

# Essentiel GIT

## Table des matières

<b>I - Présentation de GIT .....</b>	<b>4</b>
1. SVN/Subversion "centralisé" avant GIT .....	4
2. Présentation de GIT .....	4
2.1. <i>Fonctionnement distribué de GIT</i> .....	5
2.2. <i>Hébergement de référentiel GIT</i> .....	7
2.3. <i>Installation et configuration minimum :</i> .....	8
2.4. <i>GIT , Tortoise-GIT et intégration IDE</i> .....	9
<b>II - Bases locales de GIT .....</b>	<b>10</b>
1. Principales commandes de GIT (en mode local) .....	10
2. initialisation d'un projet GIT (en local) .....	12
3. Index (staging area) .....	13
3.1. <i>fichiers indexés par GIT</i> .....	13
3.2. <i>cycle des mises à jour</i> .....	14
4. indispensable <code>.gitignore</code> .....	15

5. Commit et tags.....	16
<b>III - Gestion des branches / GIT.....</b>	<b>19</b>
1. Gestion des branches avec GIT.....	19
2. Bonnes pratiques dans la gestion des branches.....	21
3. Merge.....	22
3.1. merge rapide (fast-forward).....	22
3.2. merge sans conflits.....	23
3.3. merge avec conflits.....	24
4. Git rebase (souvent sur branche privée).....	26
<b>IV - GIT en mode distant (--bare , github, ...). ....</b>	<b>27</b>
1. Commandes de GIT pour le mode distant.....	27
2. Gérer plusieurs référentiels distants.....	28
3. initialiser git en mode distant.....	30
4. pistage (track) entre branches locales et distantes.....	31
5. Git fetch et git pull.....	33
<b>V - Git-flow et aspects divers.....</b>	<b>35</b>
1. Pièges de GIT.....	35
1.1. Passwords oubliés (ou anciens passwords mémorisés).....	35
1.2. Mauvaise pratique : password en clair dans URL.....	35
2. Quelques workflows pour GIT.....	35
2.1. git-flow (évolué).....	35
2.2. github-flow ( simple).....	36
<b>VI - Annexe – GIT avec eclipse.....</b>	<b>39</b>
1. Plugin eclipse pour GIT (EGIT).....	39
1.1. Actions basiques (commit , checkout , pull , push).....	39
1.2. Résolution de conflits.....	39
<b>VII - Annexe – Accès distant via http (conf, admin).....</b>	<b>40</b>
1. Configuration d'accès distant à un référentiel Git.....	40
1.1. Accès distant (non sécurisé) via git.....	40
1.2. Accès distant sécurisé via git+ssh.....	40
1.3. Accès distant en lecture seule via http (sans webdav).....	40
1.4. Accès distant en lecture/browsing via gitweb.....	41
1.5. Accès distant "rw" via http/https (webdav).....	42
<b>VIII - Annexe – Bibliographie, Liens WEB + TP.....</b>	<b>44</b>

---

1. Bibliographie et liens vers sites "internet".....	44
2. TP.....	44

# I - Présentation de GIT

## 1. SVN/Subversion "centralisé" avant GIT

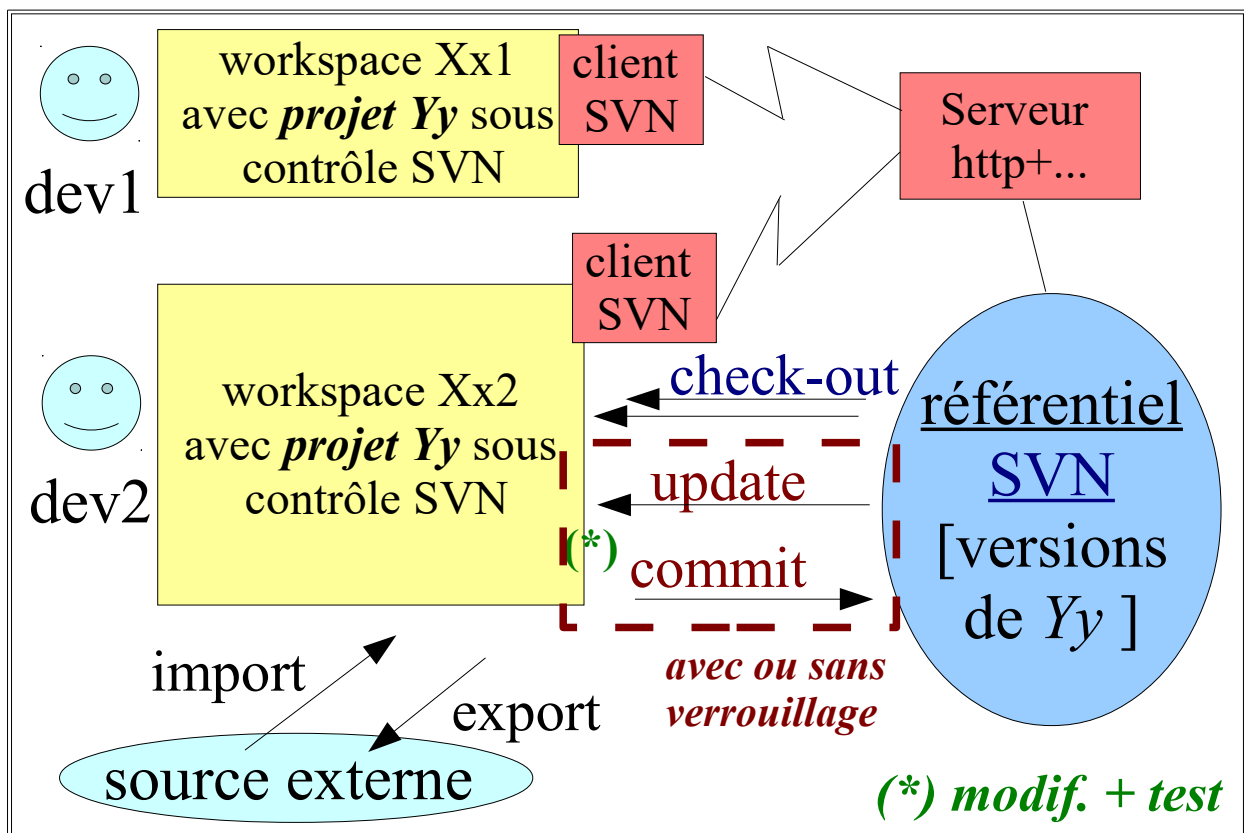
CVS = Concurrent Version System , SVN = SubVersion

CVS est un produit "Open Source" qui permet de **gérer différentes versions d'un ensemble de fichiers sources** lié au développement d'un certain module logiciel.

Différents programmeurs peuvent travailler en équipe sur des fichiers partagés au niveau d'un référentiel commun.

SVN se veut avant tout être une "*version améliorée de CVS*" qui

- ne remet pas en cause les principes fondamentaux de CVS (référentiel commun et commit,update,... )
- a refondu l'implémentation du serveur et des référentiels (meilleure gestion des transactions, protocole d'accès plus simples, ...)



Beaucoup utilisé avant GIT , SVN est/était un système de gestion de code source **centralisé** .

## 2. Présentation de GIT

## 2.1. Fonctionnement distribué de GIT

### Présentation de GIT

**GIT** est un **système de gestion du code source** (avec prise en charge des différentes **versions**) qui fonctionne en **mode distribué** .

GIT est moins centralisé que SVN . Il existe deux niveaux de référentiel GIT (local et distant).

Un référentiel GIT est plus compact qu'un référentiel SVN.

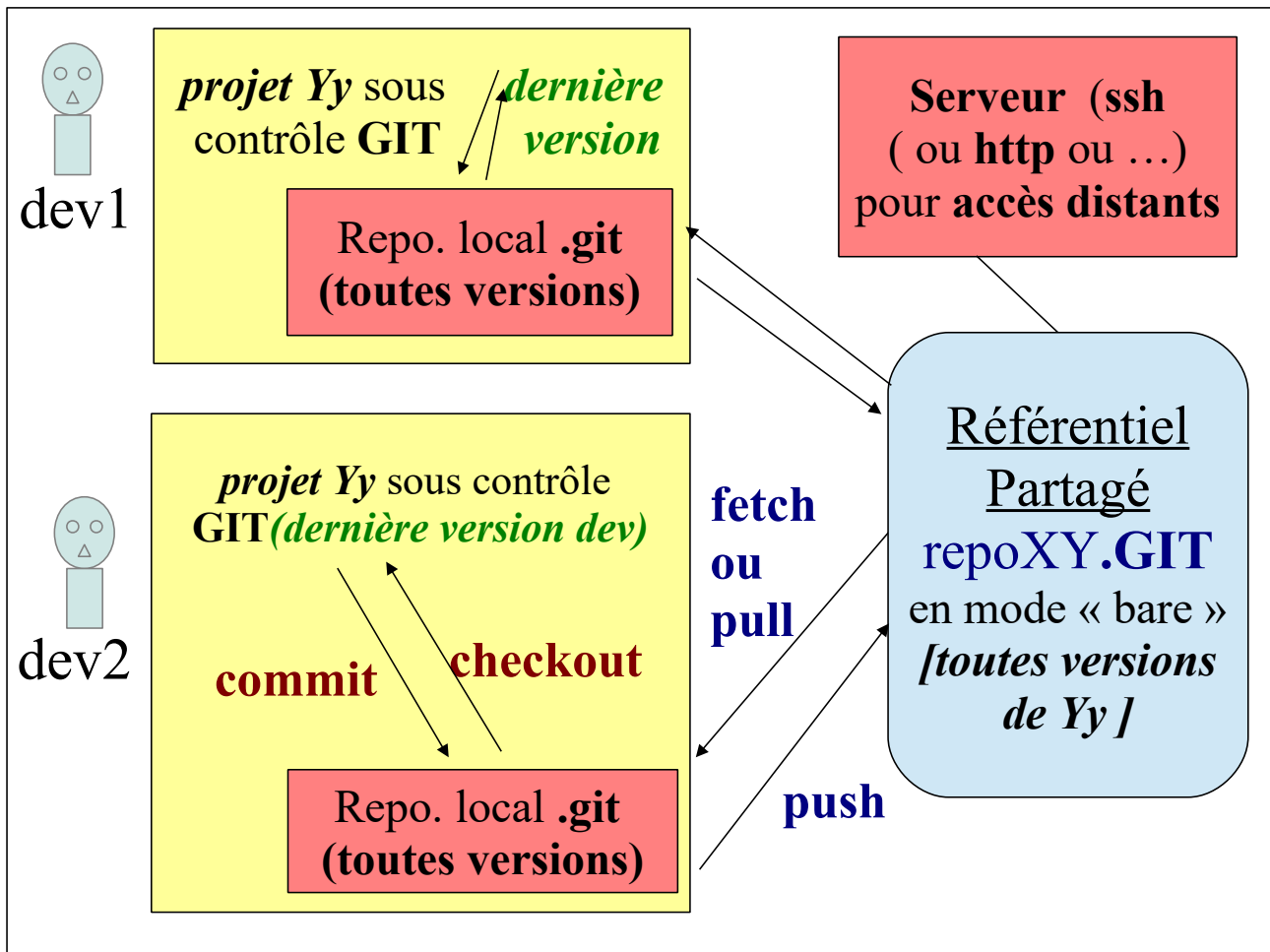
**GIT** a été conçu par **Linus Torvalds** (l'inventeur de **linux**) .

Un produit concurrent de GIT s'appelle « **Mercurial** » et offre à peu près les mêmes fonctionnalités.

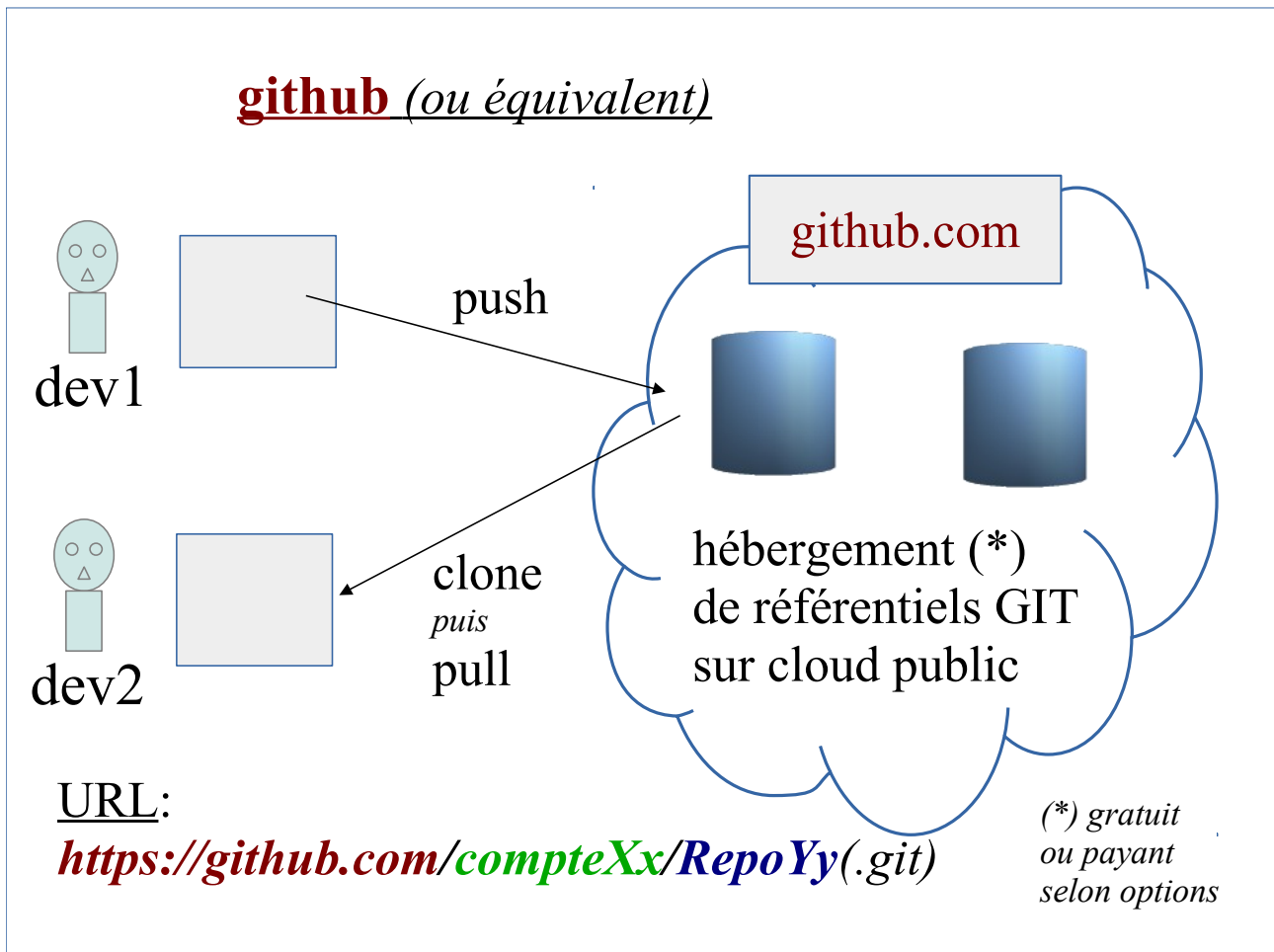
### Mode distribué de GIT

*Dans un système « scm » centralisé (tel que CVS ou SVN) , le référentiel central comporte toutes les versions des fichiers et chaque développeur n'a (en général) sur son poste que les dernières versions des fichiers.*

Dans un système « scm » distribué (tel que **GIT** ou **Mercurial**) , le référentiel central ne sert que pour échanger les modifications et chaque développeur a (potentiellement) sur son poste toutes les versions des fichiers.



## 2.2. Hébergement de référentiel GIT



Principaux sites d'hébergement de référentiels GIT :

- **github**
- **gitlab**
- **bitbucket**
- ....

## 2.3. Installation et configuration minimum :

- 1) installer "*Git for windows*" ou bien "*Git sur linux*" (via **yum** ou **apt-get** ou autre) .
- 2) **git --help**
- 3) **git config ....**

*En bref, les commandes «**commit**» et «**checkout**» de **GIT** permettent de gérer le référentiel **local** (propre à un certain développeur) et les commandes «**push**» et «**fetch / pull**» de **GIT** permettent d'effectuer des **synchronisations** avec le **référentiel partagé distant** .*

### Configuration locale de GIT

Installation de GIT sous linux (debian/ubuntu) :

```
sudo apt-get install git-core
```

Configuration locale:

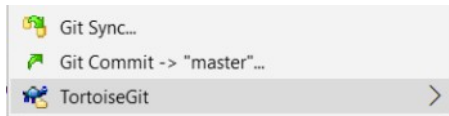
```
git config --global user.name "Nom Prénom"  
git config --global user.email "poweruser@ici_ou_la.fr"  
#...
```

```
# pour voir ce qui est configuré :  
git config --list
```

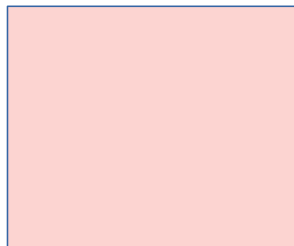


## 2.4. GIT , Tortoise-GIT et intégration IDE

### Utilisation directe ou indirecte de GIT



*Ajoute des menus contextuels à l'explorateur de fichiers*



...



*Menu "team" et perspective "GIT"*

**GIT** (en ligne de commande)

O.S. (linux ou windows ou ...)

## II - Bases locales de GIT

### 1. Principales commandes de GIT (en mode local)

Commandes GIT (locales)	Utilités
<b>git init</b>	Initialise un référentiel local git (sous répertoire caché « .git » ) au sein d'un projet neuf/originel.
<b>git clone</b> <i>url_referentiel_git</i>	Récupère une copie locale (sous le contrôle de GIT et avec toutes les versions des fichiers) d'un référentiel git existant (souvent distant)
<b>git status</b> <b>git diff</b> <i>fichier</i>	Affiche la liste des fichiers avec des changements (pas encore enregistrés par un commit) et git diff affiche les détails (lignes en + ou -) dans un certain fichier.
<b>git add</b> <i>liste_de_fichiers</i>	Ajoute un répertoire ou un fichier dans la liste des éléments qui seront pris en charge par git (lors du prochain commit).
<b>git commit</b> <i>-m message [-a]</i>	Enregistre les derniers fichiers modifiés ou ajoutés dans le référentiel git local (ceux préalablement précisés par <i>add</i> et affichés par <i>status</i> ) . si option <i>-a</i> tous les fichiers modifiés (ou supprimés) qui étaient déjà pris en charge par git seront enregistrés
<b>git checkout</b> <i>idCommit (ou branche)</i>	Récupère les (dernières ou ....) versions depuis le référentiel local
<b>git --help</b> <b>git cmde --help</b>	Obtention d'une aide (liste des commandes ou bien aide précise sur une commande)
<b>git log --stat</b> ou <b>git log -p</b>	Affiche l'historique des mises à jour -p : avec détails , --stat : résumé
<b>git branch</b> , <b>git checkout</b> <i>nomBranche</i> , <b>git merge</b>	Travailler (localement et ...) sur des branches
<b>git grep</b> <i>texte_a_rechercher</i>	Recherche la liste des fichiers contenant un texte
<b>git tag</b> <i>NomTag IdCommit</i>	Associer un <b>tag</b> parlant(ex: <b>v1.3</b> ) à un id de commit .
<b>git tag -l</b>	Visualiser la liste des tags existants
<b>git checkout</b> <b>tags</b> / <i>NomTag</i>	Récupère la version identifiée par un tag

#### Exemples :

```
#initialisation
cd p1; git init
```

#affichage des éléments non enregistrés

```
cd p1; git status
```

→ affiche:

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#       modified:   src/fl.txt
#       modified:   src/f3-renamed.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

# commit all already tracked/added :

```
cd p1
# -a pour tous les fichiers listés dans git status
git commit -a -m "my commit message"
```

#commit all (with all new and deleted) :

```
cd p1
git add pom.xml.txt src/*
git status
# git commit gère tous les fichiers ajoutés (et supprimera de l'index ceux qui
# n'existent plus si option -a)
git commit -m "my commit message" -a
```

#historique des dernières mises à jour :

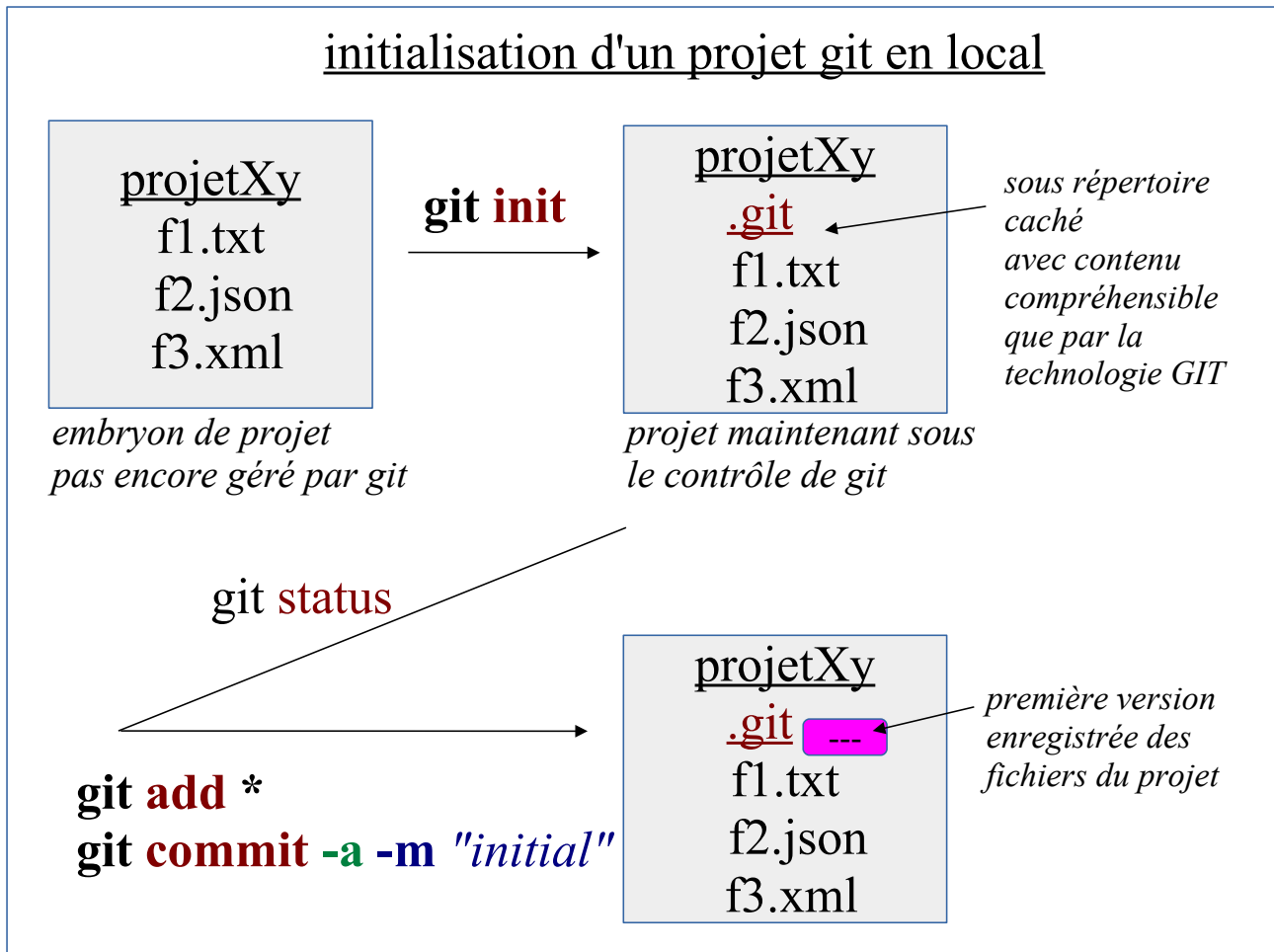
```
cd p1; git log --stat
```

---> affiche:

```
commit 93446a0f2194089d83c941a63768f212eb96e0f8
Author: developpeur fou <moi@ici_ou_la.everywhere>
Date:   Wed Dec 12 18:36:40 2012 +0100

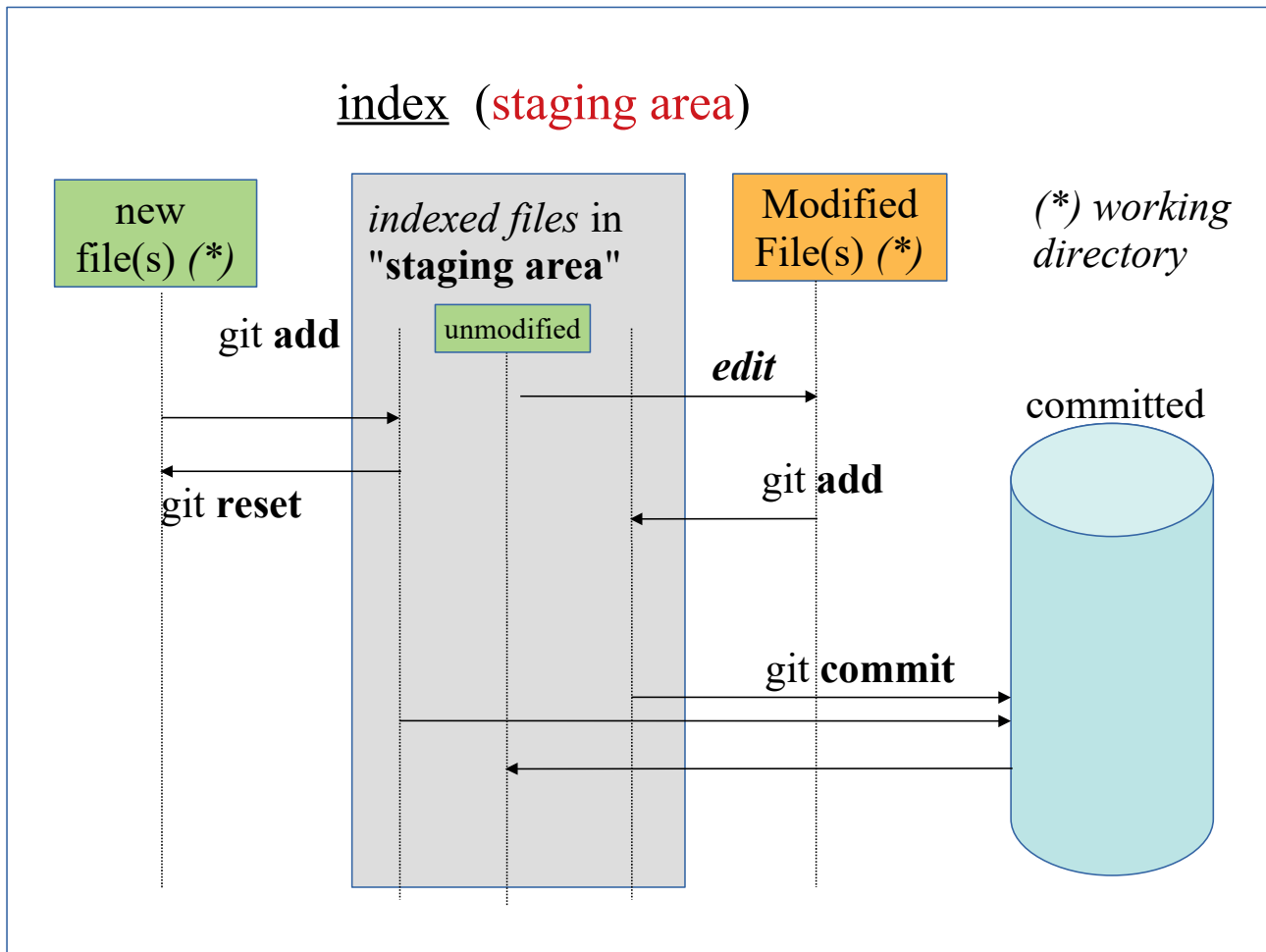
    my commit message
pom.xml.txt      | 2 ++
src/fl.txt       | 1 +
src/f3-renamed.txt | 1 +
src/f4-renamed.txt | 1 +
src/p/pf2-renamed.txt | 2 ++
5 files changed, 7 insertions(+)
```

## 2. initialisation d'un projet GIT (en local)



### 3. Index (staging area)

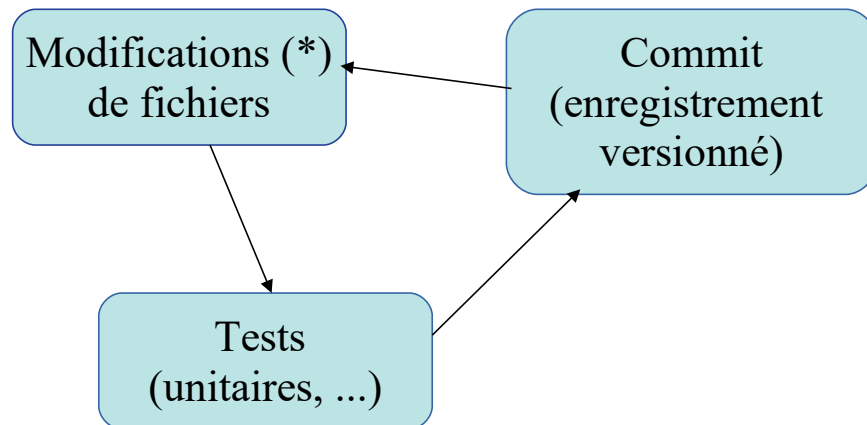
#### 3.1. fichiers indexés par GIT



Si besoin , **git reset f1.txt** permet d'annuler **git add f1.txt**

### 3.2. cycle des mises à jour

#### Cycle de mises à jour



(\*) Un fichier existant déjà lors du dernier commit et re-modifié depuis fait d'office partie de la "*staging area*" et sera par défaut re-commité. Les commandes `git add ...`, `git rm ...`, `git rm --cached ...` permettent d'ajouter ou retirer des fichiers à commiter ultérieurement avec l'option `-a`.

`git commit --amend -m nouveauMessage` permet si besoin de remplacer le message de commit

## 4. indispensable .gitignore

Le fichier caché .gitignore (à placer à la racine d'un référentiel git) est indispensable pour préciser la liste des fichiers à ne pas stocker dans le référentiel git (ex : fichiers temporaires , spécifiques à un IDE , fichiers binaires générés , ....) .

Exemple de fichier ".gitignore" pour java / maven /eclipse :

### **.gitignore**

```
target/  
*.class  
*/.settings/  
.settings/*  
.settings  
*.jar  
*.war  
*.ear
```

Exemple de fichier ".gitignore" pour npm/javascript :

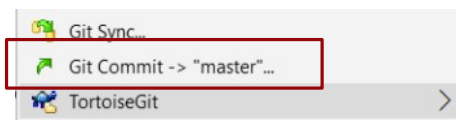
### **.gitignore**

```
node_modules  
node_modules/*  
dist/*  
dist
```

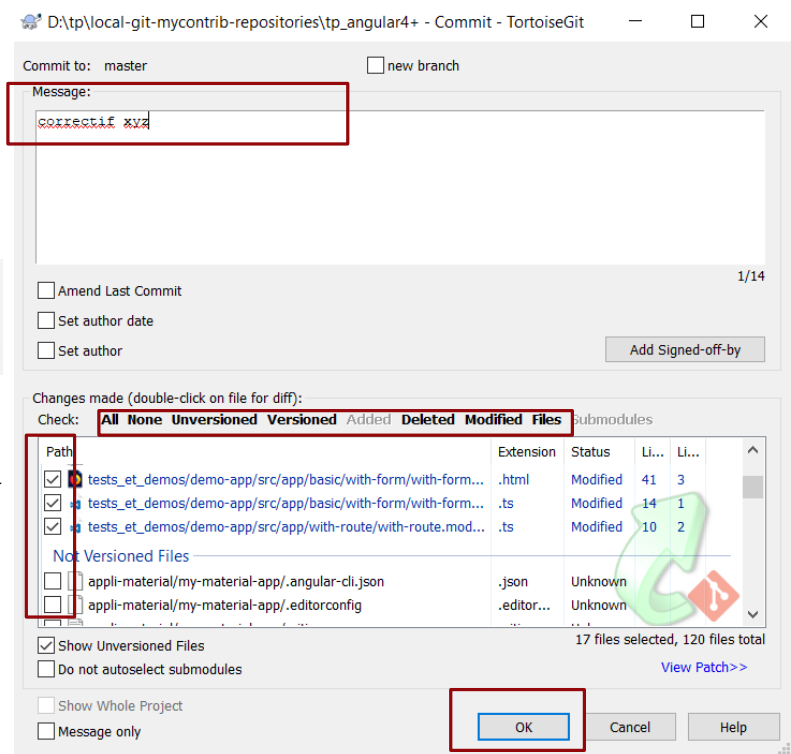
## 5. Commit et tags

### commit via tortoiseGit

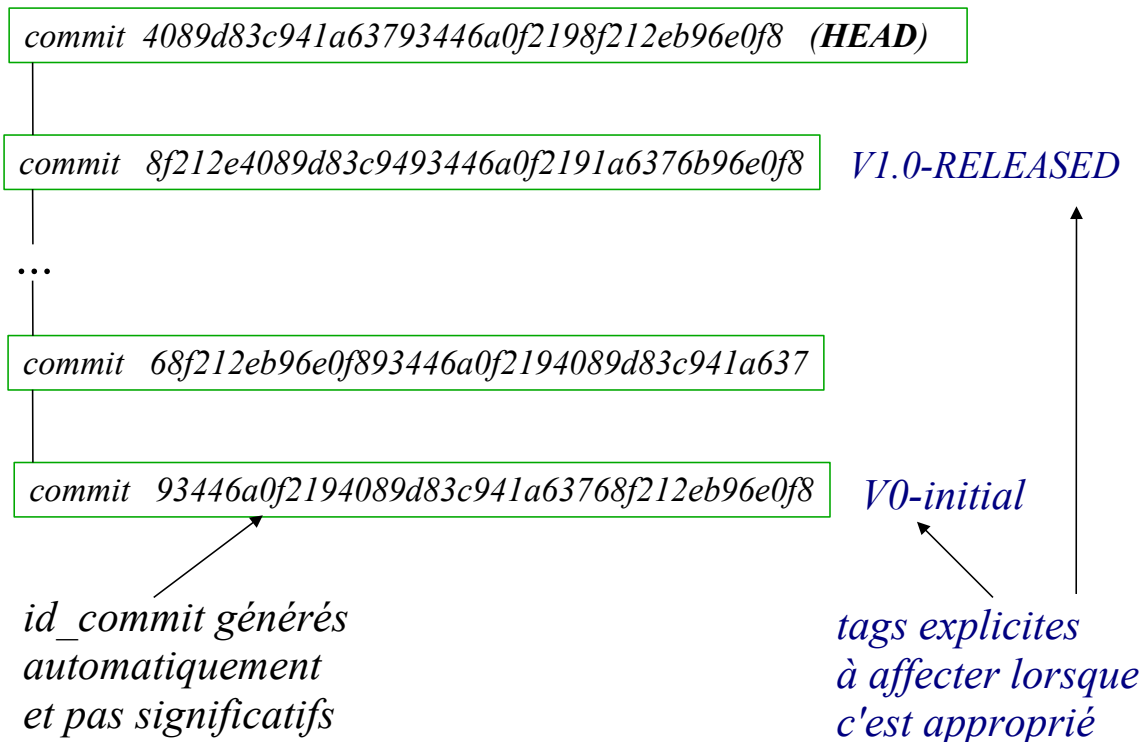
dans l'explorateur de fichiers à partir du répertoire du projet déclencher le menu contextuel **commit** --> ...



*choix des fichiers à "commiter"*





commits et tagsNB.:

Il est éventuellement possible de visualiser l'état du projet dans une révision (numéro de commit bien précis) via la commande **checkout** *numero\_de\_commit\_precis*.

Ceci dit, il ne vaut mieux pas éditer et enregistrer des fichiers juste après car on serait alors en mode "detached HEAD" (détaché d'une tête de branche) ce qui pose plein de problème par la suite.

---> pour effectuer des modifications à un niveau antérieur de l'historique, il faudra idéalement créer une nouvelle branche démarrant au niveau d'un commit\_bien\_precis.

## show-log et tag via tortoiseGit

The screenshot shows the TortoiseGit 'Log Messages' window for the repository 'D:\tp\local-git-mycontrib-repositories\tp\_angular4+'. The window displays a list of commits. The selected commit is on the 'master' branch, authored by 'didier' on '08/12/2017'. The commit message is '1.0.RELEASE'. A right-click context menu is open, showing options like 'Pull...', 'Fetch...', 'Push...', 'Diff', 'Diff with previous version', and 'Show log'. An annotation points to the commit message '1.0.RELEASE' with the text 'click droit, create tag at this version'.

Graph	Actions	Message	Author	Date
Working dir changes				
master		GitHubMyContribOrigin/master avec .bat	didier	08/12/...
		sans .bat	didier	08/12/...
		version initiale		

SHA-1: 7cf8866026e28c6dc09e4f309dd7c6c9ef6111e1

Path	Extension	Status	Li...	Li...
.gitignore	.gitignore	Modified	0	1

Showing 3 revision(s), from revision dd5a978 to revision 7cf8866 - 1 revision(s) selected, 0 file(s) selected

☐ Show Whole Project ☐ All Branches

Filter paths:  Help

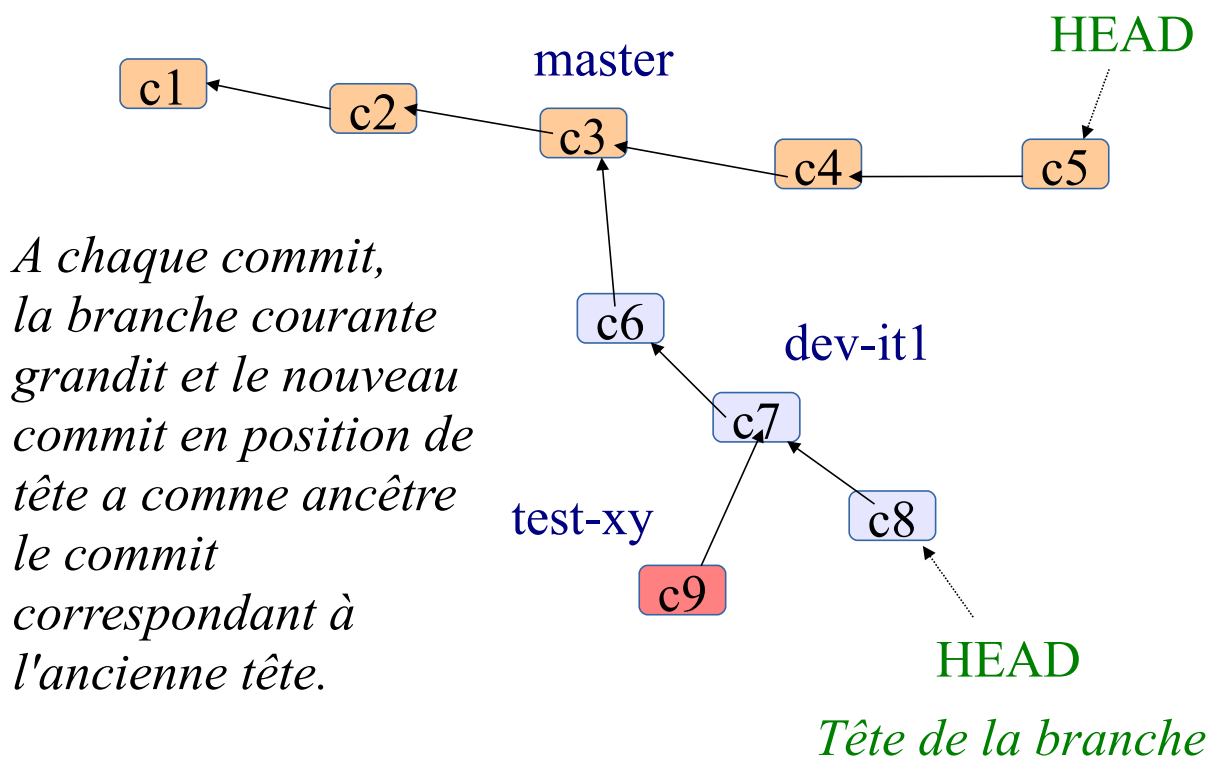
Refresh Statistics Walk Behaviour View OK

....

## III - Gestion des branches / GIT

### 1. Gestion des branches avec GIT

#### Structures des branches de GIT



## Annulation d'un commit erroné

En local, la commande **git reset --hard *idCommit*** permet de repositionner la tête de la branche courante sur un ancien commit et toute les modifications apportées par le(s) tout/s dernier(s) commit(s) seront effacées/perdues .

Si l'option **--soft** est utilisée à la place de **--hard**, les fichiers modifiés ne sont pas effacés du répertoire de travail et on peut les modifier avant d'effectuer un nouveau commit.

La syntaxe spéciale **HEAD~1** correspond à l'avant dernier commit et donc **git reset --hard HEAD~1** annule le dernier commit.

**HEAD^** ou **HEAD~1** : avant dernier commit

**HEAD^^** ou **HEAD~2** : avant avant dernier commit

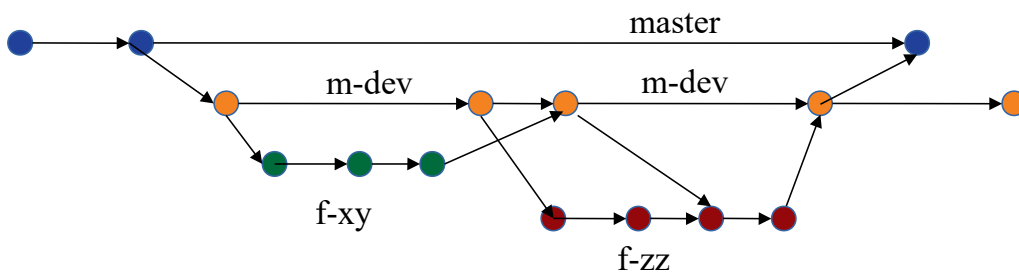
Tout projet commence avec une seule branche «**master**» .

Commandes GIT (branches)	Utilités
<b>git branch</b>	Affiche la liste des branches et précise la branche courante (*) .
<b>git branch</b> <i>nomNouvelleBranche</i>	Créer une nouvelle branche (qui n'est pas automatiquement la courante)
<b>git checkout</b> <i>nomBrancheExistante</i>	Changement de branche (avec mise à jour « checkout » des fichiers pour refléter le changement de branche) .
<b>git checkout master</b> <b>git merge</b> <i>autreBranche_a_fusionner</i>	Modifie la branche courante (ici «master») en fusionnant le contenu d'une autre branche
<b>git branch -d</b> <i>ancienneBrancheAsupprimer</i>	Supprime une ancienne branche (avec -d : vérification préalable fusion, avec -D : pas de verif , pour forcer la perte d'une branche morte)
...	

## 2. Bonnes pratiques dans la gestion des branches

## Bonnes pratiques élémentaires (branches GIT)

- ne pas directement programmer sur la branche "master" (à considérer comme la branche stable à déployer en production) mais sur une branche de développement .
- Un certain développeur peut éventuellement créer une sous branche d'ajout de fonctionnalité purement locale (sans push) pour expérimenter et tester une extension, effectuer un merge local et ultérieurement effectuer un push sur la branche parente/commune de développement .
- Attention à bien organiser (en équipe) la gestion des branches distantes (pour éviter pagaille et gros problèmes)

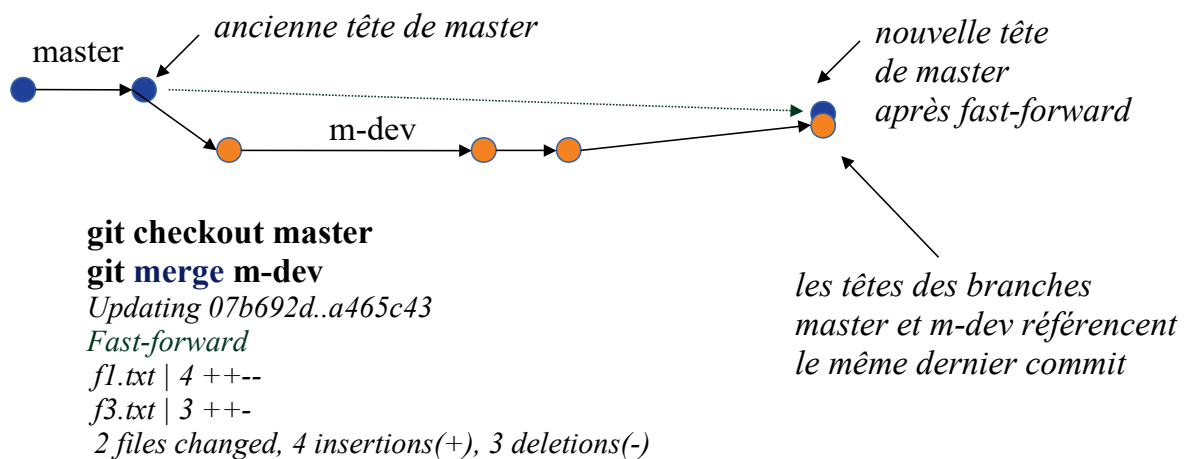


## 3. Merge

### 3.1. merge rapide (fast-forward)

#### Fast-forward (simple merge)

- Lorsque la branche à fusionner comporte quelques commits et que la branche actuelle (réceptrice de la fusion) ne comporte aucun autre commit depuis l'origine de la branche à fusionner, GIT peut effectuer un merge extrêmement rapide et simplifié appelé "fast-forward".
- Ce "fast-forward" consiste à actualiser la référence de la tête de la branche réceptrice pour qu'elle coïncide avec la tête de la branche à fusionner.

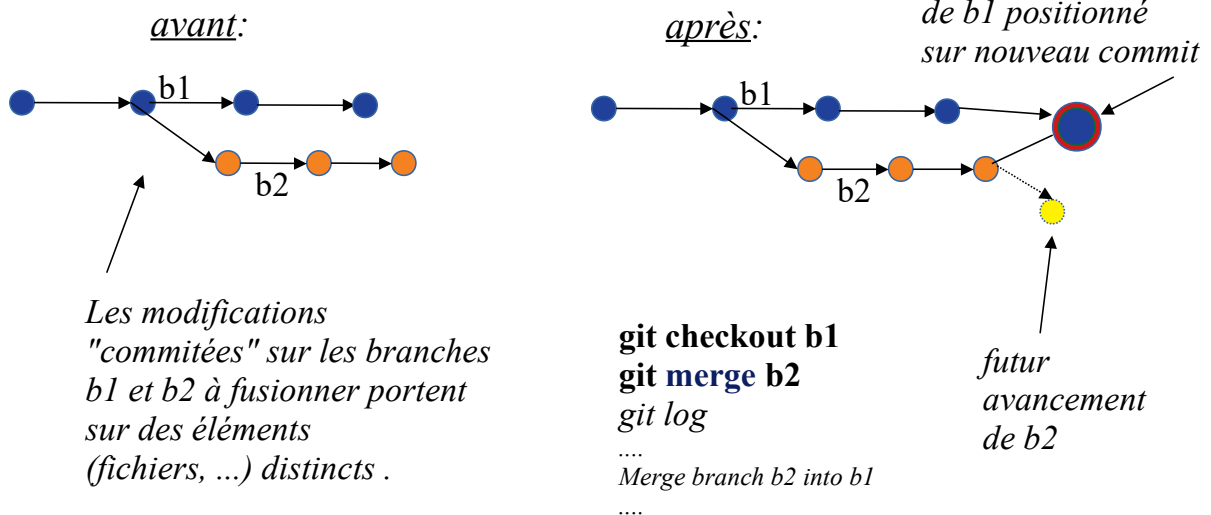


Mis à part ce cas très particulier, les autres sortes de "merge" conduiront à la création d'un nouveau commit de fusion (avec 2 ancêtres).

## 3.2. merge sans conflits

### Merge automatique (sans conflit)

- Lorsque les 2 branches à fusionner comportent quelques commits associés à des modifications/ajouts/suppressions de fichiers différents , GIT peut alors effectuer un merge automatique .
- La branche courante avance et sa nouvelle tête référence **un nouveau commit de fusion** (avec 2 ancêtres).

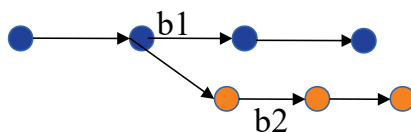


NB : Git est capable de gérer automatiquement un merge dans le cadre de modifications (de type "remplacements") apportées sur des lignes différentes d'un même fichier . Ceci n'étant qu'une heuristique , il convient de relancer certains tests unitaires (ou de relire certains fichiers) pour s'assurer que la fusion automatique de git a bien fonctionné .

### 3.3. merge avec conflits

#### Merge avec **conflit** à résoudre

- Lorsque les 2 branches à fusionner comportent toutes les 2 quelques commits associés à des modifications/ajouts/suppressions qui portent sur des fichiers communs , GIT ne peut alors pas décider de ce qu'il faut garder et les conflits doivent être résolus de manière interactive .
- via "git status" on peut connaître la liste des fichiers en conflit lors du merge
- soit le merge doit être annulé via "**git merge --abort**"  
soit **les conflits doivent être résolus** (éditions, add , commit)  
pour que le "merge" puisse aboutir .



← Les modifications  
"commitées" sur les branches  
b1 et b2 à fusionner portent  
sur quelques éléments  
(fichiers, ...) communs .

#### **git status (après merge)**

*On branch m-dev*

*You have unmerged paths.*

*(fix conflicts and run "git commit")*

*(use "git merge --abort" to abort the merge)*

*Unmerged paths:*

*(use "git add <file>..." to mark resolution)*

*both modified: fl.txt*

*no changes added to commit (use "git add" and/or "git commit -a")*



## Résolution de **conflit** (merge git)

début "git merge" avec conflits  
et "git status"

*f1.txt , ...*

```

<<<<<<< HEAD
f1 v2
aaa
=====
DEBUT
f1 v2
ABC
DEF
>>>>>> master
  
```

*éditions*  
*et tests*

*nouveaux contenus f1.txt , ...*

```

DEBUT
f1 v2
aaa bbb ccc
DEF
  
```

**git commit -m "fin merge xyz"**

**git add f1.txt ...**

**git add f1.txt**

>**git status**

*On branch m-dev*

*All conflicts fixed but you are still merging.*

*(use "git commit" to conclude merge)*

*Changes to be committed:*

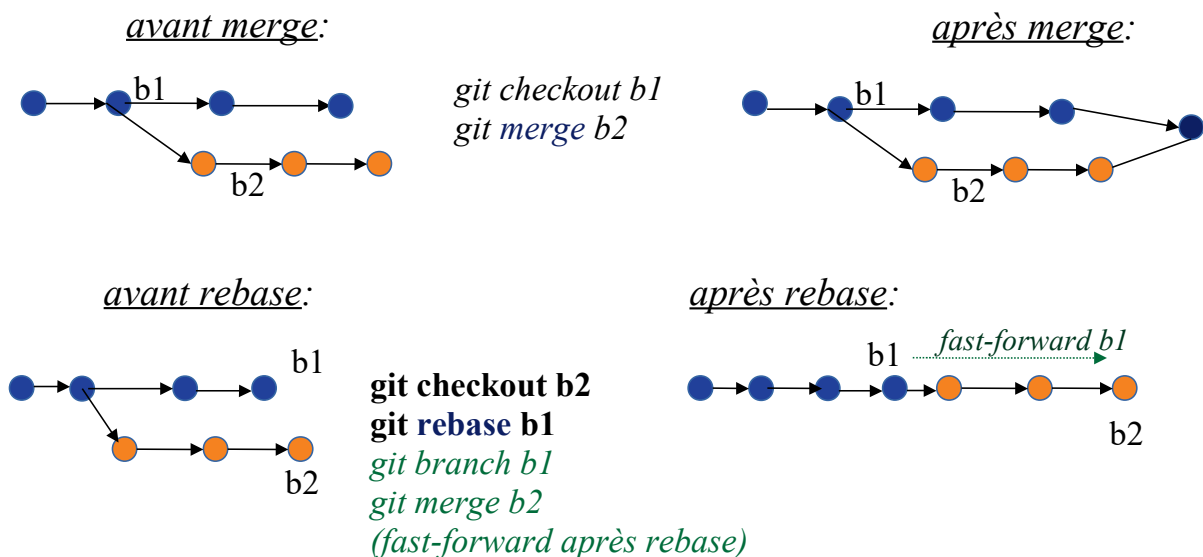
*modified: f1.txt*

**git commit -m "fin merge"**

## 4. Git rebase (souvent sur branche privée)

### git rebase

- La commande "git rebase" permet de reprendre toutes les modifications qui ont été validées sur une branche et de les rejouer sur une autre branche
- Ceci permet d'obtenir un historique linéaire (sans le nouveau commit de fusion d'un merge ordinaire)



Si lors du "rebase" certains conflits sont détectés, l'opération de "rebase" est alors temporairement stoppée. On peut alors :

- soit annuler le "rebase" via "**git rebase --abort**"
- soit résoudre le conflit de la même manière que pour un merge (édition, add ...) puis déclencher "**git rebase --continue**"

Un "rebase" est très déconseillé sur une branche publique car les réorganisations locales pourront une fois propagées mettre la pagaille chez les autres développeurs d'une équipe.

## IV - GIT en mode distant (--bare , github, ...)

### 1. Commandes de GIT pour le mode distant

Commandes GIT (mode distant)	Utilités
<b>git init --bare</b>	Initialisation d'un nouveau référentiel vide de type «nu» ou «serveur». (à alimenter par un push depuis un projet originel)
<b>git clone --bare url_repo_existant</b>	Idem mais via un clonage d'un référentiel existant
<b>git clone url_repo_sur_serveur.git</b>	Création d'une copie du projet sur un poste de développement (c'est à ce moment qu'est mémorisée l'url du référentiel « serveur » pour les futurs push et pull)
<b>git pull</b>	Rapatrie les dernières mises à jour du serveur distant (de référence) vers le référentiel local. (NB: <i>git pull</i> revient à déclencher les deux sous commandes <i>git fetch</i> et <i>git merge</i> )
<b>git push</b>	Envoie les dernières mises à jour vers le serveur distant (de référence)  <b><u>Attention:</u> le push est irréversible et personne ne doit avoir effectuer un push depuis votre dernier pull !</b>
...	

#### Exemples:

#script de création d'un nouveau référentiel GIT (coté serveur) dans /var/scm/git ou ailleurs:

```
mkdir p0.git
cd p0.git
git init --bare
git update-server-info
mv hooks/post-update.sample hooks/post-update

#nb www-data est le groupe de apache2
cd ..
sudo chgrp -R www-data p0.git

# ce repository initial et vide pourra être alimenté par un push depuis un projet "original"
# depuis ce projet original , on pourra lancer git config remote.p0.url http://localhost/git/p0.git
#           puis git push p0 master
echo "fin ?"; read fin
ou bien
# construira p1.git
```

```
git clone --bare file:///home/formation/Bureau/tp/tmp-test-git/original/p1
cd p1.git
git update-server-info
```

#récupération d'une copie du projet sur un poste de développement

```
git clone http://localhost/git/p1.git
```

#pull from serv:

```
cd p1
git pull
```

#push to serv:

```
cd p1
git push
```

## 2. Gérer plusieurs référentiels distants

Commandes GIT (mode distant)	Utilités
<b>git remote -v</b>	Affiche la liste des origines distantes (URL des référentiels distant)
<b>git remote add originXy url_repoXy</b>	ajoute une origine distante (alias associé à URL)
<b>git push -u originXy</b>	Effectue un push vers l'origine (upstream) précisé (alias associé à l'URL du référentiel distant).
<b>git push --set-upstream originXy master</b>	push en précisant la branche remote à pister (track) ceci est particulièrement utile pour un push initial vers référentiel distant vide (pas encore initialisé)
...	

Exemples :

*s\_list\_remote\_git\_url.bat*

```
git remote -v
pause
```

*s\_set\_git\_remote\_origin.bat*

```
git remote set-url origin Z:\TP\tp_angular1.git
git remote -v
```

```
pause
```

*s\_push\_to\_remote\_origin.bat*

```
git push -u origin master
```

```
pause
```

*s\_push\_to\_github.bat*

```
git remote add GitHubMyContribOrigin https://github.com/didier-mycontrib/tp_angular.git
```

```
REM didier-mycontrib / gh14.....sm..x / didier@d-defrance.fr
```

```
git push -u GitHubMyContribOrigin master
```

```
pause
```

*commit\_and\_push.bat*

```
cd /d "%~dp0"
```

```
git add *
```

```
git commit -a -m "nouvelle version"
```

```
git push -u GitHubMyContribOrigin master
```

```
pause
```

Bien qu'il soit possible de placer username et password dans l'URL ceci est une mauvaise pratique d'un point de vue sécurité .

Il vaut mieux configurer GIT pour qu'il retienne le mot de passe saisi durant une certaine période (exemple : 3600 secondes = 1 heure ) :

```
git config [ --global ] credential.helper 'cache --timeout=3600'
```

Et si nécessaire :

```
git credential-cache exit
```

pour que GIT oublie l'ancien mot de passe et que l'on puisse de ré-authentifier .

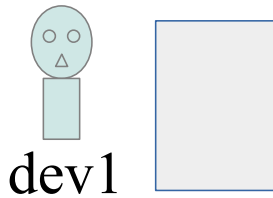
### 3. initialiser git en mode distant

#### initialiser git en mode "remote"

**1** *préparation d'un projet*

*git en mode local (java ou ...)*

*git init , .gitignore , ... , commit*



**3** *préciser url et premier push :*

**git remote add origin url\_repo**

**git push --set-upstream origin master**

**2** *préparation*

*du référentiel*

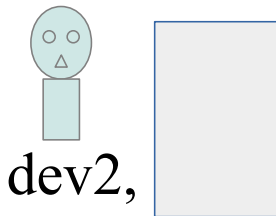
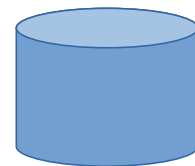
*git distant vide*

*(git init --bare)*

*ou équivalent via*

*console github*

*ou autre*



**4** *clonage(s) (avec "origin" fixée automatiquement) :*

**git clone url\_repo**

...

*Ensuite , git pull et git push de chaque coté .*

## 4. pistage (track) entre branches locales et distantes

### pistage (tracking) de branche

- Lorsqu'une branche locale est paramétrée pour pister (**track**) une branche distante , celles ci pourront ensuite être synchronisées via git push et git pull (et autres).
- Pour initialiser ce processus, il faut la première fois lancer la commande **git push** avec l'option **--set-upstream origin**
- Exemple: **git push --set-upstream origin m-dev**  
*[new branch] m-dev -> m-dev*  
*Branch m-dev set up to track remote branch m-dev from origin.*  
**git branch -vv**  
*\* m-dev 9f0d962 [origin/m-dev] ...*  
*master d41dc86 [origin/master] ...*

## **pistage (tracking) de branche (sur clones)**

- **git branch -a** permet de visualiser toutes les branches dont les branches distantes existantes :

*\* master*

*remotes/origin/HEAD -> origin/master*

*remotes/origin/m-dev*

*remotes/origin/master*

- Au niveau d'un clone secondaire , pour initialiser une branche locale pistant une branche distante existante on peut lancer la commande suivante :

**git checkout -b m-dev origin/m-dev**

*Branch m-dev set up to track remote branch m-dev from origin.*

*Switched to a new branch 'm-dev'*

**git branch -vv**

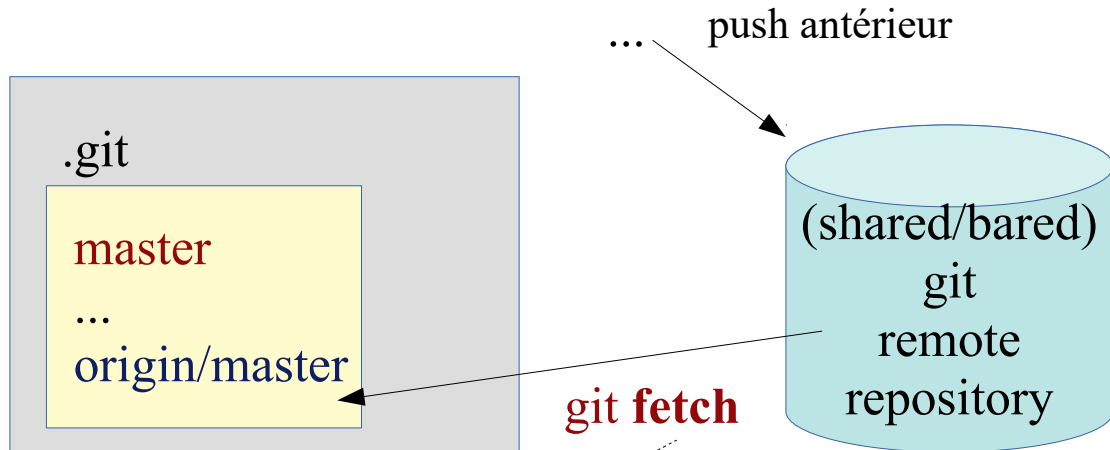
*\* m-dev 439df8e [origin/m-dev] ...*

*master d41dc86 [origin/master] ...*



## 5. Git fetch et git pull

### git fetch



② on peut ensuite , en étant placé sur la branche locale **master** voir les différences via **git branch -vv** et via **git diff origin/master**

① rapatrie (réplique) les mises à jour du référentiel distant dans la branche **origin/master** du référentiel local. Rien ne change sur la branche locale "master" ni dans le répertoire de travail (du projet).

## **git pull**

- En étant placé sur la branche master pistant la branche distante origin/master, **git pull** est équivalent à **git fetch** suivit de **git merge origin/master**
- il faut quelquefois résoudre certains conflits
- Les mises à jours distantes sont alors vues / répercutées dans le répertoire de travail (du projet)

# V - Git-flow et aspects divers

## 1. Pièges de GIT

### 1.1. Passwords oubliés (ou anciens passwords mémorisés)

```
git credential-cache exit
```

pour que GIT oublie l'ancien mot de passe et que l'on puisse de ré-authentifier .

### 1.2. Mauvaise pratique : password en clair dans URL

*NB : Lorsqu'une authentification est nécessaire , celle ci peut éventuellement **ET DANGEREUSEMENT** être placée dans L'URL du référentiel git distant :*

***https://username:password@github.com/xxx/yyy/zzz.git***

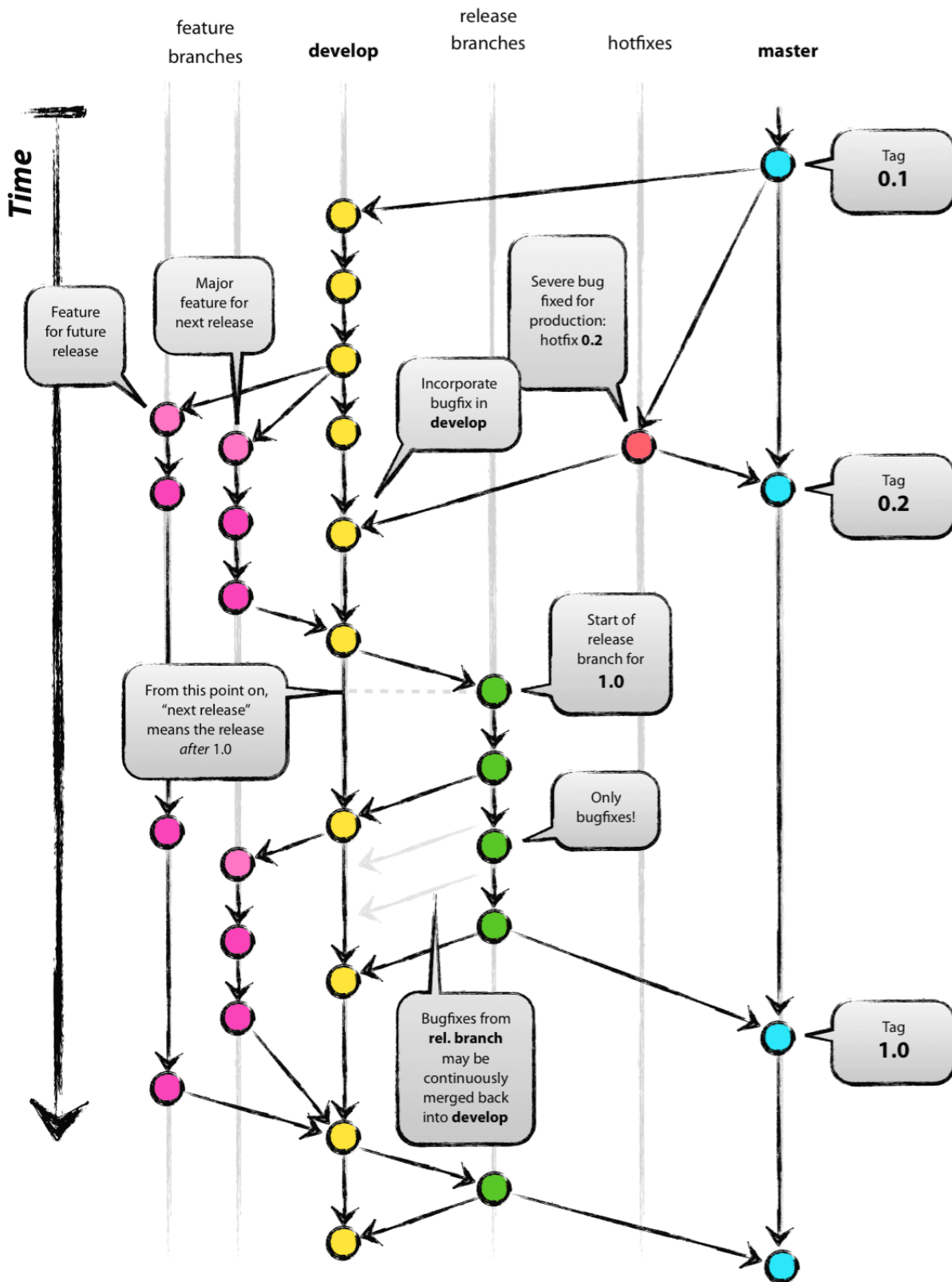
*Exemple: init\_push.bat*

```
git init
cd /d "%~dp0"
git add *
git commit -a -m "version initiale"
REM password doit être remplacé par la valeur du password !
git remote add gitHubOriginLilleSql https://didier-tp:password@github.com/didier-tp/lille_sql.git
git push -u gitHubOriginLilleSql master
pause
```

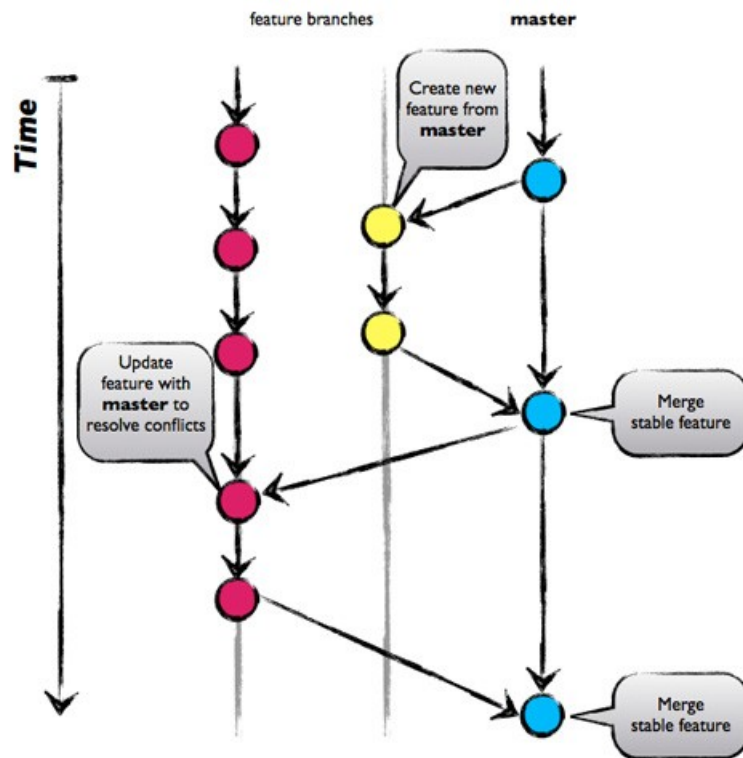
*Attention à bien régler **.gitignore** pour ne pas placer le mot de passe dans le référentiel distant !!! (autre inconvénient : password visible dans URL via **git remote -v** ) .*

## 2. Quelques workflows pour GIT

### 2.1. git-flow (évolué)



## 2.2. github-flow ( simple)



# ANNEXES



# VI - Annexe – GIT avec eclipse

## 1. Plugin eclipse pour GIT (EGIT)

Le plugin eclipse pour GIT s'appelle EGIT .

### 1.1. Actions basiques (commit , checkout , pull , push)

→ Se laisser guider par la perspective "GIT" et via le menu "Team"

### 1.2. Résolution de conflits

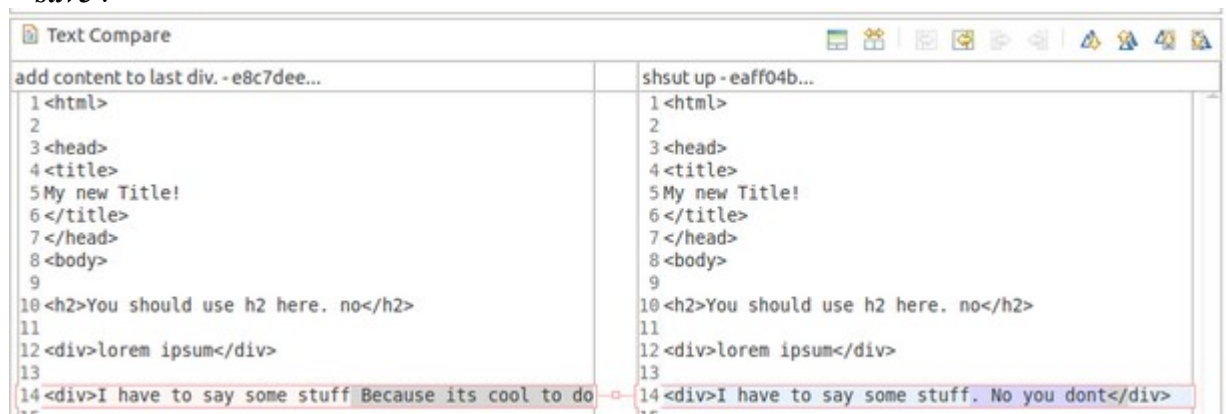
- 1) déclencher "**Team / pull**" pour récupérer (en tâche de fond) la dernière version (partagée / de l'équipe). Le plugin EGIT va alors tenter un "**auto-merge**" ("git fetch FETCH\_HEAD" suivi par "git merge").
- 2) En cas de conflit (non résoluble automatiquement) , les fichiers en conflit seront marqués d'un point rouge.

```
<<<<<< HEAD
<div>I have to say some stuff Because its cool to do so.</div>
=====
<div>I have to say some stuff. No you dont</div>
>>>>>> branch 'master' of /var/data/merge-issue.git
```

Sur chacun des fichiers en conflit , on pourra déclencher le *menu contextuel*

"**Team / Merge tool**" . (laisser par défaut la configuration de "Merge Tool" : use HEAD).

- 3) **Saisir , changer ou supprimer alors au moins un caractère dans la zone locale (à gauche) + save :**



... au cas par cas ....

- 4) Déclencher le menu contextuel "**Team / add to index**" pour ajouter le fichier modifié dans la liste de ceux à gérer (staging).
- 5) Effectuer un "**Team / commit**" local .
- 6) Effectuer un "**Team / push to upstream ...**" pour mettre à jour le référentiel distant/partagé .

## VII - Annexe – Accès distant via http (conf, admin)

### 1. Configuration d'accès distant à un référentiel Git

Si Répertoire "réseau" partagé (via NFS ou autre) , URL possible en [file:///](#)  
(ex: [file:///var/git/project.git](#))

#### 1.1. Accès distant (non sécurisé) via git

URL de type **git://nomMachineOuDomaine/xxxx/projetYy.git**  
où xxx est le chemin menant au référentiel git sur la machine "serveur" (ex: [/var/git/](#) )  
(Alias possible dans .gitconfig)

#### 1.2. Accès distant sécurisé via git+ssh

URL de type **ssh://user@hostXx/var/git/projectYy.git**

git+ssh est un tunneling sécurisé ssh pour le protocole git

Les clefs (publiques et privées) ssh sont placés dans le répertoire **\$HOME/.ssh**  
Celles ci se génèrent via la commande **ssh-keygen** .  
Lors de la génération des clefs , un mot de passe (*passphrase*) à retenir est demandé .  
Ce mot de passe peut éventuellement être vide (sécurité alors que via la clef publique) .  
La **clef publique** (à envoyer par email ou ....) correspond au fichier **id\_dsa.pub** (ou bien **id\_rsa.pub**) .

Si la partie "serveur" de ssh n'est pas encore installée , on peut alors lancer "**sudo apt-get install openssh-server**" puis éventuellement "**sudo service ssh start**" .

NB : sur certaines versions de Linux Ubuntu , la commande apt-get install ne fonctionne pas bien avec openssh-server et l'on peut dans ce cas installer alternativement openssh-server de la façon suivante :

- 1) télécharger le fichier **openssh-server\_5.9p1-5ubuntu1\_i386.deb** (via une recherche google)
- 2) lancer **sudo dpkg --install ./openssh-server\_5.9p1-5ubuntu1\_i386.deb**
- 3) redémarrer linux (ou ...) pour que le service "ssh" soit activé

#### 1.3. Accès distant en lecture seule via http (sans webdav)

Configurer un accès de apache2 vers un répertoire correspondant à un référentiel git et activer le hook "post-update" en renommant post-update.sample en post-update .



```
cd xy.git
mv hooks/post-update.sample hooks/post-update
```

## 1.4. Accès distant en lecture/browsing via gitweb

En configurant sur le poste serveur, l'extension "gitweb"(pour apache2 et git), on peut alors parcourir toute l'arborescence d'un projet GIT via un simple navigateur.

<http://localhost/gitweb/?p=p0.git>



Activation et configuration du site web "gitweb":

```
cd /var/www;
sudo mkdir gitweb;
cd gitweb;
sudo cp /usr/share/gitweb/* . ;
sudo cp /usr/share/gitweb/static/* .
```

Il faut également fixer la variable **\$projectroot** = **"/var/scm/git/"** dans le fichier **/etc/gitweb.conf**

**/etc/gitweb.conf**

```
$projectroot="/home/formation/scm/git";
# directory to use for temp files
$git_temp="/tmp";
# html text to include at home page
$home_text="indextext.html";
# file with project list; by default, simply scan the projectroot dir.
$projects_list=$projectroot;
# stylesheet to use
# I took off the prefix / of the following path to put these files inside gitweb directory directly
$stylesheet="gitweb.css";
# logo to use
$logo="git-logo.png";
# the 'favicon'
$favicon="git-favicon.png";
```

D'autre part, le module " RewriteEngine" d'apache2 doit être activé.  
Si ce n'est pas encore le cas, on l'active via la commande "**sudo a2enmod rewrite**"

Créer et configurer un nouveau fichier pour configurer gitweb sous apache2 :

**/etc/apache2/conf.d/git.conf**

```
...
<Directory /var/www/gitweb >
SetEnv GITWEB_CONFIG /etc/gitweb.conf
DirectoryIndex gitweb.cgi
Allow from all
AllowOverride all
Order allow,deny
Options +ExecCGI
AddHandler cgi-script .cgi
<Files gitweb.cgi>
    SetHandler cgi-script
</Files>
RewriteEngine on
RewriteRule ^[a-zA-Z0-9_-]+.git/?(\?.)?$ /gitweb.cgi%{REQUESTURI} [L,PT]
</Directory>
....
```

Redémarrage du service apache2:  
**service apache2 restart**

### 1.5. Accès distant "rw" via http/https (webdav)

Activer les *modules apache2* "dav" , "dav\_fs" et "dav\_lock"

```
sudo a2enmod dav
sudo a2enmod dav_fs
sudo a2enmod dav_lock
```

Créer et configurer un nouveau fichier pour configurer git sous apache2 :

**/etc/apache2/conf.d/git.conf**

```
...
Alias /git /var/scm/git/

<Location /git>
    DAV on
    #AuthType Basic
    #AuthName "Git"
    #AuthUserFile /etc/apache2/dav_git.passwd
    #Require valid-user
</Location>
...
```

Redémarrage du service apache2:

**service apache2 restart**

+ si besoin paramétrage d'autres détails (sécurité , ... ) :

```
<Directory "/var/scm/git/">
Options Indexes FollowSymLinks MultiViews ExecCGI
#DirectoryIndex index
AllowOverride None
Order allow,deny
allow from all
</Directory>
```

Eventuel paramétrage (facultatif et très délicat) pour optimiser les transferts (par paquets) via HTTP:

```
# Git-Http-Backend (for smart http push , useful with egit )
SetEnv GIT_PROJECT_ROOT /var/scm/git/
SetEnv GIT_HTTP_EXPORT_ALL
#ScriptAlias /git/ /usr/lib/git-core/git-http-backend/
ScriptAliasMatch \
    "(?x)^(git/(.*(HEAD | \
        info/refs | \
        objects/(info/[^/]+ | \
            [0-9a-f]{2}/[0-9a-f]{38} | \
            pack/pack-[0-9a-f]{40}\.(pack|idx)) | \
            git-(upload|receive)-pack))$" \
    "/usr/lib/git-core/git-http-backend/$1"

<LocationMatch "^/git/.*/git-receive-pack$">
    #AuthType Basic
    #AuthName "Git Access"
    #Require group committers
</LocationMatch>
```

## VIII - Annexe – Bibliographie, Liens WEB + TP

### 1. Bibliographie et liens vers sites "internet"

<a href="https://git-scm.com/">https://git-scm.com/</a>	site officiel de la technologie git
<a href="https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git">https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git</a>	tutoriel sur GIT

### 2. TP

Tps progressifs selon suggestions du formateur