

Git

Michel Meynard

UM

Université Montpellier

Avantages du gestionnaire de versions décentralisé Git

- travail collaboratif sur un projet commun : Git créé pour le développement de linux ;
- gestion des conflits de révisions bien plus pratique que par email ou clé USB !
- sauvegarde distante et locale du projet (même si un seul développeur) ;
- récupération possible après des modifications désastreuses !
- branches de développement multiples correspondant à des fonctionnalités différentes ou à des tests ;
- avec Git, le dépôt local contient toutes les versions, pas seulement la version courante (SVN) ; en cas de crash du dépôt distant, pas de problème !

Table des matières

- 1 Introduction
- 2 Propriétés
- 3 git log - historique du projet
- 4 git add et rm - ajout et suppression
- 5 git commit - validation
- 6 Travailler avec des dépôts distants
- 7 GitLab du Service Info. de la FDS
- 8 MaGit, un paquetage pour Emacs

Git en résumé - 1

Installation d'un client git

linux `sudo apt-get git-core gitk`

Windows Git for windows <http://code.google.com/p/msysgit/> + option TortoiseGit (graphique)
<http://code.google.com/p/tortoisegit/> ou bien <http://git-scm.com> directement !

Cygwin cocher le paquetage Git ;

Mac OS <http://code.google.com/p/git-osx-installer/>

Démarrer un dépôt git local

2 options :

- soit cloner un dépôt git existant : `git clone https://...`
- soit débiter un nouveau dépôt vide : `git init`

Git en résumé - 2

Les commandes les plus fréquentes

A exécuter depuis le répertoire racine du dépôt :

- état du dépôt : `git status`
- récupérer la dernière version du serveur : `git pull`
- modifier un fichier et l'envoyer :
 - 1 éditer le fichier ;
 - 2 `git add` le fichier ajoute un fichier **ou prend en compte** ses modifs ;
 - 3 `git commit` valider les modifications et **commente** ; plusieurs "add" avant un "commit"
 - 4 envoyer sur le serveur : `git push` ; 1 ou plusieurs "commit" avant un "push" ;

Lors du premier envoi il faut utiliser la commande "`git push origin master`" pour indiquer qu'il s'agit de la branche "master" que l'on souhaite synchroniser.

Caractéristiques

décentralisé

- branchement (fork) aisé ainsi que fusion de branches ;
- système de fichier n'utilisant pas de BD ;
- forker un projet ou le cloner permet de rapatrier localement tout l'historique du projet ;
- proposer sa contribution (pull request) au dépôt principal (mainteneur principal du projet) ;

communication avec un dépôt distant

via le protocole git (port 9418), http, https, ssh, local ;

Git versus SVN ?

différences par rapport à SVN

- **nécessité** de faire `git add` fic après chaque modif de fic !
- `git commit` valide **localement** les modifs effectuées par `git add` ... ou `git rm`
- `git commit -a` valide **localement** toutes les modifs effectuées sur des fichiers sous contrôle (mais pas les nouveaux) ;
- `git push` permet d'envoyer les derniers commit au dépôt distant ;
- `git pull` récupère les derniers commits (fetch) et les fusionne localement (merge) ;

Remarquons que `git status` permet de savoir ce que le commit suivant va avoir à traiter : fichiers non suivis, modifs non orchestrées (staged) ;

Où est le dépôt local ?

Le répertoire racine d'un dépôt contient un répertoire `.git` qui mémorise le dépôt et de nombreuses informations :

- le répertoire `objects` contient toutes les versions de fichiers ;
- le fichier `config` contient des propriétés (clé, valeur) dans des rubriques telles que `remote` (dépôt distant) ou `branch` ;
- le répertoire `info` contient un fichier `exclude` qui définit des expressions régulières correspondant au noms de fichier que ne doit pas traiter git (règles non visibles par les autres usagers) ;
- ...

Le fichier `.gitignore` contient lui les expressions régulières correspondant au noms de fichier que doit ignorer git ; Il doit être sous contrôle de version : `git add .gitignore`

Configuration

La configuration peut être globale à l'utilisateur (~/.gitconfig) ou locale au dépôt (racine/.git/config) ou même globale à la machine (--system)

A faire une fois pour tous les dépôts présents et à venir

```
git config --global user.name "Michel Meynard"
git config --global user.email "michel.meynard@univ-montp2.fr"
```

Les validations futures useront de cet identité **sauf** si une identité locale à un dépôt a été redéfini :

```
git config --local user.name "MM"
git config --local user.email "mm@free.fr"
```

Visualiser (log) l'historique des validations (commit)

Chaque commit :

- est identifié par une chaîne de 40 car. hexadécimaux (somme de contrôle SHA-1);
- est effectué par un utilisateur ayant un nom et un email (config);
- a une date;
- a un message indiquant textuellement les modifications;
- a une liste des modifications effectuées

Afficher les dernières modifications

```
$ git log -p -2
commit ddf4c11106f9cfe973a4c080cb14c69a96cdc3df
Author: Marianne Huchard <huchard@lirmm.fr>
Date: Wed Nov 19 09:09:06 2014 +0100

ajout article
diff --git a/Etat-Art/DocumentEtatArt/etatArt.tex b/Etat-A
```

Annotations: commande (pointe à \$ git log -p -2), id (pointe à commit), user (pointe à Author), date (pointe à Date), message de commentaire (pointe à ajout article), pour chacun des fichiers modifiés (pointe à diff), avant (pointe à a/Etat-Art/DocumentEtatArt/etatArt.tex), après (pointe à b/Etat-A)

Les options de git log

Cette commande possède de multiples options combinables :

- nombre des plus récents commit à visualiser (-2), par défaut tous les commits sont affichés !
- patch (-p) : affiche les différences (diff) entre chaque fichier modifié par le commit;
- recherche d'une chaîne dans les fichiers modifiés (-S"int fact(")
- voir les embranchements (--graph);
- affichage formaté (--pretty=oneline) permet de n'afficher qu'une ligne par commit;
- que les commit d'un auteur (--author=="Michel Meynard")
- en fonction des dates (--since=2.weeks), until, before, after ...

Ajout et suppression

Ajouter un répertoire

L'ajout d'un répertoire par git add rep ajoute tous les fichiers et répertoires inclus **sauf** s'ils correspondent aux expressions régulières contenues dans le fichier .gitignore de la racine du dépôt local. **L'ajout** est effectué dans l'index du commit, on peut (doit) ajouter un fichier déjà présent dans le dépôt s'il a été modifié et qu'on veut qu'il soit pris en compte dans le prochain commit !

défaire (undo) un ajout

La suppression de fichiers ou répertoires ajoutés et **non** validés est réalisée par : git reset HEAD fic1 rep2 fic3 ... Attention, ces fichiers ne sont pas supprimés de l'arborescence !

supprimer des fichiers et répertoires

```
git rm fic1 rep2 supprime récursivement les fichiers de
```

Validation

```
git commit [-a | -m <msg>] [<file>...]
```

- la validation permet de fabriquer une nouvelle version du projet ;
- par défaut, seuls les fichiers ajoutés, supprimés, ou déplacés par `git add|rm|mv` sont validés ;
- l'option `-a` (all) permet de prendre en compte les fichiers modifiés faisant déjà partie du dépôt ;
- on peut également indiquer nominativement les fichiers que l'on veut valider tandis que d'autres, pourtant modifiés, ne le seront pas !
- si l'on veut un fonctionnement à la SVN : `git commit -a`
- après un commit erroné (oubli de fichiers ou mauvais msg), `git reset` permet de revenir en arrière (comme après un mauvais `git add ...`) ;

Protocoles

Afin de collaborer, plusieurs dépôts distants peuvent coexister. 4 protocoles de communication peuvent être utilisés :

- `http(s)` : peut être utilisé anonymement ou avec une authentification (login, password) et un cryptage, comme par exemple github ; la saisie du couple (login, mot de passe) peut être évitée grâce à `git credential ...`
- `ssh` : protocole le **plus utilisé** d'URL `ssh://user@server/pathToProject.git` ou plus court `user@server/pathToProject.git`. Authentifié et crypté, `ssh` est efficace mais ne permet pas l'accès anonyme ;
- `git` : accès en lecture seulement sans authentification ;
- `local` : utile si disque commun à plusieurs utilisateurs (NFS) ;

git remote

Chaque dépôt distant a un nom (origin) et une url. (`git@...` ou `https://...` selon le protocole (`ssh` ou `https`)).

La commande `git remote` permet de gérer la liste des dépôts distants qui partagent certaines branches de votre projet.

ajouter un dépôt distant

```
git remote add origin git@gitlab.info-ufr.univ-montp2.fr:mi...
```

voir les dépôts distants

```
git remote
```

copier un dépôt distant

```
git clone
```

GitLab du SIF

GitLab est un serveur de version Git et un **site Web** permettant également :

- d'ajouter, éditer, valider des fichiers sur le dépôt distant ;
- d'éditer un Wiki ;
- de signaler des problèmes (Issue) et leur résolution ;
- de gérer des groupes et des permissions sur les projets ;

2 interfaces

- Web : administration, gestion du dépôt distant
- console : gestion de version courante sur le dépôt local

Utilisation de GitLab

- ① se connecter à l'interface web de GitLab (login et mot de passe de l'ENT);
- ② depuis le tableau de bord (*dashboard*), créer un nouveau projet vide (bouton + en haut à droite);
- ③ sur le poste client exécuter la liste des commandes affichées sur la page web selon que :
 - on démarre un nouveau projet vide : `git init`; `touch README`; `git add README`; `git commit -m 'premier commit'`; `git remote add origin git@gitlab.info-ufr.univ-montp2.fr:michel.meynard/bidon.git`; `git push -u origin master`
 - on utilise un **dépôt git déjà existant** : `cd mondepotlocal`; `git remote add origin git@gitlab.info-ufr.univ-montp2.fr:michel.meynard/bidon.git`; `git push -u origin master`

Protocoles de communication de GitLab

2 méthodes d'authentification sont possibles : via **ssh** ou via **https** :

SSH

Créer localement un couple de clés publique et privée :

```
ssh-keygen -t rsa -C "michel.meynard@univ-montp2.fr"
cat ~/.ssh/id_rsa.pub
```

copier la clé publique à la souris, dans GitLab sur le web, aller dans profile/SSH keys, cliquer sur add SSH Key, coller la clé publique

```
cd ~/.ssh/
chmod 0600 id_rsa
cd ~/test
git push -u origin master
```

Si en ligne de commande, gitLab demande un mot de passe, cela ne fonctionne pas car il faut utiliser l'authentification par ssh !

Protocoles de communication de GitLab - 2

HTTPS

Ce protocole, comme ssh, nécessite l'authentification à chaque commande git sur le dépôt distant (push, pull).

```
bidon$ git remote add origin https://gitlab.info-ufr.univ-montp2.fr/michel.meynard/bidon.git
bidon$ git push origin master
Username for 'https://gitlab.info-ufr.univ-montp2.fr': michel.meynard@univ-montp2.fr
Password for 'https://michel.meynard@univ-montp2.fr@gitlab.info-ufr.univ-montp2.fr':
Counting objects: 3, done.
...
```

Il est extrêmement pénible d'avoir à taper son login (mail institutionnel!) puis son mot de passe. On préférera donc le protocole **ssh**.

Les commandes indispensables

- `git init` crée un nouveau dépôt vide;
- `git clone` clone un dépôt distant;
- `git add` ajoute de nouveaux objets blobs dans la base des objets pour chaque fichier modifié depuis le dernier commit;
- `git pull` intègre les modifications des autres utilisateurs depuis le dépôt distant vers la copie locale (fetch puis merge);
- `git commit` valide les modifications locales sans les envoyer au dépôt;
- `git push` intègre les modifications locales sur le dépôt;
- `git branch` crée une nouvelle branche de développement;
- `git merge` fusionne plusieurs branches de développement;

MaGit sous Emacs

Emacs possède un mode natif de gestion de version générique (menu Tools/Version Control) qui n'est pas spécialisé pour Git et qui est difficile d'emploi ... Magit est un paquetage spécialisé !

- ① installer le package **magit** : M-x list-packages, sélectionner magit, install ;
- ② ouvrir un fichier ou répertoire situé dans une arborescence git ;
- ③ M-x magit-status pour obtenir l'état du dépôt local comme dans la figure suivante ;

Par la suite, la plupart des commandes magit seront réalisées via la frappe d'un seul caractère suivi éventuellement de paramètres tapés dans le mini-buffer. Pour voir la liste des commandes, taper "?" pour voir le buffer du bas de la figure suivante...

magit-status

magit commandes usuelles

Commandes courantes à effectuer sur une ligne d'une section :

- s permet d'ajouter ce fichier pour le prochain commit (git add) ; possible sur les fichiers non suivis (untracked) et sur les fichiers changés de type modified ; (to stage : mettre en scène, organiser)
- s sur les fichiers changés de type deleted, permet de supprimer (git rm) le suivi de ces fichiers ;
- i permet d'ignorer les fichiers non suivis en modifiant le fichier .gitignore ; sur les fichiers suivis (modified or deleted) permet de supprimer (git rm) et de les ignorer ;
- c (commit) permet de saisir un message de validation à valider par C-c C-c ; remarquons que l'option -a est positionné par défaut !
- P permet de pusher (menu Magit/Push) ; (F) pull (menu Magit/Pull) ...

Tab visualise les modifs effectuées

magit-log

Depuis magit-status, en tapant l (log) puis l on obtient un journal abrégé des différents commits :

Références

Git book <http://git-scm.com/book/en/v2>

git help en ligne de commande : `git help add`
tutoriel git `man gittutorial`