Sujet de TP GIT

Contenu

[Sujet de TP 2](#_Toc464666256)

[TP 0 – Configuration (config, ssh-keygen) 2](#_Toc464666257)

[TP 1 – Les débuts (add, commit, init, pull, push, status) 4](#_Toc464666258)

[TP 2 – Gestion des conflits 11](#_Toc464666259)

[TP 3 – Ignorer des fichiers 22](#_Toc464666260)

[TP 4 – Manipulation des zones de travail Git (mv, reset, rm) 24](#_Toc464666261)

[TP 5 – Manipulation des branches (branch, checkout) 26](#_Toc464666262)

[TP 6 – Fusion de branches locales et distantes 27](#_Toc464666263)

[TP 7 – Tags : Manipulation et partage (tag) 29](#_Toc464666264)

[TP 8 – Visualiser l’historique (blame, diff, log) 30](#_Toc464666265)

[TP 9 – Remonter le temps 37](#_Toc464666266)

[TP 10 – Rebase de branche 39](#_Toc464666267)

[TP 11 – Mettre de côté ses développements 40](#_Toc464666268)

[TP 12 – Réécrire l’histoire ! 41](#_Toc464666269)

[ANNEXE 43](#_Toc464666270)

[GIT 43](#_Toc464666271)

[Commandes GIT 43](#_Toc464666272)

[Opérations avancés 44](#_Toc464666273)

[Fichiers particulier 44](#_Toc464666274)

[HUGO 45](#_Toc464666275)

[Qu’est-ce qu’Hugo ? 45](#_Toc464666276)

[Quelques commandes 45](#_Toc464666277)

[MARKDOWN 46](#_Toc464666278)

[Qu’est-ce que le MarkDown ? 46](#_Toc464666279)

[Quelques commandes 46](#_Toc464666280)

# Sujet de TP

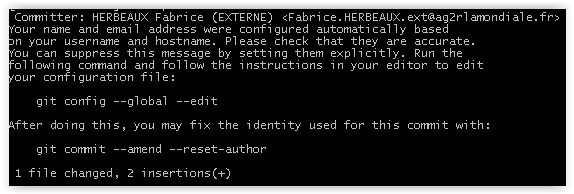
## TP 00 – Configuration (config, ssh-keygen)

Vous allez tout d’abord commencer par configurer Git sur votre poste.

Il s’agira dans un premier temps de configurer :

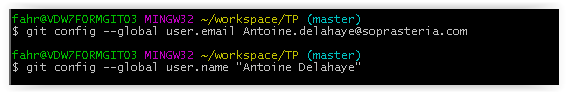
* Vos nom et prénom
* Votre email

De base, Git va chercher ces informations par rapport au compte système connecté.



Configurez ces informations à l’aide de la commande ‘git config’.

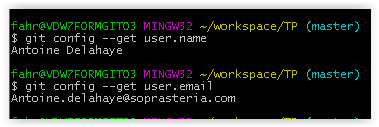
Exemple :



L’option --global permet une configuration globale qui sera prise en compte pour tous les dépôts Git à venir sur cette session utilisateur.

Il est possible de surcharger cette configuration en se positionnant dans un dépôt Git. Il suffit alors de ne pas préciser cette option pour que le nouveau paramètre ne s’applique qu’au dépôt courant.

Commandes pour vérifier que vos actions sont bien prises en compte :



La génération d’une clé ssh est indispensable pour que votre poste de travail puisse se connecter au dépôt distant (serveur GitLab).

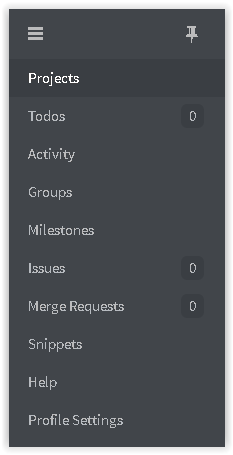
Une autre option, non utilisée pour ce TP, est la connexion https (login/mot de passe à fournir à GitLab dans ce cas).

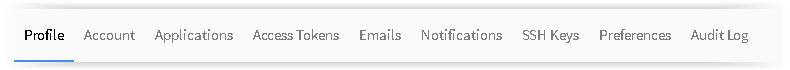
Génération de la clé ssh : Lancez la commande « ***ssh-keygen*** » depuis une console Git Bash. Appuyez ensuite trois fois sur la touche « Entrée ».

Par défaut, la clé se trouve dans un dossier caché. Exécutez la commande : ***cat ~/.ssh/id\_rsa.pub***

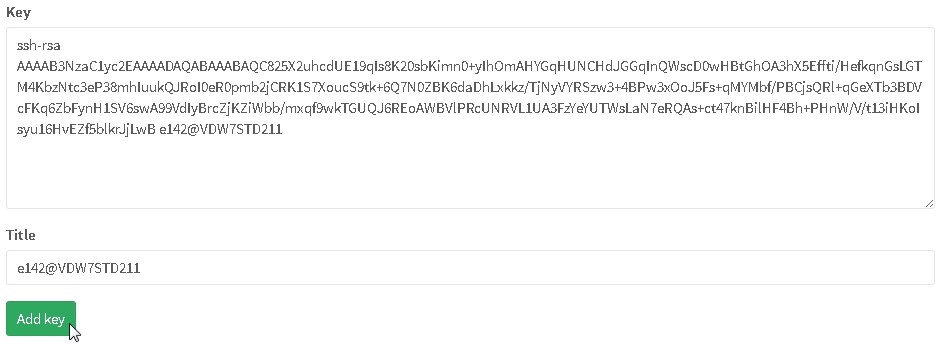
Connectez-vous à GitLab (<http://git-dev.server.lan>) avec vos identifiants Windows.

Pour ajouter la clé, allez dans « Profile Settings », puis dans « SSH Keys ».





Puis, copiez/collez la clé et ajoutez la. Exemple:



## TP 01 – Les débuts (add, commit, init, pull, push, status)

Créez un dossier « workspace » sur votre poste.

Création d’un dépôt Git initial depuis Git Bash :

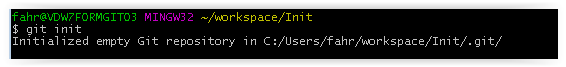
Dans « workspace », créez un dossier « depot-git » et s’y positionner.

Commande ‘cd /c/…/workspace/mon-dossier’ ou ‘cd ‘+drag&drop du chemin depuis un explorateur Windows.

En utilisant la commande ‘git init’, initialisez le répertoire que vous venez de créer comme étant un dépôt git.

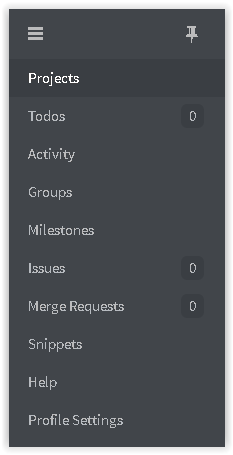
Cette commande, rarement utilisée vous sera utile pour commencer le suivi d’un nouveau projet de développement.

Exemple :

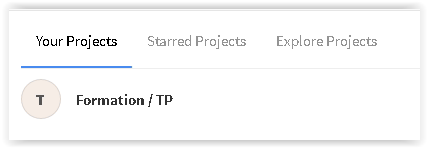


Récupérez le projet « TP » du groupe « Formation », présent sur le serveur GitLab git-dev, en utilisant la commande ‘git clone’.

#### Méthode pour trouver l’url d’un projet GitLab



Copier

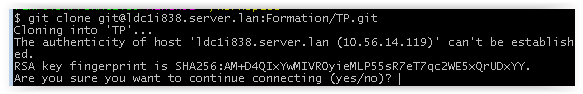


1

2

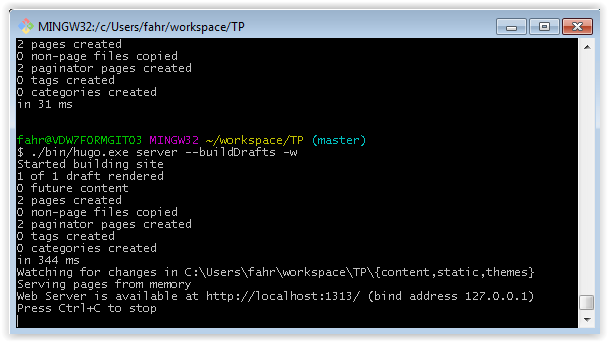
Lors de la première utilisation de la commande clone, il vous sera demandé d’accepter la connexion sh via la clé générée précédemment.

Exemple :



Le clone du projet TP permet la récupération d’un site web statique généré par le moteur Hugo.

Depuis une seconde console Git Bash, démarrez le serveur Hugo de la façon suivante :



Pour visiter le site web, allez sur <http://localhost:1313/>. Toutes les modifications effectuées sur le blog seront prises en compte immédiatement.

Travaillez maintenant en binôme (ou trinôme) et définissez communément un sujet d’article qui permettra d’enrichir le blog récupéré. Définissez un nom de fichier ‘article.md’ qui sera à créer dans le répertoire « TP/content ».

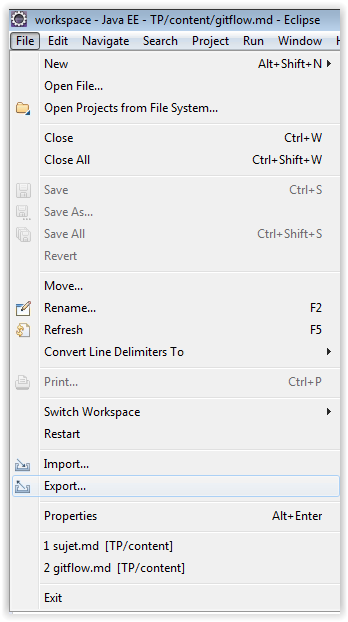
Utilisez l’annexe de ce document pour plus d’informations sur Hugo et Markdown (langage de balises).

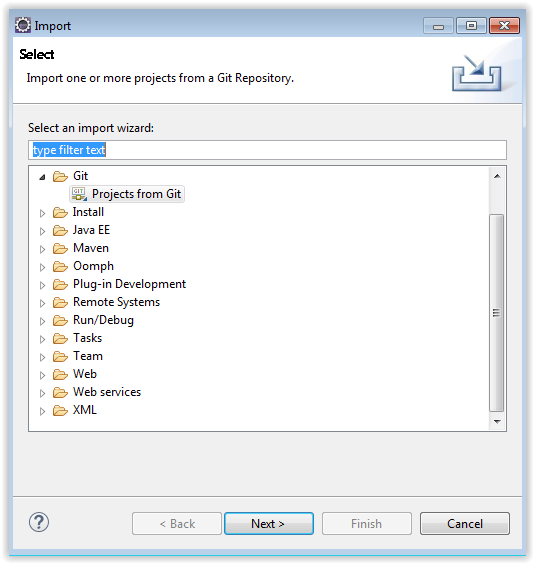
Toujours en binôme (1 poste de travail), créez le nouvel article. Exemple :

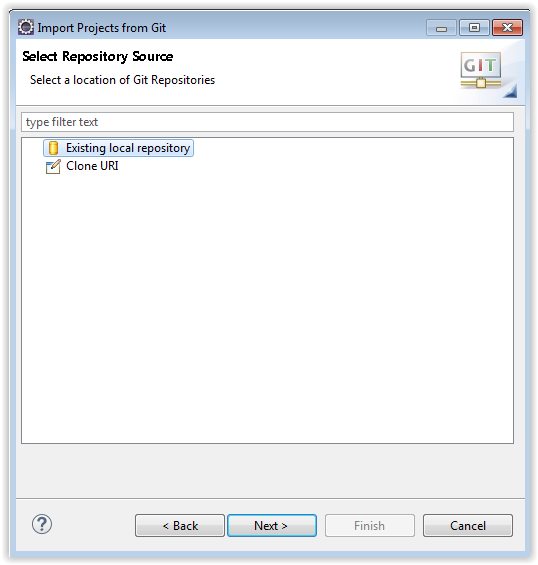


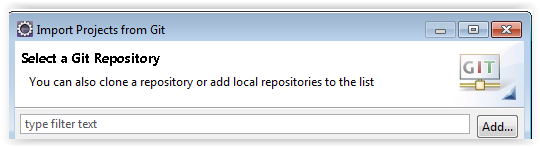
#### Importer le projet dans Eclipse

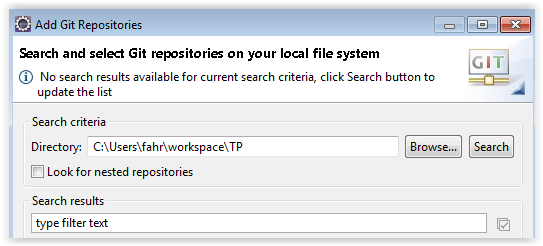
Lancez Eclipse : C:\Program Files\eclipse\eclipse.exe, puis importez le projet TP :









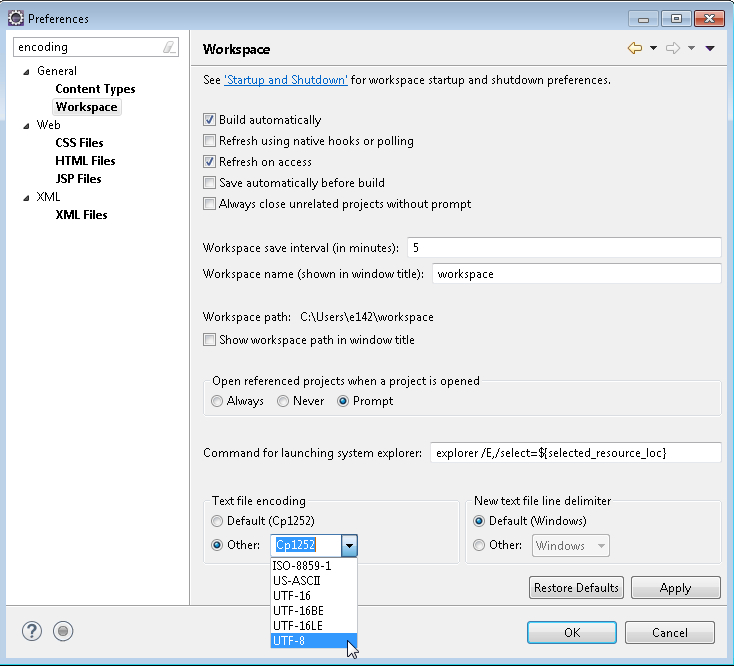


Directory : Choisissez l’emplacement de votre projet TP cloné.

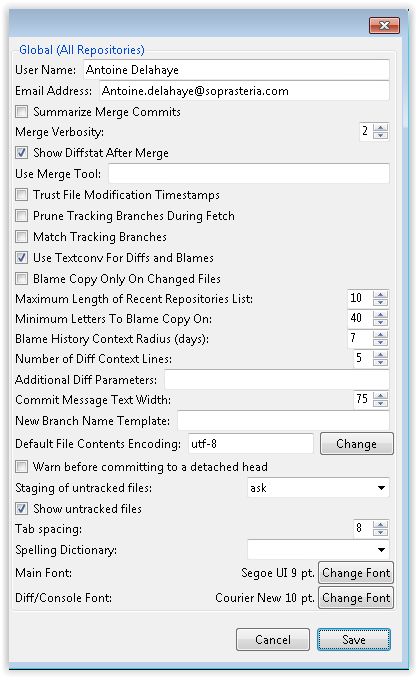
#### Configuration de l’encodage dans Eclipse et Git GUI

Le projet TP cloné étant encodé en UTF-8, il faut configurer les outils utilisés dans ce format.

Pour Eclipse, allez dans les préférences (onglet Windows -> Preferences) et saisissez encoding dans la zone de recherche :



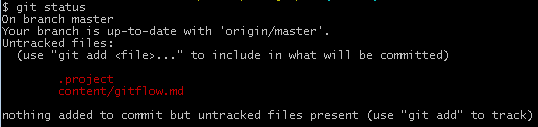
Pour Git GUI, lancez-le (commande « git gui » dans Git Bash). Ensuite, allez dans l’onglet Edit -> Options…



Depuis Eclipse, vous pouvez ouvrir TP\content\sujet.md qui servira d’exemple pour la création d’un nouvel article en Markdown.

Toujours en groupe, effectuez des modifications sur l’article créé.

Depuis Git Bash, pour voir les fichiers non suivis (untracked), modifiés (changes), à indexer (unstaged), etc., utilisez la commande ‘git status’ :



Indexez le nouvel article avec la commande ‘git add’.

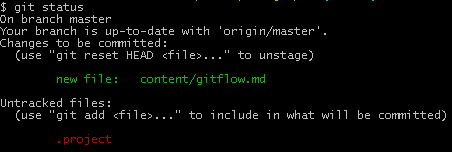
Exemple :



Vérifiez le nouvel état de votre dépôt Git avec la commande ‘git status’.

Le contenu de l’index est vert, le contenu non encore indexé est rouge.

Exemple :



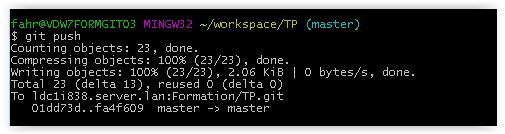
Créez un commit.

Exemple :



Partagez l’article avec la commande ‘git push’.

Exemple :



Retournez chacun à votre poste afin de récupérer les différents articles publiés.

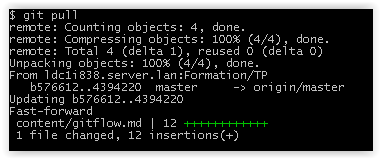
Pour cela, utilisez la commande ‘git pull’.

Exemple :

Résultat sur le poste utilisé pour la création de l’article :



Résultat sur un autre poste :



## TP 02 – Gestion des conflits

Modifiez votre article à plusieurs reprises, tout en le partageant régulièrement.

Mettez-vous d’accord par groupe pour générer des conflits que vous pourrez résoudre en expérimentant les outils Git Bash, Git GUI et le plugin GIT pour Eclipse.

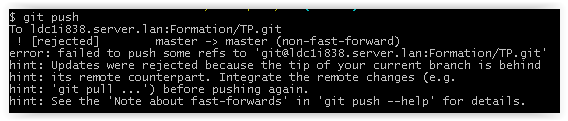
#### Utilisation de Git Bash pour la résolution de conflits

Expérimentez la commande ‘git commit –a –m "commentaire"’ qui permet de réaliser un commit des fichiers suivis modifiés sans passer par la commande ‘git add’.

Exemple :



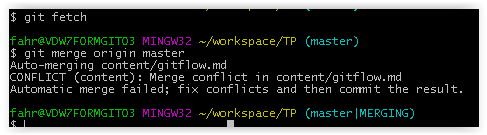
Partagez des modifications de façon à rencontrer un conflit comme ci-dessous :



Dans ce cas, il faut récupérer les modifications de la branche distante et effectuer une fusion avec les modifications de la branche locale.

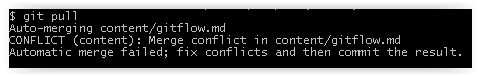
Expérimentez la méthode ‘git fetch’ suivie de la commande ‘git merge’.

Exemple :

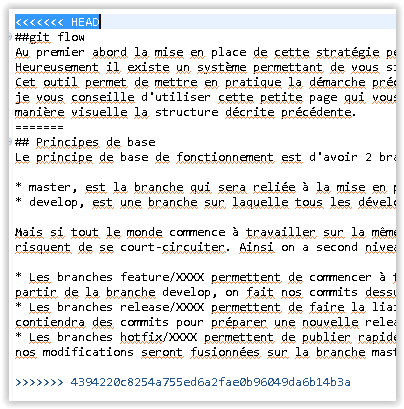


Expérimentez la méthode ‘git pull’.

Exemple :



Dans les deux cas, le fichier ouvert dans Eclipse est modifié comme dans l’exemple ci-dessous :

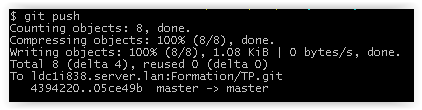


**La première zone** (délimitée par le HEAD et ====) correspond au contenu de la branche locale.

**La seconde zone** qui finit par un >>>SHA-1 correspond au contenu sur la branche distante

* Choisir ce que l’on veut garder (ne pas oublier de retirer les balises de conflits)
* Refaire un commit et un push

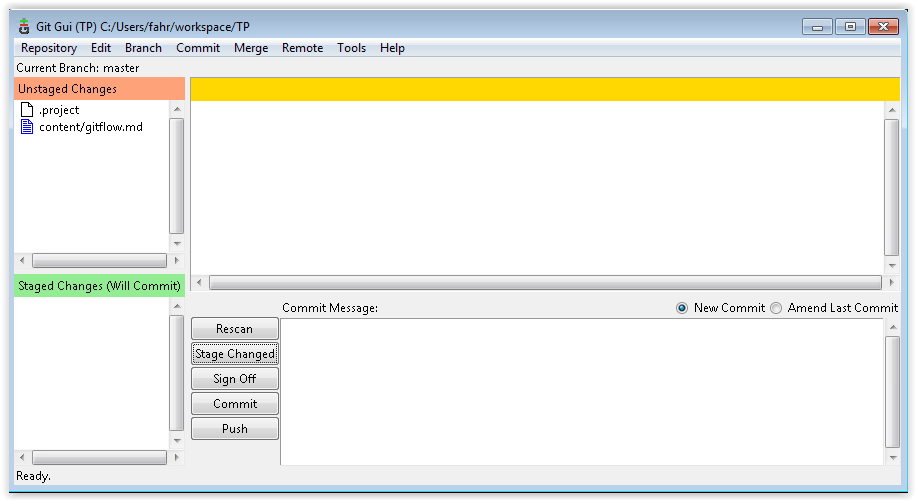




#### Utilisation de Git GUI pour la résolution de conflits

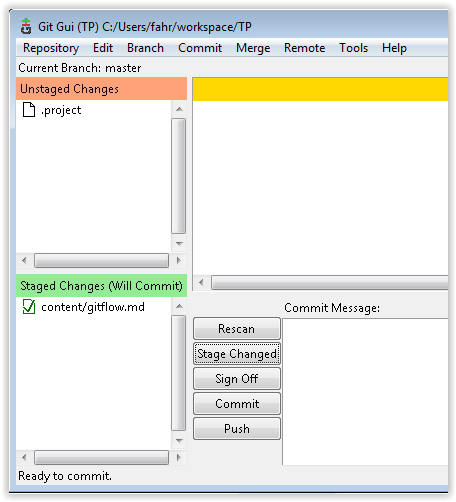
Effectuez des modifications depuis l’éditeur Markdown d’Eclipse, puis lancez Git Gui.

Effectuez le commit depuis Git Gui :



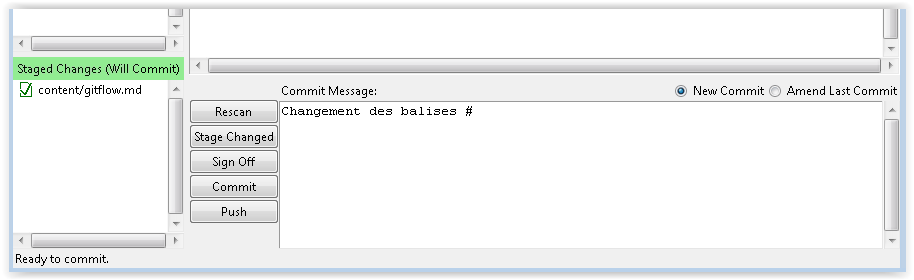
1

2



Une fois la modification indexée :

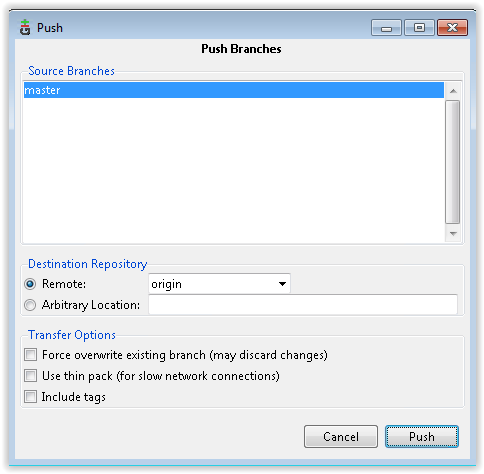
* Saisir le message du commit (obligatoire).
* Faire un commit (vous pouvez aussi utiliser le bouton « Push », qui va faire un commit suivi d’un push)
* Observez en bas de la fenêtre que le commit à bien été créé
* Poussez vos modifications



2

1

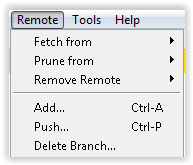




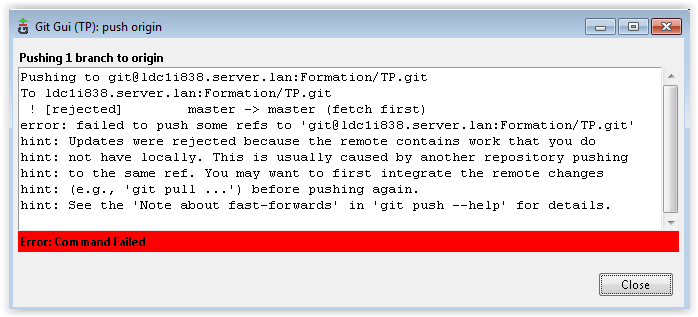
Sélectionnez la branche source

Puis le dépôt distant sur lequel pousser

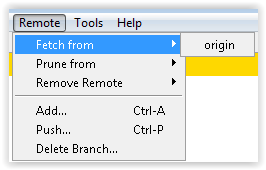
Poussez les modifications !

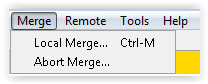


Partagez des modifications jusqu’à obtenir un conflit comme ci-dessous :



Gérer le conflit depuis Git Gui :

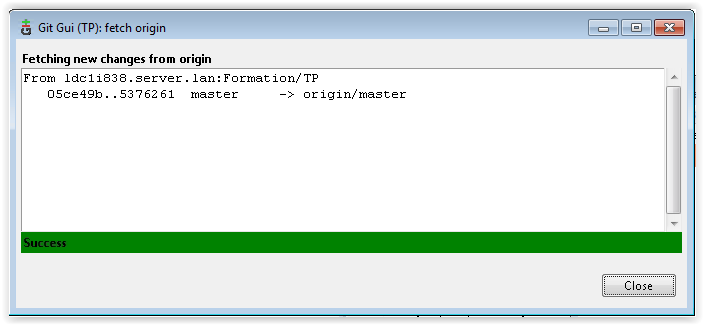


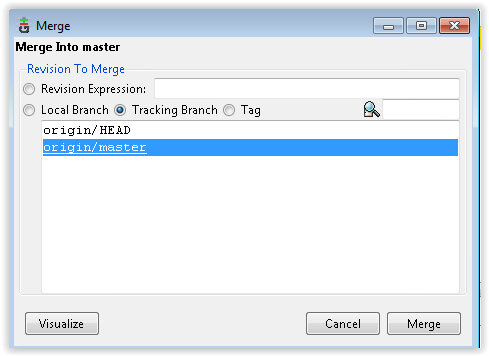


1

2

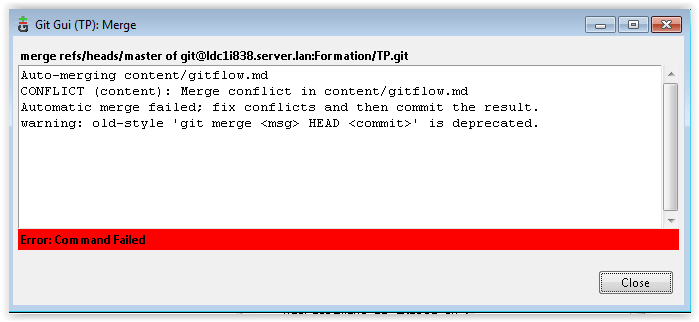
3

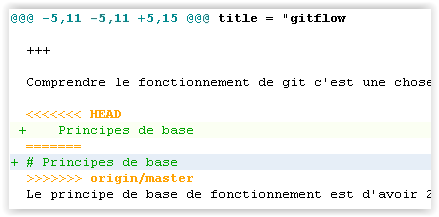




4

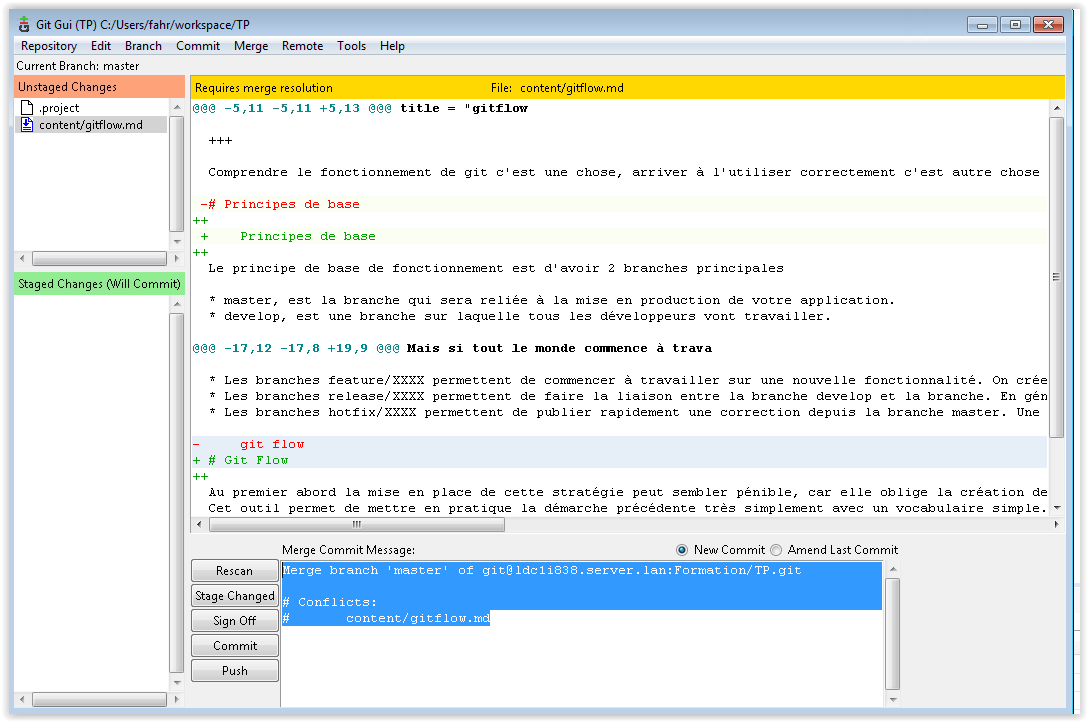
5



Nous retrouvons ensuite les balises de merge : 

Depuis l’éditeur Eclipse il faut résoudre manuellement le conflit indiqué par les balises de merge.

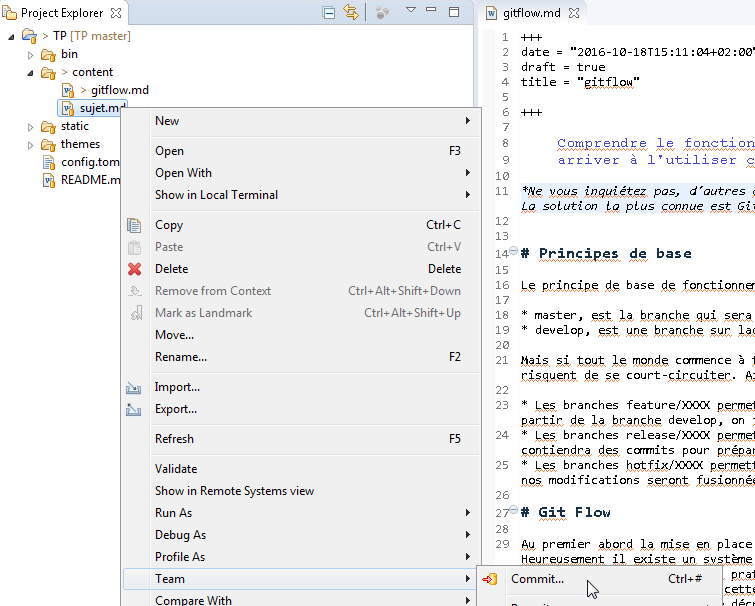
Ensuite, refaites le commit et le push depuis Git Gui :



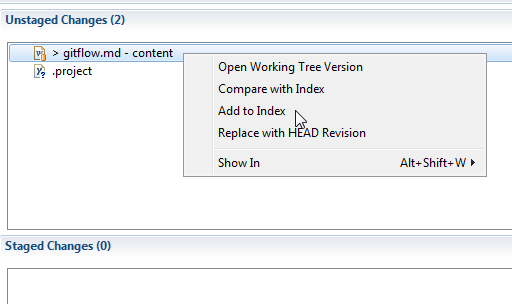
Le message du commit est généré automatiquement. Vous pouvez le modifier, si nécessaire.

#### Utilisation du plugin Eclipse pour la résolution des conflits

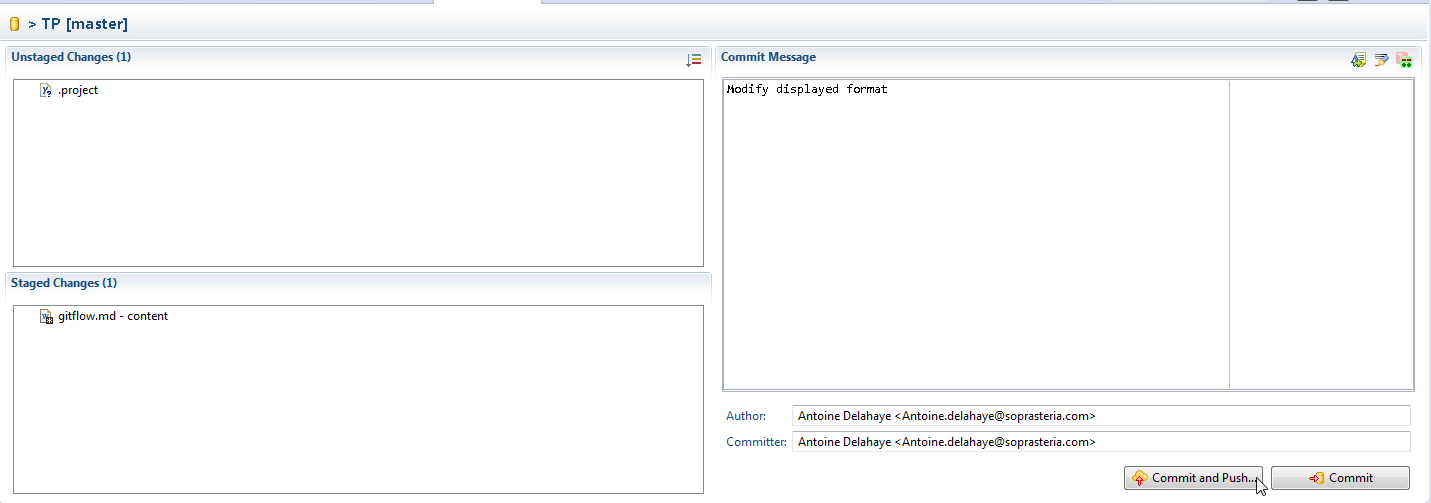
Une fois vos modifications effectuées, faites un commit :

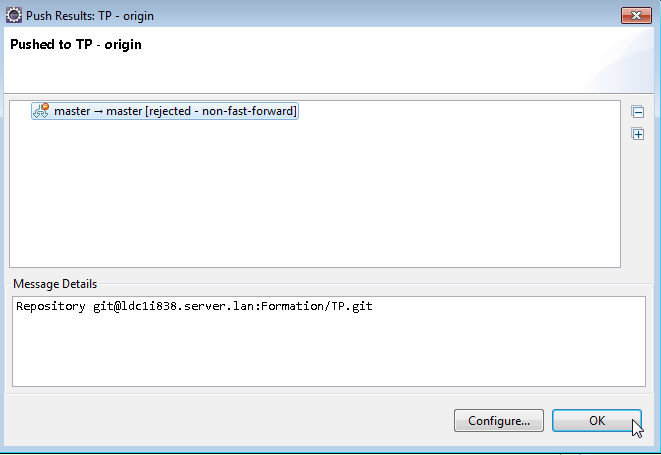


Ajoutez votre article à l’index avec l’option “Add to Index” (disponible par un clic droit sur le fichier) :



Saisissez un message pour le commit en cours. Vous pouvez utiliser l’une ou l’autre des options qui vous sont proposées (« Commit and Push… » ou « Commit »)

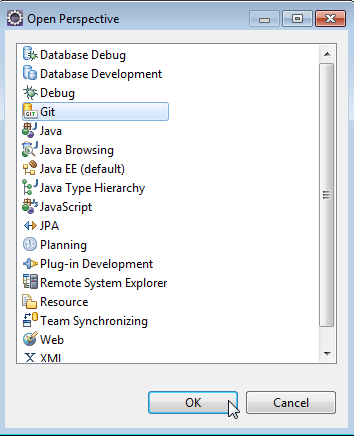


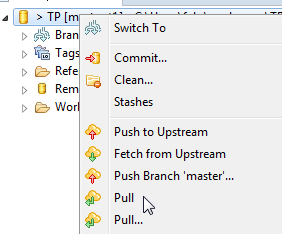


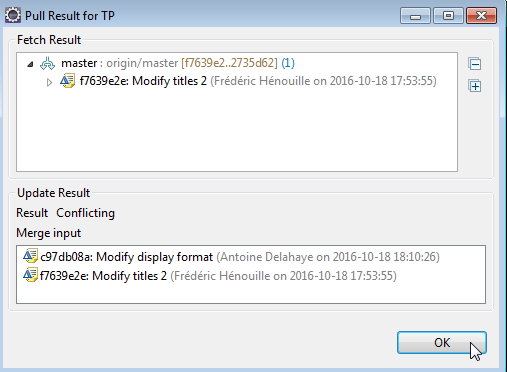
Dans notre cas nous avons commité et poussé en même temps.

Nous constatons qu’il y a une erreur …

Pour résoudre ce problème, ouvrez la perspective Git pour pouvoir récupérer les modifications présentes sur le serveur.



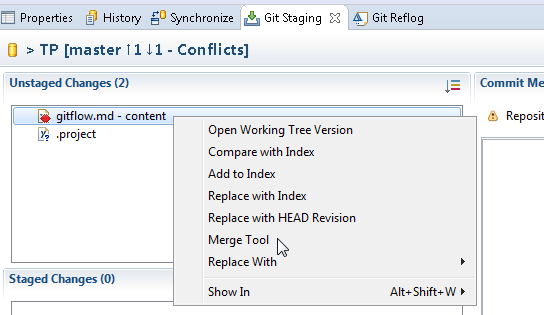




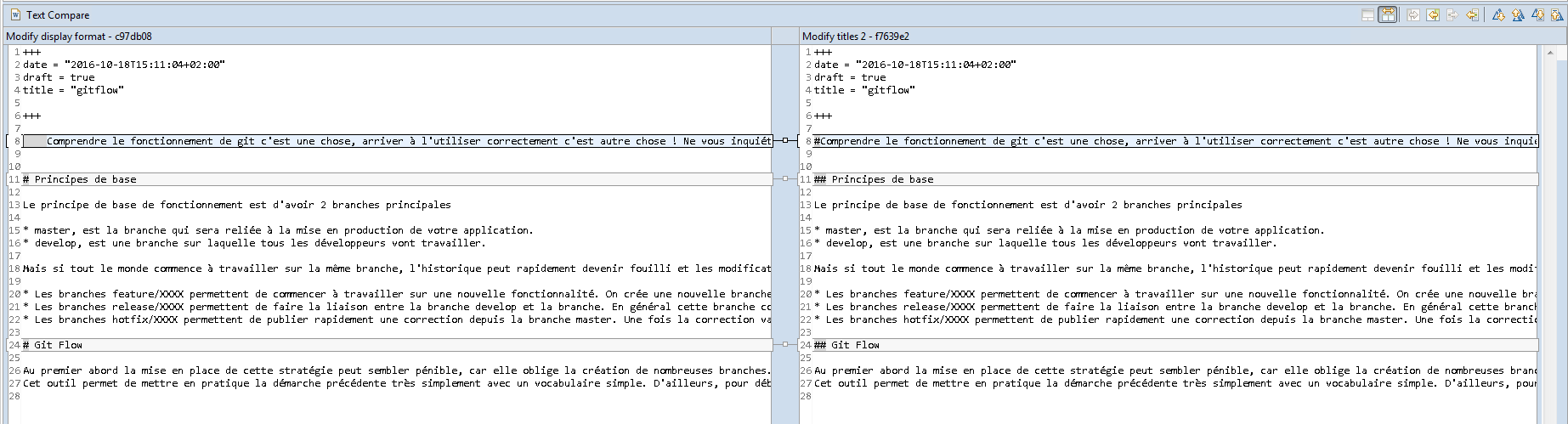
Modifications présentes sur le serveur posant problème

Tentative de résolution du problème en échec.

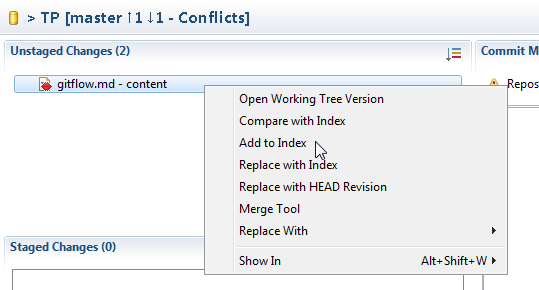
Eclipse dispose d’un outil d’aide à la résolution de conflits.

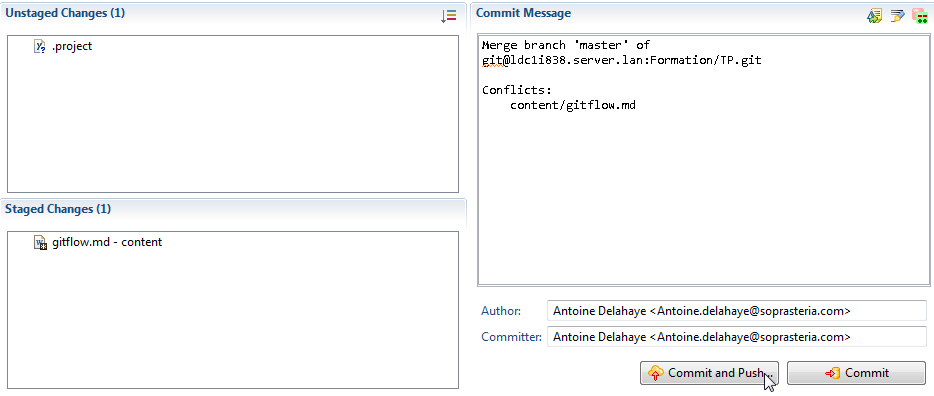


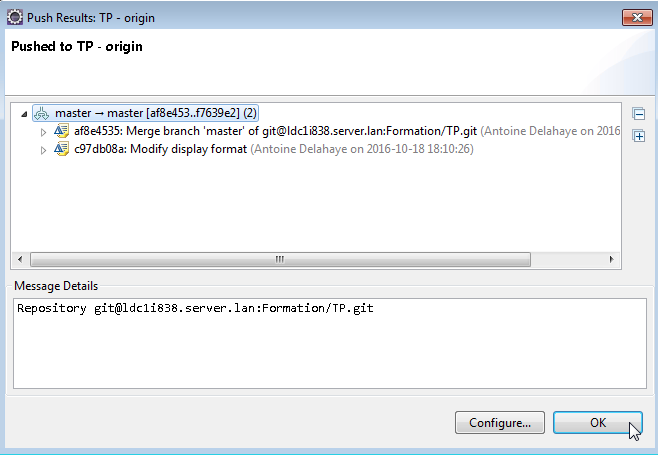
Il est possible de visualiser les différences entre 2 versions et d’utiliser les flèches en haut à droite pour appliquer certaines parties des modifications.



Une fois la fusion effectuée, refaire un commit:





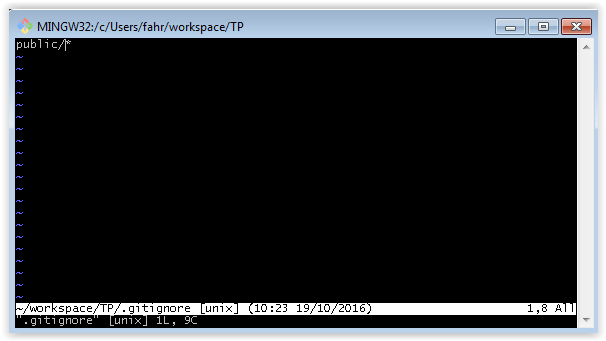


## TP 03 – Ignorer des fichiers

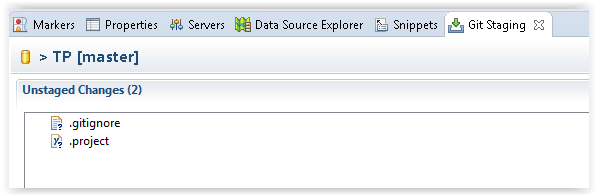
Relancez Hugo avec la commande `./bin/hugo.exe server --buildDrafts -w -d public`

Lorsque vous faites un « git status » pour voir vos fichiers indexés et non indexés, Git vous propose des fichiers dont vous ne voulez pas dans votre index ! (dossier public/)

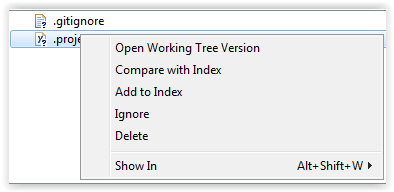
Pour pallier à ce problème, il vous suffit de modifier le fichier « .gitignore » (à la racine de votre projet) en ajoutant le dossier à exclure (S’il n’est pas présent, créez le avec la commande « touch .gitignore »).

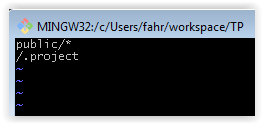


Lorsque vous retournez sur la « staging area » d’Eclipse, vous pouvez voir que seul le « .project » et « .gitignore » sont présents.



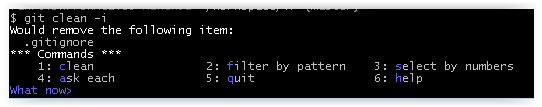
Vous pouvez aussi ignorer les fichiers directement depuis cette zone, et vérifier le fichier .gitignore.





La commande ‘git clean’ efface de l’espace de travail tous les fichiers « untracked » et non ignorés.

L’option « -i » permet de visualiser les effets de la commande « clean » sans l’appliquer.



## TP 04 – Manipulation des zones de travail Git (mv, reset, rm)

Vous vous êtes trompé lors de la création de l’article sur son nom (ou vous ne l’aimez plus), renommez votre article, indexer le et faites un commit.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git mv ./content/gitflow.md ./content/git-flow.md

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git status

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: content/gitflow.md -> content/git-flow.md

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git commit –a –m “Rename gitflow”

[master 7a4a5e1] Rename gitflow

1 file changed, 0 insertions(+), 0 deletions(-)

rename content/{gitflow.md => git-flow.md} (100%)

Vous revenez sur votre décision, annulez l’action précédente.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git log

commit 7a4a5e1d0dc6a03054d1cd273f6d706558b2c433

Author: Frédéric Hénouille <frederic.henouille.ext@ag2rlamondiale.fr>

Date: Wed Oct 19 10:11:36 2016 +0200

Rename gitflow

commit af8e453595809bf77e8bbc18cb31d4d7140d0add

Merge: c97db08 f7639e2

Author: Antoine Delahaye <Antoine.delahaye@soprasteria.com>

Date: Tue Oct 18 18:21:03 2016 +0200

Merge branch 'master' of git@ldc1i838.server.lan:Formation/TP.git

Conflicts:

content/gitflow.md

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git reset --hard HEAD~

HEAD is now at af8e453 Merge branch 'master' of [git@ldc1i838.server.lan:Formation/TP.gitgit](mailto:git@ldc1i838.server.lan:Formation/TP.gitgit)

Supprimez l’article et indexez la modification.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git rm ./content/gitflow.md

rm 'content/gitflow.md'

Annulez l’action précédente.

Exemple:

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git reset --hard HEAD

HEAD is now at af8e453 Merge branch 'master' of git@ldc1i838.server.lan:Formation/TP.git

Modifiez le contenu de votre article, effectuez un commit, puis supprimer le commit sans toucher au contenu de l’index, enfin supprimez la modification de l’index.

Utilisez les commandes

git reset --soft HEAD~

git reset --mixed HEAD

## TP 05 – Manipulation des branches (branch, checkout)

Vous devez à présent créer une branche (le nom importe peu) tout en restant sur votre branche master.

Pour cela, vous devez utiliser la commande suivante :



Pour vous rendre compte de son existence, il vous suffit simplement d’afficher l’ensemble des branches présentes en local.



La branche présentant un astérisque devant son nom, est la branche courante, celle sur laquelle vous vous trouvez en ce moment.

Si vous voulez vous placer sur cette branche, il vous faut utiliser la commande « checkout ».



Vous pouvez vérifier que vous vous trouvez bien sur celle-ci juste en regardant le nom présent entre parenthèses à côté de l’emplacement ou vous êtes.



A partir de cette branche quelconque, (ou de n’importe quelle branche), vous devez créer une branch et vous placer directement dessus. (Il faut utiliser la commande « checkout » avec une option particulière !) :



Enfin, vous supprimez votre branche quelconque, elle ne vous servira pas :



Avec l’option –d (delete), la branche sera supprimée. Cependant, s’il existe des actions en cours (merge par exemple), utilisez l’option –D pour forcer la suppression !!!

## TP 06 – Fusion de branches locales et distantes

Effectuez plusieurs commits sur la branche créée précédemment.

Partager votre branche sur le serveur GitLab.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (e142)

$ git push origin e142

Counting objects: 56, done.

Compressing objects: 100% (56/56), done.

Writing objects: 100% (56/56), 6.03 KiB | 0 bytes/s, done.

Total 56 (delta 27), reused 0 (delta 0)

remote:

remote: Create merge request for e142:

remote: http://ldc1i838.server.lan/Formation/TP/merge\_requests/new?merge\_request%5Bsource\_branch%5D=e142

remote:

To ldc1i838.server.lan:Formation/TP.git

\* [new branch] e142 -> e142

Après avoir poussé votre branche de dev sur le serveur, fusionnez la branche avec votre branche master.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (e142)

$ git checkout master

Switched to branch 'master'

Your branch is up-to-date with 'origin/master'.

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git merge e142

Updating af8e453..8ec290e

Fast-forward

content/gitflow.md | 2 ++

1 file changed, 2 insertions(+)

Récupérez les branches poussées par les voisins sur le serveur GitLab.

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git ls-remote --heads

From git@ldc1i838.server.lan:Formation/TP.git

47100f17fef2da0907089342267b0290a8e4de79 refs/heads/andl

8ec290ea714230c7bffa64aec5977cd088ad864b refs/heads/e142

490d58d7c73672fdffbc5ece3409380c2045c5a5 refs/heads/master

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git fetch origin

remote: Counting objects: 21, done.

remote: Compressing objects: 100% (21/21), done.

remote: Total 21 (delta 7), reused 6 (delta 0)

Unpacking objects: 100% (21/21), done.

From ldc1i838.server.lan:Formation/TP

af8e453..490d58d master -> origin/master

\* [new branch] andl -> origin/andl

\* [new tag] V1.0.1 -> V1.0.1

\* [new tag] V1.1 -> V1.1

Vous êtes satisfait du travail que vous avez fourni sur votre branche master locale, vous pouvez la fusionner avec la branche master (serveur).

Exemple :

e142@VDW7FORMGIT02 MINGW32 ~/workspace\_git/TP (master)

$ git push

Counting objects: 4, done.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 657 bytes | 0 bytes/s, done.

Total 4 (delta 1), reused 0 (delta 0)

To ldc1i838.server.lan:Formation/TP.git

f69c6d0..68dee9b master -> master

## TP 07 – Tags : Manipulation et partage (tag)

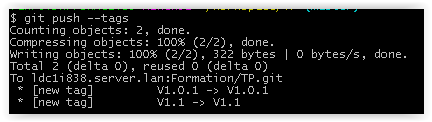
Placez un tag sur un commit de votre choix.



Par défaut (i.e sans préciser un id de commit), le tag va se positionner sur le dernier commit.

Il faut ajouter l’option « -a » pour annoter le tag et ne pas avoir un tag léger.

Pour partager les tags, il s’agit tout simplement d’un push, en précisant ce que l’on veut pousser. (Seuls les tags seront poussés avec cette commande)



Afin de récupérer les tags, utilisez ‘git fetch’.



En ce qui concerne la suppression de tag , il suffit d’utiliser l’option « -d » (delete) :



Concernant les tags, il existe une convention de nommage.

Règle de nommage à respecter pour Hawaii : vX.Y.Z.SUFFIXE

Exemples :

v4.5.67

v2.2

v2.3.4.RC1

v2.3.4.RELEASE

## TP 08 – Visualiser l’historique (blame, diff, log)

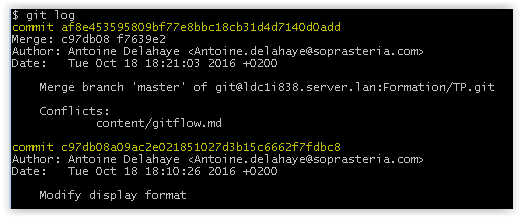
Dans ce TP, il faudra afficher l’historique des commit, afficher la différence entre deux commits et, identifier l’auteur et le commit de provenance d’une modification (sur un fichier > article en l’occurrence).

#### Utilisation de Git Bash

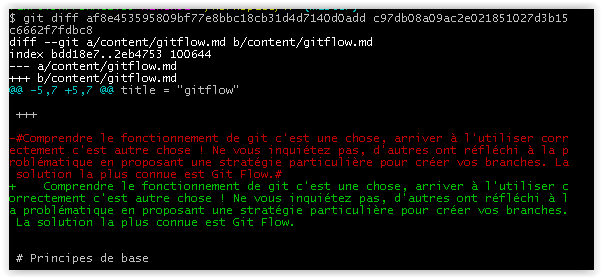
Utilisez la commande « git log ».

Vous pourrez ainsi voir l’ensemble des commits, leur id, leur auteur, leur date et le commentaire (message).

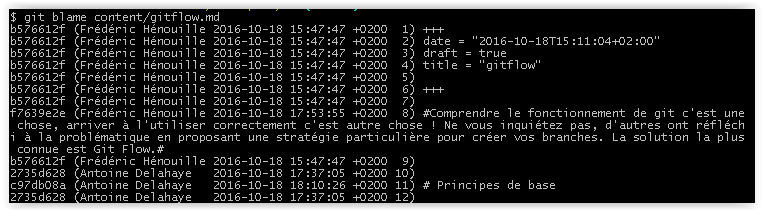
Pour avoir un historique en ASCII art, utilisez les paramètres suivants « ***--oneline --decorate --graph*** »



Pour afficher la différence entre deux commits, utilisez la commande « git diff » avec en option les deux id de commit à comparer (On peut aussi utiliser les tags, ou mixer tag/id).

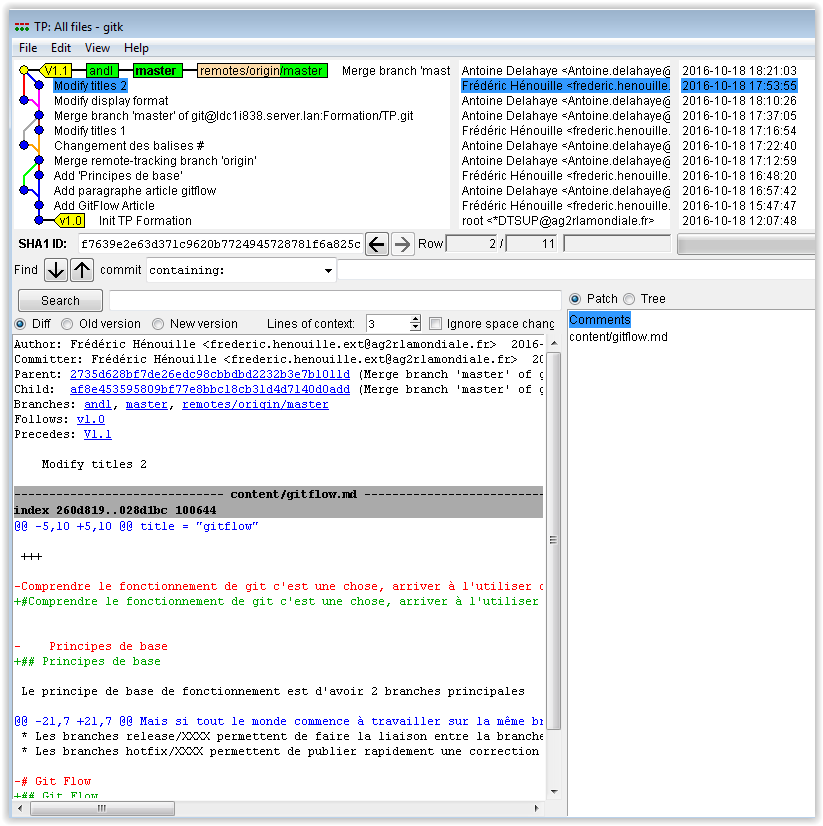


Pour voir l’auteur des modifications sur chaque ligne d’un fichier, ainsi que le commit de provenance, utilisez la commande ‘git blame file-name’.



#### Utilisation de gitk

Dès que l’on lance gitk, la fenêtre ci-dessous apparaît, et avec elle, toutes les informations de l’historique.



Id du commit sélectionné

Auteur et date du commit

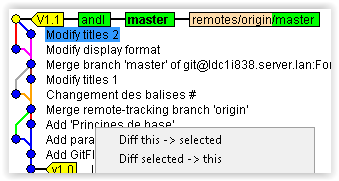
Message du commit

Zone d’affichage des modifications par rapport à la version précédente

Fichier impacté

Si vous voulez voir la différence entre deux commits qui se suivent, il suffit simplement de sélectionner le commit le plus récent des deux. (Par défaut, gitk fait la différence avec le commit précédent).

Si vous voulez voir la différence entre deux commits qui ne se suivent pas, sélectionnez en un (le plus récent par exemple), et faites un clic droit sur le second.

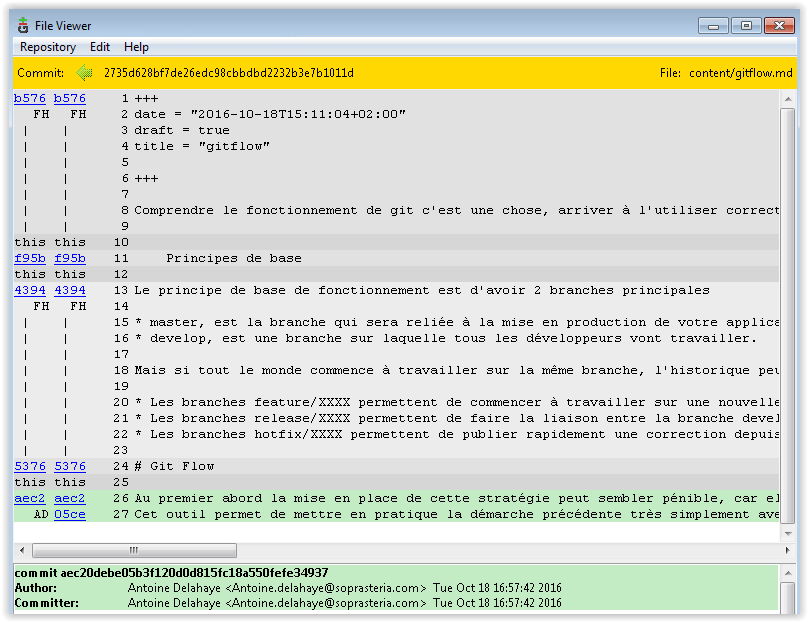


L’affichage des modifications se fait dans la zone d’affichage vu précédemment.

Pour voir en détail ligne par ligne un fichier, faites un clic droit sur celui-ci et sélectionnez l’option « Blame parent commit ».



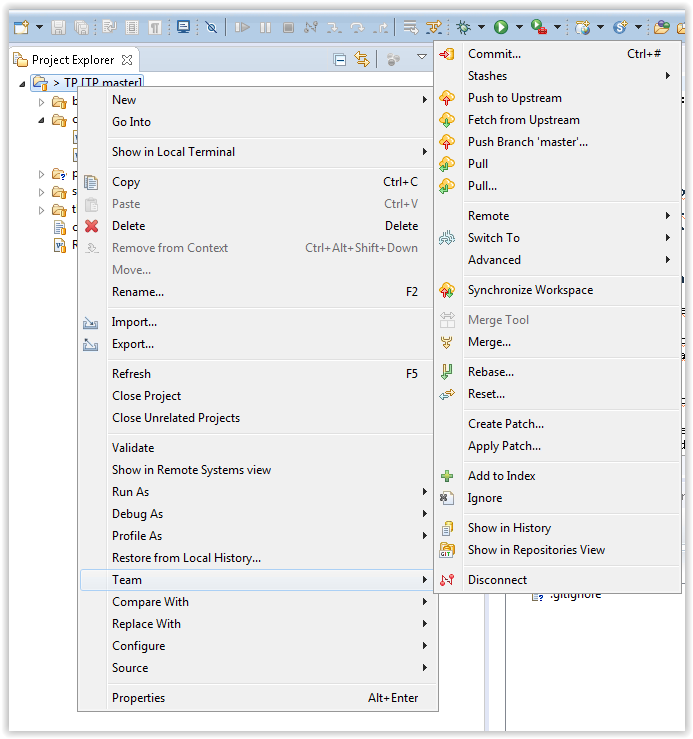
La fenêtre suivante apparaît, en surlignant des groupes de lignes (qui correspondent au même commit)



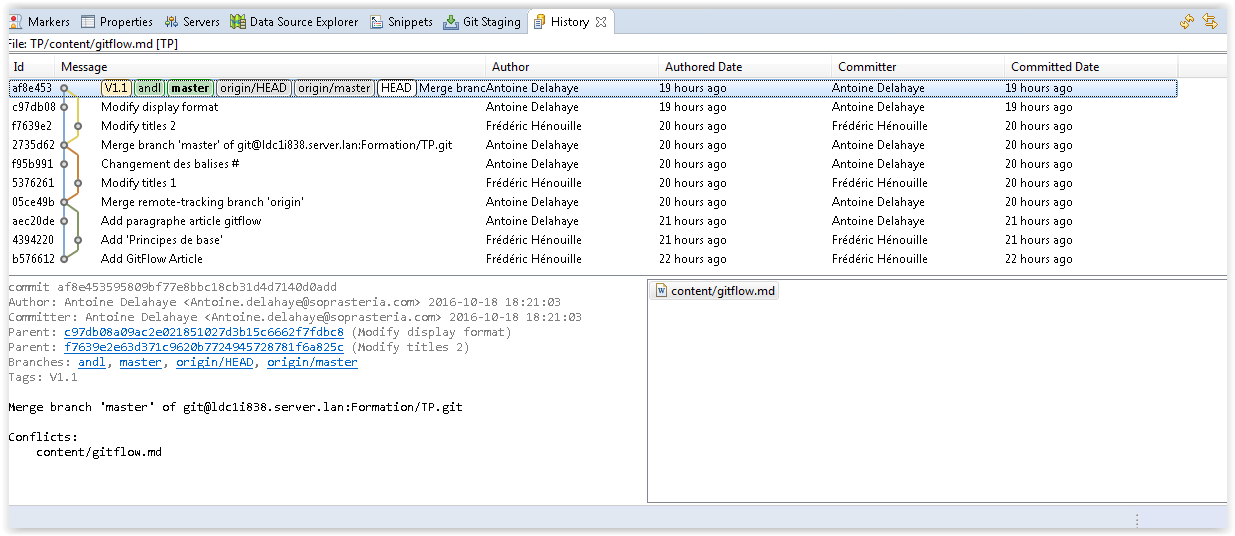
Il suffit de cliquer sur une ligne ou un groupement, pour avoir les informations de provenance

#### Utilisation du plugin eclipse

Sous Eclipse, pour avoir accès à l’historique ,faites un clic droit sur votre projet, puis dans l’option « Team », choisissez « Show in History ».



La fenêtre suivante apparaît (onglet « History »)



Refresh

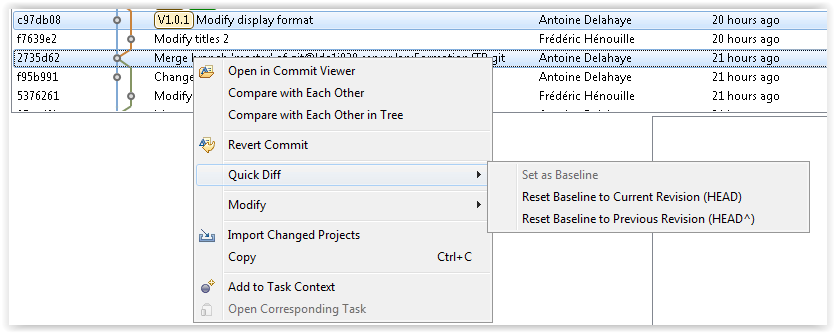
Id du commit

Message du commit

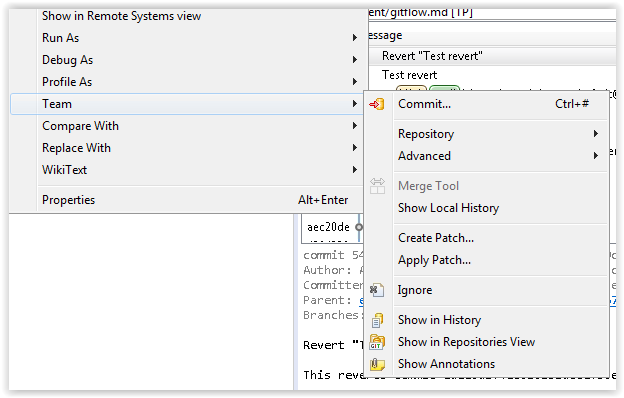
Auteur du commit

Fichier impacté

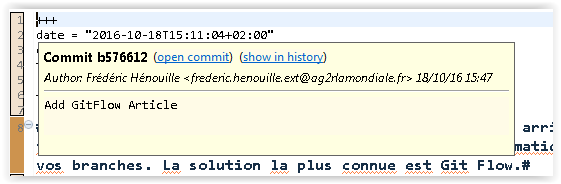
Zone regroupant les informations du commit



Pour voir les informations ligne par ligne d’un fichier, il faut utiliser l’option « Show Annotations » du menu « team ».



La vue change désormais afin d’afficher, dans l’éditeur de texte, des annotations pour chaque ligne (ou groupement). On y trouve l’id du commit (possibilité d’ouvrir le commit, ou de le voir dans l’historique), l’auteur, la date/heure, et le message.

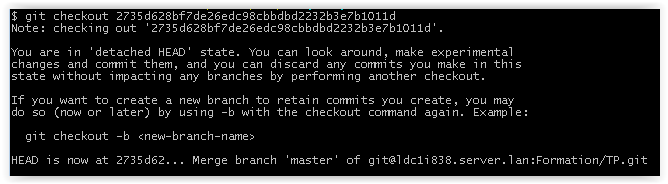


## TP 09 – Remonter le temps

Pour se positionner dans sa branche, il faut utiliser la commande « checkout » vue au TP 5.



Pour repositionner le HEAD dans un état antérieur, il faut connaître l’id du commit (visible via la commande « git log » vue au TP 4) ou le nom d’un tag (via la commande « git tag » vue au TP 7).



Lorsque l’on revient sur un commit précédent, on se retrouve avec une tête qui n’est attachée à aucune branche. Il faut alors créer une branche si l’on veut garder une trace des futurs commits sinon vous obtiendrez le message suivant lors du retour à votre branche :



Sur une branche autre que master (en recréer une si besoin), effectuez plusieurs commits, sans push.

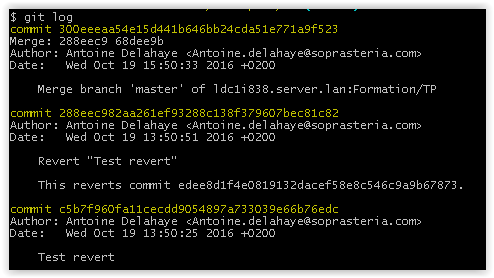
Repérez un commit parmi ceux que vous avez effectués sur la branche.

Replacez-vous sur votre master et lancez la commande suivante :



En regardant dans les logs, vous vous apercevez que le commit en question apparait désormais dans votre master.

Si on veut ‘revenir en arrière’ sur le commit c5b7f9…, il faut utiliser la commande « git revert HEAD~2 » car il y a deux commits au-dessus de celui sur lequel on veut retourner. Il va ainsi appliquer les deux patchs inverses et nous donner la possibilité de créer un commit comportant ce ‘retour arrière’.



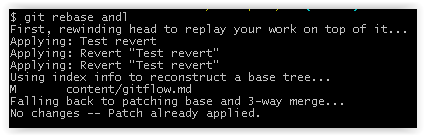


## TP 10 – Rebase de branche

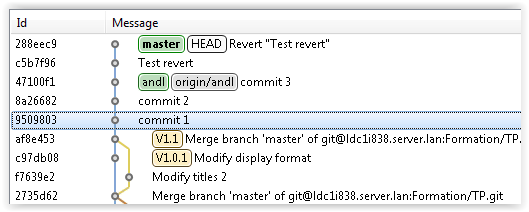
Créez une nouvelle branche (feat-login) avec les commandes vues au TP 5.

Effectuez des modifications et plusieurs commits. Observez via gitk ou le plugin Eclipse, que cette branche est bien distincte de la branche master.

Maintenant, vous pouvez faire un « git rebase feat-login» sur votre master local afin de n’avoir aucune trace de cette branche dans le futur.



Une fois la commande exécutée, vous pouvez retourner voir l’historique, que ce soit avec gitk, ou le plugin Eclipse. Vous constatez donc un historique simplifié entre les deux branches impliquées dans le rebase.



* Avant le rebase, plusieurs branches étaient visibles sur cette partie.
* Maintenant, une seule branche est visible sur celle-ci

## TP 11 – Mettre de côté ses développements

Basculez sur votre branche de développement.

Effectuez des modifications sans les remonter dans l’index

Mettez de côté votre travail

Basculez sur la branche master.

Effectuez un nouveau commit sur master.

Basculez à nouveau sur votre branche de développement.

Reprenez le travail mis de côté et faites un commit.

Commandes à utiliser : ‘git stash’ et ‘git stash pop’

## TP 12 – Réécrire l’histoire !

Fusionnez 2 commits en un seul.

Exemple : Fusion des 2 commits suivants

commit d63b85333d4105278dd303903483e97af702e5f2

Author: HENOUILLE Frédéric (EXTERNE) <E142@LV0082.lamondiale.com>

Date: Wed Oct 19 17:48:58 2016 +0200

master 2

commit c283911fadd33e74bbe98e8e4dd29648a70e259b

Author: HENOUILLE Frédéric (EXTERNE) <E142@LV0082.lamondiale.com>

Date: Wed Oct 19 17:43:23 2016 +0200

master 1

Il faut effectuer une commande rebase –i à partir du parent du commit le plus ancien à retoucher. Exemple :

E142@LV0082 MINGW32 /c/workspace/migration-git (master)

$ git rebase -i HEAD~3

Un éditeur de type vi permet alors de choisir l’action à effectuer pour les commits concernés par le rebase.

Pour fusionner 2 commits, on applique le mot-clé squash devant le commit le plus récent des 2.

On obtient ensuite un seul commit :

commit 052c2d6c5c316b1449062cc597d7eebe76015a57

Author: HENOUILLE Frédéric (EXTERNE) <E142@LV0082.lamondiale.com>

Date: Wed Oct 19 17:43:23 2016 +0200

master 1

master 2

Pour changer le commentaire de ce commit, réutilisez la commande ‘rebase –i’.

E142@LV0082 MINGW32 /c/workspace/migration-git (master)

$ git rebase -i HEAD~2

Edition vi : utiliser le mot-clé reword devant le commit dont le message est à retoucher

reword e052c2d6 master 1

pick 2e5f295 stash pop

# Rebase 2782997..2e5f295 onto 2782997 (2 commands)

#

# Commands:

# p, pick = use commit

# r, reword = use commit, but edit the commit message

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

# d, drop = remove commit

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

~

<ation-git/.git/rebase-merge/git-rebase-todo[+] [unix] (18:31 19/10/2016)1,8 All

**-- INSERT --**

Nouveau commentaire sur le commit pris en compte :

commit 20ca6227155e5b286f3c1463eb9db5b8fc46a4e4

Author: HENOUILLE Frédéric (EXTERNE) <E142@LV0082.lamondiale.com>

Date: Wed Oct 19 17:43:23 2016 +0200

master 1 et 2

# ANNEXE

## GIT

### Commandes GIT

Toutes les commandes git sont de la forme suivante :

git action [options] argument

Add Ajoute un chier dans l’index pour préparer un commit.

git add chemin/vers/fichier

branch Crée un branche de travail

git branch ma-branch

checkout Change la version de travail par une branche ou les fichiers du dernier commit

git checkout ma-branch

cherry-pick Permet de prendre un commit quelconque et de l’appliquer localement.

git cherry-pick ma-branche Permet d’appliquer le dernier commit de la branche «ma-branche» dans la branche courante.

clean Supprime les fichiers non suivis par git

git clean -f

clone Prépare une copie de travail depuis un serveur distant

git clone URL .

commit Valide l’index actuel pour créer un commit

git commit -m "Mon message de commit"

config Permet de configurer les variables de paramétrage de git L’option global permet de configurer les variables de l’utilisateur, donc valable pour l’ensemble des projets de l’utilisateur. L’option system permet de configurer les variables de tous les utilisateurs.

git config --global user.name "John Doe"

diff Met en évidence la différence entre 2 commits.

git diff

init Initialise un répertoire de travail comme un dépôt local git. Permet d’initialiser l’ensemble des 3 espaces. Exemple pour le répertoire courant :

git init .

log Observe l’historique des changements

git log

merge Fusionne 2 branches.

git merge ma-branche

mv Déplace un fichier. C’est équivalent à copier le fichier puis supprimer le fichier d’origine.

git mv chemin/vers/fichier chemin/vers/nouveau/fichier

pull Permet de recevoir une copie de l’index d’un serveur distant et de le fusionner avec l’historique local.

git pull origin master

push Permet d’envoyer une copie de l’index des changements vers un serveur distant

git push origin master

rebase Concatène 2 branches.

git rebase ma-branche

remote Permet de manipuler les adresses des serveurs distants.

git remote add origin git@git-prd.server.lan:Formation/TP.git

revert Crée un patch de retour en arrière afin d’annuler certains commits.

git revert HEAD~3 Va créer un patch pour annuler les 4 derniers commits.

rm Supprime un fichier de l’index

git rm chemin/vers/fichier

status Observe l’état de l’index et du répertoire de travail

git status

tag Permet de poser une étiquette sur un commit donné.

git tag 1.0.0

#### Associer un éditeur de texte à Git (saisie des commentaires)

git config core.editor Associe un éditeur de texte à Git (saisie des commentaires)

git config --global core.editor

'C:/Program Files (x86)/Notepad++/notepad++.exe'

### Fichiers particulier

.gitignore Liste des fichiers ou répertoires que git doit ignorer

## HUGO

### Qu’est-ce qu’Hugo ?

Hugo est un générateur de site statique. Contrairement aux autres générateurs de site qui construisent une page à chaque fois qu’un utilisateur en demande une, Hugo crée toutes les pages à la création du contenu. Hugo est optimisé pour les sites web dédié à la lecture (contrairement à l’édition).

Hugo permet la création de site web très rapide et sécurisé. Les sites web Hugo peuvent être hébergés partout (GitHub, GitLab, Google Cloud Storage, DreamHost…). Avec un temps de constructino (build) quasi instantané et l’habilité à reconstruire (rebuild) à chaque fois qu’un changement est fait, Hugo fournit un retour très rapide.

### Quelques commandes

A la racine du votre projet hugo (projet TP en l'occurrence) :

* Création d’un article
  + « ./bin/hugo.exe new nom-de-l-article.md***»***
  + Lorsque l’article est créé, il ne s’agit pas d’un simple fichier vide. Hugo va créer l’entête correspondant avec par exemple, l’attribut correspondant au <title>
* Lancer le serveur de dev
  + ***«***./bin/hugo.exe server --buildDrafts –w***»***
  + Lorsque l’on lance cette commande, le serveur http va être généré et va détecter chaque changement de contenu pour régénérer les pages. ( attribut w -> watch)

## MARKDOWN

### Qu’est-ce que le MarkDown ?

Il s’agit d’un langage de balisage léger. Il a pour but d’offrir une syntaxe facilement compréhensible et facile à écrire.

### Quelques commandes

Commande Markdown commande HTML

# Titre 1 <h1>Titre 1</h1>

## Sous Titre 1 <h2>Sous Titre 1</h2>

\* Liste à puce, item 1 <ul> Liste à puce, item 1 </ul>

\* Liste à puce, item 2 <ul> Liste à puce, item 2 </ul>

\*Italique\* <strong>Italique</strong>

\*\*Gras\*\* <i>Gras</i>

\*\*\*Gras et Italique\*\*\* <strong><i>Gras et Italique</i></strong>

Du code <code>du code</code>

Il ne faut pas oublier les 4 espace devant votre code, sinon celui-ci ne sera pas considéré comme faisant partie d’une balise code.



Ne pas oublier l'espace aprés la balise markdown de liste, **sinon** le texte sera *italique*!