



# Débuter avec GIT & Eclipse

• • •

François ANDRE



## Objectifs



Comprendre l'intérêt des gestionnaires de code



Effectuer des manipulations simples



Découvrir une stratégie de gestion des projets

# Introduction

```
fonts
skins
custom.css
ie.css
theme.css
theme-animate.css
theme-blog.css
theme-elements.css
theme-shop.css
chimp

        +image_src + 'src'],
85
86
87
88    if( !function_exists('hex2rgb') ) {
89        function hex2rgb($hex_str, $return_string = false, $separator = ',') {
90            $hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a proper hex string
91            $rgb_array = array();
92            if( strlen($hex_str) == 6 ) {
93                $color_val = hexdec($hex_str);
94                $rgb_array['r'] = 0xFF & ($color_val >> 0x10);
95                $rgb_array['g'] = 0xFF & ($color_val >> 0x8);
96                $rgb_array['b'] = 0xFF & $color_val;
97            } elseif( strlen($hex_str) == 3 ) {
98                $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
99                $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
100               $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
101           } else {
102               return false;
103           }
104       }
105   }
106 }
```

# Pourquoi ?

Utiliser un Source Code Manager



# Pour soi

---

---

Il est utile

- de pouvoir retrouver les **différentes étapes** de son travail
- d'avoir **plusieurs versions** de son travail en même temps



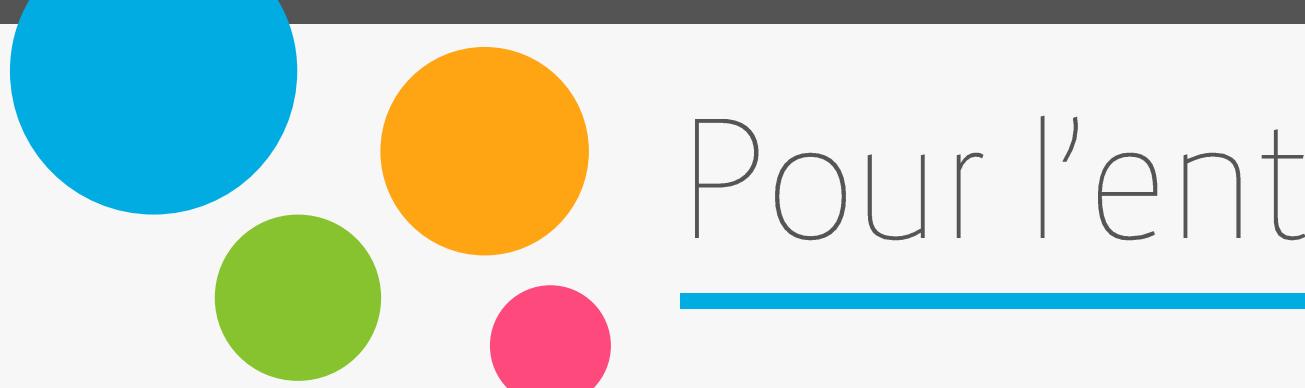


## Pour son équipe

---

Toute production - code ou **autre** - fait partie du patrimoine de son équipe et doit être **conservée** de manière pérenne (c.a.d avec des **pratiques connues**)

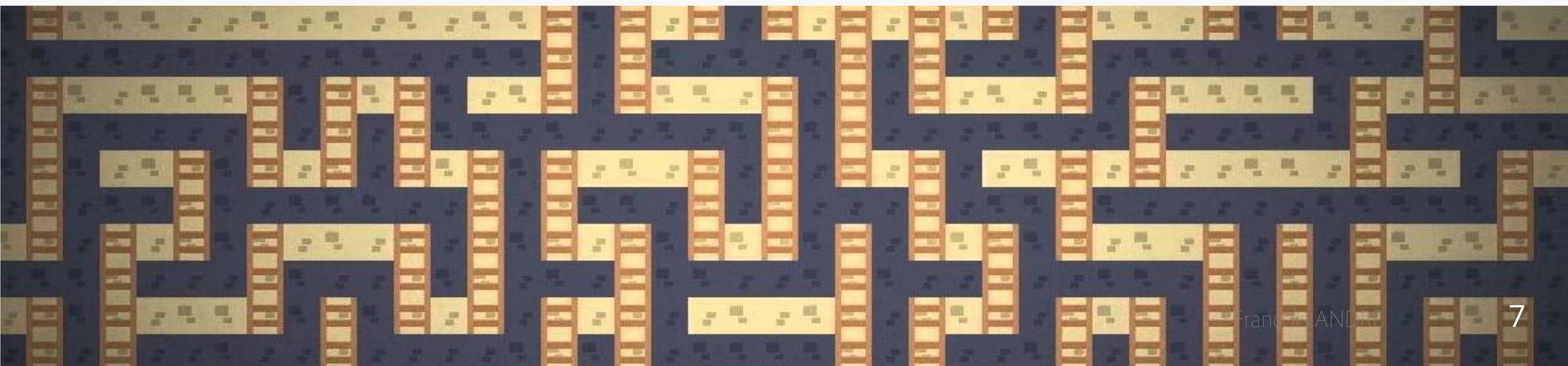




# Pour l'entreprise

---

Toute production - code ou **autre** - fait partie du patrimoine de son équipe et doit être **conservée** de manière pérenne (c.a.d avec des **pratiques connues**)





Pourquoi ?

GIT



# Pourquoi Git ?

---

- Il existe beaucoup de SCM : CVS, SVN, **Git**, Mercurial,...
- En général chaque nouvelle génération d'outil rend la précédente obsolète

Git apporte **beaucoup de souplesses** :

- Le travail déconnecté
- La possibilité de travailler facilement sur **plusieurs versions** de l'application (les *branches*)
  - La version de production
  - La version en développement
- La possibilité de **travailler à plusieurs** :
  - Avec un serveur central
  - Avec plusieurs serveurs centraux
  - Directement entre plusieurs développeurs

# Pourquoi Git ?



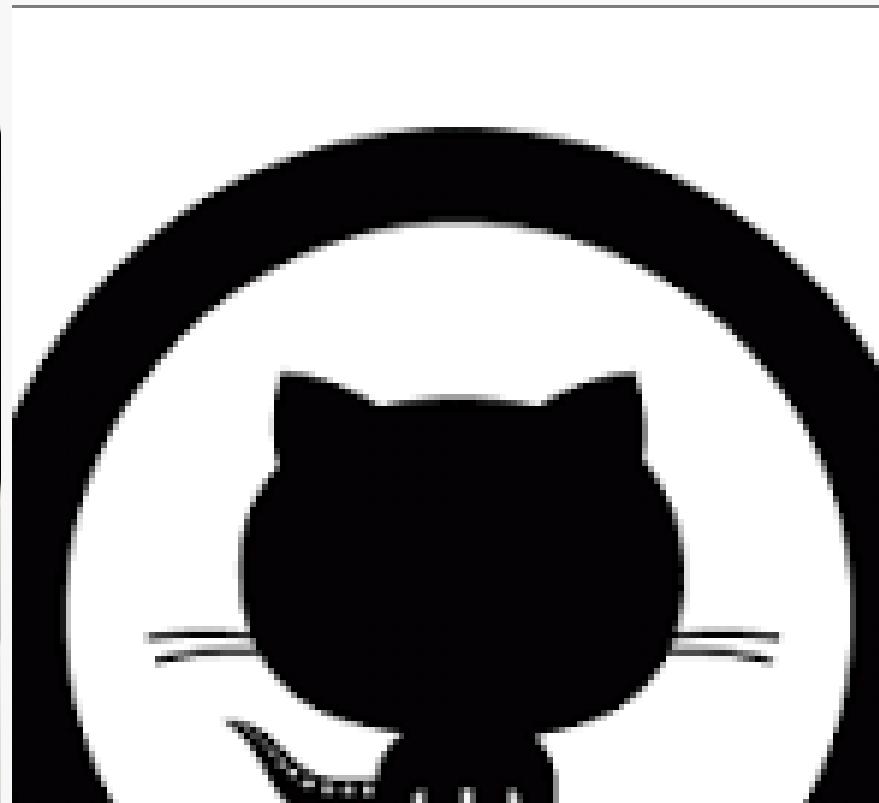
## Rapide

La plupart des opérations s'effectuent sur le poste du développeur sans passer par le réseau



## Agnostique

Git est disponible sur tous les systèmes d'exploitation. Il peut être utilisé en ligne de commande ou via un IDE.



## Social

Il existe, sur Internet, un grand nombre de forge basées sur Git. Elles ajoutent des fonctionnalités sociales permettant de proposer des évolutions, des corrections...



## Efficace

Via ses fonctionnalités, Git permet de mettre en place des organisations du développement permettant de résoudre les soucis usuels, par exemple GitFlow.





# IDE Utilisé

---

- Par défaut Git s'utilise avec l'invite de commande
- Les principaux IDE (Eclipse, Netbeans, IntelliJ ...) masquent ces commandes en les intégrant à leur interface
- Les exercices proposés utilisent l'IDE Eclipse
- Toutefois, afin de permettre le passage vers d'autres IDE et l'approfondissement de l'outil, les commandes sont indiquées en marge des exercices

## MODE COMMANDE

---

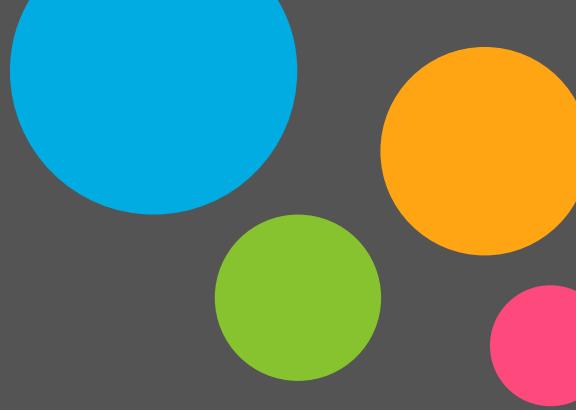
```
$ git status  
$ git add ...
```



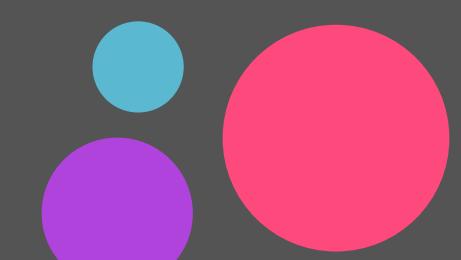
# Serveur Git

---

- Plusieurs serveurs Git en ligne et gratuits existent ([GitHub](#), [BitBucket](#),...)
- Il existe également des serveurs à installer localement ([GitLab C.E.](#))



# Quelques notions préliminaires



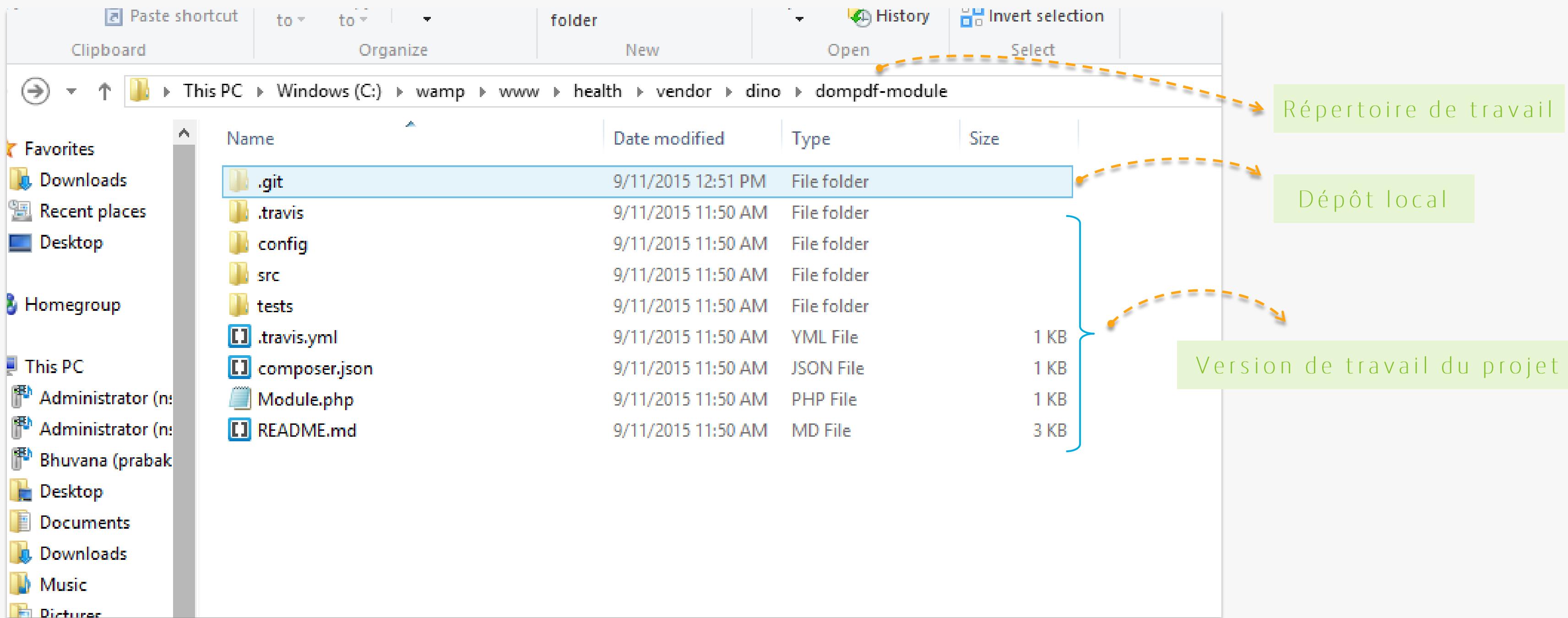


# Dépôt local et répertoire de travail

---

- Pour chaque projet, Git utilise un répertoire (.git) dans lequel il va mettre l'ensemble des données/métadonnées permettant de retracer l'histoire du projet. C'est le **dépôt local** du projet. On y trouve
  - Toutes les versions des fichiers
  - Les différentes branches
- Git va extraire de ce dépôt une version du projet dans un répertoire afin de permettre l'édition des fichiers. C'est le **répertoire de travail** du projet.
- *En général le dépôt local est un sous répertoire du projet*

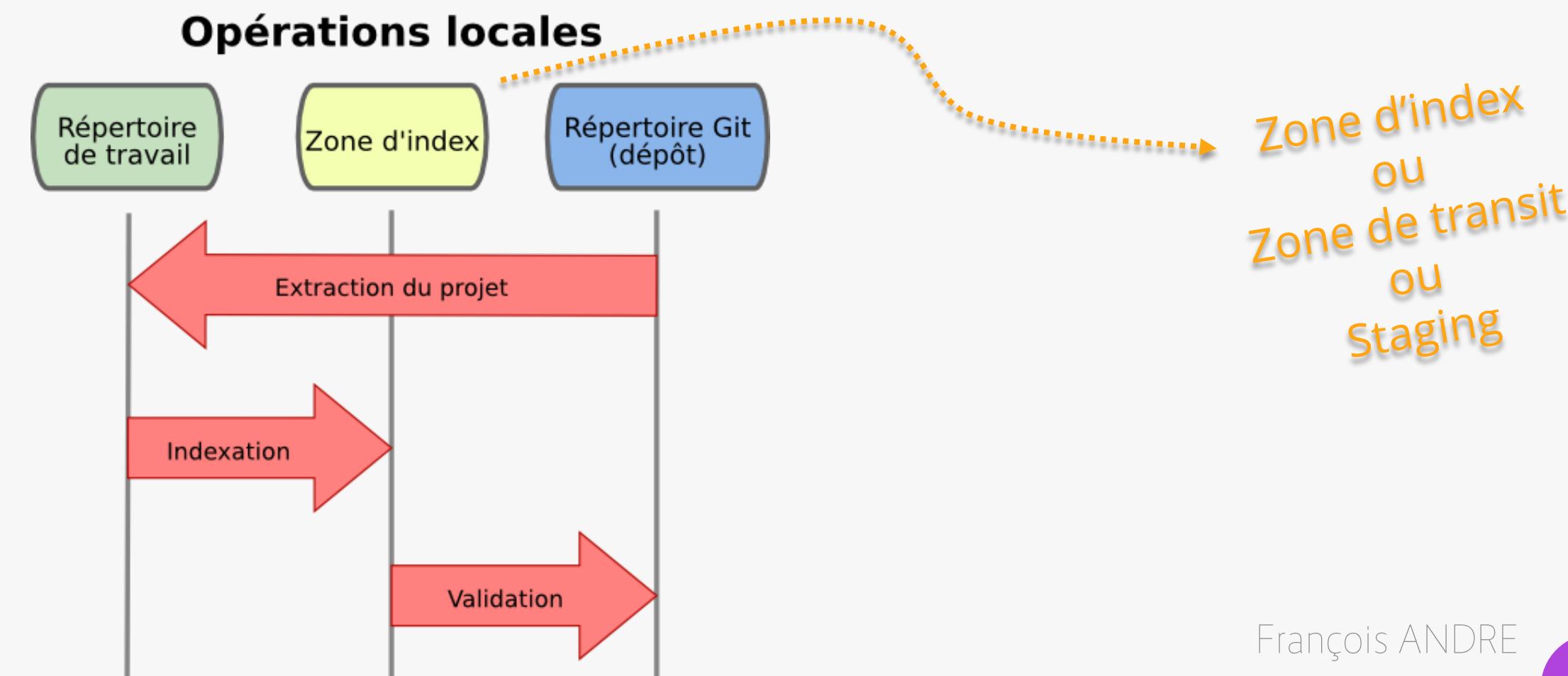
# Dépôt local et répertoire de travail



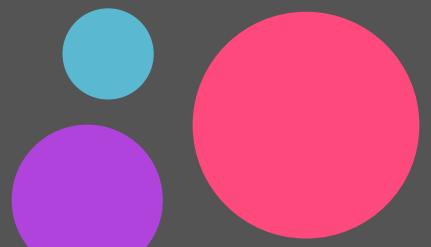
# Instantanés

Avec Git, le cycle classique est le suivant :

1. On ajoute/supprime/modifie des fichiers dans l'espace de travail
2. On groupe l'ensemble de ces modifications (on parle d'*indexation*)
3. On valide (*commit*) ces modifications dans le dépôt local en y associant un commentaire (exemple: correction de l'anomalie #145)

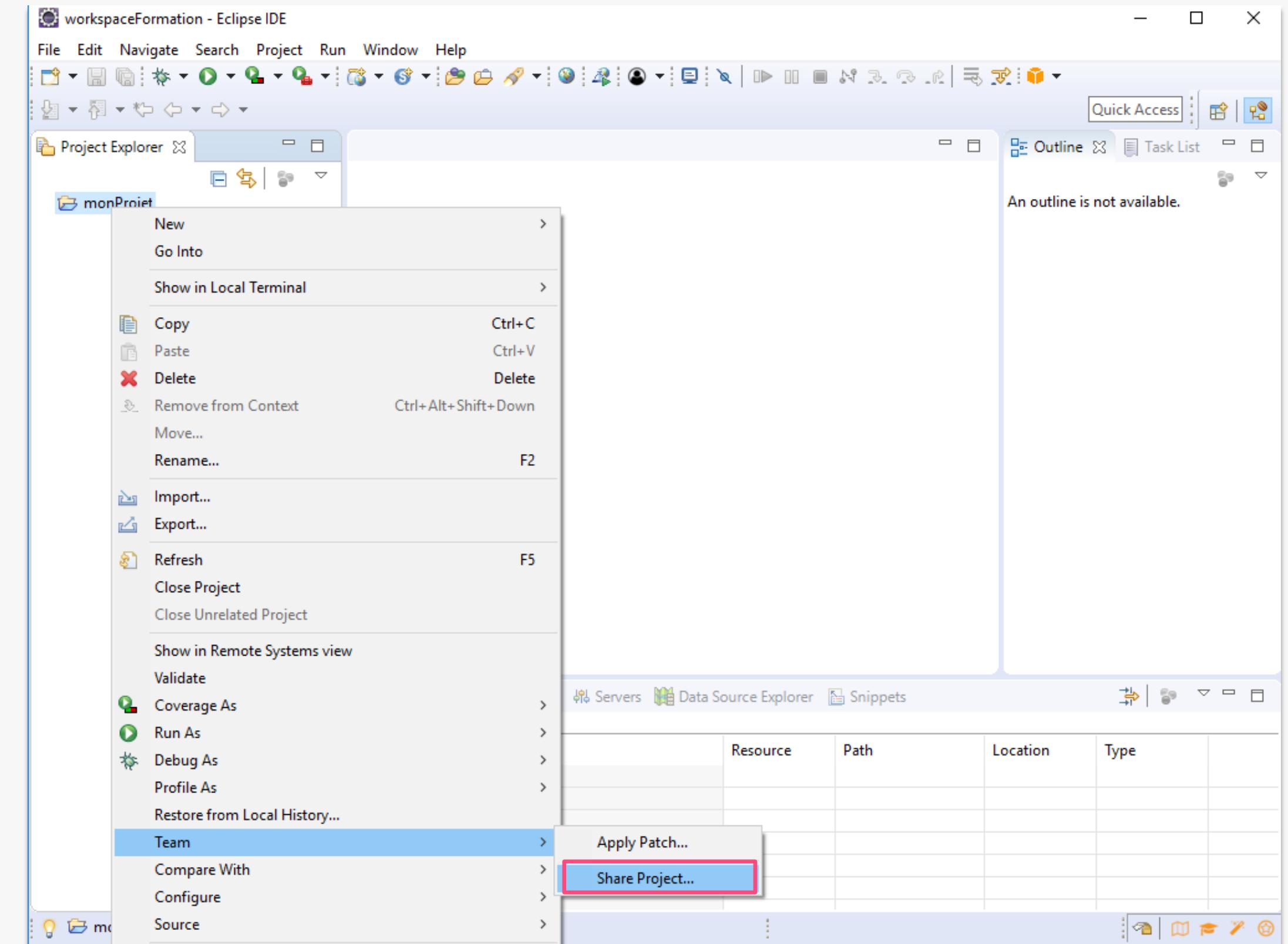


# Manipulations locales



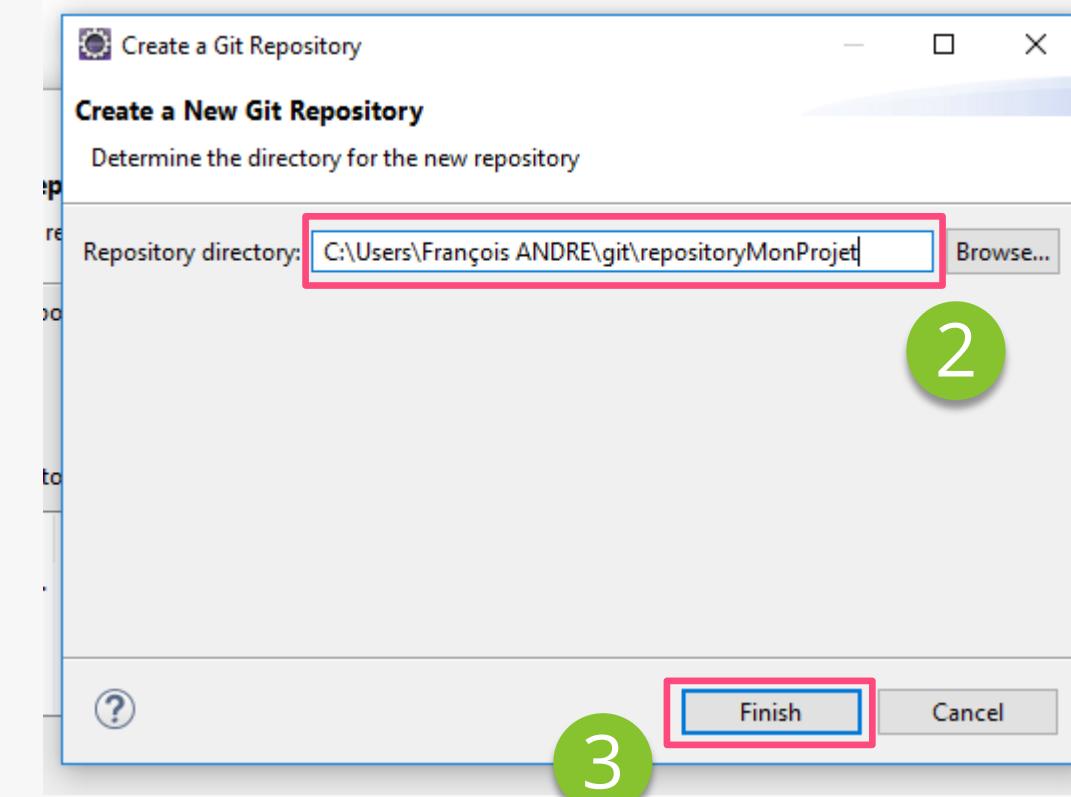
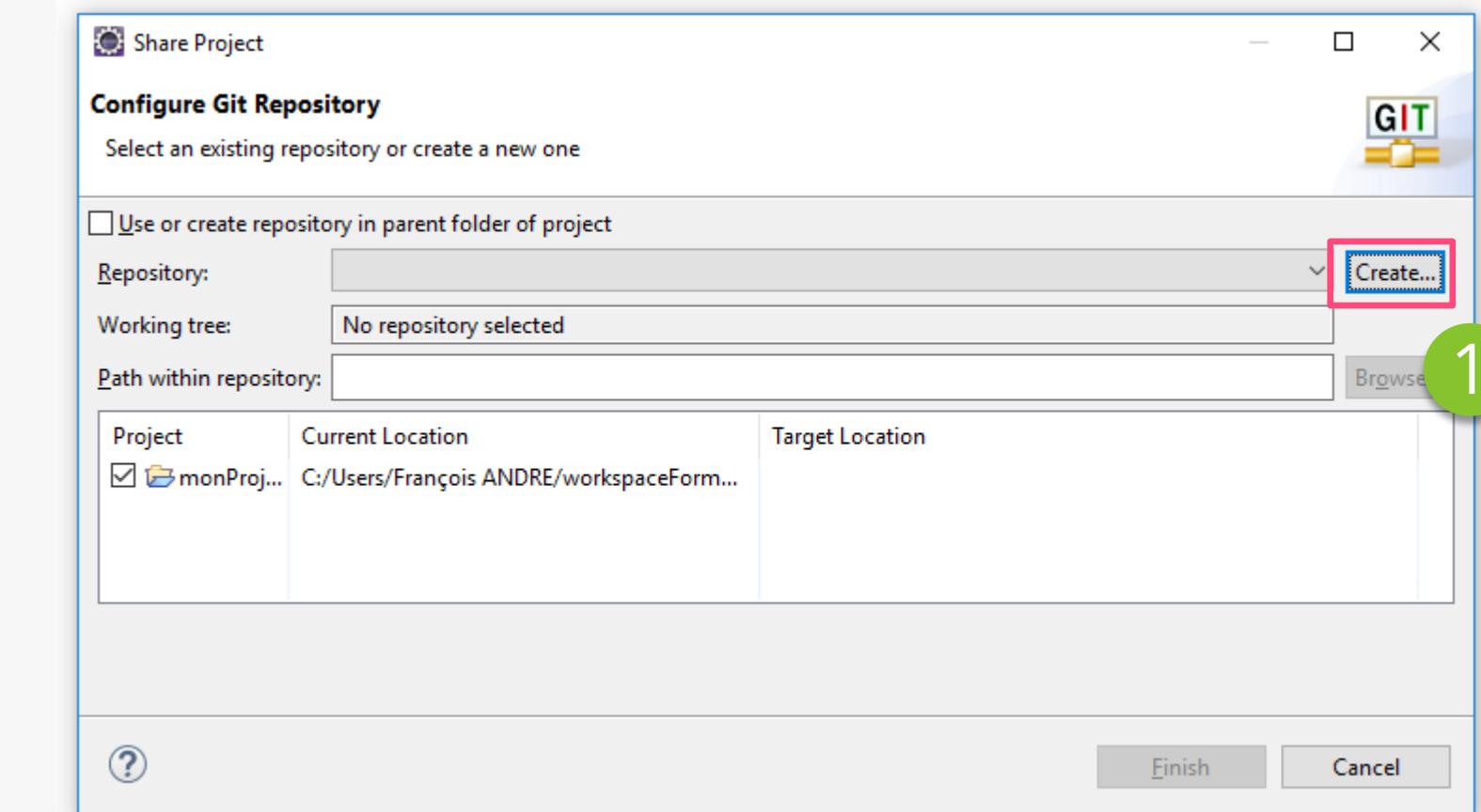
# Ajouter Git à un projet

Le projet peut déjà exister ou n'être qu'un répertoire vide



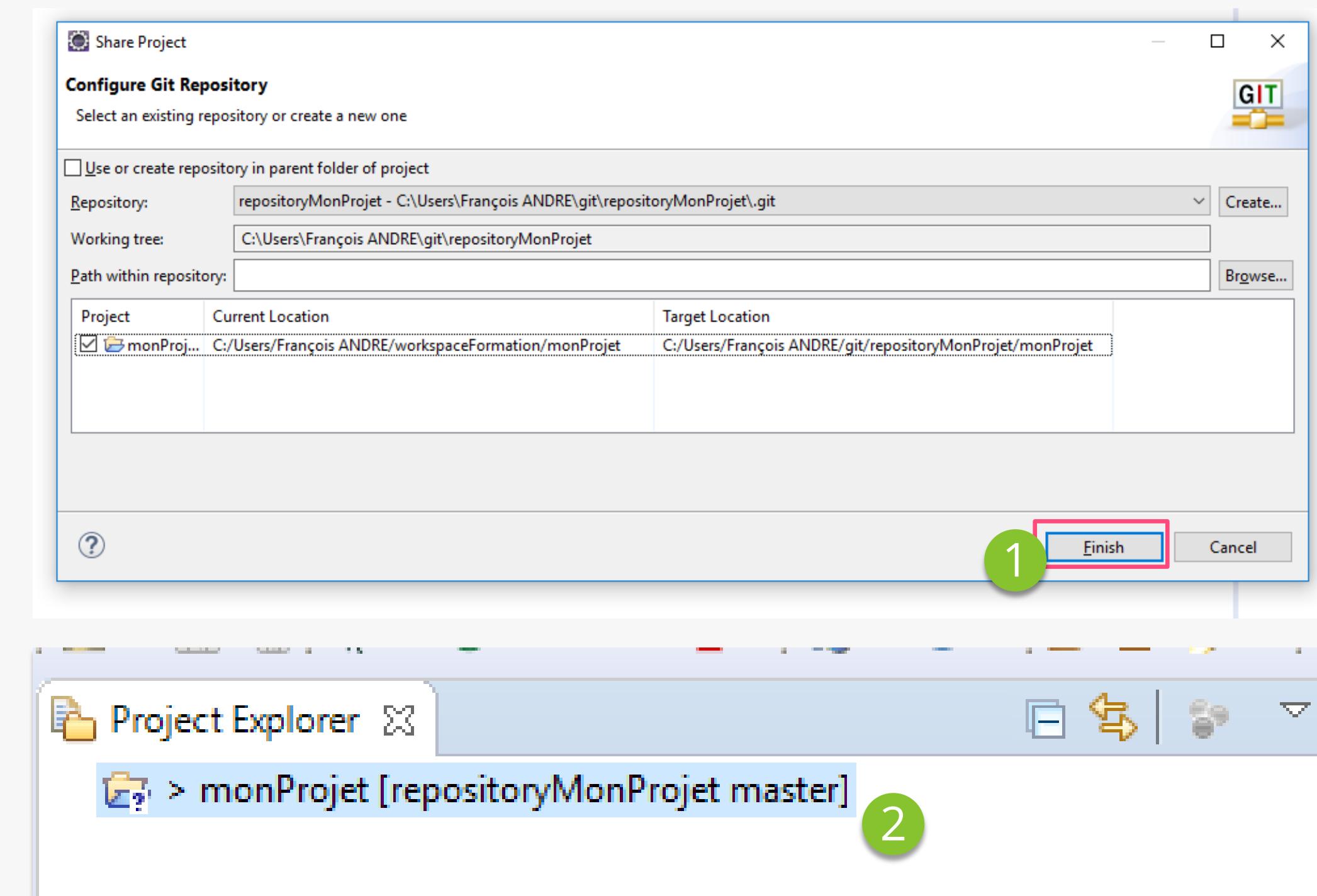
# Création du dépôt (1/2)

1. On va créer un nouveau dépôt (**Create...**)
2. On indique un répertoire (hors du workspace Eclipse)
3. On clique sur **Finish**



# Création du dépôt (2/2)

1. On clique sur **Finish** pour terminer la création du dépôt
2. Le projet indique désormais que Git est activé sur le projet





# Ajouter Git à un projet

---

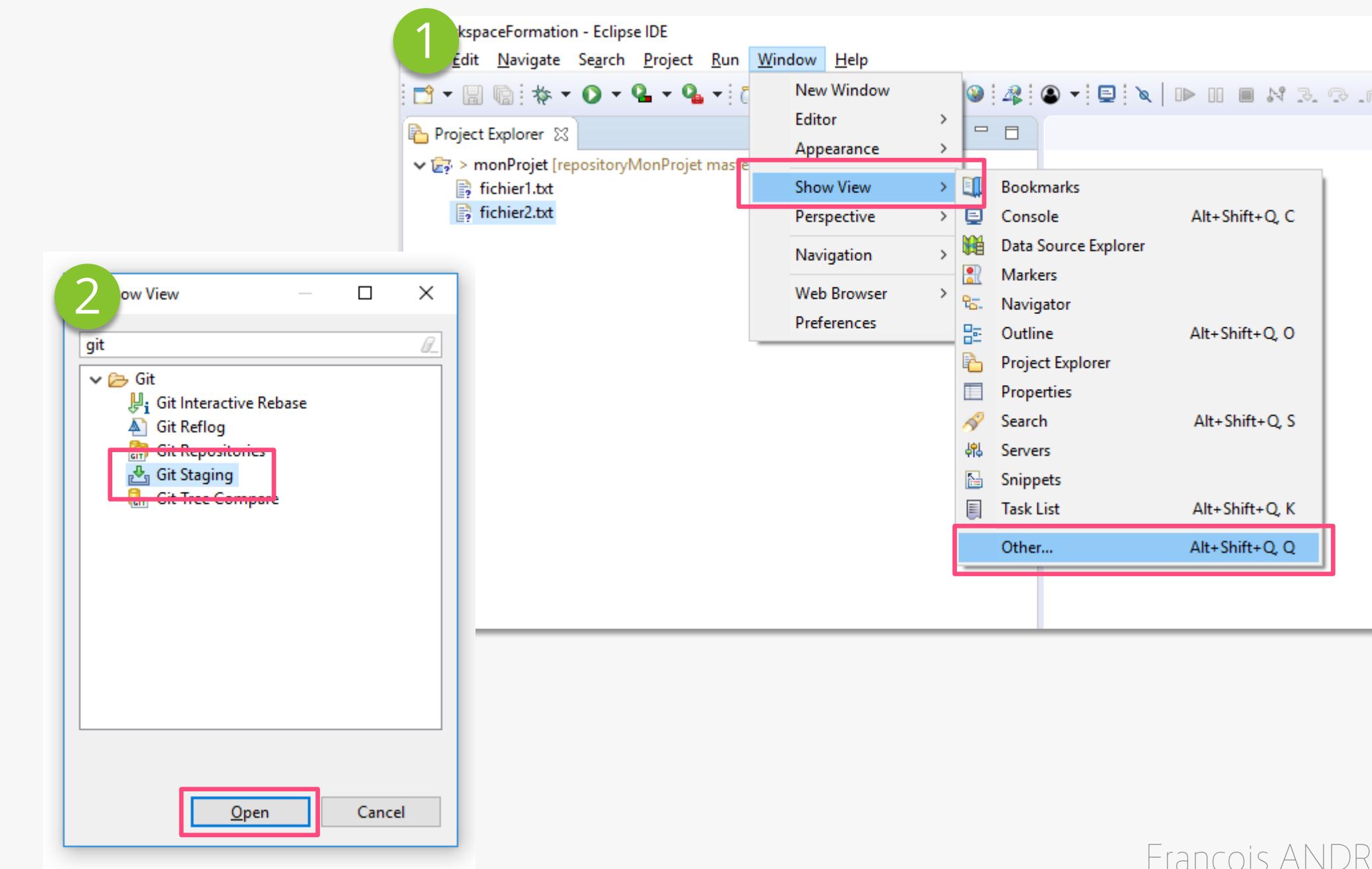
## MODE COMMANDE

```
$ git init
```

# Connaitre les fichiers en transit

Dans le projet, créez deux fichiers, *fichier1.txt* et *fichier2.txt* avec comme contenu: *Contenu du fichier X*

Ouvrir la vue *Git Staging*

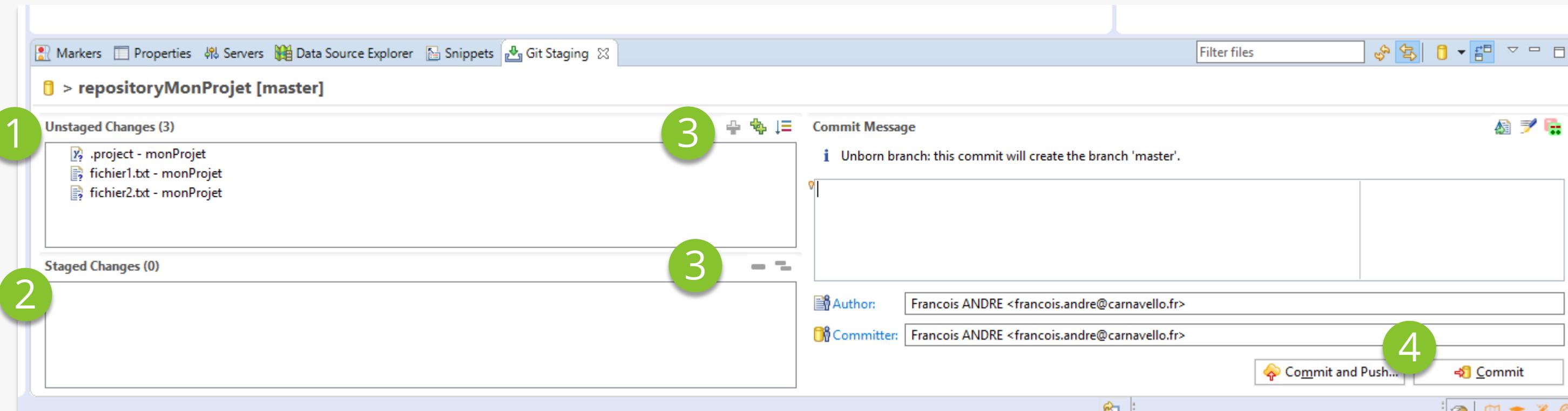


# Connaitre les fichiers en transit

Dans le projet, créez deux fichiers, *fichier1.txt* et *fichier2.txt* avec comme contenu: *Contenu du fichier X*

La vue Git Staging vous permet de

1. Lister les fichiers modifiés
2. Lister les fichiers modifiés ajoutés dans la zone de transit
3. Faire basculer les fichiers d'un état à l'autre
4. Effectuer un commit et un push

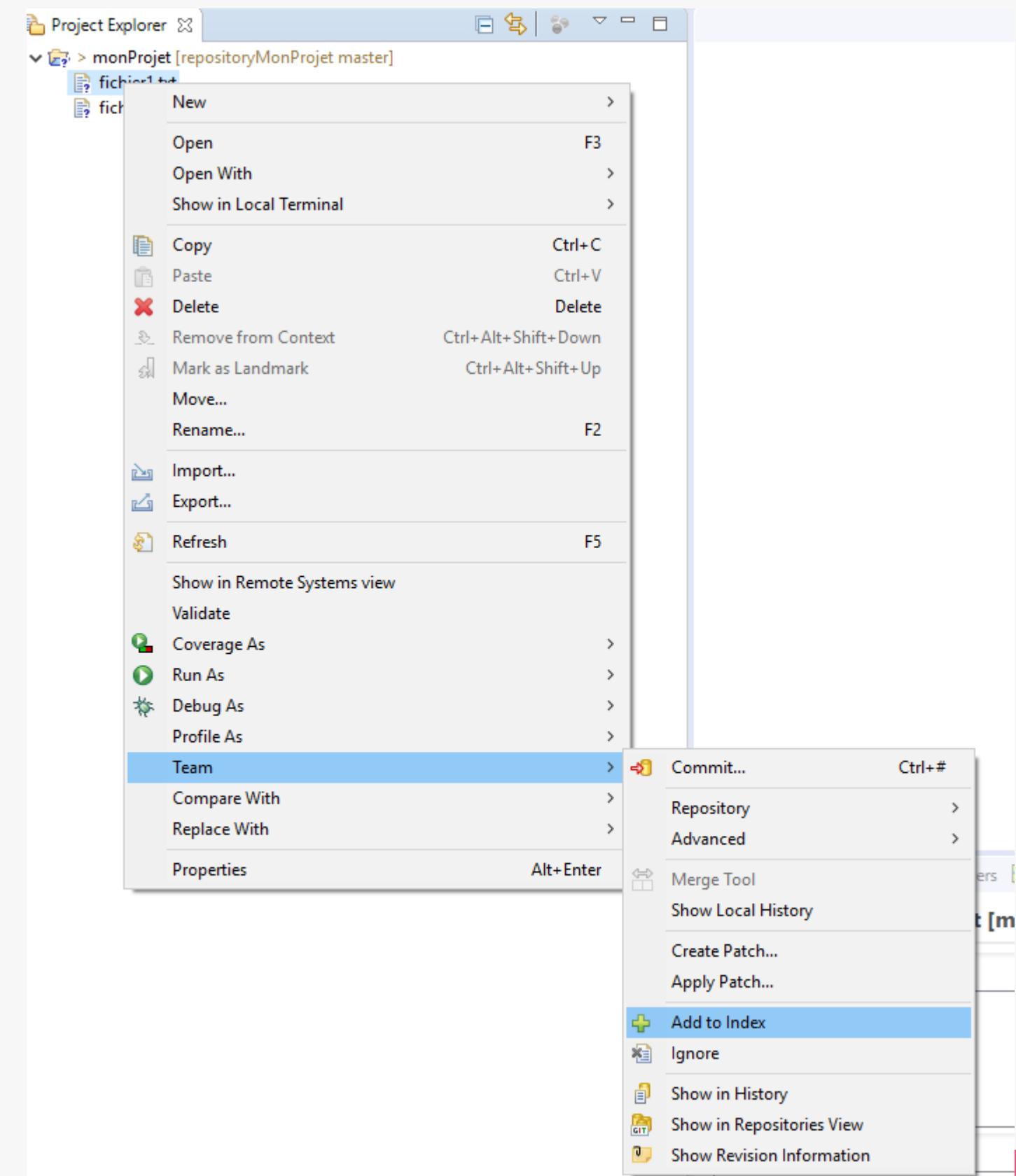


# Ajouter les fichiers à la zone de transit

Par le menu contextuel

Sélectionnez le fichier `fichier1.txt`, dans le menu contextuel cliquez sur :

**Team > Add to index**

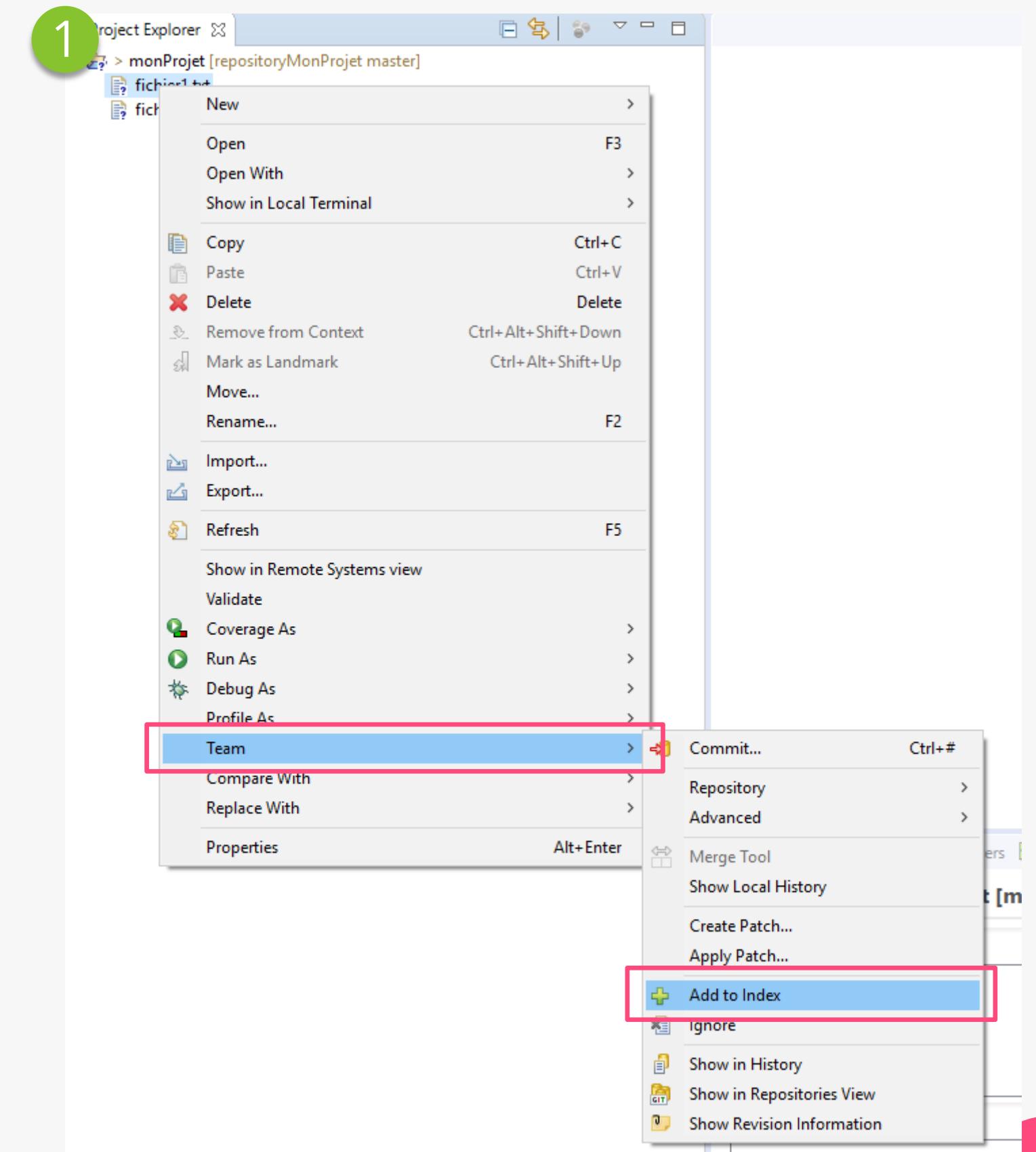
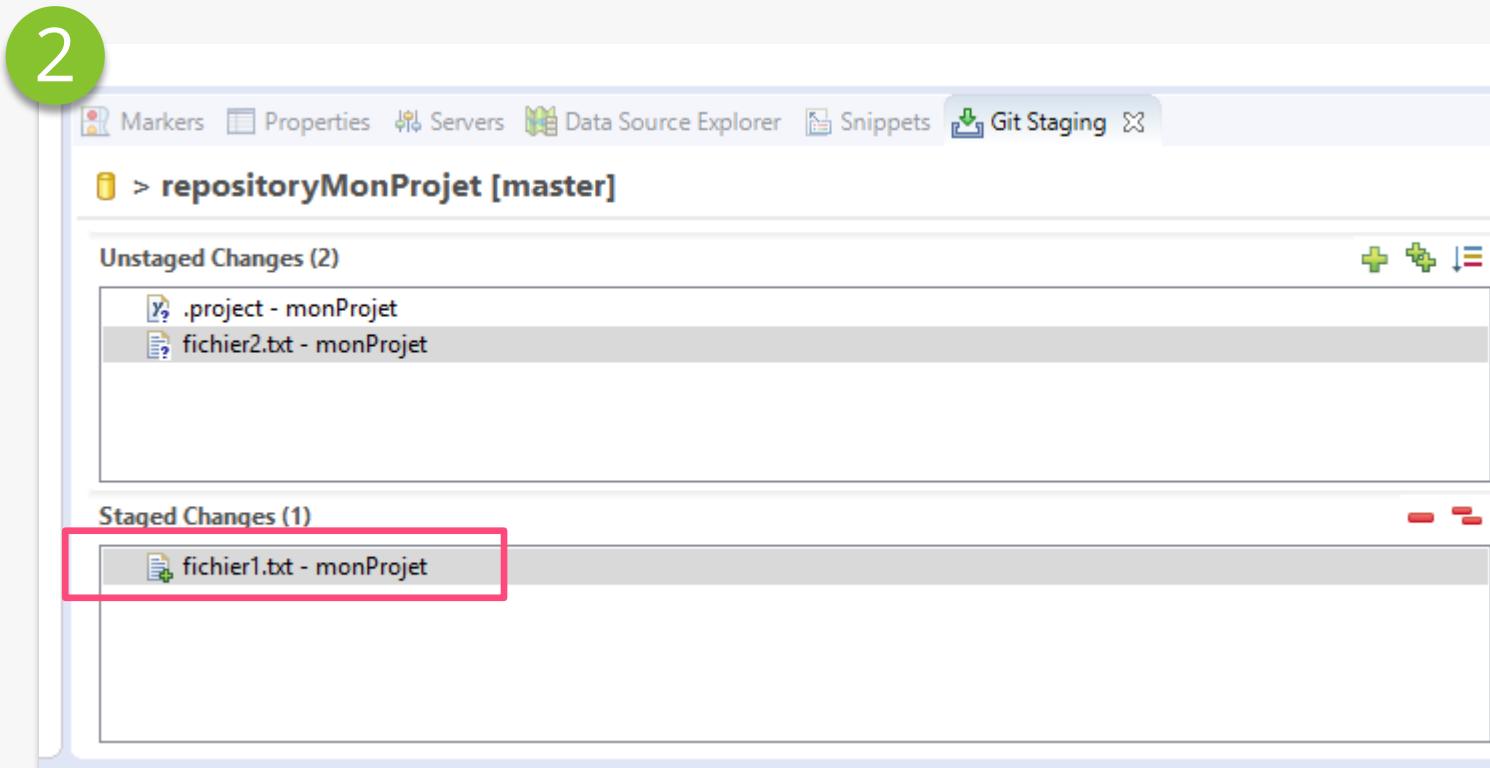


# Ajouter les fichiers à la zone de transit

Par le menu contextuel

- Sélectionnez le fichier *fichier1.txt*
- Dans le menu contextuel cliquez sur :  
**Team > Add to index**

→ Le fichier apparaît dans la zone de transit

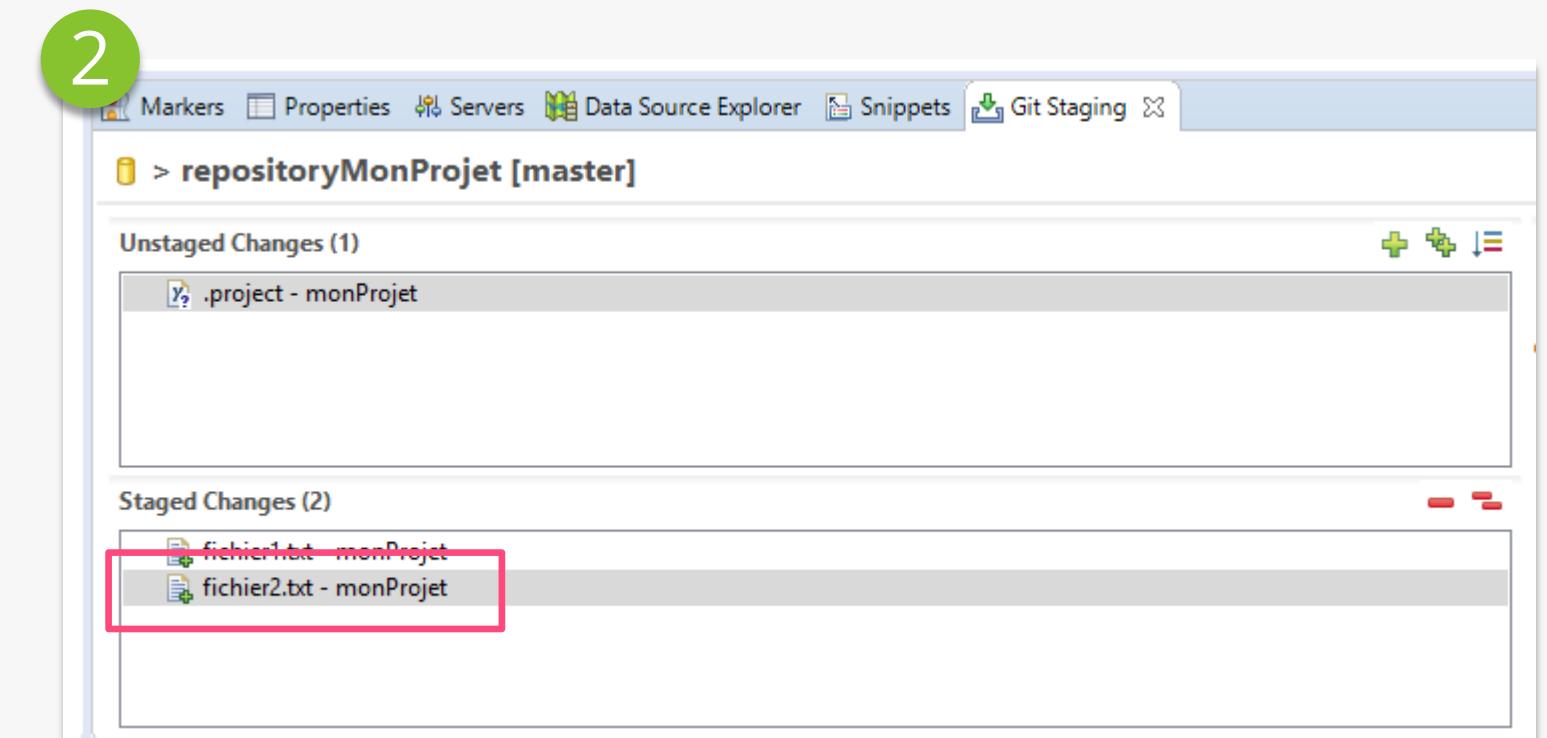
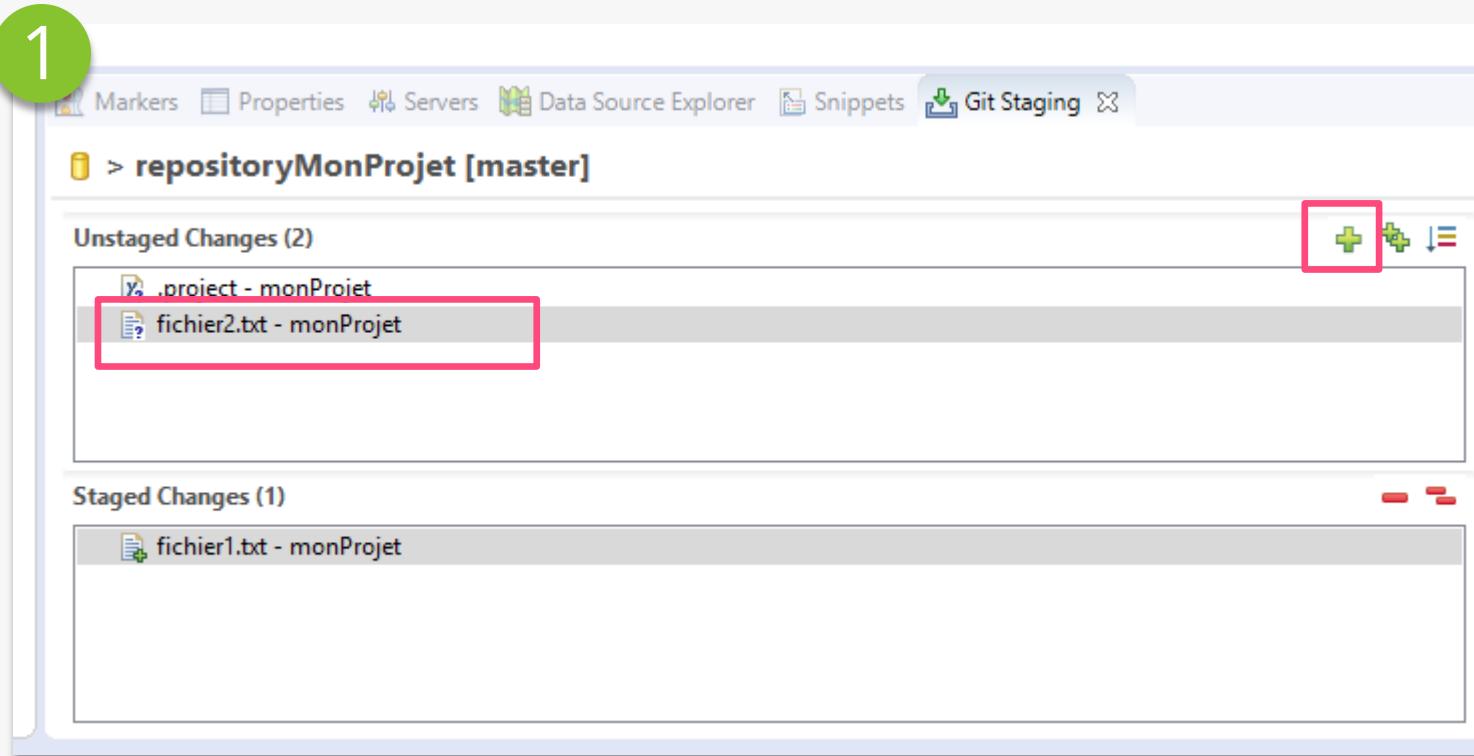


# Ajouter les fichiers à la zone de transit

Par la vue *Git Staging*

- Sélectionnez le fichier *fichier2.txt* dans la vue *Git Staging*
- Cliquez sur l'icône + (ou faites glisser le fichier vers la zone de transit)

→ Le fichier apparaît dans la zone de transit



Les fichiers sont désormais *indexés* et vont pouvoir être proposés au prochain *Commit*

# Ajouter les fichiers au dépôt local

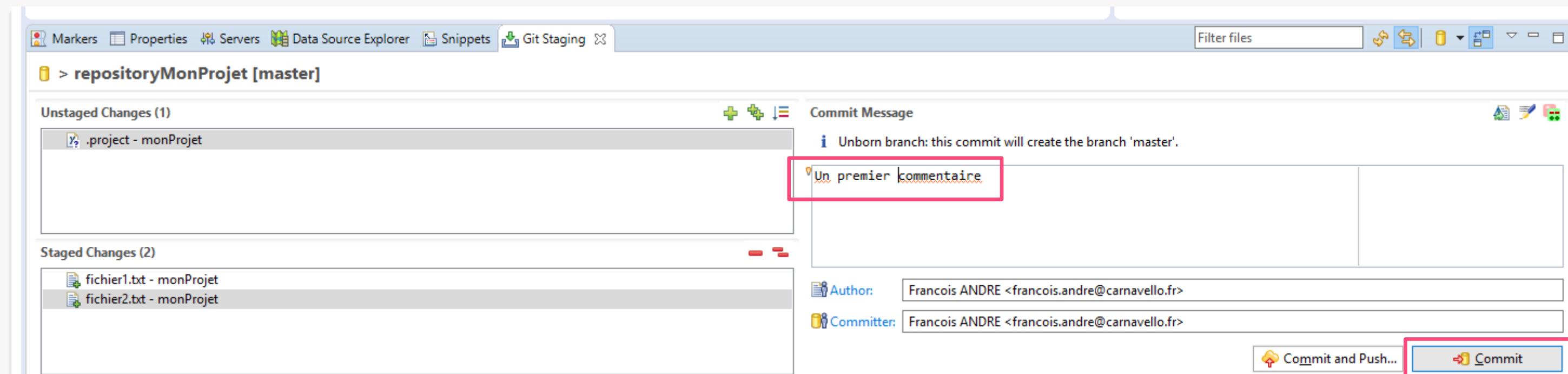
Par la vue *Git Staging*

L'ajout de fichiers au dépôt local s'appelle un **commit**.

Lors du premier commit au sein d'Eclipse vous allez devoir indiquer votre nom. Vous pouvez également le modifier à chaque commit.

A chaque commit, vous devrez renseigner un commentaire pour accompagner votre dépôt.

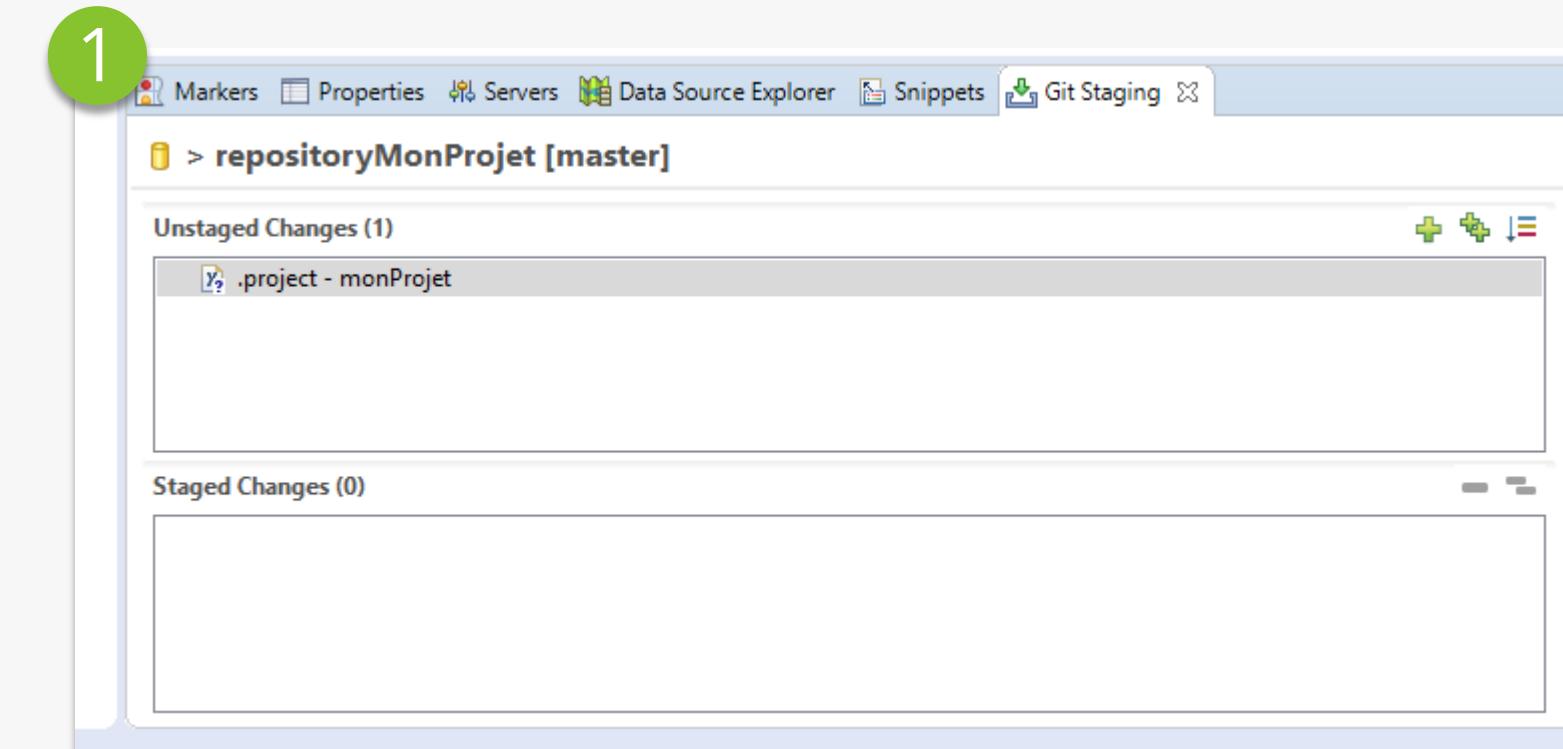
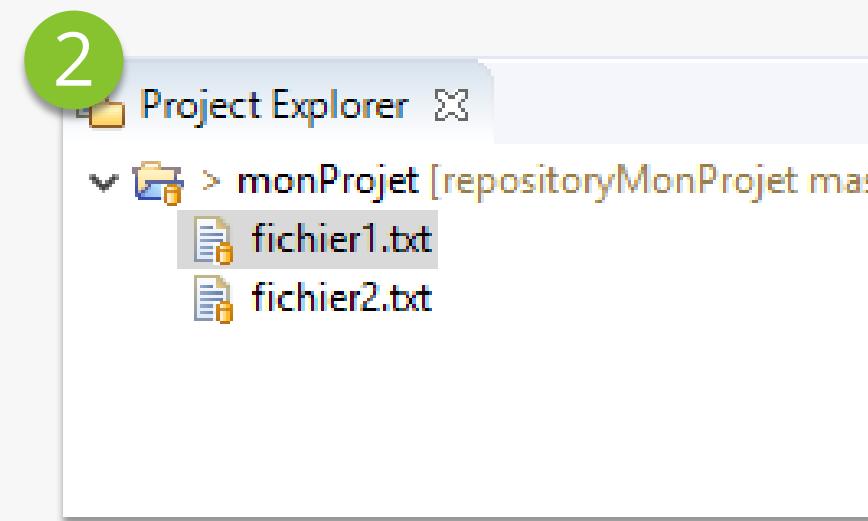
Renseignez un commentaire et cliquez sur *Commit*



# Ajouter les fichiers au dépôt local

Par la vue Git Staging

1. Les fichiers fichier1.txt et fichier2.txt disparaissent de la vue Git Staging
2. Dans l'explorateur les fichiers apparaissent sans ? ce qui signifie qu'ils sont pris en charge par Git.





# Ajouter les fichiers au dépôt local

*Par la vue Git Staging*

## MODE COMMANDE

```
$ git add monfichier  
$ git commit  
$ git commit -m "mon commentaire" (permet d indiquer directement un commentaire)  
$ git commit -a -m "mon commentaire" (permet de passer la phase d indexation et pr
```



# Ignorer des fichiers

---

Certains fichiers ne doivent pas être commités :

- Fichier spécifiques à un IDE ou à un poste de développement
- Fichiers générés (.class, .jar)
- ...

Nous allons utiliser le **mécanisme standard de Git** qui repose sur un fichier nommé `.gitignore` :

- Créez un fichier `.gitignore` à la racine du projet
- Indiquez comme contenu `.project`

Cela signifie le fichier `.project` (fichier spécifique à Eclipse) va être ignoré par Git.

- Sauvegardez le fichier
- La vue Git Staging n'affiche plus le fichier `.project`. Par contre le fichier `.gitignore` est apparu.



# Ignorer des fichiers

---

## Remarques

---

- Le fichier `.gitignore` est un fichier ordinaire : il doit être ajouté au dépôt et commité.

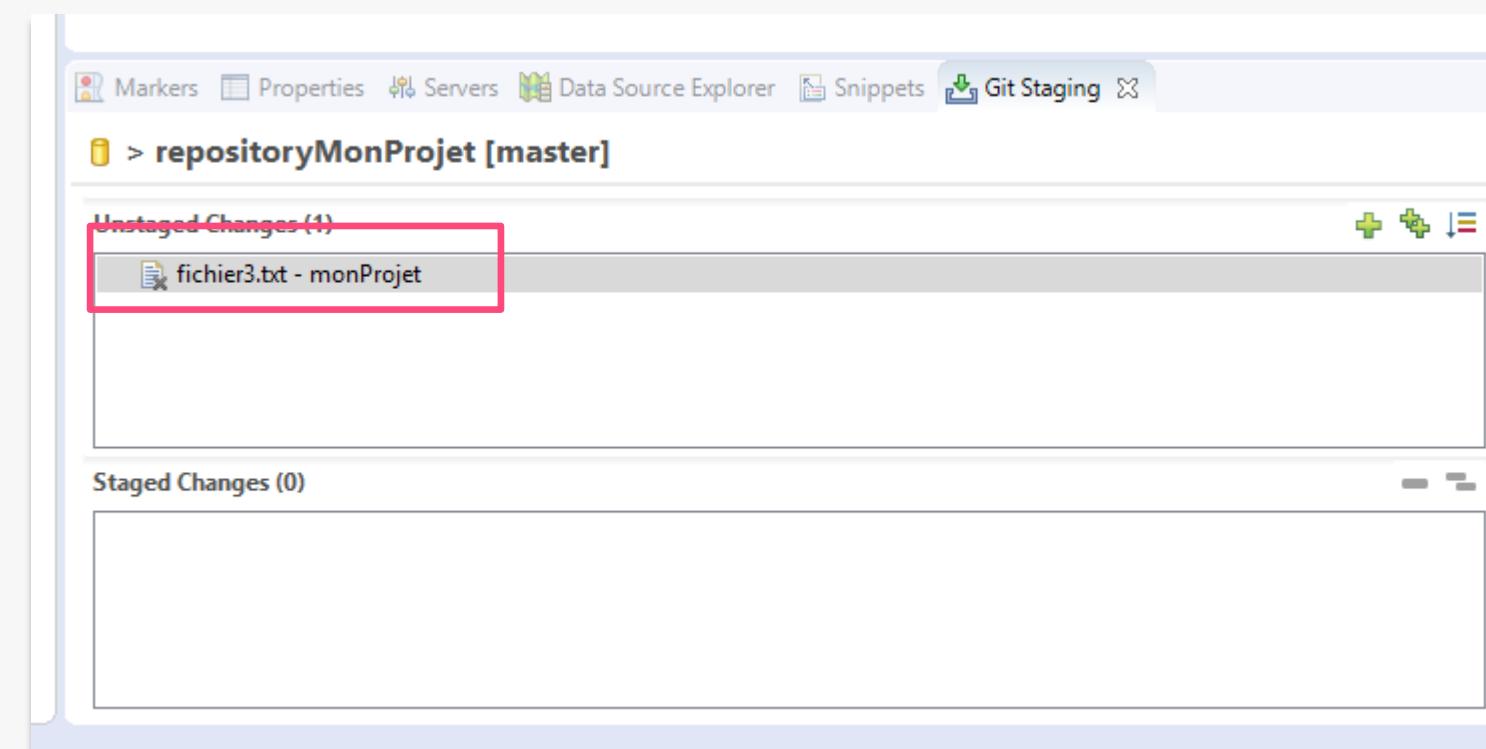
## Ajoutez le fichier `.gitignore` au dépôt local

- La syntaxe du fichier `.gitignore` est détaillée ici : <http://git-scm.com/docs/gitignore>
- Des exemples de fichiers `.gitignore` adaptés à chaque language/framework sont disponibles : <https://github.com/github/gitignore>

# Supprimer un fichier

Lorsque que l'on supprime un fichier - via l'IDE par exemple - il faut répercuter cette suppression au niveau du dépôt local **si celui-ci y était déjà présent**. Cette opération est similaire à un commit.

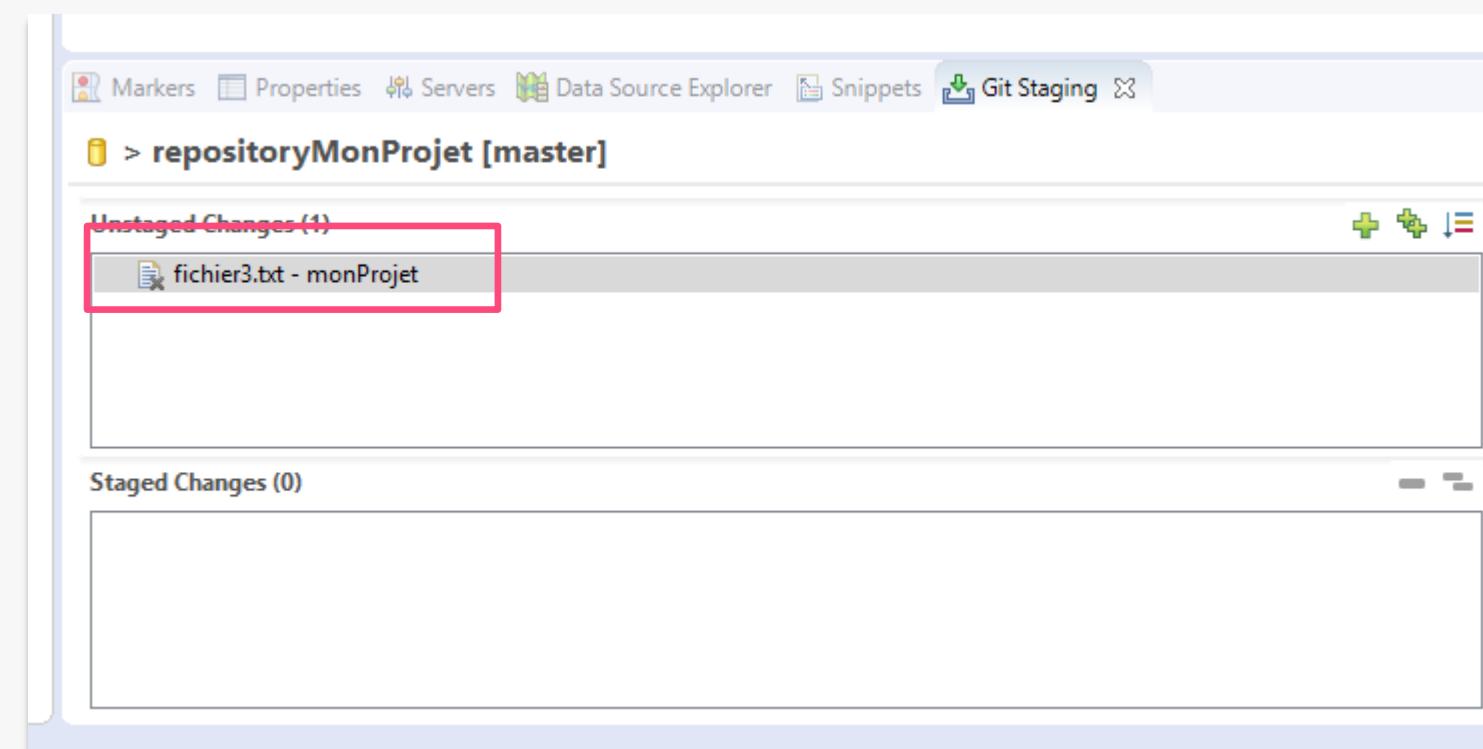
- Créez un fichier *fichier3.txt* avec comme contenu *Contenu du fichier 3*. Ajoutez ce fichier au dépôt local (ajoutez le à l'index, puis commitez-le).
- Supprimez le fichier dans l'arborescence du projet.
- Dans la vue Git Staging, le fichier *fichier3.txt* apparaît avec l'icône **x** afin de souligner qu'il s'agit de la suppression du fichier



# Supprimer un fichier

Lorsque que l'on supprime un fichier - via l'IDE par exemple - il faut répercuter cette suppression au niveau du dépôt local **si celui-ci y était déjà présent**. Cette opération est similaire à un commit.

- Créez un fichier *fichier3.txt* avec comme contenu *Contenu du fichier 3*. Ajoutez ce fichier au dépôt local (ajoutez le à l'index, puis commitez-le).
- Supprimez le fichier dans l'arborescence du projet.
- Dans la vue Git Staging, le fichier *fichier3.txt* apparaît avec l'icône **x** afin de souligner qu'il s'agit de la suppression du fichier





# Supprimer un fichier

---

- Faites glisser le fichier dans la zone de transit et commitez.

## MODE COMMANDE

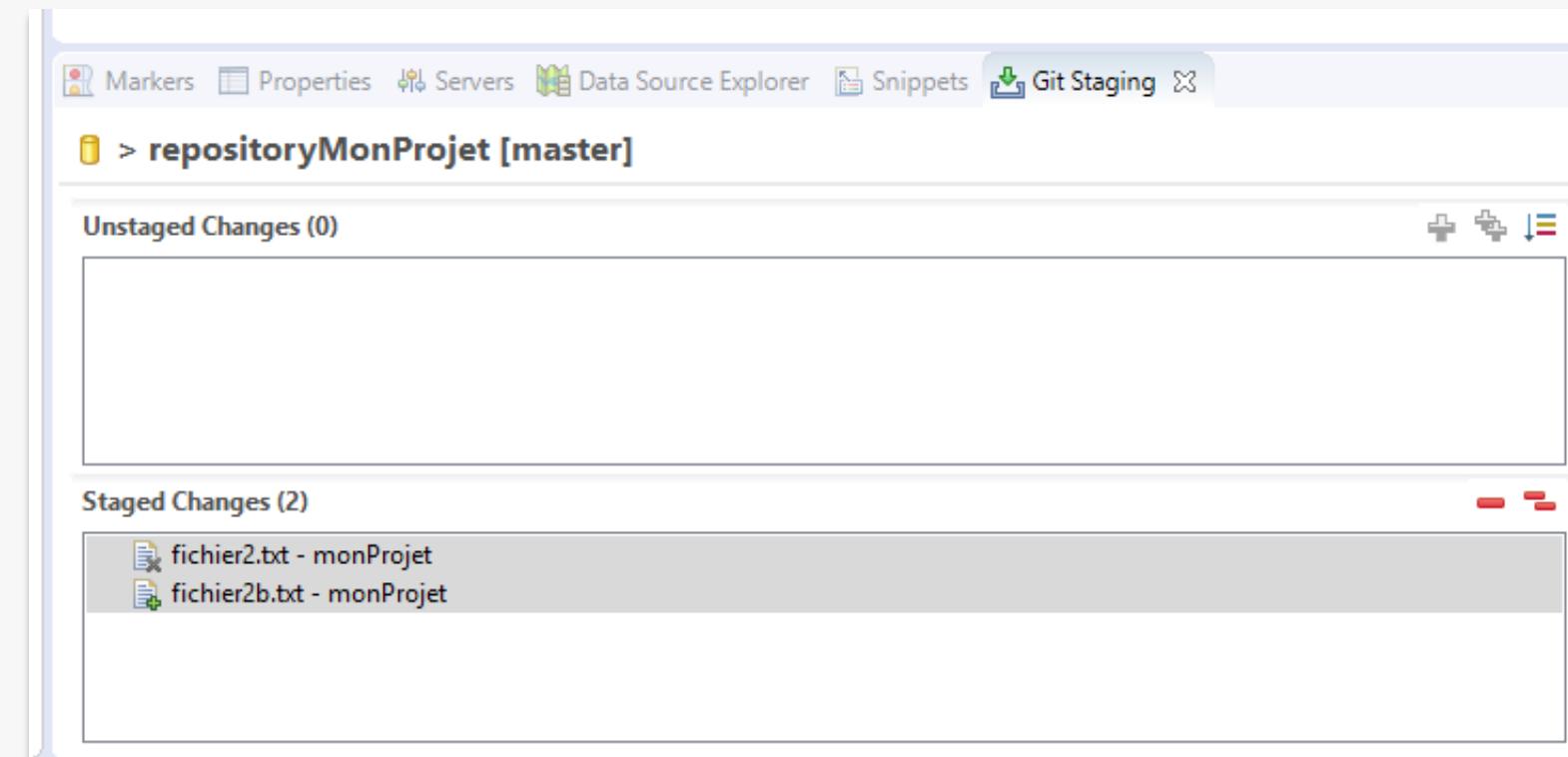
```
$ git rm monfichier  
$ git rm --cached monfichier  
(dans ce cas le fichier est supprimé du dépôt mais pas de l espace de travail)
```

# Renommer un fichier

Répercuter le changement de nom d'un fichier présent au dépôt est une action similaire à un ajout et à une suppression. La différence est que les fichiers sont déjà en zone de transit.

Renommez le fichier *fichier2.txt* en *fichier2b.txt* (via le menu contextuel)

La vue Git Staging affiche dans la zone de transit le fichier supprimé et le fichier ajouté



Commitez

François ANDRE

# Consulter l'historique

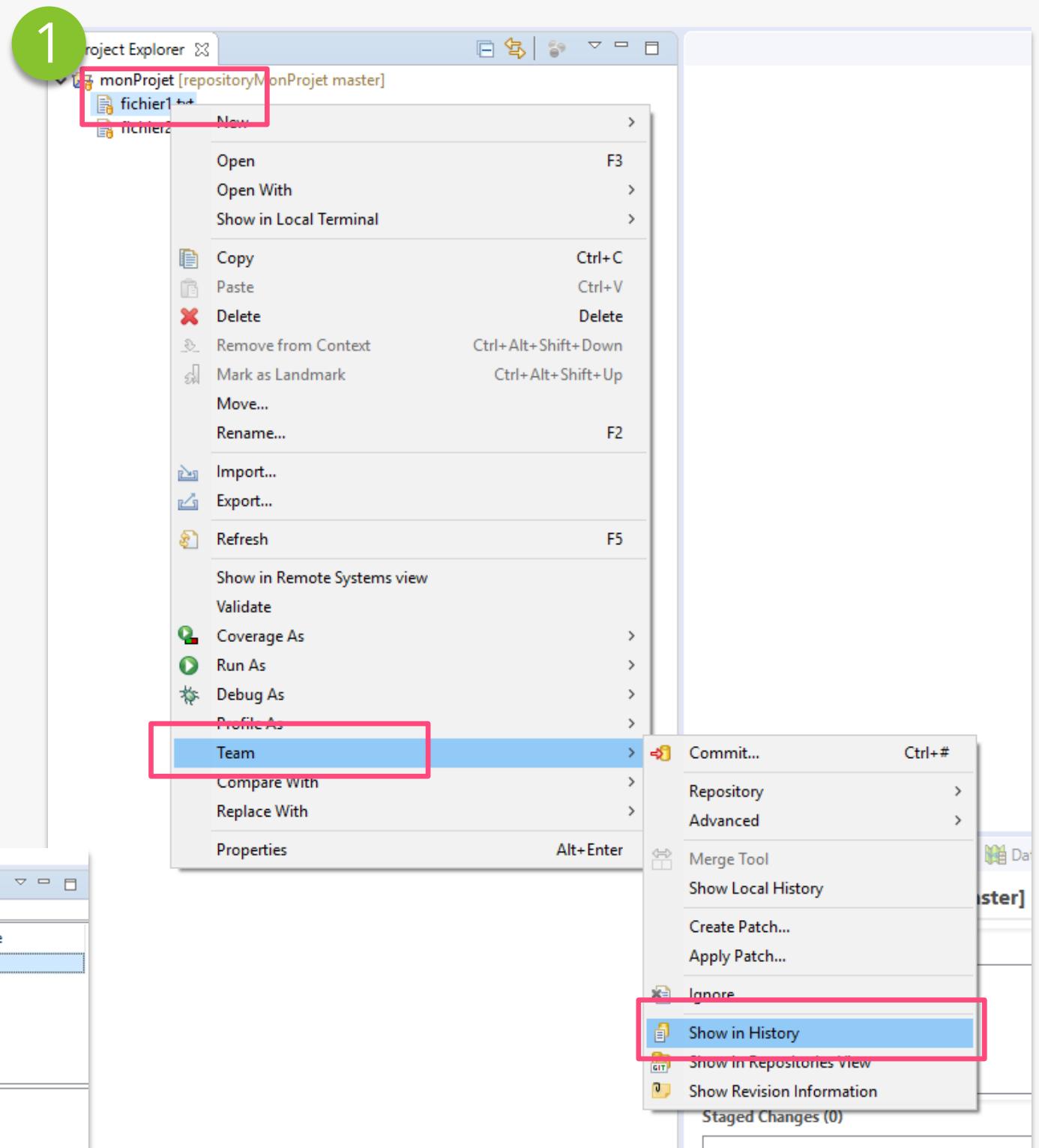
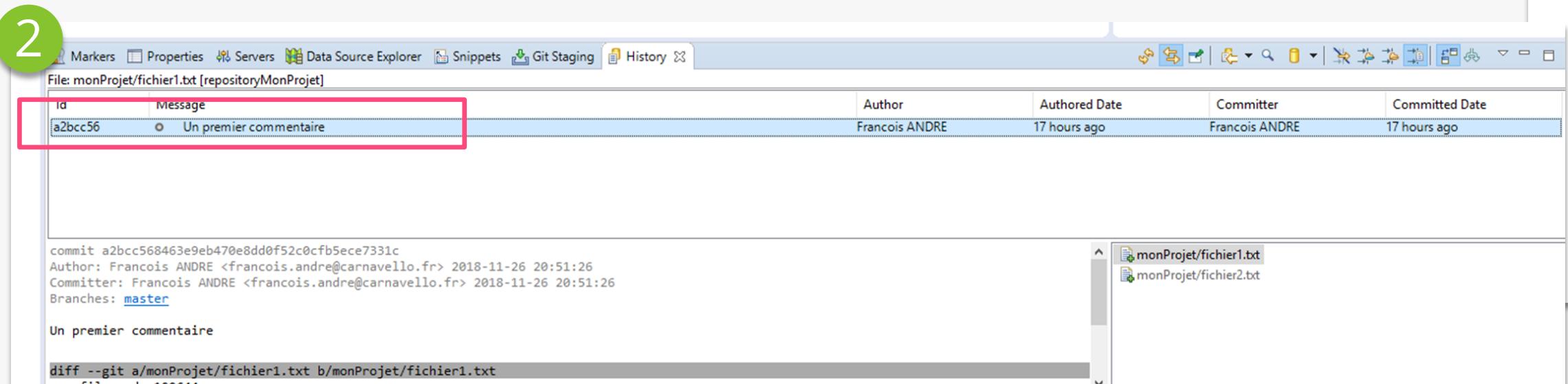
## Historique d'un fichier

L'onglet **History** permet de consulter l'historique d'un fichier :

- ses différents *commits* (avec leur messages)
- ses différentes branches

Les commits sont identifiées par un identifiant unique

Dans le menu contextuel d'un fichier, cliquer sur Team > Show in History



# Consulter l'historique

## Historique d'un projet

L'onglet **History** permet également de consulter l'historique d'un projet :

- ses différents *commits* (avec leur messages)
- ses différentes branches

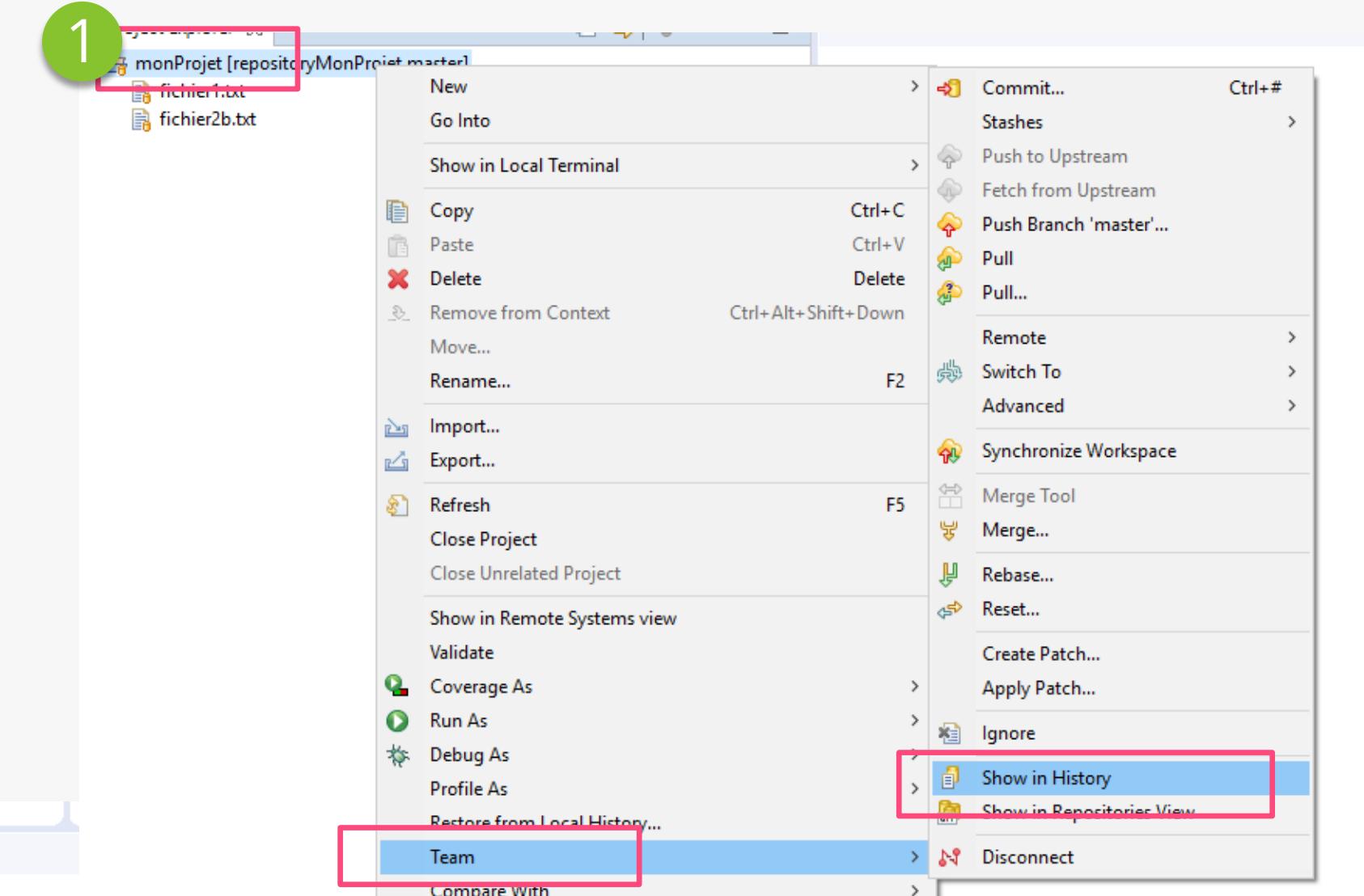
Dans le menu contextuel du projet, cliquer sur Team > Show in History

2

Id	Message	Author	Authored Date	Committer	Committed Date
9575bb9	master (HEAD) Renommage de fichier2.txt	Francois ANDRE	2 days ago	Francois ANDRE	2 days ago
fa96b38	Suppression du fichier3.txt	Francois ANDRE	2 days ago	Francois ANDRE	2 days ago
9f42a74	Ajout du fichier3.txt	Francois ANDRE	2 days ago	Francois ANDRE	2 days ago
ee62245	Ajout du .gitignore	Francois ANDRE	2 days ago	Francois ANDRE	2 days ago
a2bcc56	Un premier commentaire	Francois ANDRE	2 days ago	Francois ANDRE	2 days ago

```
commit a2bcc568463e9eb470e8dd0f52c0cfb5ece7331c
Author: Francois ANDRE <francois.andre@carnavello.fr> 2018-11-26 20:51:26
Committer: Francois ANDRE <francois.andre@carnavello.fr> 2018-11-26 20:51:26
Child: ee622459d95168ba6ffb4e2f3b4a9ad92177e6e8 (Ajout du .gitignore)
Branches: master

Un premier commentaire
```

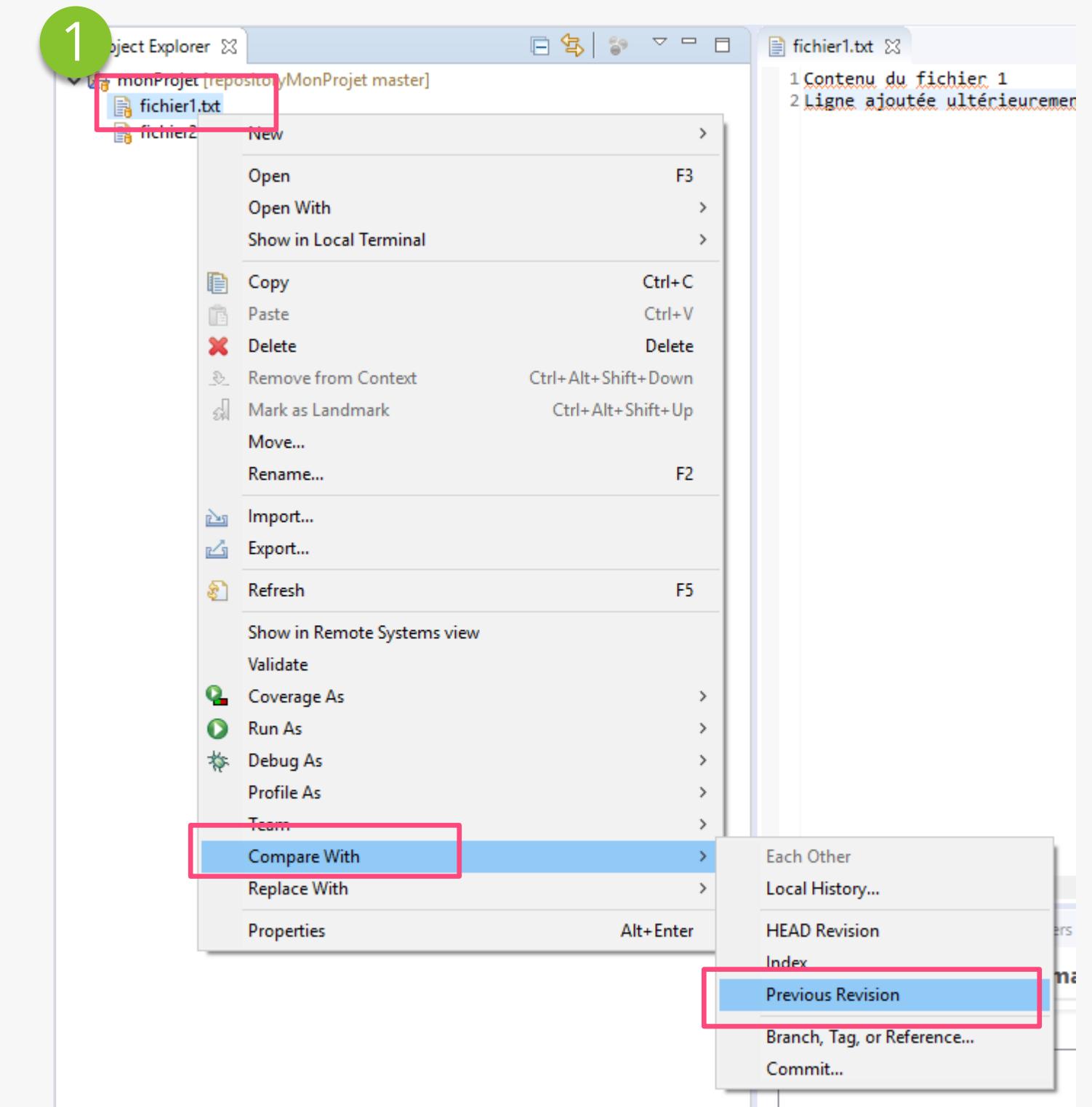


# Consulter l'historique

## *Différence entre les versions*

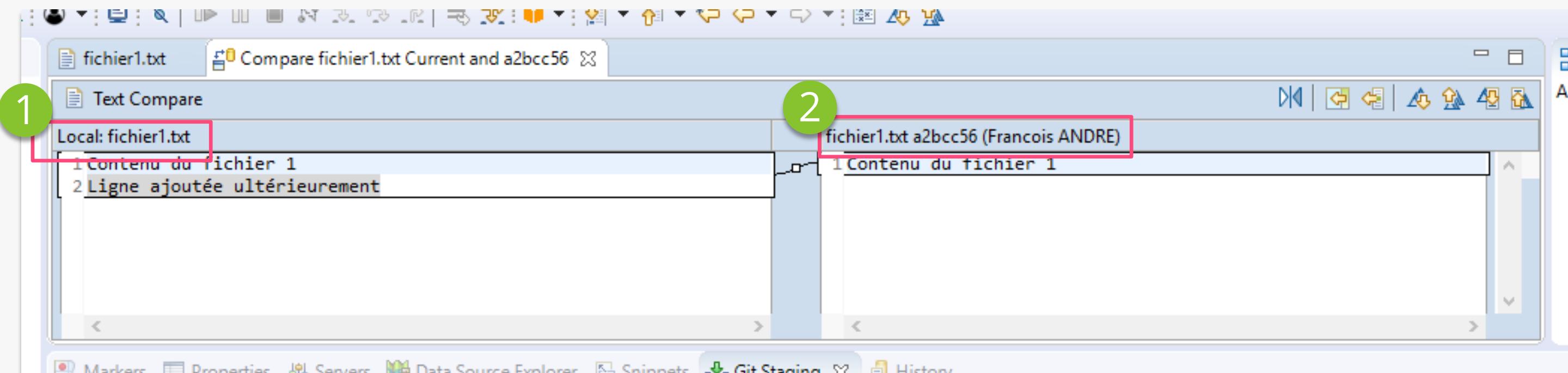
- Dans le fichier *fichier1.txt* ajoutez une ligne ayant comme contenu *Ligne ajoutée ultérieurement*
- Commitez le fichier

Dans le menu contextuel du fichier *fichier1.txt*, cliquer sur *Compare with > Previous revision*



# Consulter l'historique

*Différence entre les versions*



1. Version courante
2. Précédent commit

**La comparaison de version permet de comparer un fichier avec n'importe quelle de ses versions.**



# Consulter l'historique

---

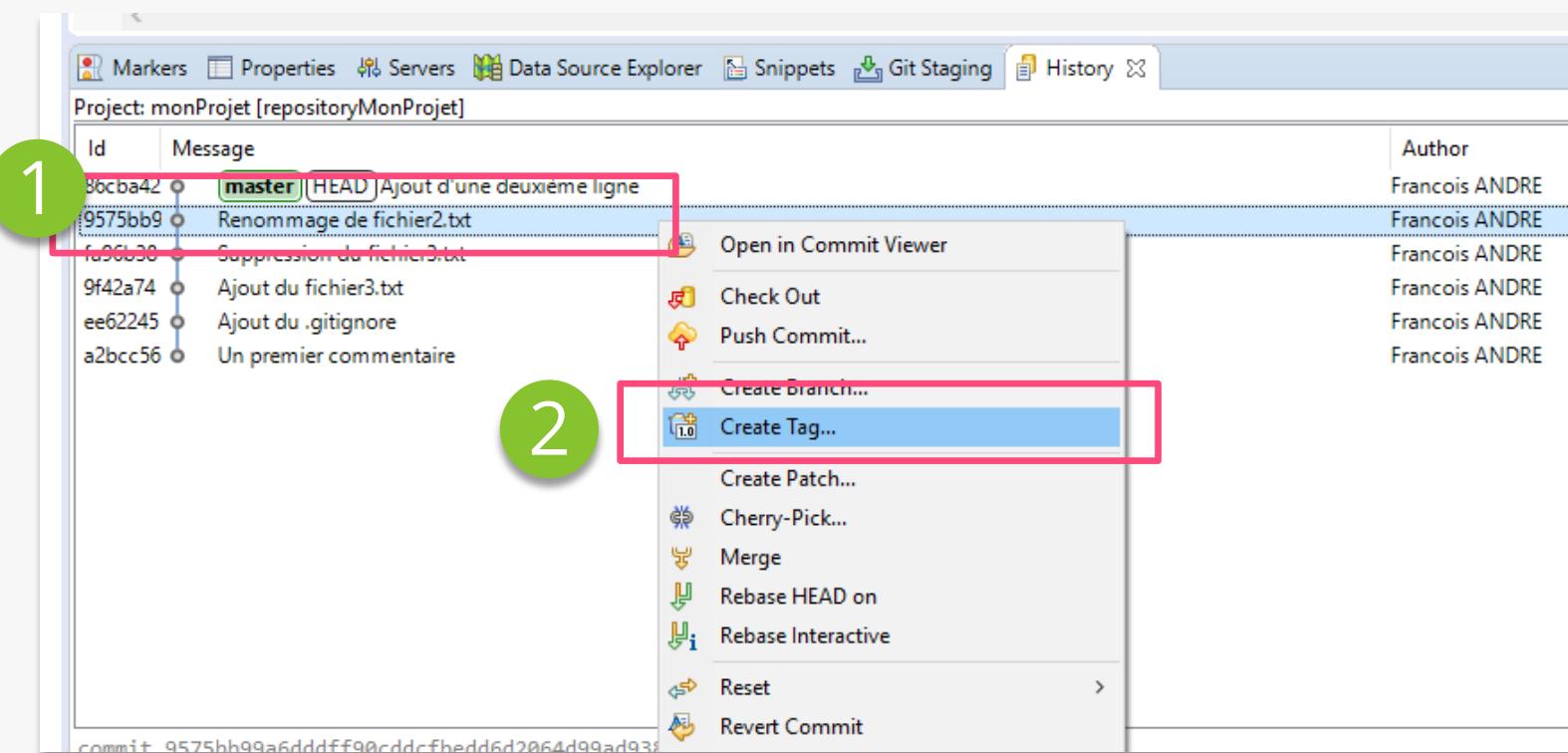
## MODE COMMANDE

```
$ git log  
$ git log --pretty=oneline  
$ git log --pretty=short  
$ git log --since="2 weeks ago"
```

# Ajouter des Etiquettes (Tag)

Une étiquette permet de **donner un nom sur un instantané** afin d'être facilement identifiable par la suite (pour faire une branche, créer une version,...). Par exemple, cela permet d'associer un numéro de version à un état du projet.

1. Dans l'onglet History, sélectionnez un ancien commit.
2. Dans le menu contextuel, sélectionnez *Create Tag...*



# Ajouter des Etiquettes (Tag)

1. Indiquez un nom de Tag
2. Indiquez un commentaire
3. Cliquez sur *Create Tag*

Le tag apparaît alors dans l'historique

The screenshot shows two windows. The top window is titled 'Create New Tag' and contains fields for 'Tag name:' (V1.0) and 'Tag message:' (Version 1.0). A green circle labeled '1' points to the 'Tag name:' field, and another green circle labeled '2' points to the 'Tag message:' field. A third green circle labeled '3' points to the 'Create Tag' button at the bottom right. The bottom window is a 'History' view showing a list of commits. One commit, with ID 9575bb9, has its message 'Renommage de fichier2.txt' highlighted with a red box. A yellow dashed arrow points from this highlighted message up to the 'Tag message:' field in the dialog window above.

ID	Message	Author	Authored Date	Committer	Committed Date
86cba42	[master] HEAD Ajout d'une deuxième ligne	Francois ANDRE	3 days ago	Francois ANDRE	3 days ago
9575bb9	V1.0 Renommage de fichier2.txt	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago
fa96b38	Suppression du fichier3.txt	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago
9f42a74	Ajout du fichier3.txt	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago
ee62245	Ajout du .gitignore	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago
a2bcc56	Un premier commentaire	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago

Note pour plus tard: un tag doit être pushé...

François ANDRE



# Ajouter des Etiquettes (Tag)

---

## MODE COMMANDE

---

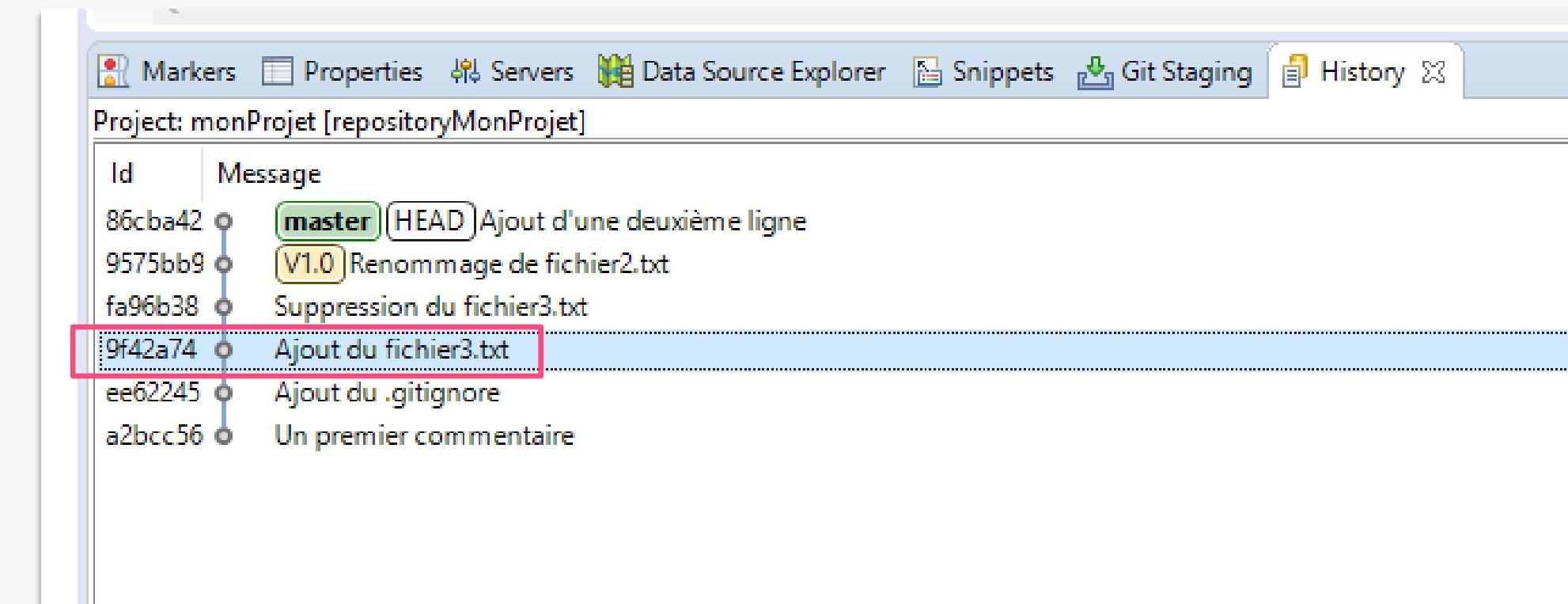
```
$ git tag -a v1.0 (ajout de l etiquette)
$ git tag -a v1.0 -m 'mon commentaire'
$ git tag -d v1.0 (suppression de l etiquette)
```

# Remplacer par une version antérieure

La commande *git checkout* permet de naviguer entre les branches et les commits (cf. infra).

Elle peut toutefois être utilisée pour restaurer un fichier dans une version plus ancienne. Par exemple *fichier3.txt (actuellement supprimé)*.

- Identifiez l'identifiant du commit contenant la version de *fichier3.txt* souhaitée via la vue History. Dans notre cas **9f42a74**



# Remplacer par une version antérieure

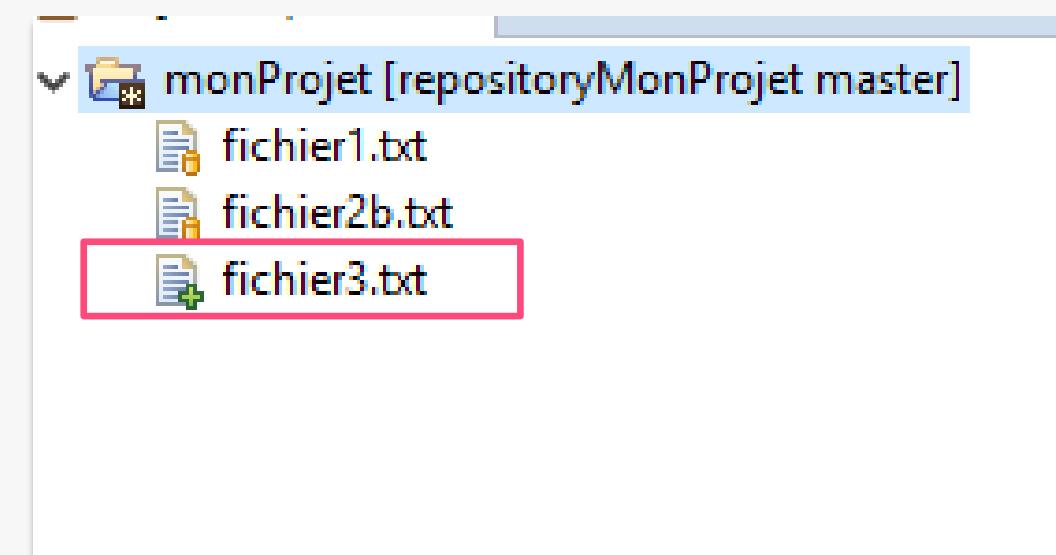
Ouvrez un terminal à la racine du projet.

Lancez la commande `git checkout 9f42a74 fichier3.txt` dans l'onglet ouvert

```
François ANDRE@UMSPW108 MINGW64 ~/git/repositoryMonProjet/monProjet (master)
$ git checkout 9f42a74 fichier3.txt
```

Rafraîchissez votre espace de travail dans Eclipse.

Le fichier a été restauré et est présent dans l'espace de travail comme un nouveau fichier.

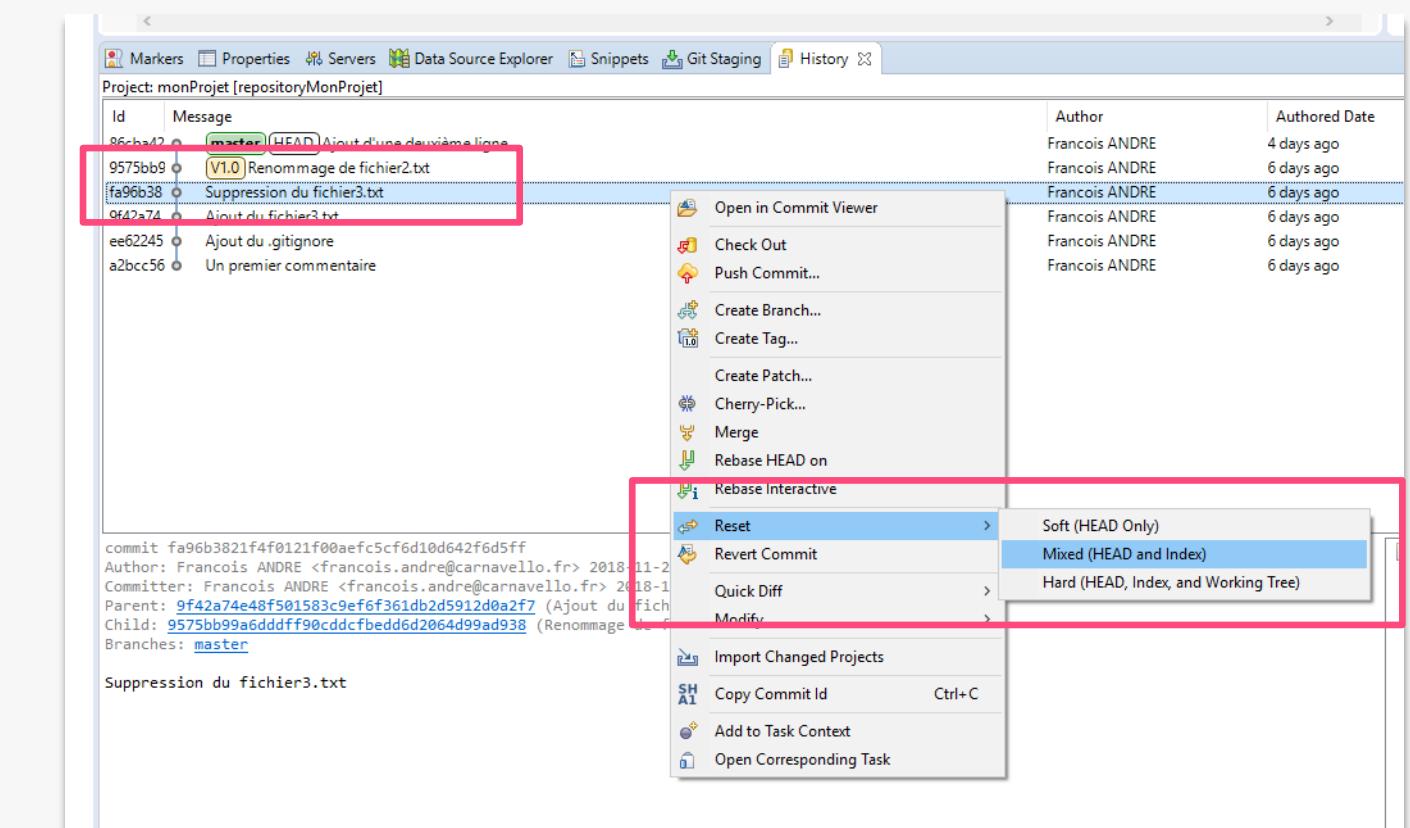


# Repartir d'un commit antérieur

Il peut être utile - mais **dangereux** - de repartir d'un *commit* plus ancien et de supprimer les *commits* survenus depuis.

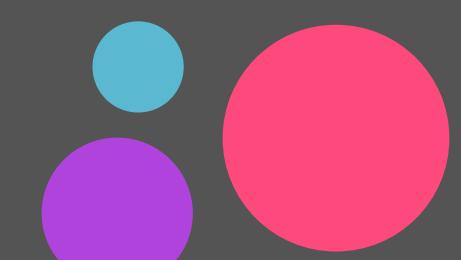
Cette opération peut être effectuée à partir d'un *commit* affiché dans l'onglet History. Il est peut-être plus **prudent** de créer une branche à partir du *commit* identifié.

Les différents types de *Reset* permettent de conserver ou supprimer les modifications depuis le *commit*.





# Les branches - Partie 1 : La théorie





# Pourquoi avoir besoin de branches ?

---

Il peut être nécessaire de travailler simultanément sur plusieurs versions du code:

- Version en production
- Version en développement
- ...

Le concept de **branche** répond à ce besoin et les différents SCM le mettent en œuvre.

Toutefois, généralement, l'implémentation proposée fait que le concept est rarement utilisé.

*Exemple:* En SVN, les branches sont gérées sur le serveur et le temps de création des branches puis de bascule d'une branche à l'autre est rédhibitoire.

**Avec Git, les branches sont très simples à gérer et le passage entre deux branches est quasi instantané.**

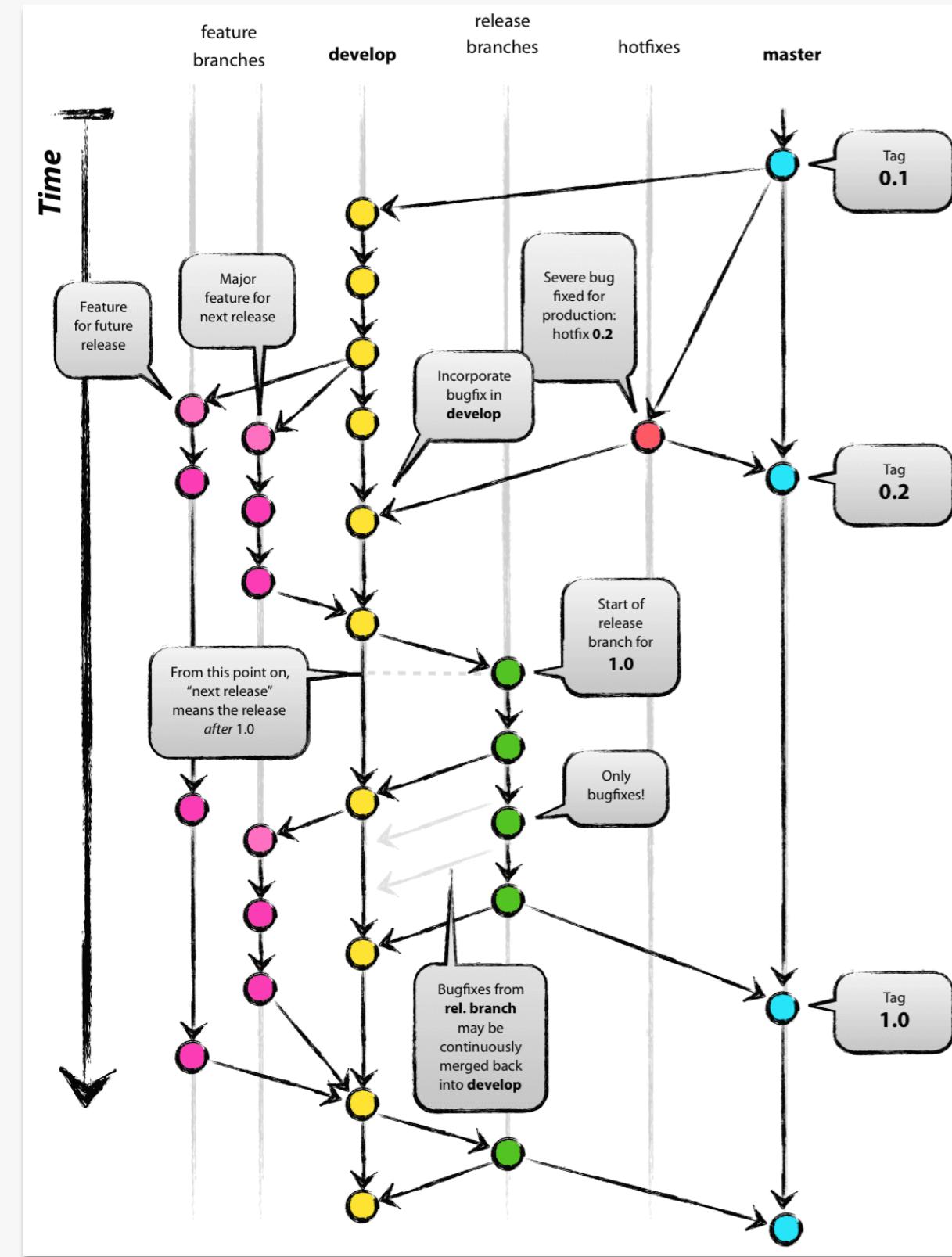


# Gitflow

---

- En 2010, Vincent Driessen publie un article intitulé *A successful Git branching model* (<http://nvie.com/posts/a-successful-git-branching-model/>) proposant une méthodologie efficace pour gérer les projets avec Git grâce aux branches.
- Cette méthodologie a été rapidement adoptée par beaucoup de grands acteurs du développement, notamment dans le cadre de méthodologie agile.

# Gitflow





# Gitflow

---

*Principaux points*

---

- **Branches pérennes**

*master* : branche correspondant au code en production

*develop* : branche correspondant au code en développement

- **Branches transitoires**

Feature : pour toute nouvelle fonctionnalité ou correction de bug

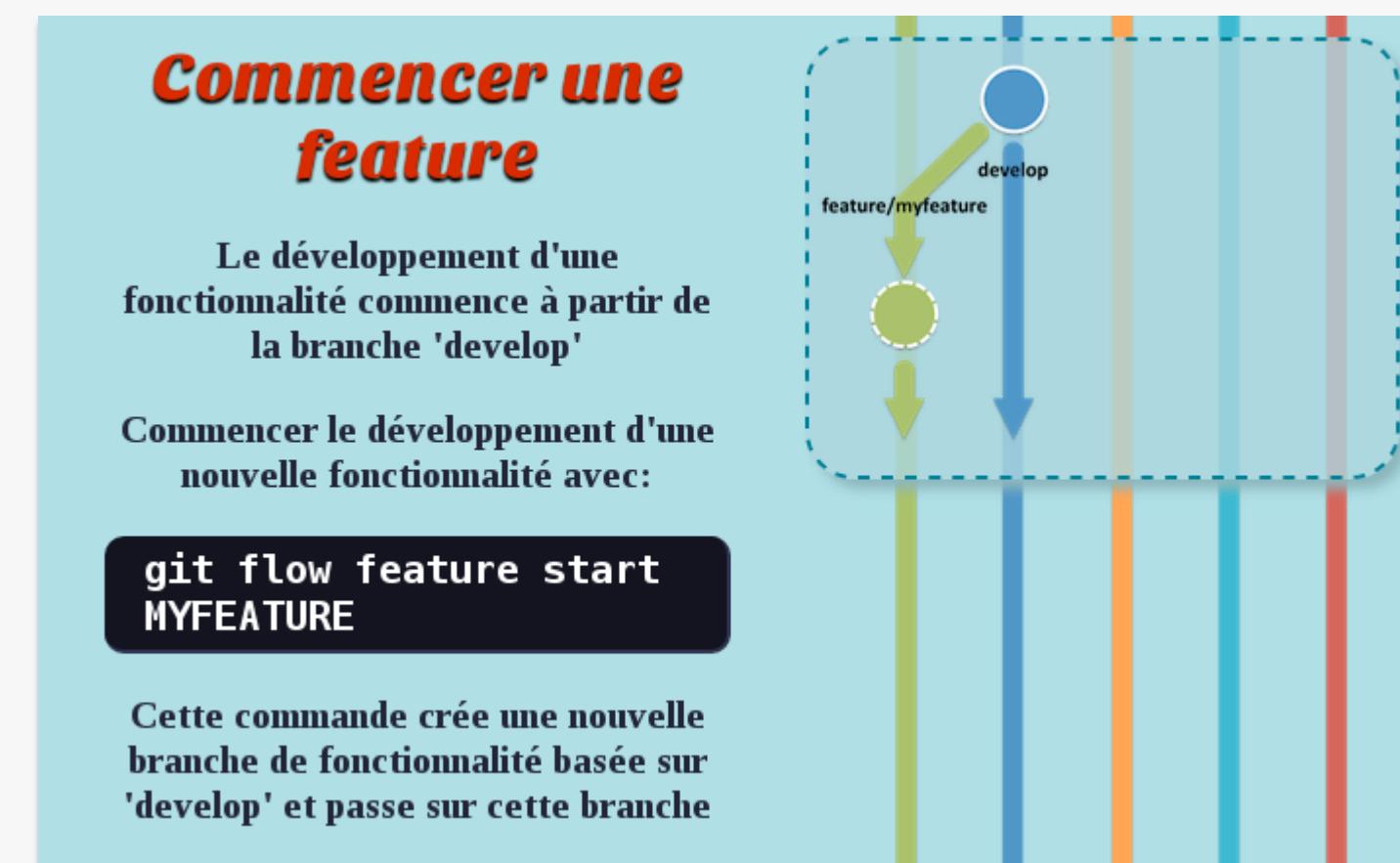
Hotfix: correction de bug en production

Release : livraison

# Gitflow

Remarque

Une surcouche à Git en mode commande est également disponible mettant en œuvre de manière explicite les principes de GitFlow. Toutefois, elle n'est pas indispensable.

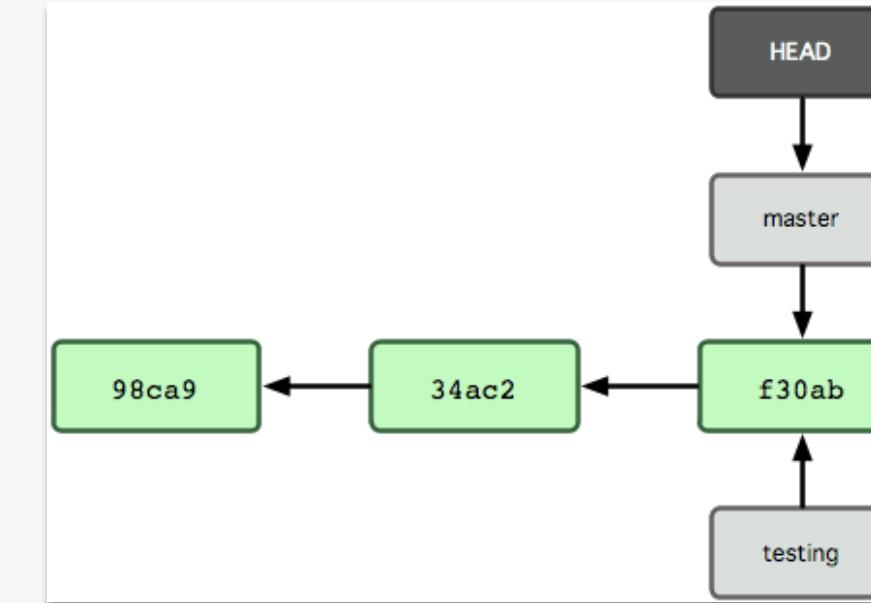


<http://danielkummer.github.io/git-flow-cheatsheet/index.fr.html>

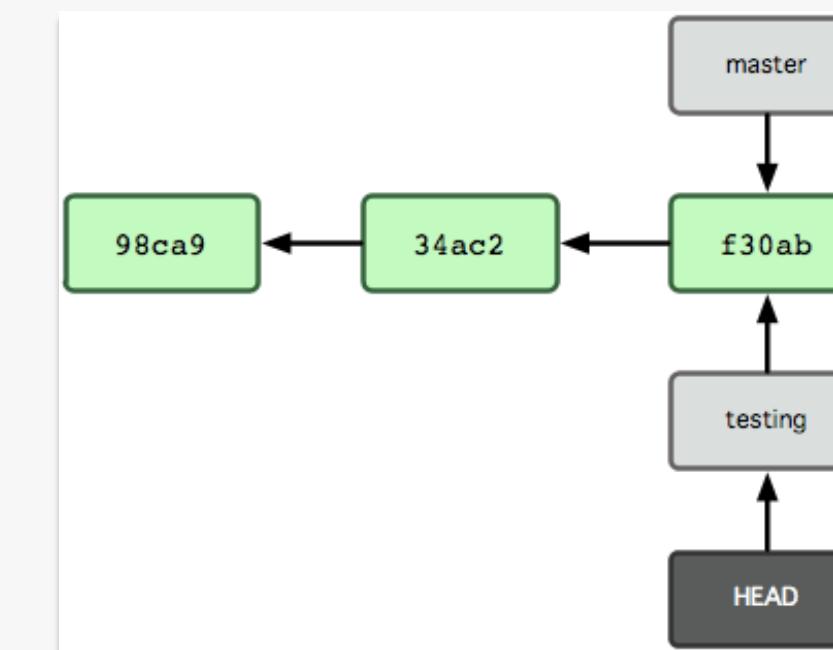
# Comment Git gère les branches ?



- Créer une nouvelle branche revient donc à ajouter un nouveau pointeur sur l'instantané courant.

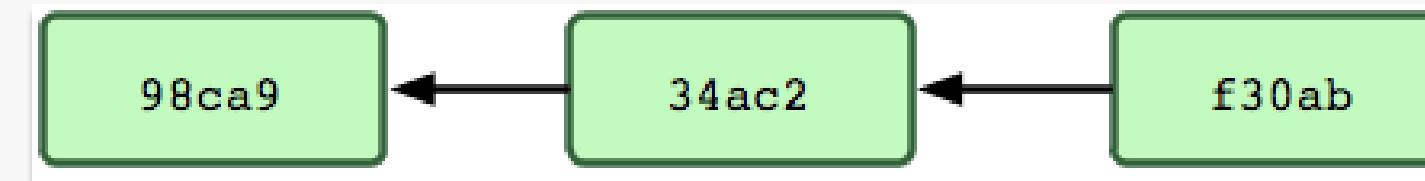


- Changer de branche consiste à basculer vers l'instantané pointé par la branche et à mettre à jour le pointeur *HEAD*.

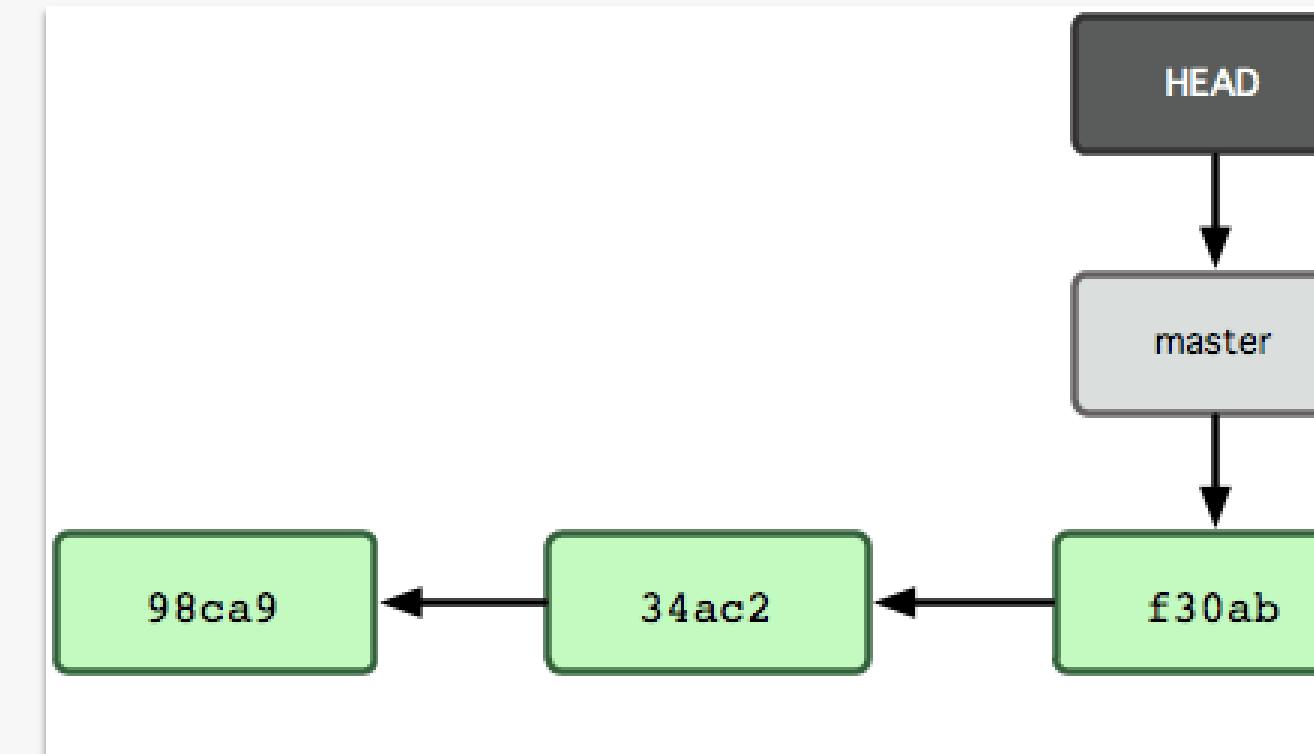


# Comment Git gère les branches ?

- Chaque *commit* entraîne la création d'un instantané identifié par un numéro unique.

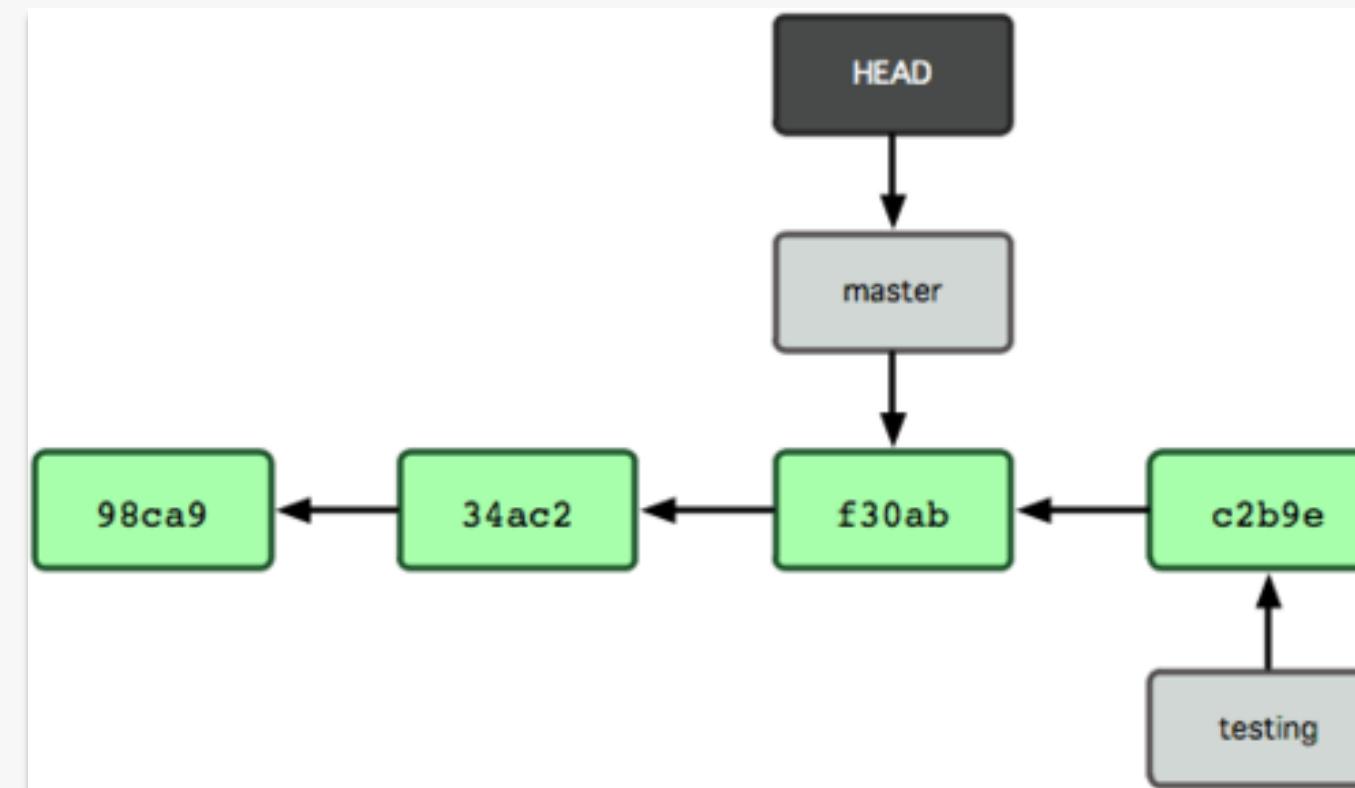


- Avec Git, **une branche est simplement un pointeur vers un instantané**
- Un pointeur spécial, *HEAD*, indique la branche courante.



# Comment Git gère les branches ?

- Lors des *commits* suivants seul le pointeur de la branche actuelle va se déplacer.





# Les fusions

---

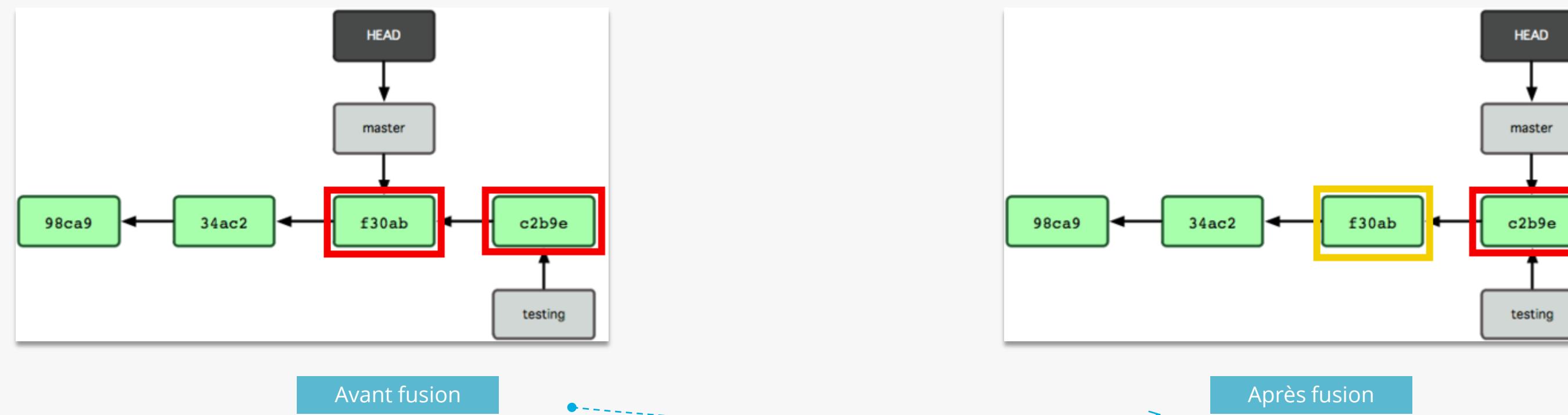
Deux cas se présentent en général

1. La branche qui va recevoir la fusion est un ancêtre de la branche à fusionner
2. La branche qui va recevoir la fusion n'est pas un ancêtre de la branche à fusionner

# Les fusions

*Cas 1: La branche qui va recevoir la fusion est un ancêtre de la branche à fusionner*

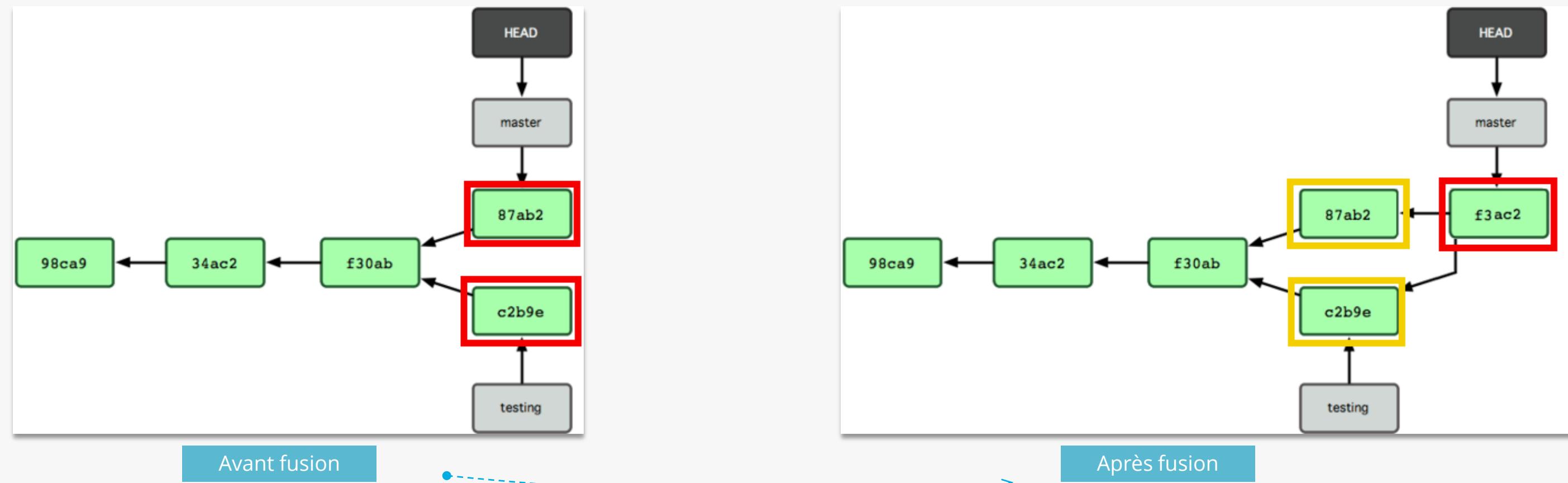
Ce cas est très simple, la fusion revient simplement à déplacer le curseur de la branche recevante sur l'instantané pointé par la branche à fusionner.



# Les fusions

*Cas 2: La branche qui va recevoir la fusion n'est pas un ancêtre de la branche à fusionner*

Ce cas est potentiellement compliqué car certains fichiers ont pu être modifiés dans les deux branches, il va falloir fusionner ces fichiers via un nouvel instantané.





# Les fusions

---

*Remarque générale*

---

**Plus on reste loin de la branche principale, plus la fusion avec celle-ci sera risquée.**

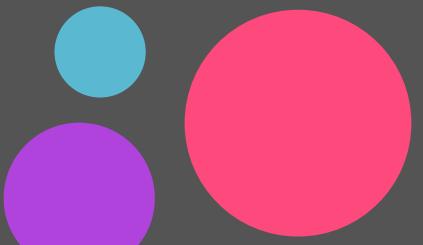
Bonnes pratiques:

- Il est prudent de travailler sur des fonctionnalités de petites tailles
- Chaque fois que l'on recommence à travailler sur un projet, il est important de commencer par se synchroniser avec la dernière version

Pour éviter les risques de fusion, certaines méthodologies – radicalement opposées à Gitflow – se basent sur un travail direct dans le *trunk* (Trunk Based Developpement)

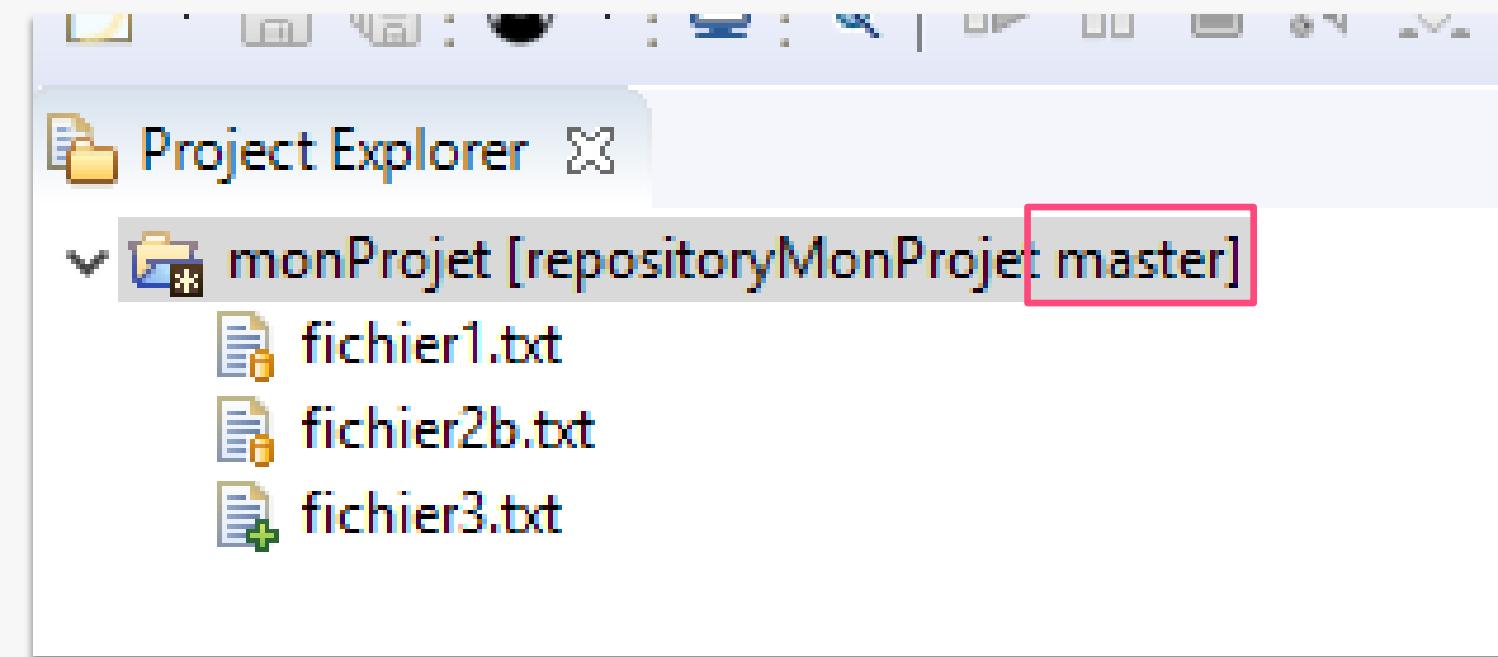


# Les branches - Partie 2 : La pratique



# Créer une branche

La branche courante est indiquée dans l'explorateur de projets au niveau du nœud projet.

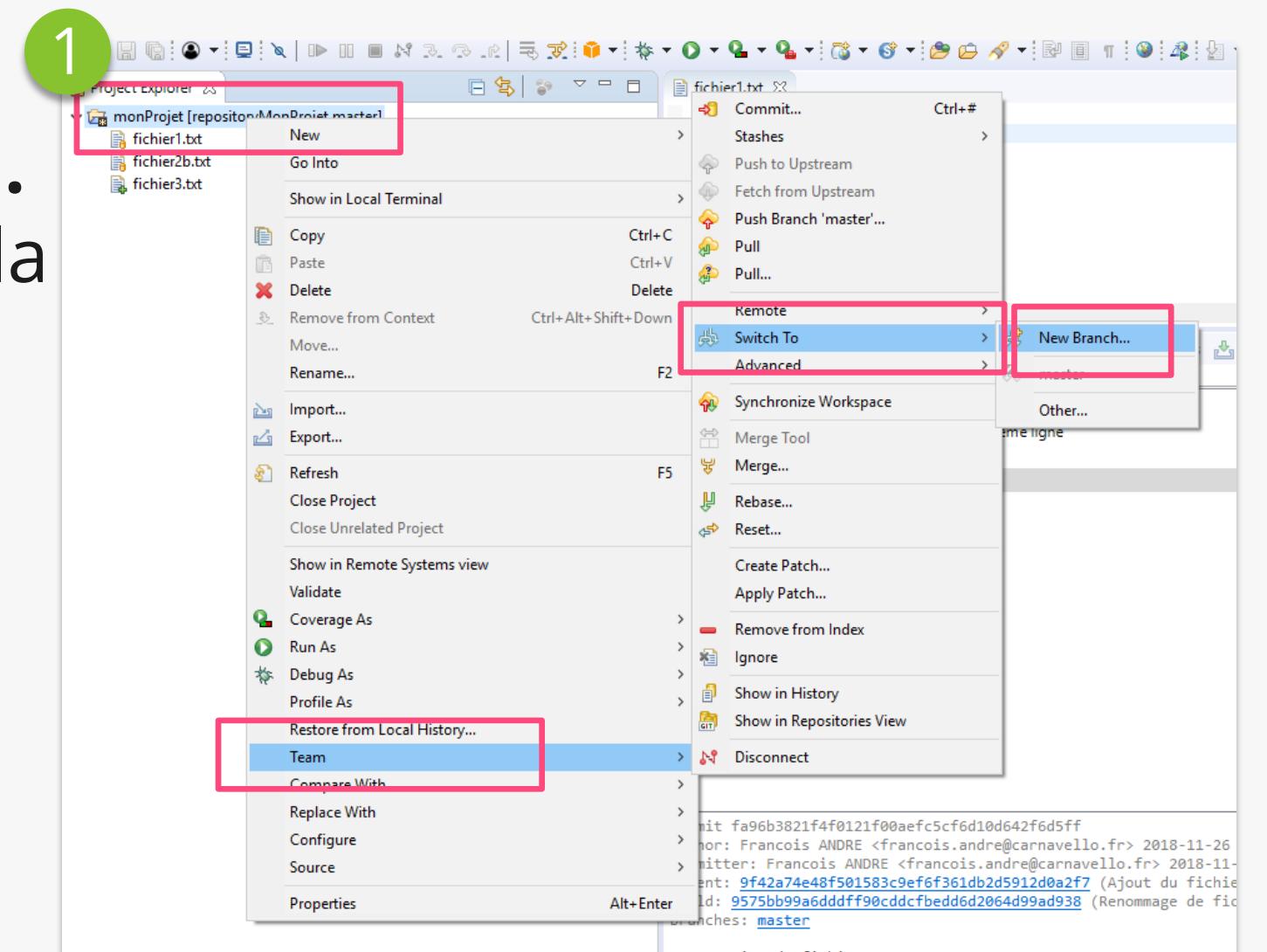
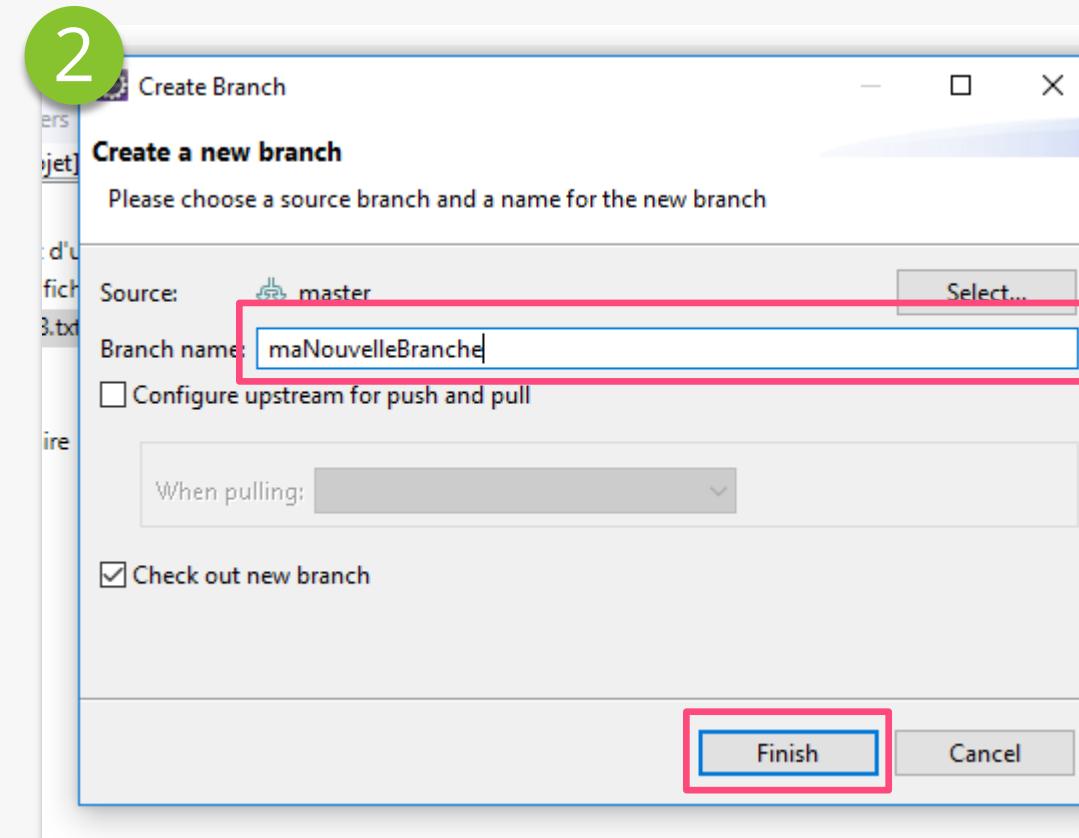


Par convention, la 1ère branche s'appelle *master*.

# Créer une branche

Pour créer une nouvelle branche et basculer dessus :

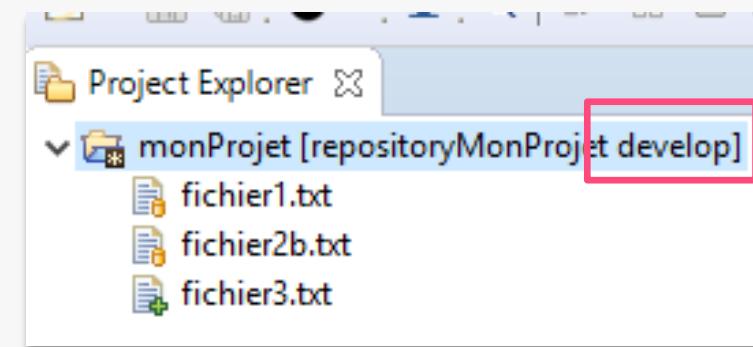
1. A partir du menu contextuel du projet, sélectionnez **Team > Switch To > New Branch...**
2. Dans la fenêtre qui s'ouvre, indiquez le nom de la branche et cliquez sur **Finish**



# Créer une branche

Créez une branche nommée *develop*

Le nœud projet indique désormais *develop* comme branche actuelle.



Dans la vue History, les deux branches *master* et *develop* apparaissent désormais au niveau du dernier commit. La branche courante est en **gras**.

ID	Message	Author
86cba42	<b>develop</b> [master] [HEAD] Ajout d'une deuxième ligne	Francois ANDRE
9575bb9	V1.0 Renommage de fichier2.txt	Francois ANDRE
fa96b38	Suppression du fichier3.txt	Francois ANDRE
9f42a74	Ajout du fichier3.txt	Francois ANDRE
ee62245	Ajout du .gitignore	Francois ANDRE
a2bcc56	Un premier commentaire	Francois ANDRE



# Créer une branche

---

Créez de la même manière une branche `feature_MaFonction` et basculez dessus

## MODE COMMANDE

```
$ git branch mabranche (creation d une branche)
$ git checkout mabranche (bascule sur une branche)
$ git checkout -b mabranche (si la branche n existe pas elle est créée,
sinon il s agit d une simple bascule)
$ git branch -m monnouveauNom (renomme la branche courante)
```

Remarque : le fichier `fichier3.txt` est présent dans la nouvelle branche

# Basculer d'une branche à l'autre

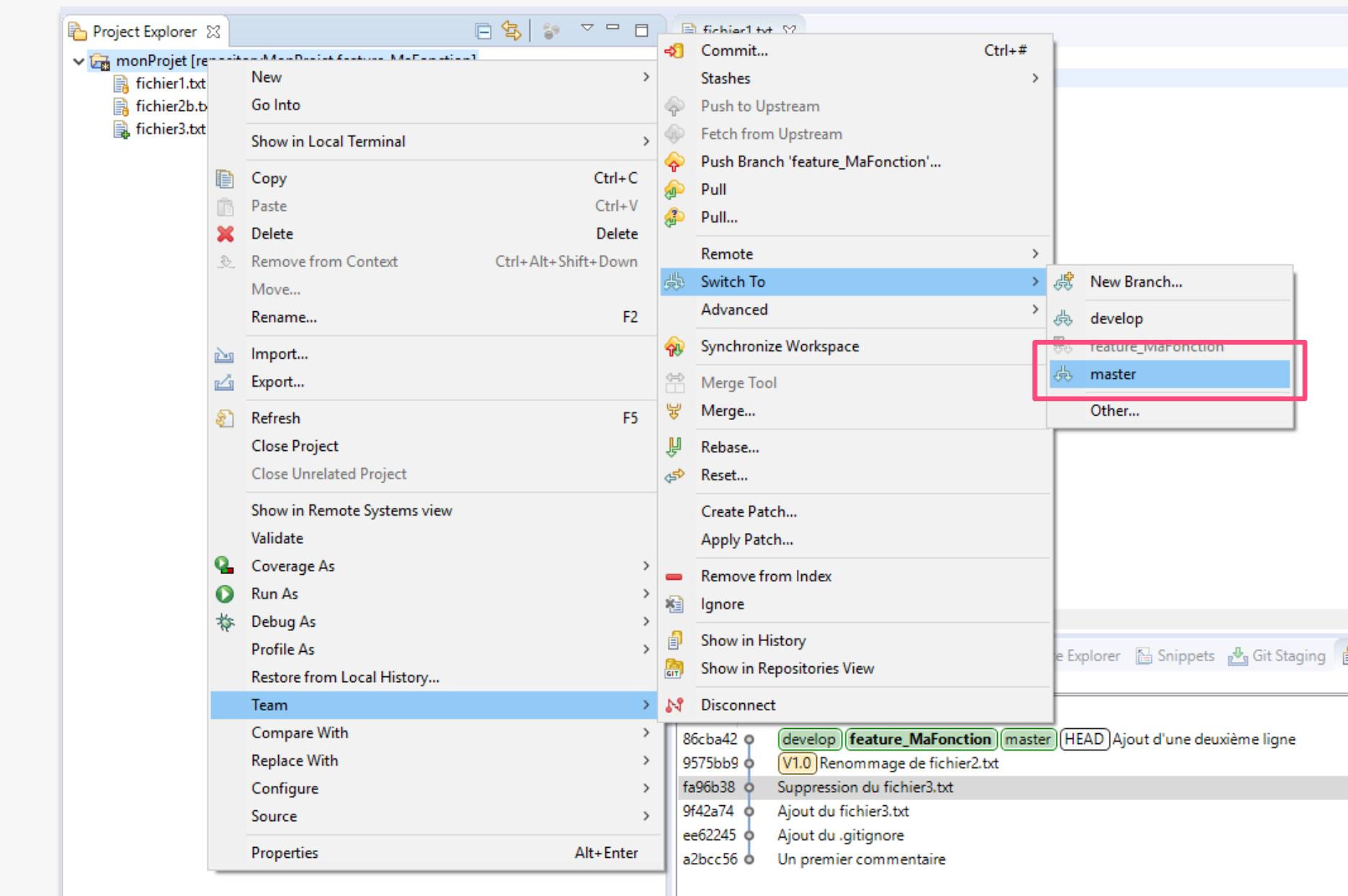
Pour basculer d'une branche à l'autre, il suffit

- d'utiliser le menu contextuel du projet.

**Team > Switch To**

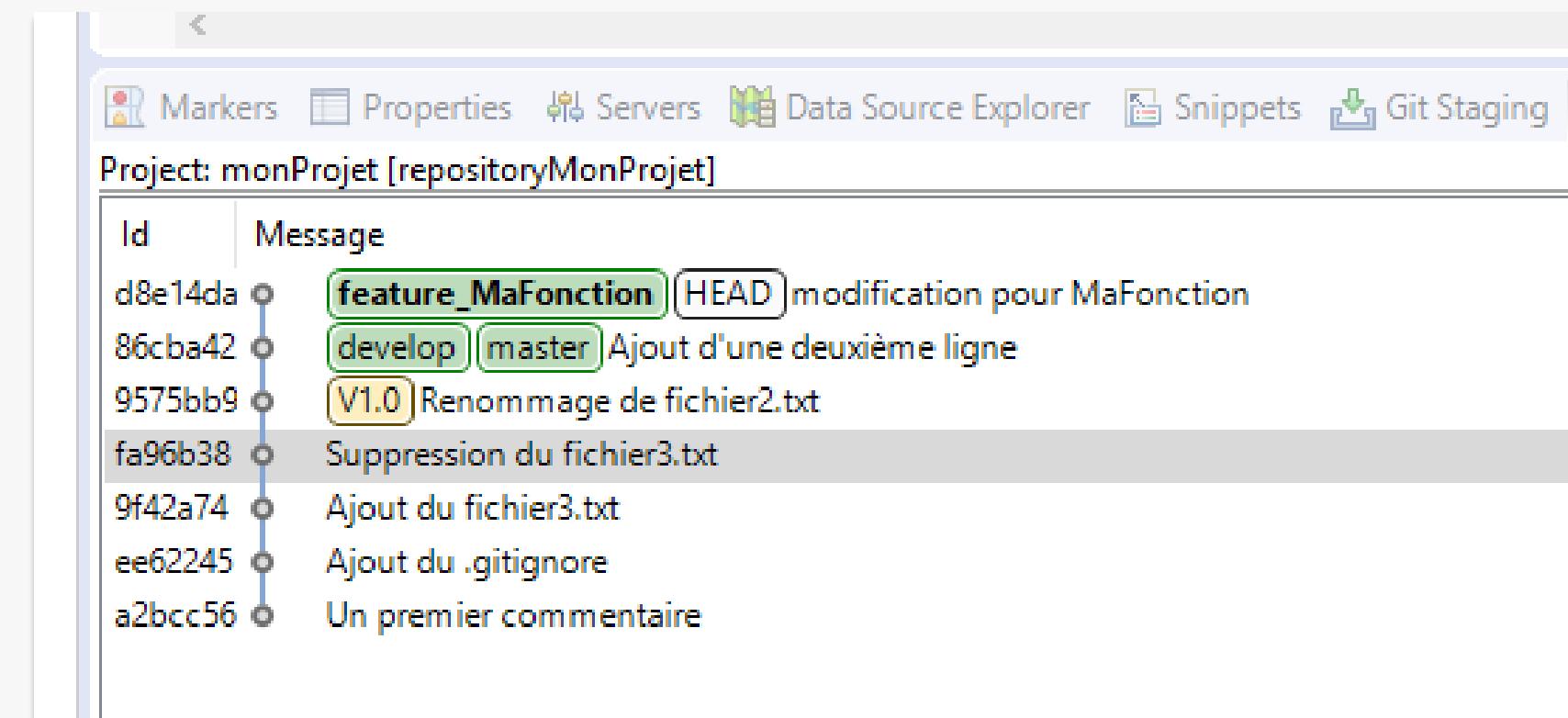
- de choisir la branche voulue

**La bascule est très rapide car totalement locale.**



# Fusion simple (Fast Forward)

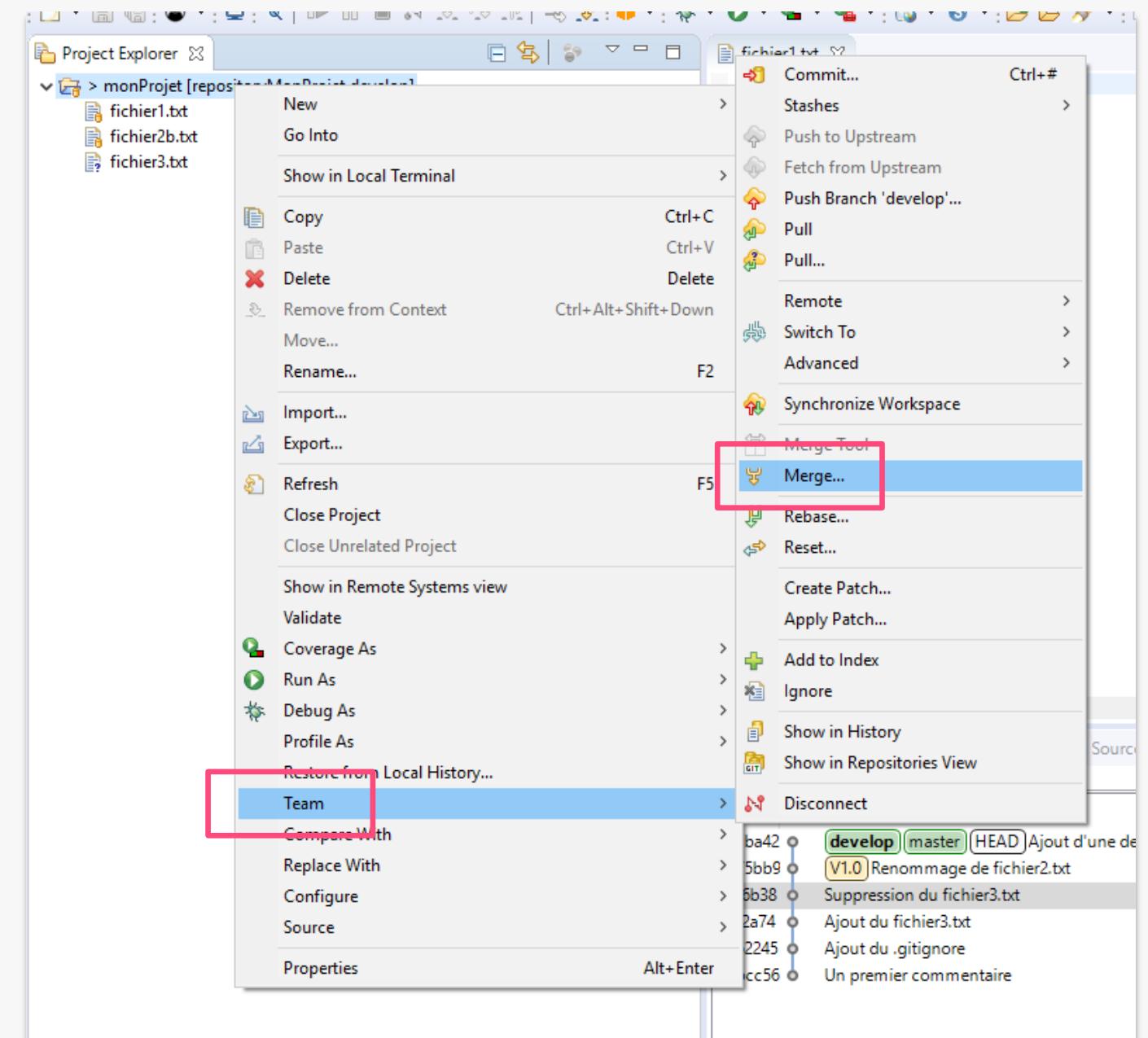
- Dans fichier1.txt ajoutez une nouvelle ligne (*Ligne ajoutée dans MaFonction*)
- Ajoutez un fichier *fichier4.txt* avec comme contenu la ligne *Contenu du fichier4.txt*
- Commitez le fichier 1 et le fichier 4 avec un commentaire identifié (*modification pour MaFonction*)



On voit que les branches *develop* et *master* sont en retard

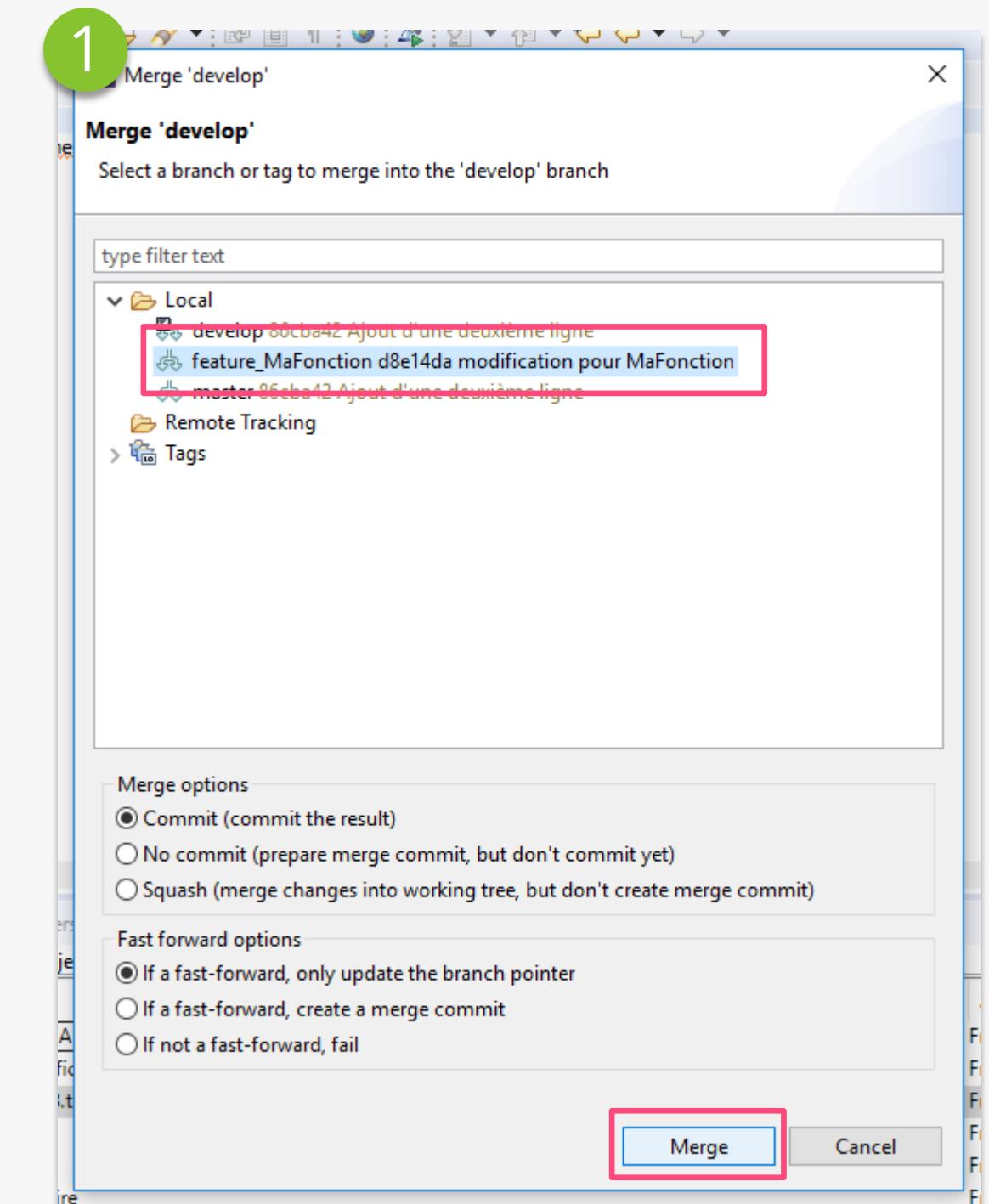
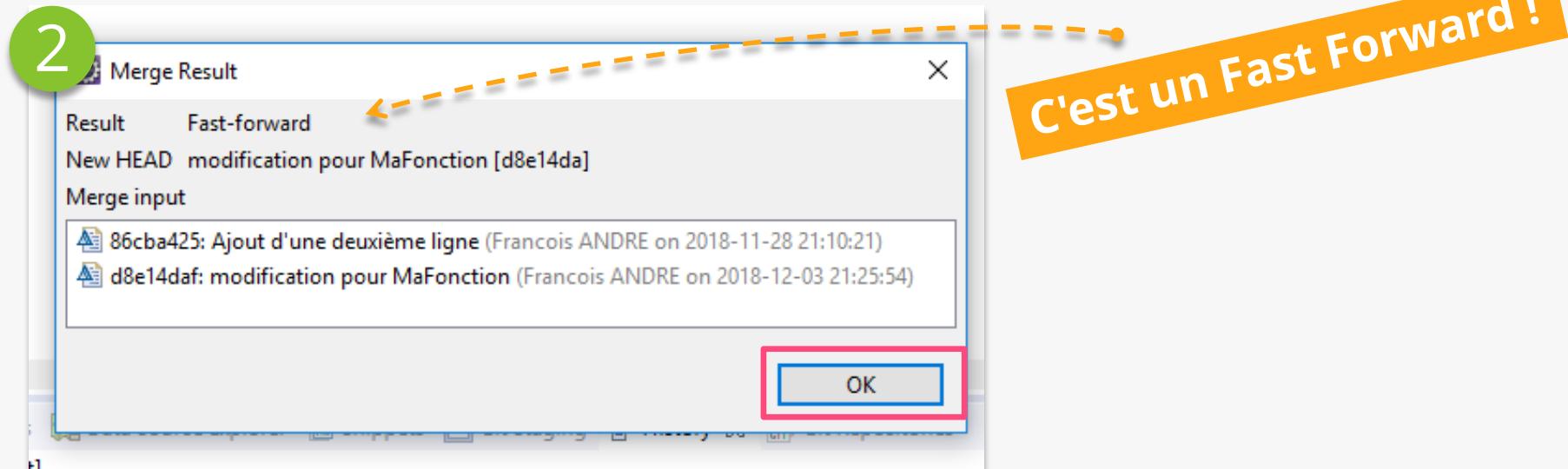
# Fusion simple (Fast Forward)

- Basculez sur la branche *develop* (le fichier *fichier1.txt* est affiché dans son état antérieur, le fichier *fichier4.txt* absent)
- Dans le menu contextuel du projet, choisissez **Team > Merge ...**



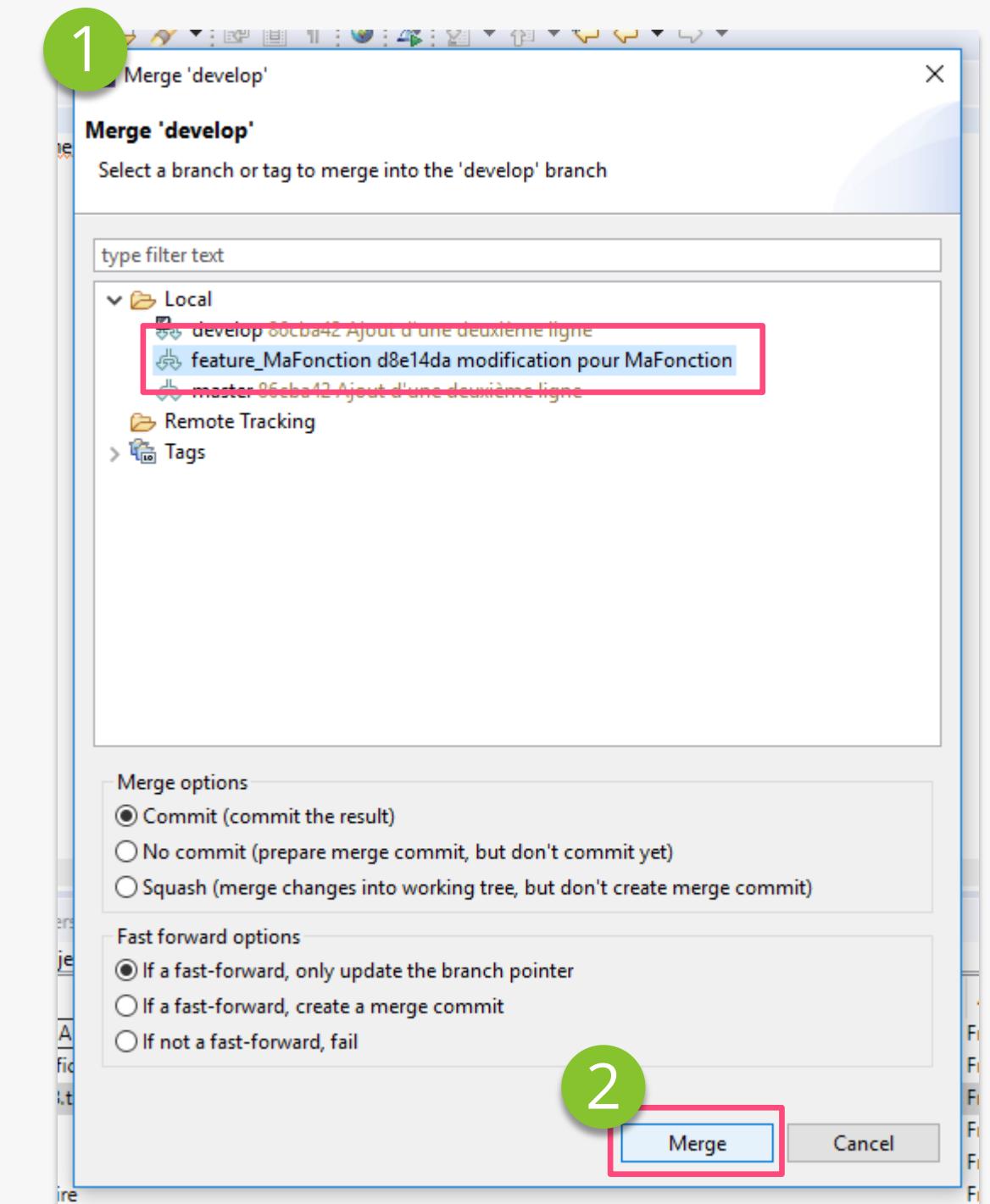
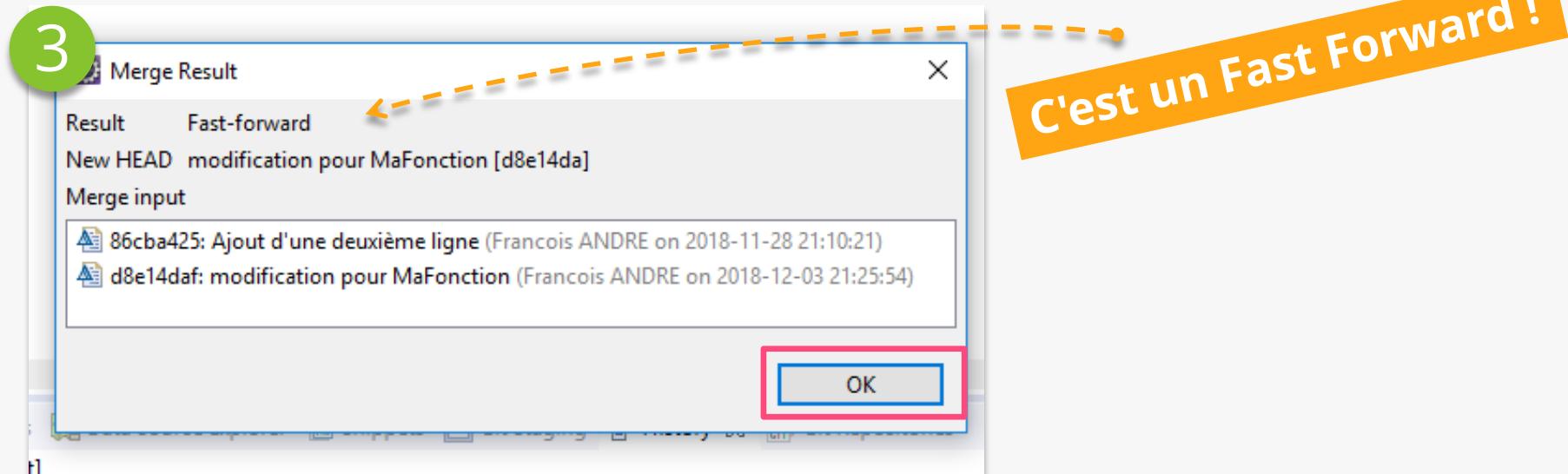
# Fusion simple (Fast Forward)

- Dans la fenêtre qui apparaît, sélectionnez la branche *feature\_MaFonction*.
- Cliquez sur *Merge*. (en conservant les options par défaut)
- Fermez la fenêtre de confirmation



# Fusion simple (Fast Forward)

1. Dans la fenêtre qui apparaît, sélectionnez la branche *feature\_MaFonction*.
2. Cliquez sur *Merge*. (en conservant les options par défaut)
3. Fermez la fenêtre de confirmation



# Fusion simple (Fast Forward)

- Le fichier 1 modifié et le fichier 4 sont bien présents
- En fait **Git n'a fait que déplacer le pointeur de la branche *develop* sur la branche *feature\_MaFonction***

Id	Message	Author
d8e14da	develop feature_MaFonction HEAD modification pour MaFonction	Francois ANDRE
86cba42	master Ajout d'une deuxième ligne	Francois ANDRE
9575bb9	V1.0 Renommage de fichier2.txt	Francois ANDRE
fa96b38	Suppression du fichier3.txt	Francois ANDRE
9f42a74	Ajout du fichier3.txt	Francois ANDRE
ee62245	Ajout du .gitignore	Francois ANDRE
a2bcc56	Un premier commentaire	Francois ANDRE

commit fa96b3821f4f0121f00aefc5cf6d10d642f6d5ff  
Author: Francois ANDRE <francois.andre@orangeville.fr> 2018-11-26 21:46:28

# Fusion avec conflit

- Basculez sur la branche *master*
- Créez une branche *Hotfix\_1* (à partir de *master*) et basculez sur elle

Bouton permettant d'afficher toutes les branches même les plus récentes...

The screenshot shows the Eclipse IDE's Git History view for a project named "monProjet". The history table lists several commits:

Id	Message	Author	Authored Date	Committer	Committed Date
d8e14da	[develop] feature_MaFonction modification pour MaFonction	Francois ANDRE	39 minutes ago	Francois ANDRE	39 minutes ago
86cba42	[Hotfix_1] [master] HEAD Ajout d'une deuxième ligne	Francois ANDRE	5 days ago	Francois ANDRE	5 days ago
9575bb9	[v1.0] Renommage de fichier2.txt	Francois ANDRE	7 days ago	Francois ANDRE	7 days ago
fa96b28	Suppression du fichier3.txt	Francois ANDRE	7 days ago	Francois ANDRE	7 days ago
9f42a74	Ajout du fichier3.txt	Francois ANDRE	7 days ago	Francois ANDRE	7 days ago
ee62245	Ajout du .gitignore	Francois ANDRE	7 days ago	Francois ANDRE	7 days ago
a2bcc56	Un premier commentaire	Francois ANDRE	7 days ago	Francois ANDRE	7 days ago

La branche *Hotfix\_1* est en retard par rapport à *develop*

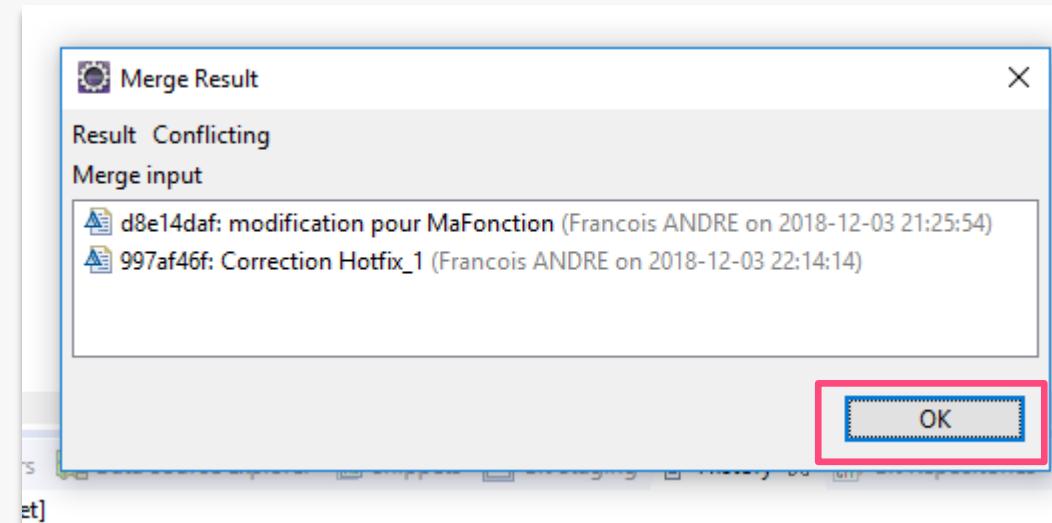
# Fusion avec conflit

- Dans le fichier *fichier1.txt* ajouter une ligne *Correction hotfix 1*
- Commitez ce fichier
- Revenez sur *master*
- Mergez la branche *Hotfix\_1* (c'est un Fast Forward !)
- Basculez sur *develop*

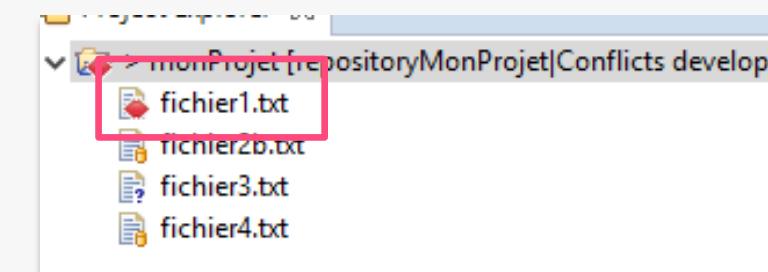
Id	Message	Author
997af46	[Hotfix_1] [master] Correction Hotfix_1	Francois ANDRE
d8e14da	[develop] [feature_MaFonction] [HEAD] modification pour MaFonction	Francois ANDRE
86cba42	Ajout d'une deuxième ligne	Francois ANDRE
9575bb9	V1.0 Renommage de fichier2.txt	Francois ANDRE
fa96b38	Suppression du fichier3.txt	Francois ANDRE
9f42a74	Ajout du fichier3.txt	Francois ANDRE
ee62245	Ajout du .gitignore	Francois ANDRE
a2bcc56	Un premier commentaire	Francois ANDRE

# Fusion avec conflit

- Supprimez le fichier *fichier3.txt* (*juste pour simplifier*)
- Demandez la fusion avec la branche *Hotfix\_1*
- Il y a dans ce cas, un conflit avec les deux versions de *fichier1.txt* (il a été modifié dans chacune des branches)



- Dans l'arborescence, le fichier *fichier1.txt* apparait avec l'icône indiquant un conflit

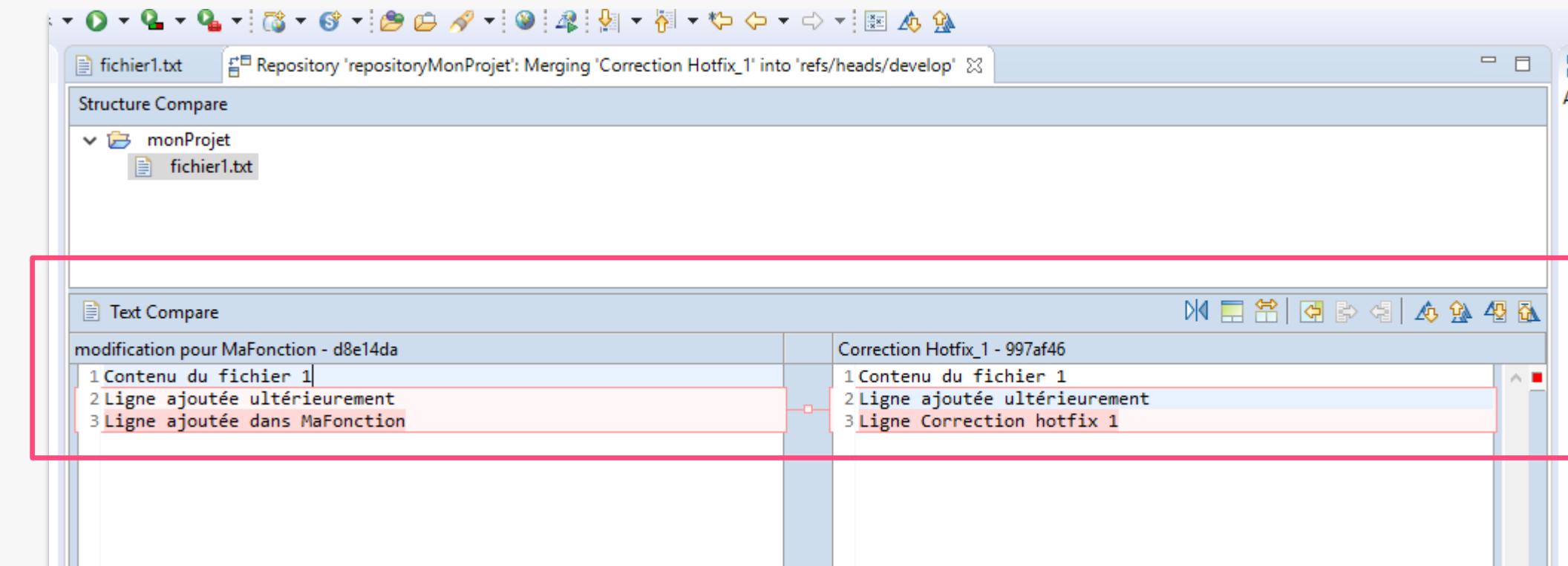


# Fusion avec conflit

- Le fichier apparaît de même dans la vue *Git Staging*



- Lorsqu'on clique sur le fichier à partir de cette vue, Eclipse nous propose un éditeur nous permettant de visualiser graphiquement les différences



# Fusion avec conflit

- Lorsqu'on ouvre le fichier directement le fichier à partir de l'arborescence projet on voit que pour l'instant Git a modifié le contenu du fichier assez brutalement.

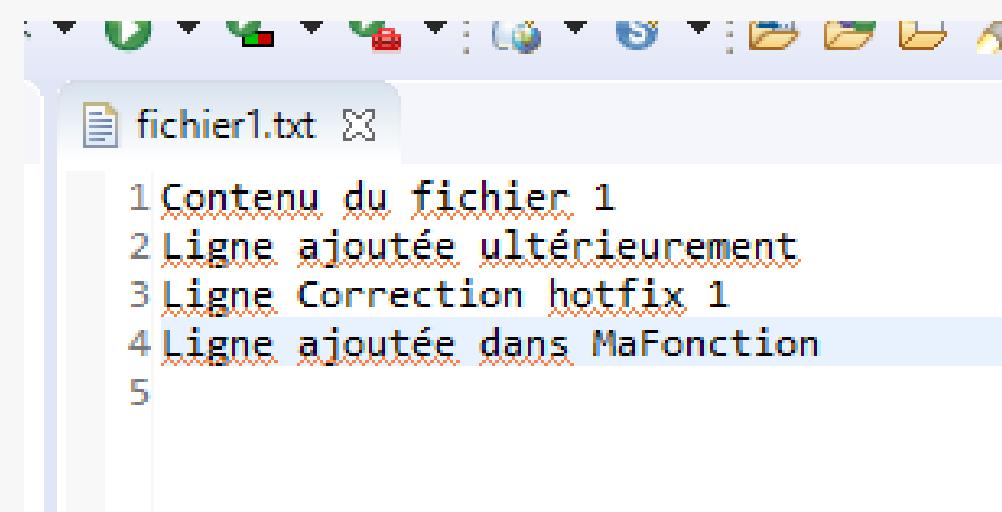
```
1 <<<<< HEAD
2 >>>>> refs/heads/Hotfix_1
3 Ligne ajoutée dans MaFonction
4 =====
5 Ligne ajoutée ultérieurement
6 Ligne Correction hotfix 1
7
```

Lignes 1 et 2 : partie commune  
Lignes 3 à 5: lignes correspondant à l'ajout fait sur le HEAD (develop)  
Lignes 5 à 7: lignes venant de la branche Hotfix\_1

- Cette vision est moins esthétique que l'éditeur graphique mais elle est plus efficace à manipuler.

# Fusion avec conflit

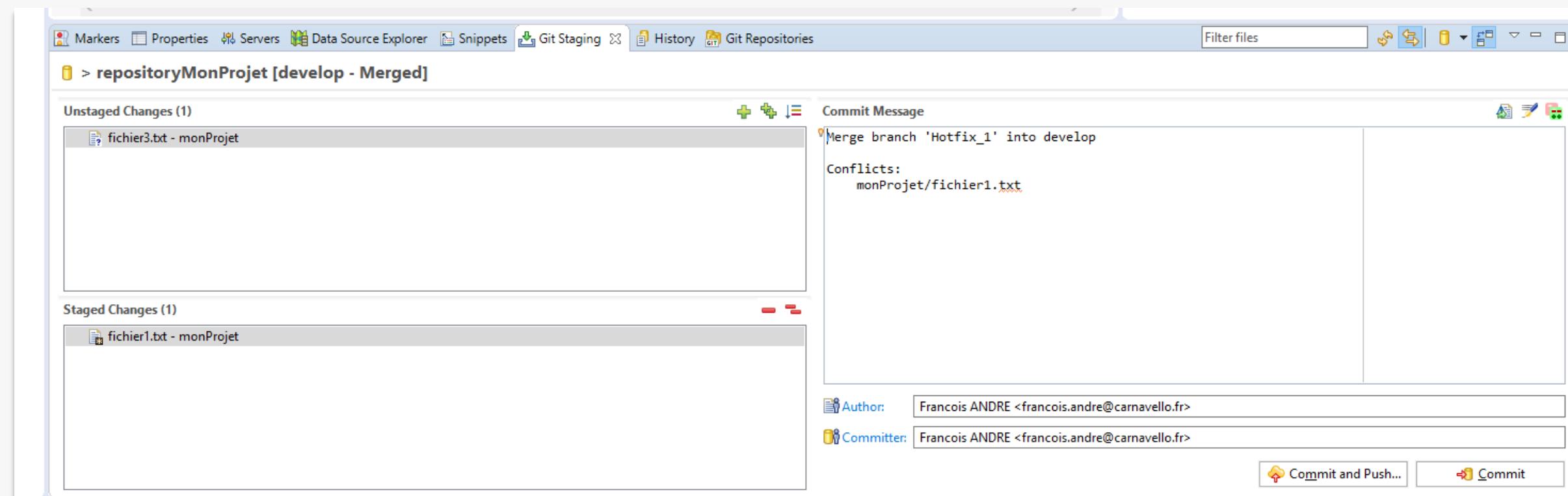
- La résolution du conflit doit être effectuée manuellement par le développeur.
- Par exemple:



- Une fois sauvé, le fichier doit être ajouté à la zone de transit
- Le conflit est alors résolu.

# Fusion avec conflit

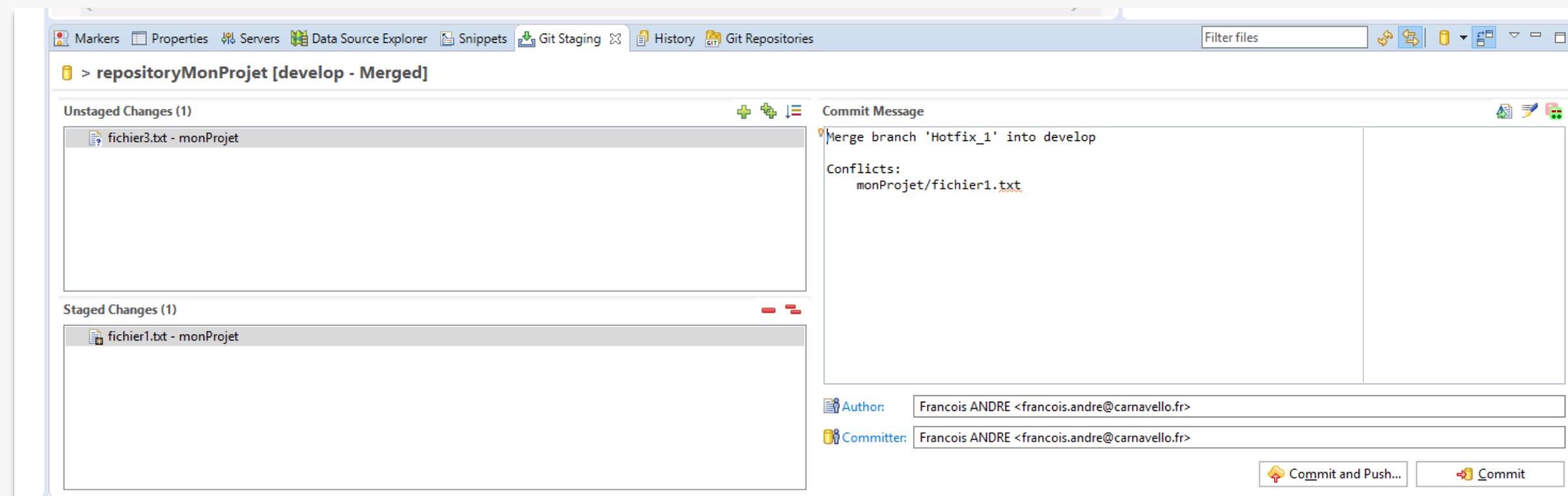
- Git propose par défaut un message de commit adapté



- Modifiez fichier1.txt pour résoudre le conflit, ajoutez le à l'index et commitez

# Fusion avec conflit

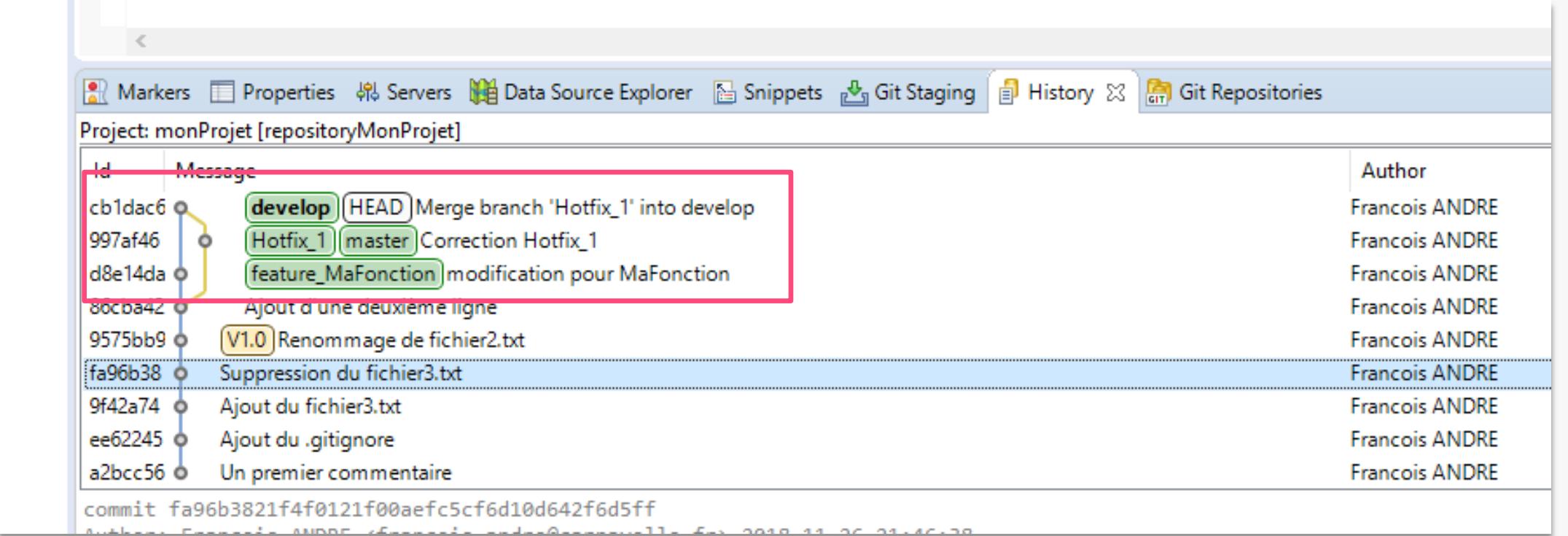
- Git propose par défaut un message de commit adapté



- Modifiez fichier1.txt pour résoudre le conflit, ajoutez le à l'index et commitez

# Fusion avec conflit

- La situation est alors la suivante :



Id	Message	Author
cb1dac6	develop HEAD Merge branch 'Hotfix_1' into develop	Francois ANDRE
997af46	Hotfix_1 master Correction Hotfix_1	Francois ANDRE
d8e14da	feature_MaFonction modification pour MaFonction	Francois ANDRE
80cc0a42	Ajout d'une deuxième ligne	Francois ANDRE
9575bb9	V1.0 Renommage de fichier2.txt	Francois ANDRE
fa96b38	Suppression du fichier3.txt	Francois ANDRE
9f42a74	Ajout du fichier3.txt	Francois ANDRE
ee62245	Ajout du .gitignore	Francois ANDRE
a2bcc56	Un premier commentaire	Francois ANDRE

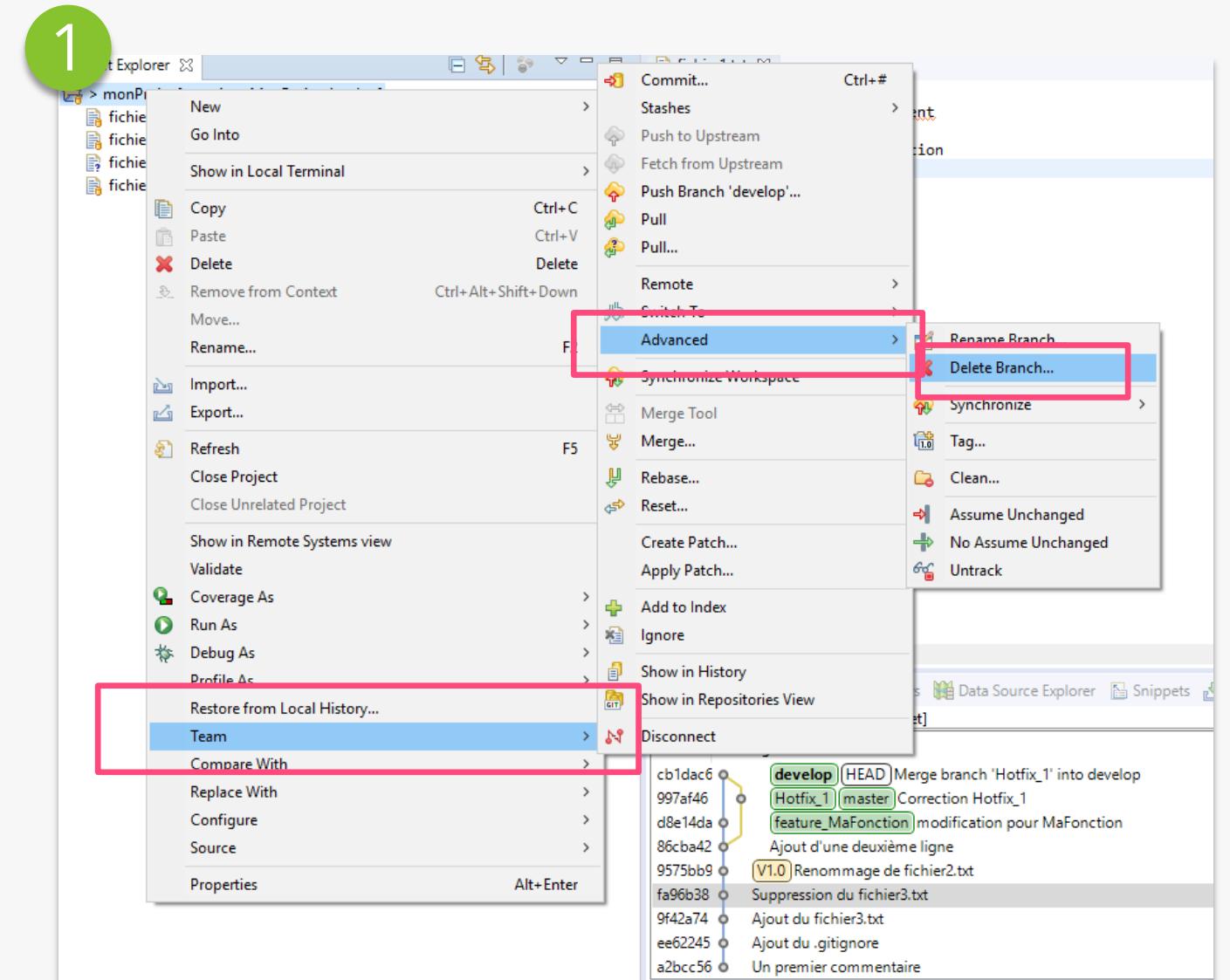
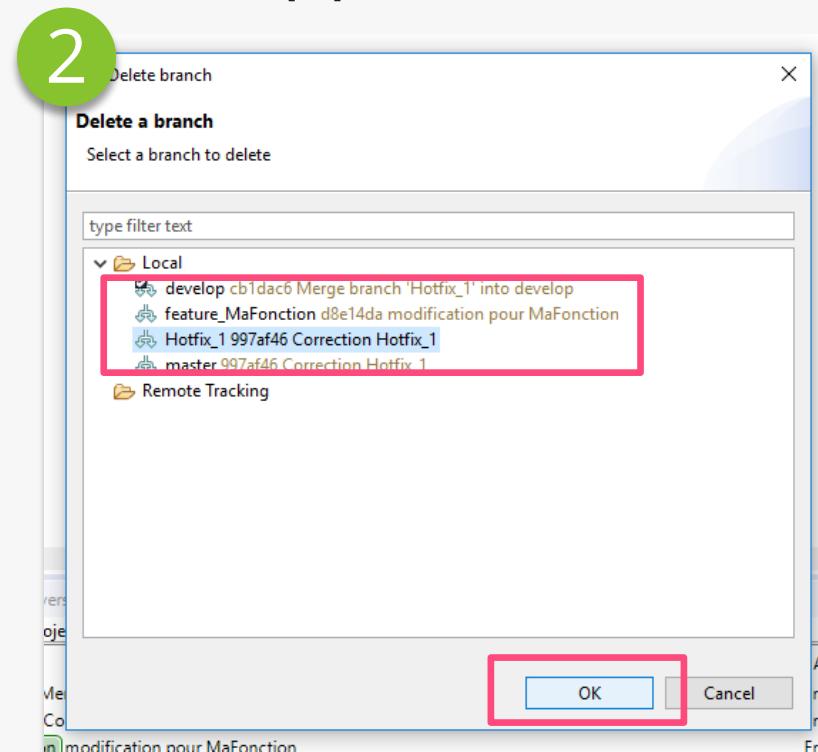
Notez la présence du *commit* de fusion.

# Suppression de branches

Les branches Hotfix\_1 et feature\_MaFonction sont des branches temporaires. **Elles peuvent être supprimées.**

La suppression d'une branche peut s'effectuer

1. Via le menu contextuel du projet: **Team > Advanced > Delete Branch...**
2. Puis en sélectionnant la branche à supprimer





# Suppression de branches

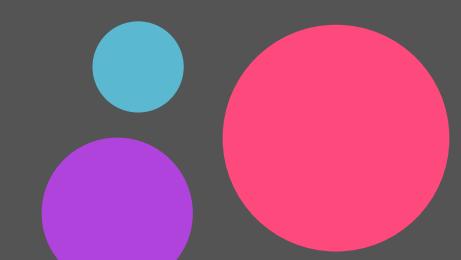
---

Supprimez les branches *Hotfix\_1* et *feature\_MaFonction*

Remarque: Git vous avertit si vous tentez de supprimer une branche n'ayant jamais été fusionnée



Échanger avec d'autres développeurs :  
les dépôts distants





# Les dépôts distants (remote)

---

## Objectifs

- Les dépôts vont permettre aux développeurs de partager leur travail.
- Ils peuvent aussi servir de dépôt de référence pour l'intégration continue

Git offre beaucoup de souplesse dans la gestion des dépôts distants :

- Nature (cloud, disque partagé, clé usb...)
- Protocole (SSH, Git, HTTPS...)
- Nombre (utilisation de plusieurs dépôts...)
- Structure (communication directe entre développeur, serveur central ...)

Dans notre exercice nous allons créer un dépôt distant ...local au poste de travail.



# Création du dépôt distant

---

- Créez un répertoire temporaire (ex: c:\tempodemo)
- Dans ce répertoire, créez un répertoire *monDepotGit.git*
- Entrez dans le répertoire avec un terminal (*Git bash* par exemple)
- Tapez la commande *git init --bare*



```
cd
François ANDRE@UMSPW108 MINGW64 /c
$ cd tempodemo

François ANDRE@UMSPW108 MINGW64 /c/tempodemo
$ mkdir monDepotGit.git

François ANDRE@UMSPW108 MINGW64 /c/tempodemo
$ cd monDepotGit.git/

François ANDRE@UMSPW108 MINGW64 /c/tempodemo/monDepotGit.git
$ git init --bare
Initialized empty Git repository in C:/tempodemo/monDepotGit.git/
François ANDRE@UMSPW108 MINGW64 /c/tempodemo/monDepotGit.git (BARE:master)
$ ls
config  description  HEAD  hooks/  info/  objects/  refs/
François ANDRE@UMSPW108 MINGW64 /c/tempodemo/monDepotGit.git (BARE:master)
$ |
```



# Création du dépôt distant

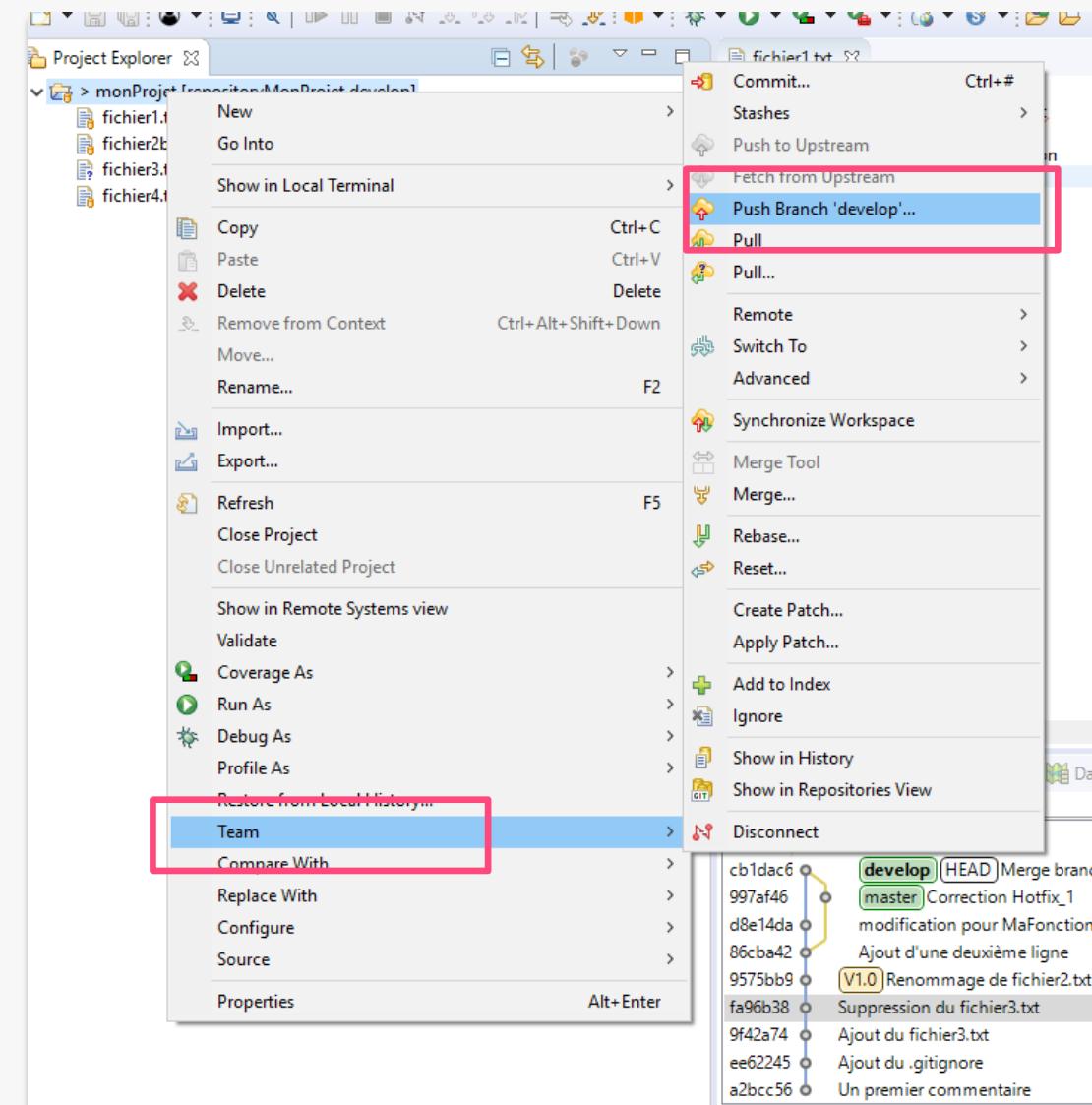
---

## Remarques:

- Par convention, le nom des dépôts Git finit par *.git*
- On voit, en inspectant le contenu du dépôt que Git à initié un certains nombre d'éléments (le *HEAD*, les *refs*, ...)
- On voit également que ce dépôt utilise par défaut une branche *master*.

# Connexion au dépôt distant

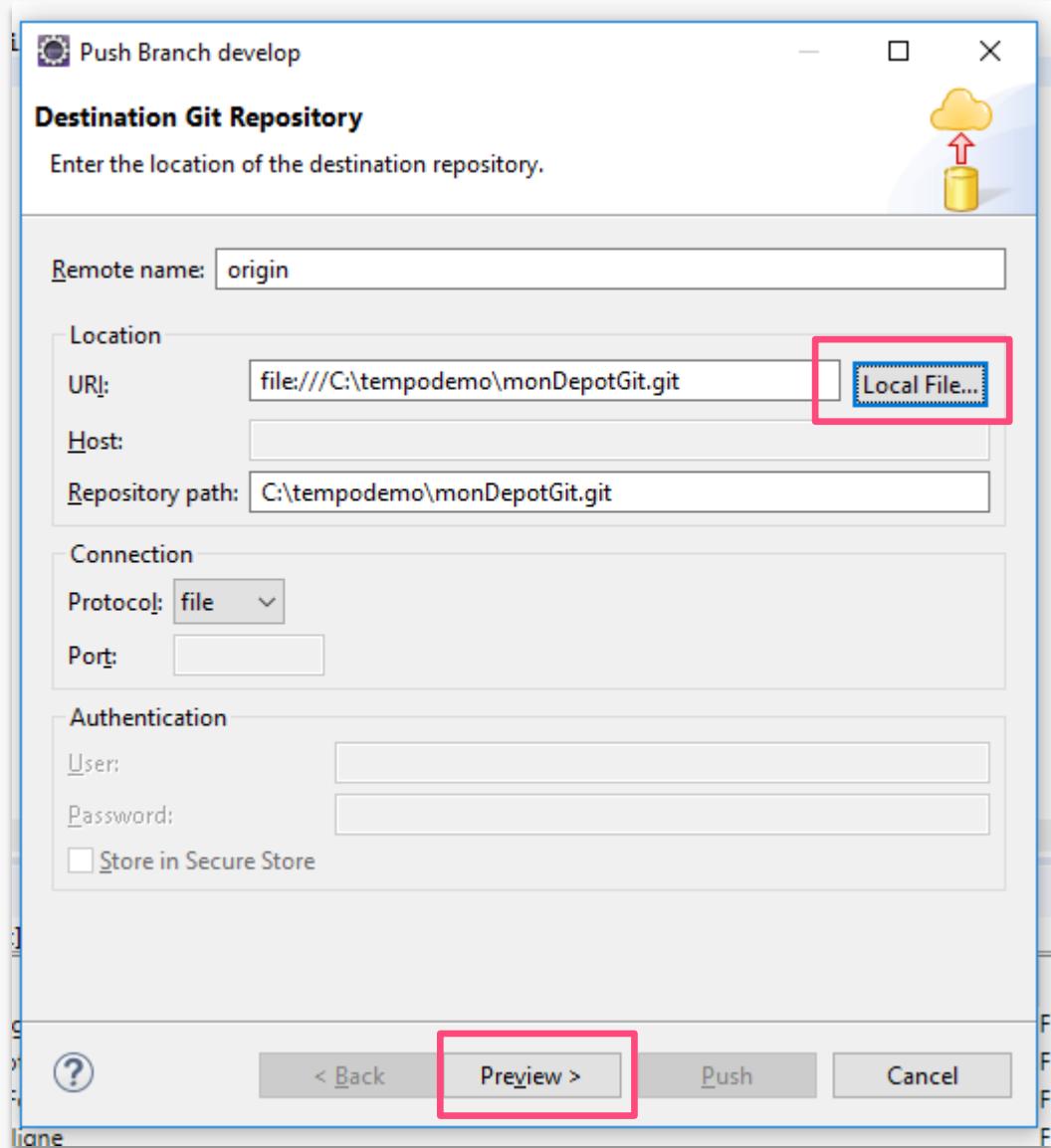
Un moyen simple de se connecter à un dépôt est de faire un *push* d'une branche existante à partir du menu contextuel du projet.



- Initiez la demande de push sur la branche *develop*

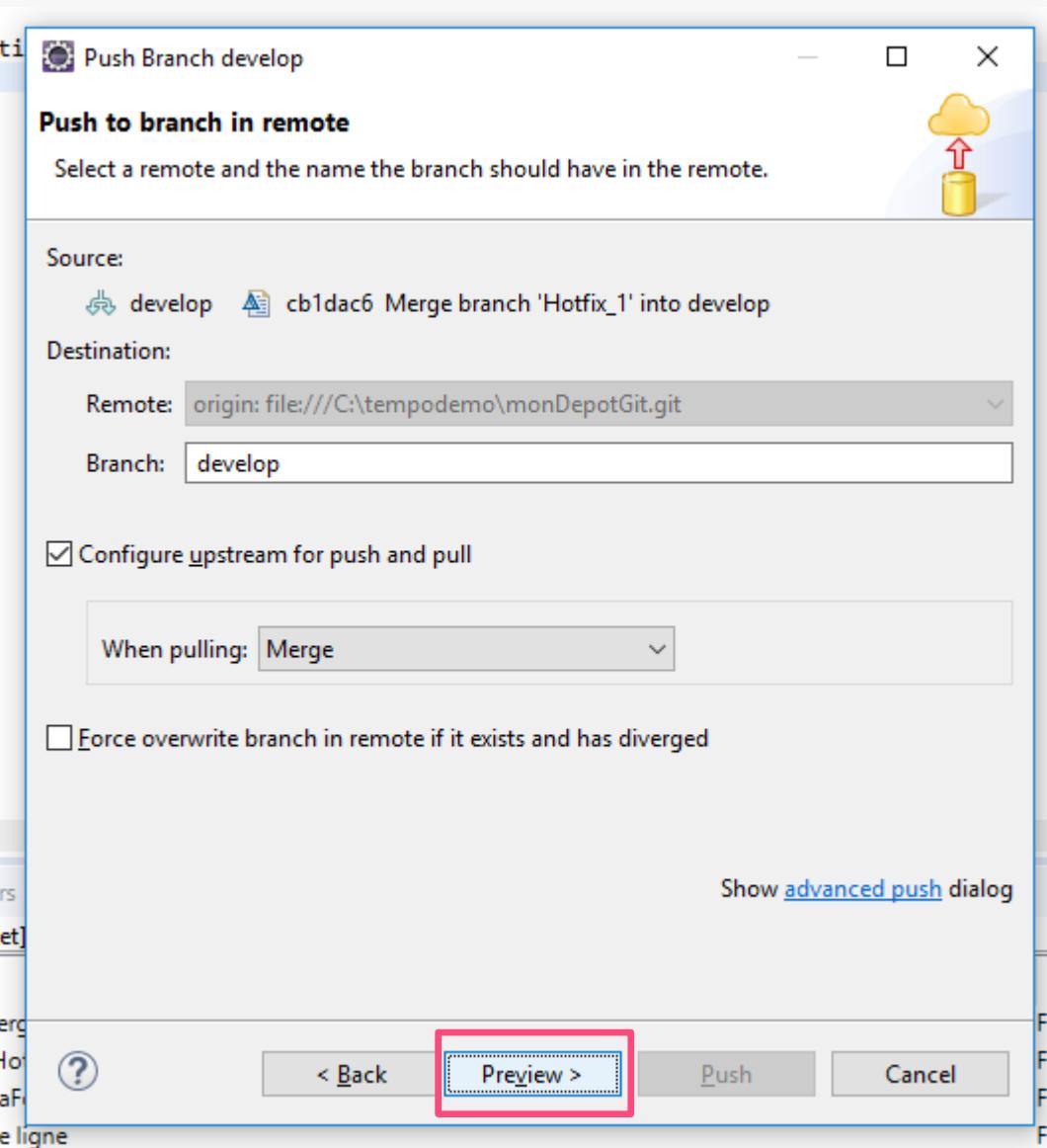
# Connexion au dépôt distant

- Sélectionnez le répertoire du dépôt via le bouton Local File...
- Puis cliquez sur Preview



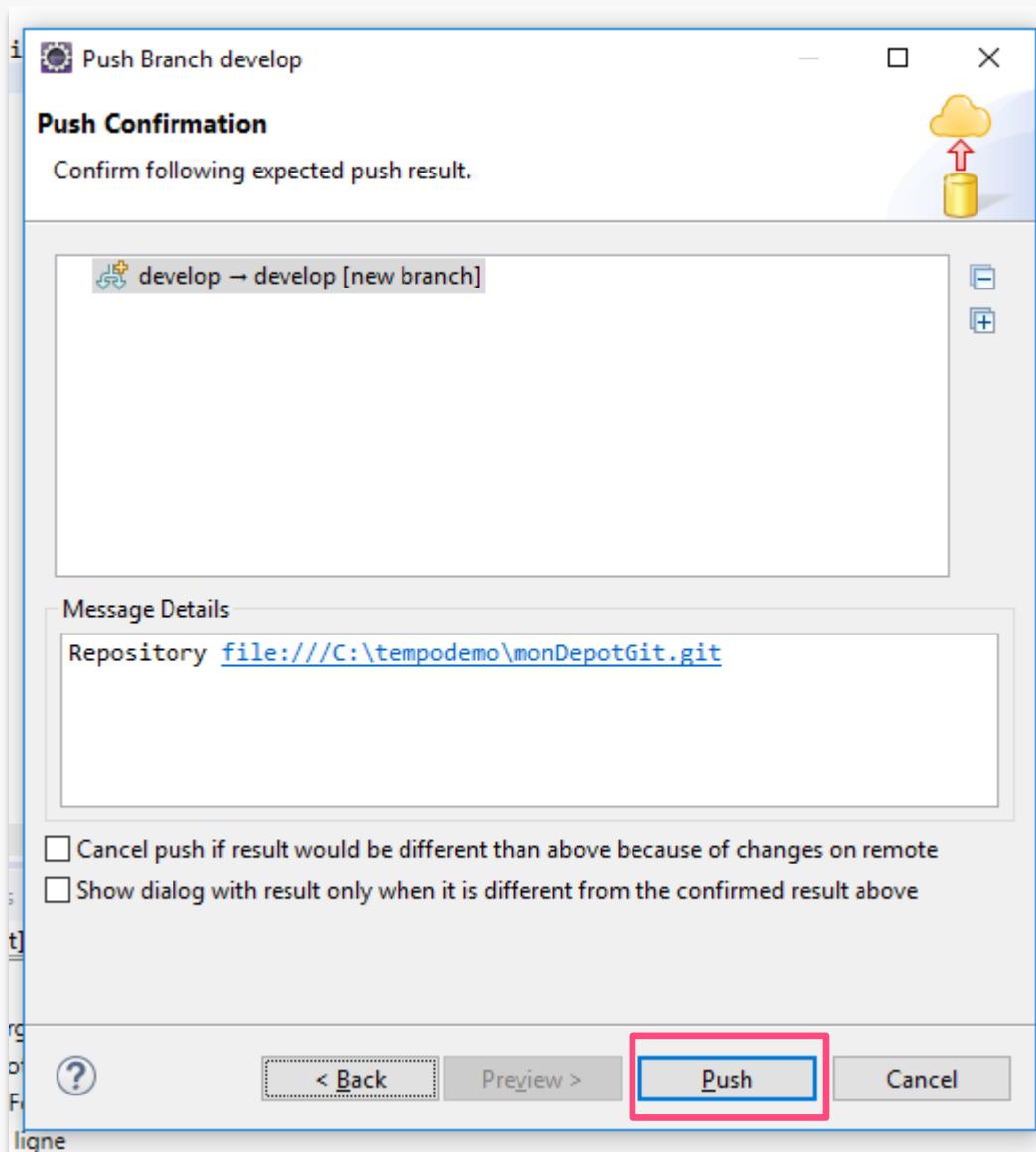
# Connexion au dépôt distant

- Cliquez sur Preview



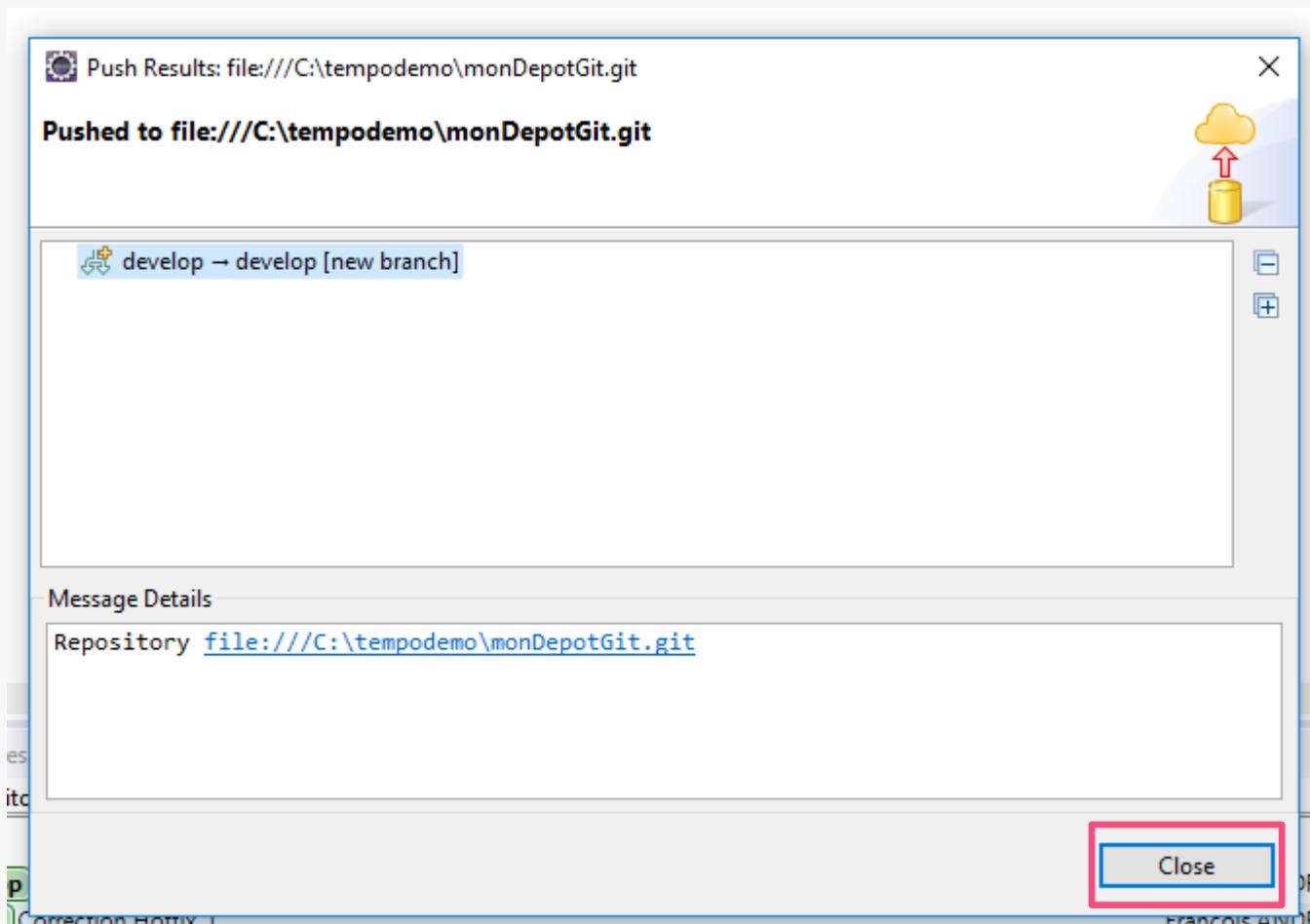
# Connexion au dépôt distant

- Cliquez sur Push pour terminer



# Connexion au dépôt distant

- Validez la fenêtre de confirmation





# Connexion au dépôt distant

---

## Remarques

- *origin* est le nom d'usage pour le serveur distant central.
- L'URL indique le protocole de connexion :

**HTTPS** : [https://github.com/francoisandre/gtdrosea\\_workshop.git](https://github.com/francoisandre/gtdrosea_workshop.git)

**SSH** : git@github.com:francoisandre/gtdrosea\_workshop.git

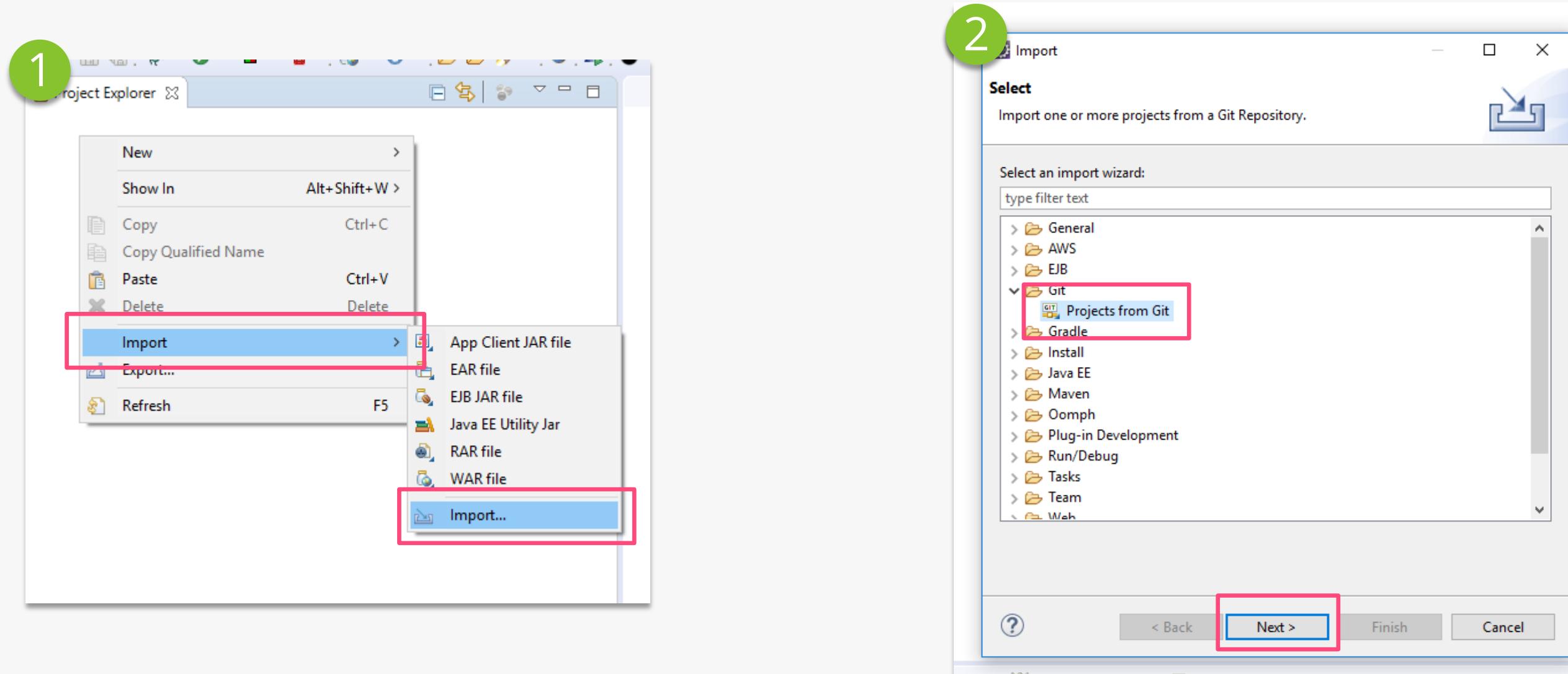
### MODE COMMANDE

```
$ git remote add aliasduremote cheminuremote (pour ajouter un remote)  
$ git push aliasduremote nomdemabranche (pour transférer une branche sur un remote)
```

# Récupération du dépôt distant

## Ajout de la vue Git Repository

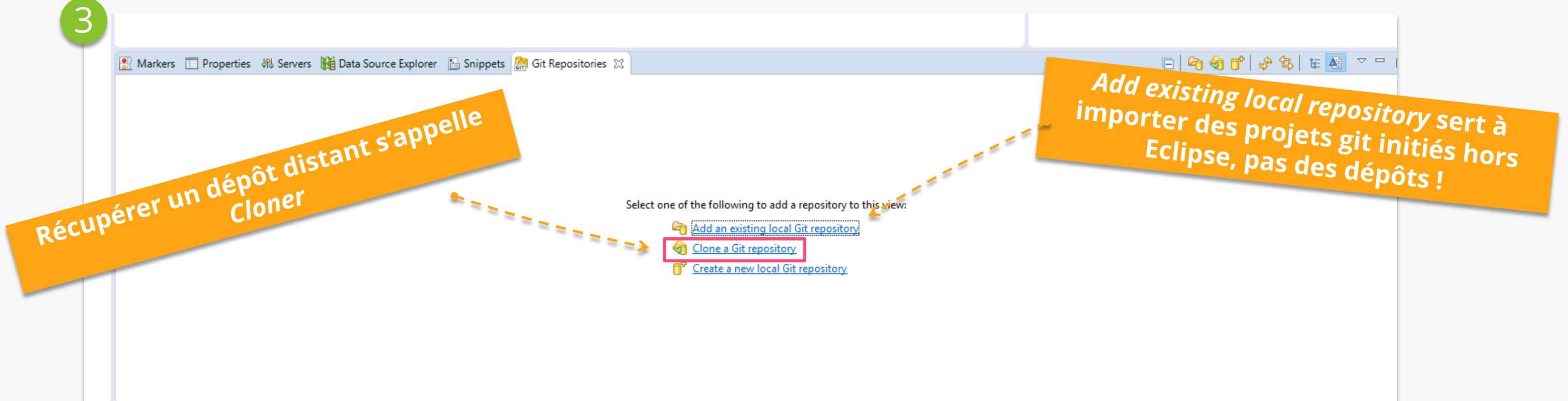
Nous allons récupérer le projet à partir du dépôt distant pour simuler un deuxième développeur. Pour cela nous allons lancer un deuxième Eclipse, dans un deuxième workspace.



# Récupération du dépôt distant

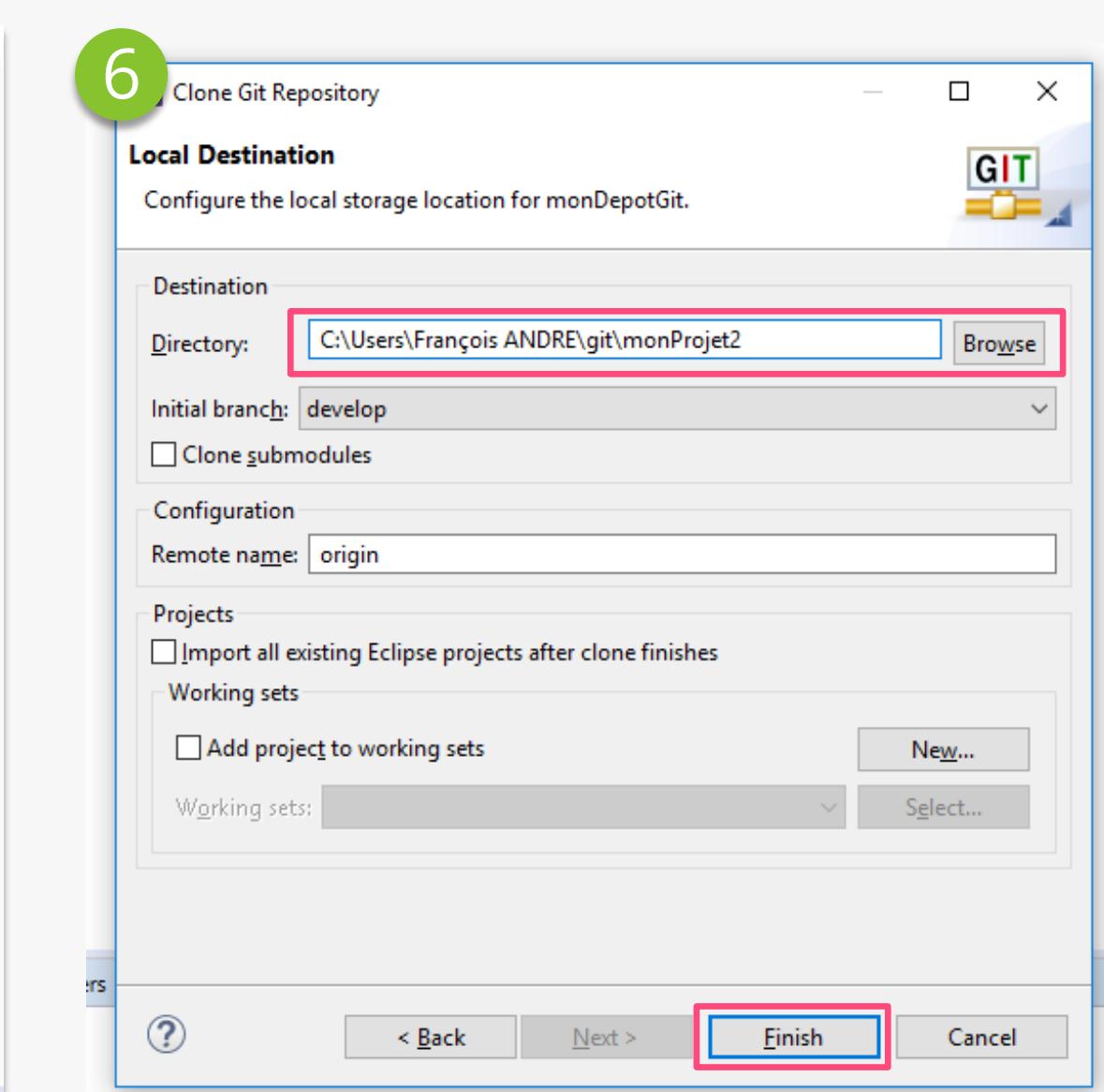
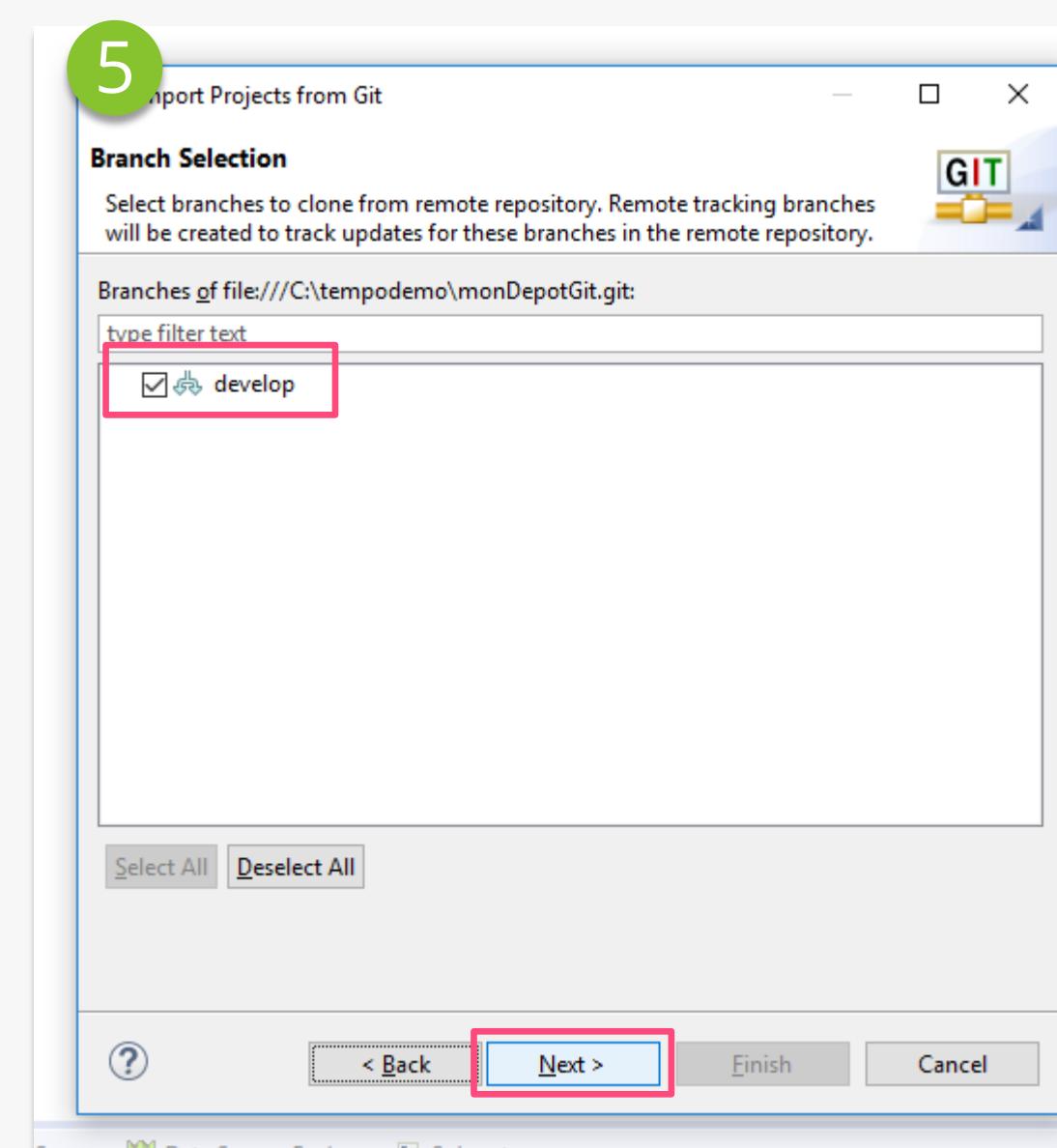
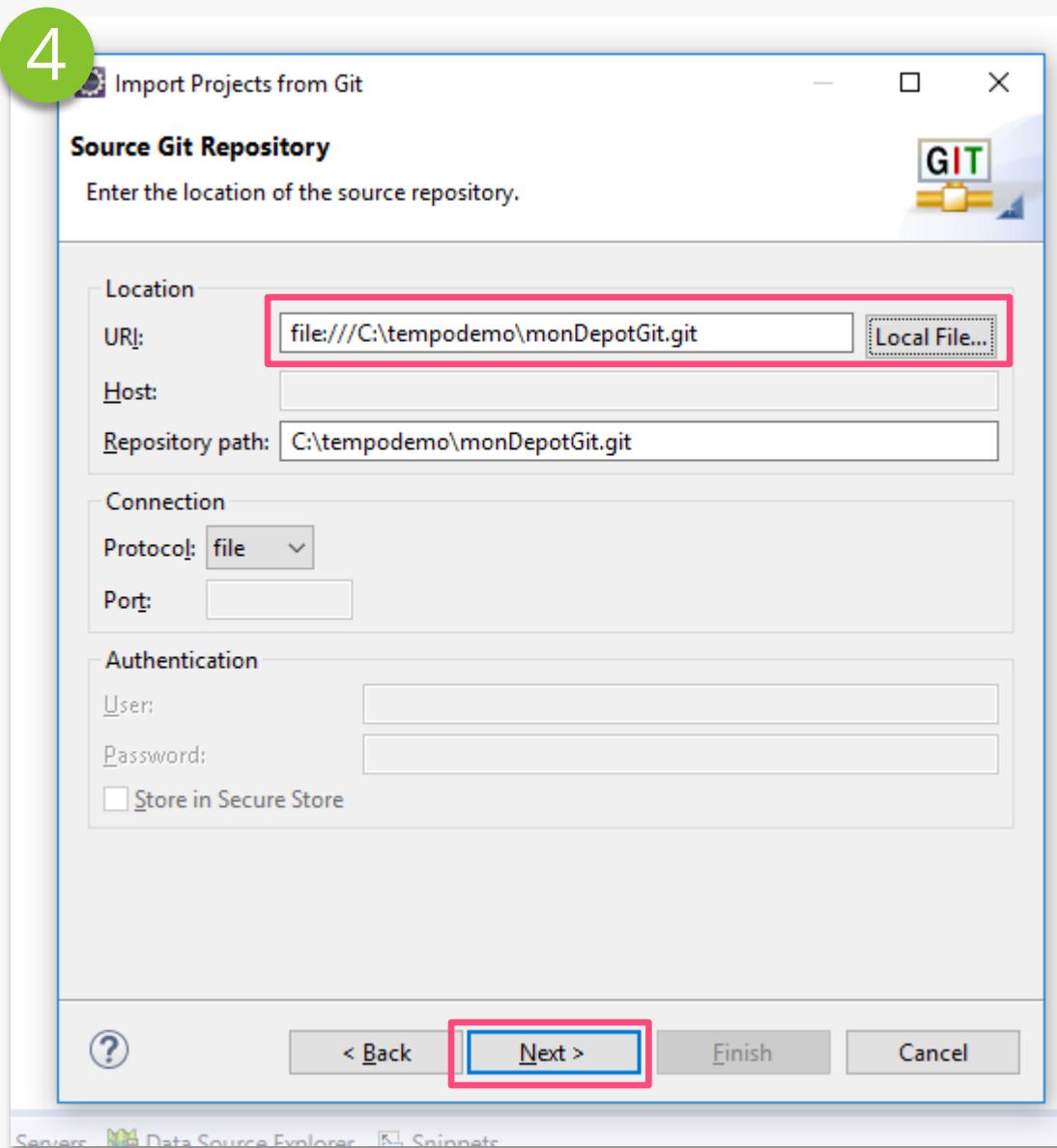
*Clonage du dépôt*

3



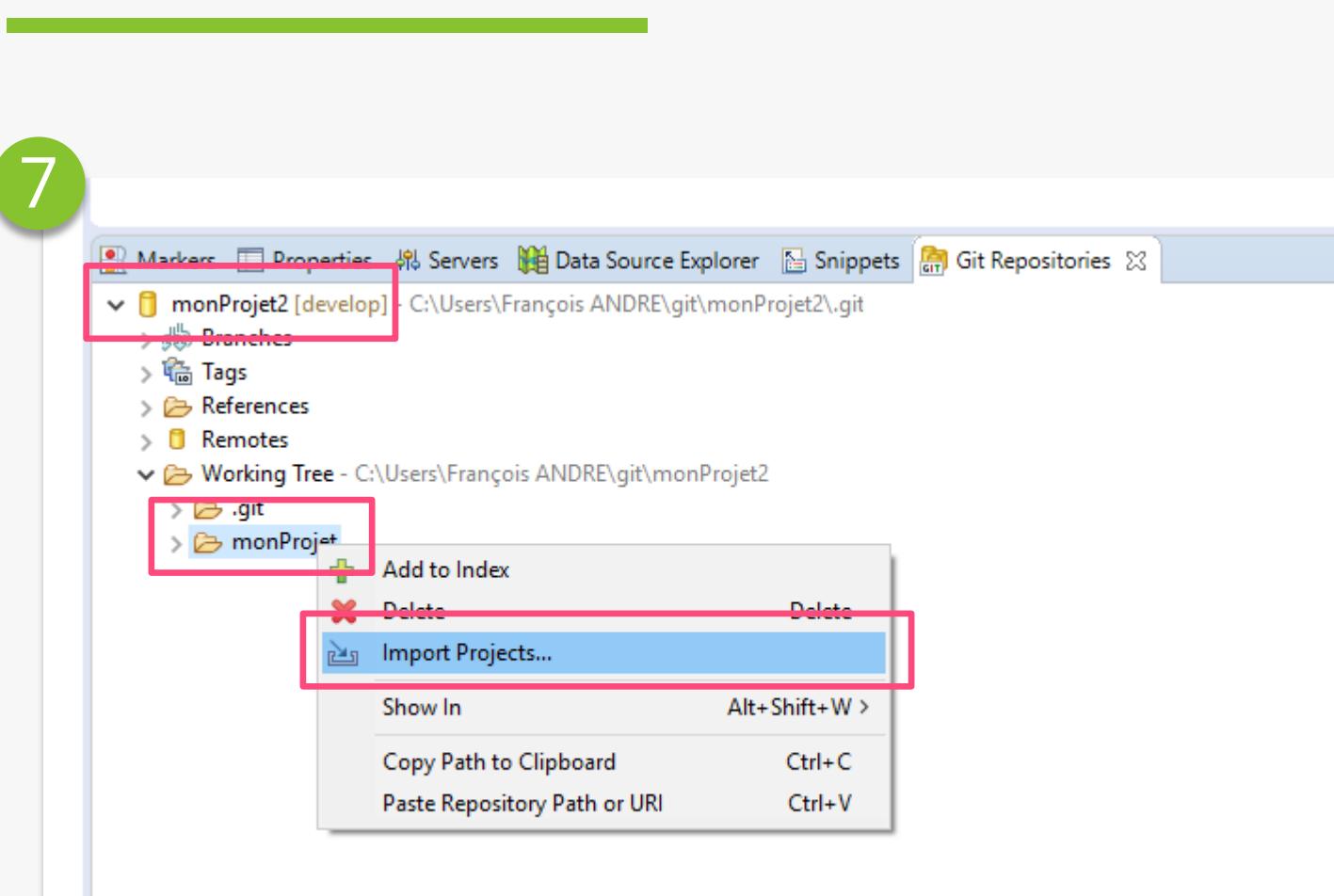
# Récupération du dépôt distant

## Clonage du dépôt

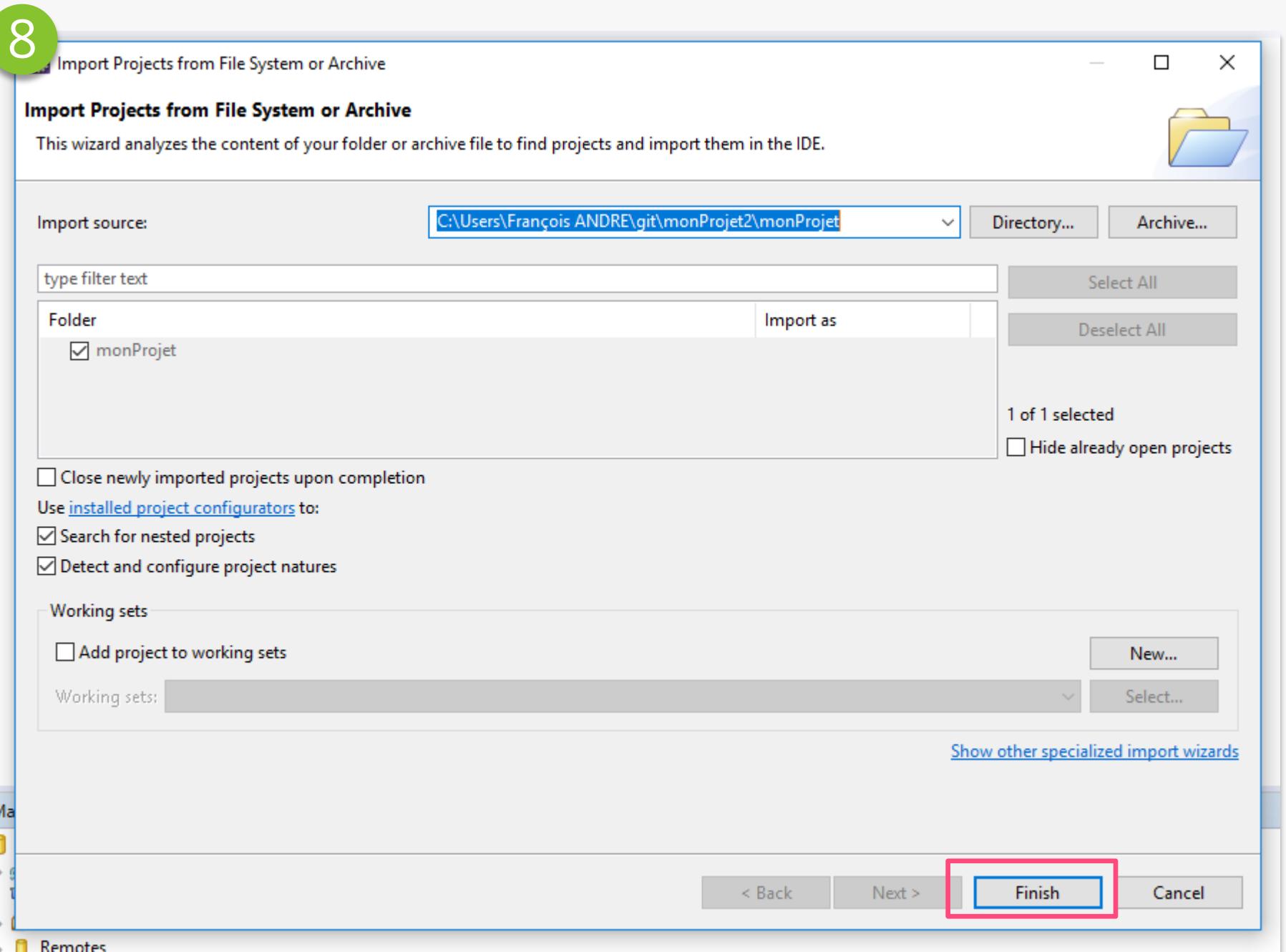


# Récupération du dépôt distant

## Clonage du dépôt



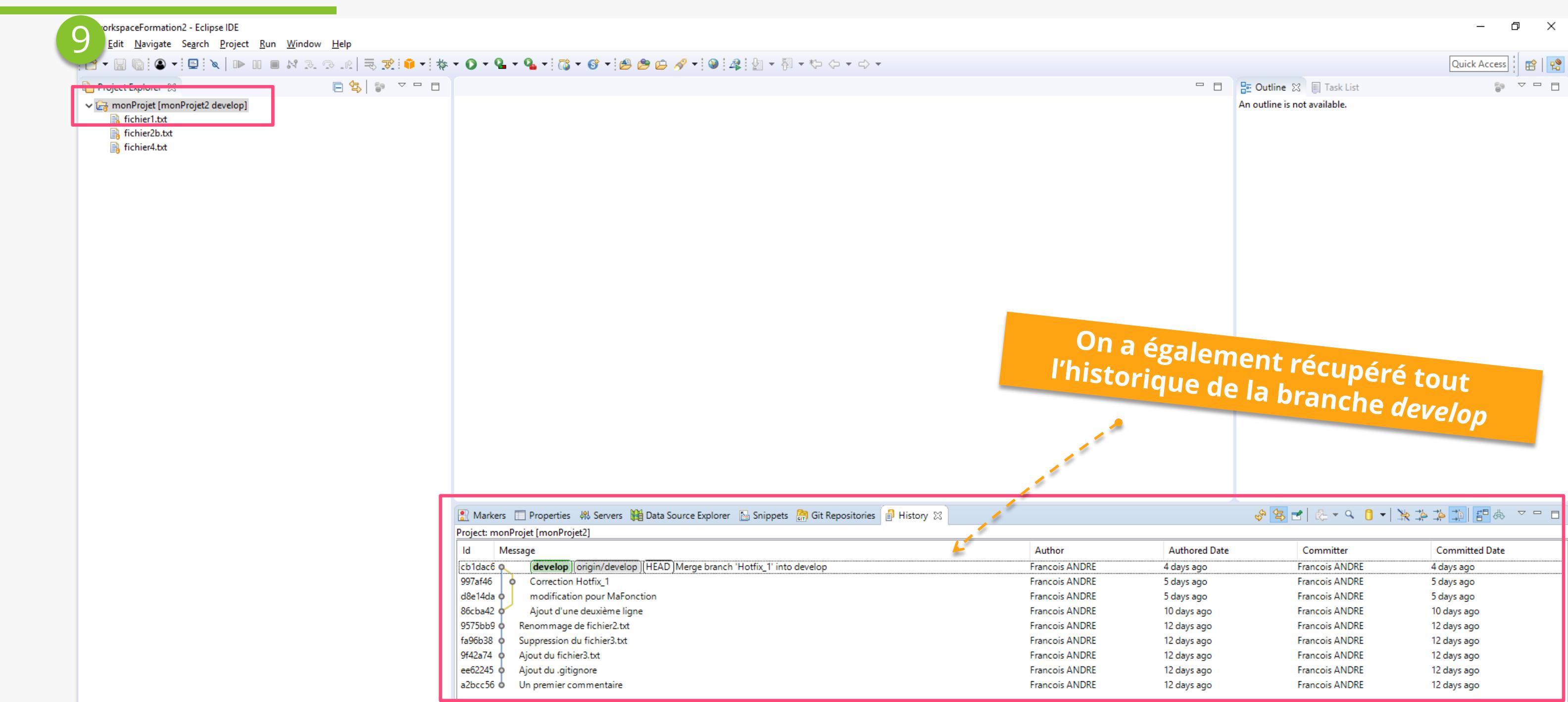
7



8

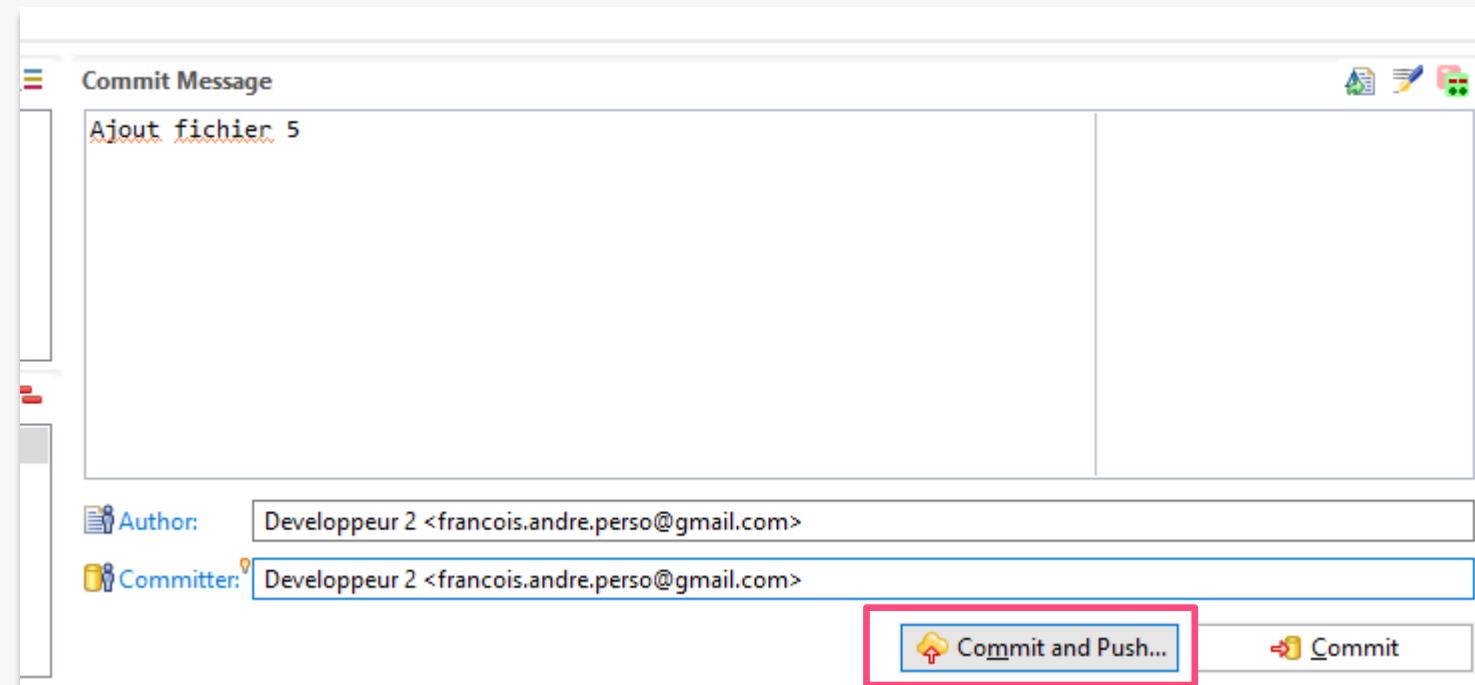
# Récupération du dépôt distant

## Clonage du dépôt



# Transfert du commit (push)

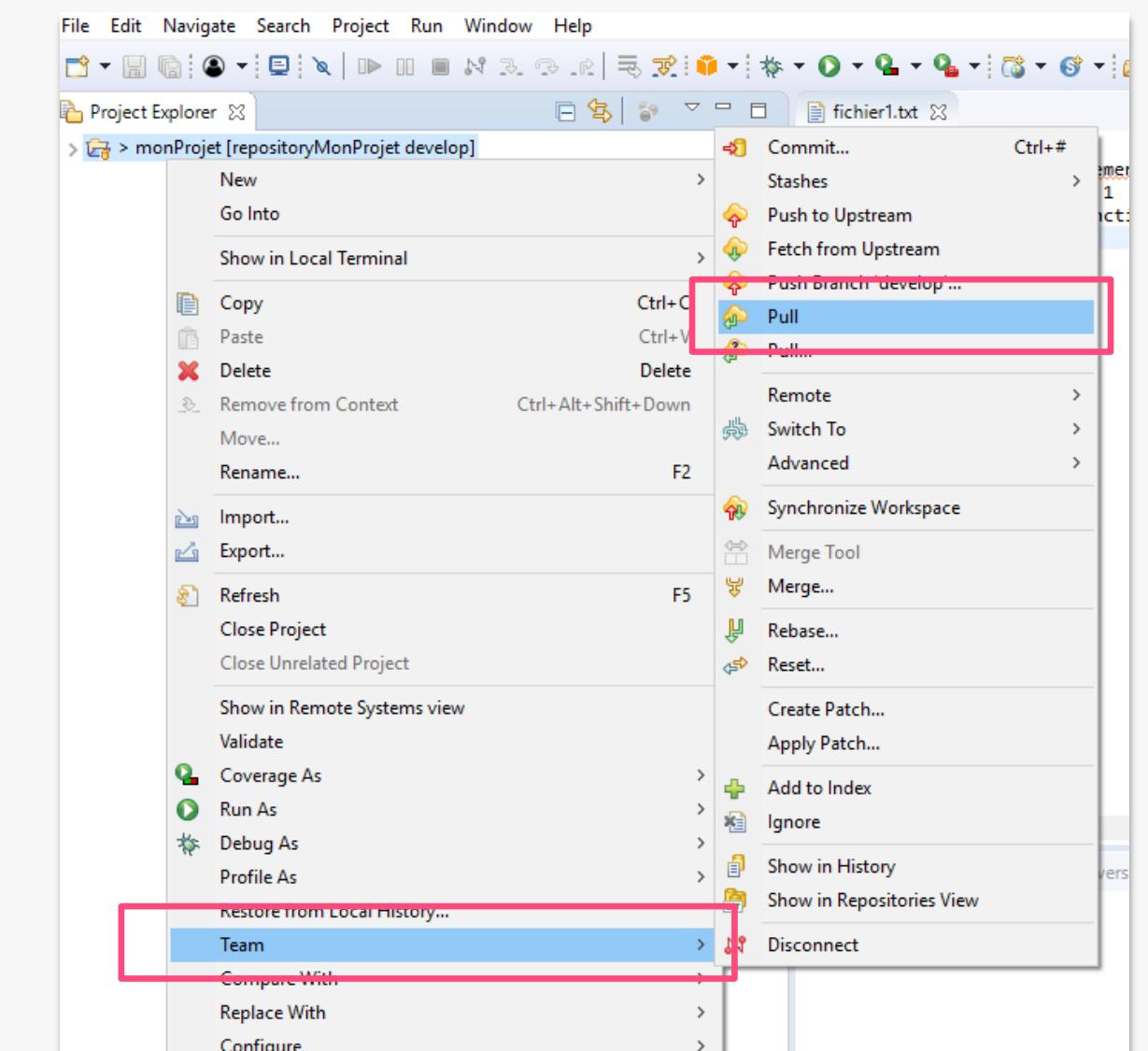
- Dans le workspace du deuxième développeur, créez un fichier *fichier5.txt* avec comme contenu *Contenu du fichier fichier5.txt*
- Faites un *commit* de ce fichier, mais dans la vue *Git Staging*
  - Indiquez comme auteur **Developpeur 2**
  - Choisissez l'option **Commit and push...**



- Fermez la fenêtre de confirmation

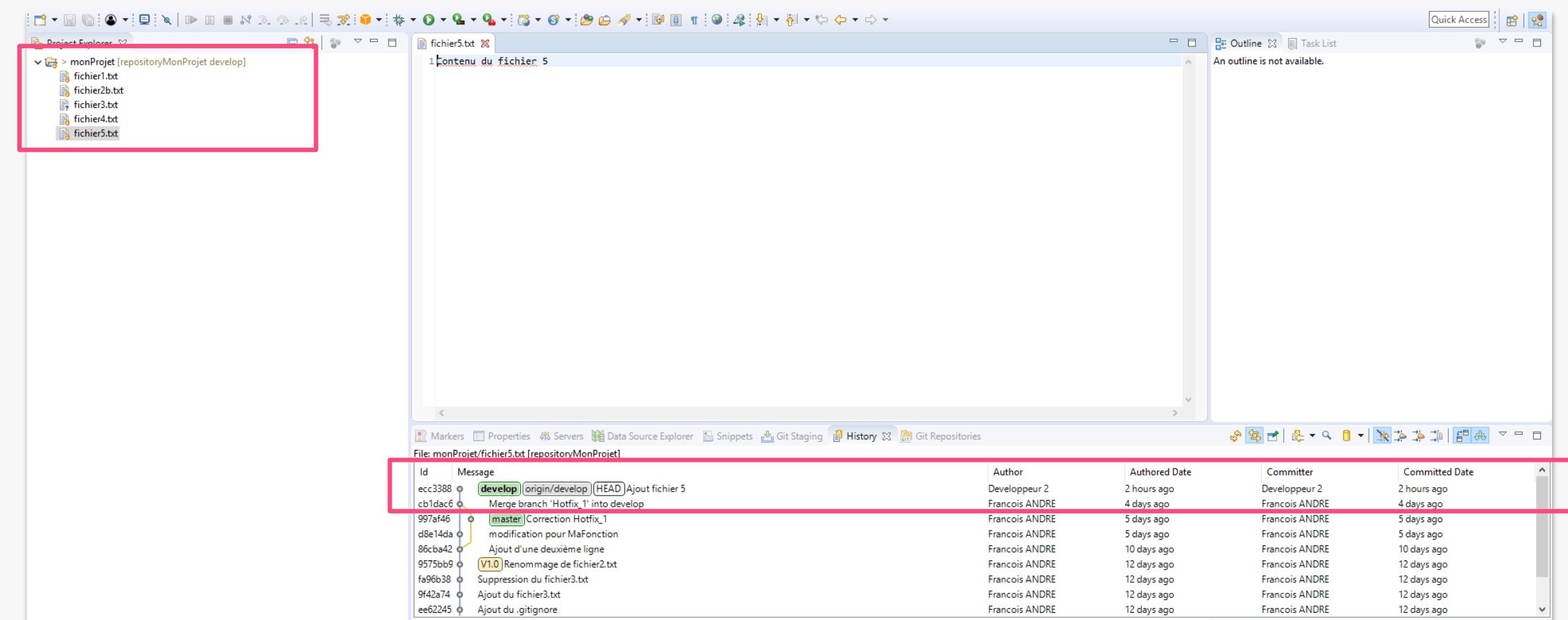
# Récupération du commit (pull)

- Dans le workspace du premier développeur récupérez le nouveau contenu du dépôt distant:
  - Via le menu contextuel du projet, sélectionnez **Team > Pull**
  - Fermez la fenêtre de confirmation

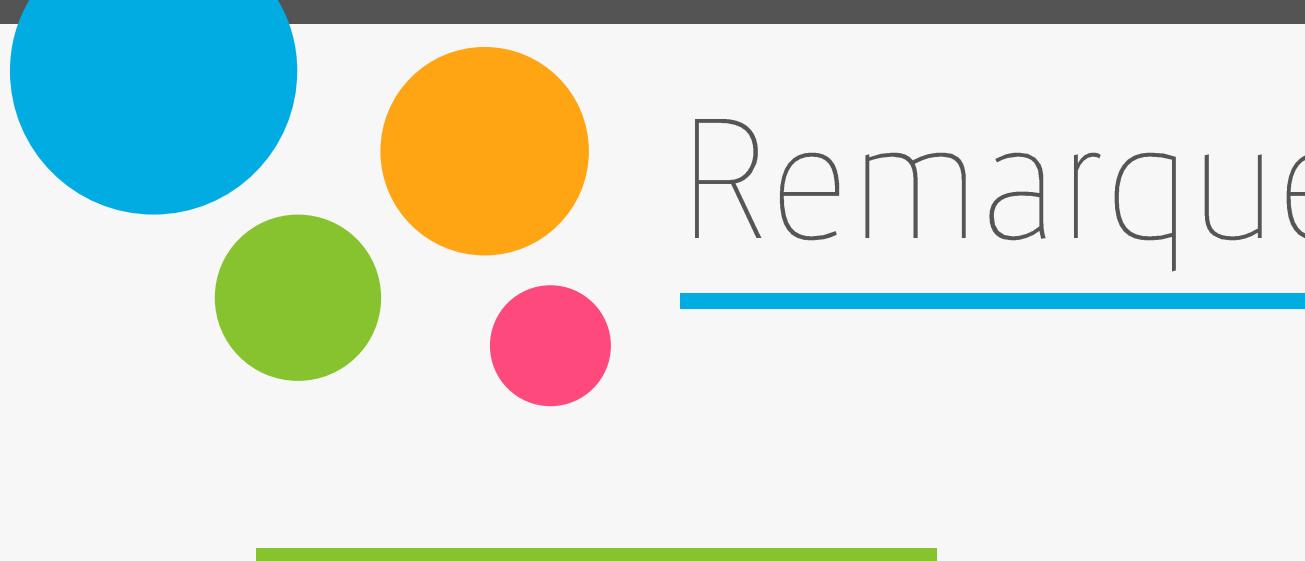


# Récupération du commit (pull)

- Le contenu du commit est désormais récupéré sur le poste du premier développeur



- Git permet d'identifier les développeurs à l'origine de chaque modification
- Ce cycle de pull/commit/push est l'élément fondamental du travail avec Git

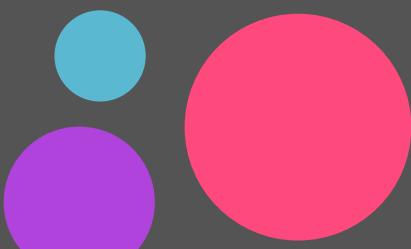


# Remarques sur les remotes

---

- Il est possible d'avoir plusieurs remotes pour un même projet. Cela permet de travailler directement entre sous-équipes.
- Avec les *remotes*, les développeurs vont être confrontés aux problèmes classiques de fusions, vus précédemment
- Des nouveaux problèmes vont apparaître : suppression de branches distantes, ...

# Conclusion





## Pour aller plus loin

---

**Git** propose des fonctionnalités avancées comme le Rebase (modification de l'historique), les hooks, ...

Ces ressources permettent d'avancer dans sa maîtrise de Git.

- **Pro Git** (par Scott Chacon)

- Ouvrage de référence
- Disponible gratuitement en français : <https://git-scm.com/book/fr/v2>  
(différents formats: pdf, html, epub...)

- **Tutoriel(s)**

- <http://www.grafikart.fr/formations/git>

# Merci

Des questions?

