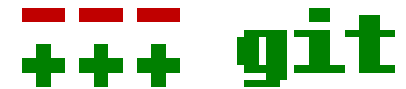


---

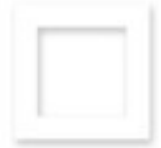
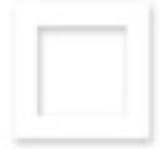
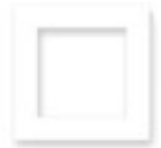
# Gestion de code source avec Git

Grégory Colpart <reg@evolix.fr>



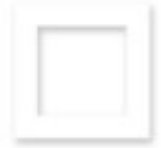
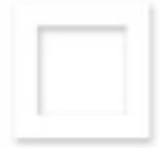
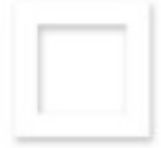
## Sommaire :

- Introduction à la gestion de code source
- Présentation de Git
- Git en action
- (Questi | discussi | boiss)ons



**evolix**

<http://www.evolix.fr/>



# Introduction à la gestion de code source

---

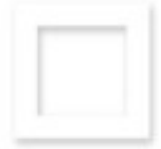
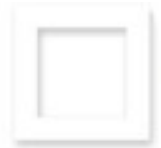
*Gestion de code source* est une traduction « barbare » pour :

- VCS (Version Control System) ou
- SCM (Source Code Management)

Un VCS permet de (sauve)garder l'historique d'un projet et d'y accéder + ou - facilement. Il permet souvent un développement collaboratif en minimisant les conflits entre les modifications de chaque développeur.

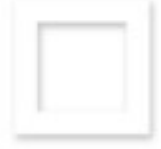
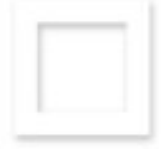
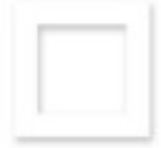
Exemples de VCS :

- CPOLD : *cp main.c main.c.old ;-*)
- CVS, SVN, GNU Arch
- Git, Mercurial, Bazaar



evolix

<http://www.evolix.fr/>



# Introduction à la gestion de code source

---

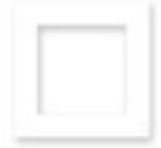
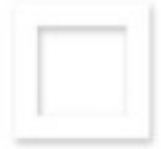
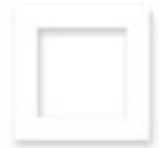
## Fonctions basiques d'un VCS :

- Un dépôt contenant l'historique de tous les fichiers : *repository*
- Ajouter des fichiers ou modifications de fichiers : *commit*
- Ajouter/supprimer/renommer des fichiers/répertoires
- Afficher des modifications entre différents états : *diff*
- Marquer tout un code source dans un état précis : *tag*
- Créer des branches
- Récupérer des modifications d'autres branches et (tenter de) les appliquer : *merge*
- Revenir en arrière : *revert*
- Mettre à jour un projet : *update*
- Récupérer un projet : *checkout* (ou *clone*)

# Présentation de Git

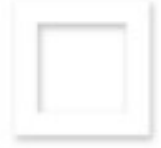
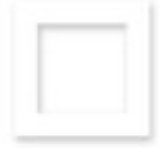
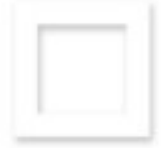
---

- Créé sur mesure pour le développement du noyau Linux en remplacement de *BitKeeper*
- Première version publiée en 2005
- Peut être aussi considéré comme une sorte de système de fichiers ou de format de stockage !
- VCS distribué : il n'y pas obligatoirement de serveur de référence. Chaque développeur récupère la totalité d'un repository.
- Officiellement orienté vers la rapidité  
“*Fast version control system*”



**evolix**

<http://www.evolix.fr/>

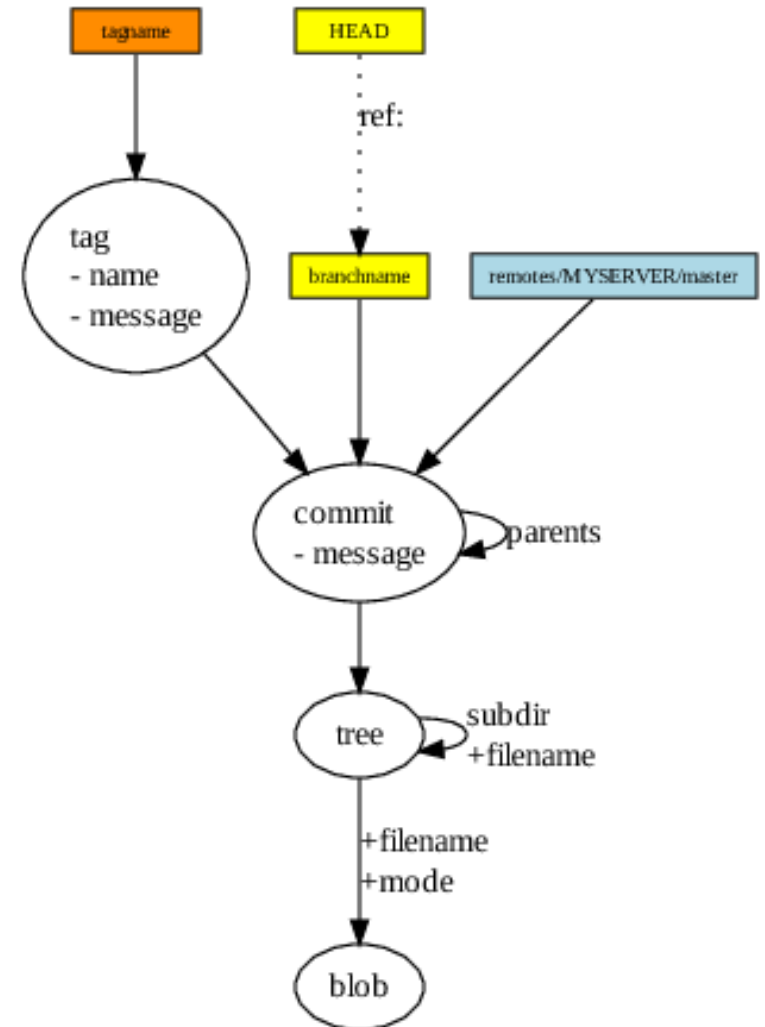


# Présentation de Git

## Fonctionnement interne de Git :

Git stocke l'historique des fichiers sous forme d'objets pouvant être des diffs/deltas compressés (*blob*), des métadatas (noms de fichiers, permissions, etc.) associées aux deltas (*tree*), et enfin un lien (*commit*) permettant de référencer des états et de les ordonner.

Une branche est tout simplement un lien vers un commit !



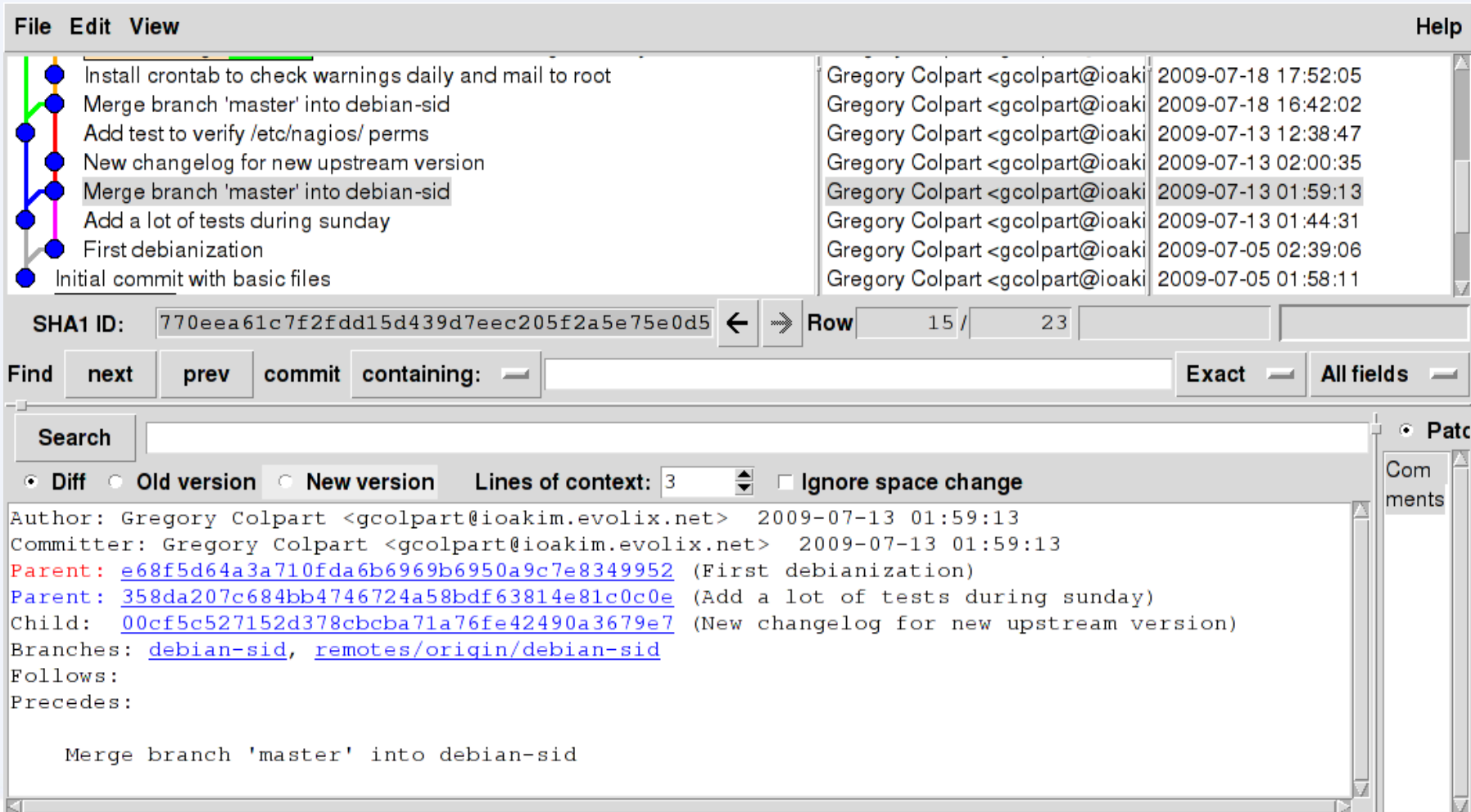
evolix

<http://www.evolix.fr/>

# Présentation de Git

Chaque *commit* est identifié par un identifiant SHA1.

On note aussi les informations *Parent:* et *Child:* qui permettent de situer un *commit* dans l'historique.



The screenshot shows a Git GUI interface with a commit history list and a detailed view of a specific commit.

**Commit History:**

Commit Message	Author	Date
Install crontab to check warnings daily and mail to root	Gregory Colpart <gcolpart@ioaki>	2009-07-18 17:52:05
Merge branch 'master' into debian-sid	Gregory Colpart <gcolpart@ioaki>	2009-07-18 16:42:02
Add test to verify /etc/nagios/ perms	Gregory Colpart <gcolpart@ioaki>	2009-07-13 12:38:47
New changelog for new upstream version	Gregory Colpart <gcolpart@ioaki>	2009-07-13 02:00:35
Merge branch 'master' into debian-sid	Gregory Colpart <gcolpart@ioaki>	2009-07-13 01:59:13
Add a lot of tests during sunday	Gregory Colpart <gcolpart@ioaki>	2009-07-13 01:44:31
First debianization	Gregory Colpart <gcolpart@ioaki>	2009-07-05 02:39:06
Initial commit with basic files	Gregory Colpart <gcolpart@ioaki>	2009-07-05 01:58:11

**SHA1 ID:** 770eea61c7f2fdd15d439d7eec205f2a5e75e0d5

**Row:** 15 / 23

**Find:** next prev commit containing: [ ] Exact All fields

**Search:** [ ] Patc

**Diff:**  Diff  Old version  New version Lines of context: 3  Ignore space change

**Author:** Gregory Colpart <gcolpart@ioakim.evolix.net> 2009-07-13 01:59:13  
**Committer:** Gregory Colpart <gcolpart@ioakim.evolix.net> 2009-07-13 01:59:13  
**Parent:** [e68f5d64a3a710fda6b6969b6950a9c7e8349952](#) (First debianization)  
**Parent:** [358da207c684bb4746724a58bdf63814e81c0c0e](#) (Add a lot of tests during sunday)  
**Child:** [00cf5c527152d378cbcb71a76fe42490a3679e7](#) (New changelog for new upstream version)  
**Branches:** [debian-sid](#), [remotes/origin/debian-sid](#)  
**Follows:**  
**Precedes:**

Merge branch 'master' into debian-sid

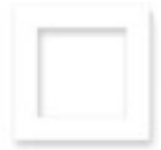
# Présentation de Git

---

Toutes les informations sur un repository Git sont stockés dans un répertoire **.git/** à la racine du projet.

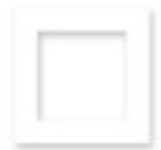
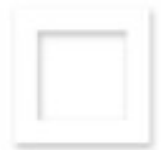
```
$ ls -a ~/GIT/project/  
.. foo .git bar
```

```
$ ls -lR ~/GIT/project/.git/  
./config  
./HEAD : la branche courante  
./hooks/ : hooks precommit, postcommit, etc.  
./objects/: tous les objets  
./refs/heads/ : les branches locales  
./refs/remotes/ : les branches distantes  
./refs/tags/ : les tags  
etc.
```



**evolix**

<http://www.evolix.fr/>



# Git en action

---

Il est **très** facile de tester Git. Un binaire git, les manpages et un /tmp sont l'idéal :

```
$ cd /tmp/ && mkdir project.git && cd project.git
```

```
$ man git-init
```

```
$ git init
```

```
$ ls -a
```

```
... .git
```

```
$ echo foo > bar
```

```
$ git status
```

```
Untracked files: bar
```

```
$ git add bar
```

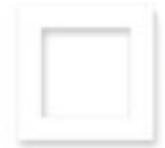
```
Changes to be committed: newfile: bar
```

```
$ git commit -m "First Commit"
```

```
Created initial commit fccf929: First Commit
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```

```
create mode 100644 bar
```



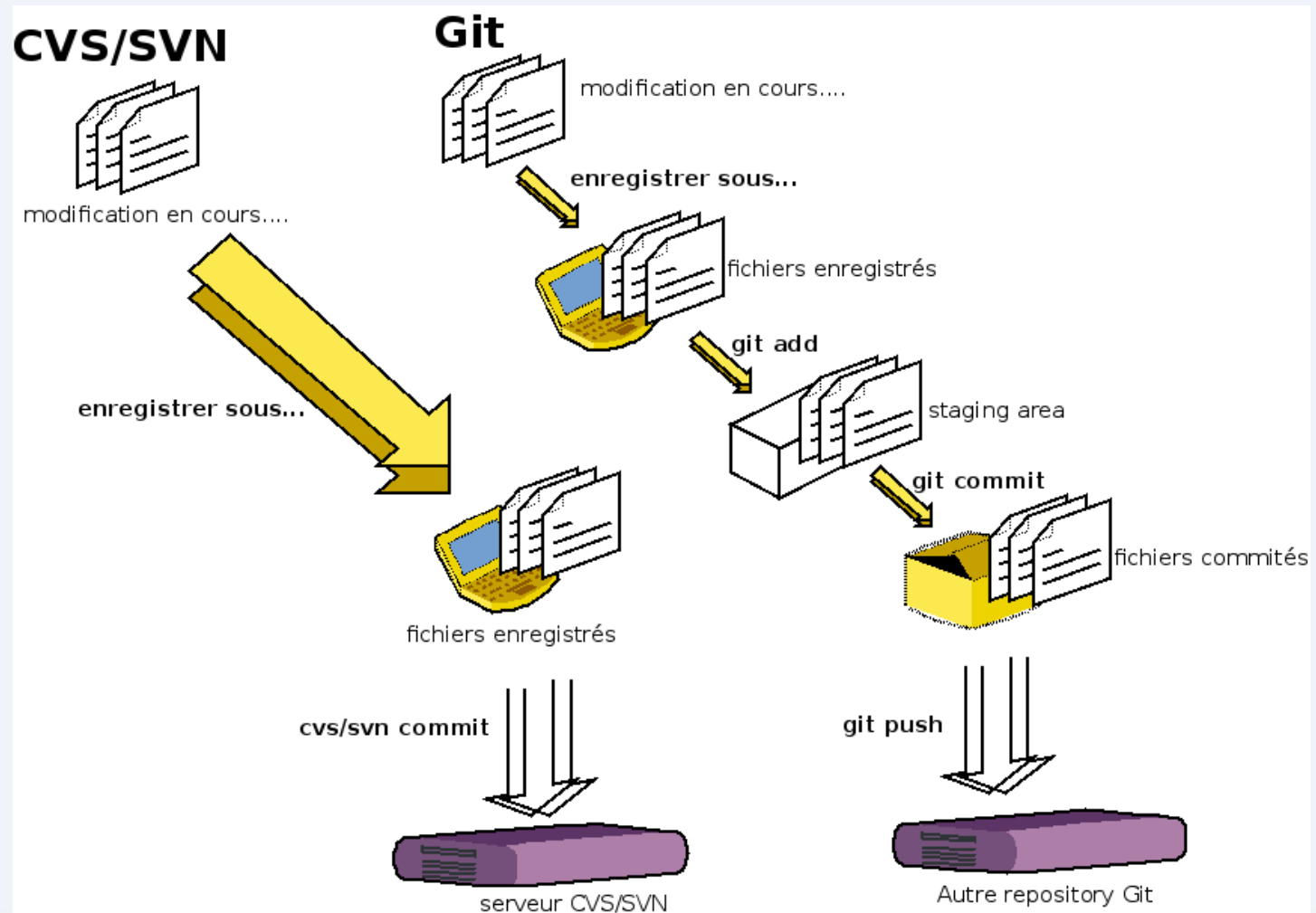
**evolix**

<http://www.evolix.fr/>



# Git en action

<À Mon Humble Avis>Il peut être un peu difficile d'apprendre Git... surtout lorsque l'on connaît déjà un VCS non distribué (Git, SVN). Néanmoins, cela permet d'apprendre de nouveaux concepts qui permettront de gagner en efficacité et en qualité.</>



# Git en action : entre repository

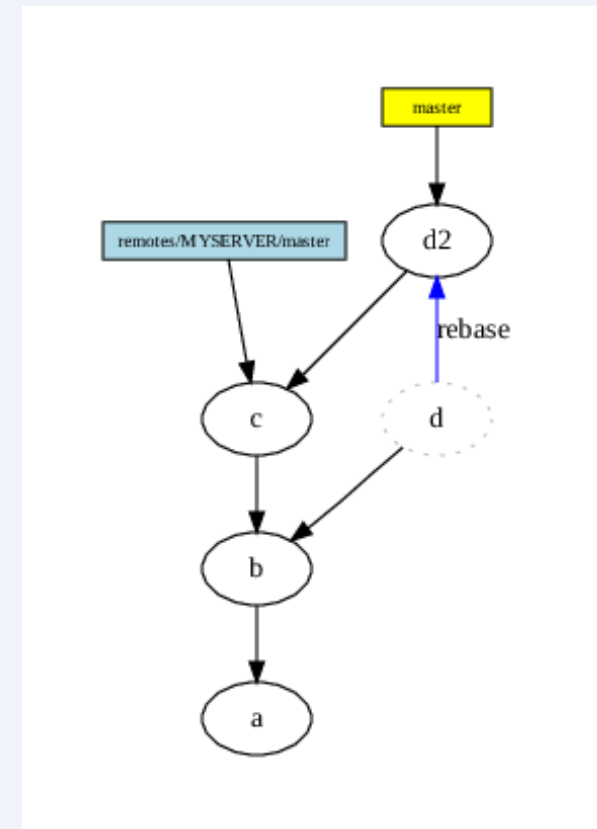
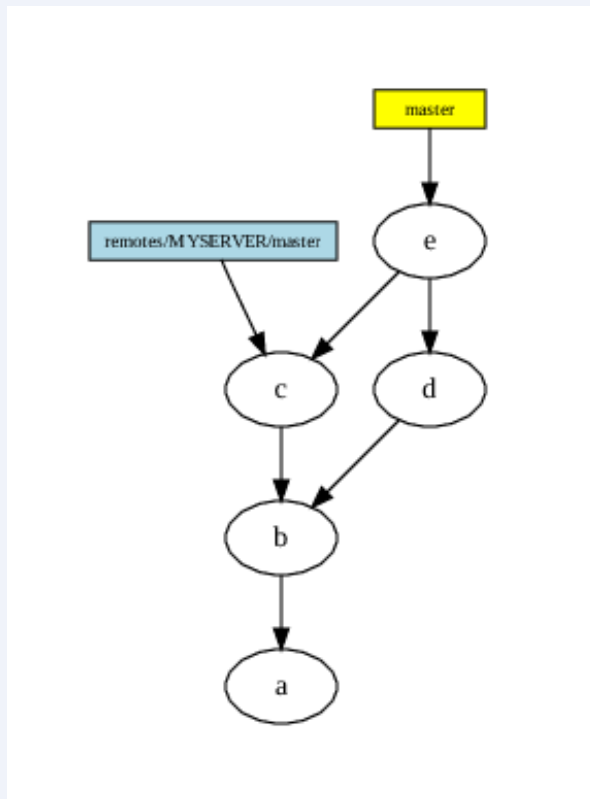
Protocoles supportés : ssh:// /path/to/ git:// http://

Récupération initiale: *git clone*

Mise-à-jour de son repository : *git pull*

Envoi des modifications : *git push*

Principe du *git rebase* :

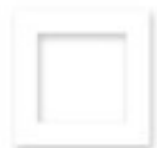
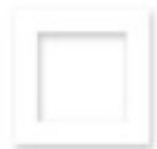
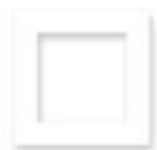


<http://www.evolix.fr/>

# Git en action : les branches

---

- Création d'une branche : ***git branch mytest master***
- On “est” dans une branche : ***git checkout mytest***  
(Dans un shell, il est pratique d'afficher la branche courante)
- Renommer une branche : ***git branch -m mytest mysupertest***
- Supprimer une branche : ***git branch -d mysupertest***
- La récupération d'un repository crée des branches distantes, souvent référencées en : *remotes / origin / master*, etc.
- Si l'on veut travailler dessus, les branches distantes sont *trackées* avec une branche locale : `git config => .git/config`
- Utiliser vraiment un VCS, c'est utiliser des branches et faire des merge, Git est fait pour ça : donc... soyez branchés :-)



**evolix**

<http://www.evolix.fr/>



# Git en action : un historique propre

---

Git permet facilement de ré-écrire son historique. Cela permet d'avoir un repository « propre » constitué d'un ensemble de diffs significatifs et non pollués.

*git reset HEAD^*

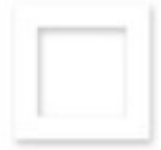
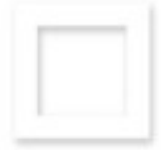
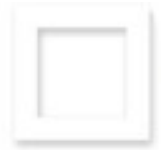
*git commit --amend*

*git rebase -i HEAD~N*

*[no branch] git commit --amend*

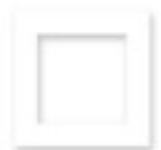
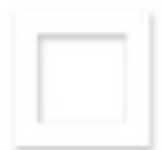
*[no branch] git rebase --continue*

Le principe est de préparer ses diffs sur sa machine. Une fois publié (si publication il y a !), il ne faut plus les toucher.



**evolix**

<http://www.evolix.fr/>

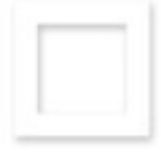
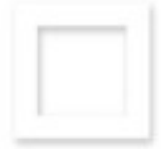
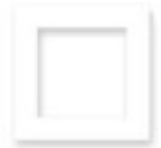


# Git en action : organisation

---

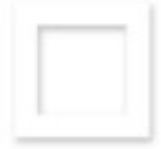
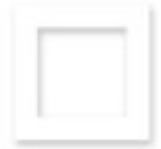
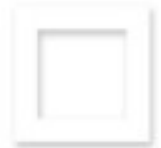
Décentralisé ne veut pas forcément dire ...qu'il n'y a pas de repository central. On distingue plusieurs organisations pour utiliser Git de façon collaborative :

- Un repository de “publication” accessible à tous les développeurs en écriture,
- Un “release manager” qui gère seul (souvent par mail) l'accès à un repository ...accessible aux développeurs uniquement en lecture,
- Un véritable organisation décentralisée où les échanges se font directement entre les développeurs (pull/push)



**evolix**

<http://www.evolix.fr/>



# Git en action : divers

---

Importer CVS/SVN/Arch/etc. dans Git :

git-cvsmimport, git-svnimport, git-archimport, etc.

Travailler dans un repository SVN avec Git :

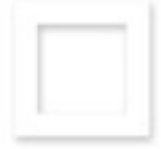
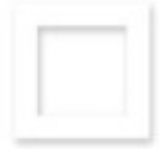
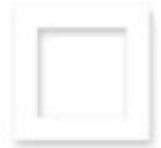
git-svn permet de travailler avec un repository local Git et de les envoyer dans un repo SVN

Git, un filesystem versionné ?

Utilisation pour /etc/, backups SQL, etc.

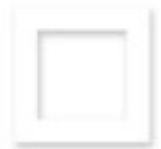
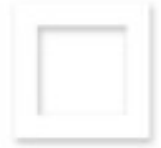
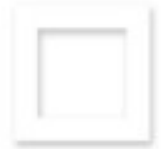
Outils ?

Gitweb, gitk, Trac, Egit (Eclipse), git\_remote\_branch, etc.  
repo.or.cz, GitHub, SourceForge, forge.evolix.org, etc.



**evolix**

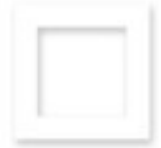
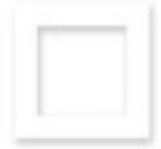
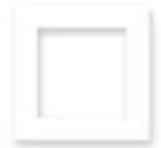
<http://www.evolix.fr/>



# Git en action : I like that !

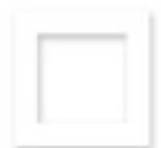
---

- `git clean -d -f`
- `git reset --hard`
- `git add -i / git add -p`
- `git-rebase -i`
- `git cherry-pick`
- `git stash`
  
- Et vous ?



**evolix**

<http://www.evolix.fr/>



# That's all folks !

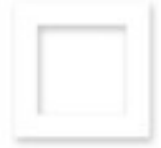
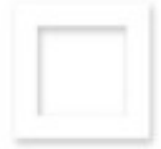
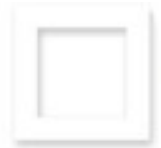
---

Ressources :

<http://www.git-scm.com/>

[http://en.wikipedia.org/wiki/Git\\_\(software\)](http://en.wikipedia.org/wiki/Git_(software))

<http://www.kernel.org/pub/software/scm/git/docs/>



**evolix**

<http://www.evolix.fr/>

