annuler des modifications non archivées 10](#_Toc492645383)

[8.2 Annuler des modifications archivées sur un dépôt local 11](#_Toc492645384)

[8.3 Annuler des modifications archivées sur un dépôt distant 12](#_Toc492645385)

# Introduction

Ce guide d’utilisation explique comment réaliser certaines opérations courantes de Git avec Visual Studio 2015. Il n’a pas vocation à être exhaustif ni à expliquer en détail les concepts de Git. Si un point particulier n’est pas couvert par ce guide, merci de vous référer à la documentation officielle de Microsoft[[1]](#footnote-1) ou de Git[[2]](#footnote-2).

# La fenêtre *Team Explorer*

Dans Visual Studio 2015, la gestion du contrôleur de source se fait principalement par la fenêtre *Team Explorer*. La fenêtre *Team Explorer* comporte plusieurs sections qui fonctionnent comme des onglets et qui sont accessible par le menu déroulant en haut de la fenêtre.

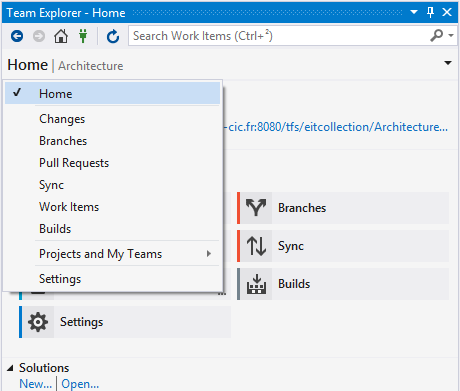


Figure - La liste des sections est accessible par le menu déroulant

# Sélectionner la branche de travail

Avant de commencer à coder, il est important de s’assurer que l’on travaille sur la branche adéquate. La branche sélectionnée par défaut au premier chargement du dépôt est la branche *master*, mais il peut être nécessaire d’en changer. Si vous travaillez sur la mauvaise branche, vous ne serez pas en mesure de publier vos commits.

En fonction du type de code et du flux de travail choisi, la branche de travail peut être la branche *master*, une branche de développement, de *hotfix* ou de *feature*. Les développements peuvent ensuite être reportés dans des branches plus basses par des ***pull request***s.

La section ***Branches*** de la fenêtre *Team Explorer* permet de sélectionner la branche de travail. La branche courante peut aussi être sélectionnée en en bas à droite dans la barre d’état.

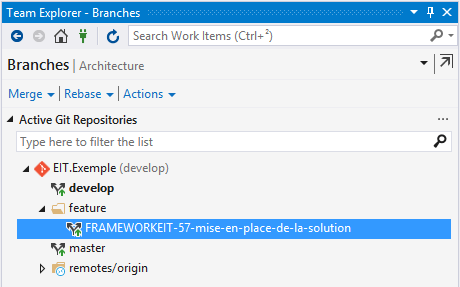


Figure – La section *Branches* permet de sélectionner la branche de travail

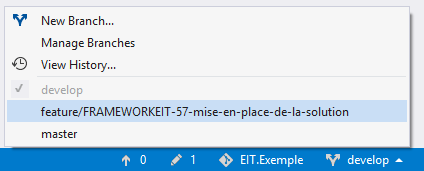


Figure - La branche de travail peut être sélectionnée dans la barre d'état

# Consulter l’historique des modifications

Avec Git, un historique de modifications est composé d’une succession de *commits*. Visual Studio propose des interfaces graphiques pour consulter les historiques. Il est possible de consulter l’historique des modifications à partir d’une branche ou d’un fichier.

## Consulter l’historique d’une branche

L’historique de la branche courante du dépôt local est accessible depuis le menu *Actions* de la section *Changes* de la fenêtre *Team Explorer*. Dans le menu *Actions*, cliquez sur *View History* pour accéder à l’historique local des *commits*.

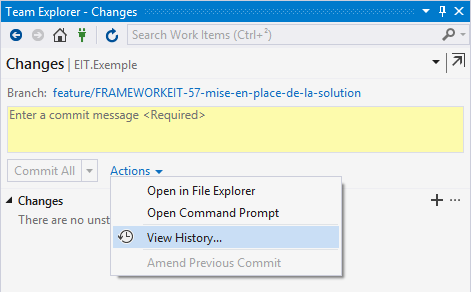


Figure - Le menu *Actions* de la section *Changes* permet de consulter l'historique des *commits*

Vous pouvez également cliquer sur le nom de la branche dans la barre d’état puis cliquer sur ***View History…***

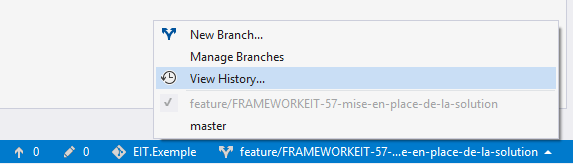


Figure - Accès à l'historique de la branche courante depuis la barre d'état

Il est possible de consulter l’historique d’une autre branche locale ou distante à partir de la section *branches* de la fenêtre Team Explorer. Faites un clic-droit sur la branche dont vous souhaitez consulter l’historique pour ouvrir le menu contextuel puis cliquez sur ***View History…***

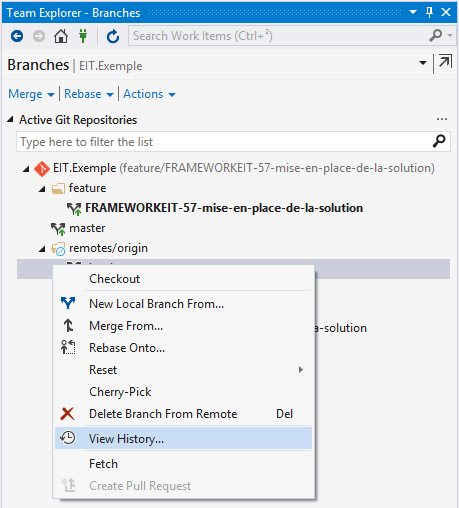


Figure - Consultation de l'historique d'une branche

L’historique d’une branche retrace tous les *commits* de la vie de la branche sur le dépôt, quelques soient les fichiers qui ont été modifiés par les *commits*. Pour chaque *commit*, l’historique affiche l’identifiant, l’auteur, la date et la première ligne du message. Un graphique permet également de visualiser l’enchaînement et les fusions.

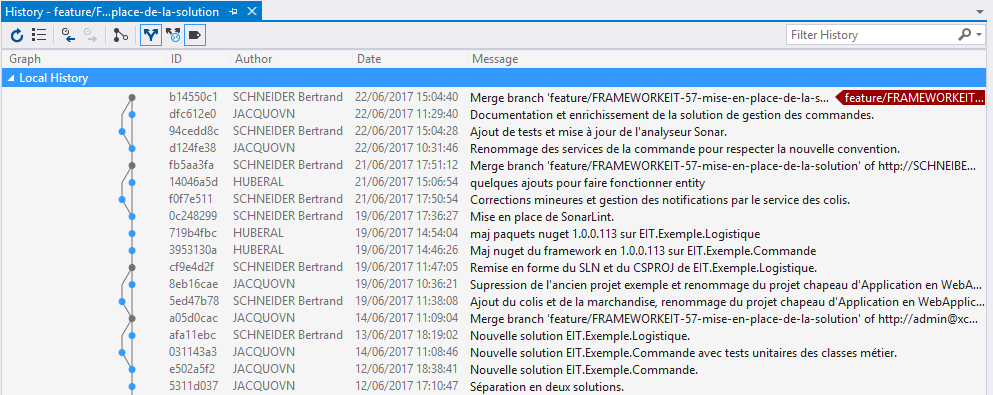


Figure – Fenêtre d'historique d'une branche Git dans Visual Studio

Pour visualiser les détails d’un *commit*, double-cliquez sur la ligne correspondante. Ceci ouvrira les une section *Commit Details* dans la fenêtre *Team Explorer*. Le détail affiche notamment la liste des fichiers modifiés, ajoutés ou supprimés. Vous pouvez visualiser les modifications effectuées sur un fichier en double-cliquant sur son nom.

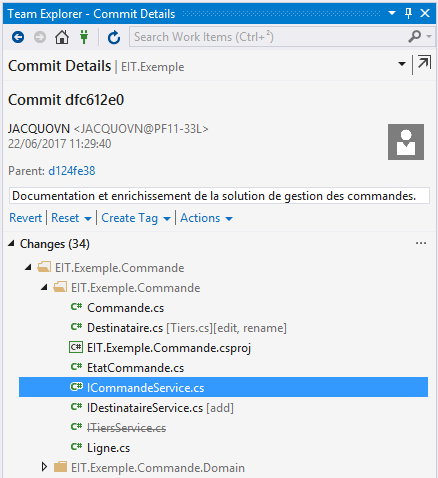
**

Figure - Fenêtre *Team Explorer* affichant le détail d'un *commit*

## Consulter l’historique d’un fichier

Il est possible de consulter l’historique d’un fichier. L’historique n’affichera alors que les commits ayant modifié le fichier. Pour consulter l’historique d’un fichier, faites un clic-droit sur un fichier dans la fenêtre *Solution Explorer*. Dans le menu contextuel qui s’ouvre, cliquez sur ***View History…***

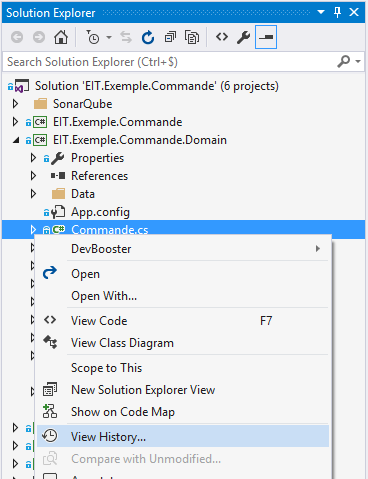


Figure - Consultation de l'historique d'un fichier

Sous vous voulez consulter les modifications effectuées par le commit sur l’un des fichiers, faites un clic-droit sur le commit dans la fenêtre d’historique pour ouvrir un menu contextuel puis cliquez sur ***Compare with Previous…***

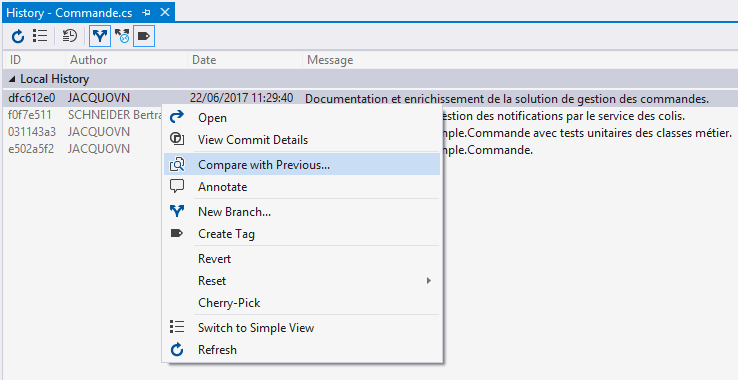


Figure - Comparaison d'un *commit* avec le précédent à partir de l’historique d'un fichier

# Archiver des modifications

Lorsque des modifications ont été effectuées sur le code et qu’elles sont fonctionnelles, elles peuvent être archivées dans un ***commit***. Un *commit* représente une « étape » sur la ligne des versions du code, de manière analogue à un *changeset* pour RTC.

**Attention !** Avec Git, archiver des modifications permet de les historiser dans votre dépôt local mais ne les envoie pas sur le dépôt distant.

## Archiver toutes les modifications

Pour archiver des modifications, rendez-vous dans la section ***Changes*** de la fenêtre *Team Explorer*. Cette section contient la liste des fichiers modifiées que vous allez archiver.

Vous devez renseigner un commentaire qui décrit les fonctionnalités contenues dans le *commit*. La première ligne du commentaire constitue le titre du *commit*. La première ligne doit respecter les éventuelles normes de nommage des *commits* en vigueur.

Une fois le commentaire rédigé, vérifiez que la liste des fichiers correspond au contenu souhaité du commit et cliquez sur ***Commit All***. Vos modifications sont alors archivées, il ne faudra pas oublier de les publier.

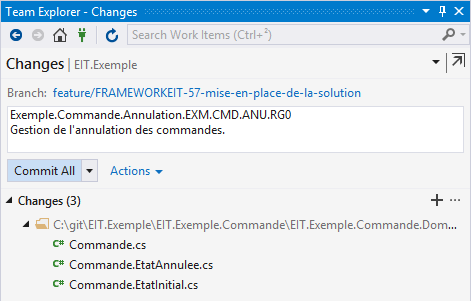


Figure - Archivage des modifications associées à une fonctionnalité

## Archiver une partie des modifications

Il est possible d’archiver uniquement une partie des fichiers modifiés. Ceci est fait à partir de l’opération ***stage***. L’opération *stage* permet de préparer des fichiers à placer dans un *commit*. Il est ensuite possible de constituer un *commit* uniquement à partir de ces éléments.

Dans la section *Changes*, faites un clic-droit sur le fichier concerné pour ouvrir le menu contextuel puis cliquez sur *stage*. Une nouvelle action ***Commit Staged*** devient alors disponible. Cliquez sur ce bouton pour constituer un commit uniquement à partir des fichiers *staged*.

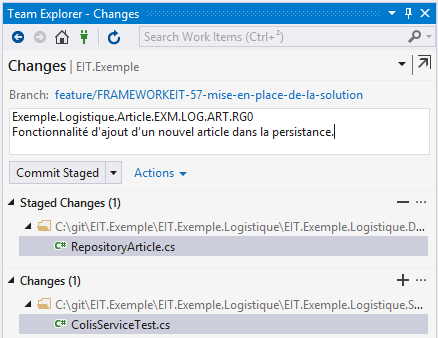


Figure - Il est possible de constituer un commit uniquement à partir des fichiers *staged*

## Annuler l’archivage de modifications

Parfois, la création d’un *commit* est une erreur car il manque des modifications, ou car plusieurs *commits* non publiés devraient être fusionnés en un seul pour une question de lisibilité. Il est possible d’annuler l’archivage des modifications en effectuant un ***mixed reset*** sans effacer les modifications elles-mêmes.

Le *mixed reset* consiste à se replacer à l’état d’un commit antérieur mais sans effacer les modifications des *commits* suivants. Les *commits* plus récents seront alors supprimés, mais les modifications associées seront conservées. Les modifications désarchivées pourront alors être retravaillées puis replacées dans un nouveau commit.

1. Dans l’historique local, identifiez le dernier *commit* que vous souhaitez conserver.
2. Vérifiez que tous les *commits* plus récents vous appartiennent et qu’ils n’ont pas été envoyés sur un dépôt distant.
3. Faites ensuite un clic-droit sur le dernier commit à conserver. Dans le menu contextuel qui s’ouvre, cliquez sur ***Reset and Keep Changes (--mixed)*** pour revenir à l’état du *commit* sélectionné.
4. Rafraîchissez ensuite l’historique pour vérifier que vos derniers *commits* ont bien disparu.
5. Vérifiez dans la section *Changes* de la fenêtre *Team Explorer* que les modifications des *commits* disparus sont à nouveau prêtes à être archivées.

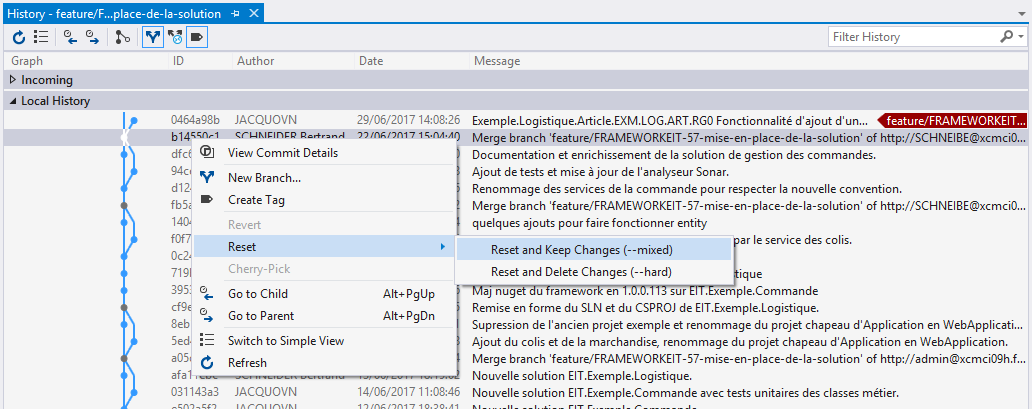


Figure - Utilisation de l'historique pour revenir à l'état d'un commit spécifique en conservant les changements

**Attention !** Il est important qu’aucun des *commits* plus récents que le *commit* de référence n’aient été envoyés sur un dépôt distant. Si c’est le cas, ces *commits* reviendront à la prochaine synchronisation avec le dépôt distant.

**Attention !** Evitez de cliquer sur *Reset and Delete Changes (--hard)* car les modifications contenues dans les *commits* supprimés seront alors perdues !

# Récupérer les dernières modifications

Les modifications sur le dépôt courant sont récupérées depuis le dépôt distant sous la forme de *commits*. Pour récupérer la liste des derniers *commits* du dépôt distant, il faut d’abord effectuer un ***fetch***. Rendez-vous dans la section ***Synchronization*** de la fenêtre *Team Explorer* puis cliquez sur *fetch*.

Il existe deux méthodes pour récupérer les derniers commits : avec une opération ***pull*** ou avec une opération ***rebase***.

## Récupérer les modifications avec une opération pull

L’opération *pull*[[3]](#footnote-3) consiste à récupérer les derniers *commits* de la branche distante et à les replacer à la fin de votre branche locale. L’opération pull ne pose pas de problème si vous n’avez pas de commits non publiés sur votre branche locale mais peut produire des commits de ***merge*** dans le cas inverse.

Dans la section *Synchronization*, cliquez sur *pull* pour récupérer les commits distants dans votre dépôt local. Si vous avez des modifications en cours qui ne sont pas archivées, Visual Studio vous demandera de les archiver avant d’effectuer un *pull*.

Mais si vous avez des modifications archivées mais non publiées, il peut vous être demandé d’effectuer un *merge* en cas de conflit lors de la récupération des *commits* du dépôt distant. Vous devrez résoudre les conflits dans chacun des fichiers concernés. Dans la fenêtre de résolution des conflits, cliquez sur ***Merge*** pour fusionner les fichiers. Une fois la fusion terminée, un *commit* de *merge* contenant le code fusionné sera automatiquement créé.

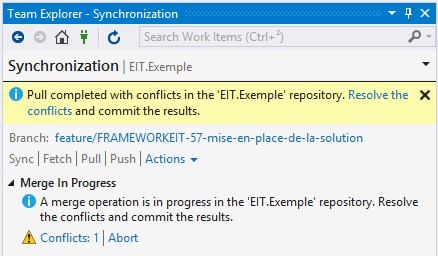


Figure 4 - La synchronisation indique un conflit lors du *pull*

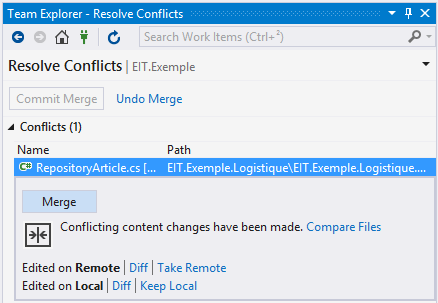


Figure 5 – L’interface propose de fusionner manuellement les fichiers en conflit

## Récupérer les modifications avec une opération rebase

L’opération *rebase* consiste à récupérer l’historique d’une branche et à rejouer les derniers *commits* de la branche de travail à la fin de l’historique. On peut donc récupérer les dernières modifications à l’aide d’un *rebase* de la branche locale sur la branche distante. Contrairement au *pull*, ce sont les *commits* locaux qui sont placés à la fin de l’historique et non les *commits* distants. La récupération des modifications avec *rebase* permet de limiter les *commits* de *merge*.

**Remarque :** Les *commits* placés en fin d’historique ne sont pas exactement les même que les commits de la branche locale, mais des copies dont contenu est identique.

Pour effectuer un *rebase* rendez-vous dans la section *Branches* de la fenêtre *Team Explorer*. Localisez la branche distante qui correspond à votre branche de travail. Les branches distantes se trouvent sous le répertoire *remotes/origin*. Ouvrez le menu contextuel de la branche distante puis cliquez sur ***Rebase Onto…*** Dans le formulaire qui apparaît, vérifiez que le nom des branches est correct puis cliquez sur le bouton ***Rebase*** pour terminer l’opération.

**Attention !** Vérifiez bien que la branche distante correspond à votre branche de travail. Par exemple, si vous travaillez sur la branche feature/logistique votre rebase doit se faire sur la branche origin/feature/logistique.

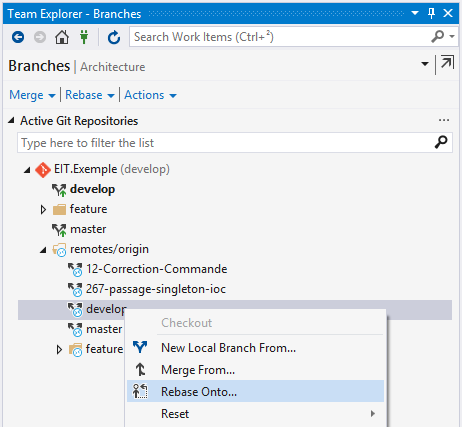


Figure 6 - Le menu contextuel d'une branche distante propose d'effectuer un *rebase*

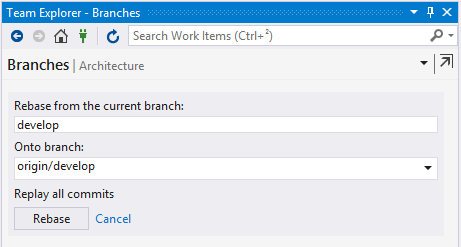


Figure 7 - Le formulaire de *rebase* permet de vérifier le nom des branches

# Publier des modifications

Une fois les modifications archivées, elles doivent être publiées sur le dépôt distant avec l’opération ***push***. Avant d’effectuer un *push*, la liste des *commits* doit être synchronisée entre le dépôt distant et le dépôt local. Si votre dépôt local n’est pas à jour, Git vous le signalera et vous devrez effectuer un *pull* avant de faire votre *push*.

Pour publier vos modifications, rendez-vous dans la section *Synchronisation* de de la fenêtre *Team Explorer*. Vérifiez attentivement la liste des *commits* qui seront publiés, notamment que leur nommage est conforme aux règles et s’ils ne doivent pas être fusionnés ou découpés. Si la liste des *commits* vous semble bonne, cliquez sur *push*.

# Annuler des modifications

Si vous souhaitez annuler des modifications, la marche à suivre dépend du cas dans lequel vous vous trouvez :

* Vos modifications ne sont pas encore archivées
* Vos modifications sont archivées mais uniquement sur un dépôt local
* Vos modifications sont déjà archivées sur un dépôt distant

## Annuler des modifications non archivées

Il est très simple d’annuler des modifications non archivées. Il vous suffit de sélectionner les éléments que vous souhaitez remettre à l’état initial dans la section *Changes* de la fenêtre *Team Explorer*, de faire un clic-droit pour ouvrir le menu contextuel, et de cliquer sur ***Undo Changes…*** Une boite de dialogue vous demandera alors de confirmer l’annulation.

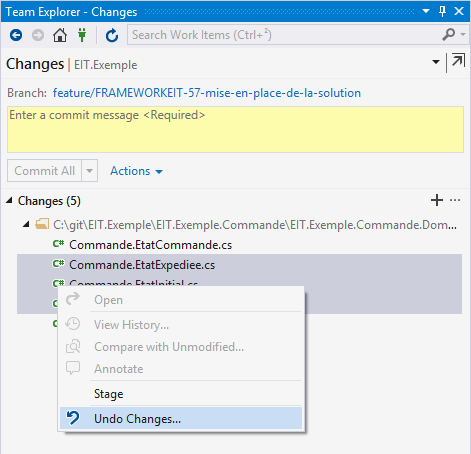


Figure - Il est possible d'annuler des modifications depuis la section *Changes*

Il est également possible d’annuler des modifications sur des fichiers directement depuis la fenêtre *Solution Explorer*. De la même manière, il vous faut sélectionner les éléments à annuler, faire un clic droit pour obtenir le menu contextuel et cliquer sur ***Undo Changes…*** Une boite de dialogue vous demandera alors de confirmer l’annulation.

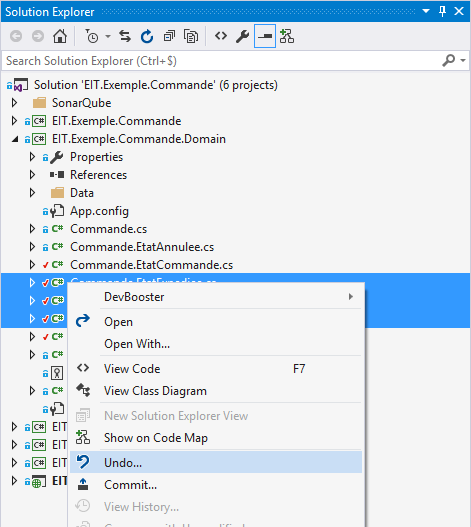


Figure - Il est possible d'annuler des modifications depuis la fenêtre *Solution Explorer*

Dans les deux cas, la commande va remettre les éléments sélectionnés dans l’état où ils étaient au dernier *commit*. Ce retour arrière est effectué grâce à l’opération ***checkout*** de Git.

## Annuler des modifications archivées sur un dépôt local

Si vous souhaitez annuler des modifications qui ont déjà été archivées sur un dépôt local, vous devrez remettre le dépôt à l’état d’un *commit* précédent avec l’opération ***hard reset*** de Git. Cette manipulation peut être effectuée depuis Visual Studio.

Concrètement, vous devez replacer l’état de votre dépôt local à celui d’un *commit* précédent. Tous les *commits* plus récents que le *commit* sélectionné seront alors ignorés. Ainsi, tant que ces *commits* plus récents n’ont pas été envoyés sur un dépôt distant, ceci vous permet d’annuler vos dernières modifications archivées.

1. Dans l’historique local, identifiez le dernier *commit* que vous souhaitez conserver.
2. Vérifiez que tous les *commits* plus récents vous appartiennent et qu’ils n’ont pas été envoyés sur un dépôt distant.
3. Faites ensuite un clic-droit sur le dernier commit à conserver. Dans le menu contextuel qui s’ouvre, cliquez sur ***Reset and Delete Changes (--hard)*** pour revenir à l’état du *commit* sélectionné.
4. Rafraîchissez ensuite l’historique pour vérifier que vos derniers *commits* ont bien disparu.

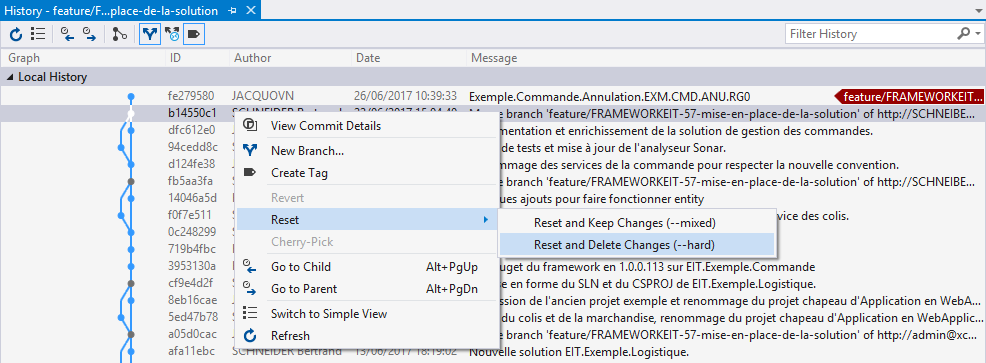


Figure - Utilisation de l'historique pour revenir à l'état d'un commit spécifique

**Attention !** Il est important qu’aucun des *commits* plus récents que le *commit* de référence n’aient été envoyés sur un dépôt distant. Si c’est le cas, ces *commits* reviendront à la prochaine synchronisation avec le dépôt distant.

## Annuler des modifications archivées sur un dépôt distant

Il n’est pas permis d’effacer totalement des modifications déjà archivées sur un dépôt distant. Si vous souhaitez annuler des modifications qui sont déjà archivées sur un dépôt distant, vous devrez envoyer un nouveau commit qui annule les modifications du précédent. Ce commit peut être généré automatiquement par Git avec l’opération ***revert***. Cette manipulation peut être effectuée depuis Visual Studio.

1. Dans l’historique local, faites un clic-droit sur le *commit* que vous souhaitez annuler. Dans le menu contextuel, cliquez sur ***Reset and Delete Changes (--hard)*** pour revenir à l’état du *commit* sélectionné.
2. Rafraîchissez ensuite l’historique pour vérifier qu’un *commit* d’annulation a bien été produit en tête de l’historique.
3. Envoyez le *commit* d’annulation généré par le *revert* sur le dépôt distant.

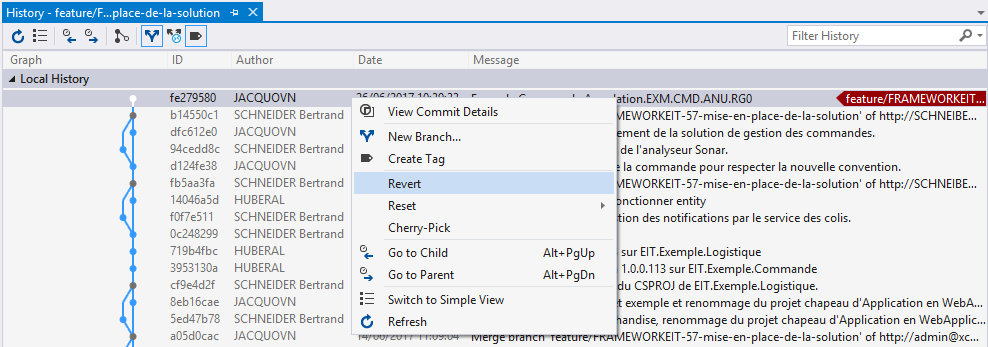


Figure - Utilisation de l'historique pour créer un commit d'annulation

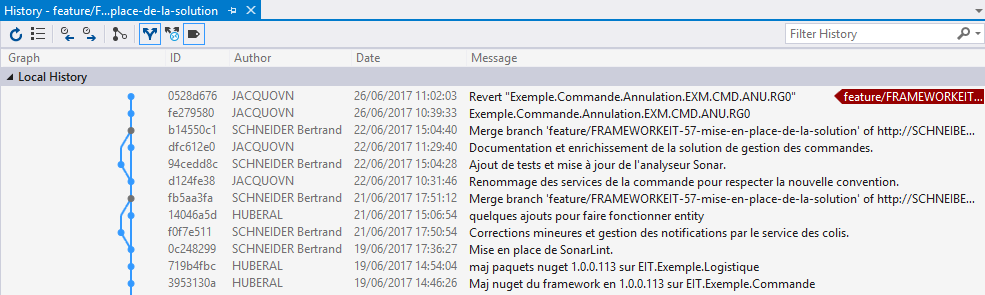


Figure - Apparition d'un commit d'annulation après le *revert*

1. <https://www.visualstudio.com/en-us/docs/git> [↑](#footnote-ref-1)
2. <https://git-scm.com/doc> [↑](#footnote-ref-2)
3. <https://www.visualstudio.com/en-us/docs/git/tutorial/pulling> [↑](#footnote-ref-3)