C08-git-svn - COURS LINUX - MTN

Guillaume ASTIER

26/02/16

Table des matières

Introduction

On retiendra surtout que Git :

- est très rapide ;
- sait travailler par branches (versions parallèles d'un même projet) de façon très flexible ;
- est assez complexe, il faut un certain temps d'adaptation pour bien le comprendre et le manipuler, mais c'est également valable pour les autres outils ;
- est à l'origine prévu pour Linux. Il existe des versions pour Windows mais pas vraiment d'interface graphique simplifiée. Il est donc à réserver aux développeurs ayant un minimum d'expérience et… travaillant de préférence sous Linux.

Une des particularités de Git, c'est l'existence de sites web collaboratifs basés sur Git comme GitHub. GitHub, par exemple, est très connu et utilisé par de nombreux projets : jQuery, Symfony, Ruby on Rails…

C'est une sorte de réseau social pour développeurs : vous pouvez regarder tous les projets évoluer et décider de participer à l'un d'entre eux si cela vous intéresse. Vous pouvez aussi y créer votre propre projet : c'est gratuit pour les projets open source et il existe une version payante pour ceux qui l'utilisent pour des projets propriétaires.

GitHub fournit le serveur où les développeurs qui utilisent Git se rencontrent. C'est un excellent moyen de participer à des projets open source et de publier votre projet !

installation

Télécharger git pour Mac OSX

Télécharger git pour Windows

Télécharger git pour Linux

créer un nouveau dépôt

créez un nouveau dossier, ouvrez le et exécutez la commande

```
1 git init
```

pour créer un nouveau dépôt.

---

cloner un dépôt

créez une copie de votre dépôt local en exécutant la commande

```
1 git clone /path/to/repository
```

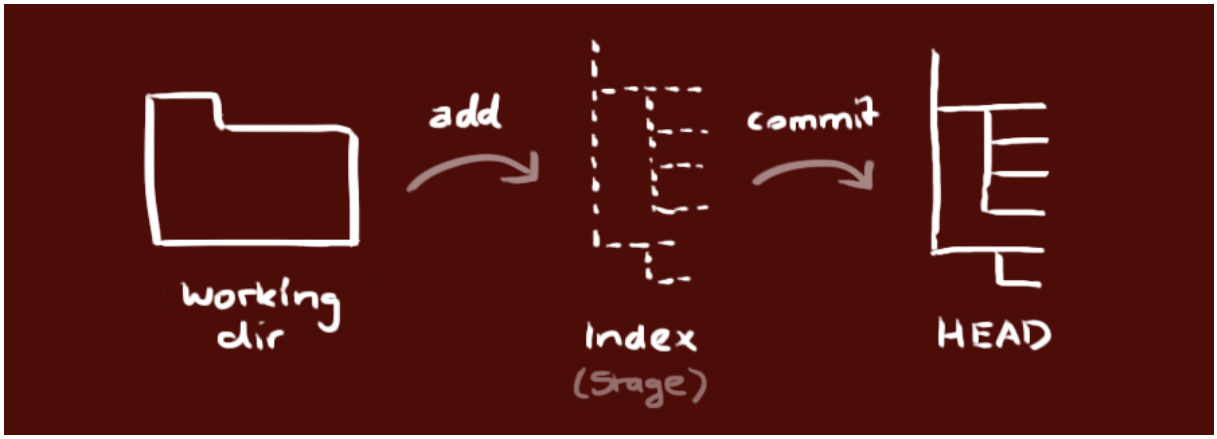
si vous utilisez un serveur distant, cette commande sera

```
1 git clone username@host:/path/to/repository
```

---

arbres

votre dépôt local est composé de trois “arbres” gérés par git. le premier est votre ‘espace de travail’ qui contient réellement vos fichiers. le second est un ‘Index’ qui joue un rôle d’espace de transit pour vos fichiers et enfin ‘HEAD’ qui pointe vers la dernière validation que vous ayez faite.



ajouter & valid

Vous pouvez proposer un changement (l’ajouter à l’**Index**) en exécutant les commandes

```
1 git add <filename>
2
3 git add *
```

C’est la première étape dans un workflow git basique. Pour valider ces changements, utilisez

```
git commit -m "Message de validation"
```

Le fichier est donc ajouté au **HEAD**, mais pas encore dans votre dépôt distant.

envoyer des changements

Vos changements sont maintenant dans le **HEAD** de la copie de votre dépôt local. Pour les envoyer à votre dépôt distant, exécutez la commande

```
git push origin master
```

Remplacez *master* par la branche dans laquelle vous souhaitez envoyer vos changements.

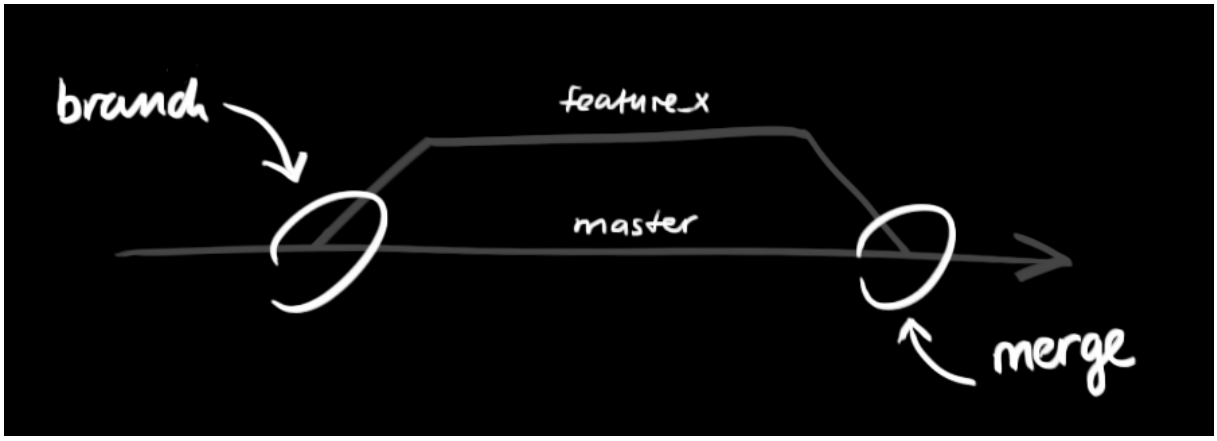
Si vous n'avez pas cloné votre dépôt existant et voulez le connecter à votre dépôt sur un serveur distant, vous devez l'ajouter avec

```
git remote add origin <server>
```

Maintenant, vous pouvez envoyer vos changements vers le serveur distant sélectionné

branches

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche *master* est la branche par défaut quand vous créez un dépôt. Utilisez les autres branches pour le développement et fusionnez ensuite à la branche principale quand vous avez fini.



créer une nouvelle branche nommée "feature_x" et passer dessus pour l'utiliser

```
git checkout -b feature_x
```

---

retourner sur la branche principale

```
git checkout master
```

et supprimer la branche

```
git branch -d feature_x
```

---

une branche n'est *pas disponible pour les autres* tant que vous ne l'aurez pas envoyée vers votre dépôt distant

```
git push origin <branch>
```

---

mettre à jour & fusionner

pour mettre à jour votre dépôt local vers les dernières validations, exécutez la commande

```
git pull
```

dans votre espace de travail pour *récupérer* et *fusionner* les changements distants.

pour fusionner une autre branche avec la branche active (par exemple master), utilisez

```
git merge <branch>
```

---

dans les deux cas, git tente d'auto-fusionner les changements. Malheureusement, ça n'est pas toujours possible et résulte par des *conflits*. Vous devez alors régler ces *conflits* manuellement en éditant les fichiers indiqués par git. Après l'avoir fait, vous devez les marquer comme fusionnés avec

```
git add <filename>
```

---

après avoir fusionné les changements, vous pouvez en avoir un aperçu en utilisant

```
git diff <source_branch> <target_branch>
```

tags

il est recommandé de créer des tags pour les releases de programmes. c'est un concept connu, qui existe aussi dans SVN. Vous pouvez créer un tag nommé *1.0.0* en exécutant la commande

```
git tag 1.0.0 1b2e1d63ff
```

---

le *1b2e1d63ff* désigne les 10 premiers caractères de l'identifiant du changement que vous voulez référencer avec ce tag. Vous pouvez obtenir cet identifiant avec

```
git log
```

vous pouvez utiliser moins de caractères de cet identifiant, il doit juste rester unique.

remplacer les changements locaux

Dans le cas où vous auriez fait quelque chose de travers (ce qui bien entendu n'arrive jamais ;) vous pouvez annuler les changements locaux en utilisant cette commande

```
git checkout -- <filename>
```

---

cela remplacera les changements dans votre arbre de travail avec le dernier contenu du HEAD. Les changements déjà ajoutés à l'index, aussi bien les nouveaux fichiers, seront gardés.

Si à la place vous voulez supprimer tous les changements et validations locaux, récupérez le dernier historique depuis le serveur et pointez la branche principale locale dessus comme ceci

```
git fetch origin

git reset --hard origin/master
```

conseils utiles

Interface git incluse

```
1 gitk
```

utiliser des couleurs dans la sortie de git

```
1 git config color.ui true
```

afficher le journal sur une seule ligne pour chaque validation

```
1 git config format.pretty oneline
```

utiliser l'ajout interactif

```
1 git add -i
```

Introduction

: C'est quoi la gestion de version

La gestion de versions consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers (généralement en texte). Essentiellement utilisée pout la gestion des codes source, elle peut être cependant utilisé dans la gestion de paquets systèmes, dans la gestion de fiches matériel ou encore la gestion de projets et des fichiers qui le compose.

Schématiquement, on passera de la version N à la version N + 1 en appliquant une modification. Une modification constitue donc l'évolution entre deux versions.

Gestion de versions centralisée :

l n'existe qu'un seul dépôt des versions qui fait référence. Cela simplifie la gestion des versions mais est contraignant pour certains usages comme le travail sans connexion au réseau, ou tout simplement lorsque l'on travaille sur des branches expérimentales ou contestées.

Gestion de versions décentralisée :

La Gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un

moyen à ces développeurs de s'échanger leur travaux respectifs. De fait, il existe plusieurs dépôts pour un même logiciel.

- GNU RCS (1982)
- CVS (1990)
- CVSNT (1992)
- SVN (2000)
- Rational ClearCase (1992) (Propriétaires)
- SNIFF RCS ++ (Propriétaires)

- Mercurial (2005)
- Git (2005)
- Veracity (2011)
- BitKeeper (1998) (Propriétaires)
- Plastic SCM (2007) (Propriétaires)

Commit :

Apporter une modification au projet, ou, plus formellement, enregistrer un changement dans la base de données de gestion de versions, pour qu'il puisse être ajouté dans une version future du projet

Messages enregistrés / commentaires :

Quelques commentaires joints à chaque commit, décrivant la nature et le but du commit. Les messages enregistrés font partie des documents les plus importants d'un projet : ils font le lien entre le langage très technique des modifications du code et le langage plus compréhensible qui se rapporte aux fonctionnalités, aux corrections de bogues et à la progression du projet.

Mise à jour :

Demander que les autres modifications (commit) soient incorporées dans votre propre version du projet, c'est à dire, mettre votre copie à jour. C'est une opération de routine, la plupart des développeurs mettent à jour leur code plusieurs fois par jour.

Dépôt :

Une base de données au sein de laquelle les modifications sont stockées. Certains logiciels de gestion de versions sont centralisés : il y a un unique dépôt maître qui conserve toutes les modifications du projet. D'autres sont décentralisés : chaque développeur possède son propre dépôt et les modifications peuvent être partagées entre les dépôts de manière arbitraire.

Retrait / chekout:

L'obtention d'une copie du projet depuis le dépôt. Un retrait produit en général une arborescence de répertoires appelée « copie de travail »

Copie de travail :

L'arborescence personnelle d'un développeur contient les fichiers du code source du projet,

Révision :

C'est une incarnation précise d'un fichier ou dossier particulier dans un état spécifique à un instant T.

Ex : Si le projet commence avec la révision 6 du fichier F et qu'ensuite quelqu'un modifie F on parlera alors de la révision 7

Diff :

Un diff montre quelles lignes ont été modifiées et comment, en ajoutant quelques lignes de contexte d'un côté ou de l'aut

Branche :

Une copie du projet, sous gestion de versions mais isolée, afin que les modifications de cette branche n'affectent pas le reste du projet

Fusion :

Transférer une modification d'une branche à une autre. Cela englobe la fusion du tronc principal vers d'autres branches et inversement

---

Conflit :

C'est ce qui se passe quand deux personnes tentent de faire des changements au même endroit du code.

Verrouiller :

Une manière de se réserver les modifications sur un fichier ou un dossier particulier.

Tags :

Lorsqu'un ensemble de modifications aboutit à une version stable que l'on souhaite pouvoir retrouver à l'avenir, il faut faire un tag du repository

---

A la base, les sources d'un projet sont disponibles sur la branche principale (trunk). Les développeurs y récupèrent les sources (checkout) sur leur propre environnement de travail, et y ajoutent leurs versions modifiées (commit).

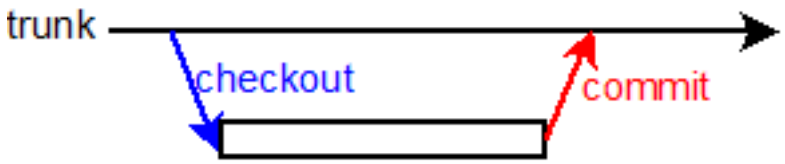


Figure 1: Utilisation Classique

Plusieurs utilisateurs peuvent récupérer les sources et committer leurs modifications.

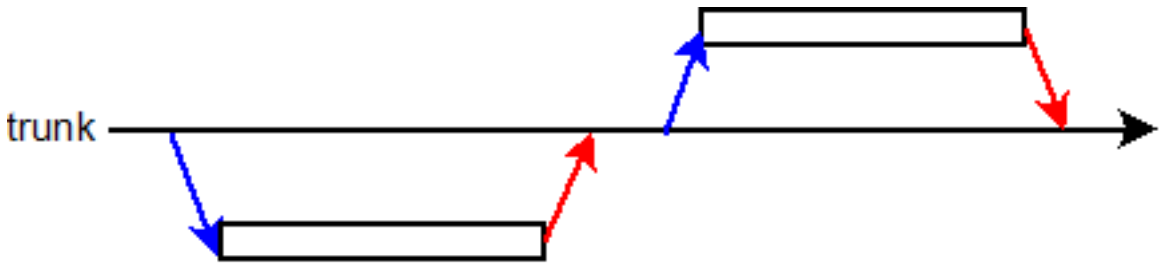


Figure 2: Multi user

La mise-à-jour (update) permet de récupérer les modifications commitées par les autres utilisateurs. Une bonne pratique : avant de committer, il faut mettre-à-jour pour récupérer les dernières modifications committées par les autres utilisateurs.

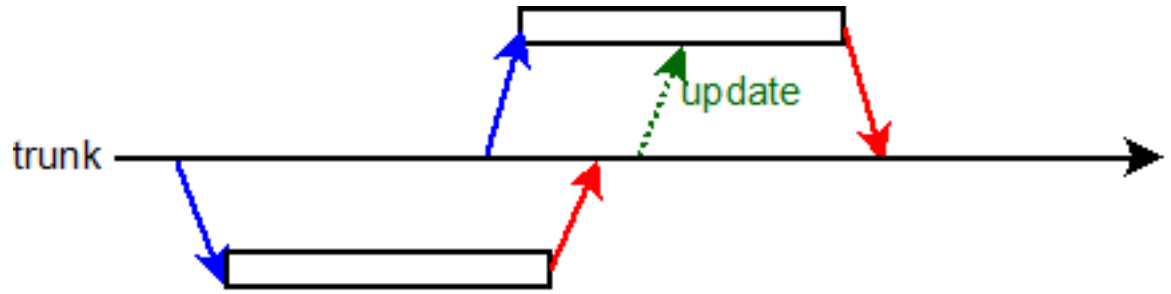


Figure 3: Update

Lorsqu'un ensemble de modifications aboutit à une version stable que l'on souhaite pouvoir retrouver à l'avenir, il faut faire un tag du repository.

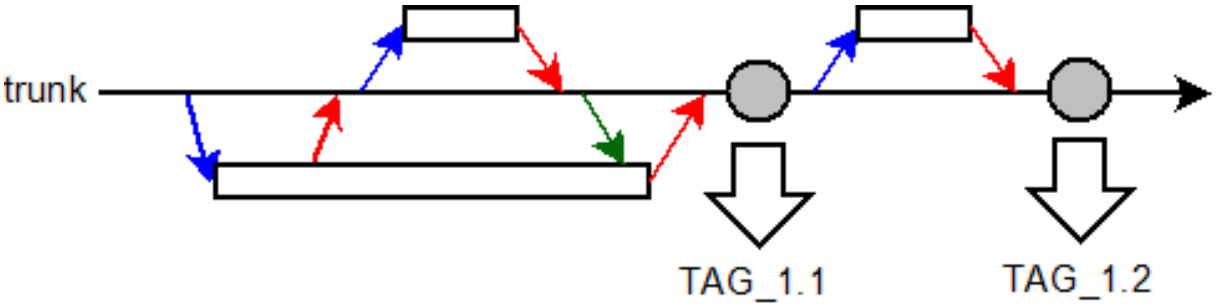


Figure 4: Update

Exempl complexe :

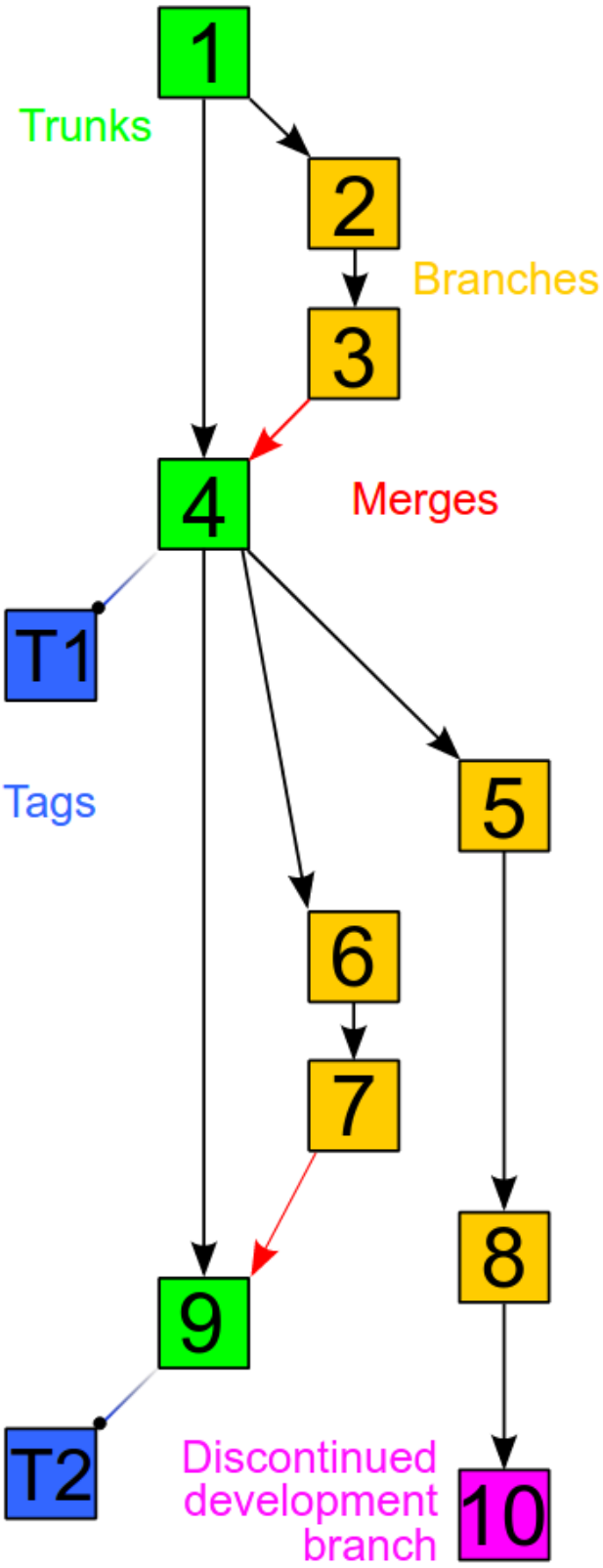


Figure 5: Update

Les commandes SVN

Projet déjà existant au sein d'un dépôt

Si le projet existe déjà au sein du dépôt, une seule commande suffit pour effectuer un checkout et récupérer la dernière version des fichiers :

il s'agit de la commande svn co.

```
$ svn co [URL]
A  file1
A  file2
Checked out revision 36.
```

Récupération de la dernière version du projet

```
$ svn update
U  index.htm
Updated to revision 37.
```

Mise à jour des modifications dans le dépôt

```
$ svn commit -m "Ajout d'un commentaire"
ou
$ svn co
```

Récupération de la dernière version

Avant de CI vous voulez retouvez la version de votre dernier update

```
$ svn revert [File]
```

Récupération d'une version antérieure du dépôt

```
$ svn update -r 36  [File]
U [File]
Updated to revision 36.
```

Ajout d'un fichier

```
$ svn add [FILE]
```

Suppression d'un fichier

```
svn delete [File]
# puis commit
svn commit -m "Suppression d'un fichier"
```

Renommer un fichier

```
$ svn move file1 newnamefile1
```
