

Développer avec Git - Formation TortoiseGit

Exercices

Sommaire

[Introduction 8](#_Toc428453537)

[1 - Exercice 1 – Configurer Git avec TortoiseGit 9](#_Toc428453538)

[1.1 Configurer le chemin vers le dossier bin de l’installation Git 9](#_Toc428453539)

[1.2 Configurer l’utilisateur 9](#_Toc428453540)

[2 - Exercice 2 – Cloner un dépôt distant 11](#_Toc428453541)

[2.1 Cloner le dépôt Git 11](#_Toc428453542)

[2.2 Installer les hooks dans le dépôt local 14](#_Toc428453543)

[3 - Exercice 3 – Enregistrer les changements : *Commit* 15](#_Toc428453544)

[3.1 Créer un fichier Fichier1.txt dans le répertoire du dépôt local 15](#_Toc428453545)

[3.2 Consigner ce nouveau fichier dans le dépôt local Git 15](#_Toc428453546)

[3.3 Recommencer avec un second fichier Fichier2.txt 17](#_Toc428453547)

[4 - Exercice 4 – Annuler un changement : *Revert change by this commit* 20](#_Toc428453548)

[4.1 Modifier un fichier et consigner ces modifications dans le dépôt Git 20](#_Toc428453549)

[4.2 Annuler le changement 20](#_Toc428453550)

[5 - Exercice 5 – Travailler sur une nouvelle branche de fonctionnalité 24](#_Toc428453551)

[5.1 Créer une branche 24](#_Toc428453552)

[5.2 Créer un nouveau fichier FichierFA.txt sur cette branche et enregistrer les changements sur cette branche 26](#_Toc428453553)

[5.3 Parcourir l’historique des révisions 26](#_Toc428453554)

[6 - Exercice 6 – Mettre à jour une branche par rapport à une autre avec un Rebase 30](#_Toc428453555)

[6.1 Changer de branche et se positionner sur la branche *develop* 30](#_Toc428453556)

[6.2 Ajouter un nouveau fichier Fichier3.txt et consigner les modifications sur cette branche 30](#_Toc428453557)

[6.3 Modifier le fichier Fichier3.txt et consigner les modifications 30](#_Toc428453558)

[6.4 Visualiser l’historique des révisions sur cette branche *develop* 31](#_Toc428453559)

[6.5 Visualiser les fichiers présents dans la branche de fonctionnalité et l’historique des révisions 31](#_Toc428453560)

[6.6 Mettre à jour la branche de fonctionnalité par rapport à la branche *develop* grâce à un Rebase 33](#_Toc428453561)

[6.7 Parcourir l’arbre de révision sur la branche de fonctionnalité et sur la branche *develop* 35](#_Toc428453562)

[6.8 Schéma récapitulatif de l’état des 2 branches avant et après le *Rebase* 37](#_Toc428453563)

[7 - Exercice 7 – Mettre à jour une branche par rapport à une autre avec un Merge 38](#_Toc428453564)

[7.1 Vérifier que l’espace de travail est positionné sur la branche de fonctionnalité 38](#_Toc428453565)

[7.2 Ajouter un nouveau fichier et consigner les modifications sur cette branche 38](#_Toc428453566)

[7.3 Ajouter un nouveau fichier sur la branche *develop* et consigner les modifications sur cette branche 39](#_Toc428453567)

[7.4 Mettre à jour la branche *develop* par rapport à la branche de fonctionnalité grâce à un Merge 39](#_Toc428453568)

[7.5 Parcourir l’arbre de révision sur la branche *develop* et sur la branche de fonctionnalité 40](#_Toc428453569)

[7.6 Schéma récapitulatif de l’état des 2 branches avant et après le *Merge* 42](#_Toc428453570)

[8 - Exercice 8 – Gérer les conflits lors d’un *Merge* 43](#_Toc428453571)

[8.1 Modifier le fichier Fichier2.txt depuis une nouvelle branche de fonctionnalité. Consigner les changements sur cette branche. 43](#_Toc428453572)

[8.2 Faire une autre modification dans le fichier Fichier2.txt depuis la branche *develop* 43](#_Toc428453573)

[8.3 Mettre à jour la branche *develop* par rapport à la branche de fonctionnalité grâce à un *Merge* 44](#_Toc428453574)

[8.4 Visualiser le conflit lors du *Merge* 45](#_Toc428453575)

[8.5 Résoudre le conflit avec l’éditeur de conflit 45](#_Toc428453576)

[8.6 Terminer le *Merge* en effectuant un commit du fichier modifié 47](#_Toc428453577)

[9 - Exercice 9 – Récupérer les changements apportés par un commit et les reproduire par un *Cherry Pick this commit …* 50](#_Toc428453578)

[9.1 Se positionner sur la branche de fonctionnalité *feature/fonctionnaliteA* 50](#_Toc428453579)

[9.2 Visualiser l’arbre de révision de toutes les branches et repérer le commit correspondant à la création du fichier Fichier41.txt. Reproduire ce commit sur la branche de fonctionnalité feature/fonctionnaliteA 50](#_Toc428453580)

[9.3 Visualiser l’arbre de révision de toutes les branches après le *Cherry Pick this commit…* 53](#_Toc428453581)

[10 - Exercice 10 – Poser une étiquette pour référencer une version et corriger la version 54](#_Toc428453582)

[10.1 Créer une branche de version, faire une correction de bug et consigner les modifications 54](#_Toc428453583)

[10.2 Poser une étiquette sur la version candidate à la mise en production 56](#_Toc428453584)

[10.3 Développer sur la branche *develop* et enregistrer les modifications 57](#_Toc428453585)

[10.4 Apporter une correction sur la branche R1 et enregistrer les modifications 58](#_Toc428453586)

[10.5 Reporter les corrections effectuées sur la branche de version sur la branche develop grâce à un Merge 59](#_Toc428453587)

[10.6 Schéma récapitulatif de l’état des 3 branches avant et après le *Merge* 61](#_Toc428453588)

[11 - Exercice 11 – Pousser et récupérer des contributions 62](#_Toc428453589)

[11.1 Faire un deuxième clone du dépôt distant 62](#_Toc428453590)

[11.2 Visualiser qu’aucun des fichiers créés dans le précédent dépôt local ne sont présents 62](#_Toc428453591)

[11.3 Pousser les contributions effectuées précédemment sur le premier dépôt local sur le dépôt distant 62](#_Toc428453592)

[11.4 Récupérer sur le deuxième dépôt local les contributions précédemment poussées sur le dépôt distant 64](#_Toc428453593)

[11.5 Faire un troisième clone du dépôt distant. Vérifier que ce dépôt local n’est pas vide 69](#_Toc428453594)

[11.6 Développer sur le troisième dépôt local, enregistrer les changements et les pousser sur le dépôt distant 70](#_Toc428453595)

[11.7 Développer sur le premier dépôt local, enregistrer les changements mais avant de les pousser sur le dépôt distant, récupérer les contributions développées sur le troisième dépôt local 70](#_Toc428453596)

[11.8 Pour finir, et être complètement cohérent, mettre à jour le troisième dépôt local avec les contributions poussées sur de dépôt distant depuis le premier dépôt local 73](#_Toc428453597)

**Liste des figures**

[Figure 1- Menu contextuel *TortoiseGit > Settings* 9](#_Toc427918719)

[Figure 2- Configuration du chemin vers le dossier bin de l’installation Git 9](#_Toc427918720)

[Figure 3 - Fenêtre après modification du radio-bouton *Global* 10](#_Toc427918721)

[Figure 4 - Configuration Utilisateur 10](#_Toc427918722)

[Figure 5 - Choix du menu contextuel *Git Clone …* 11](#_Toc427918723)

[Figure 6 – URL du dépôt distant 11](#_Toc427918724)

[Figure 7 - User pour se connecter au dépôt distant 12](#_Toc427918725)

[Figure 8 - Mot de passe pour se connecter au dépôt distant 13](#_Toc427918726)

[Figure 9 - Fin de l'opération Clone d'un dépôt distant 13](#_Toc427918727)

[Figure 10 - Résultat Clone du dépôt distant 14](#_Toc427918728)

[Figure 11 - Menu contextuel *Nouveau > Document texte* 15](#_Toc427918729)

[Figure 12 – Menu contextuel *TortoiseGit > Add…* 15](#_Toc427918730)

[Figure 13 – Liste des fichiers à ajouter à l’index 16](#_Toc427918731)

[Figure 14 - Fenêtre de fin de l'ajout à l'index 16](#_Toc427918732)

[Figure 15 - Résultat de l'ajout du fichier Fichier1.txt à l'index 16](#_Toc427918733)

[Figure 16- Saisie du message de *Commit* 17](#_Toc427918734)

[Figure 17 - Résultat de l'ajout du fichier Fichier2.txt à l'index 17](#_Toc427918735)

[Figure 18- Menu contextuel *Git Commit -> « develop » …* 18](#_Toc427918736)

[Figure 19- Ajout d'un message de C*ommit* et C*ommit* du fichier Fichier2.txt 18](#_Toc427918737)

[Figure 20 - Fenêtre de fin du *Commit* 19](#_Toc427918738)

[Figure 21 - Menu contextuel *TortoiseGit >* *Show log* 20](#_Toc427918739)

[Figure 22 - Historique 21](#_Toc427918740)

[Figure 23 - Sélection de l'action *Revert change by this commit* depuis l'arbre de révision 21](#_Toc427918741)

[Figure 24 - Confirmation de l'opération *Revert change by this commit* 22](#_Toc427918742)

[Figure 25 - Modification du message par défaut de l'opération *Revert change by this commit* 22](#_Toc427918743)

[Figure 26 - Fenêtre de fin du *Revert commit by this change* 23](#_Toc427918744)

[Figure 27- Nouveau *Commit* suite au *Revert change by this commit* 23](#_Toc427918745)

[Figure 28 - Menu contextuel *TortoiseGit > Create Branch...* 24](#_Toc427918746)

[Figure 29 - Création d'une branche 25](#_Toc427918747)

[Figure 30 - Fenêtre de fin de création d'une branche 25](#_Toc427918748)

[Figure 31 – Espace de travail positionné sur la nouvelle branche 25](#_Toc427918749)

[Figure 32 - Sélection de l'historique depuis la branche de fonctionnalité 26](#_Toc427918750)

[Figure 33 - Historique depuis la branche de fonctionnalité 26](#_Toc427918751)

[Figure 34 - Menu contextuel *TortoiseGit > Switch/Checkout...* 27](#_Toc427918752)

[Figure 35 - Choix de la branche *develop* 27](#_Toc427918753)

[Figure 36 - Fenêtre de fin du *Switch/Checkout...* 27](#_Toc427918754)

[Figure 37- Sélection de l'historique depuis la branche *develop* 28](#_Toc427918755)

[Figure 38- Historique depuis la branche *develop* 28](#_Toc427918756)

[Figure 39 - Arbre de révision montrant les branches *develop* et de fonctionnalité 29](#_Toc427918757)

[Figure 40 - Espace de travail positionné sur la branche *develop* 30](#_Toc427918758)

[Figure 41 - Fichiers présents sur la branche *develop* 31](#_Toc427918759)

[Figure 42 – Historique des révisions sur la branche *develop* 31](#_Toc427918760)

[Figure 43 - Menu contextuel *TortoiseGit > Switch/Checkout* pour se positionner sur la branche de fonctionnalité 32](#_Toc427918761)

[Figure 44 - Choix de la branche de fonctionnalité 32](#_Toc427918762)

[Figure 45 - Fenêtre de fin du *Switch/Checkout...* 32](#_Toc427918763)

[Figure 46- Fichiers présents sur la branche de fonctionnalité 33](#_Toc427918764)

[Figure 47- Historique des révisions sur la branche de fonctionnalité 33](#_Toc427918765)

[Figure 48 - Menu contextuel *TortoiseGit >* *Rebase* 33](#_Toc427918766)

[Figure 49 - Choix de la branche servant à la mise à jour de la branche de fonctionnalité par l'opération *Rebase* 34](#_Toc427918767)

[Figure 50- Fenêtre de fin du *Rebase* 35](#_Toc427918768)

[Figure 51- Arbre de révision sur la branche de fonctionnalité après le *Rebase* 36](#_Toc427918769)

[Figure 52- Arbre de révision sur la branche *develop* après le *Rebase* 37](#_Toc427918770)

[Figure 53- Etat des 2 branches avant et après le *Rebase* 37](#_Toc427918771)

[Figure 54- Espace de travail positionné sur la branche de fonctionnalité 38](#_Toc427918772)

[Figure 55 - Menu contextuel *TortoiseGit > Merge* depuis la branche *develop* 39](#_Toc427918773)

[Figure 56 - Choix de la branche à fusionner lors du *Merge* et message associé 39](#_Toc427918774)

[Figure 57 - Résultat du *Merge* 40](#_Toc427918775)

[Figure 58 - Arbre de révision de la branche de fonctionnalité après le *Merge* 41](#_Toc427918776)

[Figure 59 - Arbre de révision de la branche develop après le *Merge* 42](#_Toc427918777)

[Figure 60 - Etat des 2 branches avant et après le *Merge* 42](#_Toc427918778)

[Figure 61- Arbre de révision de la branche de fonctionnalité feature/FonctionnaliteB 43](#_Toc427918779)

[Figure 62- Arbre de révision de la branche *develop* 44](#_Toc427918780)

[Figure 63 - Fenêtre de résultat d'un *Merge* avec conflit 45](#_Toc427918781)

[Figure 64 - Fichier en conflit 45](#_Toc427918782)

[Figure 65 - Menu contextuel *TortoiseGit > Edit conflicts* 45](#_Toc427918783)

[Figure 66 - Mise en exergue du conflit en positionnant côte à côte les contributions 46](#_Toc427918784)

[Figure 67 - Résultat obtenu après sélection de la contribution souhaitée pour résoudre le conflit 47](#_Toc427918785)

[Figure 68 - Fenêtre indiquant que le fichier peut être marqué comme résolu 47](#_Toc427918786)

[Figure 69 – *Commit* du fichier sur lequel le conflit a été résolu 48](#_Toc427918787)

[Figure 70 - Fin du *Merge* après résolution des conflits par commit du fichier modifié 49](#_Toc427918788)

[Figure 71- Arbre de révision après le *Merge* conflictuel 49](#_Toc427918789)

[Figure 72 - Repérage du commit à reproduire dans une autre branche 50](#_Toc427918790)

[Figure 73 - Case à cocher *All Branches* cochée dans l’arbre de révision 51](#_Toc427918791)

[Figure 74 - Menu contextuel *Cherry Pick this commit …* 51](#_Toc427918792)

[Figure 75 - Fenêtre de confirmation associée au menu contextuel *Cherry Pick this commit…* 52](#_Toc427918793)

[Figure 76 - Fenêtre de fin du *Cherry Pick this commit...* 52](#_Toc427918794)

[Figure 77- Arbre de révision après le *Cherry Pick this commit…* 53](#_Toc427918795)

[Figure 78- Nouvelle branche de version 55](#_Toc427918796)

[Figure 79- Différentes branches créées 55](#_Toc427918797)

[Figure 80- Menu contextuel *TortoiseGit >* *Create Tag…* depuis l'objet *Tags* 56](#_Toc427918798)

[Figure 81- Création d'une étiquette de version 56](#_Toc427918799)

[Figure 82- Arbre de révision sur la branche de version après la pose de l'étiquette 57](#_Toc427918800)

[Figure 83- Arbre de révision sur la branche *develop* avant le *Merge* 58](#_Toc427918801)

[Figure 84- Arbre de révision sur la branche de version R1 avant le *Merge* 59](#_Toc427918802)

[Figure 85- Arbre de révision sur la branche *develop* après le *Merge* 60](#_Toc427918803)

[Figure 86- Arbre de révision sur la branche de version R1 après le *Merge* 60](#_Toc427918804)

[Figure 87- Etat des 3 branches avant et après le *Merge* 61](#_Toc427918805)

[Figure 88 - Fichiers disponibles sur la branche *develop* d'un dépôt fraichement cloné 62](#_Toc427918806)

[Figure 89 - Menu contextuel *TortoiseGit >* *Push...* 62](#_Toc427918807)

[Figure 90 - Informations sur la branche et le dépôt distant sur lequel seront remontées les contributions 63](#_Toc427918808)

[Figure 91 - Information de login sur le dépôt distant - matricule SESAME 63](#_Toc427918809)

[Figure 92- Information de login sur le dépôt distant – mot de passe SESAME 64](#_Toc427918810)

[Figure 93 - Résultat du *Push* 64](#_Toc427918811)

[Figure 94 - Menu contextuel *TortoiseGit >* *Fetch...* 64](#_Toc427918812)

[Figure 95 - Choix de la branche distante 65](#_Toc427918813)

[Figure 96- Information de login sur le dépôt distant - matricule SESAME 65](#_Toc427918814)

[Figure 97- Information de login sur le dépôt distant – mot de passe SESAME 65](#_Toc427918815)

[Figure 98 - Fin du *Fetch* 66](#_Toc427918816)

[Figure 99 - Arbre de révision après le *Fetch* 66](#_Toc427918817)

[Figure 100 - Vérification que la branche *develop* est la branche courante 67](#_Toc427918818)

[Figure 101- Choix de la branche remotes/*origin/develop* lors d’un *Merge* 67](#_Toc427918819)

[Figure 102- Fenêtre de résultat d’un *Fast-forward* lors d'un *Merge* 68](#_Toc427918820)

[Figure 103- Arbre de révision après le *Merge* déclenché par l’utilisateur, suite à un Fetch 68](#_Toc427918821)

[Figure 104 - Nouveau clone 69](#_Toc427918822)

[Figure 105- Fichiers présents dans le dépôt local après le clone 69](#_Toc427918823)

[Figure 106- Arbre de révision du troisième dépôt local après le clone 70](#_Toc427918824)

[Figure 107- Menu contextuel *TortoiseGit > Pull…* 71](#_Toc427918825)

[Figure 108 - Choix de la branche distante lors d'un *Pul* 71](#_Toc427918826)

[Figure 109 - Information de login sur le dépôt distant - matricule SESAME 71](#_Toc427918827)

[Figure 110 - Information de login sur le dépôt distant - mot de passe SESAME 71](#_Toc427918828)

[Figure 111- Résultat du *Pull* (*Fetch* + *Merge*) 72](#_Toc427918829)

[Figure 112- Arbre de révision après le *Pull* 72](#_Toc427918830)

# Introduction

Ce document a pour but de présenter les exercices réalisés lors de la formation Git pour développeur.

Dans ce document, Git est toujours utilisé depuis Windows. Il ne s’agit donc pas ici d’utiliser Git en lignes de commande ou avec d’autres clients.

Ce document peut être utilisé lors de la formation mais également de façon individuelle par toute personne souhaitant avoir une première approche Git. Il est suffisamment illustré pour pouvoir appréhender seul les principaux concepts et surtout pratiquer.

**Pour la formation, se connecter sur le poste avec son propre matricule (a ou x) et non celui de la salle de formation**.

# Exercice 1 – Configurer Git avec TortoiseGit

But de l’exercice : Configurer Git avec TortoiseGit

1. Créer un répertoire dédié aux exercices sous D:\Git en fonction de la position du poste dans la salle (par exemple FormationGit01TortoiseGit),
2. Se positionner sur ce dossier et sélectionner le menu contextuel *TortoiseGit > Settings* (Figure1)

Figure - Menu contextuel *TortoiseGit > Settings*

## Configurer le chemin vers le dossier bin de l’installation Git

Il convient de vérifier que le champ *Git.exe Path* contient bien le chemin vers le dossier bin de l’installation de Git.

1. Se positionner sur l’onglet *General*.
2. Vérifier que le champ *Git exe Path* contient la valeur *C:\Program Files (x86)\Freeware\Git\1.9.5\bin.*  (Figure2)

Figure - Configuration du chemin vers le dossier bin de l’installation Git

## Configurer l’utilisateur

Il convient de renseigner le nom et l’adresse mail de l’utilisateur.

1. Se positionner sur l’onglet Git.
2. Pour pouvoir modifier les informations, sélectionner le radio-bouton *Global* (Figure 4)
3. Accepter les termes de la fenêtre qui s’ouvre (Figure 3)

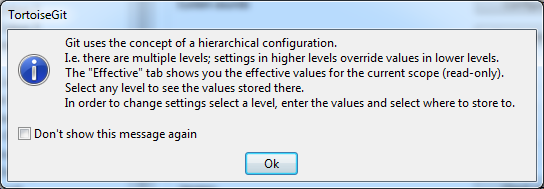


Figure - Fenêtre après modification du radio-bouton *Global*

1. Renseigner les champs *Name* et *Email* (Figure 4)

Le résultat obtenu est le suivant :

Figure - Configuration Utilisateur

**Dans le cadre de la formation**, le name sera renseigné en fonction du poste de formation :

Poste1 -> git01 prénom nom Poste2 -> git 02 prénom nom Poste3 -> git03 prénom nom

Poste4 -> git04 prénom nom Poste5 -> git05 prénom nom Poste6 -> git06 prénom nom

Poste7 -> git07 prénom nom Poste8 -> git08 prénom nom Poste9 -> git09 prénom nom

Poste10 -> git10 prénom nom

# Exercice 2 – Cloner un dépôt distant

But de l’exercice : Cloner un dépôt distant existant avec TortoiseGit et se positionner directement sur la branche *develop*

## Cloner le dépôt Git

1. Sélectionner le menu contextuel *Git Clone …* depuis le répertoire précédemment créé dans lequel mis sera le dépôt local (Figure 5)

Figure - Choix du menu contextuel *Git Clone …*

1. Renseigner les paramètres (Figure 6)

* URI : fournie par le formateur,
* Directory : dossier dans lequel sera cloné le dépôt distant
* Cocher la case *Branch* et indiquer qu’il s’agira d’une branche *develop*

Figure – URL du dépôt distant

Liste des URI à utiliser en fonction du numéro du poste de formation :

<https://dgitd.dns21.socgen/git/Formations/FormationGit01> <https://dgitd.dns21.socgen/git/Formations/FormationGit02>

<https://dgitd.dns21.socgen/git/Formations/FormationGit03> <https://dgitd.dns21.socgen/git/Formations/FormationGit04>

<https://dgitd.dns21.socgen/git/Formations/FormationGit05> <https://dgitd.dns21.socgen/git/Formations/FormationGit06>

<https://dgitd.dns21.socgen/git/Formations/FormationGit07> <https://dgitd.dns21.socgen/git/Formations/FormationGit08>

<https://dgitd.dns21.socgen/git/Formations/FormationGit09> <https://dgitd.dns21.socgen/git/Formations/FormationGit10>

1. Renseigner le user (Figure7)

* User : nom de l’utilisateur SESAME,

1. Renseigner le mot de passe (Figure 8)

* Password : mot de passe SESAME.

**Dans le cadre de la formation**, liste des users et mots de passe à utiliser en fonction du numéro du poste de formation :

Poste1 -> User : git01 Mot de passe : git01 Poste2 -> User : git02 Mot de passe : git02

Poste3 -> User : git03 Mot de passe : git03 Poste4 -> User : git04 Mot de passe : git04

Poste5 -> User : git05 Mot de passe : git05 Poste6 -> User : git06 Mot de passe : git06

Poste7 -> User : git07 Mot de passe : git07 Poste8 -> User : git08 Mot de passe : git08

Poste9 -> User : git09 Mot de passe : git09 Poste10 -> User : git10 Mot de passe : git10

Figure - User pour se connecter au dépôt distant

Figure - Mot de passe pour se connecter au dépôt distant

En fin d’opération, la fenêtre suivante s’affiche :

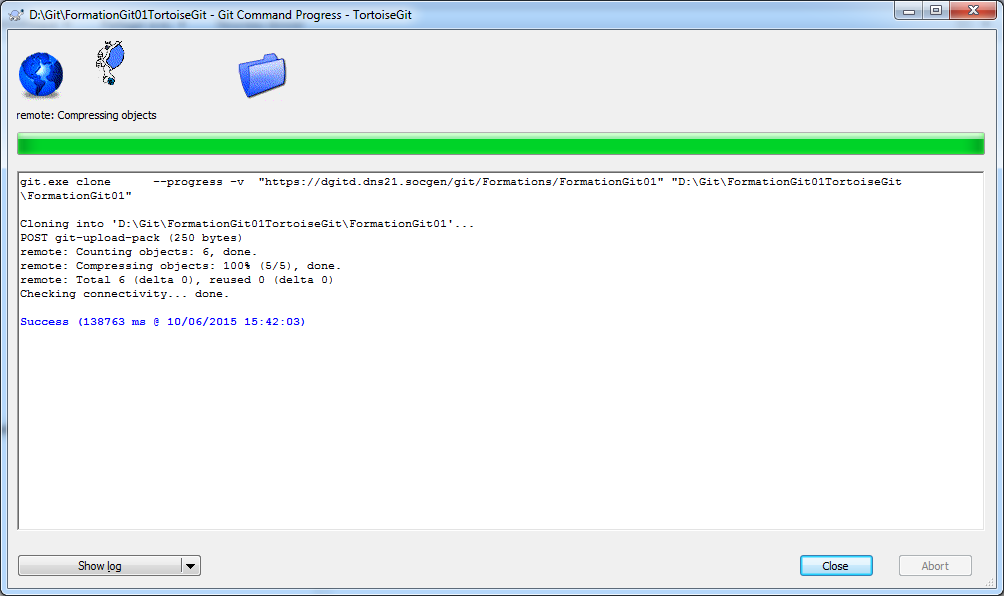


Figure - Fin de l'opération Clone d'un dépôt distant

Le résultat obtenu est le suivant :

Sous le répertoire précédemment créé, un répertoire du nom du dépôt est créé. Il contient au moins les fichiers suivants.

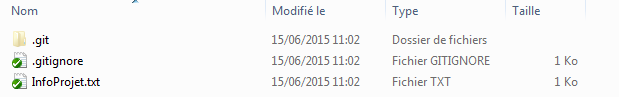


Figure - Résultat Clone du dépôt distant

## Installer les hooks dans le dépôt local

1. Recopier le fichier S:\gcl\_sgcorp\Git\hooks\commit-msg dans le répertoire .git\hooks du dépôt local.

# Exercice 3 – Enregistrer les changements : *Commit*

But de l’exercice : Ajouter deux fichiers via Windows dans le dépôt local sur la branche *develop*

Etat de la branche *develop* au début et à la fin de l’exercice :

Début Fin

## Créer un fichier Fichier1.txt dans le répertoire du dépôt local

1. Se positionner dans le répertoire représentant le dépôt local
2. Créer un fichier Fichier1.txt à partir du menu contextuel *Nouveau > Document texte* (Figure 11)



Figure - Menu contextuel *Nouveau > Document texte*

## Consigner ce nouveau fichier dans le dépôt local Git

1. Ajouter le fichier à l’index grâce au menu contextuel *TortoiseGit > Add…* (Figure 12)

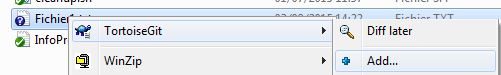


Figure – Menu contextuel *TortoiseGit > Add…*

1. Finaliser l’ajout à l’index du fichier, en cliquant sur le bouton *OK* (Figure 13)

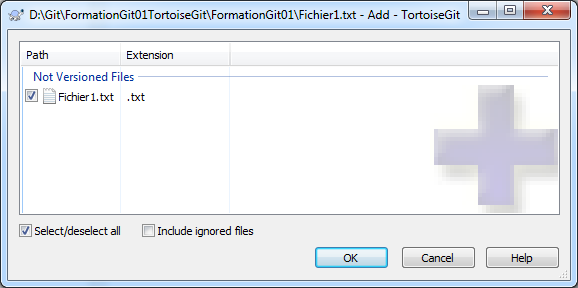


Figure – Liste des fichiers à ajouter à l’index

1. L’ajout est terminé lorsque la fenêtre (Figure 14) s’affiche. Le résultat obtenu est présenté figure 15.

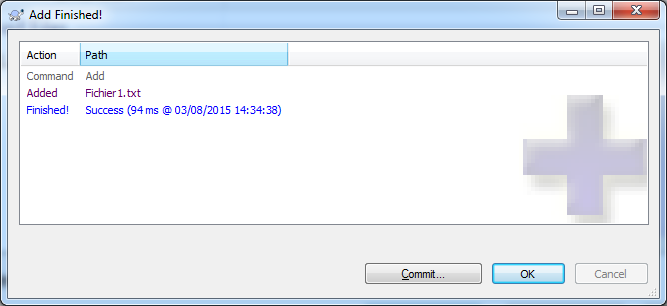


Figure - Fenêtre de fin de l'ajout à l'index

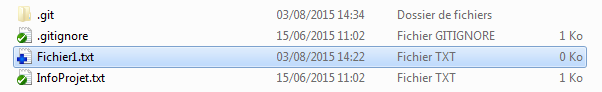


Figure - Résultat de l'ajout du fichier Fichier1.txt à l'index

1. Depuis, la fenêtre précédente (Figure 14), il est possible d’enchaîner directement avec la consignation des modifications en cliquant sur le bouton *Commit …* Dans la fenêtre qui s’affiche (Figure 16), saisir un message de *Commit* qui respecte les règles de nommage. Finaliser le *Commit* en cliquant sur le bouton *OK*.

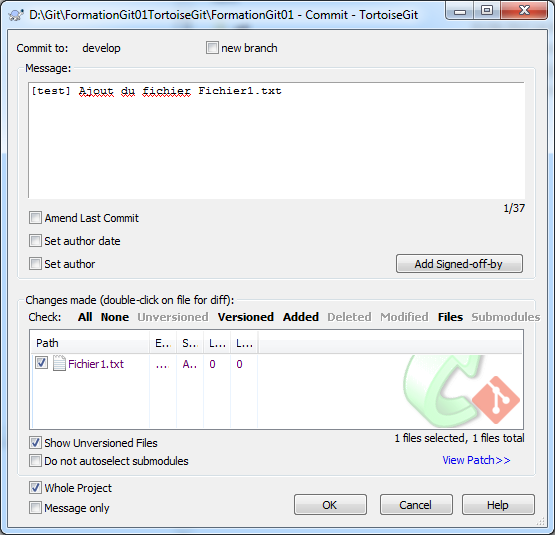


Figure - Saisie du message de *Commit*

## Recommencer avec un second fichier Fichier2.txt

1. Créer un nouveau fichier Fichier2.txt (Figure11)
2. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
3. Le résultat obtenu est le suivant (Figure 17)

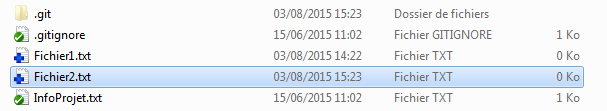


Figure - Résultat de l'ajout du fichier Fichier2.txt à l'index

1. Consigner le nouveau fichier dans le dépôt local Git (Figures 18, 19 et 20)

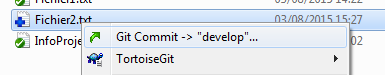


Figure - Menu contextuel *Git Commit -> « develop » …*

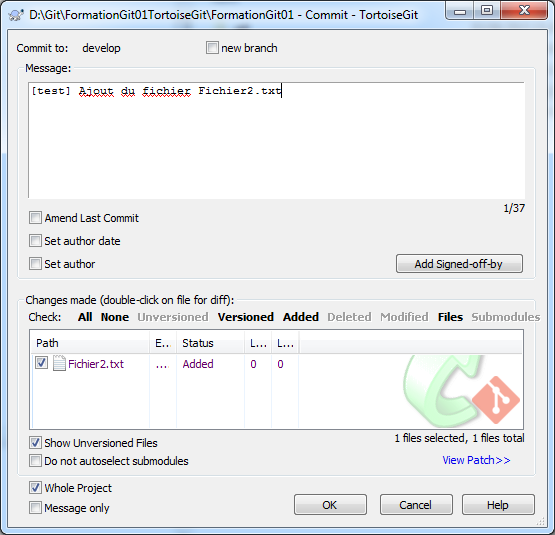
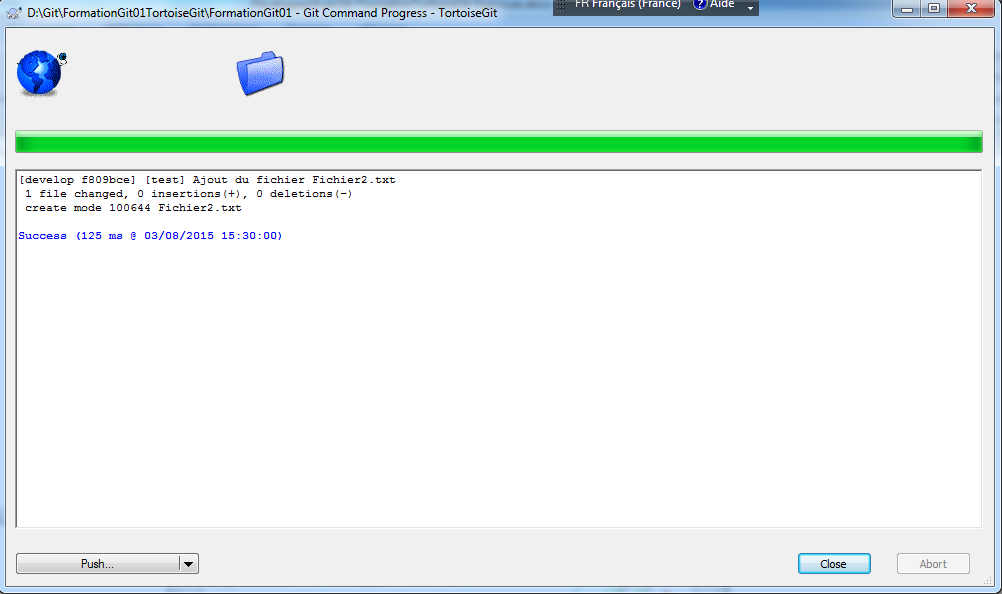


Figure - Ajout d'un message de C*ommit* et C*ommit* du fichier Fichier2.txt

Respecter les règles de nommage pour le message de commit. Les accents sont acceptés dans ces messages.

Le commit est terminé avec succès à l’affichage de la fenêtre suivante (Figure 20) :

Figure - Fenêtre de fin du *Commit*

# Exercice 4 – Annuler un changement : *Revert change by this commit*

But de l’exercice : Annuler un changement dans le dépôt local sur la branche *develop*

Etat de la branche *develop* au début et à la fin de l’exercice :

Début Fin

## Modifier un fichier et consigner ces modifications dans le dépôt Git

1. Modifier le fichier Fichier1.txt en rajoutant une ligne « Hello world ! »
2. Faire un *Commit* avec le message « [test] Commit à annuler par la suite » (Figures 18, 19 et 20)

## Annuler le changement

1. Visualiser l’historique (Figure 22) en utilisant le menu contextuel *TortoiseGit > Show log* (Figure 21) depuis le dossier parent.

Figure - Menu contextuel *TortoiseGit >* *Show log*

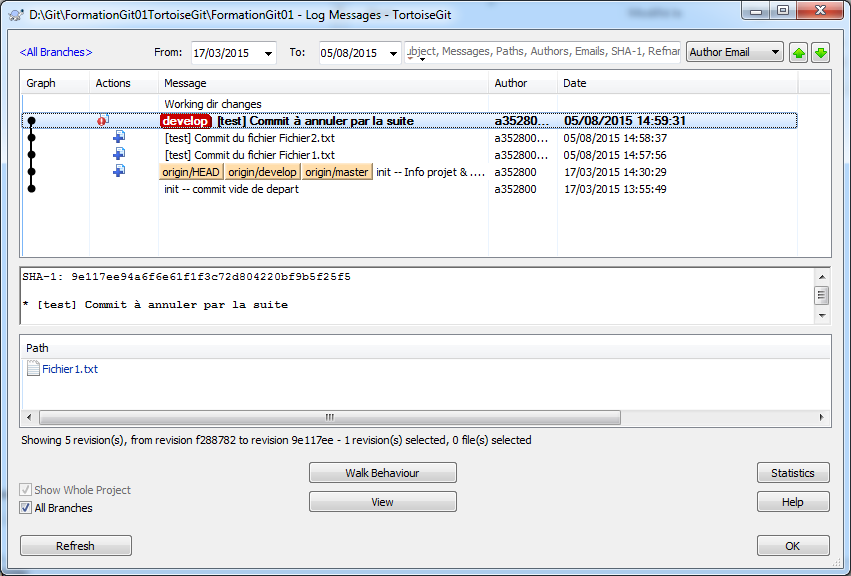


Figure - Historique

1. Sélectionner le C*ommit* « Commit à annuler par la suite » et faire un *Revert change by this commit* par le menu contextuel (Figure 23)

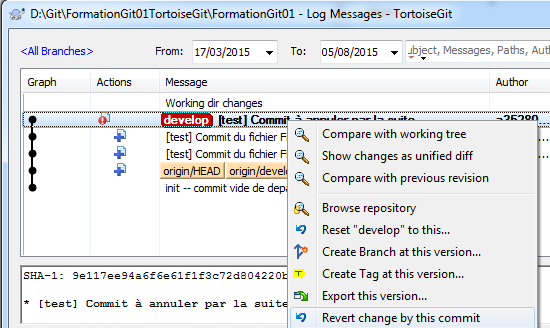


Figure - Sélection de l'action *Revert change by this commit* depuis l'arbre de révision

1. Cliquer sur le bouton Commit de la fenêtre de confirmation (Figure 24)

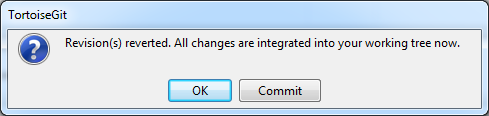


Figure - Confirmation de l'opération *Revert change by this commit*

1. Modifier éventuellement le message de commit par défaut afin qu’il respecte les règles de nommage et finaliser l’opération *Revert change by this commit* en cliquant sur le bouton *OK*. (Figure 25)

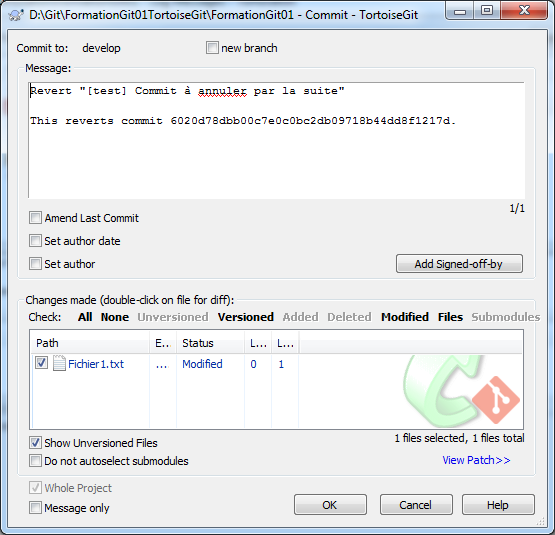


Figure - Modification du message par défaut de l'opération *Revert change by this commit*

L’opération *Revert change by this commit* est terminée avec succès à l’affichage de la fenêtre suivante (Figure 26) :

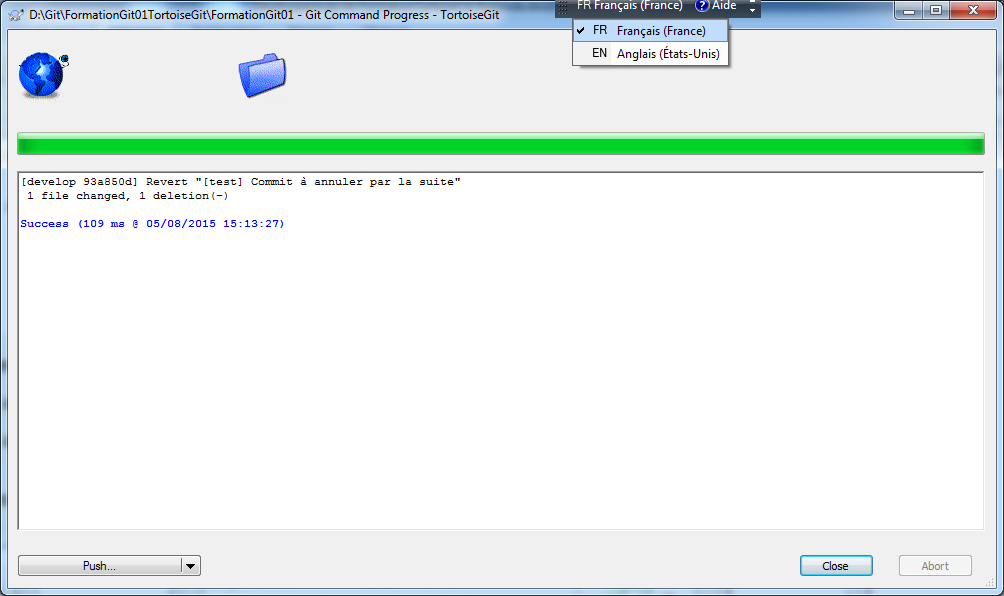


Figure - Fenêtre de fin du *Revert commit by this change*

1. Constater qu’un nouveau *Commit* est créé sur la branche *develop* (Figure 27)

## 

Figure - Nouveau *Commit* suite au *Revert change by this commit*

# Exercice 5 – Travailler sur une nouvelle branche de fonctionnalité

But de l’exercice : Ajouter une branche de fonctionnalité et enregistrer des modifications sur cette branche

Etat des branches *develop* et *feature* au début et à la fin de l’exercice :

Début Fin

## Créer une branche

1. Utiliser le menu contextuel *TortoiseGit > Create Branch…* (Figure 28) depuis le dossier parent.

Figure - Menu contextuel *TortoiseGit > Create Branch...*

1. Nommer cette branche de fonctionnalité en respectant les règles de nommage *feature/<nom de la fonctionnalité développée sous cette branche>* (Figure 29). Ne pas mettre d’espaces dans le nom de la fonctionnalité.
   * Base On : *HEAD (develop)* (option par défaut)
   * *Switch to new branch* : case cochée pour se positionner directement sur cette branche

Figure - Création d'une branche

L’opération *Create Branch* est terminée avec succès à l’affichage de la fenêtre suivante (Figure 30) :

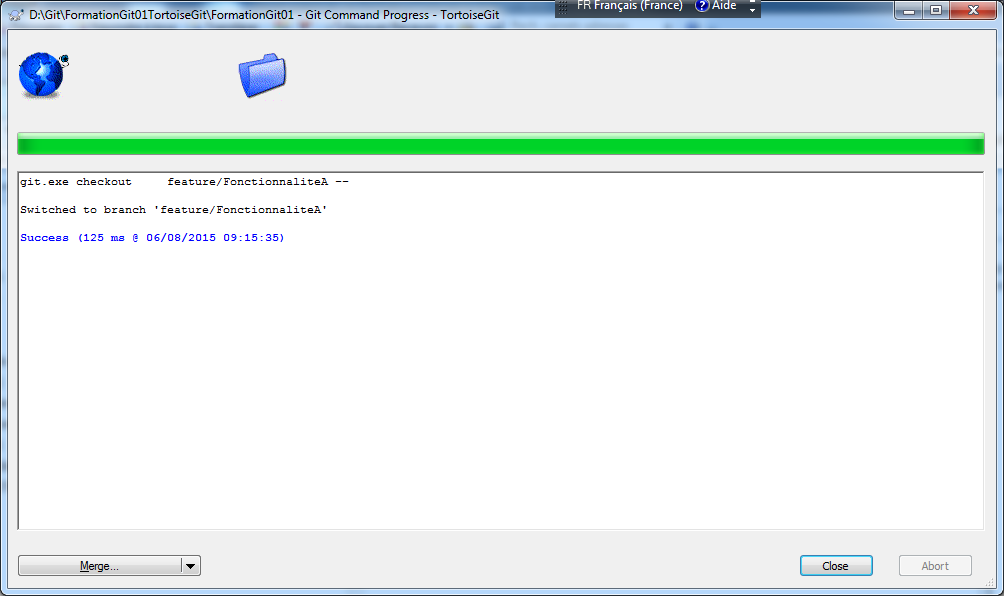


Figure - Fenêtre de fin de création d'une branche

1. Remarquer que le clic droit sur le dossier parent n’offre plus l’option *Git Commit -> « develop »…* mais *Git Commit -> « feature/FonctionnaliteA »…* indiquant que la branche courante est « feature/FonctionnaliteA ». (Figure 31)

Figure – Espace de travail positionné sur la nouvelle branche

## Créer un nouveau fichier FichierFA.txt sur cette branche et enregistrer les changements sur cette branche

1. Créer un nouveau fichier FichierFA.txt
2. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
3. Consigner le nouveau fichier dans le dépôt Git (Figures 18, 19 et 20)

## Parcourir l’historique des révisions

1. Visualiser l’historique (Figure 33) en utilisant le menu contextuel *TortoiseGit > Show log* (Figure 32) depuis le dossier parent. Vérifier au préalable que l’espace de travail est bien positionné sur la branche de fonctionnalité (affichage du menu contextuel *Git Commit -> « feature/FonctionnaliteA »…*)

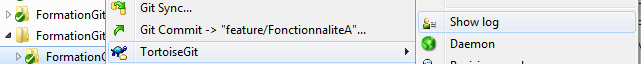


Figure - Sélection de l'historique depuis la branche de fonctionnalité



Figure - Historique depuis la branche de fonctionnalité

1. Se positionner sur la branche *develop* par le menu contextuel *TortoiseGit >* Switch/*Checkout…* (Figure 34). Sélectionner la branche *develop* dans la fenêtre suivante *(Figure 35)*

Figure - Menu contextuel *TortoiseGit > Switch/Checkout...*

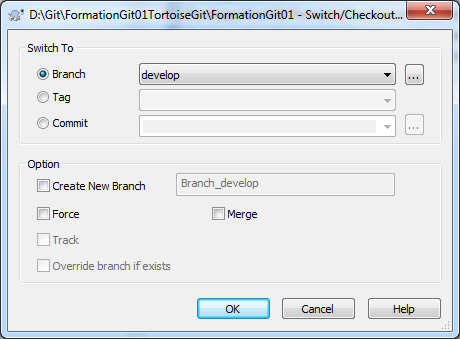


Figure - Choix de la branche *develop*

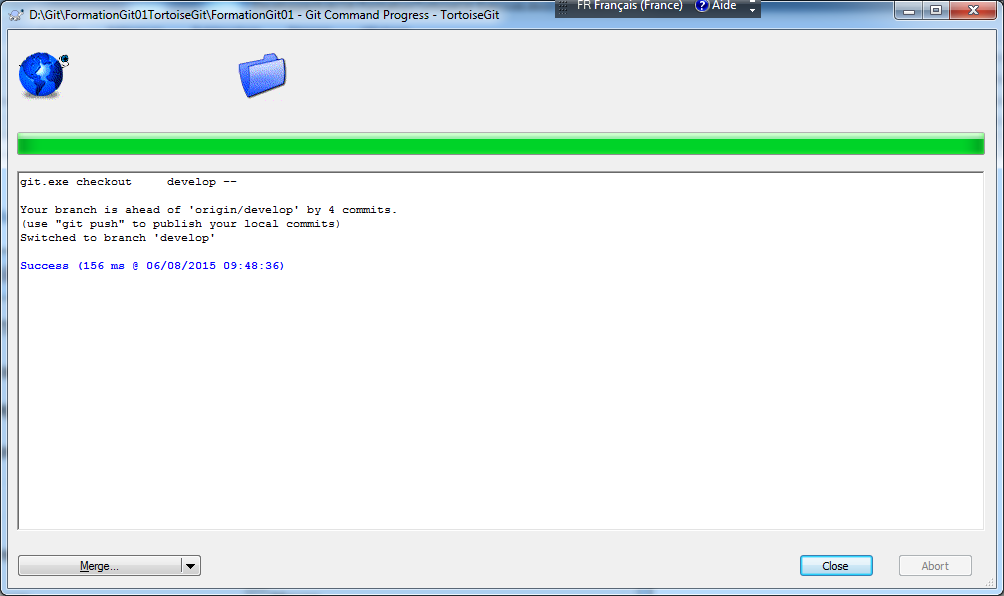


Figure - Fenêtre de fin du *Switch/Checkout...*

1. Visualiser l’historique depuis la branche *develop* (Figure 38) en utilisant le menu contextuel *TortoiseGit > Show log* (Figure 37) depuis le dossier parent.

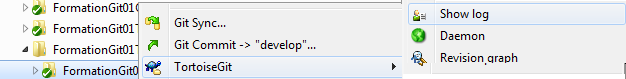


Figure - Sélection de l'historique depuis la branche *develop*

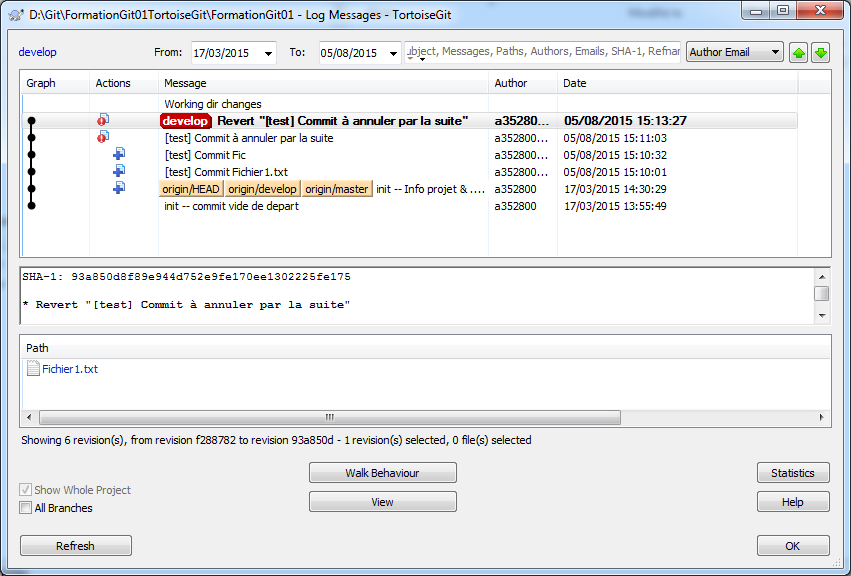


Figure - Historique depuis la branche *develop*

Il est possible de visualiser les 2 branches dans le même arbre de révision en cochant la case *All Branches* de la fenêtre qui présente l’historique (Figure 38)

Dans ce cas, l’arbre de révision des 2 branches est le suivant. Seul le **HEAD** change de branche pour indiquer la branche courante. (Figure 39)

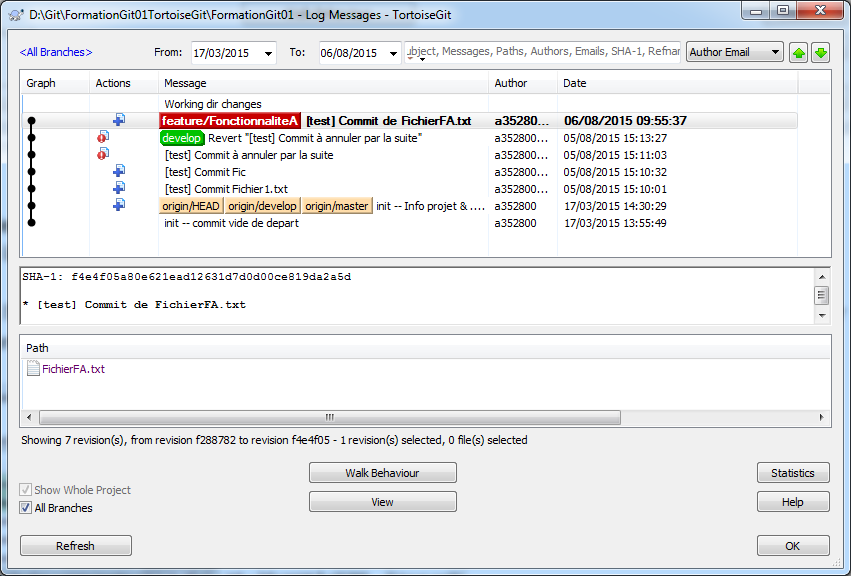


Figure - Arbre de révision montrant les branches *develop* et de fonctionnalité

Dans le reste de l’exercice, cette case sera décochée sauf indication contraire.

# Exercice 6 – Mettre à jour une branche par rapport à une autre avec un Rebase

But de l’exercice : Continuer de développer sur la branche *develop* puis mise à jour de la branche de fonctionnalité par un *Rebase*

Etat des branches *develop* et *feature* avant et après l’exercice :

Début Fin

## Changer de branche et se positionner sur la branche *develop*

1. Se positionner sur la branche *develop* par le menu contextuel *TortoiseGit > Switch/Checkout…* (Figure 34). Sélectionner la branche *develop* dans la fenêtre suivante (Figure 35) et visualiser le résultat (Figure 36)
2. Vérifier grâce au menu contextuel que la branche sur laquelle est positionné l’espace de travail est *develop* (Figure 40)



Figure - Espace de travail positionné sur la branche *develop*

## Ajouter un nouveau fichier Fichier3.txt et consigner les modifications sur cette branche

1. Créer un nouveau fichier Fichier3.txt
2. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
3. Consigner le nouveau fichier dans le dépôt Git (Figures 18, 19 et 20)

## Modifier le fichier Fichier3.txt et consigner les modifications

1. Modifier le fichier Fichier3.txt (ajouter Hello world 3 !)
2. Faire un *Commit* avec le message « Commit modification Fichier3.txt » (Figure 18, 19 et 20)
3. Sur la branche *develop*, 3 fichiers sont présents (Fichier1.txt, Fichier2.txt et Fichier3.txt) (Figure 41)

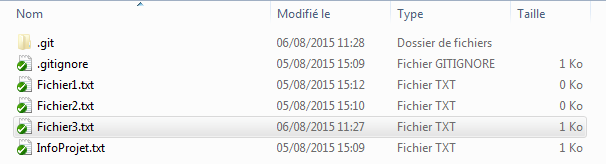


Figure - Fichiers présents sur la branche *develop*

## Visualiser l’historique des révisions sur cette branche *develop*

1. Visualiser l’historique depuis la branche *develop* (Figure 42) en utilisant le menu contextuel *TortoiseGit > Show log* (Figure 37) depuis le dossier parent.

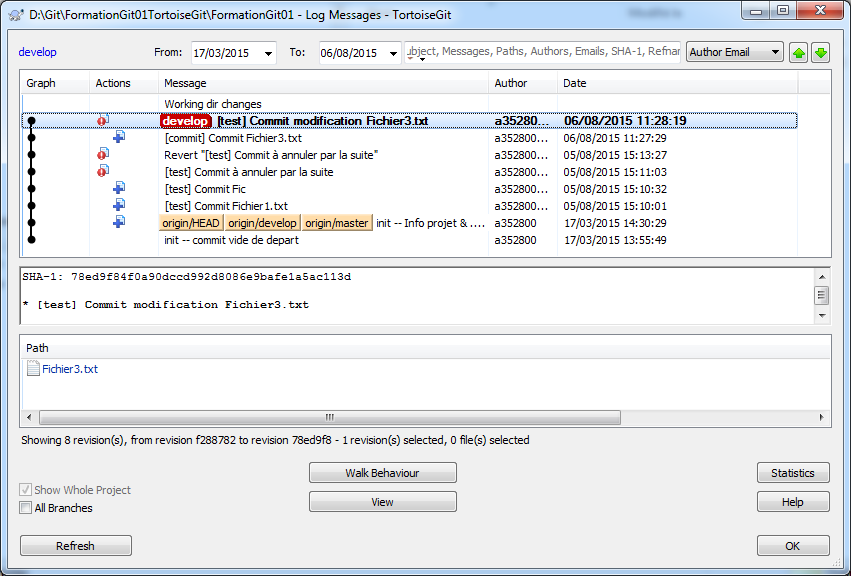


Figure – Historique des révisions sur la branche *develop*

## Visualiser les fichiers présents dans la branche de fonctionnalité et l’historique des révisions

1. Se positionner sur la branche de fonctionnalité par le menu contextuel *TortoiseGit > Switch/Checkout…* (Figure 43). Sélectionner la branche de fonctionnalité dans la fenêtre suivante (Figure 44) et visualiser le résultat (Figure 45)



Figure - Menu contextuel *TortoiseGit > Switch/Checkout* pour se positionner sur la branche de fonctionnalité

Figure - Choix de la branche de fonctionnalité

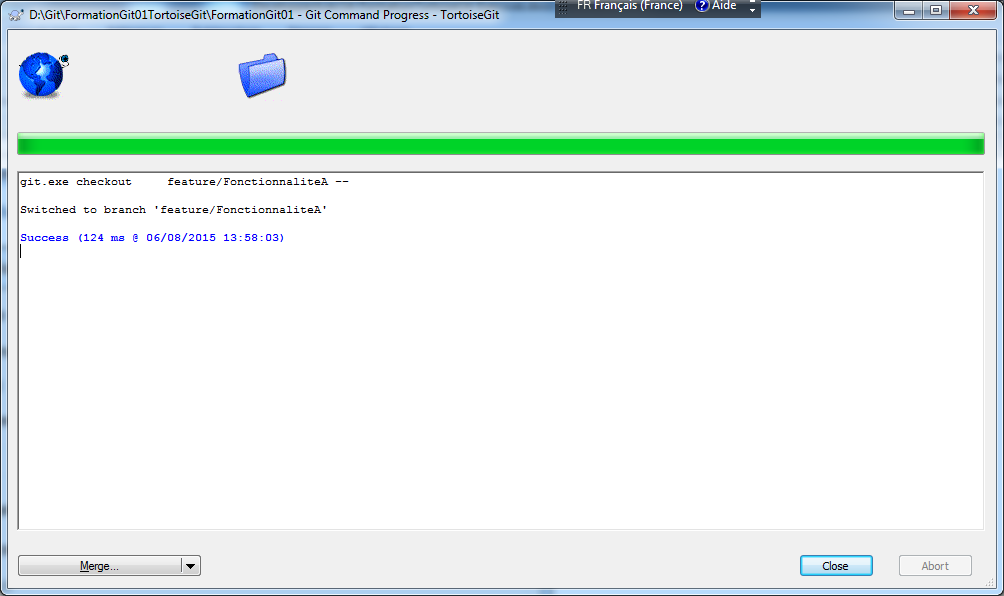


Figure - Fenêtre de fin du *Switch/Checkout...*

1. Sur la branche de fonctionnalité, 3 fichiers sont présents (Fichier1.txt, Fichier2.txt et FichierFA.txt) (Figure 42)

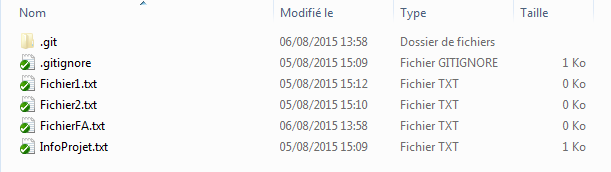


Figure - Fichiers présents sur la branche de fonctionnalité

1. Visualiser l’historique (Figure 47) en utilisant le menu contextuel *TortoiseGit > Show log* (Figure 32) depuis le dossier parent.



Figure - Historique des révisions sur la branche de fonctionnalité

## Mettre à jour la branche de fonctionnalité par rapport à la branche *develop* grâce à un Rebase

1. Se positionner sur la branche de fonctionnalité par le menu contextuel *TortoiseGit > Switch/Checkout …* (Figures 43, 44 et 45). Normalement, on y est déjà.
2. Sélectionner le menu contextuel *TortoiseGit >* *Rebase…* (Figure 48).

Figure - Menu contextuel *TortoiseGit >* *Rebase*

1. Choisir la branche à partir de laquelle la mise à jour doit être effectuée dans le menu déroulant *Upstream*. En l’occurrence, sélectionner la branche *develop*. Finaliser le *Rebase* en cliquant sur le bouton *Start Rebase*. (Figure 49)

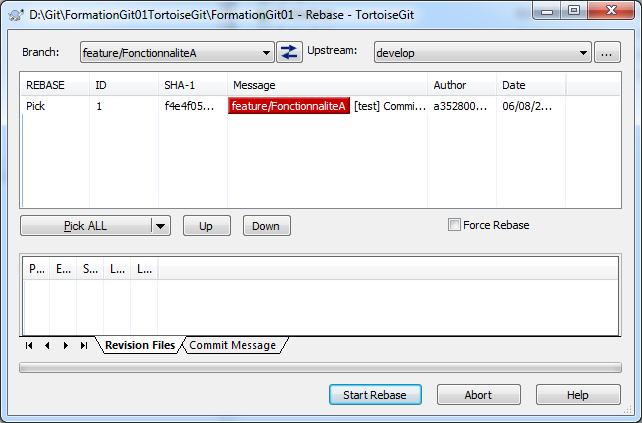


Figure - Choix de la branche servant à la mise à jour de la branche de fonctionnalité par l'opération *Rebase*

L’opération *Rebase* est terminée à l’affichage de la fenêtre suivante (Figure 50). Finaliser l’opération en cliquant sur le bouton *Done*.

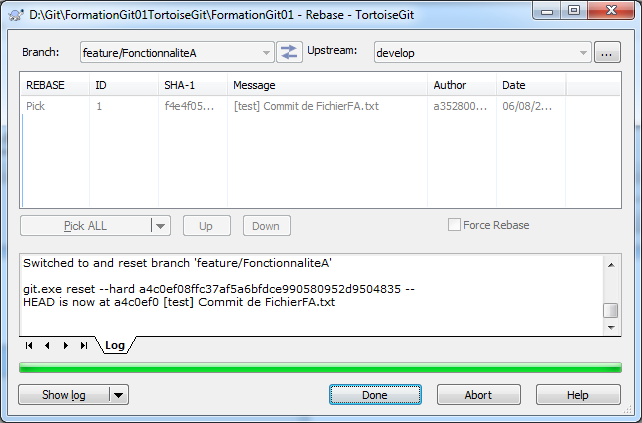


Figure - Fenêtre de fin du *Rebase*

## Parcourir l’arbre de révision sur la branche de fonctionnalité et sur la branche *develop*

1. Se positionner sur la branche de fonctionnalité par le menu contextuel *TortoiseGit > Switch/Checkout* (Figures 43, 44 et 45)
2. Visualiser l’arbre de révision sur la branche de fonctionnalité (Figure 51) grâce au menu contextuel *TortoiseGit > Show log* (Figure 32) depuis le dossier parent.

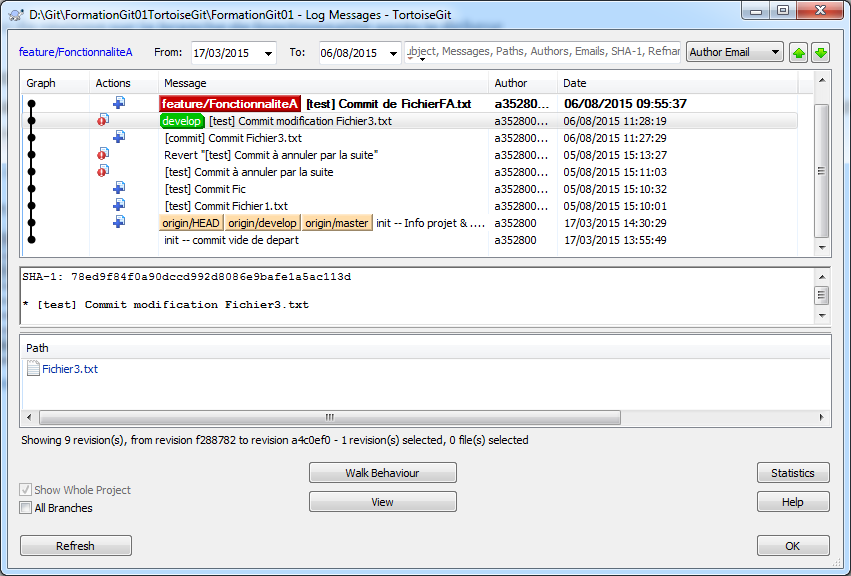


Figure - Arbre de révision sur la branche de fonctionnalité après le *Rebase*

1. Se positionner sur la branche *develop* par le menu contextuel *TortoiseGit > Switch/Checkout* (Figures 34, 35 et 36)
2. Visualiser l’arbre de révision sur la branche *develop* (Figure 52) grâce au menu contextuel *TortoiseGit > Show log* (Figure 37) depuis le dossier parent.

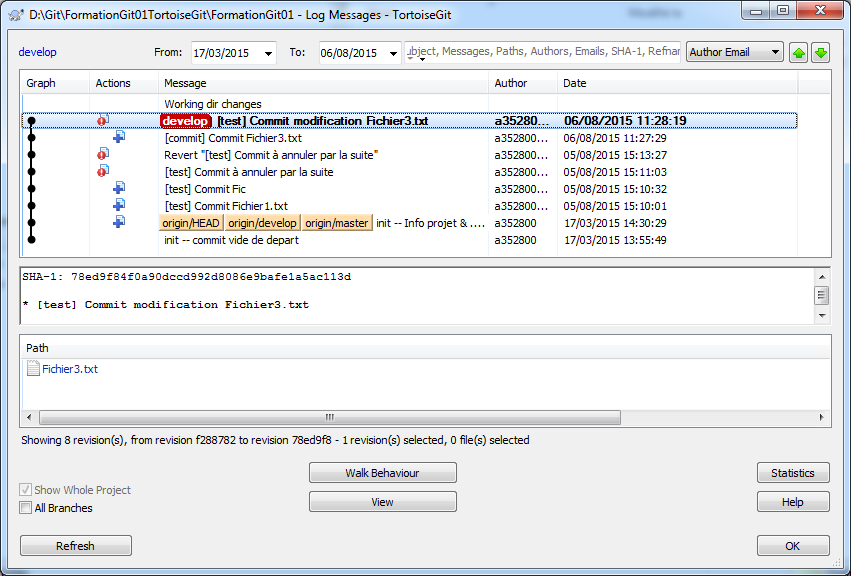


Figure - Arbre de révision sur la branche *develop* après le *Rebase*

## Schéma récapitulatif de l’état des 2 branches avant et après le *Rebase*

Avant le *Rebase* Après le *Rebase*

Figure - Etat des 2 branches avant et après le *Rebase*

# Exercice 7 – Mettre à jour une branche par rapport à une autre avec un Merge

But de l’exercice : Continuer de développer sur la branche de fonctionnalité puis mettre à jour la branche *develop* par un *Merge*

Etat des branches develop et feature au début et à la fin de l’exercice :

Début Fin

## Vérifier que l’espace de travail est positionné sur la branche de fonctionnalité

1. Vérifier que le menu contextuel depuis le dossier parent affiche *Git Commit -> « feature/FonctionnaliteA »…* (Figure 54)

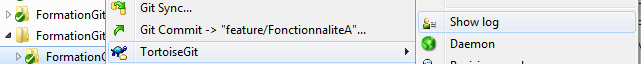


Figure - Espace de travail positionné sur la branche de fonctionnalité

1. Si ce n’est pas le cas, utiliser le menu contextuel *TortoiseGit > Switch/Checkout…* depuis le dossier parent (Figure 43). Sélectionner la branche de fonctionnalité dans la fenêtre suivante (Figure 44) et visualiser le résultat (Figure 45)

## Ajouter un nouveau fichier et consigner les modifications sur cette branche

1. Créer un nouveau fichier Fichier4.txt
2. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
3. Consigner le nouveau fichier dans le dépôt Git (Figures 18, 19 et 20)

## Ajouter un nouveau fichier sur la branche *develop* et consigner les modifications sur cette branche

1. Se positionner sur la branche *develop* par le menu contextuel *TortoiseGit > Switch/Checkout…* (Figure34). Sélectionner la branche *develop* dans la fenêtre suivante (Figure 35).
2. Créer un nouveau fichier Fichier41.txt
3. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
4. Consigner le nouveau fichier dans le dépôt Git (Figures 18, 19 et 20)

## Mettre à jour la branche *develop* par rapport à la branche de fonctionnalité grâce à un Merge

1. Se positionner, si ce n’est pas déjà le cas, sur la branche *develop* (Figures 34, 35 et 36)
2. Depuis le dossier parent, sélectionner le menu contextuel TortoiseGit > *Merge* (Figure 55).

Figure - Menu contextuel *TortoiseGit > Merge* depuis la branche *develop*

1. Dans *From*, sélectionner la branche qui va fusionner avec la branche courante à mettre à jour (Figure 56). Ne pas oublier de renseigner le message associé au *Merge*.

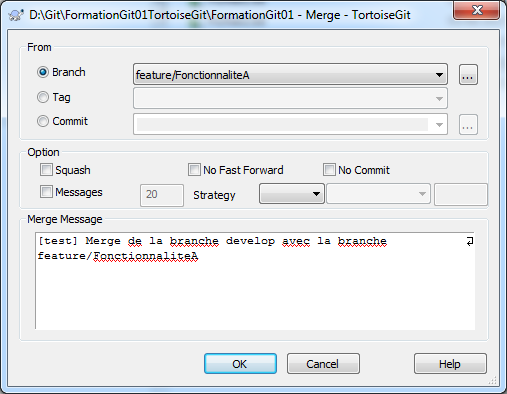


Figure - Choix de la branche à fusionner lors du *Merge* et message associé

1. Le résultat du *Merge* apparaît alors dans une fenêtre (Figure 57)

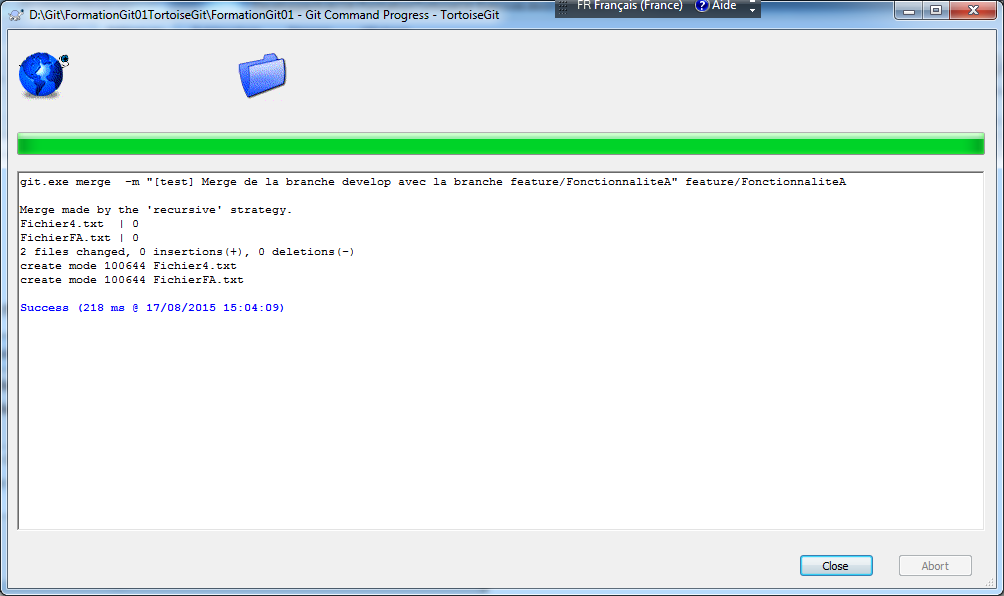


Figure - Résultat du *Merge*

## Parcourir l’arbre de révision sur la branche *develop* et sur la branche de fonctionnalité

1. Se positionner sur la branche de fonctionnalité depuis le dossier parent par le menu contextuel *TortoiseGit > Switch /* *Checkout …* (Figures 43, 44 et 45)
2. Visualiser l’arbre de révision sur la branche de fonctionnalité (Figure 58) grâce au menu contextuel *TortoiseGit > Show log* (Figure 32) depuis le dossier parent. Constater que rien n’a changé pour la branche de fonctionnalité feature/fonctionnaliteA

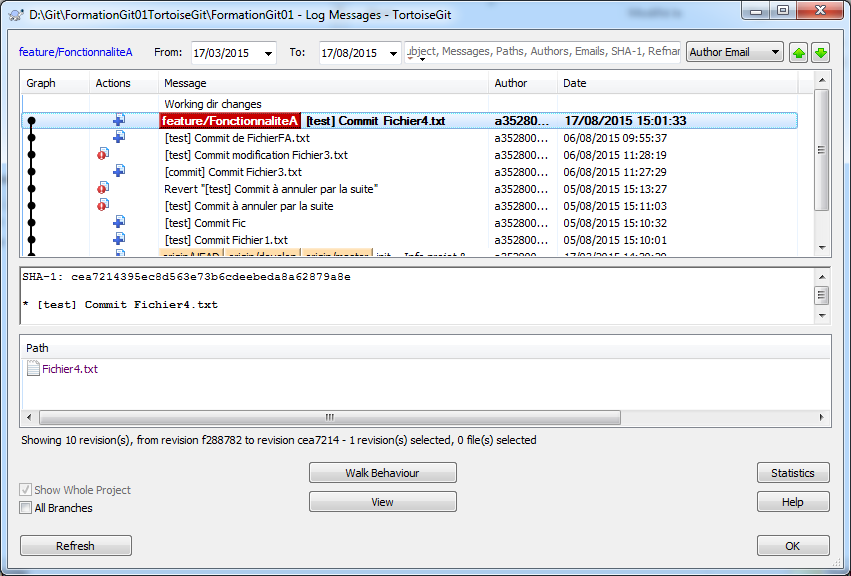


Figure - Arbre de révision de la branche de fonctionnalité après le *Merge*

1. Se positionner sur la branche *develop* par le menu contextuel *TortoiseGit > Switch / Checkout* … (Figures 34, 35 et 36) depuis le dossier parent.
2. Visualiser l’arbre de révision sur la branche *develop* (Figure 59) grâce au menu contextuel *TortoiseGit > Show log* (Figure 37) depuis le dossier parent.

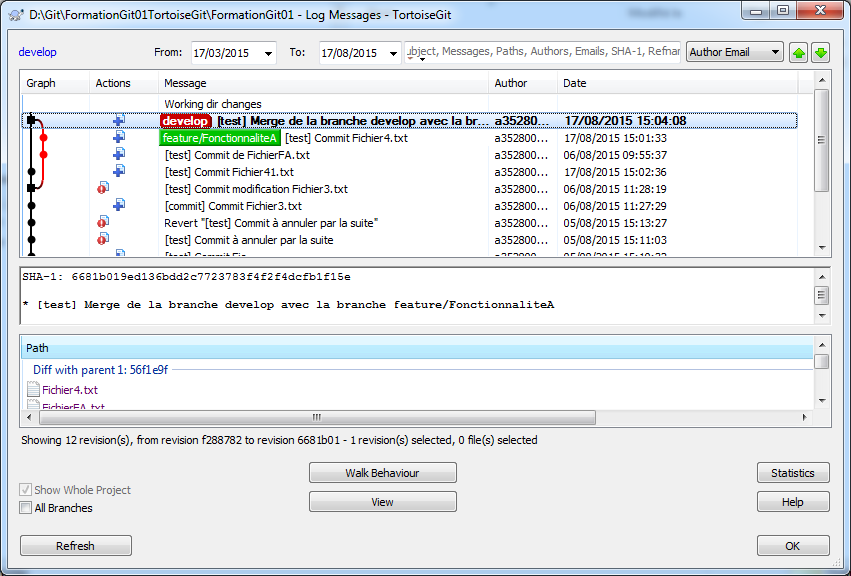


Figure - Arbre de révision de la branche develop après le *Merge*

## Schéma récapitulatif de l’état des 2 branches avant et après le *Merge*

Avant le *Merge* Après le *Merge*

Figure - Etat des 2 branches avant et après le *Merge*

# Exercice 8 – Gérer les conflits lors d’un *Merge*

But de l’exercice : Développer sur 2 branches dans un même fichier dans des zones de proximité. Fusionner les 2 branches par un *Merge*. Visualiser un conflit entre 2 Commit et le résoudre grâce à l’éditeur de conflits.

## Modifier le fichier Fichier2.txt depuis une nouvelle branche de fonctionnalité. Consigner les changements sur cette branche.

1. Créer une nouvelle branche de fonctionnalité nommée feature/FonctionnaliteB (Figures 28, 29 et 30), ayant pour source la branche *develop*
2. Dans le fichier Fichier2.txt, ajouter la ligne telle que définie ci-dessous.

Hello world 2 FonctionnaliteB

1. Consigner le fichier modifié dans le dépôt Git (Figures 18, 19 et 20)
2. Visualiser l’arbre de révision (Figure 61) grâce au menu contextuel *TortoiseGit >* *Show log* depuis le dossier parent (Figure 32)

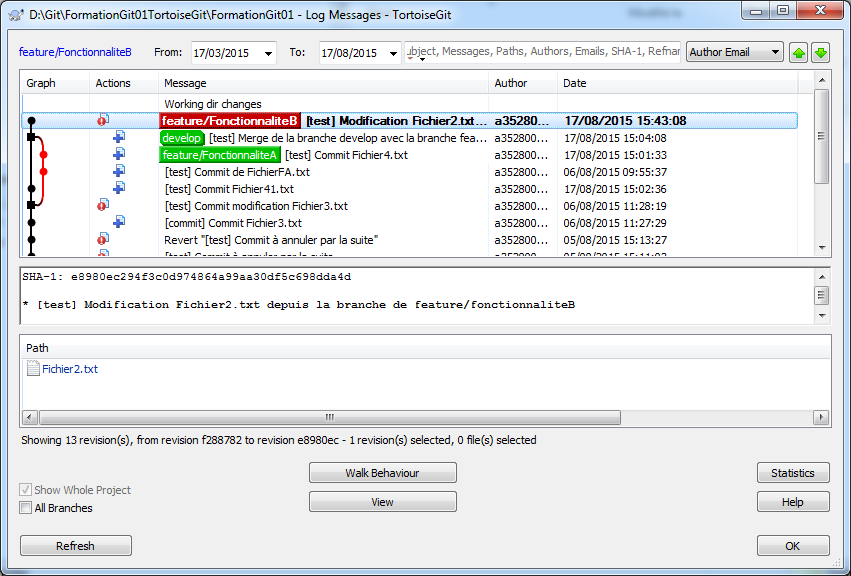


Figure - Arbre de révision de la branche de fonctionnalité feature/FonctionnaliteB

## Faire une autre modification dans le fichier Fichier2.txt depuis la branche *develop*

1. Se positionner sur la branche *develop* grâce au menu contextuel *TortoiseGit > Switch / CheckOut…* sur le dossier parent(Figures 34, 35 et 36)
2. Dans le fichier Fichier2.txt, ajouter la ligne telle que définie ci-dessous.

Hello world 2 develop

1. Consigner le fichier modifié dans le dépôt Git (Figures 18, 19 et 20)
2. Visualiser l’arbre de révision (Figure 62) grâce au menu contextuel *TortoiseGit >* *Show log* depuis le dossier parent (Figure 37)

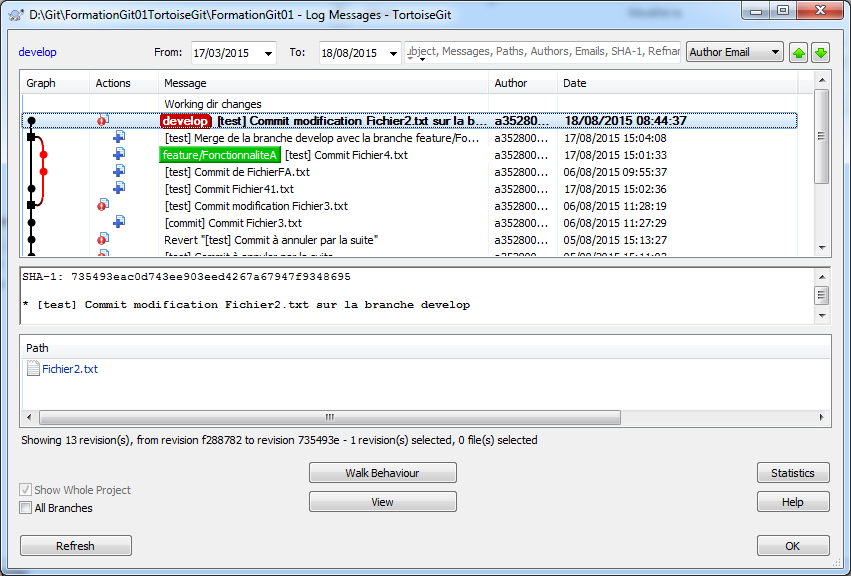


Figure - Arbre de révision de la branche *develop*

## Mettre à jour la branche *develop* par rapport à la branche de fonctionnalité grâce à un *Merge*

1. Vérifier que l’espace de travail est positionné sur la branche *develop* (Figure 40). Si ce n’est pas le cas, se positionner sur la branche *develop* (Figures 34, 35 ou 36)
2. Depuis le dossier parent, sélectionner le menu contextuel *TortoiseGit >* *Merge* (Figure 55).
3. Dans *From*, sélectionner la branche qui va fusionner avec la branche courante à mettre à jour, en l’occurrence feature/FonctionnaliteB (Figure 56). Ne pas oublier de renseigner le message associé au *Merge*.
4. La fenêtre de résultat fait apparaitre un conflit (Figure 63)

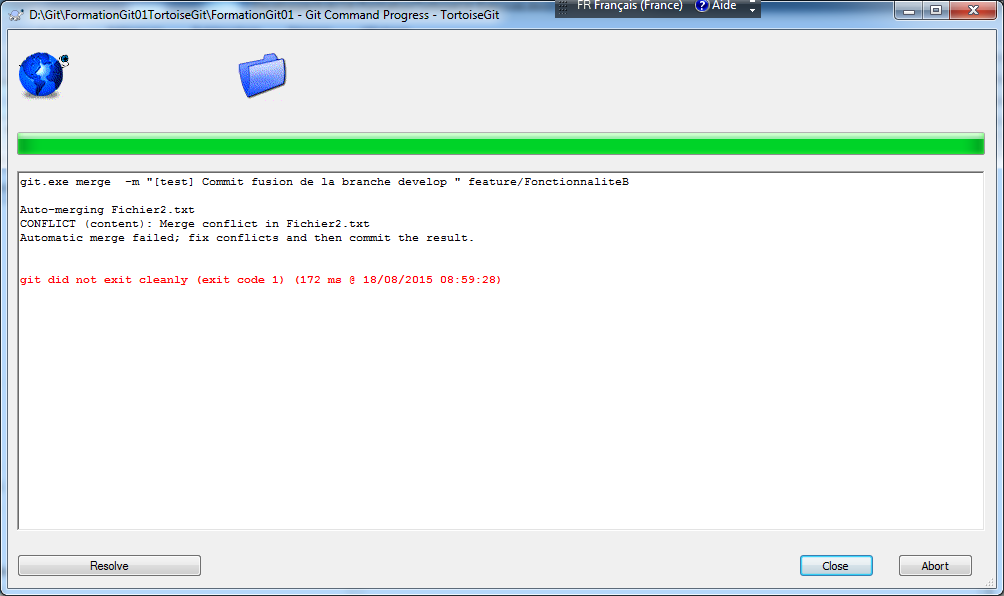


Figure - Fenêtre de résultat d'un *Merge* avec conflit

## Visualiser le conflit lors du *Merge*

1. Comme les 2 branches comportent des modifications divergentes sur des zones de proximité dans les fichiers de source, un conflit est identifié lors du *Merge* (Figure 63)
2. Le fichier en conflit est marqué d’une icône jaune avec un point d’exclamation (Figure 64)

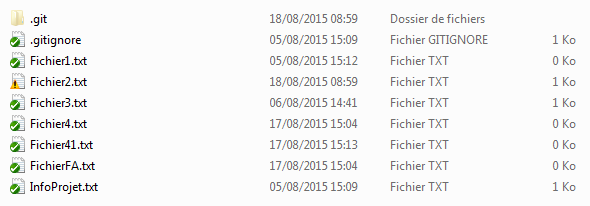


Figure - Fichier en conflit

## Résoudre le conflit avec l’éditeur de conflit

1. Sur le fichier à éditer, sélectionner le menu contextuel *TortoiseGit > Edit conflicts* (Figure 66)

Figure - Menu contextuel *TortoiseGit > Edit conflicts*

1. Visualiser le conflit (Figure 66) et utiliser le menu contextuel pour résoudre le conflit (Figure 67).

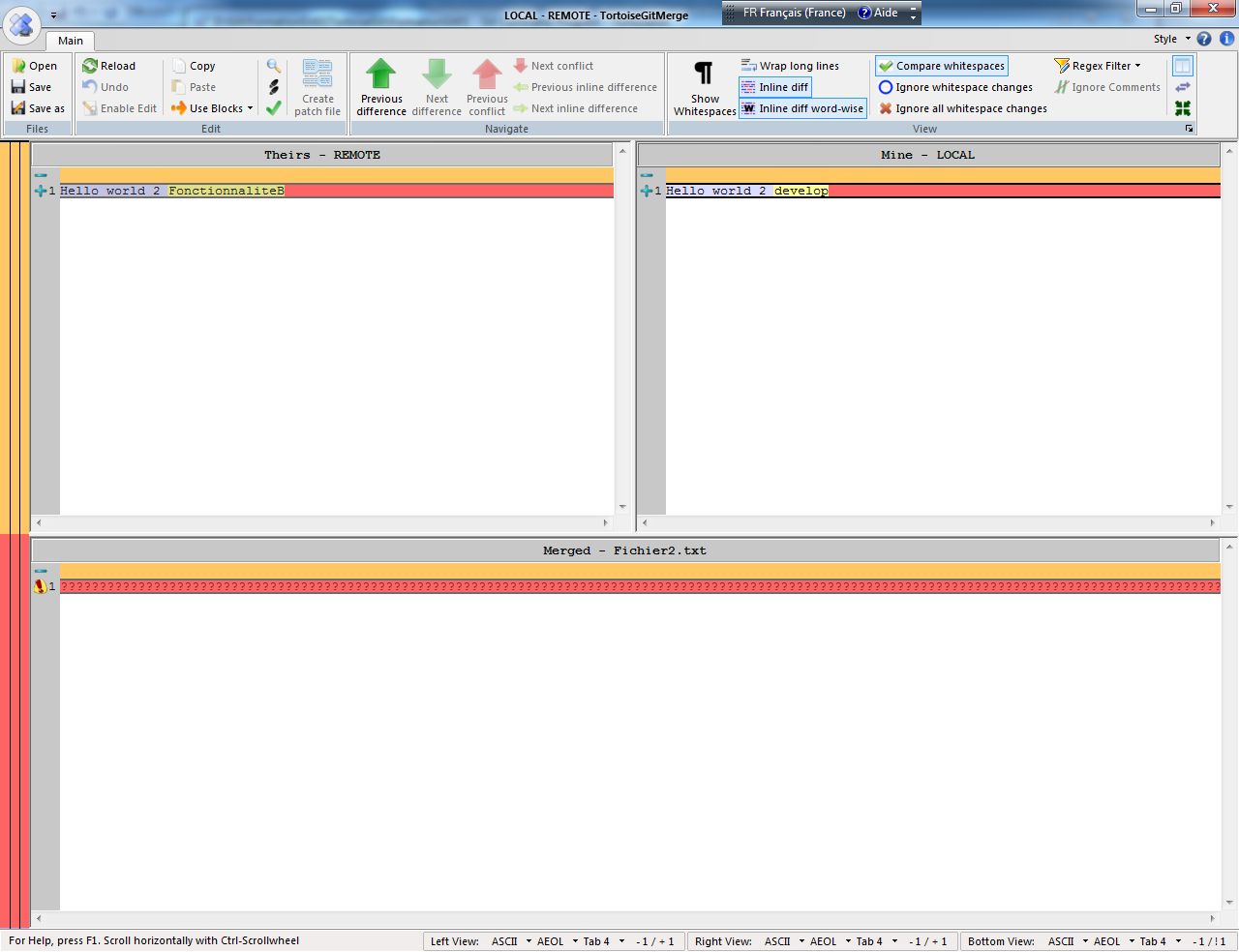


Figure - Mise en exergue du conflit en positionnant côte à côte les contributions

1. Le résultat obtenu après la résolution du conflit est le suivant (Figure 67) :

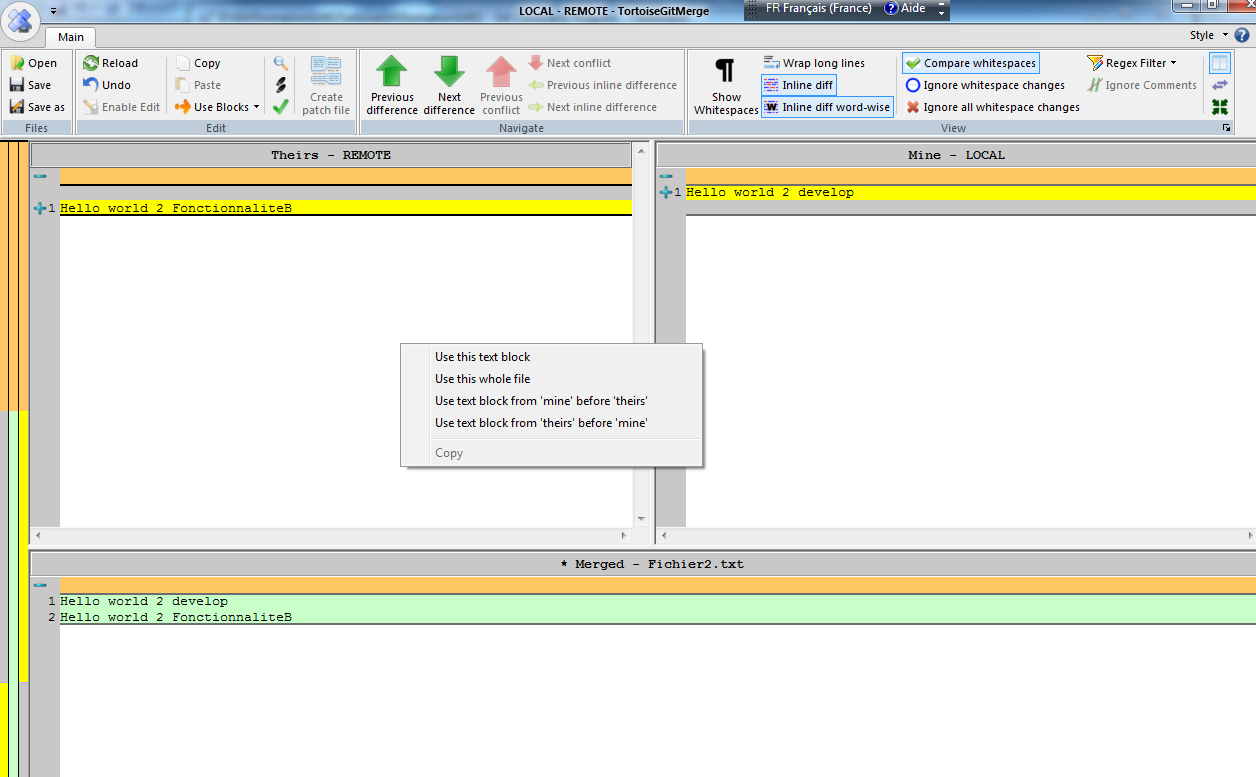


Figure - Résultat obtenu après sélection de la contribution souhaitée pour résoudre le conflit

1. Penser à sauver les modifications (ctrl +s). Le fichier peut alors être marqué comme résolu (Figure 68)

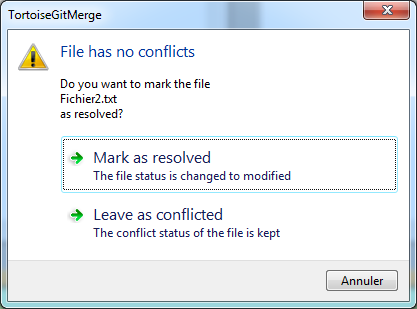


Figure - Fenêtre indiquant que le fichier peut être marqué comme résolu

## Terminer le *Merge* en effectuant un commit du fichier modifié

1. Sur le fichier sur lequel le conflit a été résolu, sélectionner le menu contextuel *Git Commit -> « develop »…* (Figure 18)
2. Modifier éventuellement le message de commit afin d’indiquer comment a été résolu le conflit (Figure 69). L’opération est terminée à l’affichage de la fenêtre suivante (Figure 70).



Figure – *Commit* du fichier sur lequel le conflit a été résolu

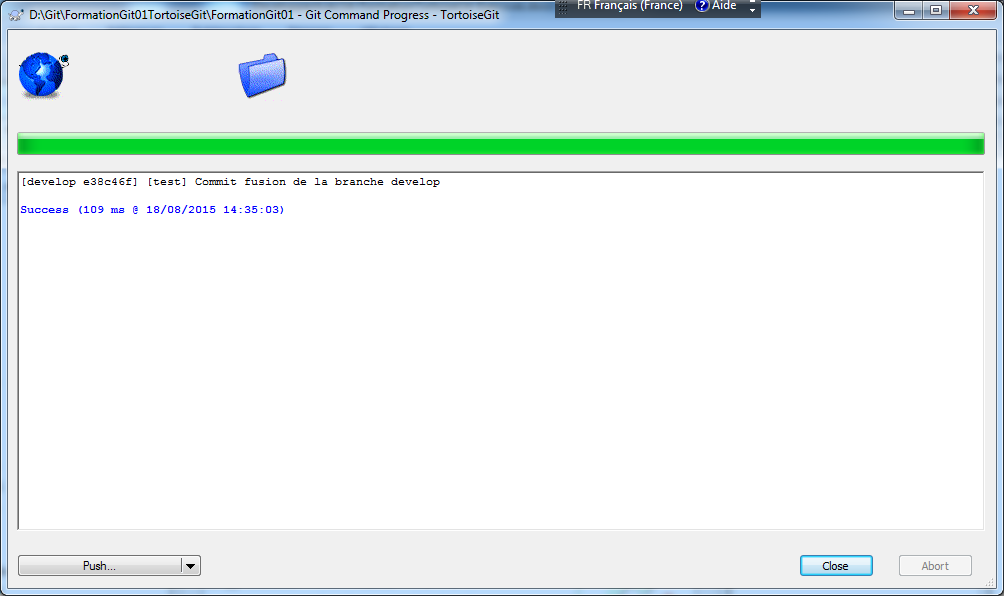


Figure - Fin du *Merge* après résolution des conflits par commit du fichier modifié

1. Visualiser l’arbre de révision après le *Merge* (Figure 71) grâce au menu contextuel *Tortoise Git > Show log* depuis le dossier parent (Figure 37)

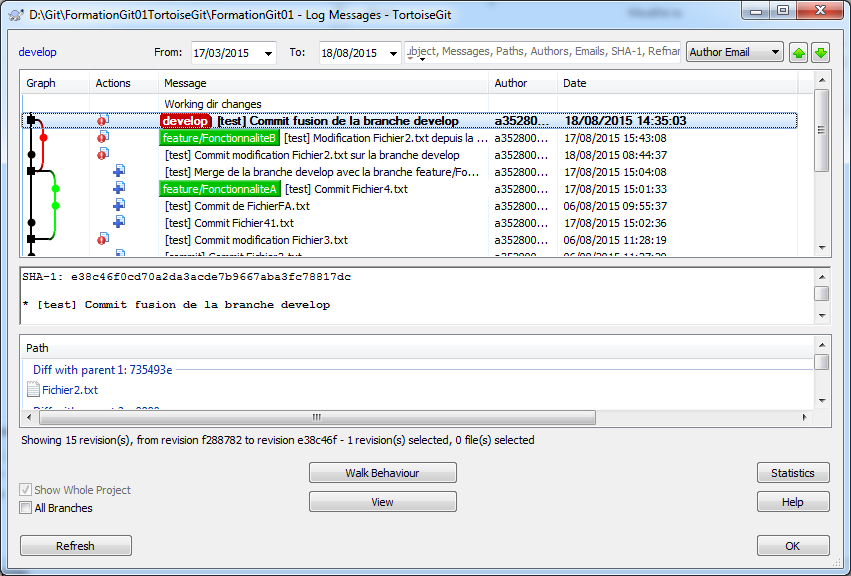


Figure - Arbre de révision après le *Merge* conflictuel

# Exercice 9 – Récupérer les changements apportés par un commit et les reproduire par un *Cherry Pick this commit …*

But de l’exercice : Récupérer les changements associés au commit ayant permis de créer le fichier Fichier41.txt sur la branche *develop* et les reproduire sur la branche de fonctionnalité feature/fonctionnaliteA

## Se positionner sur la branche de fonctionnalité *feature/fonctionnaliteA*

1. Vérifier que le menu contextuel depuis le dossier parent affiche *Git Commit -> « feature/FonctionnaliteA »…* (Figure 54)
2. Si ce n’est pas le cas, se positionner sur la branche de fonctionnalité par le menu contextuel *TortoiseGit > Switch/Checkout…* (Figure 43). Sélectionner la branche de fonctionnalité dans la fenêtre suivante (Figure 44) et visualiser le résultat (Figure 45)

## Visualiser l’arbre de révision de toutes les branches et repérer le commit correspondant à la création du fichier Fichier41.txt. Reproduire ce commit sur la branche de fonctionnalité feature/fonctionnaliteA

1. Visualiser l’arbre de révision de toutes les branches(Figure 72) grâce au menu contextuel *TortoiseGit >* *Show log* depuis le dossier parent (Figure 32).
2. Repérer le commit correspondant à la création du fichier Fichier41.txt (Figure72)



Figure - Repérage du commit à reproduire dans une autre branche

1. S’assurer que la case à cocher *All Branches* est bien cochée afin de visualiser l’historique de toutes les branches (Figure 73)

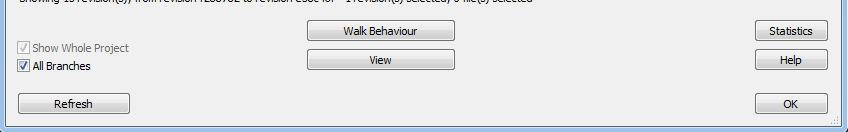


Figure - Case à cocher *All Branches* cochée dans l’arbre de révision

1. Depuis le commit repéré, sélectionner le menu contextuel *Cherry Pick this commit…* (Figure 74). Valider la fenêtre de confirmation en cliquant sur le bouton *Continue* (Figure 75)



Figure 74 - Menu contextuel *Cherry Pick this commit …*

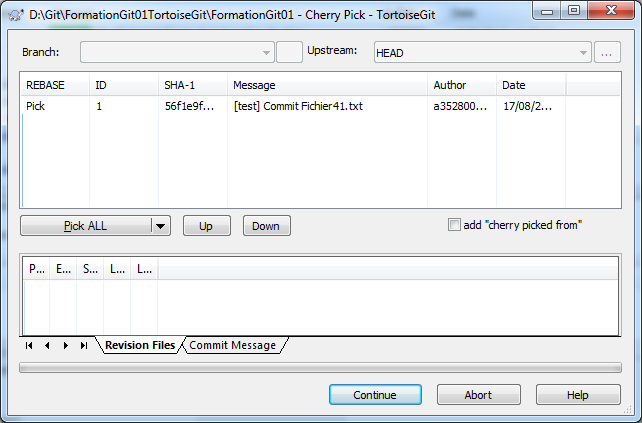


Figure - Fenêtre de confirmation associée au menu contextuel *Cherry Pick this commit…*

1. L’opération est terminée à l’affichage de la fenêtre suivante (Figure 76). Pour la finaliser il suffit de cliquer sur le bouton *Done*.

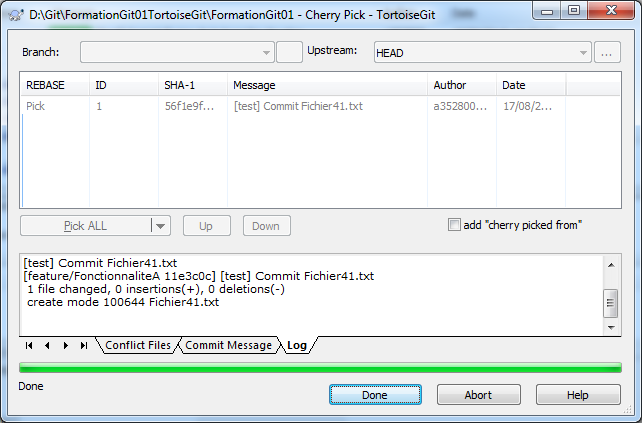


Figure - Fenêtre de fin du *Cherry Pick this commit...*

## Visualiser l’arbre de révision de toutes les branches après le *Cherry Pick this commit…*

1. L’arbre de révision s’affiche alors automatiquement (Figure 77)

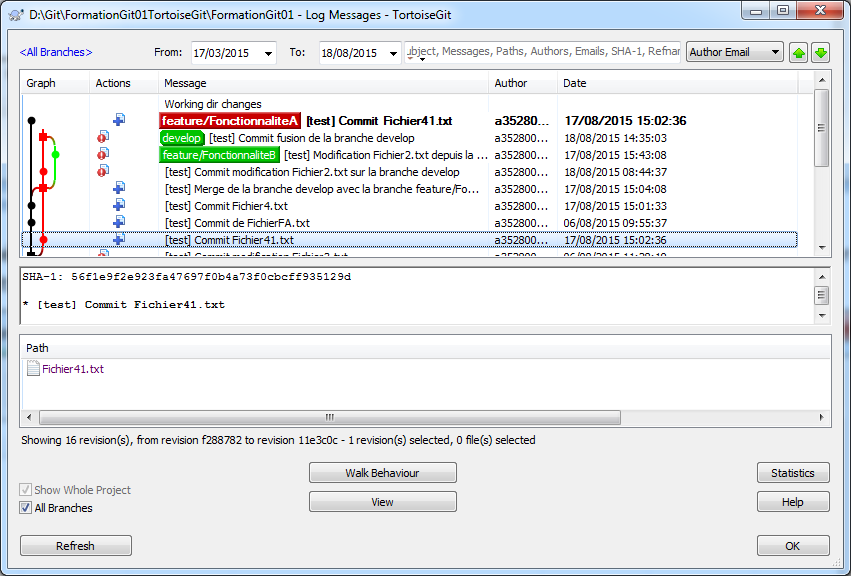


Figure - Arbre de révision après le *Cherry Pick this commit…*

# Exercice 10 – Poser une étiquette pour référencer une version et corriger la version

But de l’exercice : Créer une branche de version, poser une étiquette pour référencer une version, faire une modification sur la branche de version et la reporter sur la branche *develop*

Etat des branches *develop*, *feature* et version au début et à la fin de l’exercice :

Début Fin

## Créer une branche de version, faire une correction de bug et consigner les modifications

1. Créer une branche de version (Figures 28, 29 et 30) ayant pour source la branche *develop*. Respecter les règles de nommage en la nommant R1 (Figure 78).
2. Visualiser les différentes branches grâce au menu contextuel *TortoiseGit > Switch/Checkout…* (Figure 79).

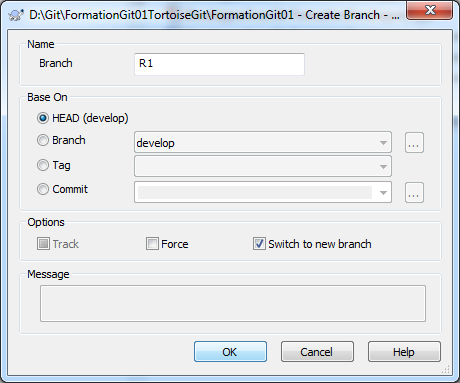


Figure - Nouvelle branche de version

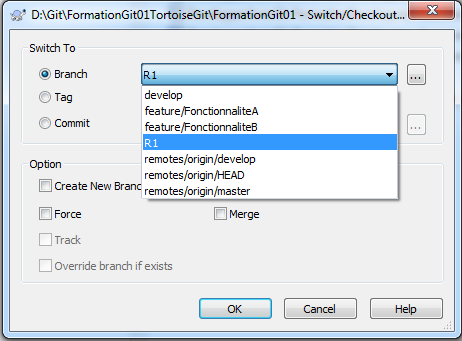


Figure - Différentes branches créées

1. Modifier le fichier Fichier4.txt en ajoutant par exemple une ligne Hello world 4. Enregistrer les changements sur cette branche en consignant le fichier modifié dans le dépôt Git (Figures 18, 19 et 20)

## Poser une étiquette sur la version candidate à la mise en production

1. Sélectionner le menu contextuel *TortoiseGit > Create Tag…* à partir du dossier parent (Figure 80)

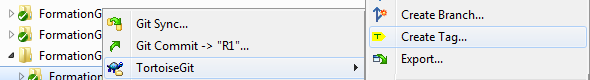


Figure - Menu contextuel *TortoiseGit >* *Create Tag…* depuis l'objet *Tags*

1. Nommer l’étiquette V1.0.0 en respectant la règle de nommage *[<préfixe>]<x.y.z>*, puis finaliser en cliquant sur le bouton *OK* (Figure 81)
2. Visualiser l’arbre de révision sur la branche de version R1 créée précédemment afin de voir la pose de l’étiquette (Figure 82)

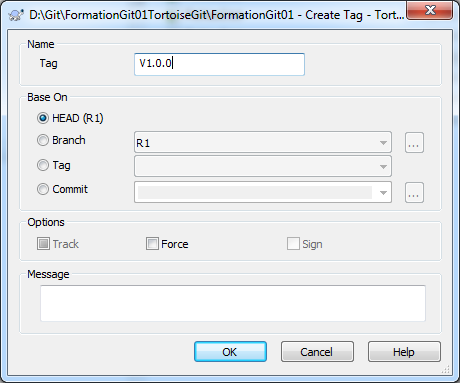


Figure - Création d'une étiquette de version

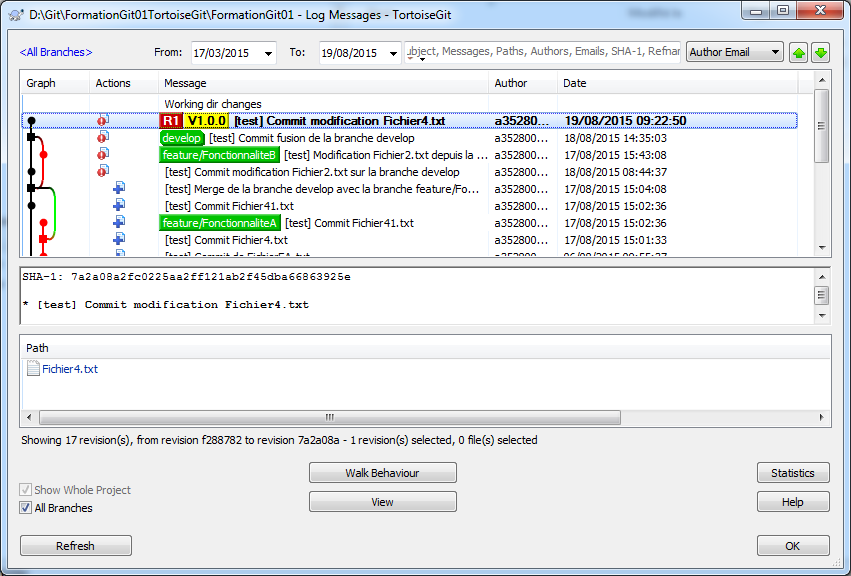


Figure - Arbre de révision sur la branche de version après la pose de l'étiquette

## Développer sur la branche *develop* et enregistrer les modifications

1. Se positionner sur la branche *develop* grâce au menu contextuel *TortoiseGit > Switch/Checkout …* sur le dossier parent (Figures 34, 35 et 36)
2. Modifier le fichier Fichier1.txt sur la branche *develop* en ajoutant par exemple la ligne Hello world 1
3. Consigner les modifications apportées au fichier dans le dépôt Git (Figures 18, 19 et 20)
4. Visualiser l’arbre de révision (Figure 83) sur la branche *develop* grâce au menu contextuel *TortoiseGit > Show log* (Figure 37)

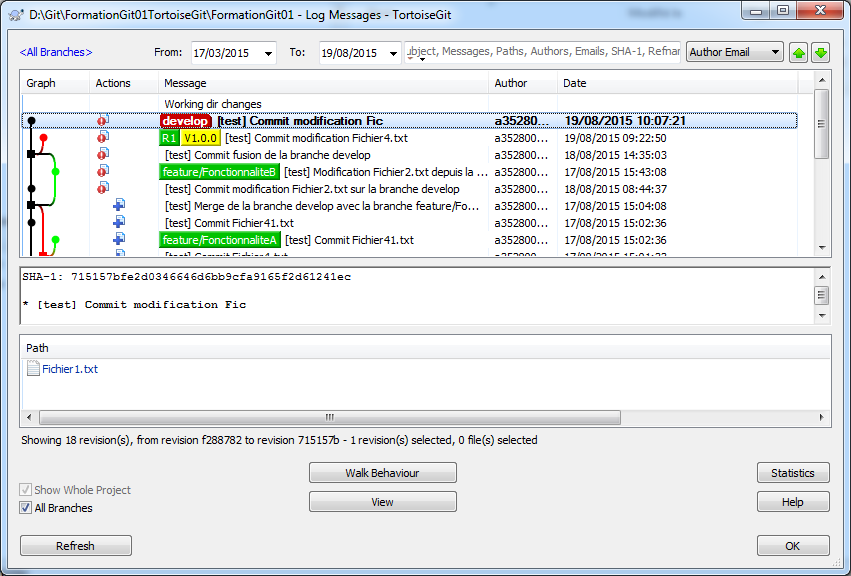


Figure - Arbre de révision sur la branche *develop* avant le *Merge*

## Apporter une correction sur la branche R1 et enregistrer les modifications

1. Se positionner sur la branche de version R1
2. Modifier le fichier Fichier4.txt sur la branche de version R1 en ajoutant par exemple la ligne Hello world 4 R1
3. Consigner le fichier dans le dépôt Git (Figures 18, 19 et 20)
4. Visualiser l’arbre de révision (Figure 84) sur la branche R1grâce au menu contextuel *TortoiseGit > Show log* (Figure 32)

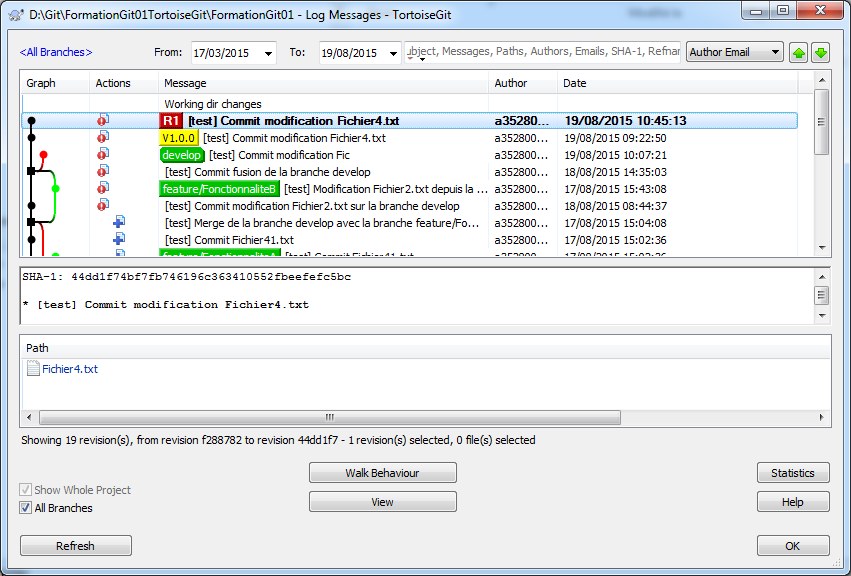


Figure - Arbre de révision sur la branche de version R1 avant le *Merge*

## Reporter les corrections effectuées sur la branche de version sur la branche develop grâce à un Merge

1. Se positionner sur la branche *develop* (Figures 34, 35 et 36)
2. Depuis le dossier parent, sélectionner le menu contextuel *TortoiseGit > Merge* (Figure 55)
3. Dans *From*, sélectionner la branche qui va fusionner avec la branche courante à mettre à jour, en l’occurrence R1 (Figure 56). Ne pas oublier de renseigner le message associé au *Merge*.
4. Visualiser l’arbre de révision depuis la branche *develop* (Figure 85)
5. Se positionner sur la branche de version R1 et visualiser l’arbre de révision depuis la branche de version R1 (Figure 86)

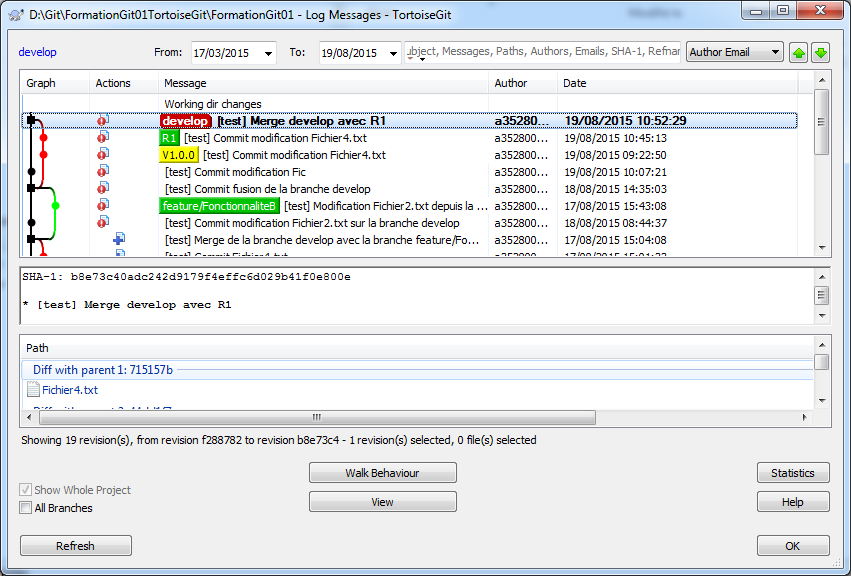


Figure - Arbre de révision sur la branche *develop* après le *Merge*

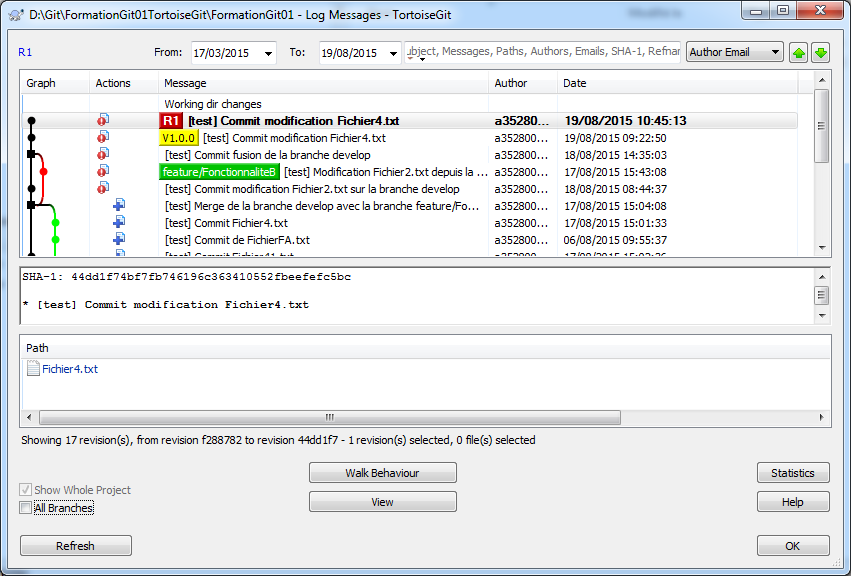


Figure - Arbre de révision sur la branche de version R1 après le *Merge*

## Schéma récapitulatif de l’état des 3 branches avant et après le *Merge*

Avant le *Merge* Après le *Merge*

Figure - Etat des 3 branches avant et après le *Merge*

# Exercice 11 – Pousser et récupérer des contributions

But de l’exercice : Développer à plusieurs sur trois dépôts locaux distincts. Pousser et récupérer les contributions de chacun

## Faire un deuxième clone du dépôt distant

1. Cloner le dépôt Git distant et l’associer à un répertoire différent du premier (Figures 5, 6, 7, 8 et 9).

Le répertoire local de destination sera différent du premier clone.

## Visualiser qu’aucun des fichiers créés dans le précédent dépôt local ne sont présents

1. Les seuls fichiers disponibles dans le répertoire associé à ce nouveau dépôt local correspondent à l’init du dépôt distant avec les fichiers InfoProjet.txt et .gitignore (Figure 88)

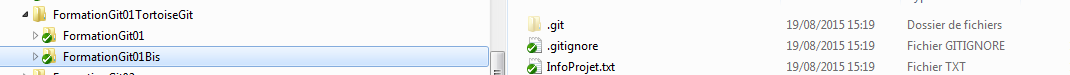


Figure - Fichiers disponibles sur la branche *develop* d'un dépôt fraichement cloné

## Pousser les contributions effectuées précédemment sur le premier dépôt local sur le dépôt distant

1. Se positionner dans le répertoire relatif au dépôt local sur lequel tous les développements ont été effectués.
2. Se positionner sur la branche *develop* (Figures 34, 35 et 36)
3. Sélectionner *TortoiseGit >* *Push…* dans le menu contextuel (Figure 89)

Figure - Menu contextuel *TortoiseGit >* *Push...*

1. Enchainer ensuite les 4 panneaux suivants (Figures 90, 91, 92 et 93).

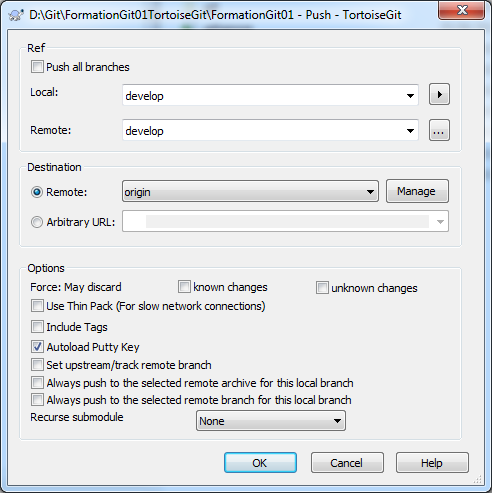


Figure - Informations sur la branche et le dépôt distant sur lequel seront remontées les contributions

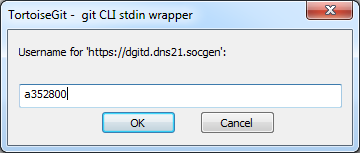


Figure - Information de login sur le dépôt distant - matricule SESAME

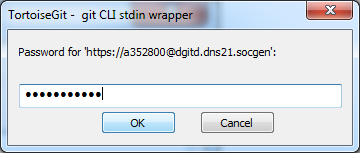


Figure - Information de login sur le dépôt distant – mot de passe SESAME

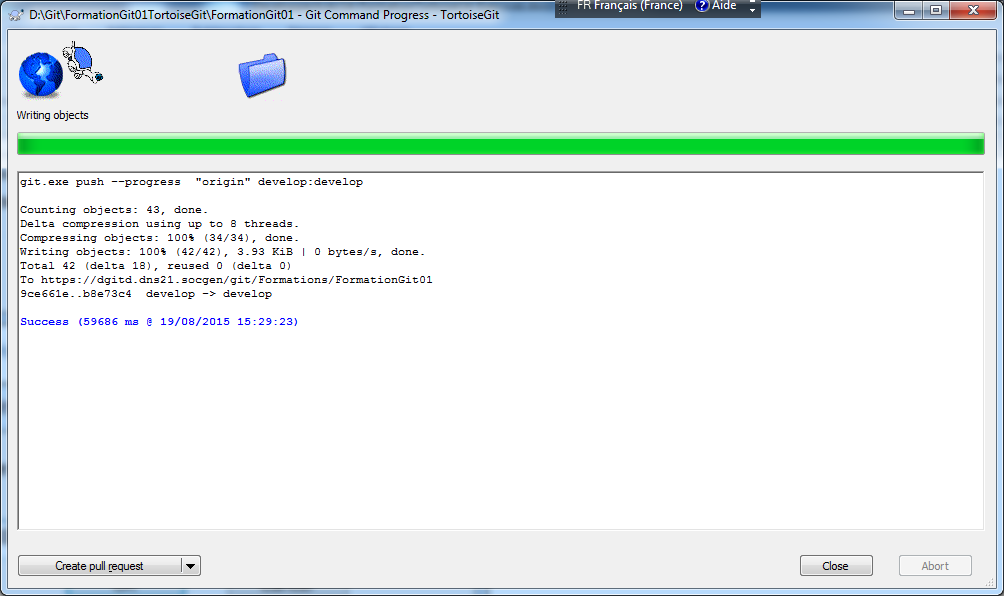


Figure - Résultat du *Push*

1. Se rendre compte que tous les commits parents ont été envoyés vers le dépôt distant, même ceux effectués sur une autre branche.
2. Pour être exhaustif, il faudrait effectuer un *Push* de la branche de fonctionnalité si celle-ci est partageable et partagée.

## Récupérer sur le deuxième dépôt local les contributions précédemment poussées sur le dépôt distant

1. Se repositionner dans le deuxième répertoire relatif au dépôt local à mettre à jour
2. Sélectionner le menu contextuel *TortoiseGit >* *Fetch* depuis le dossier parent (Figure 93)

Figure - Menu contextuel *TortoiseGit >* *Fetch...*

1. Enchaîner ensuite les 4 panneaux suivants (Figures 95, 96, 97 et 98).

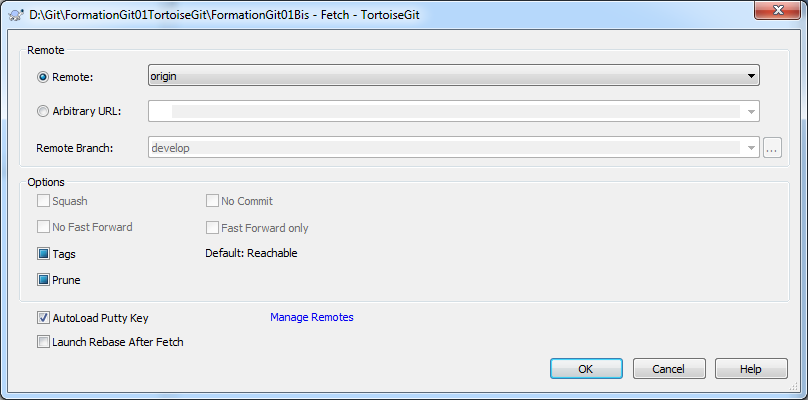


Figure - Choix de la branche distante



Figure - Information de login sur le dépôt distant - matricule SESAME

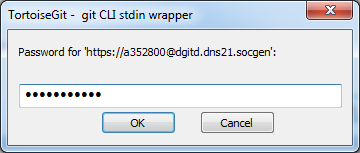


Figure - Information de login sur le dépôt distant – mot de passe SESAME

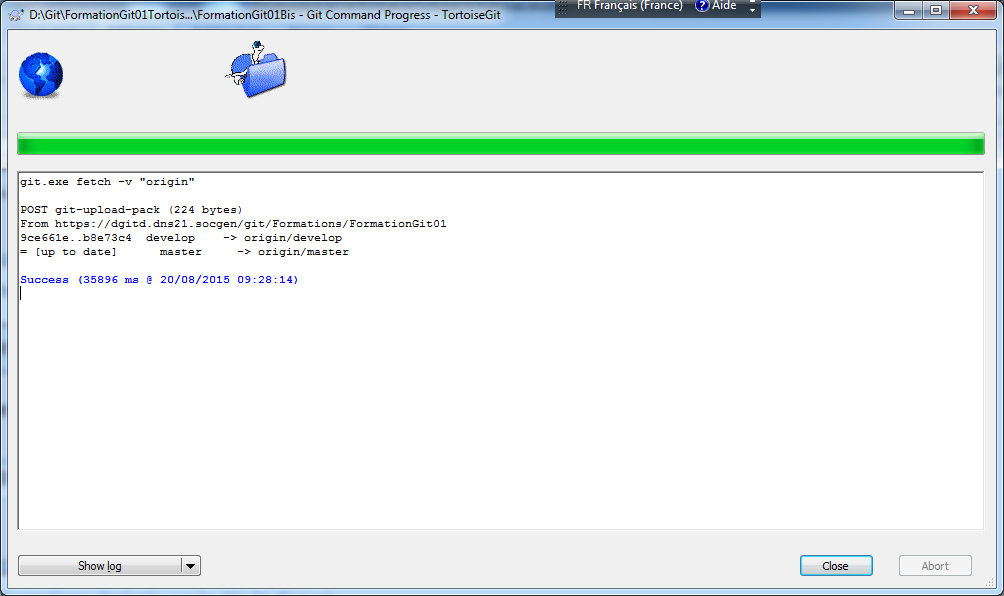


Figure - Fin du *Fetch*

1. Visualiser l’arbre de révision et observer que le pointeur *develop* est toujours positionné sur le nœud d’origine (Figure 99)

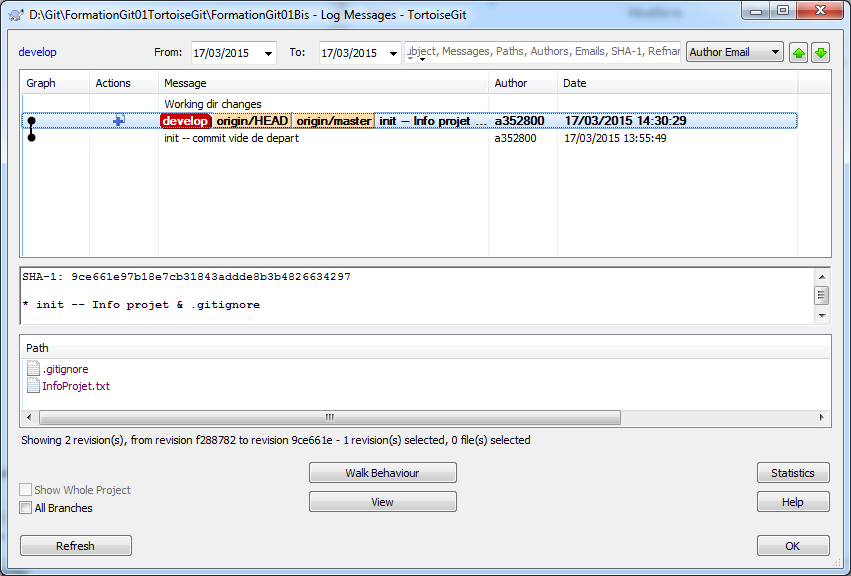


Figure - Arbre de révision après le *Fetch*

1. Bien vérifier que la branche *develop* est la branche courante, c’est-à-dire pointée par **HEAD**. Pour cela utiliser le menu contextuel sur le dossier parent (Figure 100). Il doit apparaître *GitCommit -> «****develop****» …*

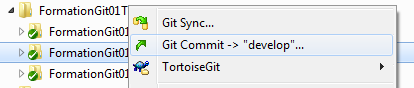


Figure - Vérification que la branche *develop* est la branche courante

1. Mettre à jour la branche *develop* à partir de la branche remotes/*origin/develop* (Figure 101) grâce au menu contextuel *TortoiseGit >* *Merge* sélectionné depuis le dossier parent (Figure 55)

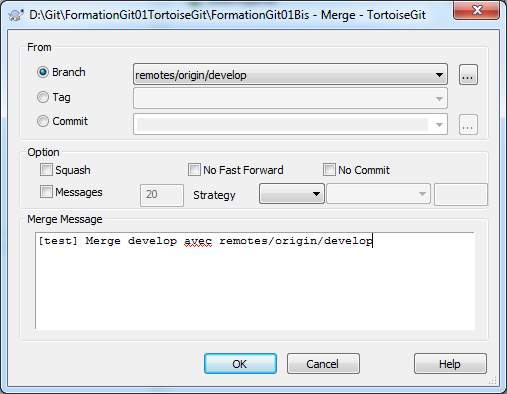


Figure - Choix de la branche remotes/*origin/develop* lors d’un *Merge*

Pour éviter des opérations inutiles, Git détecte, lors du *Merge*, qu’au lieu de combiner le sommet des 2 branches (*develop* et *remotes/origin/develop*) puisque l’une est à la suite de l’autre, il est préférable d’avancer la référence *develop*. Ceci s’appelle un Fast-forward (Figure 102).

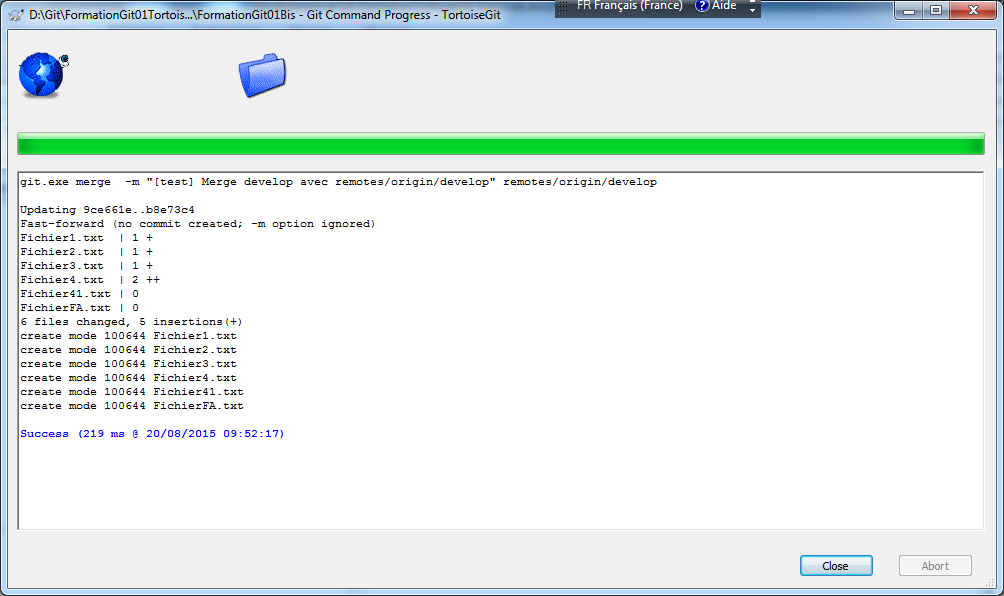


Figure - Fenêtre de résultat d’un *Fast-forward* lors d'un *Merge*

1. Dans l’arbre de révision, vérifier que le pointeur *develop* n’est plus positionné sur le nœud d’origine (Figure 103)

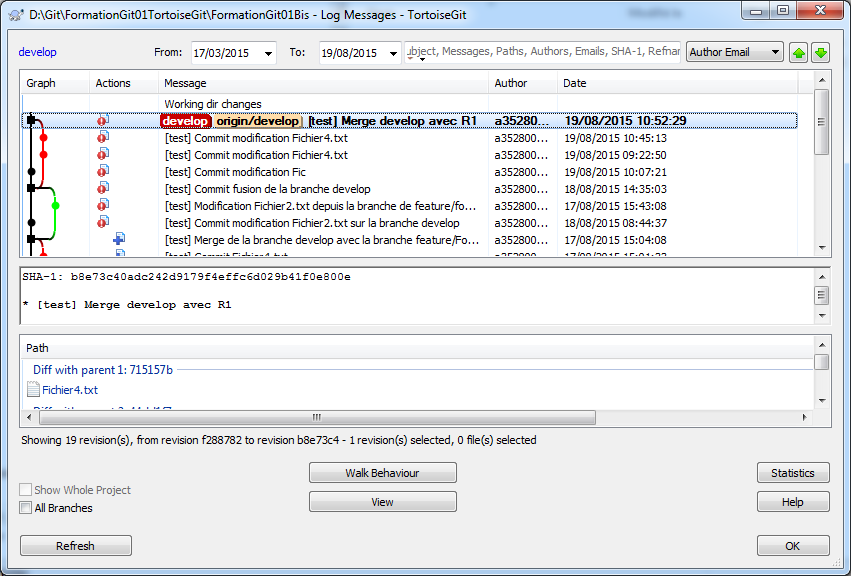


Figure - Arbre de révision après le *Merge* déclenché par l’utilisateur, suite à un Fetch

## Faire un troisième clone du dépôt distant. Vérifier que ce dépôt local n’est pas vide

1. Cloner le dépôt Git distant (Figures 5, 6, 7, 8, 9 et 104). Le répertoire local de destination sera différent du premier et du deuxième clone.

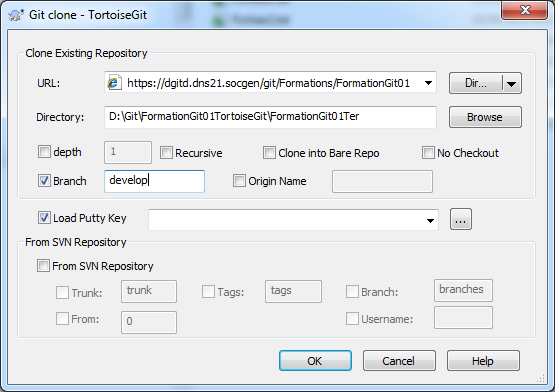


Figure - Nouveau clone

1. Visualiser que les fichiers mis dans le dépôt distant sont bien dans le répertoire associé au dépôt local (Figure 105).

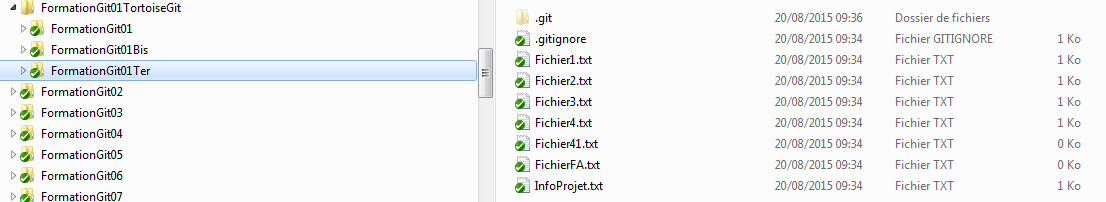


Figure - Fichiers présents dans le dépôt local après le clone

1. Visualiser l’arbre de révision et constater que le pointeur *develop* est bien positionné sur la dernière action faite dans le premier dépôt local (Figure 106)

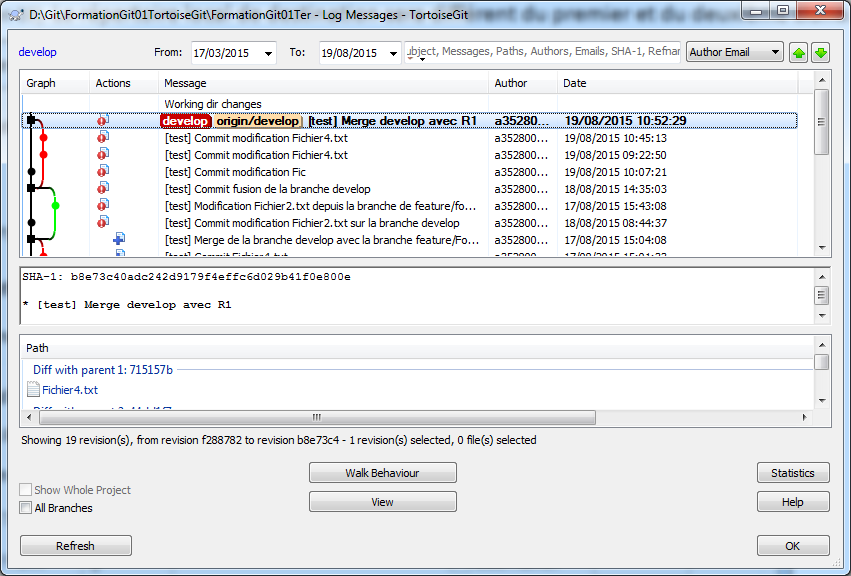


Figure - Arbre de révision du troisième dépôt local après le clone

## Développer sur le troisième dépôt local, enregistrer les changements et les pousser sur le dépôt distant

1. Créer un nouveau fichier FichierC31.txt
2. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
3. Consigner le nouveau fichier dans le dépôt local Git (Figures 18, 19 et 20)
4. Pousser les contributions sur le dépôt distant. Depuis le dossier parent, sélectionner le menu contextuel *TortoiseGit >* *Push …* (Figure 89). Enchaîner les panneaux (Figures 90, 91, 92 et 93)

## Développer sur le premier dépôt local, enregistrer les changements mais avant de les pousser sur le dépôt distant, récupérer les contributions développées sur le troisième dépôt local

1. Se repositionner dans le premier répertoire relatif au premier dépôt local, sur la branche *develop*
2. Créer un nouveau fichier Fichier5.txt
3. Ajouter le fichier à l’index Git (Figures 12, 13 et 14)
4. Consigner le nouveau fichier dans le dépôt local Git (Figures 18, 19 et 20)
5. Ne pas pousser les contributions sur le dépôt distant. Il faut d’abord mettre à jour le dépôt local par rapport au dépôt distant qui a été modifié à l’étape précédente. Afin d’impacter la version de travail, il est préférable d’effectuer un *Pull* (*Fetch* + *Merge*).
6. Sélectionner le menu contextuel *TortoiseGit > Pull…* depuis le dossier parent (Figure 107). Enchainer les panneaux suivants (Figures 108, 109, 110 et 111).

Figure - Menu contextuel *TortoiseGit > Pull…*



Figure - Choix de la branche distante lors d'un *Pul*

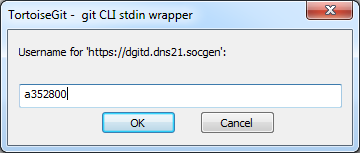


Figure - Information de login sur le dépôt distant - matricule SESAME



Figure - Information de login sur le dépôt distant - mot de passe SESAME

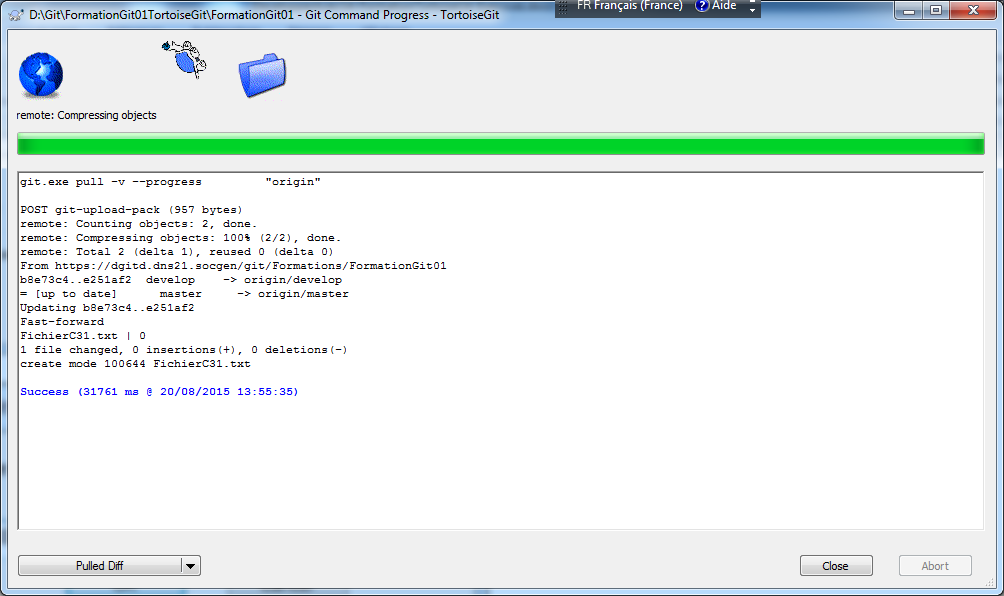


Figure - Résultat du *Pull* (*Fetch* + *Merge*)

1. Visualiser l’arbre de révision après le *Pull* (Figure 112)

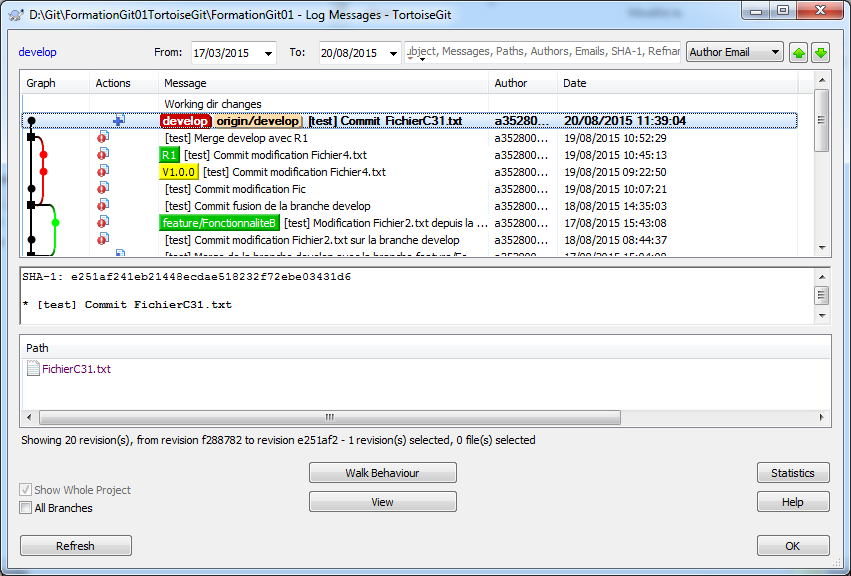


Figure - Arbre de révision après le *Pull*

1. Pousser les contributions sur le dépôt distant. Sélectionner le menu contextuel *TortoiseGit >* *Push …* depuis le dossier parent (Figure 89). Enchaîner les panneaux (Figures 90, 91, 92 et 93)

## Pour finir, et être complètement cohérent, mettre à jour le troisième dépôt local avec les contributions poussées sur de dépôt distant depuis le premier dépôt local

1. Mettre à jour le troisième dépôt local en se repositionnant dans le troisième répertoire relatif au troisième dépôt local.
2. Sélectionner le menu contextuel *TortoiseGit > Pull…* depuis le dossier parent (Figure 107). Enchaîner les panneaux suivants (Figures 108, 109, 110 et 111).