



Global Knowledge®

LE GESTIONNAIRE DE SOURCES

Présentation

Qui suis-je ?



Présentation

Qui êtes-vous ?



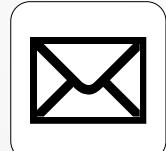
Logistique



Pause en milieu de session



**Vos questions sont les bienvenues.
N'hésitez pas !**



Feuille d'évaluation à remettre remplie en fin de session

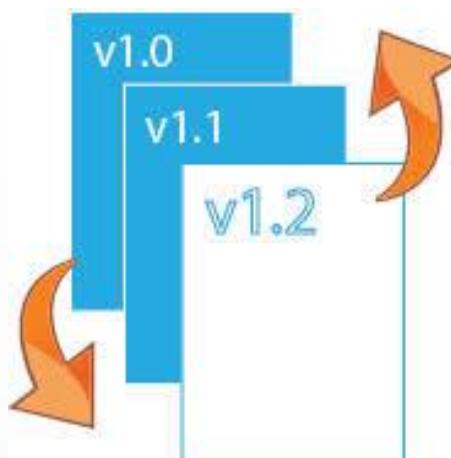


**Merci d'éteindre
vos téléphones**

Agenda

➤ Agenda

- Les objectifs de la gestion de sources
- Les fonctionnalités
- Les différents gestionnaires de sources
- Les problématiques d'intégration des changements



Les objectifs de la gestion de sources

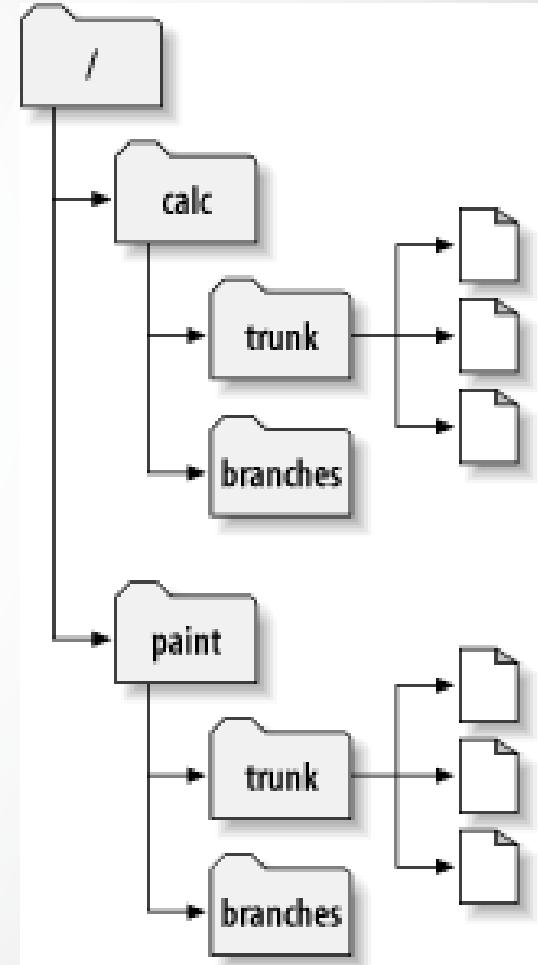
- Gérer les activités de lecture, écriture et fusion sur les sources du projet.
- Conserver un historique des modifications apportées aux sources:
 - Par qui, quand, quoi ?
- Permettre de revenir en arrière en cas de besoin.
- Permettre de travailler en équipe sur les mêmes sources d'un projet.
 - Gérer les accès concurrents.
 - Gérer les conflits.
- Permettre de travailler simultanément sur plusieurs versions d'un logiciel.

Les objectifs de la gestion de sources

- Permet de retrouver un fichier source supprimé.
- Fonctionne avec tout type de fichier (.txt, .php, .java, .jpg, .exe, ...).
- Garantir la sécurité de la configuration du logiciel:
 - Autorisation.
 - Confidentialité.
 - Intégrité.
 - Disponibilité.

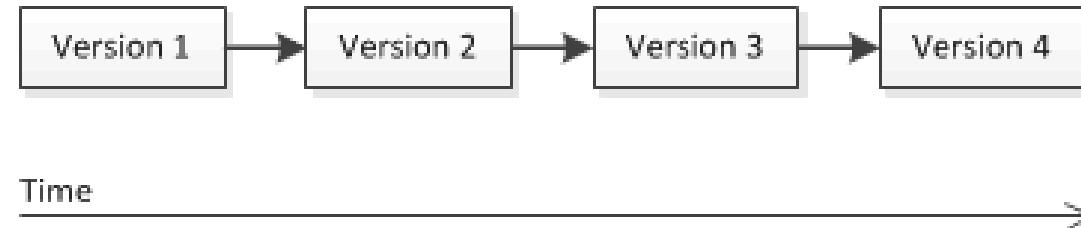
Les fonctionnalités : Repository

- Un dépôt (local ou distant) qui contient toutes les versions des sources du projet.
- Une base de données des changements apportés à la configuration du logiciel.
- Le dépôt a une structure arborescente :

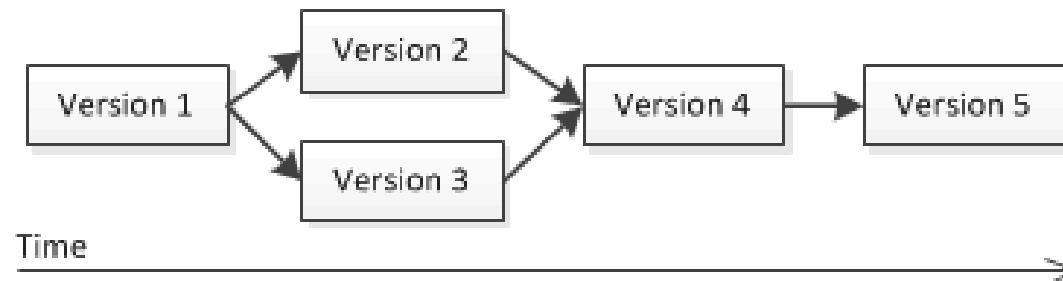


Les fonctionnalités : Version

- Une version ou révision est une instance d'un fichier source qui est strictement distincte des autres instances.
- Les versions peuvent se succéder de manière linéaire dans le temps.

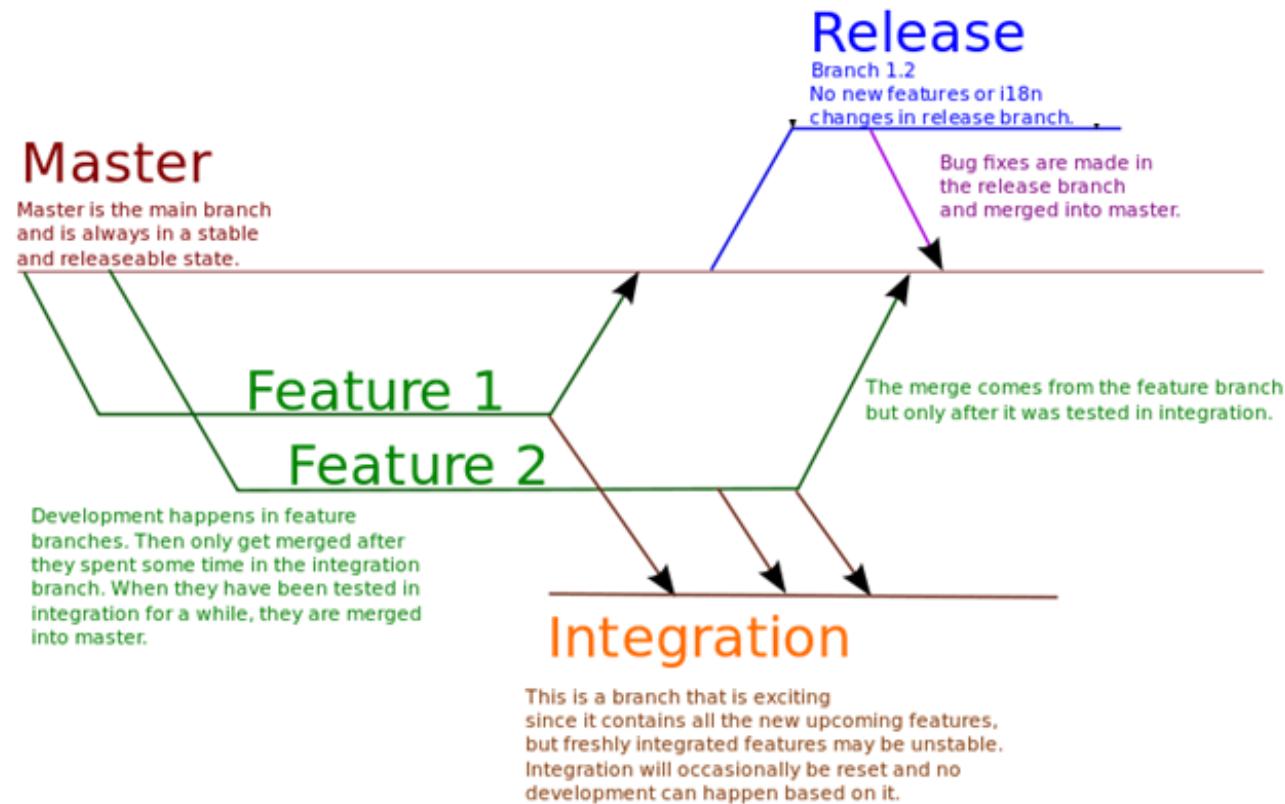


- En cas de modifications concurrentes d'une version, des splits et des merges peuvent avoir lieu.



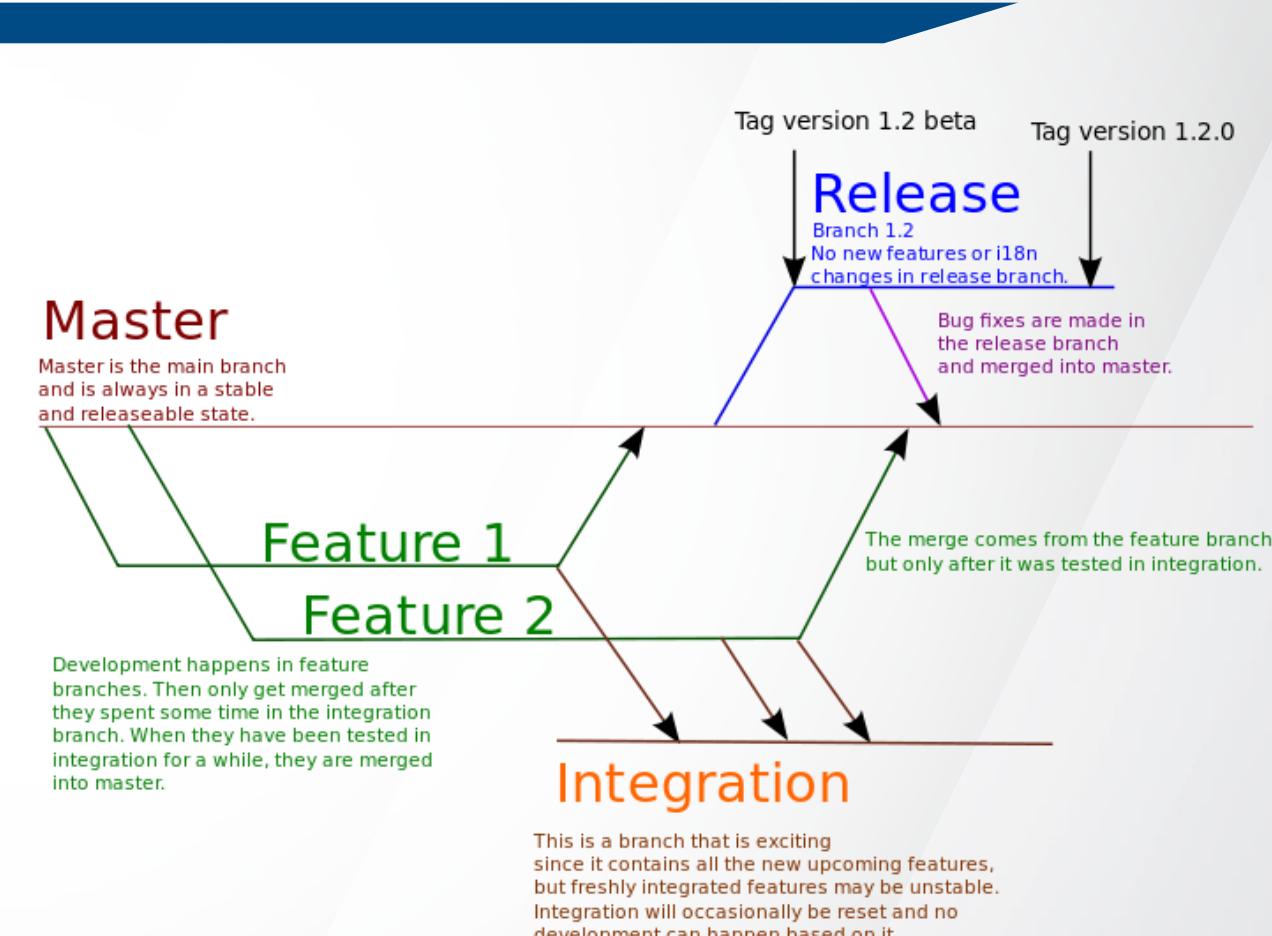
Les fonctionnalités : Branche

- Permet de développer en parallèle plusieurs versions du logiciel.
- Permet de corriger un problème sur une ancienne version du logiciel.
- Permet de fusionner après une divergence.



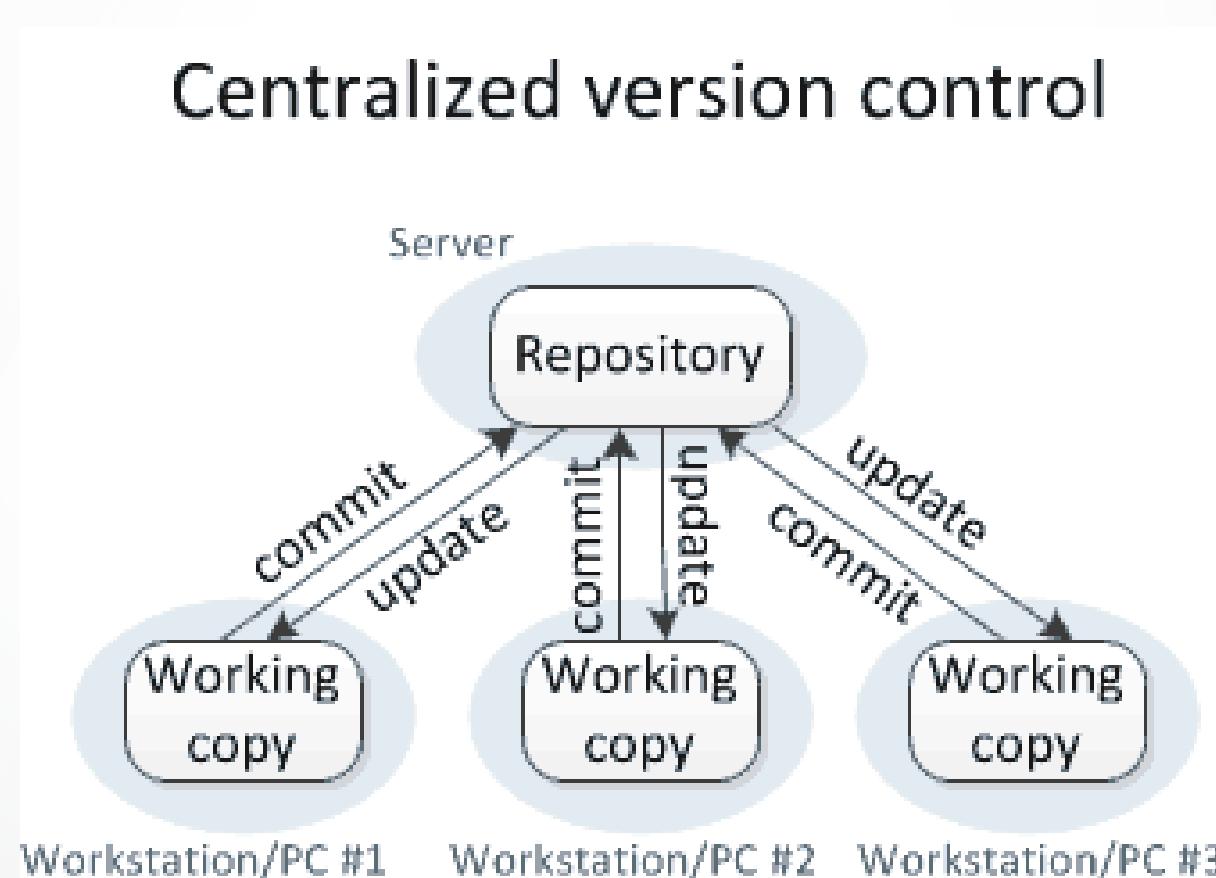
Les fonctionnalités : Tag

- Donner un nom explicite à une version du logiciel pour pouvoir y accéder facilement.
- Permet de définir les versions du projet.
- Permet de nommer des branches.



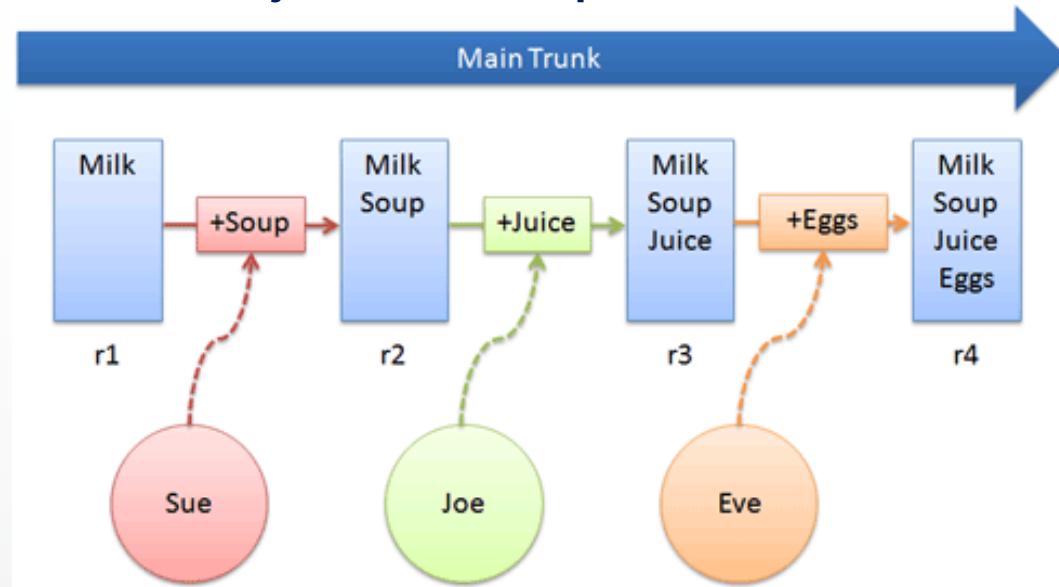
Les différents gestionnaires de sources : Modèle centralisé (1)

➤ Centralized Version Control System – CVCS



Les différents gestionnaires de sources : Modèle centralisé (2)

- Le dépôt est stocké dans un endroit partagé.
- Chaque développeur a une copie de travail du dépôt.
- Utilise le mode de verrouillage pour gérer les développements concurrents sur un même fichier source.
- Il est nécessaire d'avoir la connexion au dépôt pour commiter ses modifications ou mettre à jour sa copie de travail.



Les différents gestionnaires de sources : Modèle centralisé (3)

- Avantages :
 - Technologie éprouvée.
 - Structure simple.
 - Gestion et utilisation simples à mettre en œuvre.
 - Largement disponible (IDE, Forges).
- Faiblesses :
 - Très sensible aux pannes, impossible de travailler hors connexion.
 - Inadapté aux très grands projets, le temps de mise à jour du dépôt est trop long.

Les différents gestionnaires de sources : Modèle centralisé – Exemple Subversion

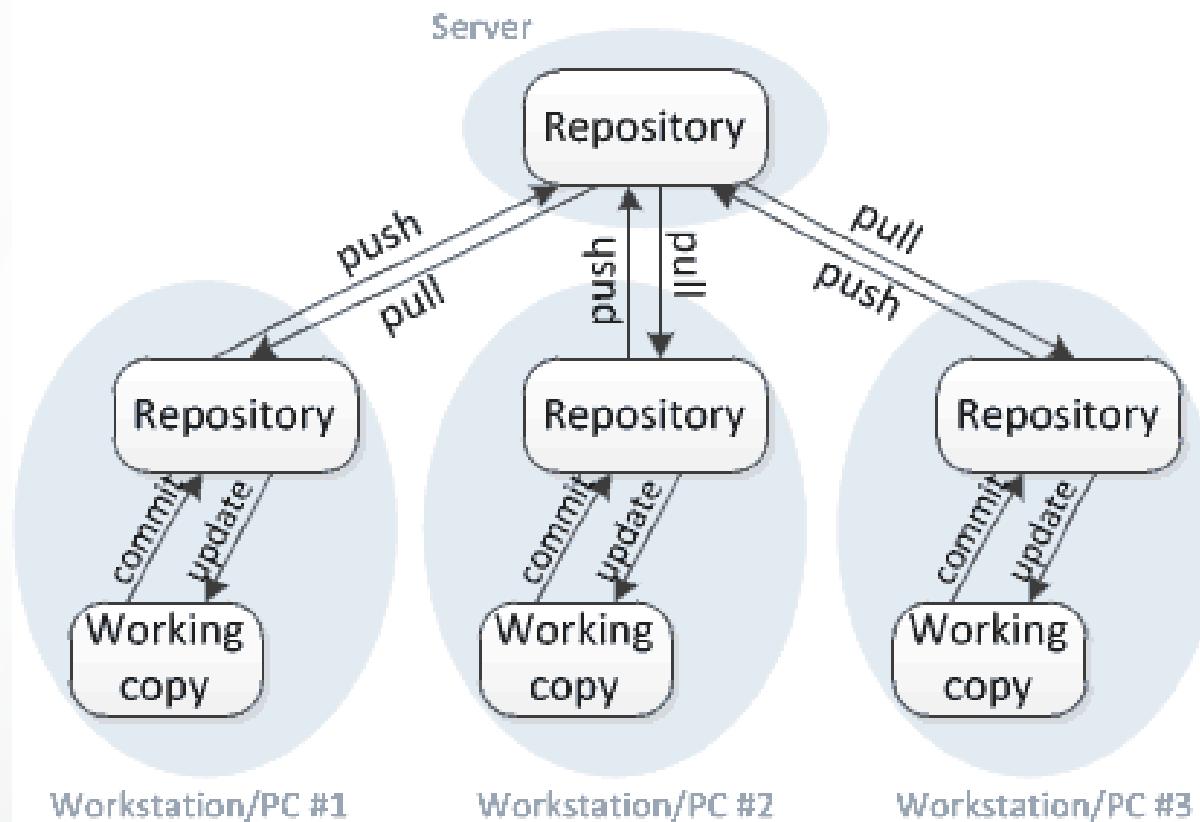


- Créé en 2000 pour remplacer CVS.
- Logiciel libre.
- Multi-plateforme (linux, windows, MacOS, ...).
- Documentations très riches, forums actifs.
- Implémente des protocoles réseaux sécurisés (HTTPS) .
- Interfaces graphiques.
- Intégré dans plusieurs IDE tel que Eclipse.

Les différents gestionnaires de sources : Modèle distribué (1)

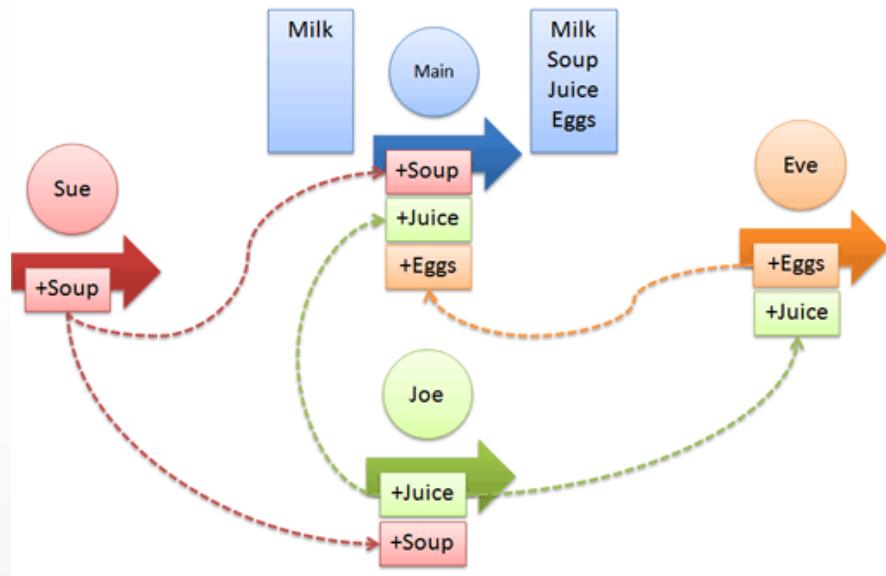
➤ Distributed Version Control System – DVCS

Distributed version control



Les différents gestionnaires de sources : Modèle distribué (2)

- Chaque développeur a sa copie avec des branches et des tags privés.
- Utilise le mode de fusion pour gérer les développements concurrents sur un même fichier source.



Les différents gestionnaires de sources : Modèle distribué (3)

➤ Avantages :

- Travail déconnecté, accès aux dépôts en local sur le poste de travail du développeur.
- Moins sensible aux pannes.
- Rapidité, le réseau n'est utilisé que pour les opérations de synchronisation.
- Branches privées.
- Modèle de développement autorisé très souple.

➤ Faiblesses :

- Gestion et utilisation plus compliquées.
- Risque de divergence, peut devenir très complexe structurellement.

Les différents gestionnaires de sources : Modèle distribué – Exemple Git

- Créé en 2005 par Linus Torvalds pour la gestion des sources du noyau Linux.
- Logiciel libre.
- Multi-plateforme (linux, windows, MacOS, ...).
- Documentations très riches, forums actifs.
- Implémente des protocoles réseaux sécurisés (HTTPS) .
- Interfaces graphiques.
- Intégré dans plusieurs IDE tel que Eclipse.

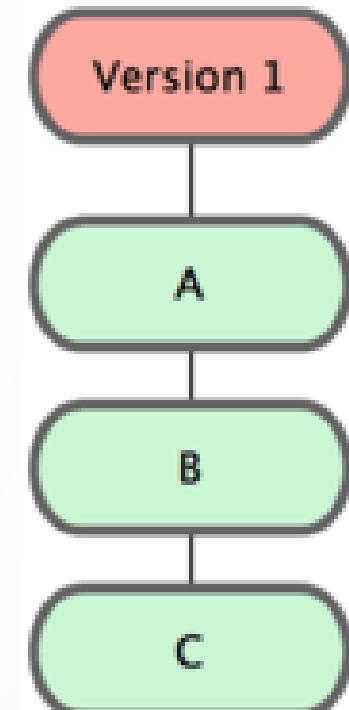


Gestion de sources avec Git: Le fonctionnement de base

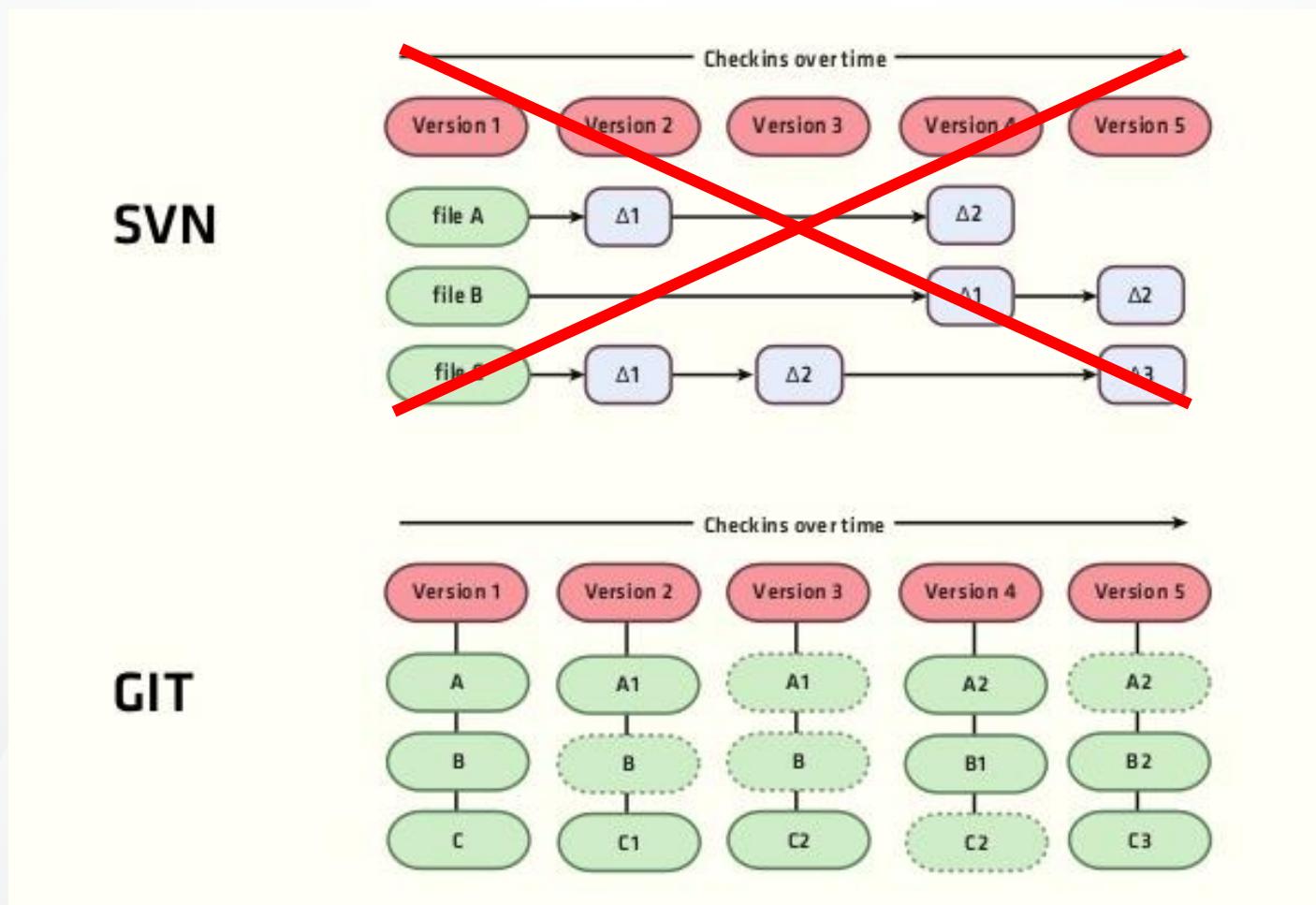
- Git stocke pour chaque état du projet un instantané de tous les fichiers.
 - Un fichier non modifié depuis le dernier état est stocké comme un pointeur vers sa version précédente.
- La plupart des opérations de Git ne nécessitent que des fichiers et des ressources locaux. Pour la quasi-totalité de ces opérations, aucun appel réseau n'est nécessaire.
- Une somme de contrôle (checksum) vérifie chaque état par une empreinte SHA-1 et permet ainsi d'assurer l'intégrité des données.
 - Toutes les corruptions de fichier ou erreurs de transfert sont détectables.
 - Somme de contrôle: 80f2a4d12d5cc8cefefaf9f0e015f2d11ae43b35
- La réversibilité des opérations est assurée avec Git.

Gestion de sources avec Git: Le fonctionnement de base

- Un projet est composés de **3 fichiers** A, B et C.
- Le **commit initial** permet de checkiner les fichiers A, B et C. Il s'agit de la première version du projet.
- Les fichiers A et C sont modifiés et les changements sont checkinés. Les versions **A1** et **C1** sont alors créées. Il s'agit de la deuxième version du projet.
- Git a alors créé un instantané du projet composé des fichiers qui ont été modifiés, A et C. Les fichiers qui n'ont pas subit de modification **ne sont pas stockés**, B.

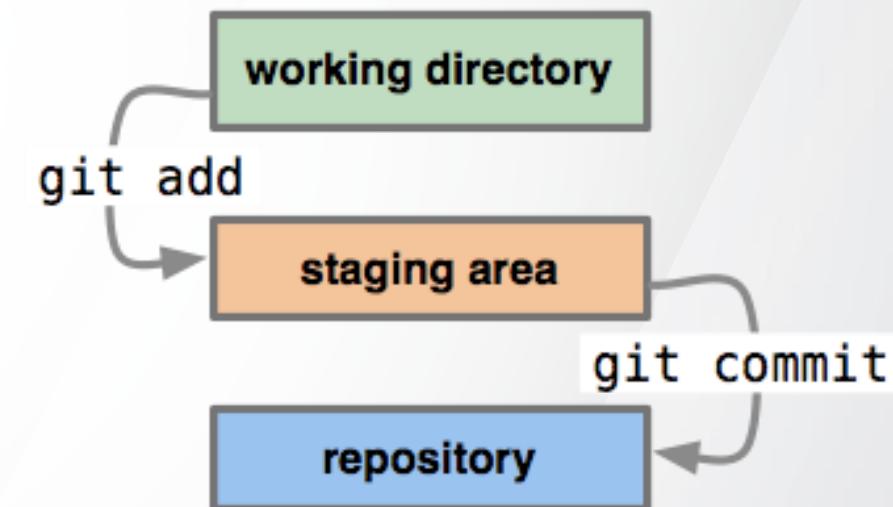


Gestion de sources avec Git: Le fonctionnement de base



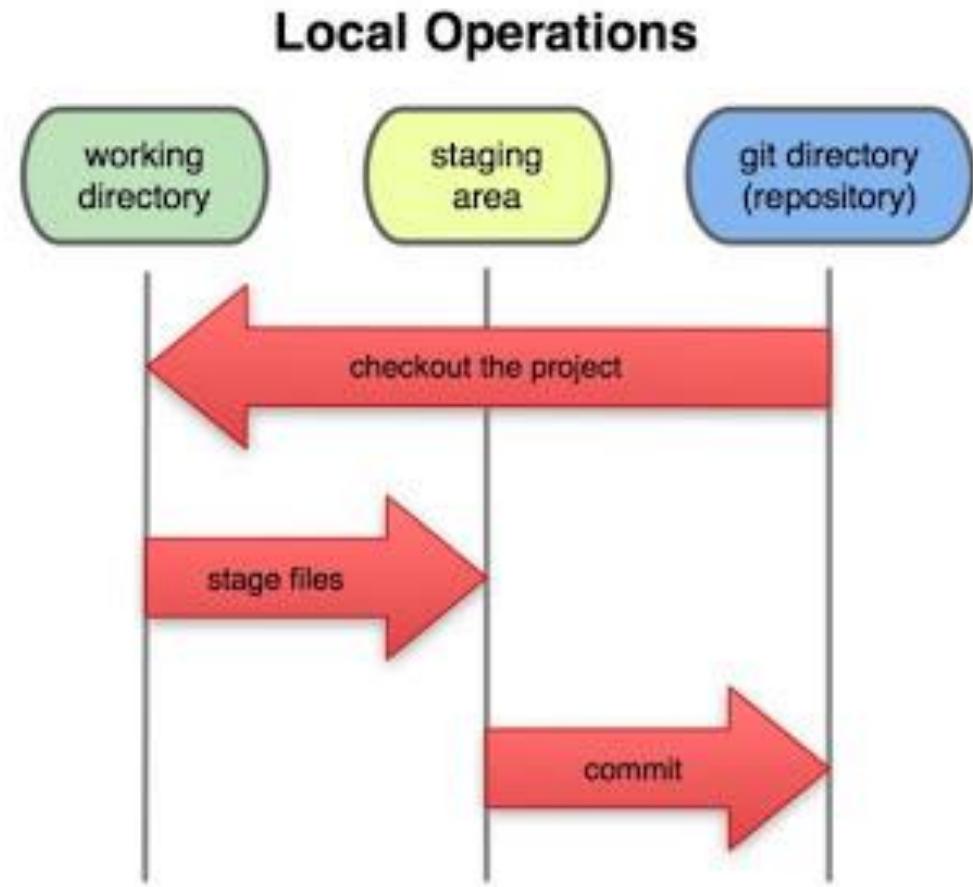
Gestion de sources avec Git : Workflow

- Le répertoire de travail (**working directory**) est le répertoire dans lequel se fait les développements.
 - Ce répertoire contient des fichiers **suivis** par Git ainsi que des fichiers non-suivis.
- La zone d'index (**staging area**) contient les fichiers qui feront partie du **prochain commit**.
- Le répertoire Git (**repository**) est l'endroit où Git stocke les métadonnées et la base de données des objets du projet.
 - Ce répertoire contient tous les fichiers suivi et tout l'historique du projet.

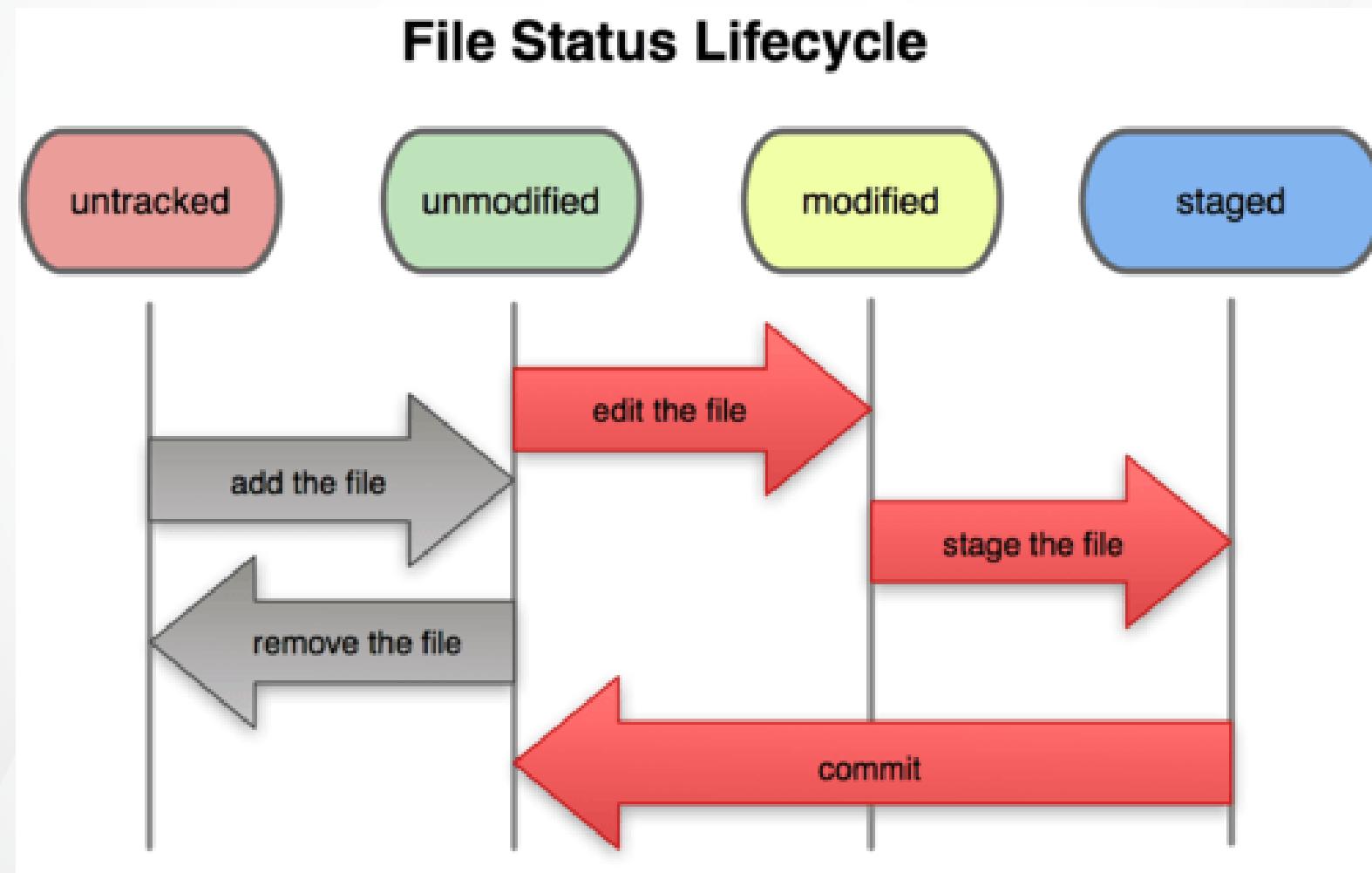


Gestion de sources avec Git : Workflow

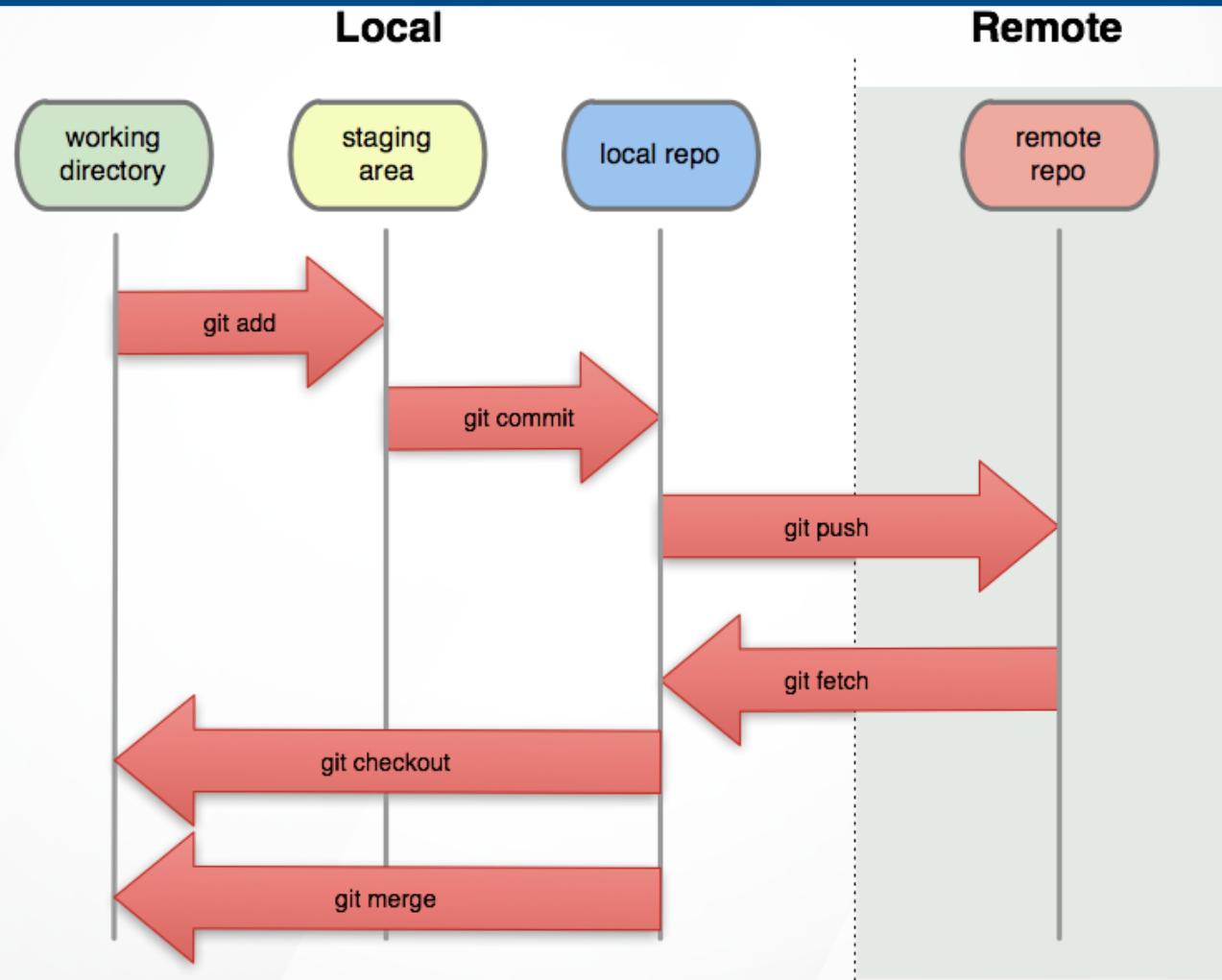
- Modification des fichiers dans le répertoire de travail (working directory).
- Indexation du contenu modifié (git add), en vue du prochain instantané.
- Validation (commit) basculant les modifications dans le répertoire Git (repository).



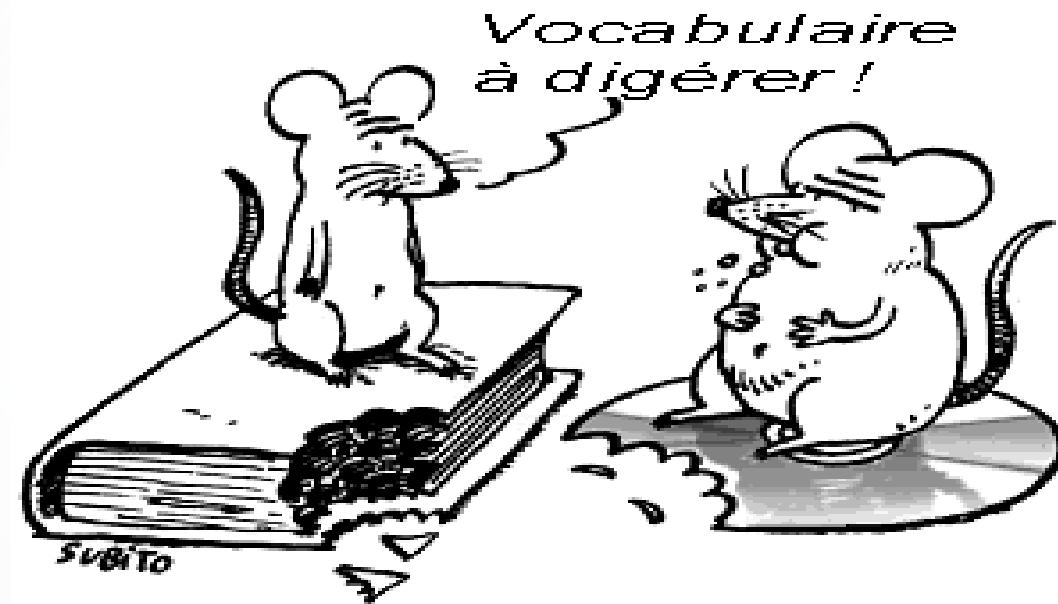
Gestion de sources avec Git : Workflow



Gestion de sources avec Git : Workflow



Gestion de sources avec Git : Vocabulaire Git



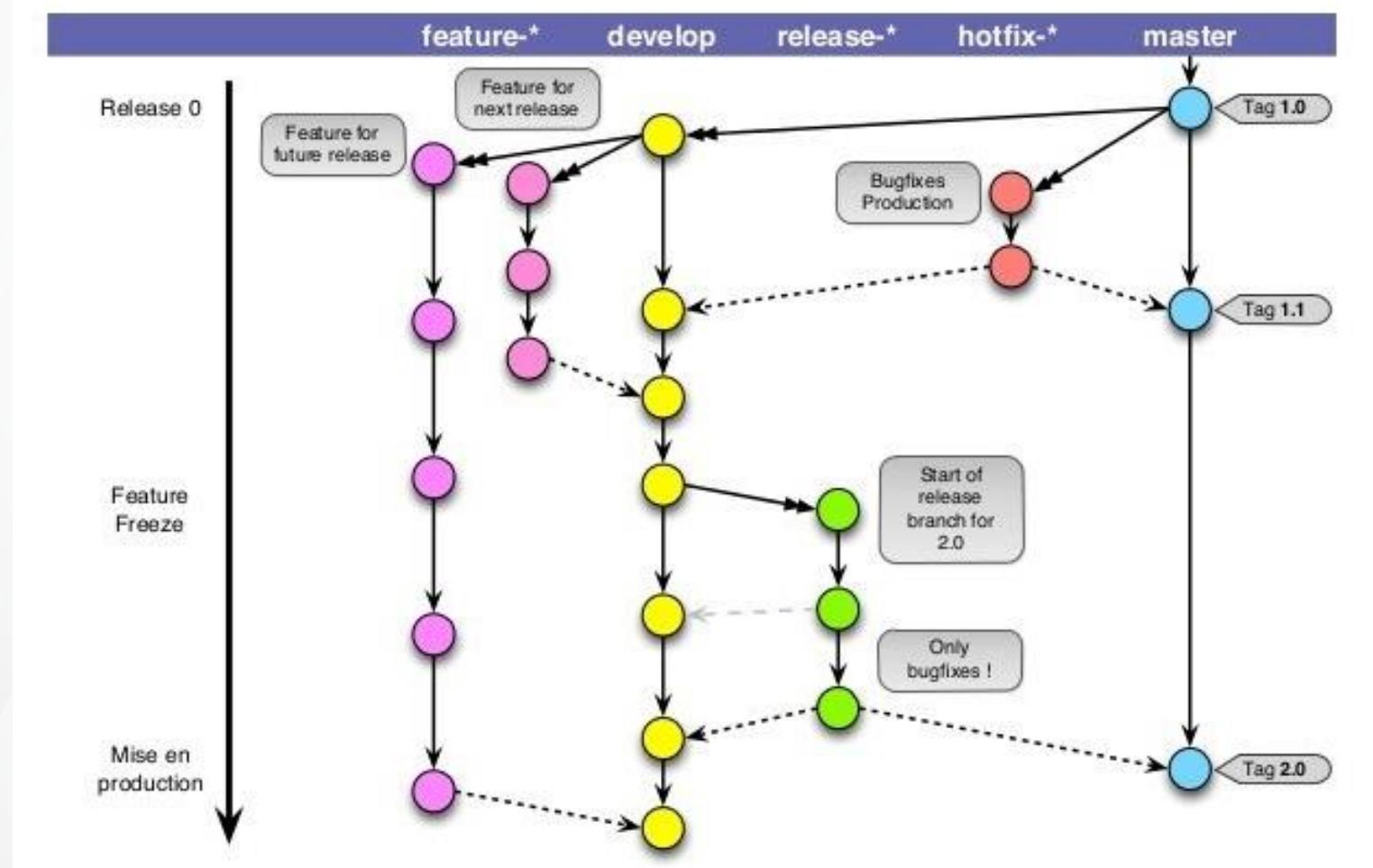
Gestion de sources avec Git : Vocabulaire Git

- **Ajout à l'index**: insérer les changements dans l'index Git afin d'indiquer qu'ils devront être versionnés.
- **Commit**: consigner les changements dans le dépôt Git local.
- **Reset**: annuler de manière irréversible des changements effectués.
- **Push**: envoyer les nouvelles modifications (commits) vers le dépôt distant.
- **Pull**: récupérer des nouveaux commits depuis le dépôt distant et mettre à jour l'espace de travail.
- **Fetch**: récupérer des nouveaux commits depuis le dépôt distant.

Gestion de sources avec Git : Vocabulaire Git

- **Merge**: créer un nouveau commit ayant pour parents 2 branches fusionnées.
- **Rebase**: déplacer une suite de commits et la référence associée vers un nouvel emplacement.
- **Cherry pick**: récupérer les modifications apportées par un ou plusieurs commits et les réappliquer à un nouvel emplacement.
- **Stash**: mettre de coté le travail en cours avant de changer de branche.
- **Branche**: référence évoluant lors d'un commit.
- **Etiquette**: référence fixes permettant d'identifier facilement un commit particulier.

Gestion de sources avec Git : Vocabulaire Git



Les commandes



git init

- La commande git init crée un dépôt Git.
- Elle permet de convertir un projet existant, sans version en un dépôt Git ou d'initialiser un nouveau dépôt vide.
- La plupart des autres commandes Git ne sont pas disponibles hors d'un dépôt initialisé.
- Il s'agit donc généralement de la première commande que vous exécuterez dans un nouveau projet.

git init --bare

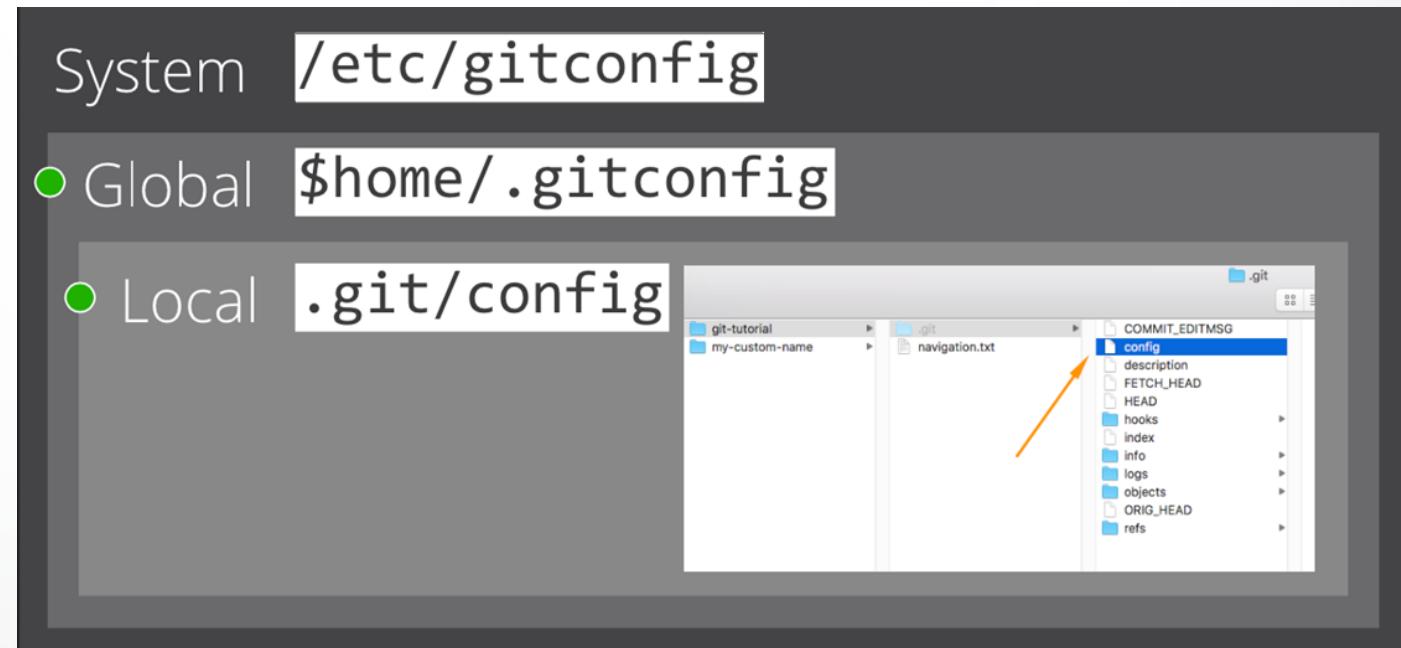
- Le flag **--bare** crée un dépôt qui ne dispose pas de répertoire de travail, rendant impossibles les éditions de fichiers et les commits de changements dans ce répertoire.
- Vous créeriez un **dépôt brut** depuis lequel exécuter les commandes git push et git pull, mais dans lequel vous **ne pourriez jamais faire de commit**.

git config

- Le cas d'usage le plus élémentaire pour git config consiste à l'appeler avec un nom de configuration, qui affichera la valeur définie pour ce nom.
- Les noms de configuration sont des chaînes délimitées par des points et composées d'une « section » et d'une « clé » en fonction de leur hiérarchie.
- Par exemple : user.email
 - Dans cet exemple, « email » est une propriété enfant du bloc de configuration de l'utilisateur.
 - Cela renvoie l'adresse e-mail configurée, le cas échéant, que Git associera à des commits créés en local.

git config : Niveaux de configuration

- La commande git config peut accepter des arguments de manière à spécifier le niveau de configuration sur lequel agir.
- Les niveaux de configuration suivants sont disponibles :
 - **--local**
 - **--global**
 - **--system**



git config : Niveaux de configuration

--local

- Par défaut, git config écrira à un niveau local si aucune option de configuration n'est transmise.
- La configuration au niveau local s'applique au contexte dans lequel le dépôt git config est appelé.
- Les valeurs de configuration locales sont stockées dans un fichier situé dans le répertoire .git du dépôt : .git/config

git config : Niveaux de configuration

- **--global**
- La configuration au niveau global varie en fonction de l'utilisateur, ce qui signifie qu'elle s'applique à un utilisateur du système d'exploitation.
- Les valeurs de la configuration globale sont stockées dans un fichier situé dans le répertoire de base d'un utilisateur.
~/.gitconfig sur les systèmes Unix et C:\Users\\.gitconfig sous Windows

git config : Niveaux de configuration

- **--system**
- La configuration de niveau système est appliquée à une machine complète. Cela concerne tous les utilisateurs d'un système d'exploitation et tous les dépôts. Le fichier de configuration de niveau système réside dans un fichier gitconfig extrait du chemin d'accès à la racine du système.
- Il se trouve dans /etc/gitconfig sur les systèmes Unix. Sous Windows, ce fichier se trouve dans C:\Documents and Settings\All Users\Application Data\Git\config sous Windows XP, et dans C:\ProgramData\Git\config sous Windows Vista et les versions plus récentes.

Git diff

Git diff : utilisation

- `$ git diff`
 - Différence entre l'index et les working files
- `$ git diff --staged`
 - Différence entre l'index et le HEAD
- `$ git diff HEAD`
 - Différence entre le HEAD et les working files
- `$ git diff <commit1> <commit2>`

Git diff output

➤ Patch sample

```
diff --git a/foo.c b/foo.c
index 30cf169..8de130c2 100644
--- a/foo.c
+++ b/foo.c
@@ -1,5 +1,5 @@
#include <string.h> int check (char *string) {
    - return !strcmp(string, "ok");
    + return (string != NULL) && !strcmp(string, "ok");
}
```

Git diff output

➤ Patch sample

```
diff --git a/foo.c b/foo.c
```

- `diff --git` : on va utiliser un diff spécifique à git
- `a & b` : indices pour différencier entre les deux fichiers comparés (il s'agit du même fichier)

Git diff output

➤ Patch sample

index 30cf169..8de130c2 100644

- 30cf169..8de130c2 : les ids des blobs qui représentent les différents versions du fichiers
- 100644 : type de fichier + permission : fichier ordinaire + permission 644

Git diff output

➤ Patch sample

--- a/foo.c

+++ b/foo.c

- les signes moins afficheront des lignes dans la version A mais absentes de la version B;
- Les signes plus, lignes manquantes en A mais présentes en B.
- Si l'un d'entre eux est /dev/null ➔ le fichier est en cours de suppression

Git diff output

➤ Patch sample

```
@@ -1,5 +1,5 @@
```

```
#include <string.h> int check (char *string) {  
    - return !strcmp(string, "ok");  
    + return (string != NULL) && !strcmp(string, "ok");
```

```
}
```

- Section de différence, ou “hunk,”
- @@ ➔ hunk in the A and B versions;
- Le hunk commence de 1 et s'étend pour 5 lignes dans les deux versions.
- Les lignes suivantes commençant par espace définissent le contexte:
- Les lignes commençant par signes (-) moins et (+) plus correspondent aux changements entre les versions A et B

Travailler avec les branches

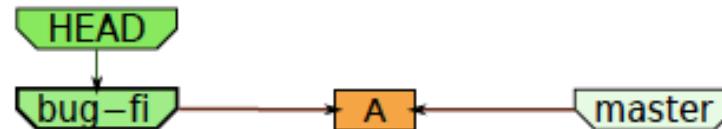
git init / git clone

start at some tree



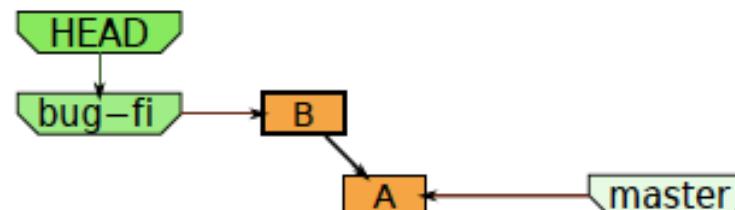
git checkout -b / git branch

```
$ git checkout -b bug-fix
```



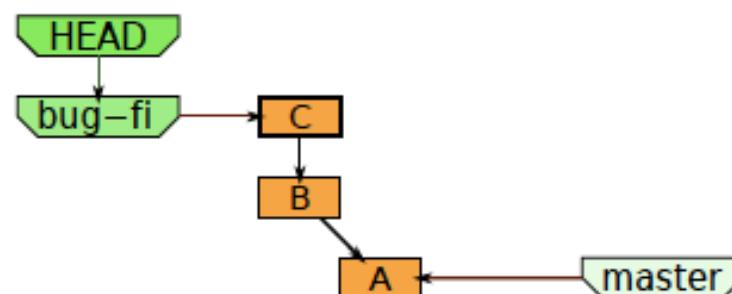
git add .. git commit / git commit -a

```
$ git commit -a -m“B”
```

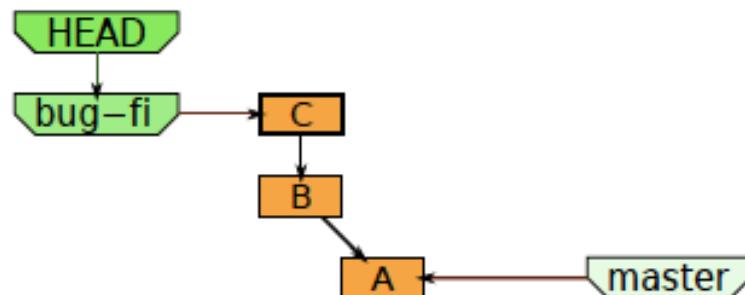


git add .. git commit / git commit -a

```
$ git commit -a -m“C”
```



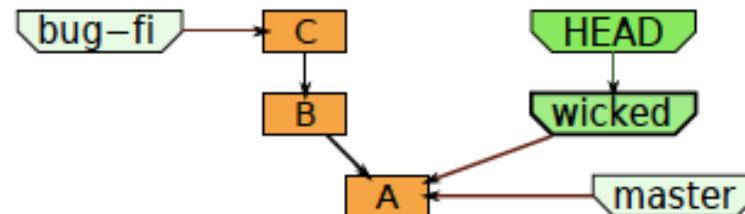
you have a *wicked* idea



git checkout -b / git branch

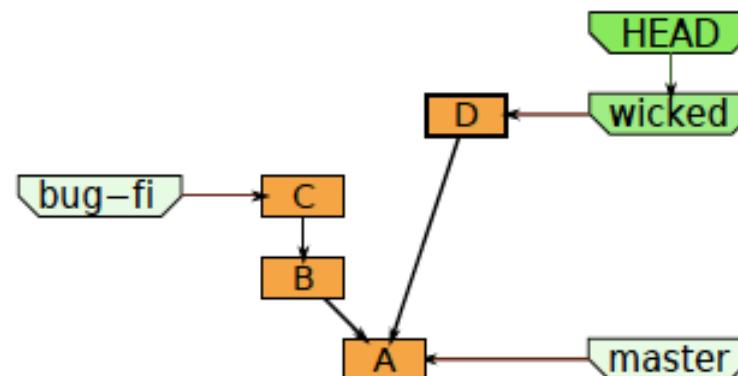
you have a *wicked* idea

```
$ git checkout -b wicked master
```



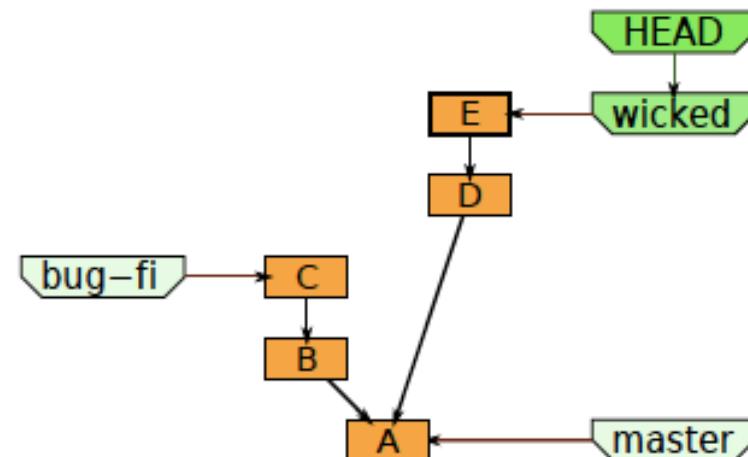
git add .. git commit / git commit -a

```
$ git commit -a -m“D”
```

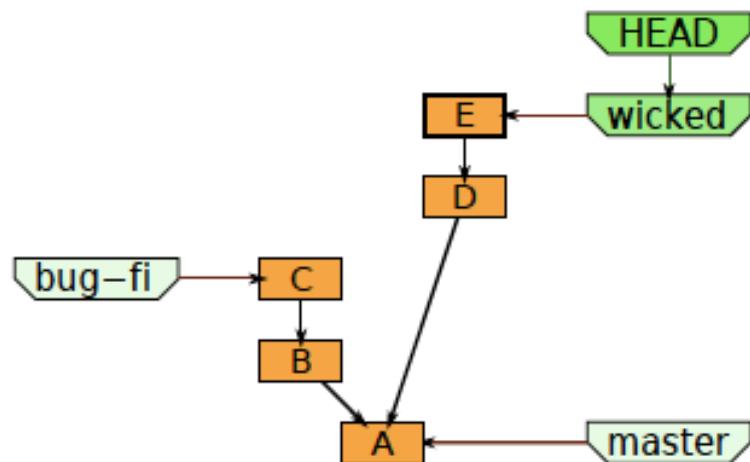


git add .. git commit / git commit -a

```
$ git commit -a -m"E"
```



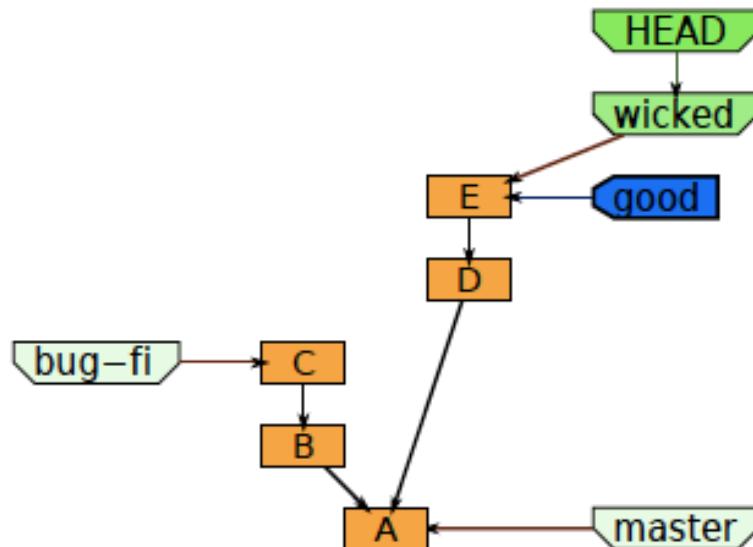
you're getting somewhere



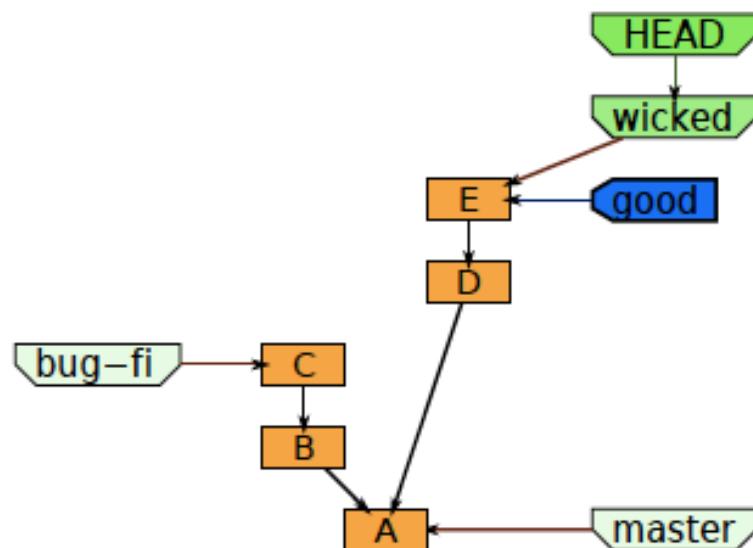
git tag

you're getting somewhere

```
$ git tag -a -m“got somewhere” good
```



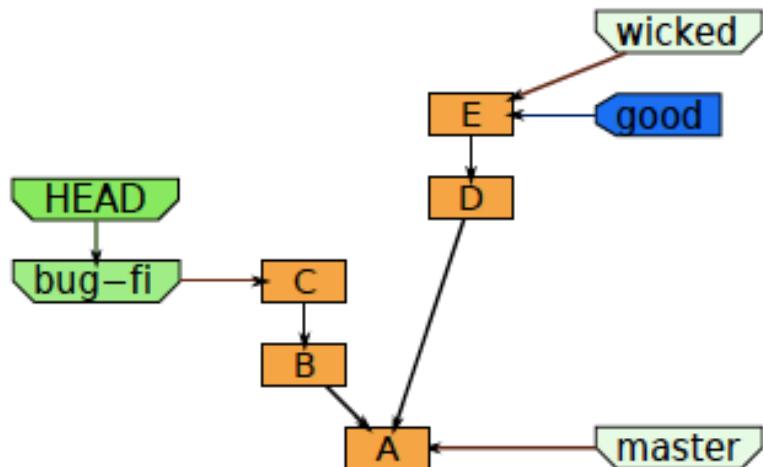
manager asks about the bug



git checkout

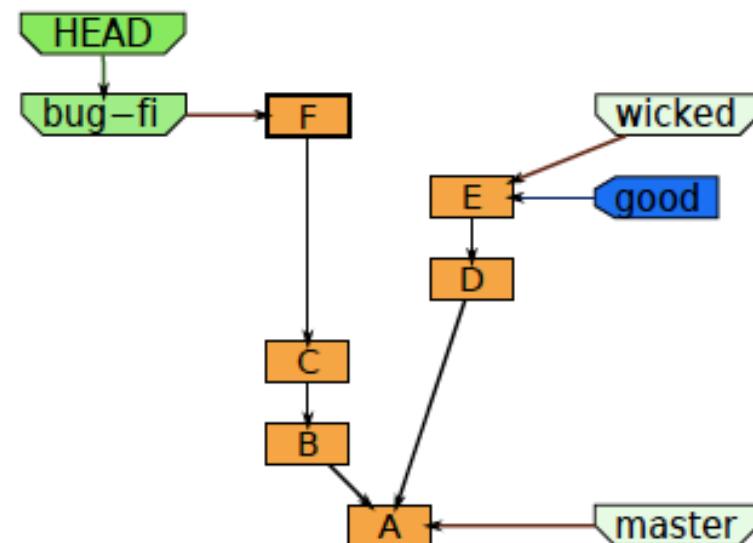
manager asks about the bug

```
$ git checkout bug-fix
```

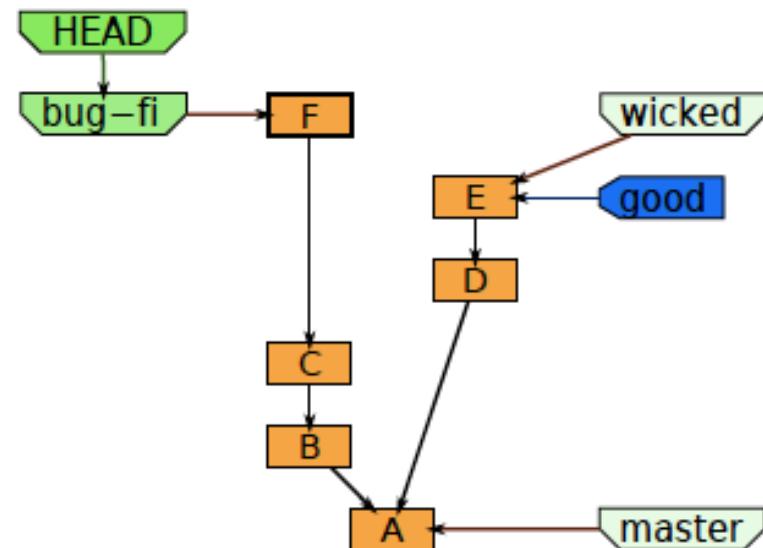


git add .. git commit / git commit -a

```
$ git commit -a -m“F”
```



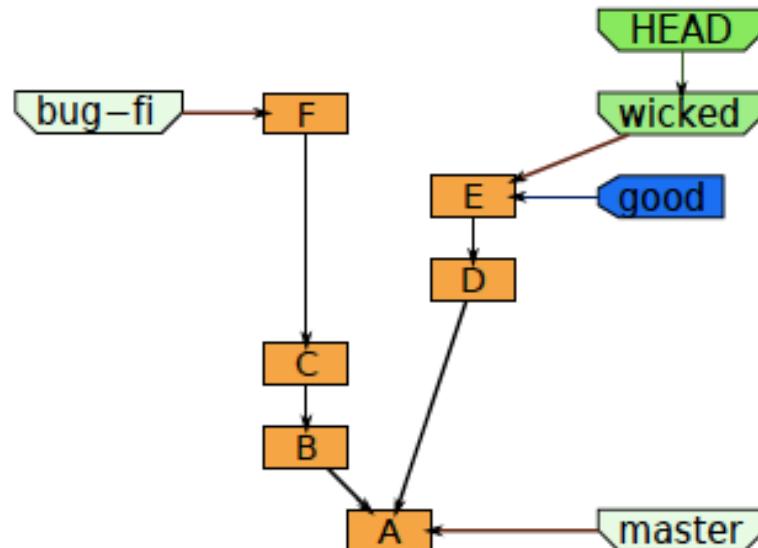
your mind is elsewhere...



git checkout

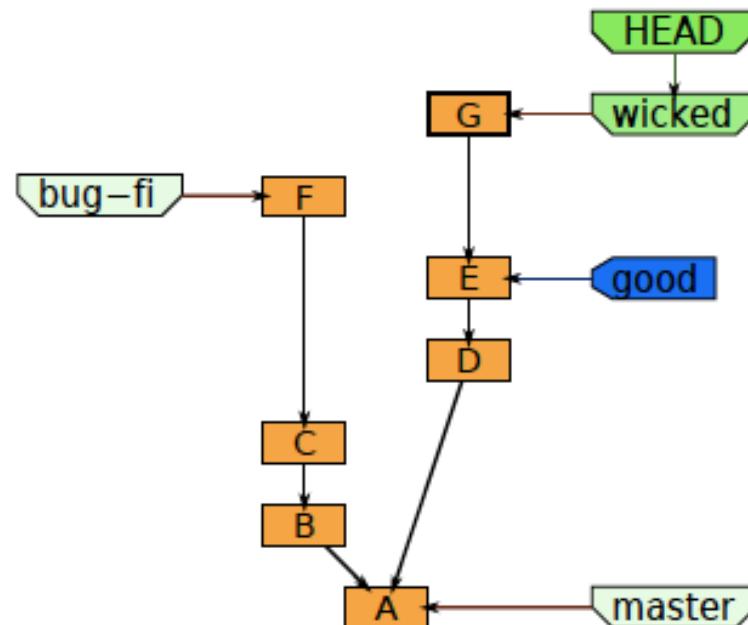
your mind is elsewhere...

```
$ git checkout wicked
```



... so you finish off the *wicked* feature

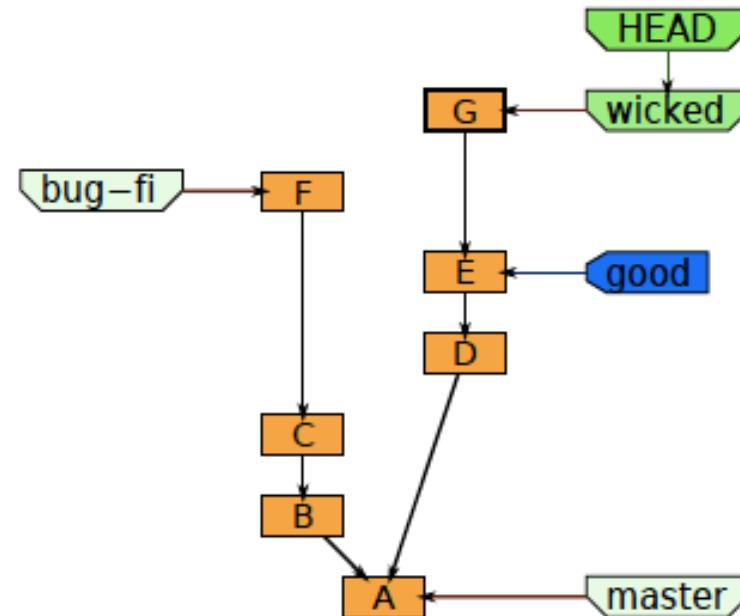
```
$ git commit -a -m“G”
```



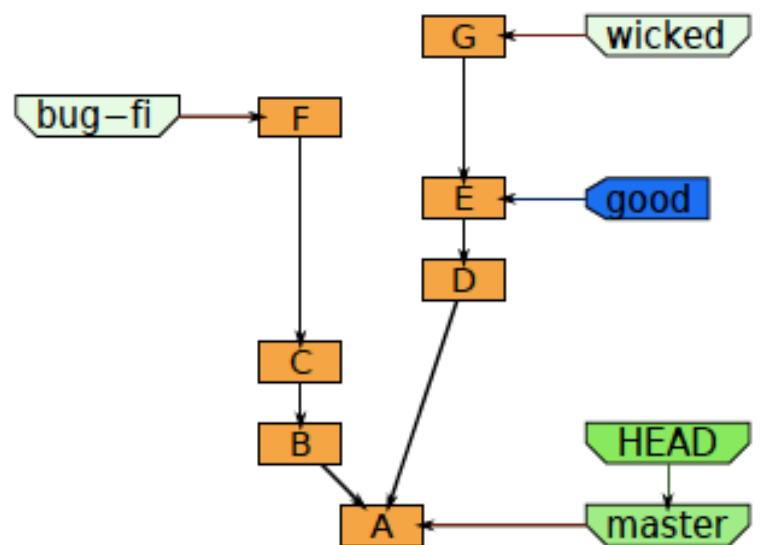
feature's done

bug is fixed

... time to merge

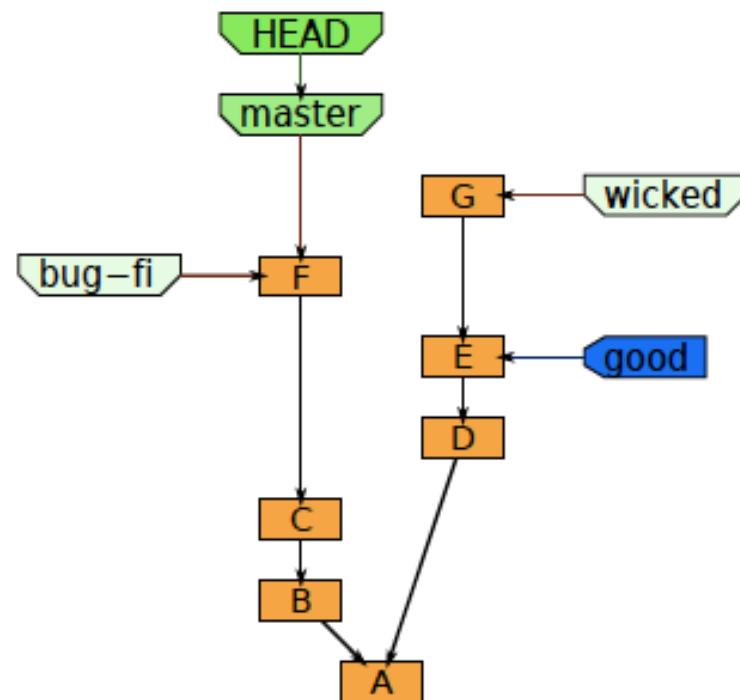


```
$ git checkout master
```



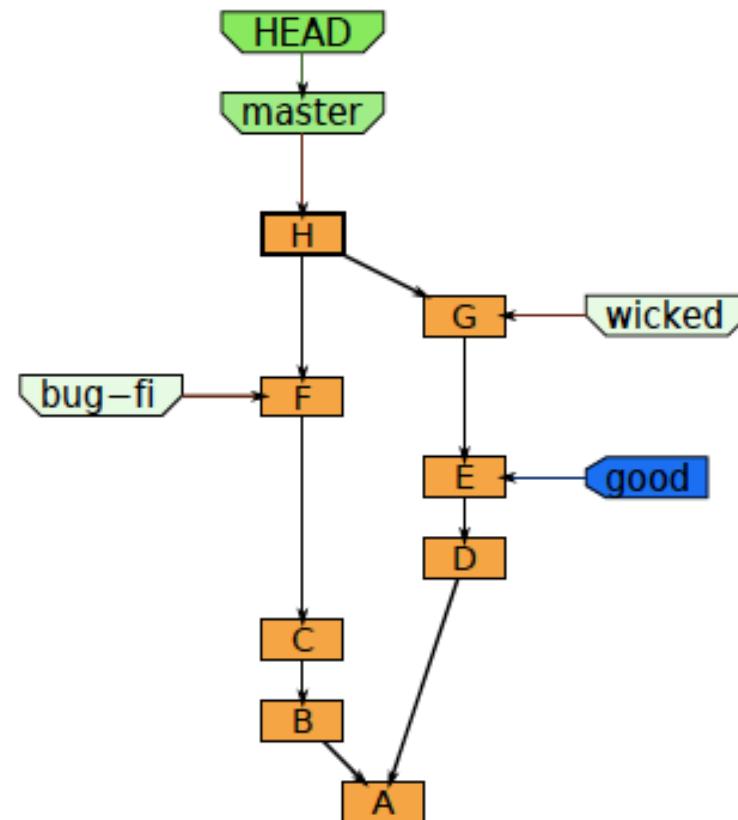
git reset

```
$ git reset --hard bug-fix
```



git merge

```
$ git merge wicked
```



git reflog

- Git garde une trace des mises à jour sur la pointe des branches à l'aide d'un mécanisme appelé journaux de référence ou « reflog ».

Utilisation de Git

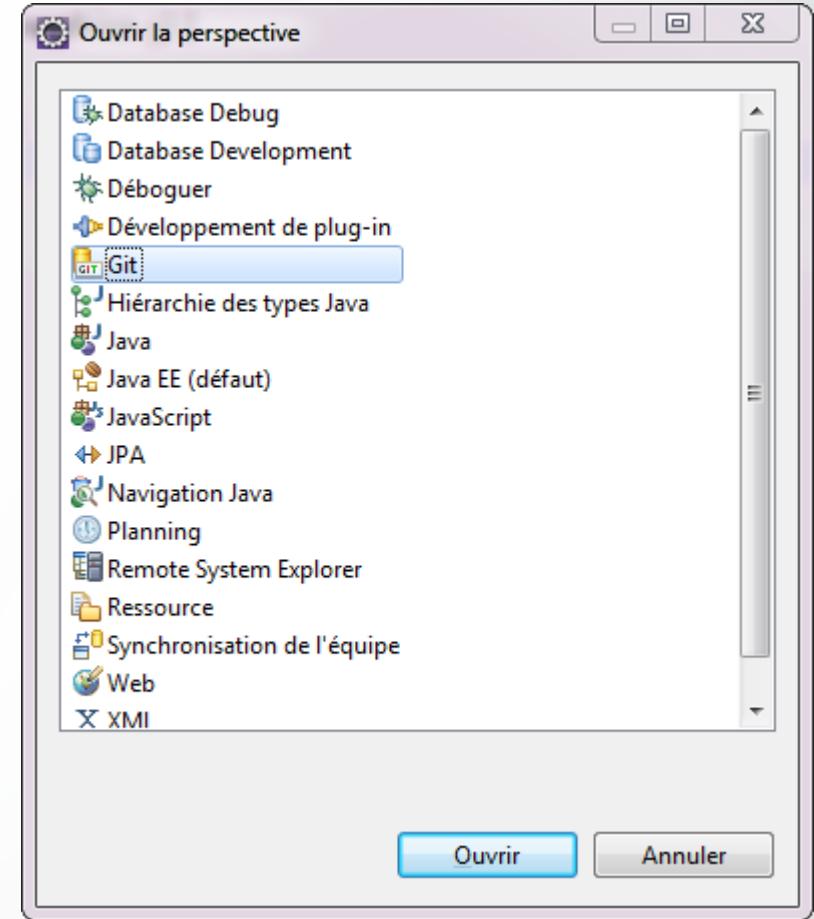


Git et Eclipse



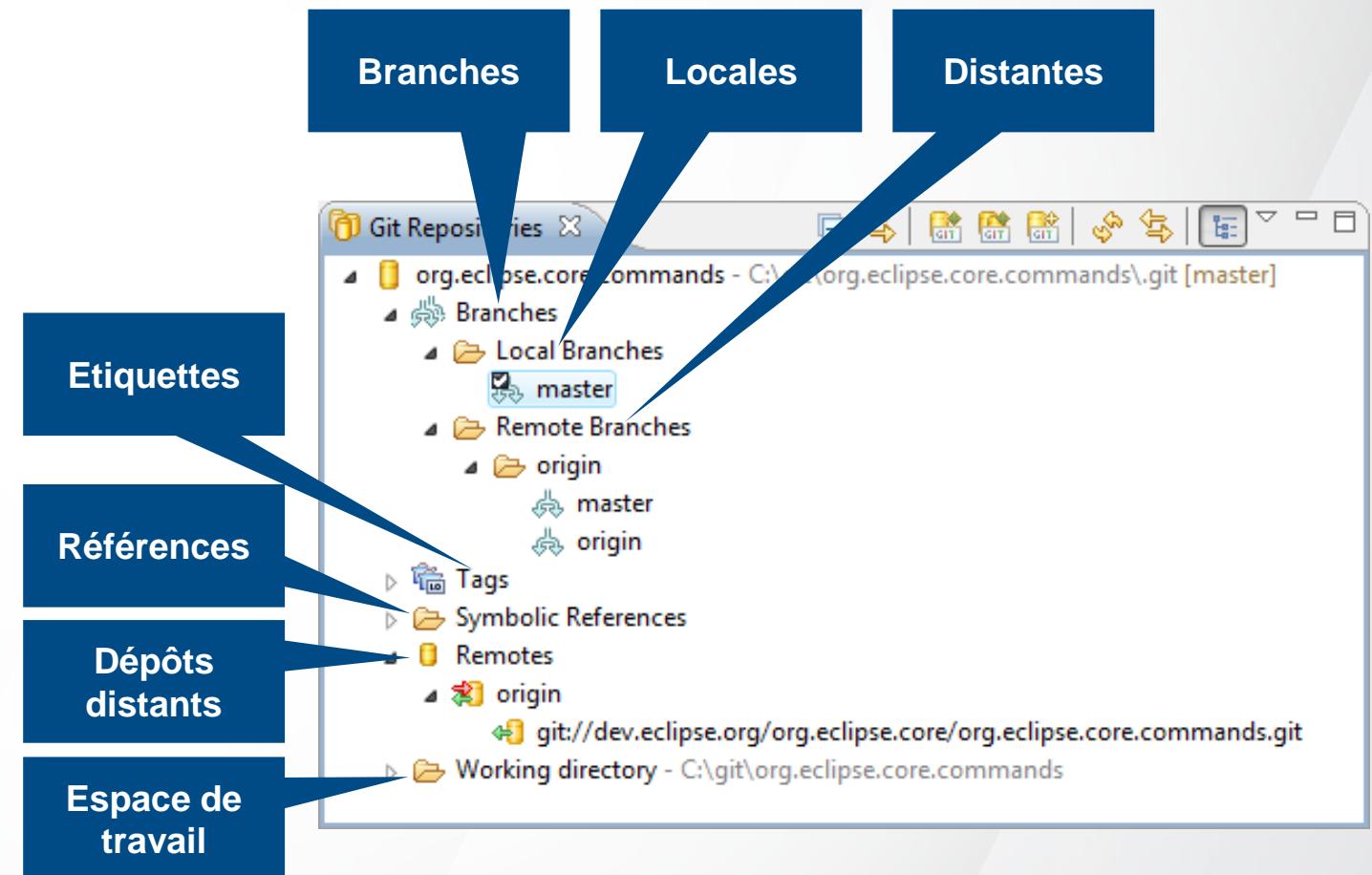
Git et Eclipse

- Implémentation de Git pour Eclipse via le plugin EGit, plugin construit sur la bibliothèque JGit.
- Basé sur JGit, une bibliothèque Java implémentant les commandes Git:
 - Les routines d'accès aux repositories.
 - Les protocoles réseau.
 - Les algorithmes de contrôle de version de base.
- Inclus dans Eclipse depuis la version Juno (eclipse 4.2 sorti en 2013)
- Propose une perspective Eclipse dédiée.



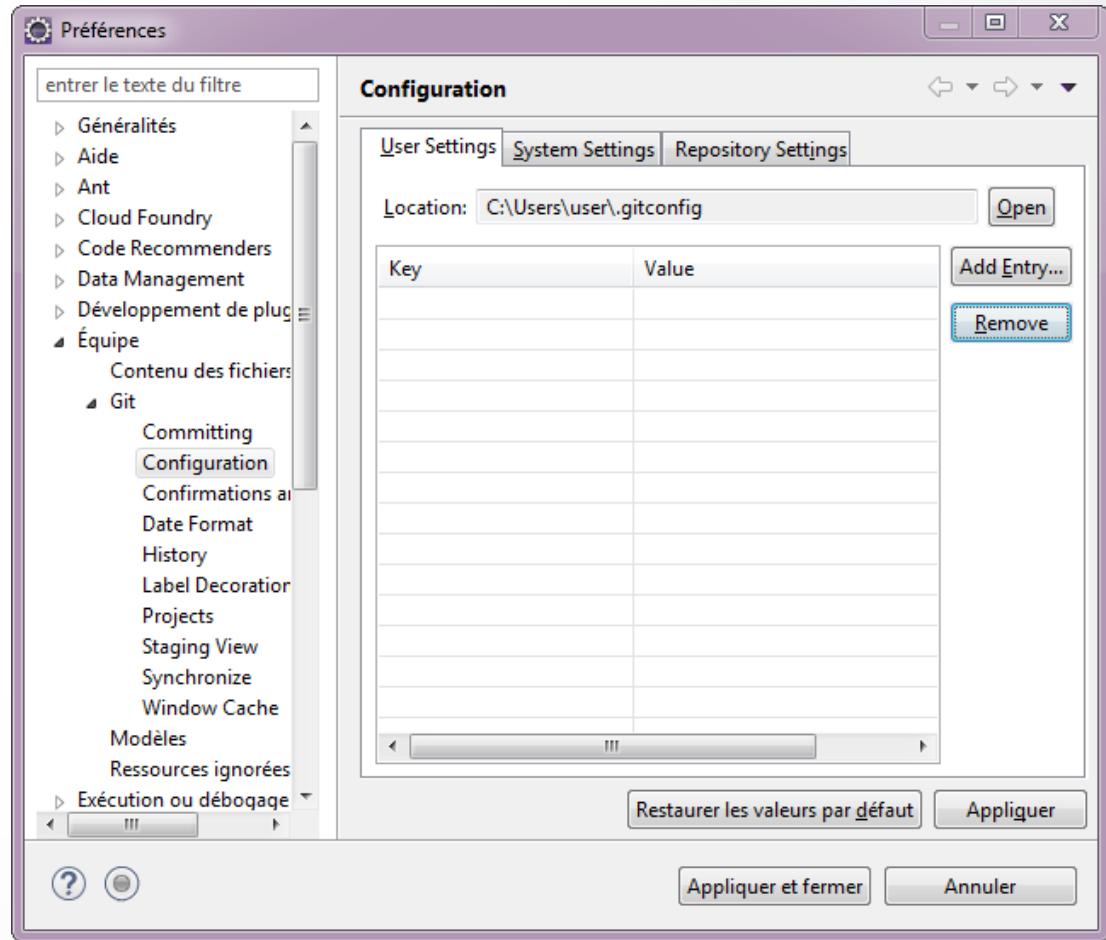
Git et Eclipse : La vue Git Repositories

- La vue Git Repositories permet d'avoir une vue d'ensemble Git dans Eclipse.
- Les commandes Git sont disponibles depuis cette vue via des menus contextuels.



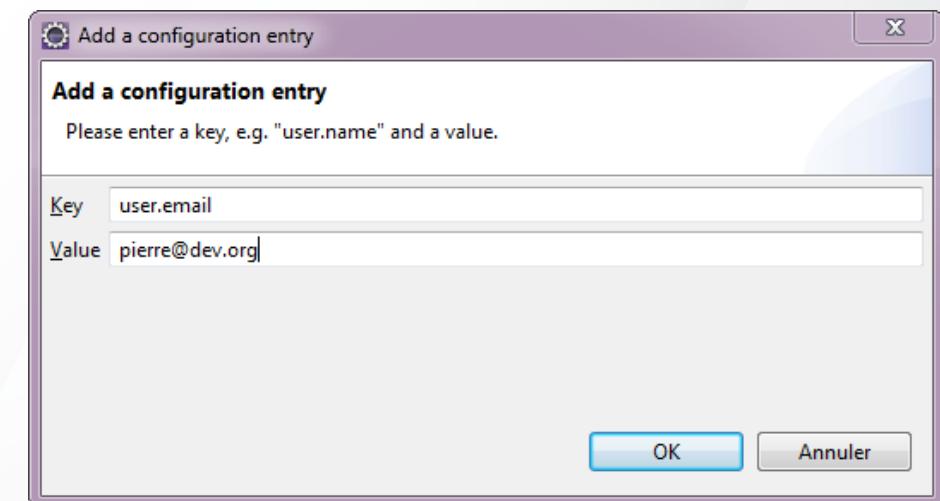
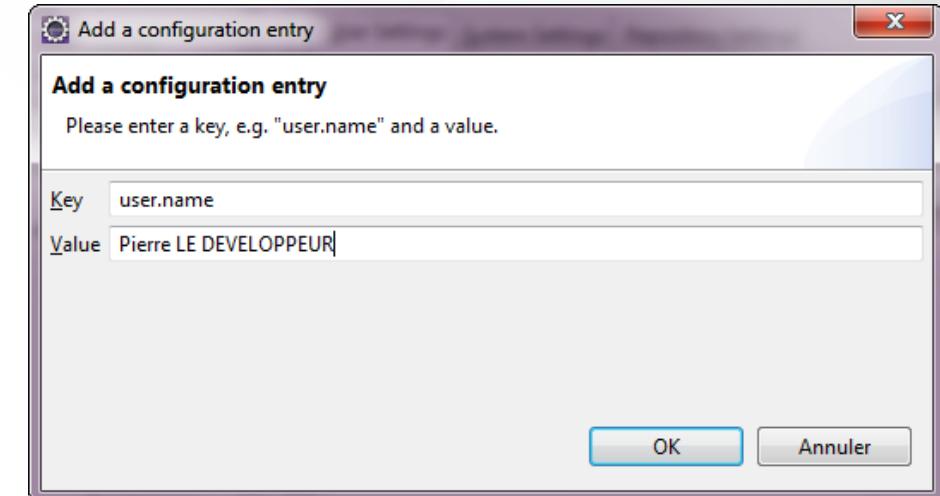
Gestion de sources avec Git: Configuration de Git dans Eclipse

- Ouvrir la fenêtre Préférences:
 - Fenêtre > Préférences
- Naviguer vers la rubrique:
 - Équipe > Git > Configuration



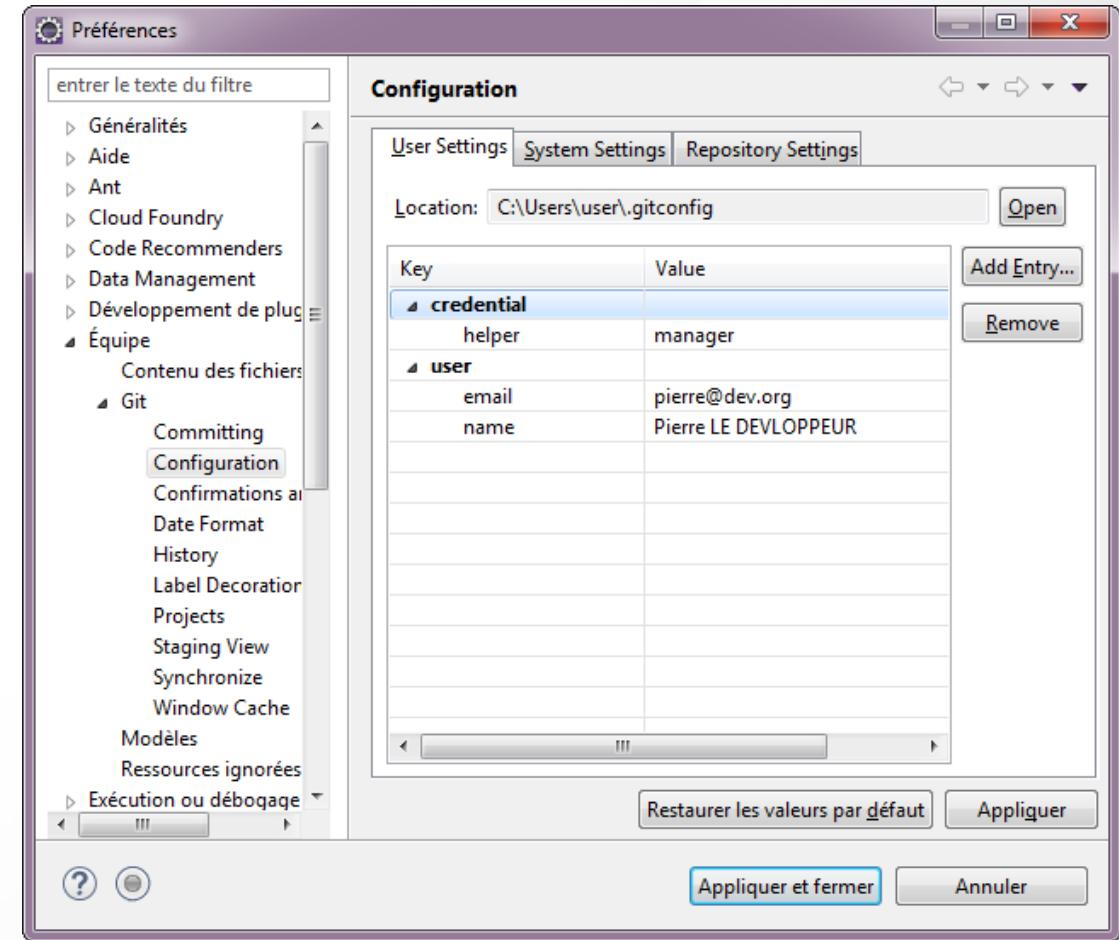
Gestion de sources avec Git: Configuration de Git dans Eclipse

- Ajouter les entrées:
 - user.name
 - user.email
 - credential.helper
- Les entrées user permettront d'identifier l'auteur de chaque action Git.
- Les nouvelles entrées seront sauvegardées dans le fichier **.gitconfig** sous le répertoire personnel de l'utilisateur connecté.



Gestion de sources avec Git: Configuration de Git dans Eclipse

- La configuration de Git est à faire une seule fois.
- Elle est sauvegardée et réutilisée par la suite à chaque fois que l'utilisateur aura besoin d'interagir avec GIT.
- La configuration Git peut être mise à jour en cas de besoin.

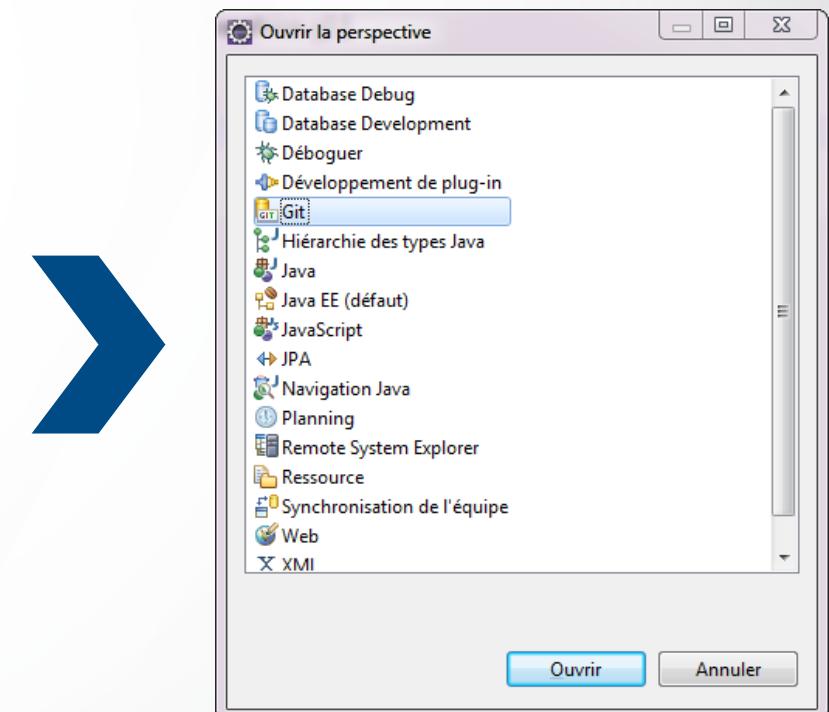
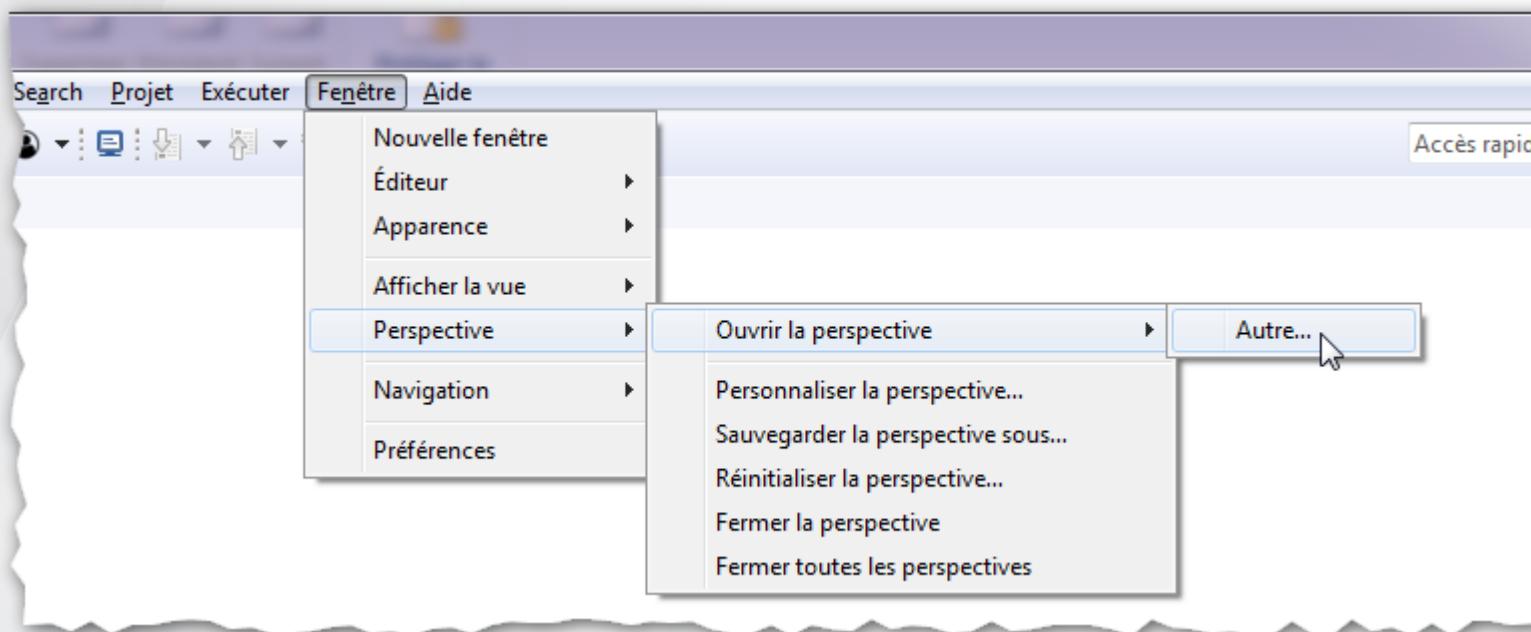


Gestion de sources avec Git : Cloner un dépôt distant

- Le dépôt local sert à enregistrer toutes les modifications apportées à l'espace de travail.
- Il est lié à un dépôt distant avec lequel il échange par des opérations de récupération et d'envoi.
- Le clonage d'un dépôt distant permet de configurer et positionner le dépôt distant lors des opération de récupération et d'envoi des modifications.
- Il est possible de cloner plusieurs fois un même dépôt distant afin d'avoir plusieurs dépôts locaux permettant de travailler dans des espaces de travail différents.

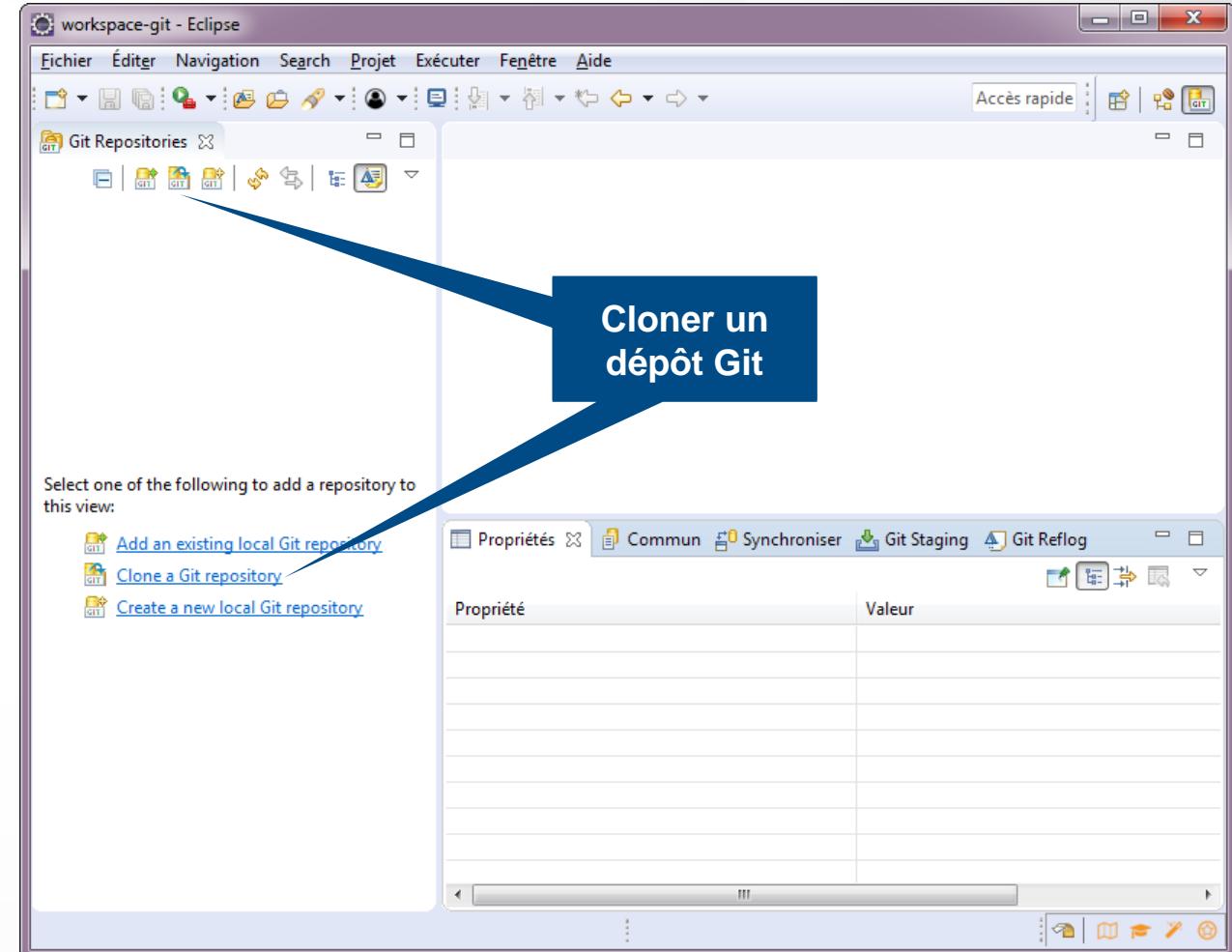
Gestion de sources avec Git : Cloner un dépôt distant

- Ouvrir la perspective Git:
 - Fenêtre > Perspective > Ouvrir la perspective > Autre > Git

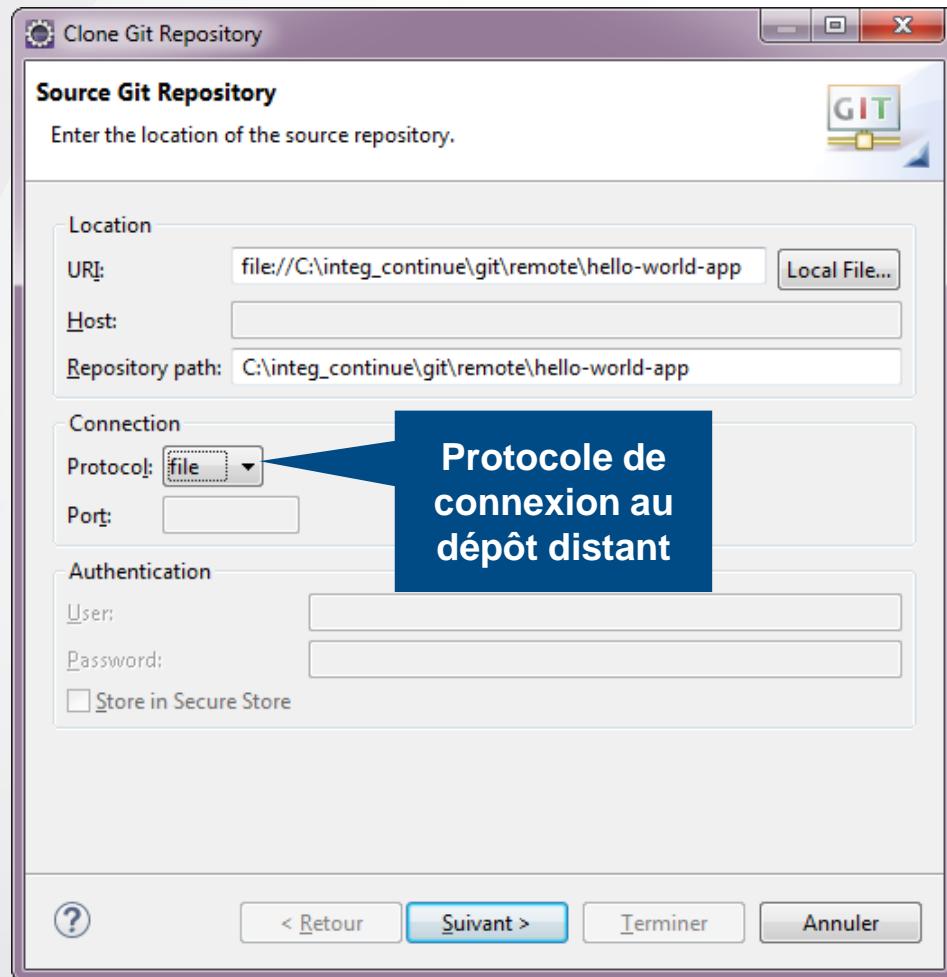


Gestion de sources avec Git : Cloner un dépôt distant

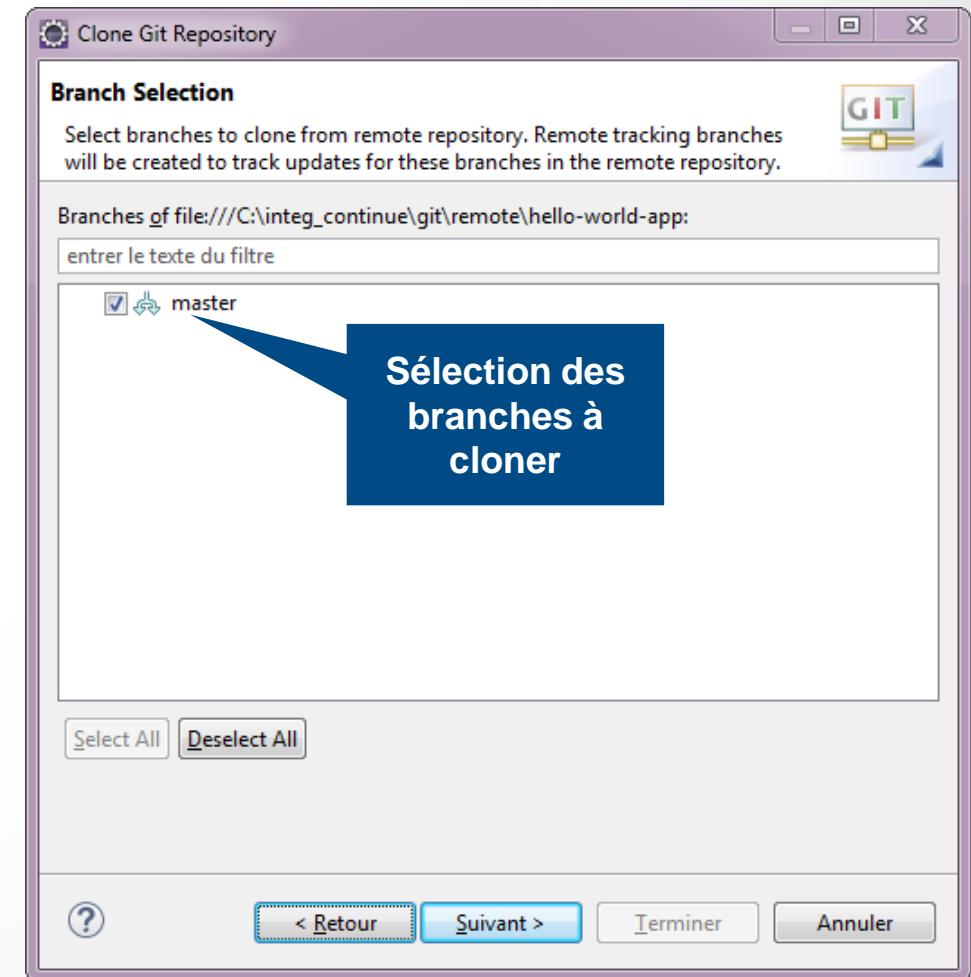
- Sélectionner Clone Git Repository and add the clone to this view.



Gestion de sources avec Git : Cloner un dépôt distant

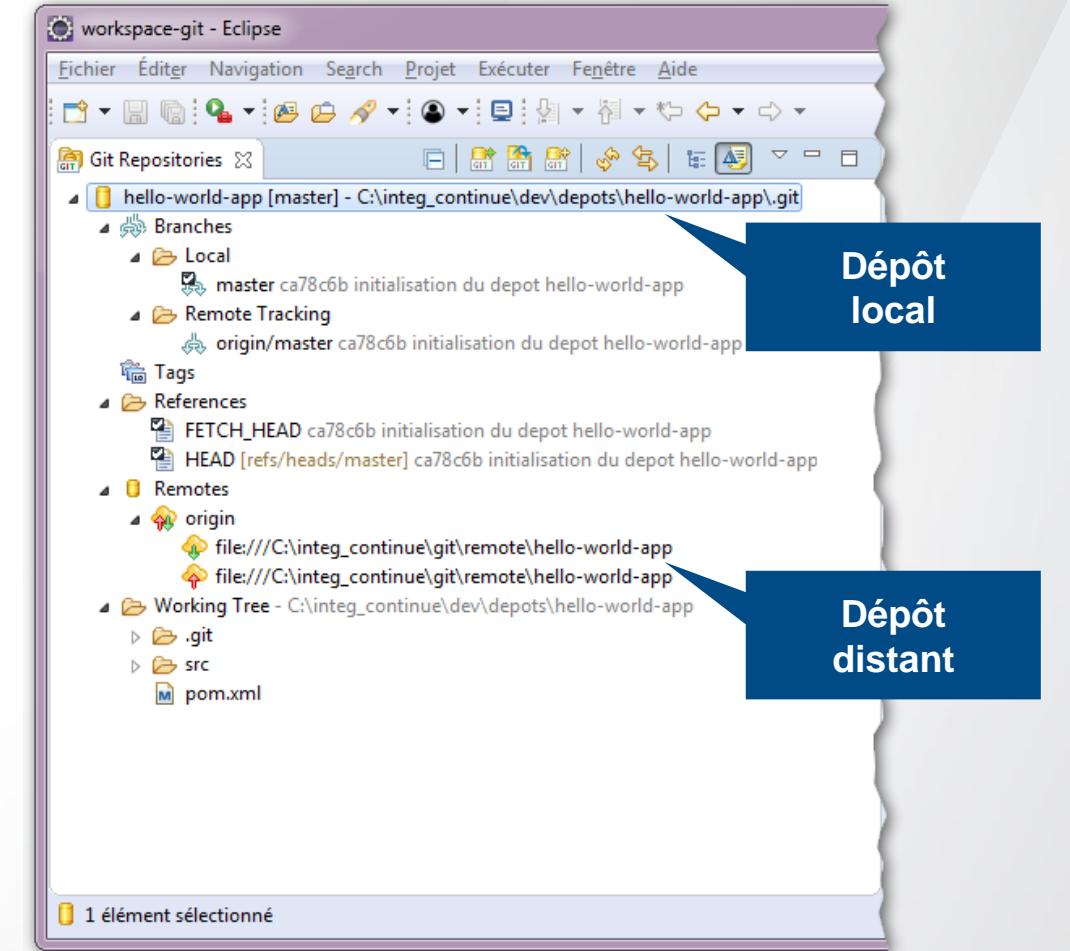
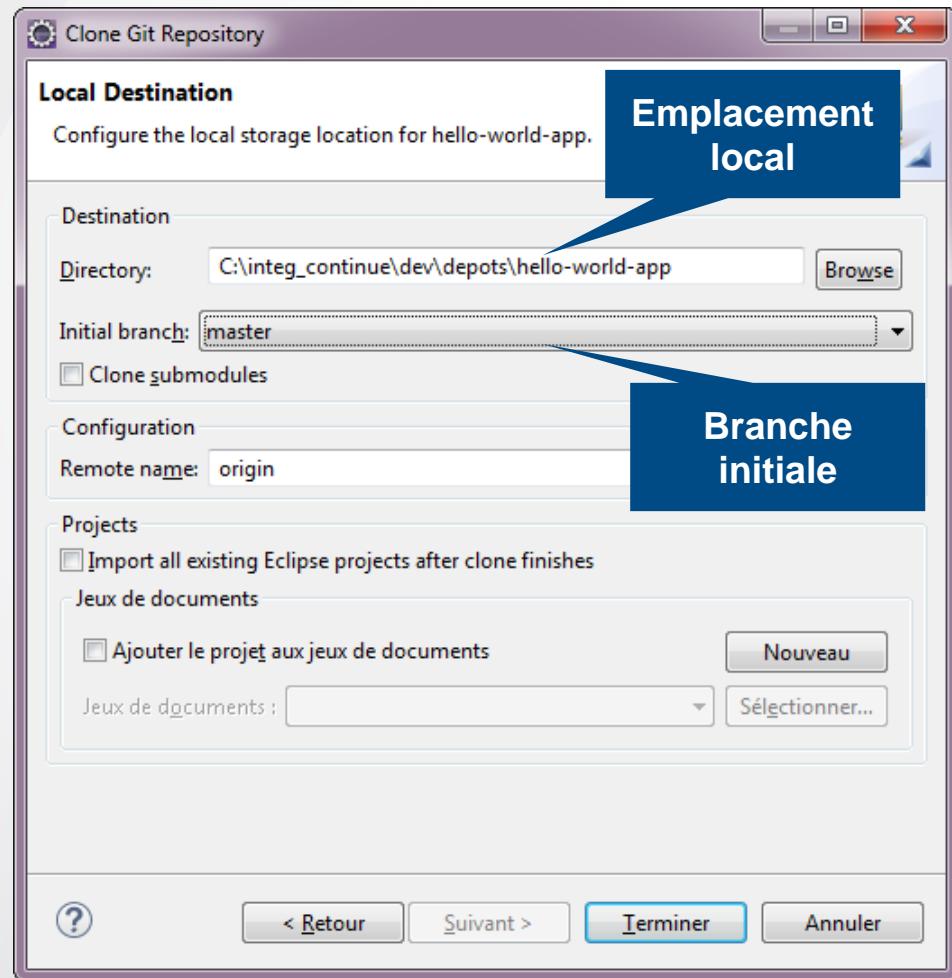


Protocole de
connexion au
dépôt distant



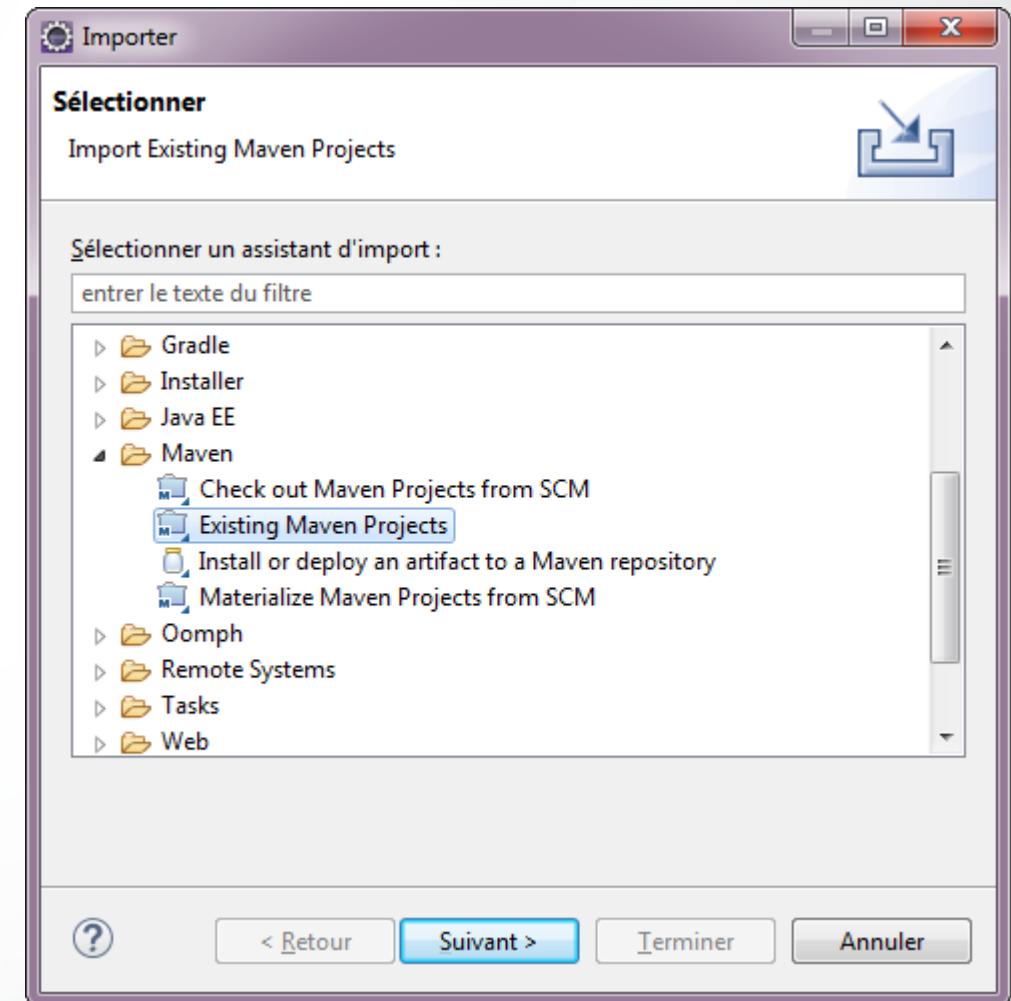
Sélection des
branches à
cloner

Gestion de sources avec Git : Cloner un dépôt distant

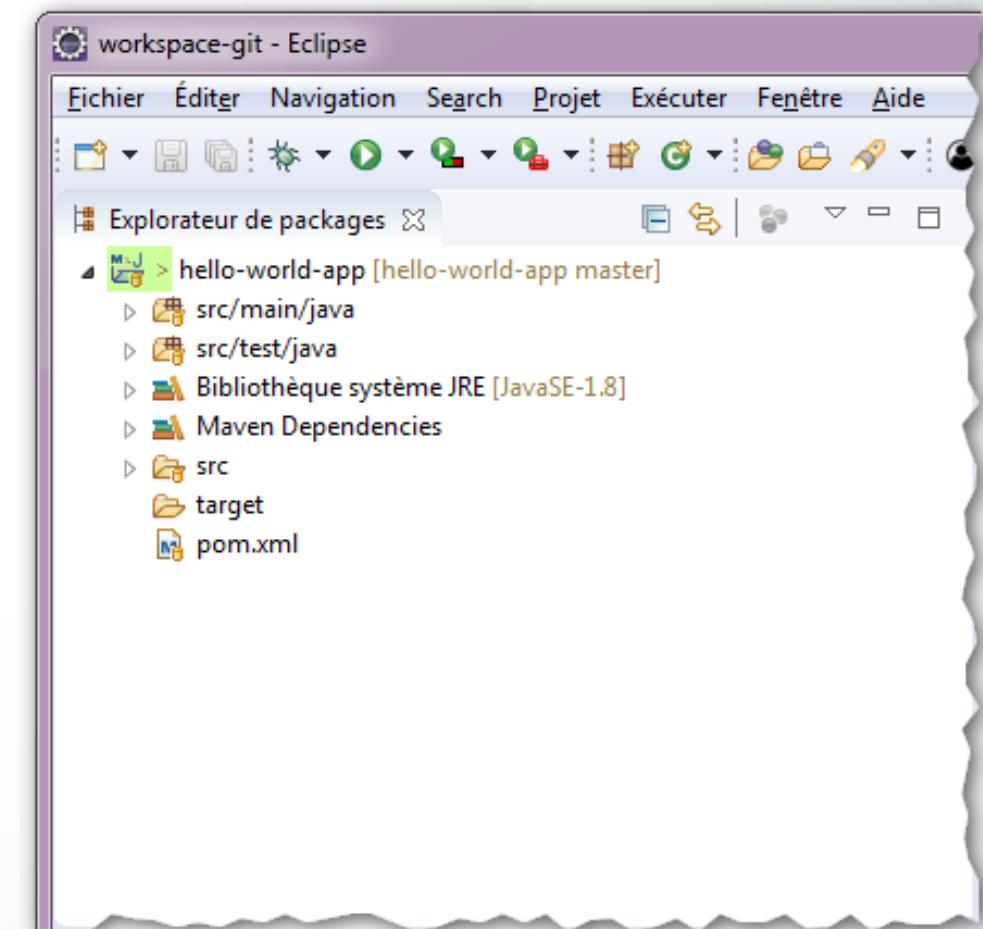
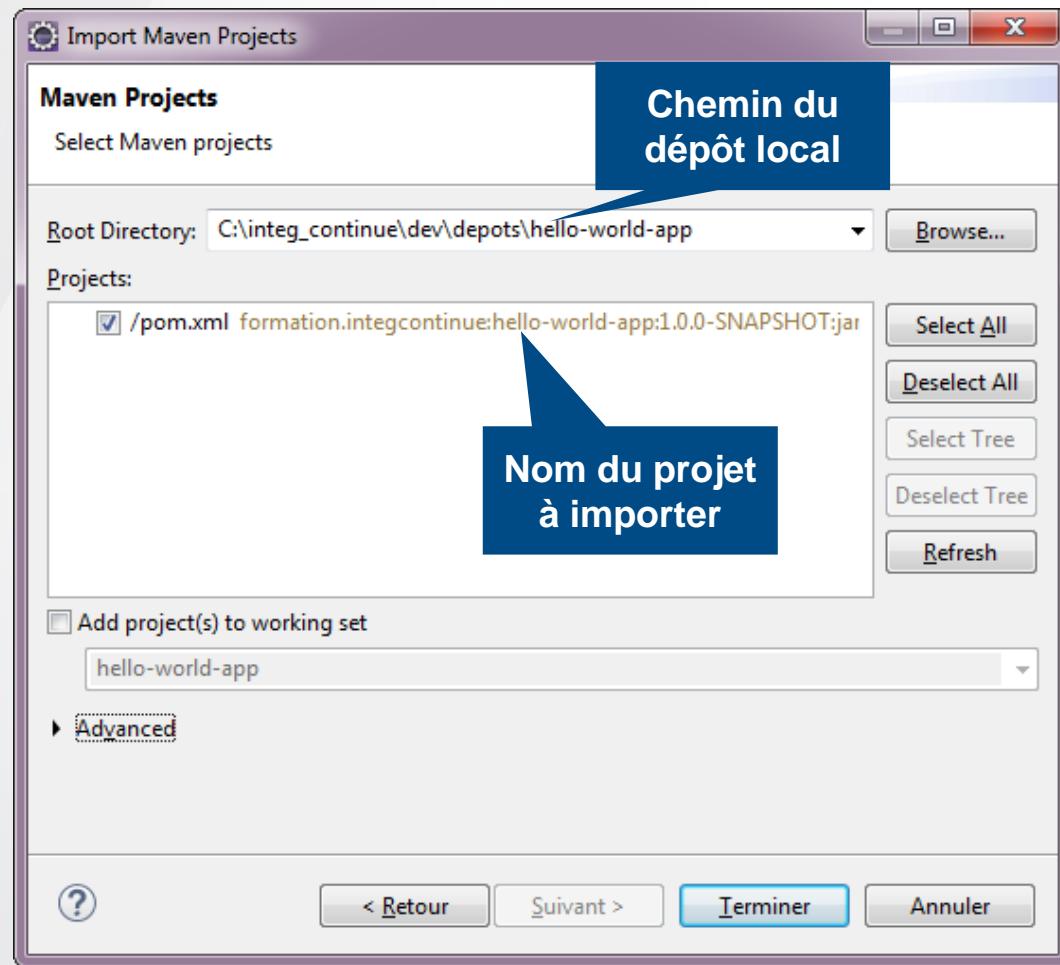


Gestion de sources avec Git: Associer un projet Eclipse au dépôt local Git

- Ouvrir la perspective Java
 - Fenêtre > Perspective > Ouvrir la perspective > Autre > Java
- Depuis le menu contextuel du projet Java sélectionner:
 - Importer > Existing Maven Projects

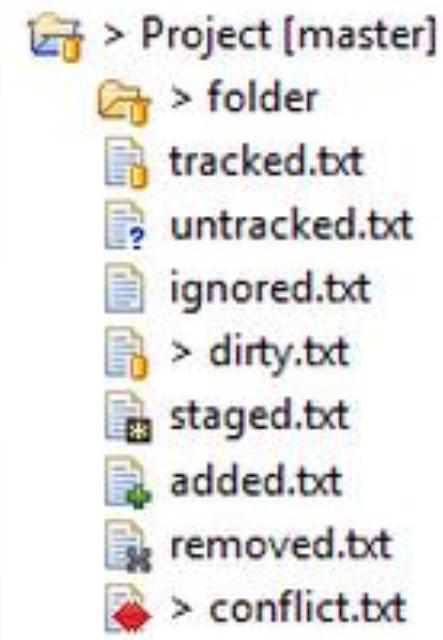


Gestion de sources avec Git: Associer un projet Eclipse au dépôt local Git



Gestion de sources avec Git: Associer un projet Eclipse au dépôt local Git

- La vue Explorateur de packages affiche les icônes sur les fichiers pour montrer leur statut.
- Les décorateurs les plus importants sont:
 - Suivi: le fichier a été commité dans Git et n'a pas été modifié depuis.
 - Non suivi: le fichier n'est ni stagé ni commisé.
 - Ignoré: le fichier est marqué pour être ignoré par les opérations Git.
 - Modifié: le fichier a été modifié depuis le dernier commit.
 - Stagé: les changements dans le fichier seront inclus dans la prochain commit.

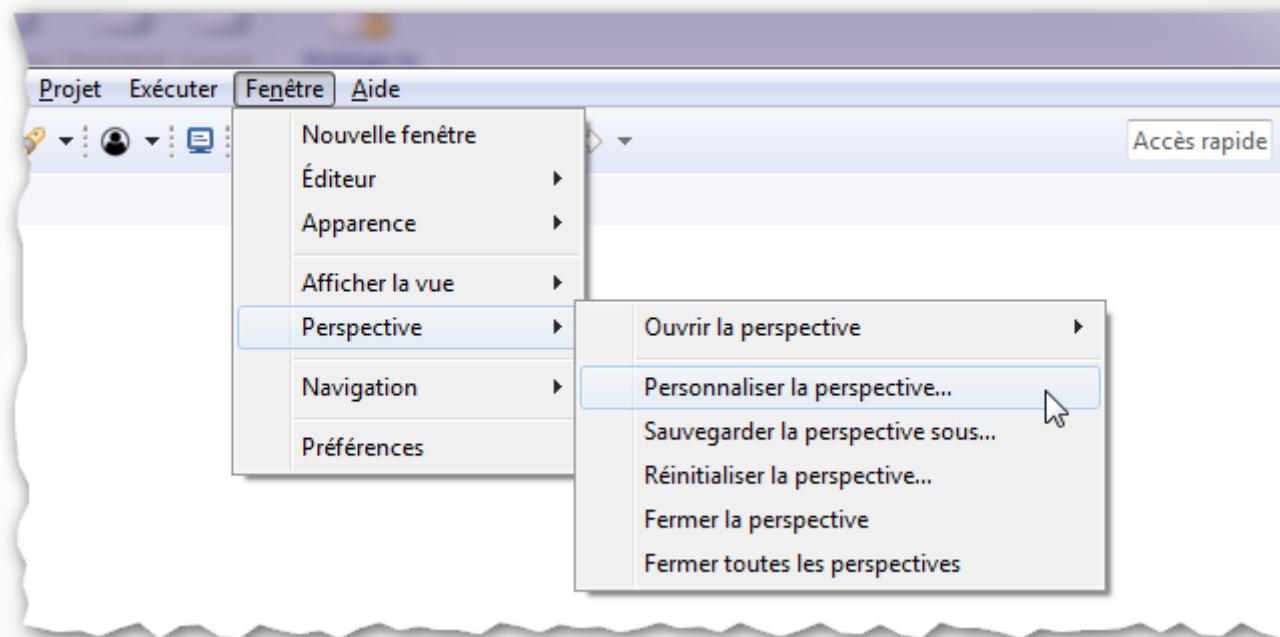


Gestion de sources avec Git: Associer un projet Eclipse au dépôt local Git

- Ajouté: stagé mais pas encore commité.
- Supprimé: la ressource est retirée du référentiel Git.
- Conflit: un conflit de fusion existe pour le fichier.

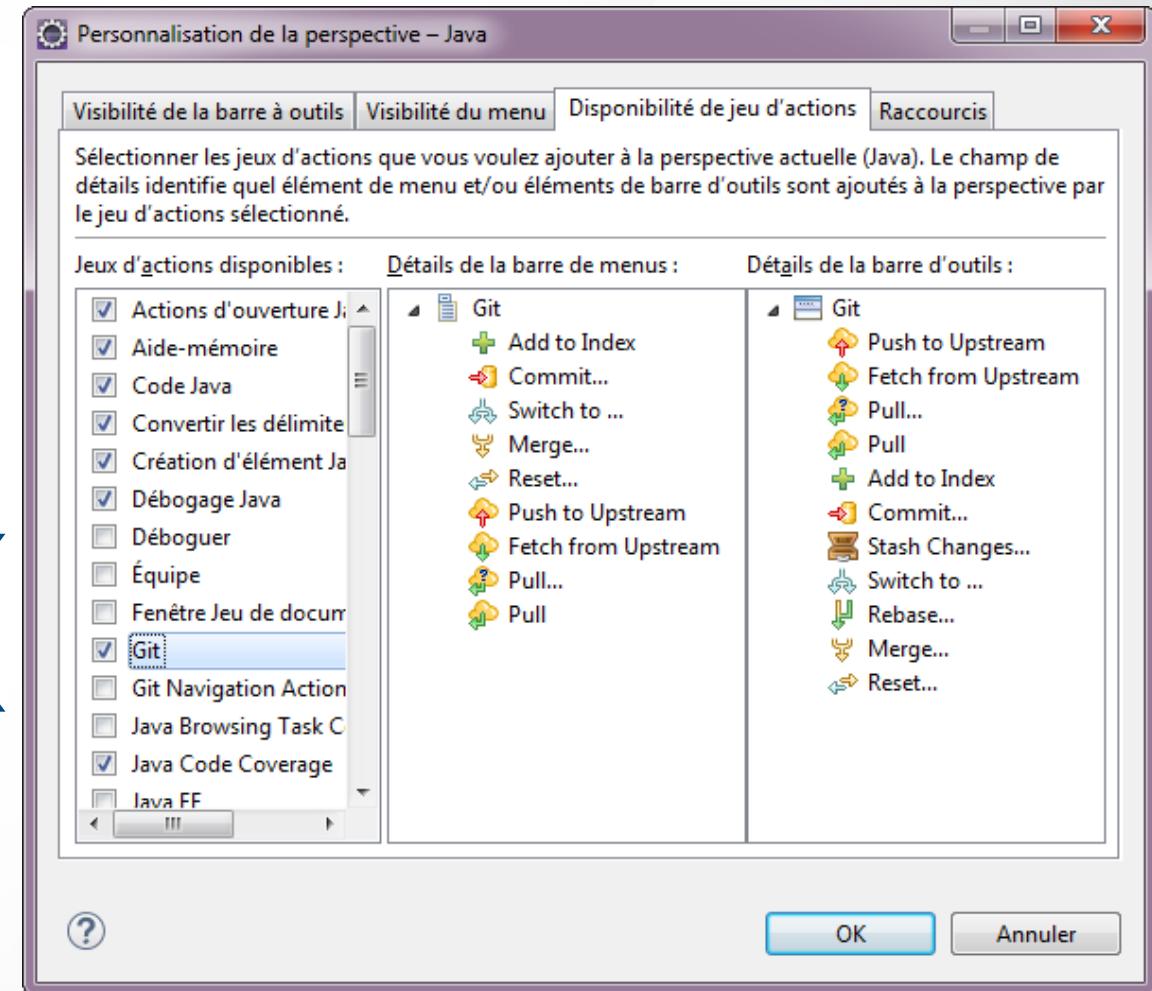
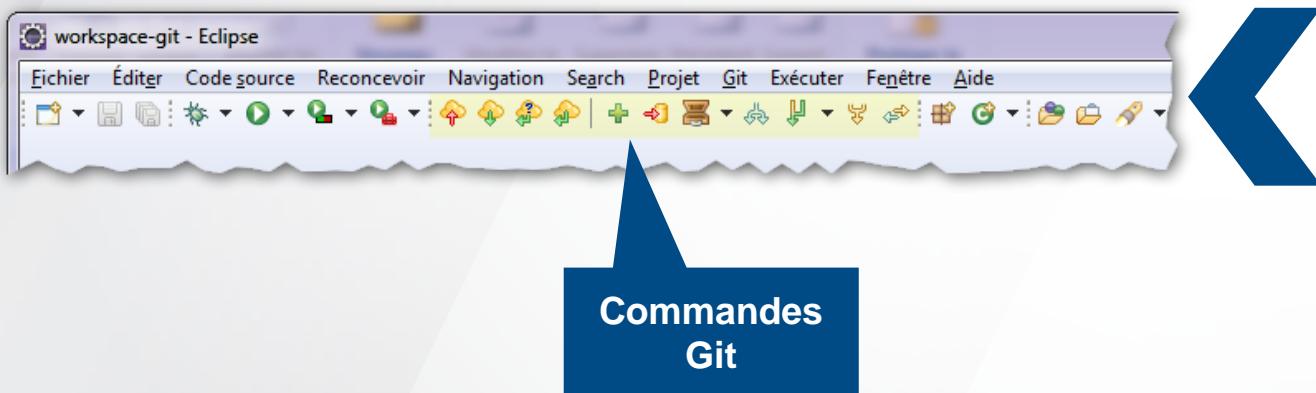
Gestion de sources avec Git: Ajouter les commandes Git à la perspective Java

- Dans la perspective Java, ouvrir la fenêtre Personnaliser la perspective



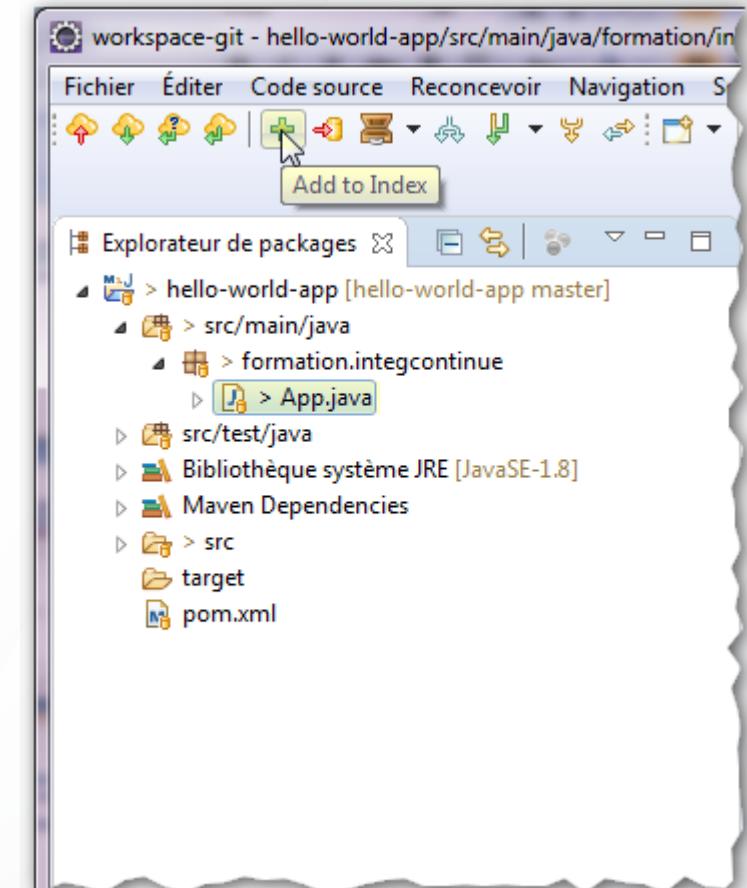
Gestion de sources avec Git: Ajouter les commandes Git à la perspective Java

- Naviguer vers l'onglet Disponibilité de jeu d'actions et cocher la case Git.
- Le groupe de commandes Git apparaitra alors dans une barre d'outils dédiée.

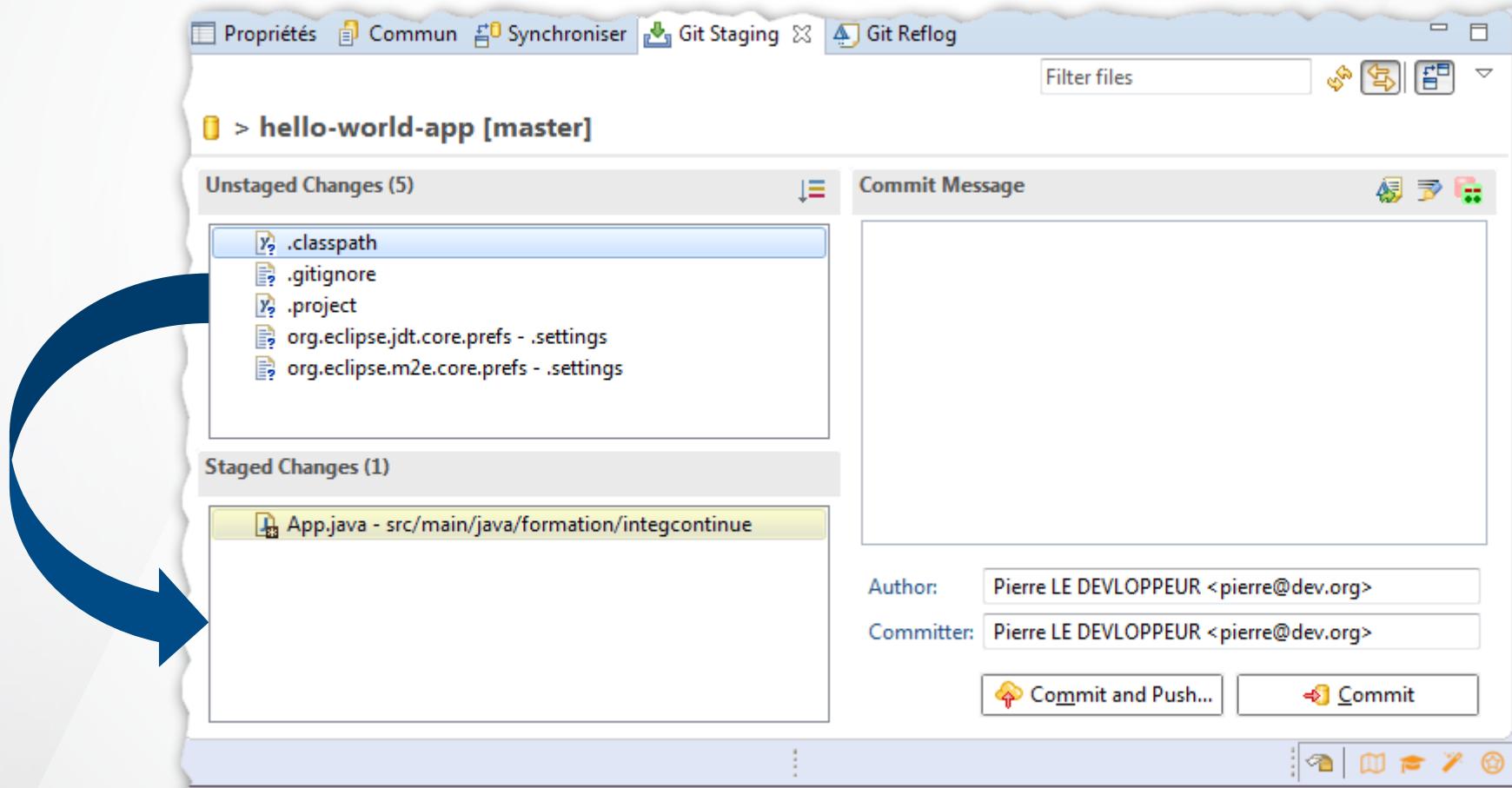


Gestion de sources avec Git: Ajouter un élément à l'index Git

- Après la modification des fichiers dans l'espace de travail, on peut insérer ces changements dans l'index Git.
- Dans la perspective Java, sélectionner le fichier et activer le bouton de commande Add to Git Index.
- Il est possible de réaliser la même action dans la perspective Git. Dans la vue Git Staging pour ajouter un fichier à l'index, le faire passer du panneau Unstaged changes au panneau Staged Changes

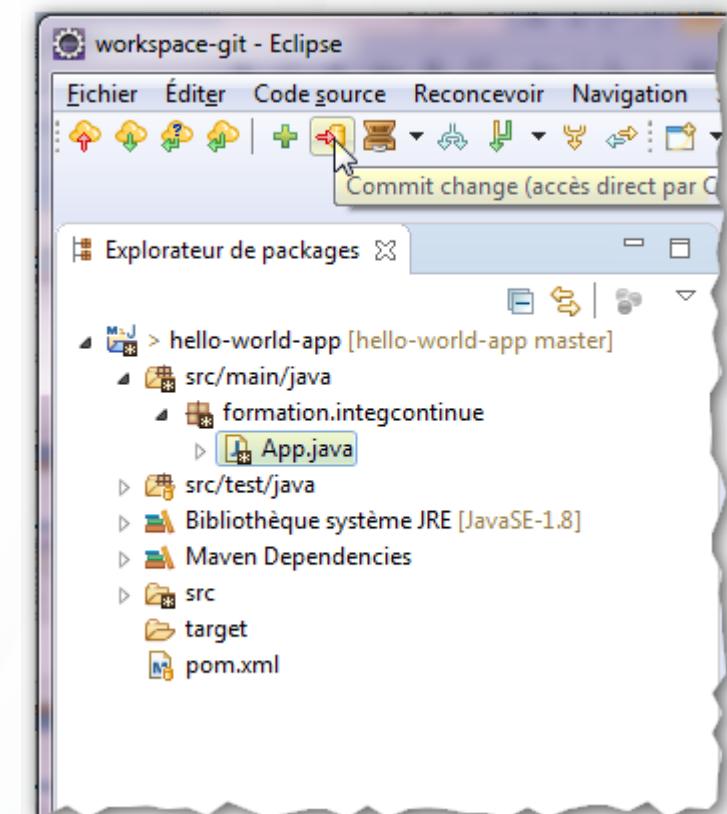


Gestion de sources avec Git: Ajouter un élément à l'index Git



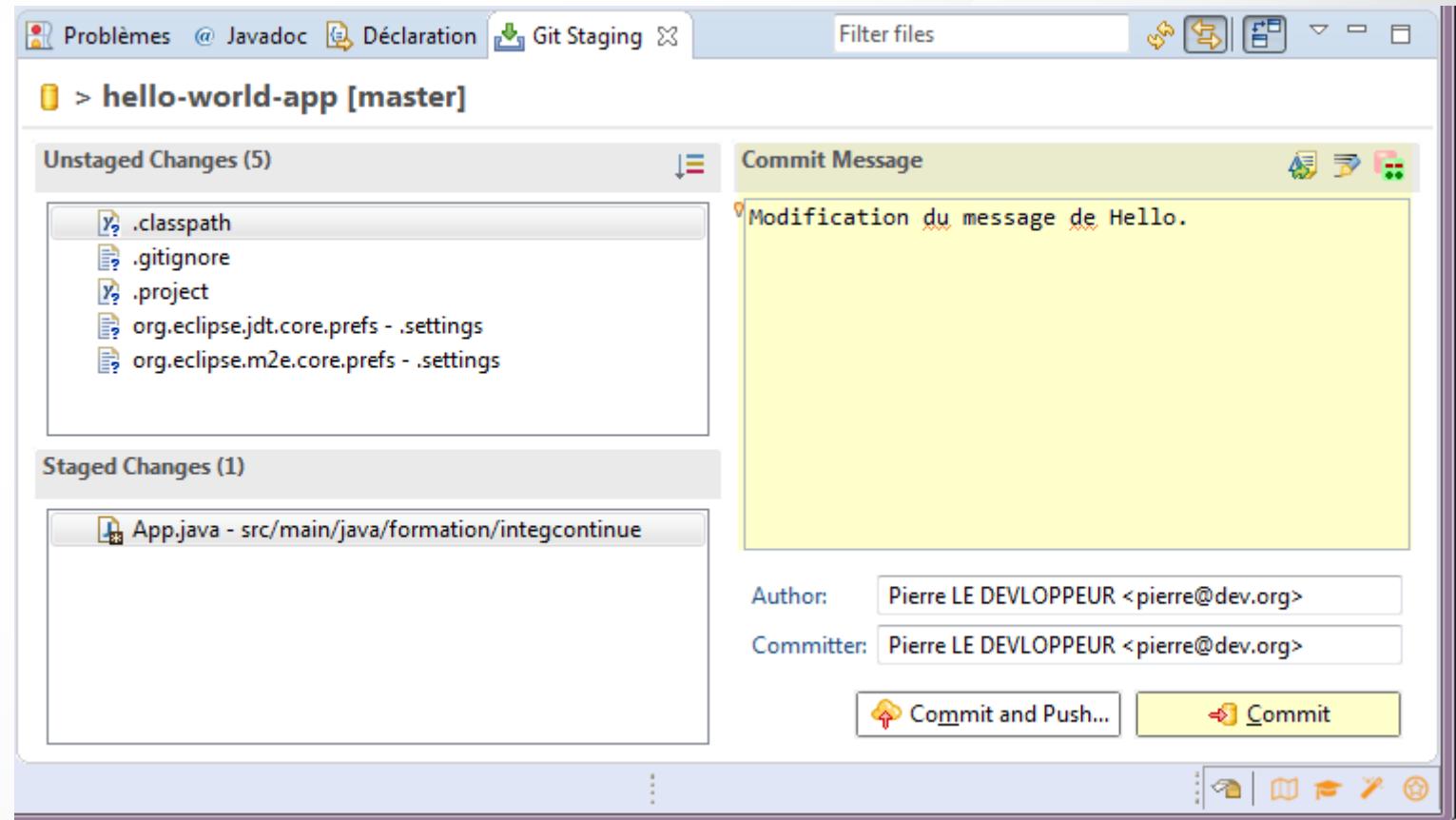
Gestion de sources avec Git: Enregistrer les changements

- Après avoir ajouté, modifié ou supprimé des fichiers dans l'espace de travail on peut consigner ces changements dans le dépôt Git.
- Il est préférable des les avoir ajoutés au préalable à l'index Git.
- Dans la perspective Java, sélectionner le fichier et activer le bouton de commande Commit change.
- La vue Git Staging est alors affichée.
- Un message de commit est à saisir pour finaliser le commit.



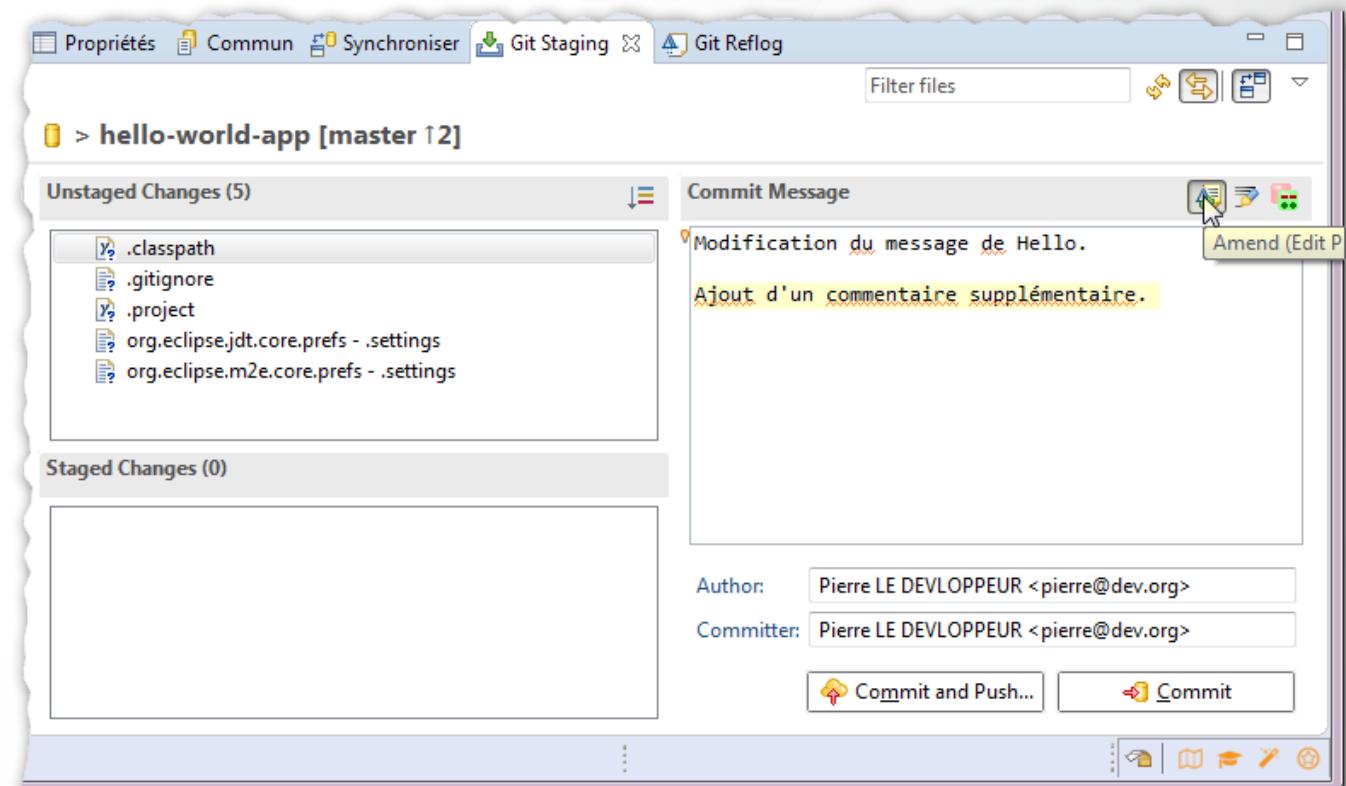
Gestion de sources avec Git: Enregistrer les changements

- La même action est également réalisable dans la perspective Git, la vue Git Staging.
- L'utilisation du bouton Commit and push rend le commit irréversible. Il ne sera plus possible de le modifier.



Gestion de sources avec Git: Enregistrer les changements

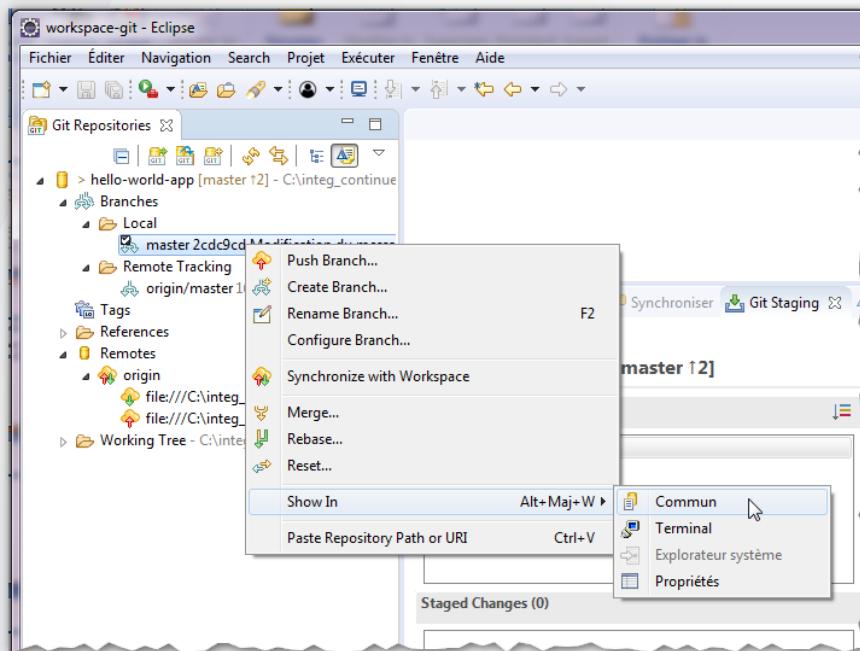
- Il est possible de modifier, dans le dépôt local, le dernier commit effectué sur une branche, à la fois au niveau du contenu et du message de commit.
- Activer le bouton Amend last commit pour reprendre le dernier commit.



Gestion de sources avec Git: Historique des changements

- La vue Commun permet de voir, en mode graphe, l'ensemble des commits.
- On peut y sélectionner une branche, une étiquette, un fichier spécifique.
- En activant l'arbre de révision, on peut regarder les contributions et se positionner sur toute version de l'objet par son commit.
- Pour afficher l'historique des changement se mettre dans la perspective Git et positionner sur la branche locale à visualiser.
- Via le menu contextuel naviguer vers Show In > Commun.

Gestion de sources avec Git: Historique des changements



The screenshot shows the Eclipse IDE interface with the title 'workspace-git - Eclipse'. The 'Git Staging' view is open, displaying a commit history table for the repository 'hello-world-app'. The table has columns for 'Id', 'Message', 'Author', 'Authored Date', 'Committer', and 'Commit Date'. The commits listed are:

Id	Message	Author	Authored Date	Committer	Commit Date
2cdc9cd	[master] [HEAD] Modification du message de Hello.	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38
067f135	Modification du message de Hello.	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38
16f8c6d	[origin/master] Refactoring du nom de package	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38
dab8fae	Refactoring du nom de package	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38
9b1ce05	Mise à jour nom du projet	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38
ca78c6b	initialisation du depot hello-world-app	Pierre LE DEVELOPPEUR	2018-05-06 22:38	Pierre LE DEVELOPPEUR	2018-05-06 22:38

Below the table, a commit message editor shows the message 'Modification du message de Hello.' and the note 'Ajout d'un commentaire supplémentaire.' To the right, a code editor displays the Java file 'src/main/java/formation/integcontinue/App.java' with the code:

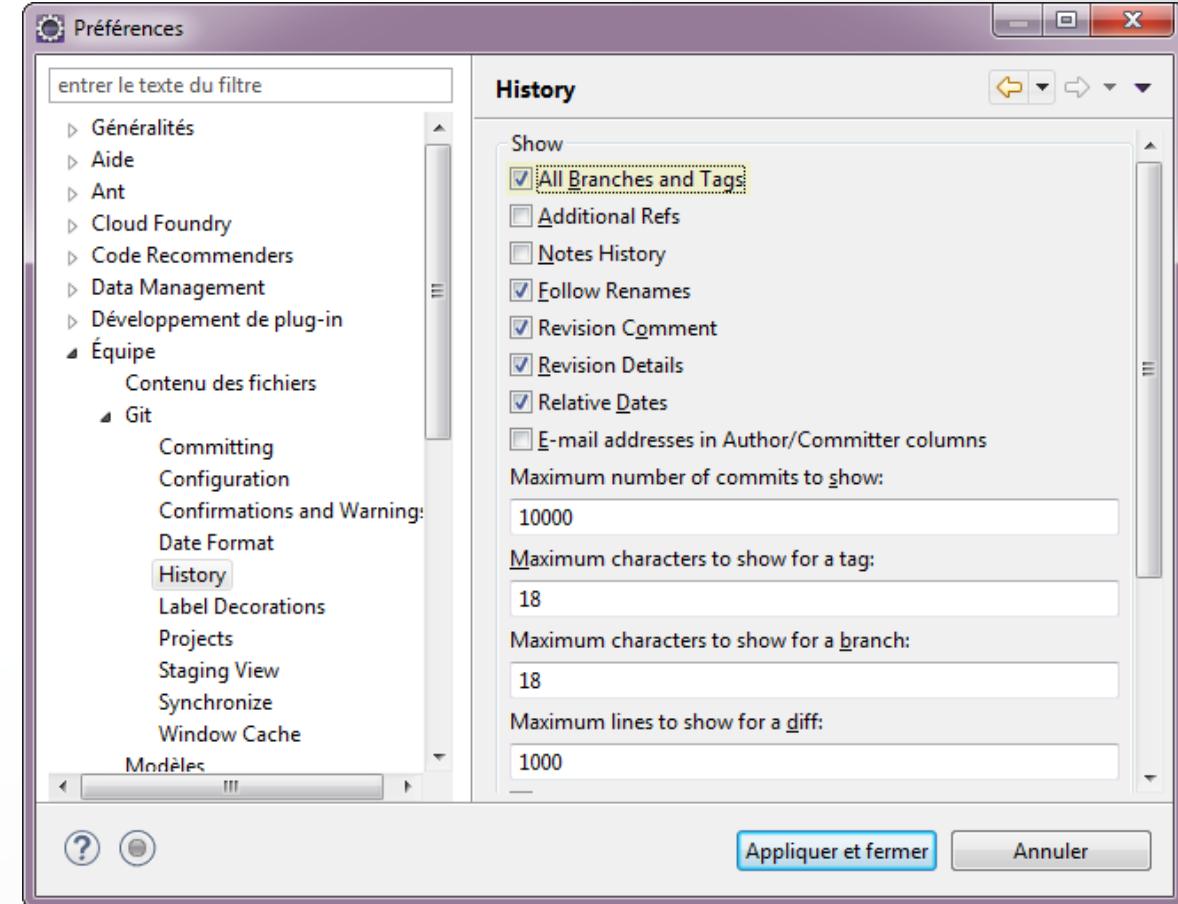
```
modification du message de Hello.
```

Four callout boxes point to specific elements in the interface:

- 'Graphe des commits' points to the commit history table.
- 'Commit en cours' points to the commit message editor.
- 'Message du commit' points to the commit message in the editor.
- 'Contenu du commit' points to the code editor showing the Java file content.

Gestion de sources avec Git: Historique des changements

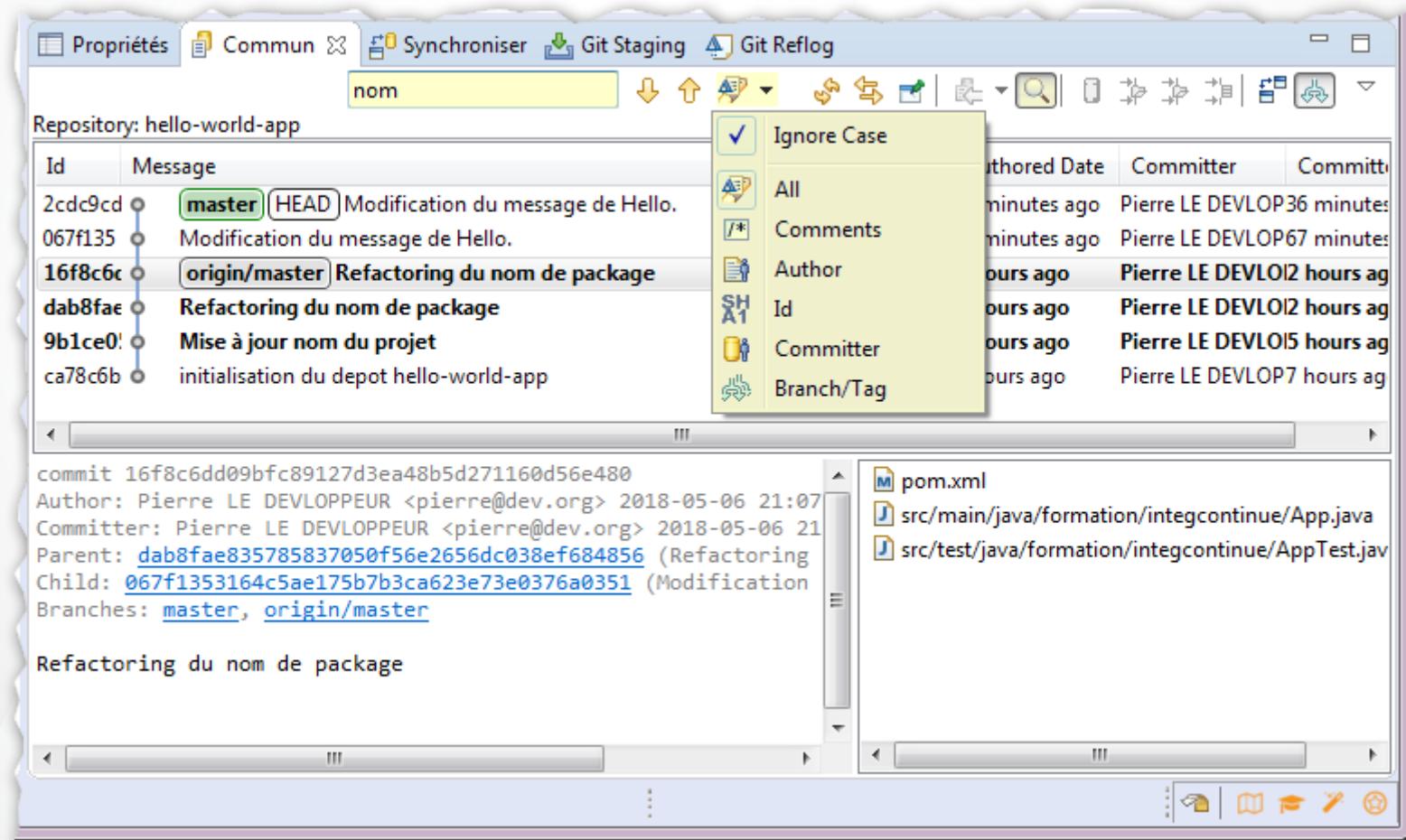
- Pour avoir une vision d'ensemble et voir l'arbre de révision complet cocher la case All Branches and Tags dans les préférences Team > Git > History.



Gestion de sources avec Git: Historique des changements

- Il est possible de rechercher des commits en fonction de l'auteur, de l'ID ou du commentaire de commit.
- Pour cela, activez la barre d'outils Afficher la recherche et tapez une chaîne de recherche dans le champ Rechercher.
- Les commits correspondant à la recherche sont mis en surbrillance.

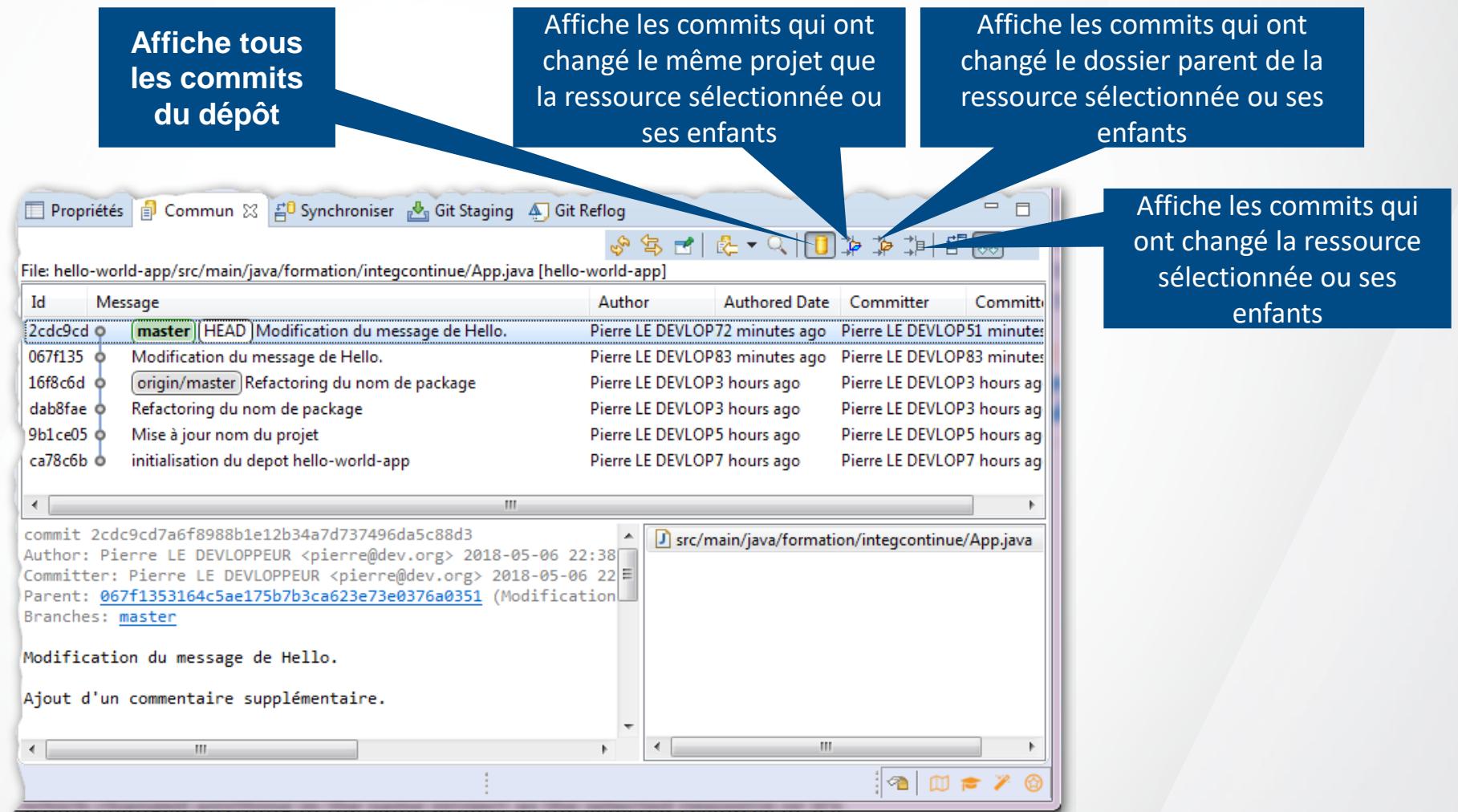
Gestion de sources avec Git: Historique des changements



Gestion de sources avec Git: Historique des changements

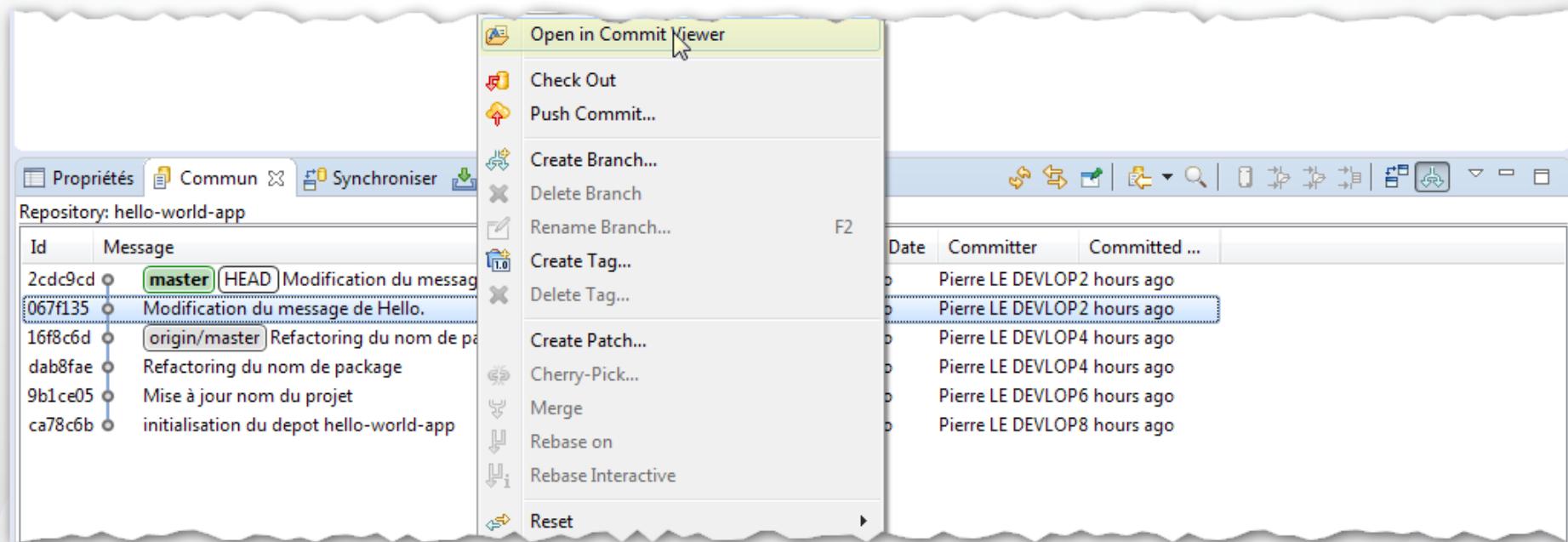
- La liste des commits affichée dans la vue Commun peut être filtrée pour personnaliser les commits affichés.
- Par défaut, la vue Commun filtre l'historique en fonction de la sélection en cours et affiche uniquement la branche active.
- Il est possible de filtrer en fonction des ressources.

Gestion de sources avec Git: Historique des changements



Gestion de sources avec Git: Afficher un changement

- Le contenu d'un commit peut être visualisé dans le Commit viewer.
- Dans la perspective Git, dans la vue Commun sélectionner le commit à visualiser, afficher le menu contextuel et activer le menu Open in Commit Viewer.



Gestion de sources avec Git: Afficher un changement



Screenshot of a Git commit interface showing a message and a file diff.

Commit 067f1353164c5ae175b7b3ca623e73e0376a0351 hello-world-app

Pierre LE DEVELOPPEUR <pierre@dev.org> on 2018-05-06 22:27:16 Parent: 16f8c6dd09bfc89127d3

Message
Modification du message de Hello.

Files (1)
src/main/java/formation/integcontinue/App.java

Branches (1)
master

Commit Diff

Screenshot of a Git commit interface showing a detailed diff view.

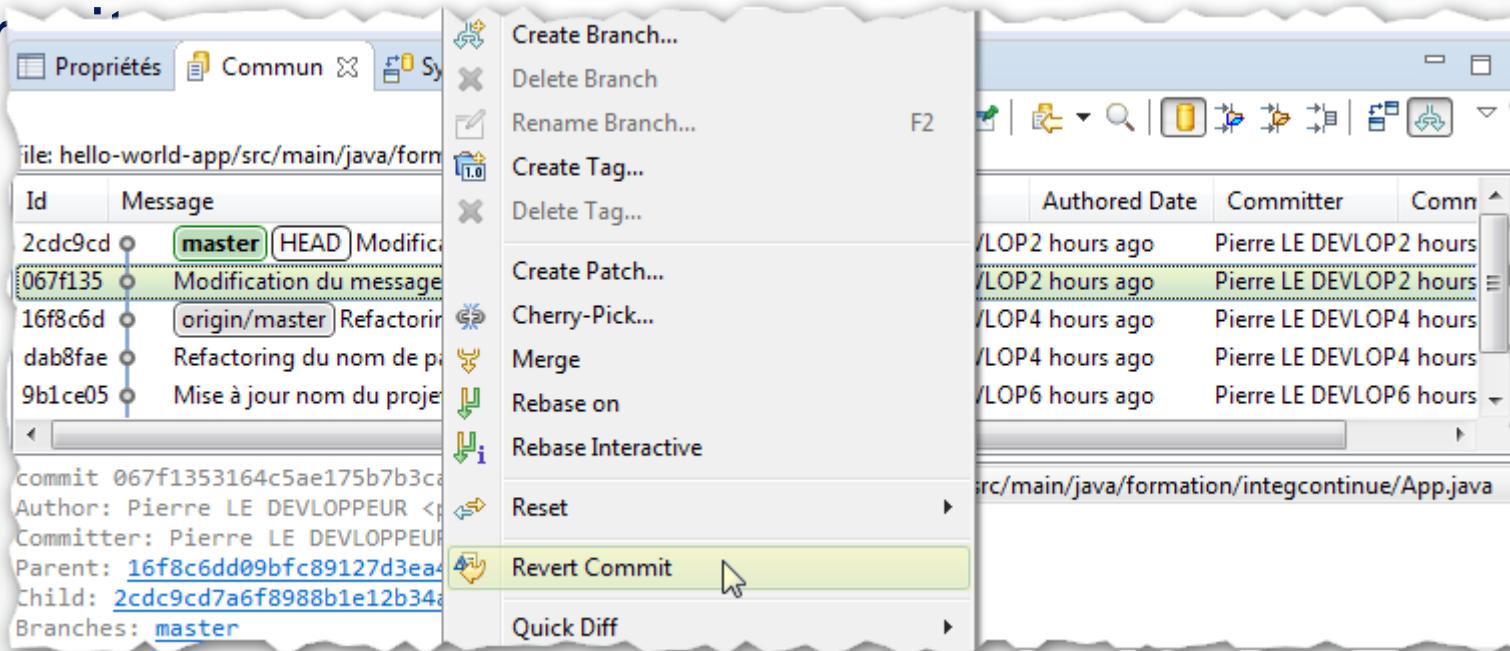
Commit 067f1353164c5ae175b7b3ca623e73e0376a0351 hello-world-app

```
diff --git a/src/main/java/formation/integcontinue/App.java b/src/main/java/formation/integ
index 219df3c..29ac8b8 100644
--- a/src/main/java/formation/integcontinue/App.java
+++ b/src/main/java/formation/integcontinue/App.java
@@ -8,6 +8,6 @@
8 {
9     public static void main( String[] args )
10    {
11 -     System.out.println( "Hello World!" );
11 +     System.out.println( "Hello World for Git users!" );
12    }
13 }
```

Commit Diff

Gestion de sources avec Git: Annuler un changement

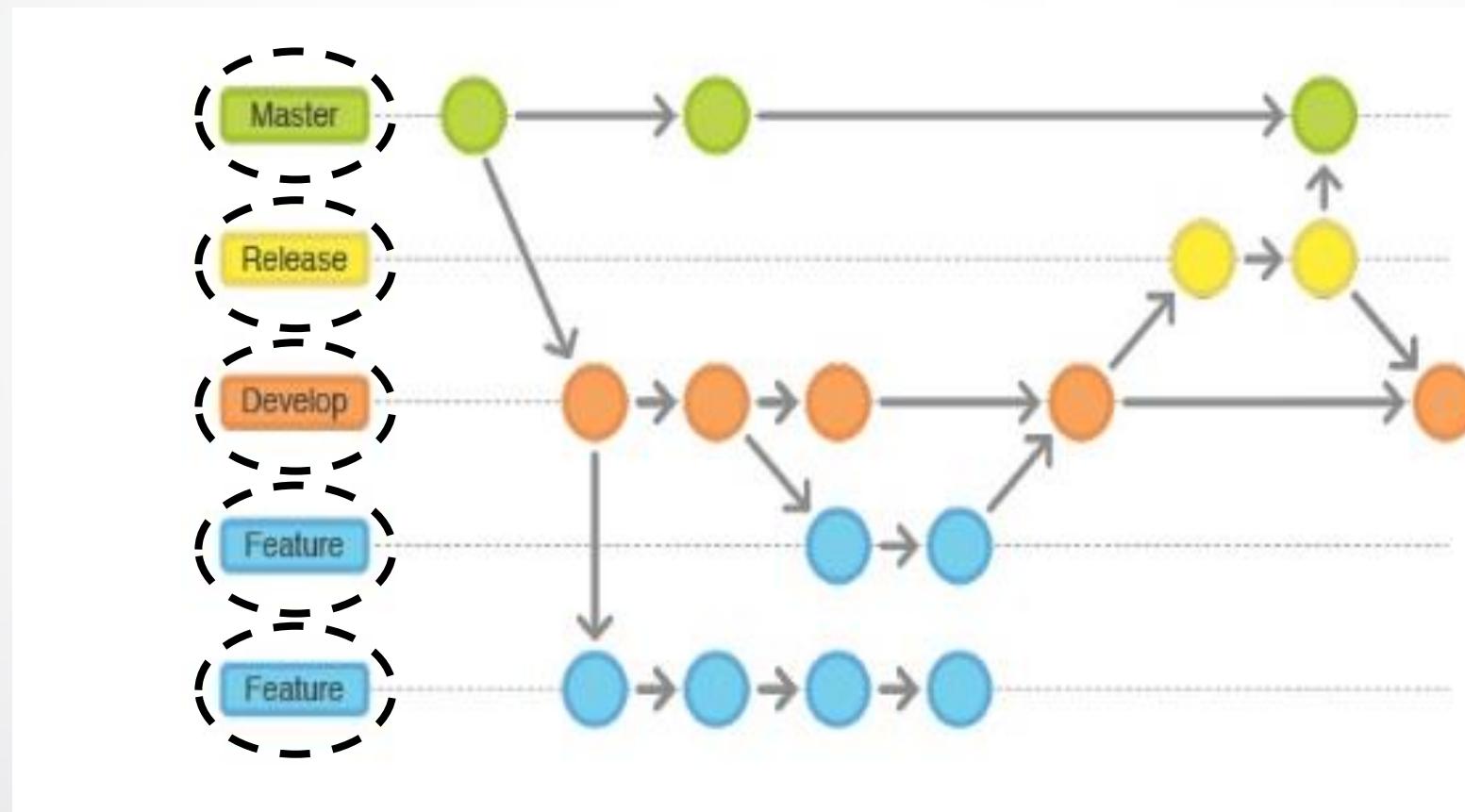
- A partir d'un commit, on peut revenir sur celui-ci en effectuant un Revert Commit. Un nouveau commit est alors créé.
- Dans la perspective Git, dans la vue Commun sélectionner le commit à annuler, afficher le menu contextuel et activer le menu Revert Com



Gestion de sources avec Git : Créer une branche

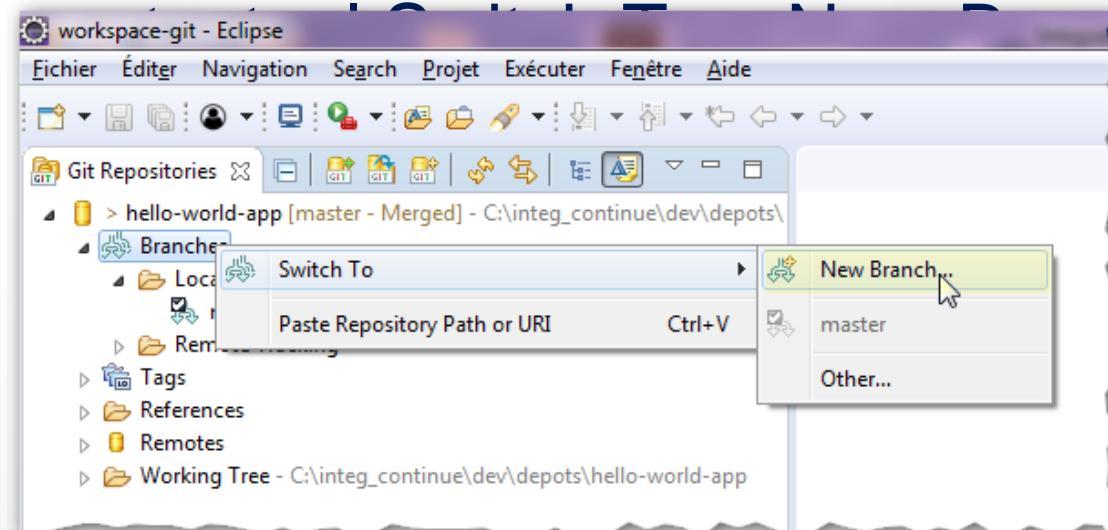
- Les branches sont des références évoluant lors du commit.
- La création d'une branche consiste à pointer vers un nouvel élément.
- Le commit courant s'appelle HEAD. Si HEAD référence une branche alors cette branche est appelée la branche courante.
- Toute branche est identifiable par son SHA-1.

Gestion de sources avec Git : Créer une branche

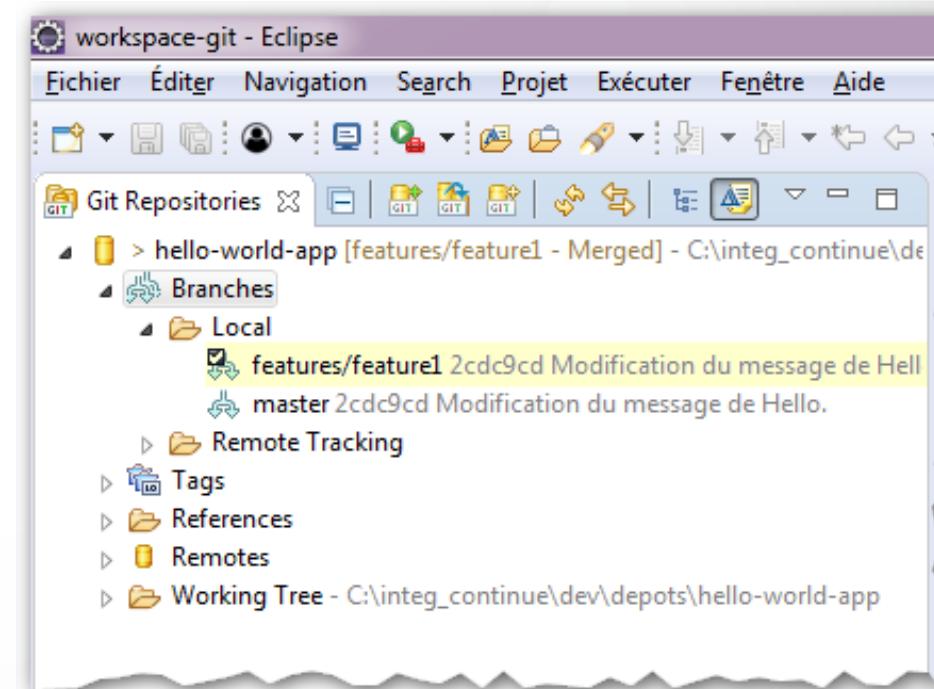
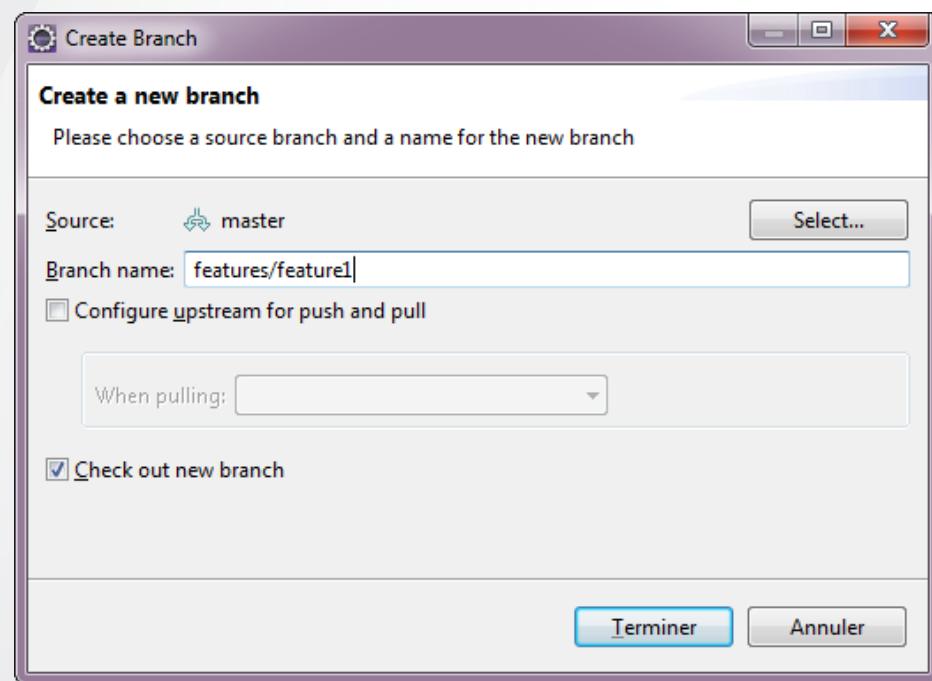


Gestion de sources avec Git : Créer une branche

- Dans Eclipse, la création de nouvelles branches Git est proposée via le menu contextuel de l'élément Branches de la vue Git Repositories.
- Dans la vue Git Repositories ouvrir le dépôt et se positionner sur l'objet Branches.
- Utiliser le menu



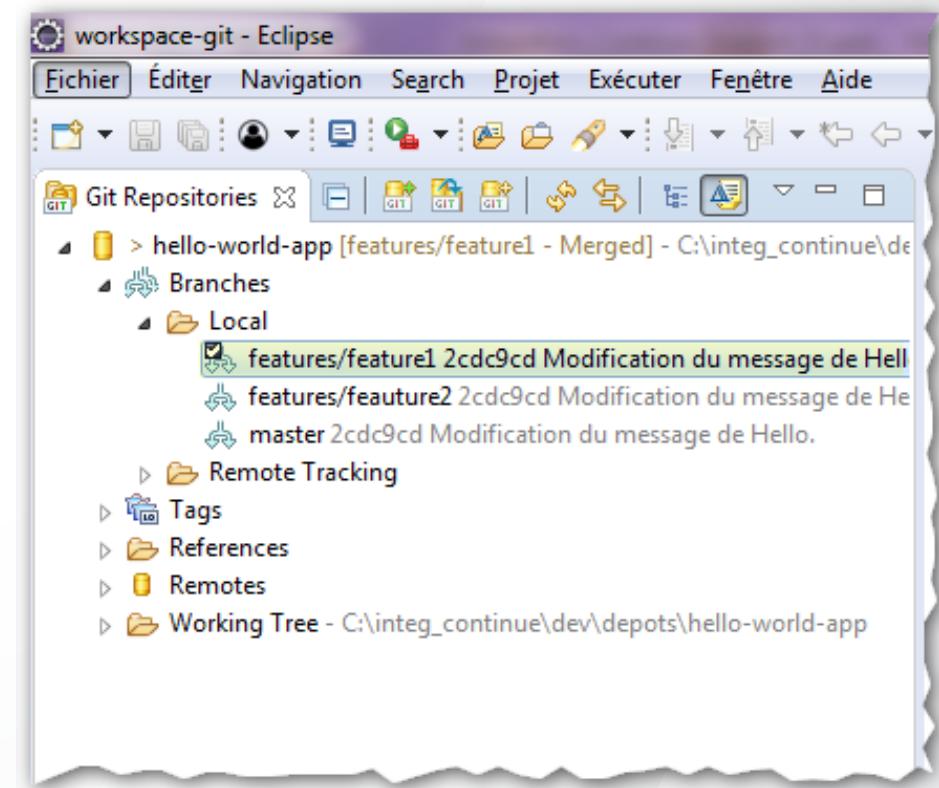
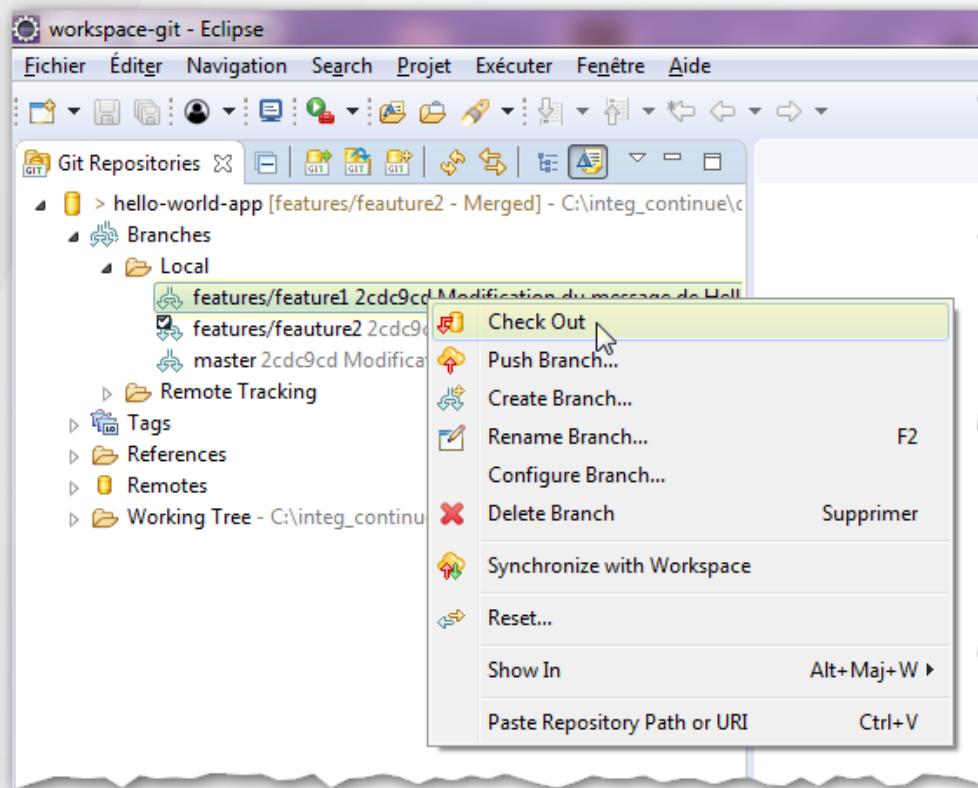
Gestion de sources avec Git : Créer une branche



Gestion de sources avec Git : Créer une branche

- Par le menu contextuel de la vue Git Repositories, on peut positionner l'espace de travail sur une branche existante en effectuant un checkout de la branche sélectionnée.
- Se positionner sur la branche sur laquelle on souhaite positionner l'espace de travail.
- Utiliser le menu contextuel checkout.

Gestion de sources avec Git : Créer une branche

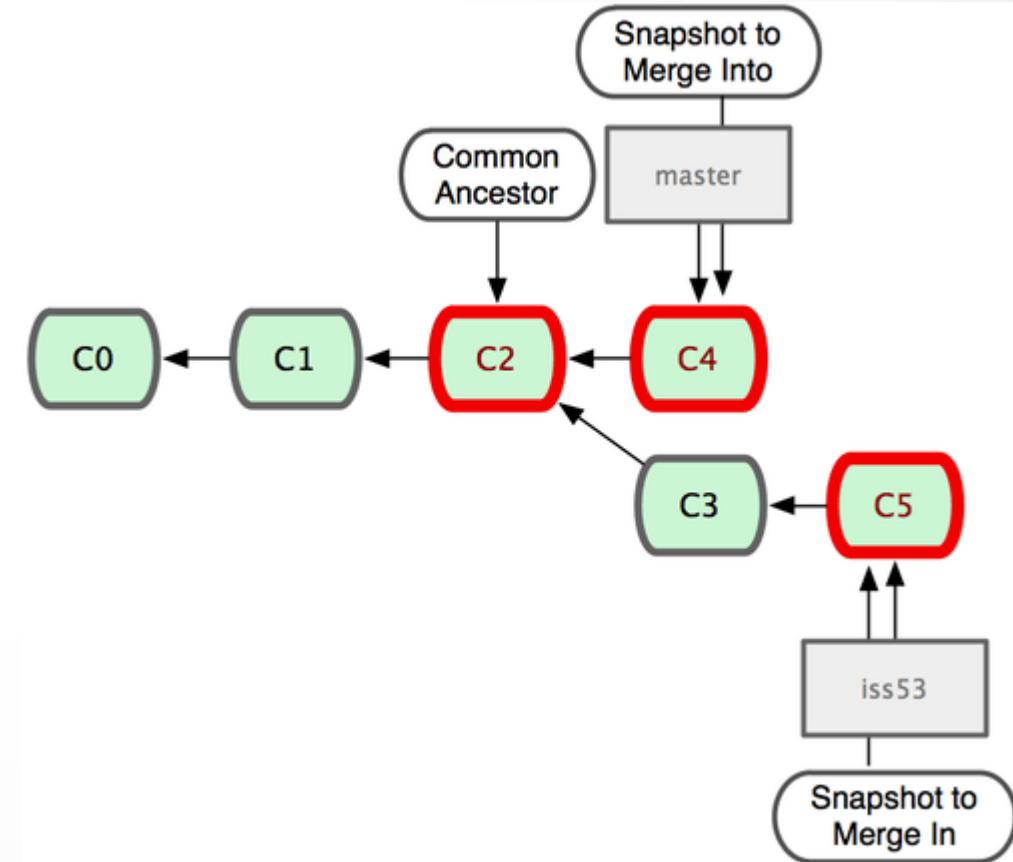
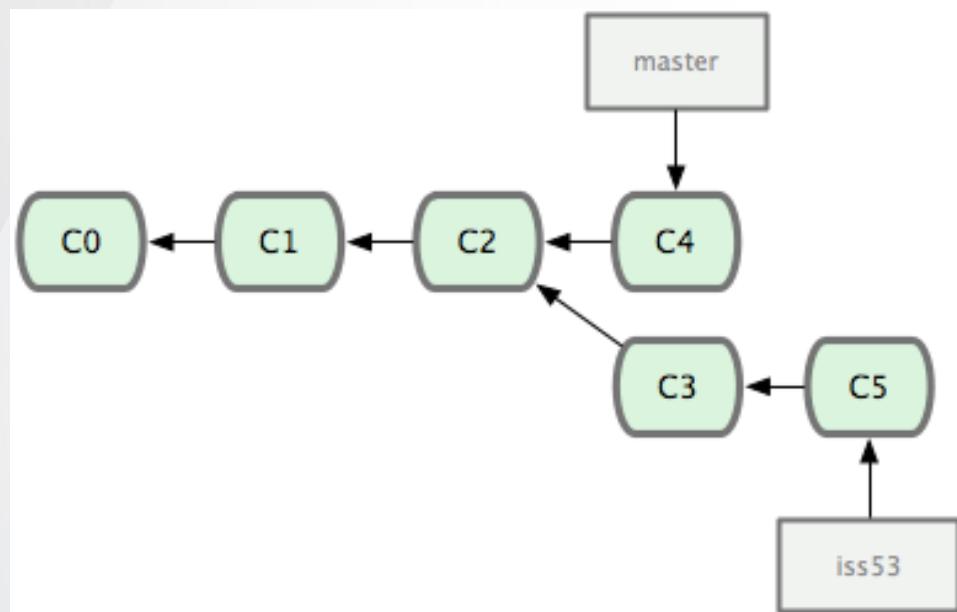


Gestion de sources avec Git: Fusion de deux branches

- Merger signifie créer un commit regroupant deux commits.
- La plupart des conflits sont résolus automatiquement. Il faudra cependant en résoudre certains soi-même manuellement.
- La commande Merge crée un nouveau commit ayant pour parents les deux branches fusionnées.
- La branche destinatrice du Merge est la branche courante, référencée par le HEAD. Seule la branche courante est modifiée, elle pointe sur le nouveau commit.

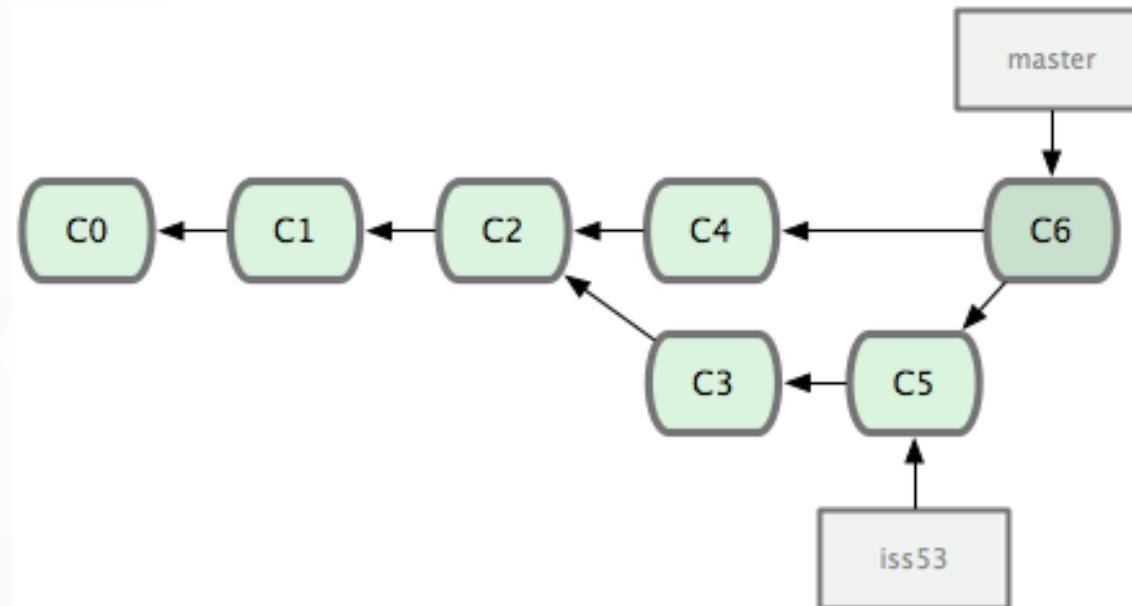
Gestion de sources avec Git: Fusion de deux branches

Avant le merge



Gestion de sources avec Git: Fusion de deux branches

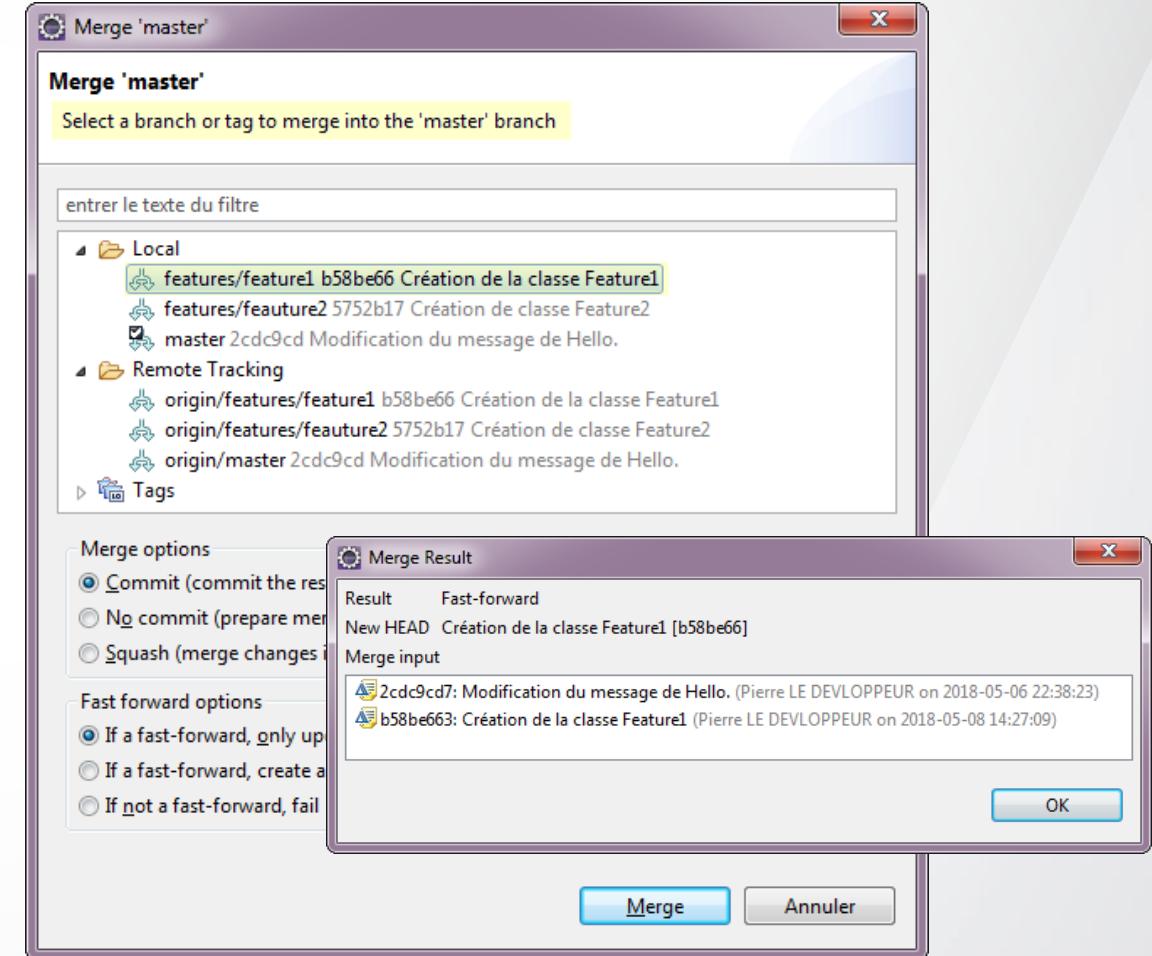
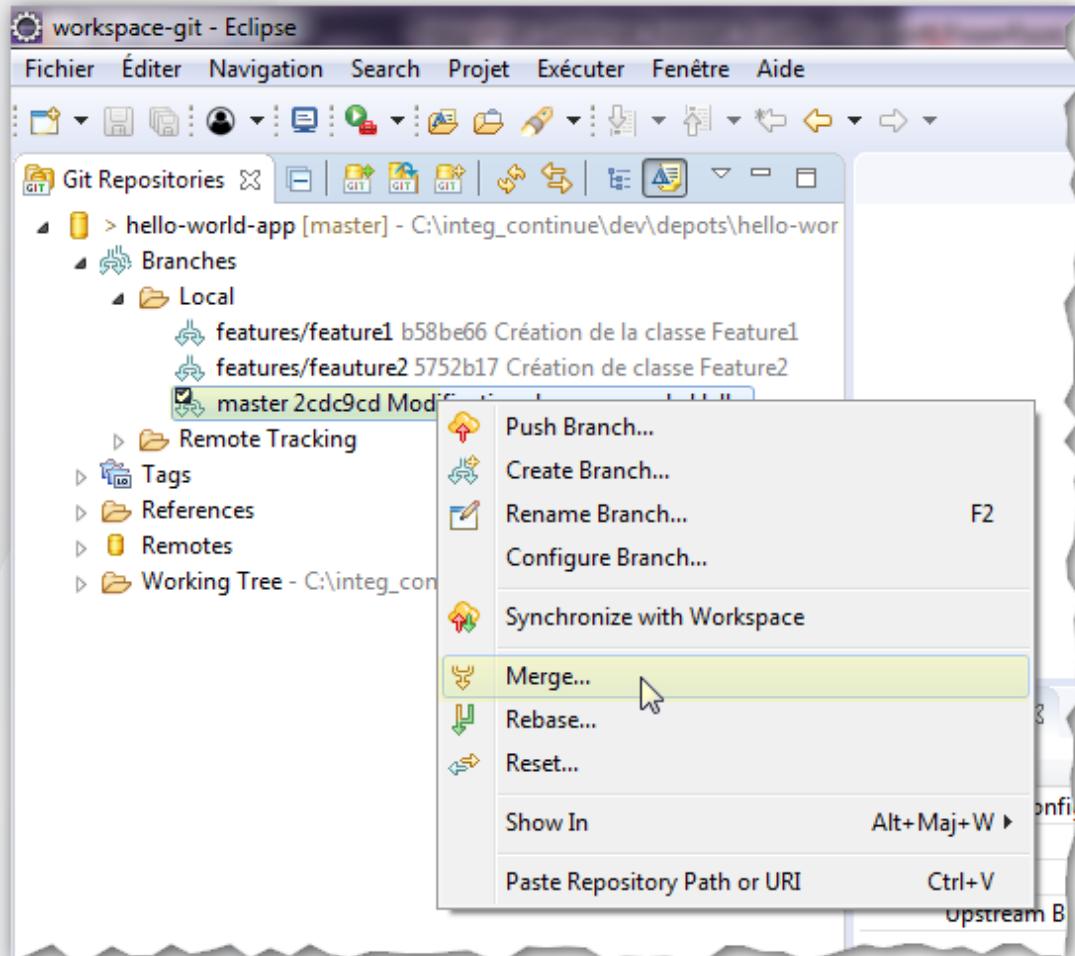
Résultat du merge



Gestion de sources avec Git: Fusion de deux branches

- Pour mettre à jour une branche par rapport à une autre, il faut au préalable positionner l'espace de travail sur la branche à mettre à jour par un Checkout.
- Dans la perspective Git, la vue Git Repositories se positionner sur la branche sur laquelle on souhaite positionner l'espace de travail en utilisant la commande Git Checkout.
- Depuis la branche source servant à la mettre à jour la branche cible, sélectionner le menu contextuel Merge.

Gestion de sources avec Git: Fusion de deux branches



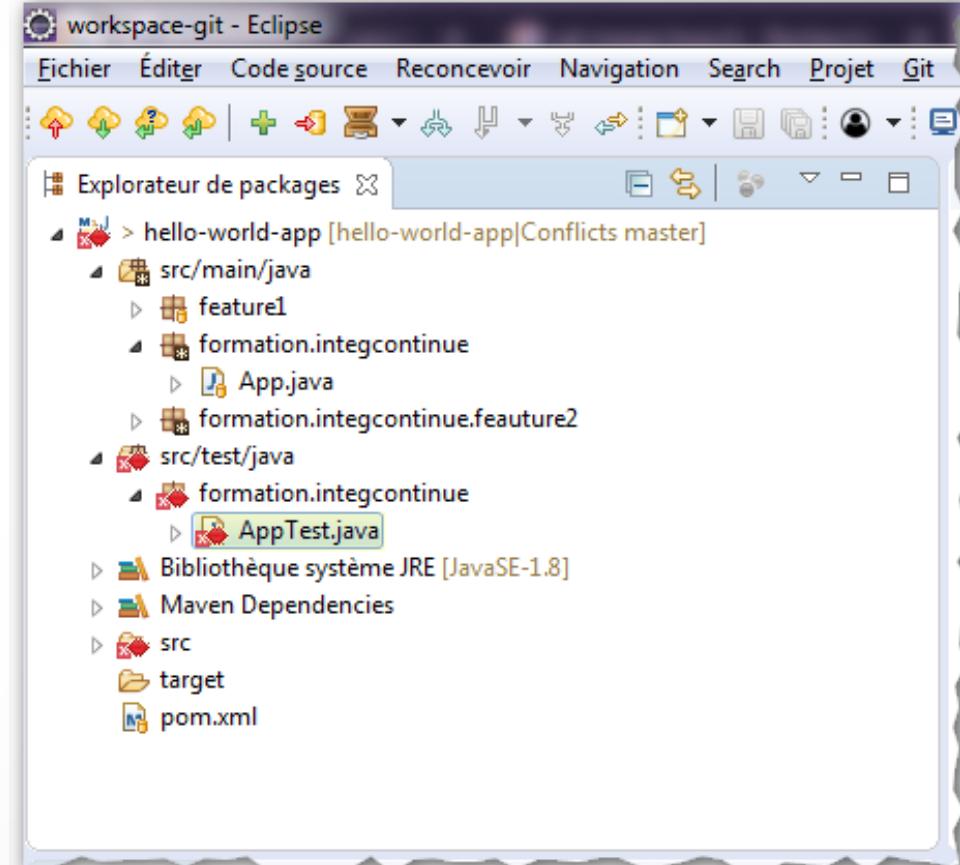
Gestion de sources avec Git: Fusion de deux branches

- En cas de conflit lors de la fusion de deux branches:
 - Retrouver le fichier problématique, en repérant les fichiers décorés avec une icone rouge.
 - Résoudre le ou les conflits en utilisant l'outil Merge Tool.
 - Ajouter le fichier résolu à l'index.
 - Lors tous les conflits sont résolus, faire un commit.
- Si la fusion des deux branches n'a pas produit de conflit le résultat du Merge est commité automatiquement.

Gestion de sources avec Git: Fusion de deux branches

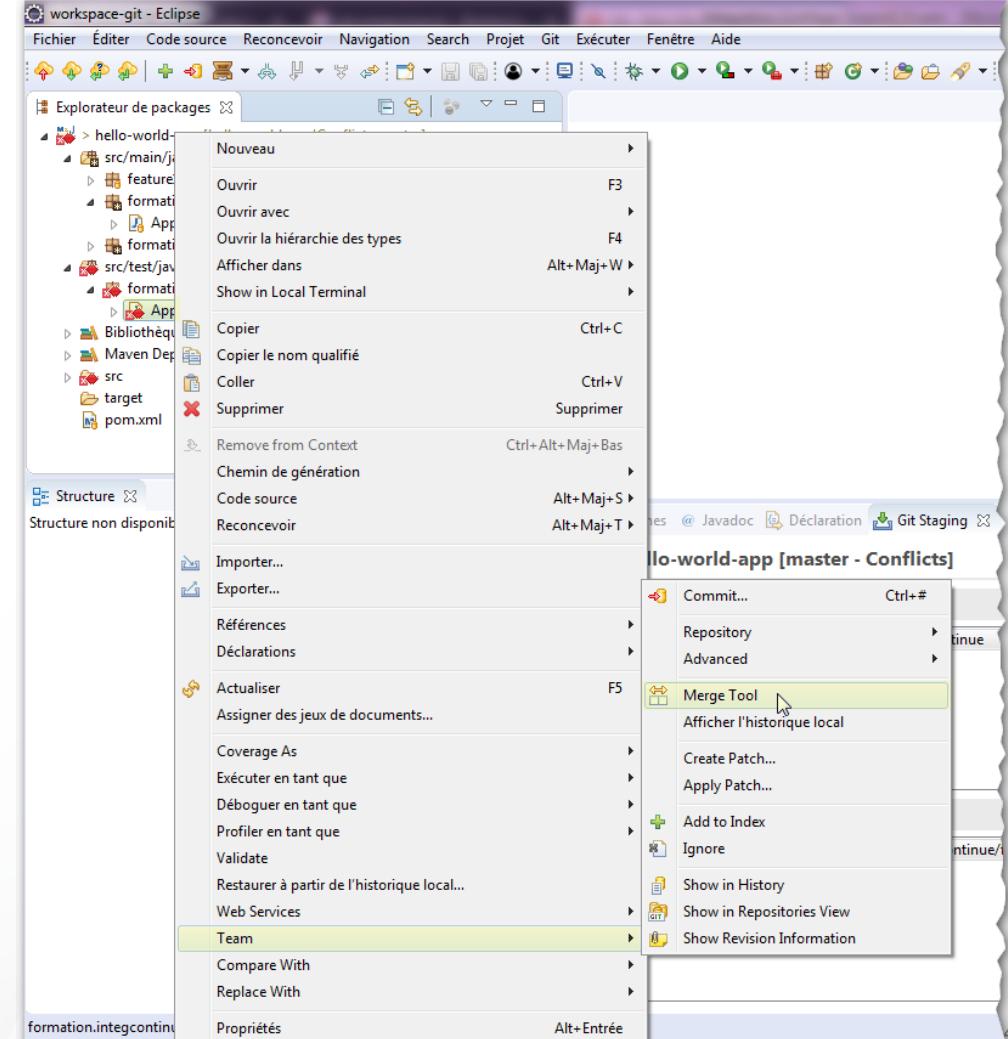
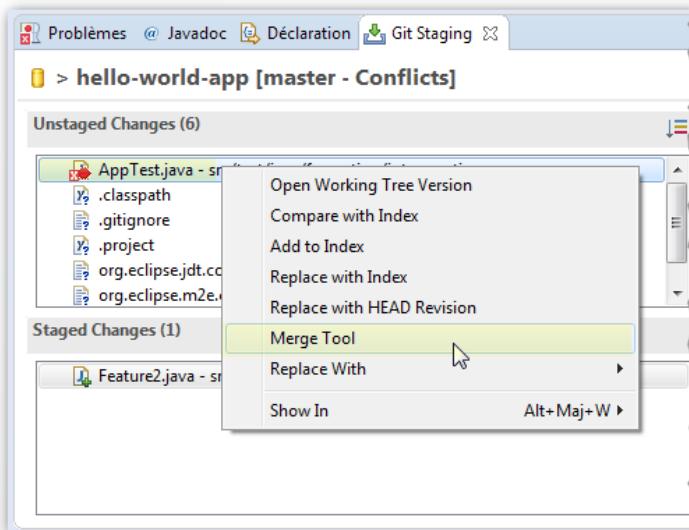
- L'élément en conflit est marqué d'une icone rouge.
- Dans le fichier source en conflit, les contributions des deux branches sont affichées.

```
AppTest.java
27
28 <<<<< HEAD
29 /**
30 * Rigourous Test :-)
31 */
32 public void testApp()
33 {
34     assertTrue( true );
35     assertTrue( false );
36 }
37 =====
38 /**
39 * Rigourous Test :-)
40 */
41 public void testApp() {
42     assertTrue(true);
43 }
44 >>>>> refs/heads/features/feauture2
45 }
```



Gestion de sources avec Git: Fusion de deux branches

- L'outil de merge aide à résoudre le conflit.
- Il s'active via le menu contextuel Team > Merge Tool de la perspective Java, vue Explorateur de package ou par le menu contextuel Merge Tool sur le fichier en conflit dans la vue Git Staging.



Gestion de sources avec Git: Fusion de deux branches

The screenshot shows a Java code comparison interface with two panes. The left pane displays the 'Comparaison de la structure' (Structure Comparison) for the file 'AppTest.java' located at 'formation/integcontinue'. The right pane displays the 'Comparaison des structures Java' (Java Structure Comparison) for the class 'AppTest'. Below these are two code editors:

- Modification du test - a0578b2:** This editor shows the original code from the 'suite()' method.

```
21 }
22
23 /**
24 * @return the suite of tests being tested
25 */
26 public static Test suite()
27 {
28     return new TestSuite( AppTest.class );
29 }
30
31 /**
32 * Rigorous Test :-)
33 */
34 public void testApp()
35 {
36     assertTrue( true );
37     assertTrue( false );
38 }
39 }
```
- Formatage du code - 6a930fc:** This editor shows the formatted code from the 'suite()' method.

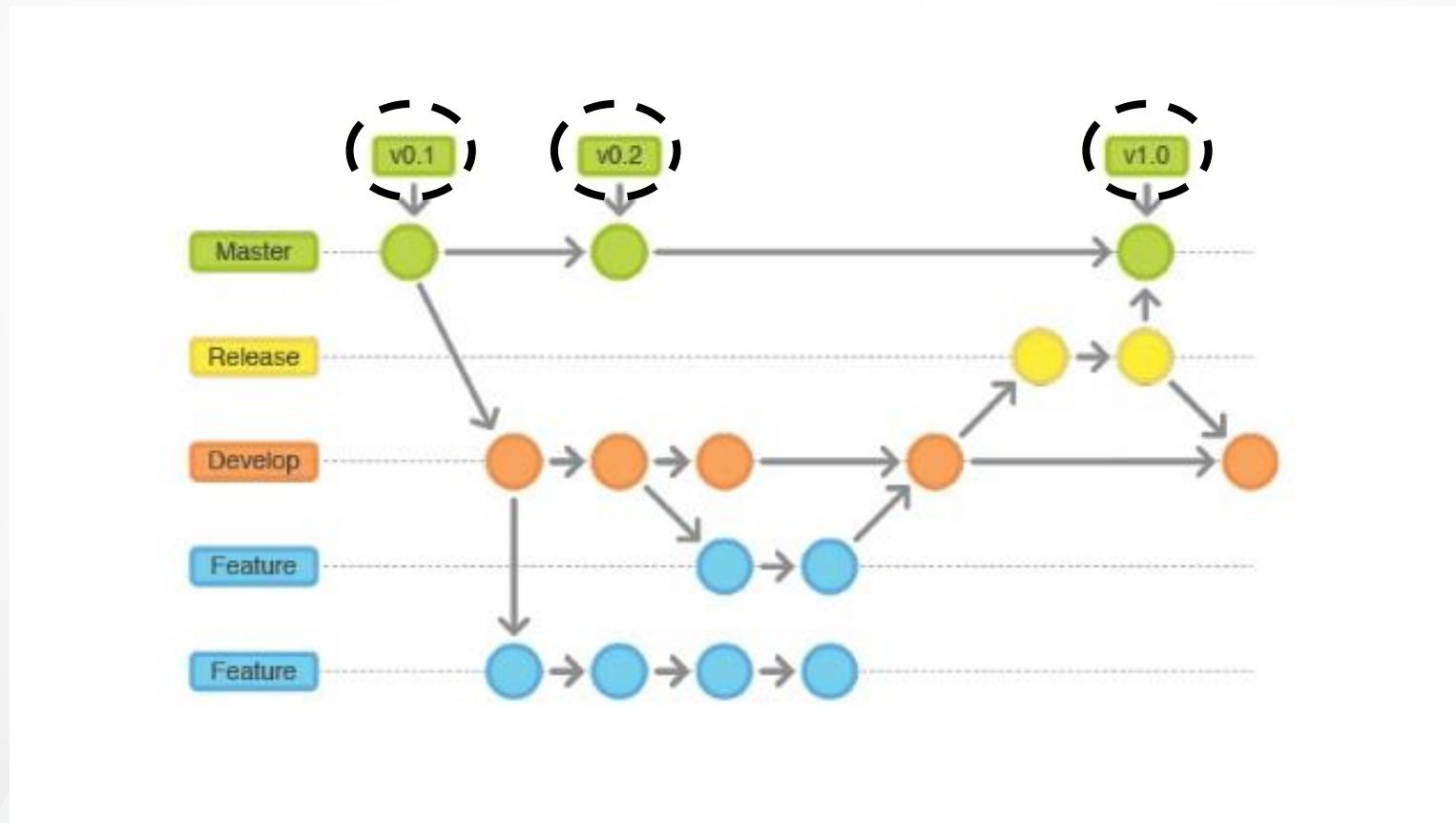
```
17     public AppTest(String testName) {
18         super(testName);
19     }
20
21 /**
22 * @return the suite of tests being tested
23 */
24 public static Test suite(){
25     return new TestSuite(AppTest.class);
26 }
27
28 /**
29 * Rigorous Test :-)
30 */
31 public void testApp(){
32     assertTrue(true);
33 }
34 }
35 }
```

The code in the 'testApp()' method of the left editor is highlighted with a red rectangle, indicating it is a conflict or a change specific to that branch.

Gestion de sources avec Git : Créer un Tag

- Le tag est une référence fixe vers certains commits.
- Le but d'un tag est de pouvoir identifier facilement un commit particulier, par exemple une version d'un logiciel.
- Il existe 2 types de tag, le tag simple qui est une référence basique vers un commit et le tag annoté qui contient des informations supplémentaires.

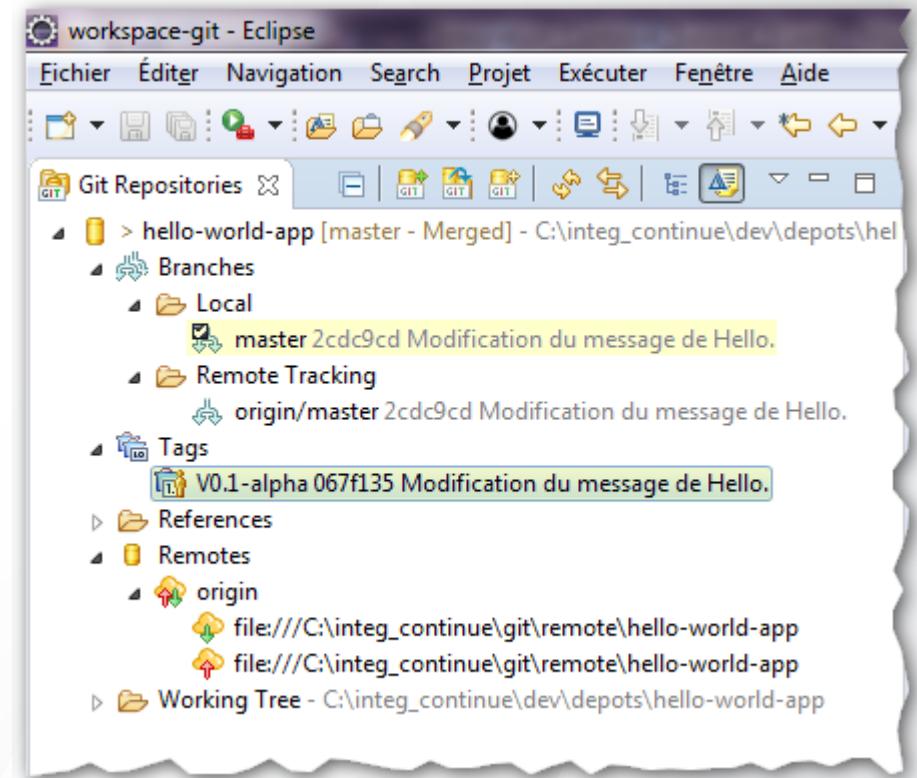
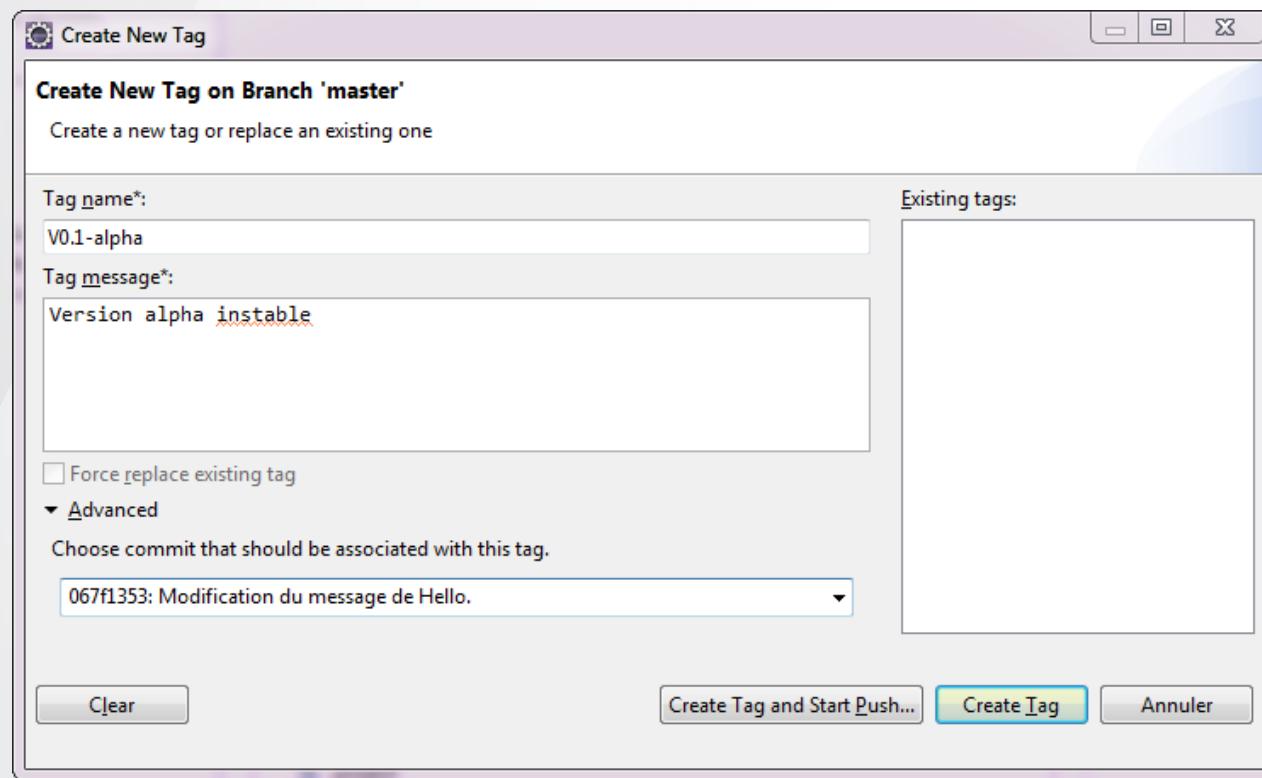
Gestion de sources avec Git : Créer un Tag



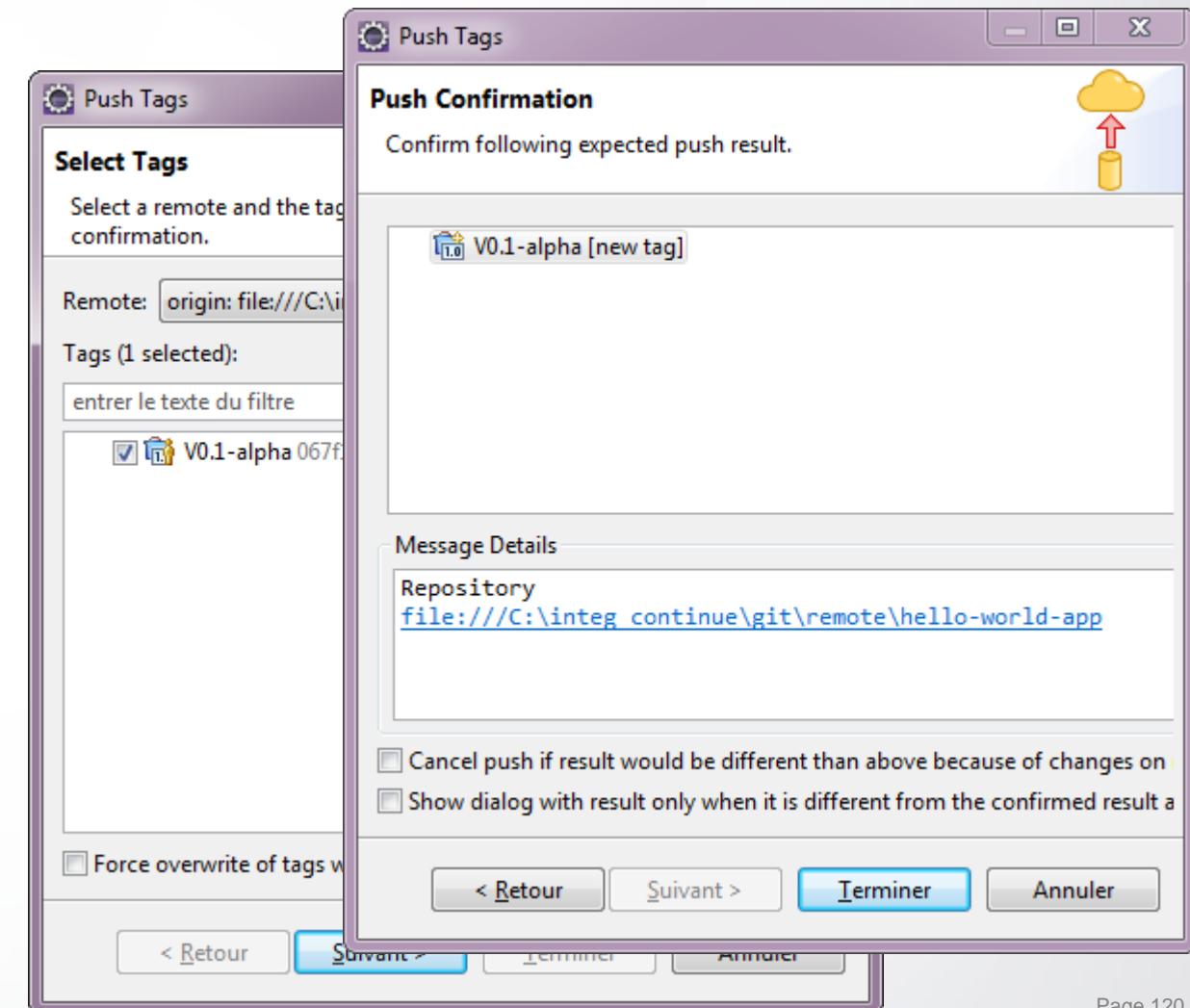
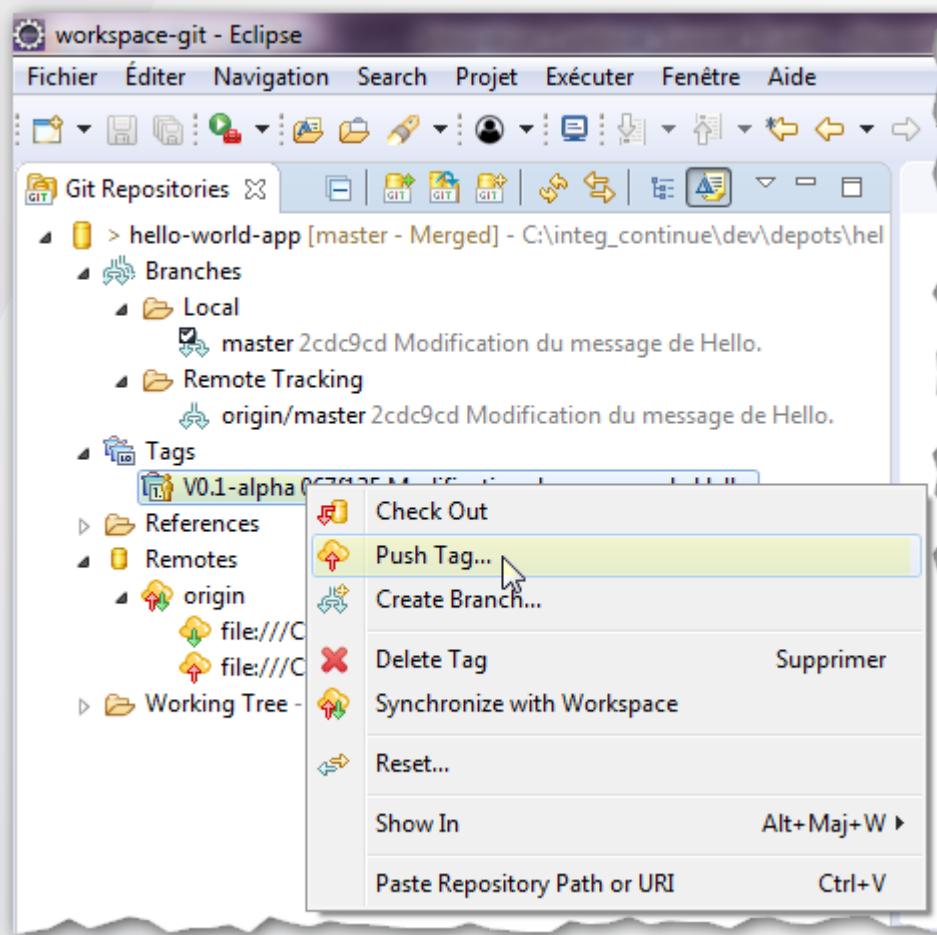
Gestion de sources avec Git : Créer un Tag

- Les tags créés depuis Eclipse sont associés par défaut avec le dernier commit de la branche en cours.
- Dans la perspective Git, vérifier que la branche sur laquelle il est souhaité d'appliquer le tag est bien la branche active.
- Activer le menu contextuel de la vue Git Repositories sur l'élément tags.
- Fournir un nom et une description du tag.
- Choisir le commit à associer avec le tag.
- Si le tag doit être partagé avec le dépôt central, un push du tag sera alors nécessaire.

Gestion de sources avec Git : Créer un Tag

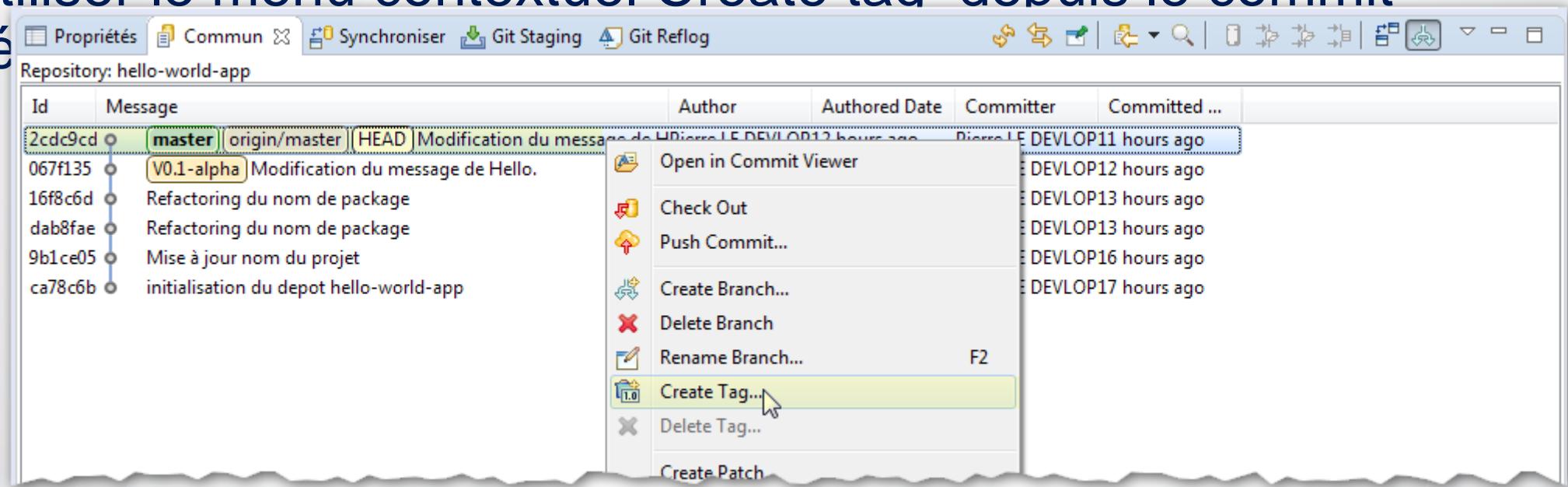


Gestion de sources avec Git : Créer un Tag



Gestion de sources avec Git : Créer un Tag

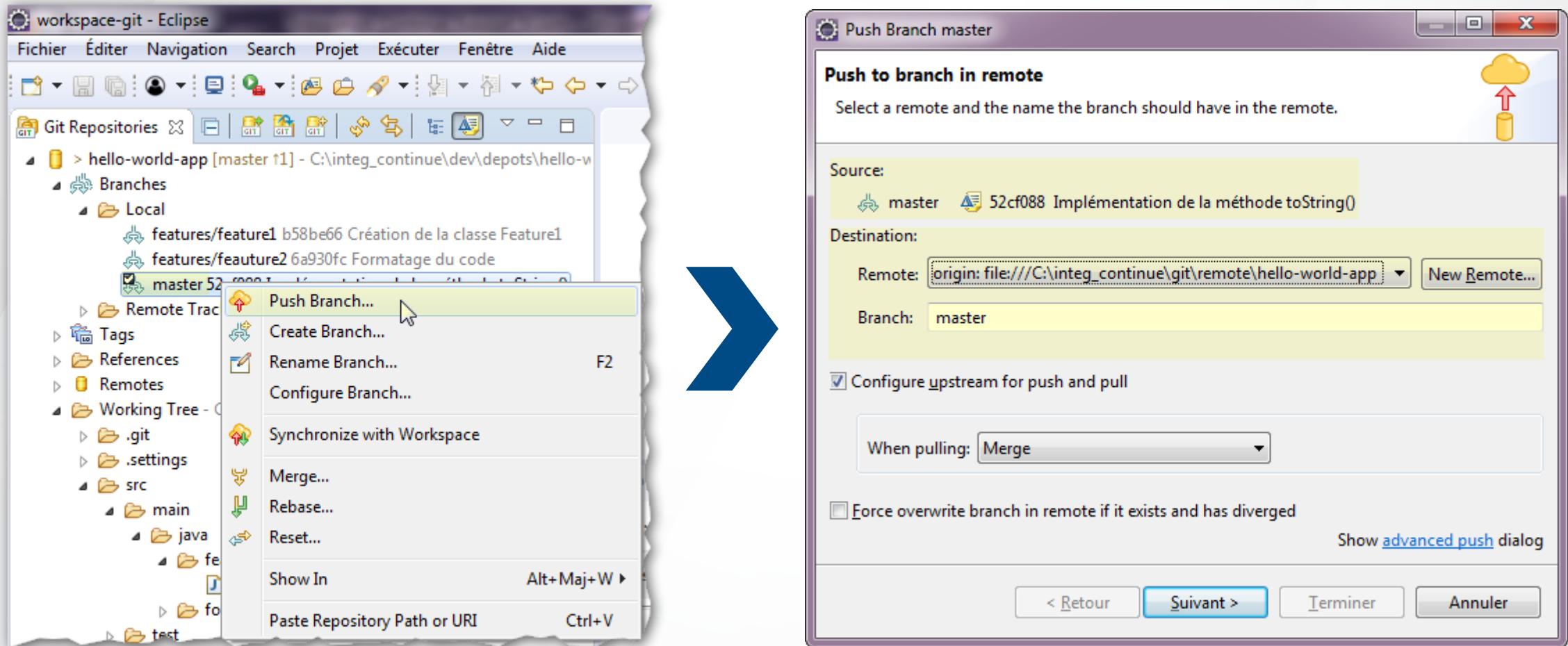
- Il est également possible de créer un tag sur un commit ou une branche depuis l'historique des changements dans la perspective Git.
- Utiliser le menu contextuel Create tag depuis le commit sélectionné



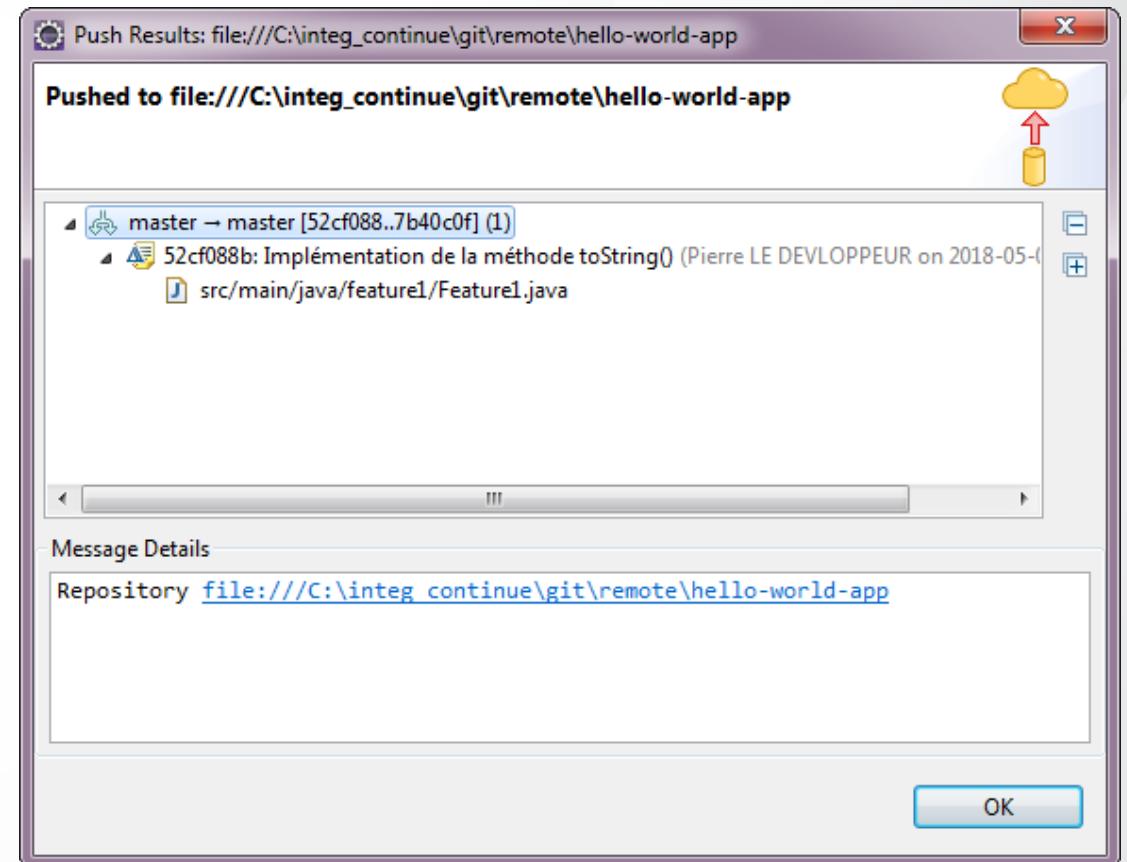
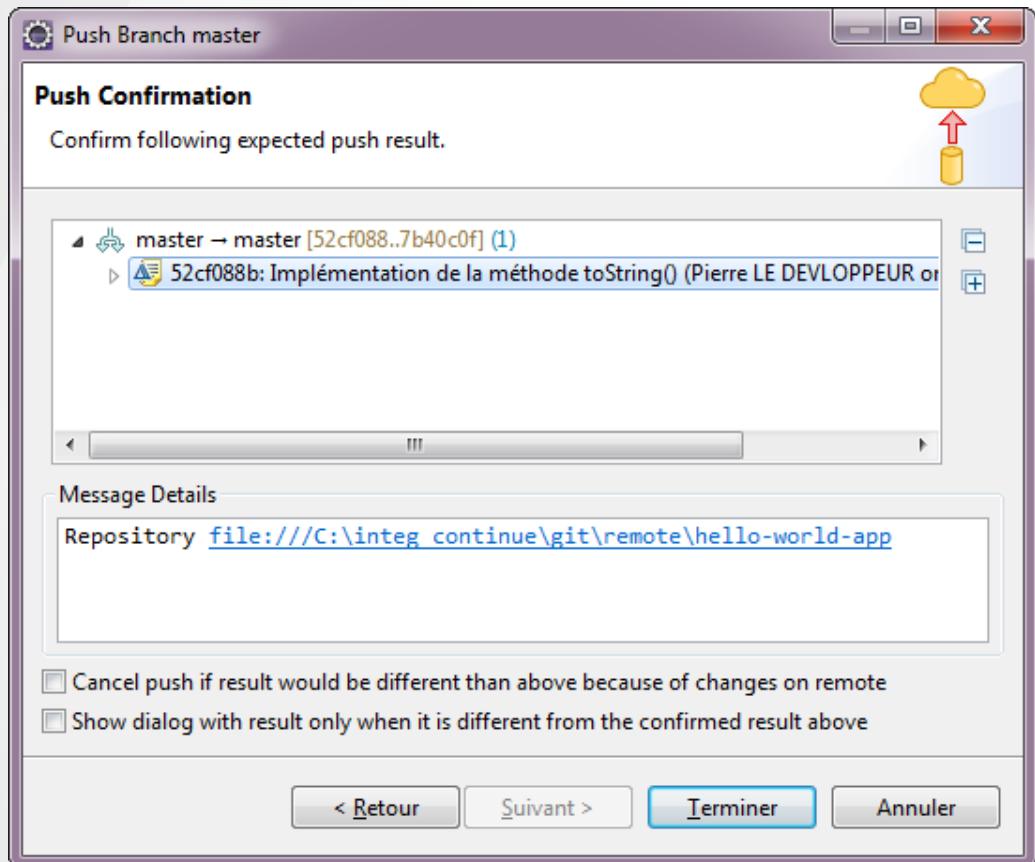
Gestion de sources avec Git: Pousser des changements vers un dépôt distant

- Pour partager le travail effectué, les nouveaux commits sont envoyés vers le dépôt distant.
- Sans la perspective Git, Vue Git Repositories vérifier que la branche courante est bien celle souhaitée.
- Utiliser le menu contextuel Push Branch depuis l'élément Branches > <nom de la branche> de la vue Git Repositories.
- Sélectionner le dépôt distant et visualiser les commits à envoyer et .

Gestion de sources avec Git: Pousser des changements vers un dépôt distant

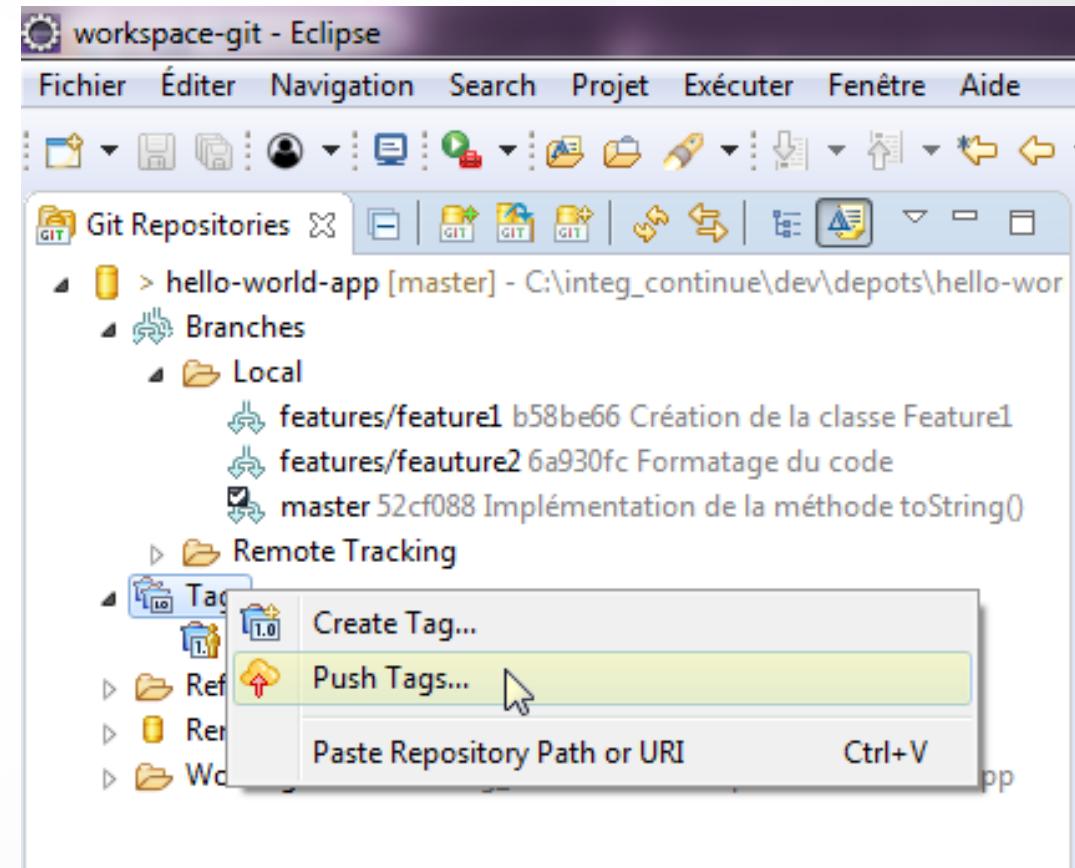


Gestion de sources avec Git: Pousser des changements vers un dépôt distant

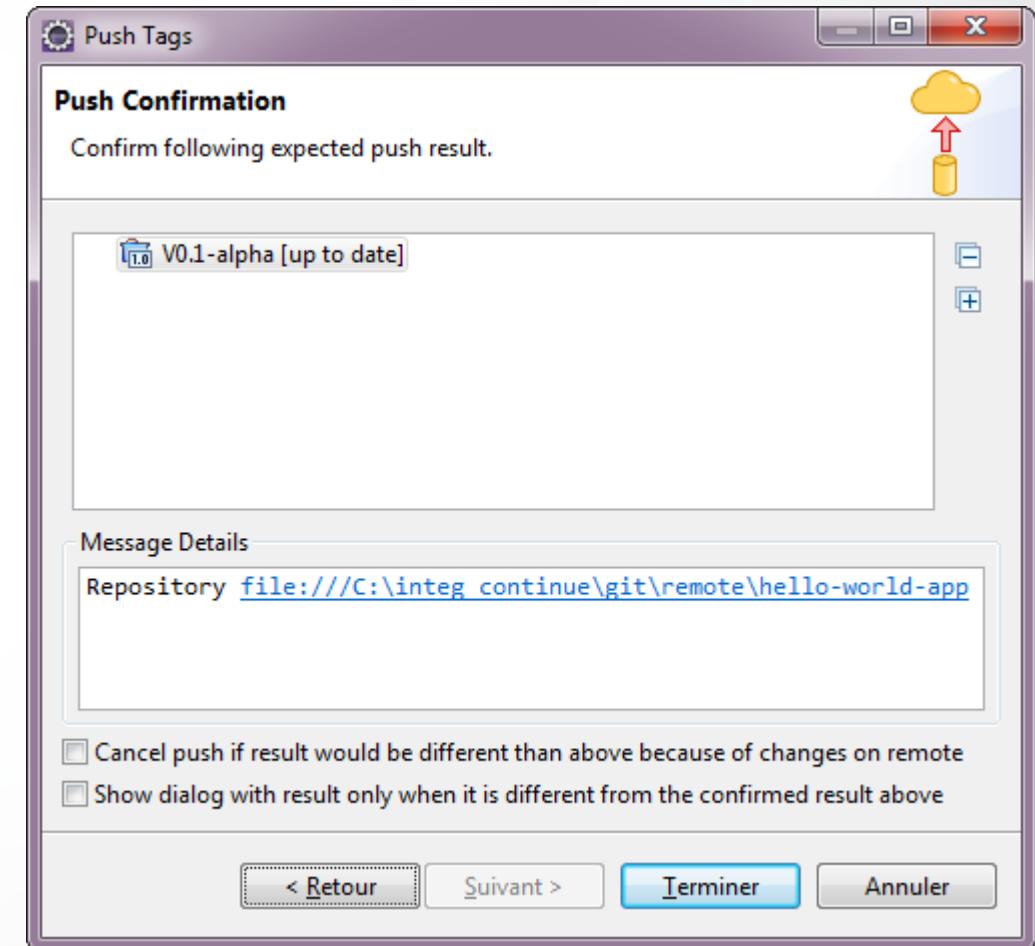
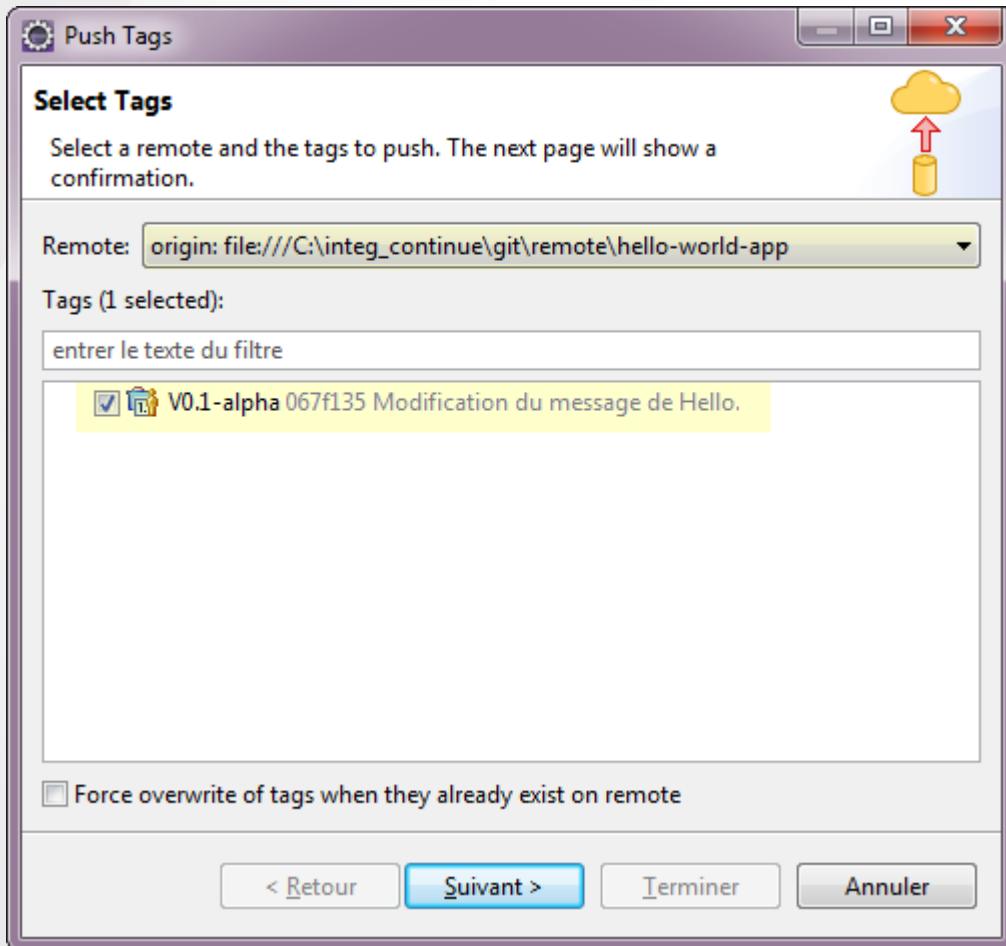


Gestion de sources avec Git: Pousser des changements vers un dépôt distant

- Les tags doivent être envoyés vers le dépôt distant de manière indépendante des branches.
- Utiliser le menu contextuel Push Tags depuis l'élément Tags de la vue Git Repositories.
- Sélectionner les tags à envoyer et visualiser la confirmation.



Gestion de sources avec Git: Pousser des changements vers un dépôt distant

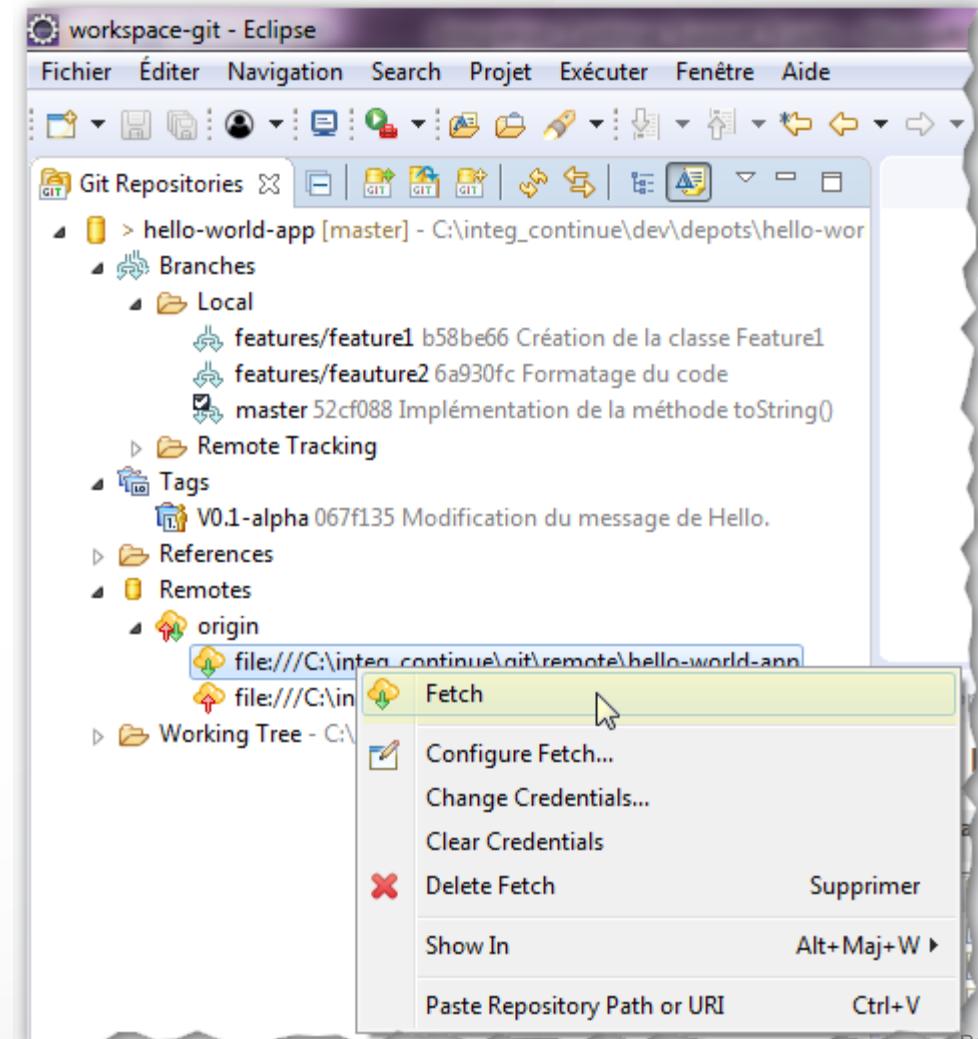


Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant

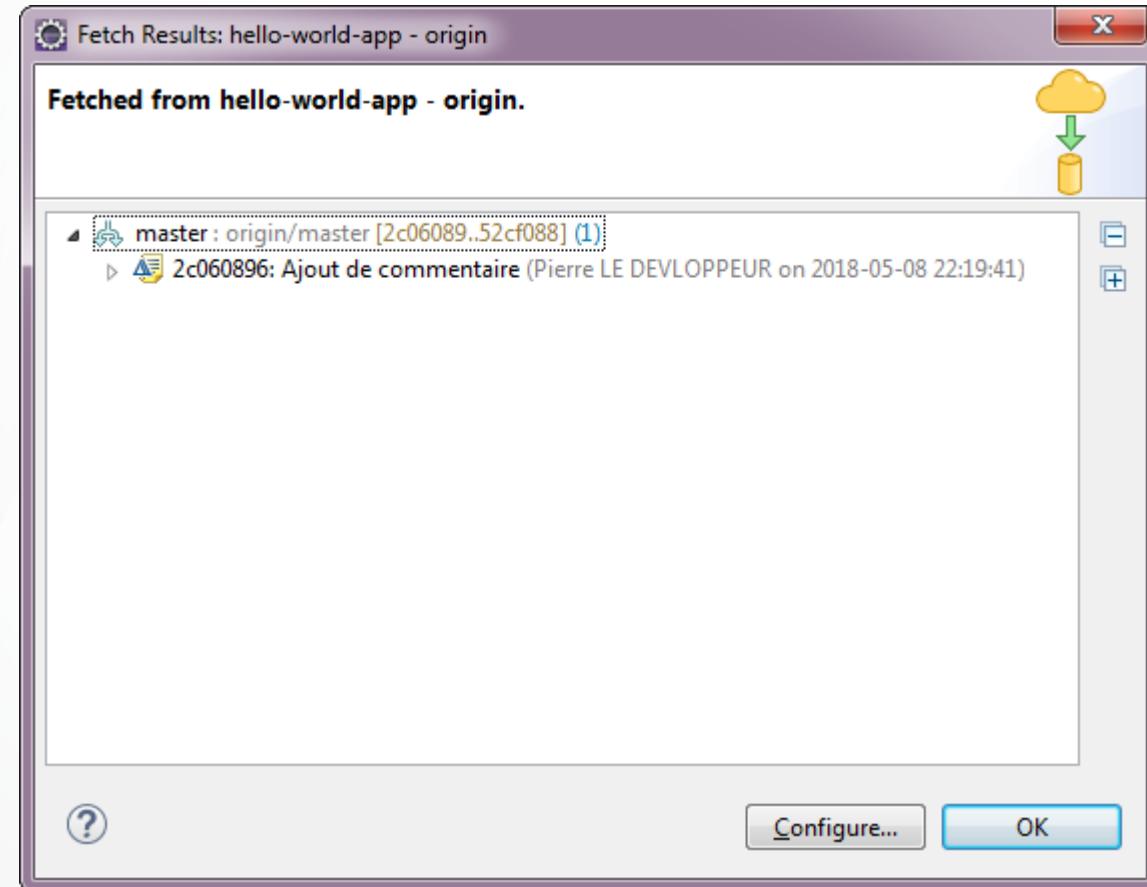
- Les changements sont récupérés avec la commande Fetch.
- Les informations récupérées sont uniquement les références disponibles ainsi que tous les commits associés.
- Cette récupération ,ne nécessite pas de récupérer tout le contenu du dépôt distant, seulement la différence.
- Aucune référence n'est modifiée en local, seuls de nouveaux objets sont ajoutés à la base locale.
- L'espace de travail n'est pas mis à jour. Pour le mettre à jour, il faut ensuite fusionner la branche origine/master du dépôt distant avec la branche master du dépôt local sur laquelle est positionné l'espace de travail.

Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant

- La commande Fetch s'exécute via le menu contextuel sur l'élément Remotes de la vue Git Repositories.
- Visualiser tous les commits récupérés.
- Le Merge n'étant pas fait automatiquement, il reste donc à faire.



Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant

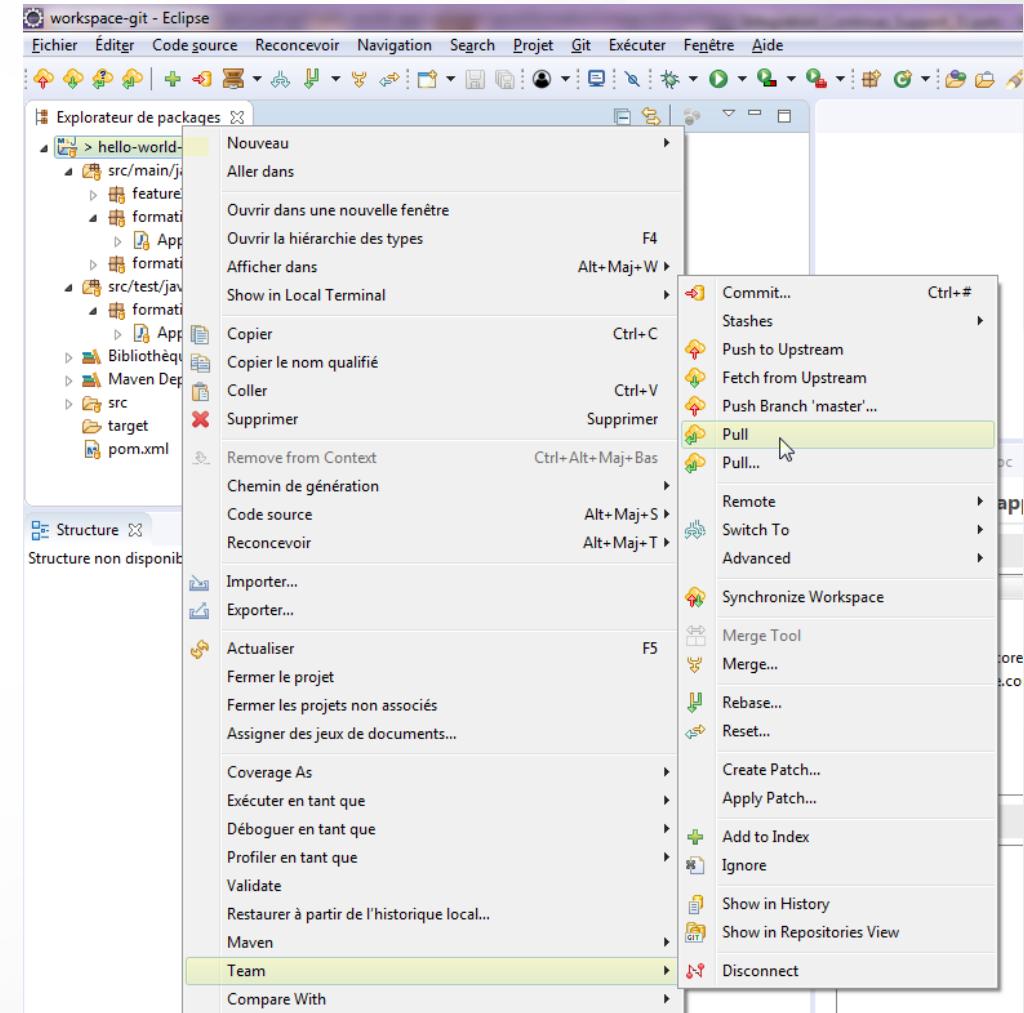


Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant

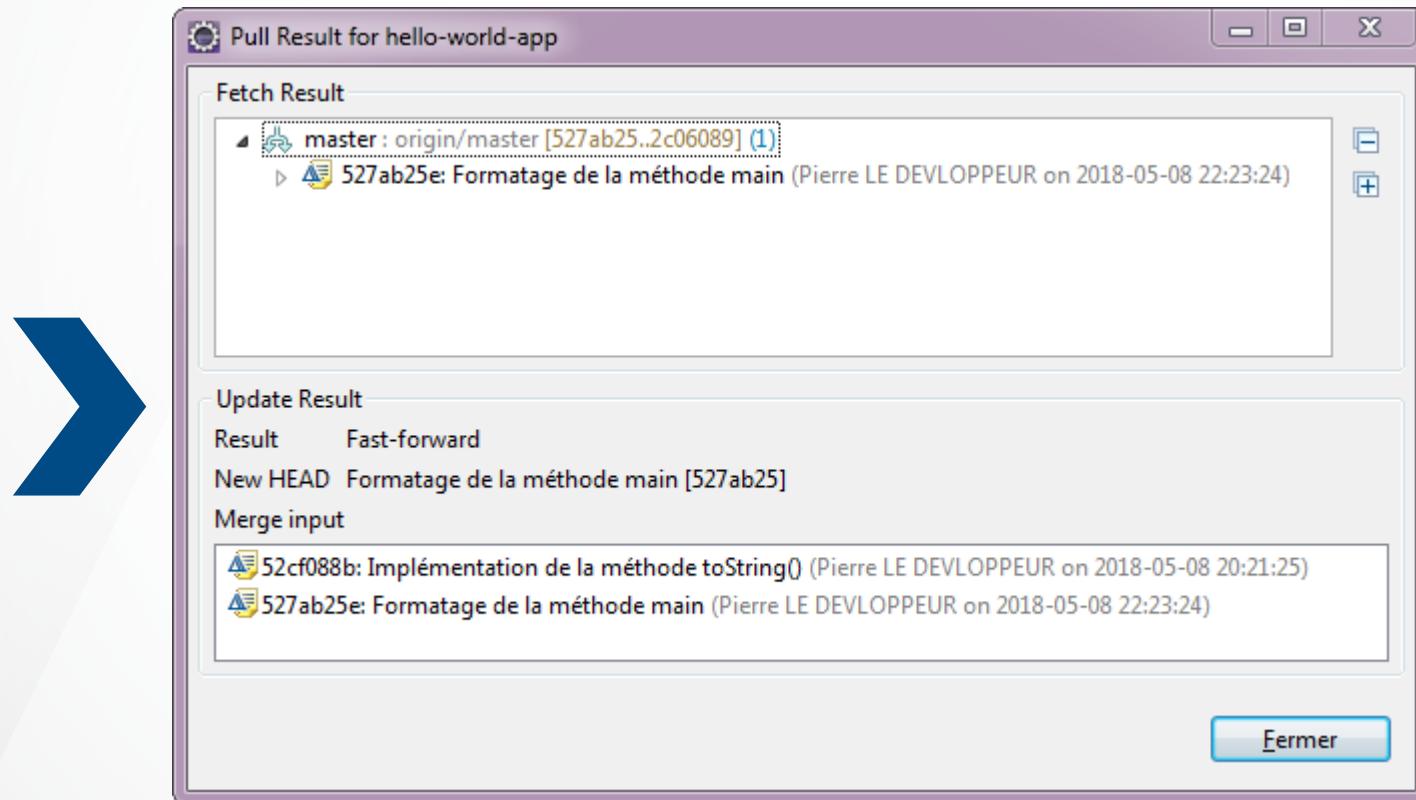
- Il est aussi possible de charger et incorporer directement les derniers changements apportés sur une branche en utilisant la commande Pull.
- La première action faite par cette commande est la même que la commande Fetch, c.a.d. que les derniers changements sont rapatriés.
- Après cette récupération, Git essaye de fusionner la branche origine/nom-branche avec la branche sur laquelle est positionné l'espace de travail avec un Merge automatique.

Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant

- La commande Pull s'exécute via le menu contextuel du projet Java Team > Pull.
- L'espace de travail étant mis à jour le pointeur de la branche courante est positionné sur le commit associé au Merge automatique.



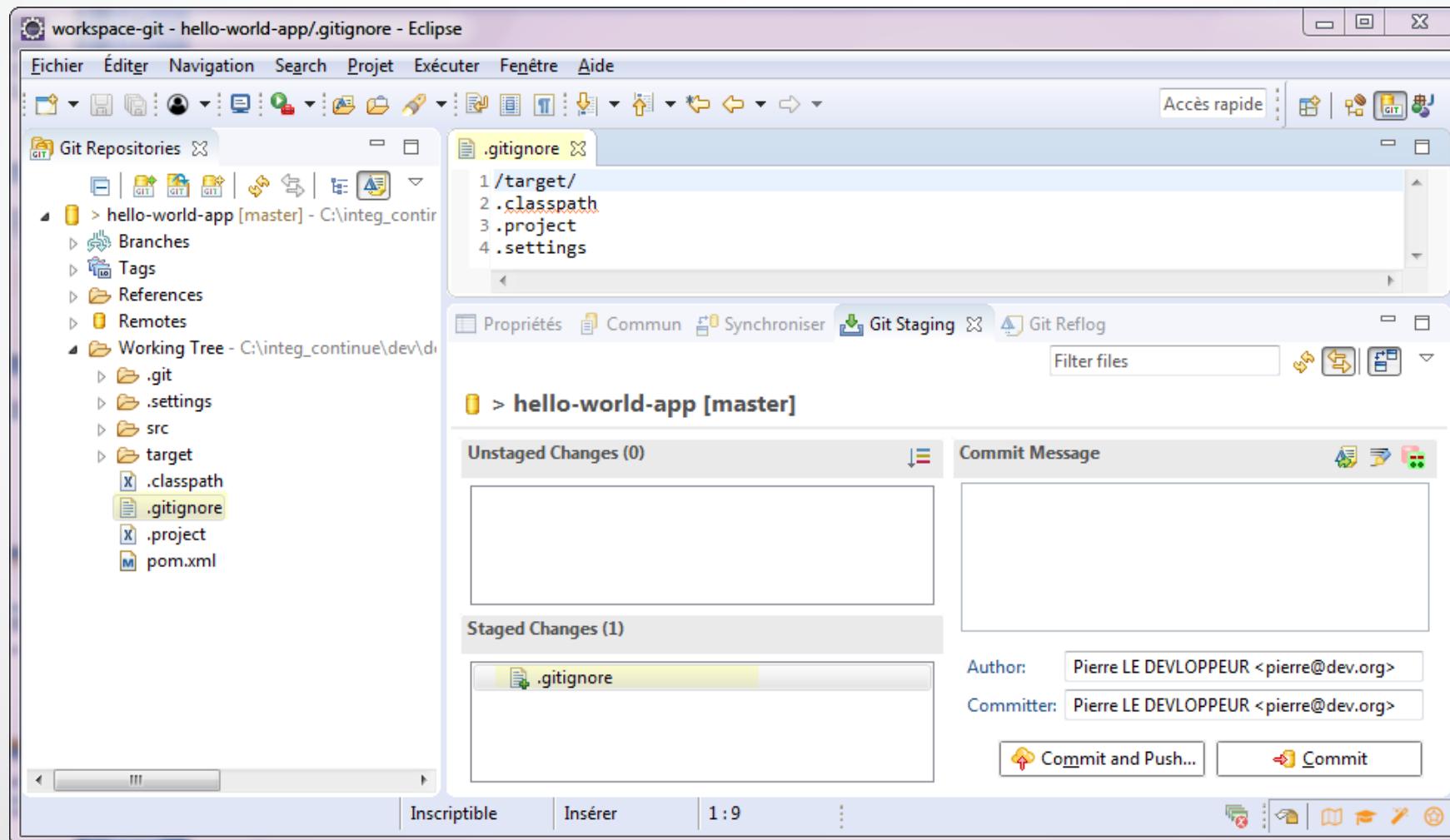
Gestion de sources avec Git: Récupérer des changements depuis un dépôt distant



Gestion de sources avec Git : Le fichier .gitignore

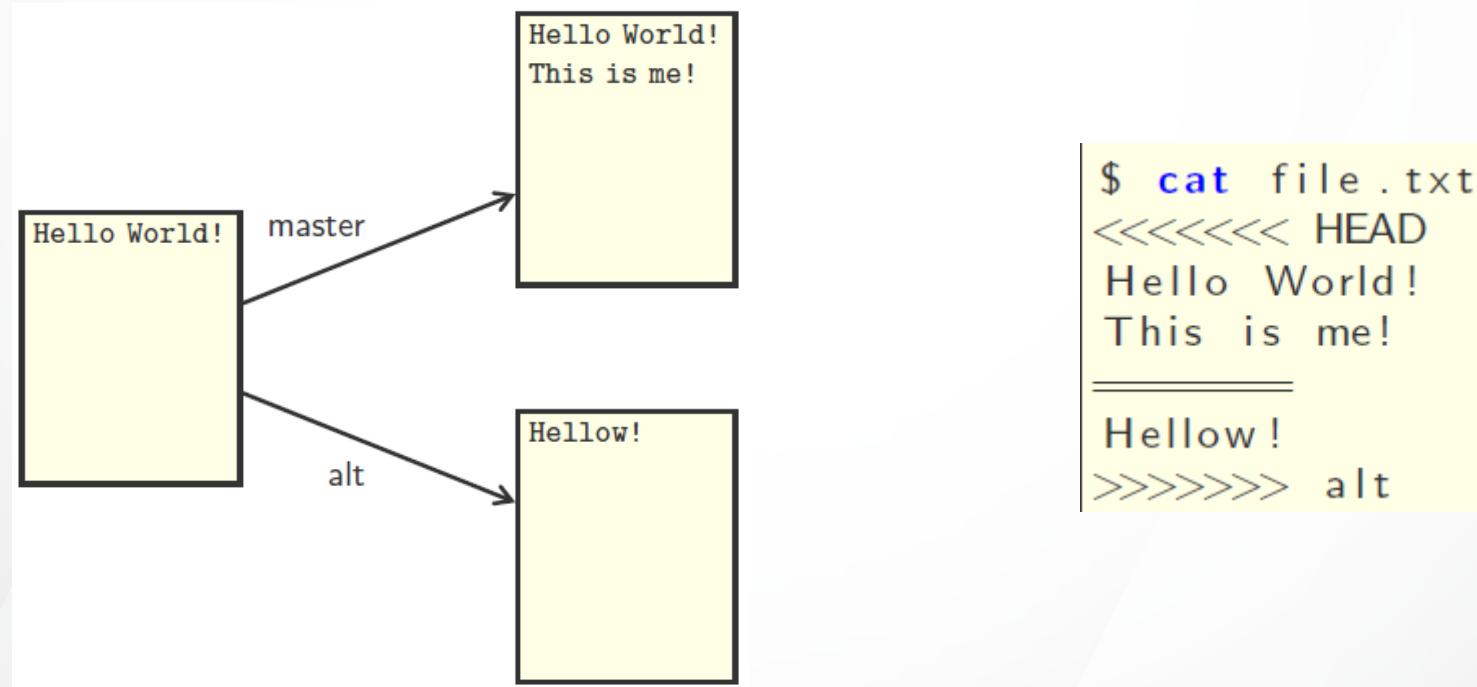
- Si vous ne souhaitez pas que certains fichiers ou répertoires soient mis en gestion de configuration sous git vous pouvez les exclure du suivi git en utilisant le fichier .gitignore.
- Par exemple le répertoire target, un répertoire temporaire produit par Maven, ne doit pas être ajouté à la gestion de la configuration.
- Il peut y avoir plusieurs fichiers .gitignore à différents niveaux de répertoire.

Gestion de sources avec Git : Le fichier .gitignore



Problématiques de fusion des changements

- Plus la quantité de codes publiés est grosse plus le risque de conflit augmente.



Problématiques de fusion des changements

- La publication régulière du code source :
 - Réduit le risque de conflit.
 - Facilite l'identification de la modification à l'origine du problème.
- Pour pouvoir publier régulièrement le code source il faut :
 - Publier les modifications apportées à la fin de chaque tâche de développement.
 - Faire des petits changements et ne pas apporter des changements à plusieurs composants dans une seule tâche.

Problématiques de fusion des changements : Bonnes pratiques

- Essayez de travailler le plus possible avec des fichiers texte et le moins possible avec des fichiers binaires.
- Faites des updates le plus souvent et régulièrement possible, et obligatoirement avant de commencer à travailler sur un fichier.
- Faites des commits réguliers, dès que vous avez fini de travailler sur un fichier, et si la tâche prend du temps, un commit intermédiaire permettra de sauvegarde votre travail.
- Prévoyez un planning pour savoir qui peut travailler et quand sur les fichiers binaires notamment.