

Une courte introduction à la Gestion de Versions

Amélie Cordier, Catarina Ferreira da Silva, Hind Benfenatki



Préambule / Preamble

(parce que les commandes sont en anglais,
et parce que ce cours est un cours à trous...)

Systèmes de gestion de version

=

Version Control System (VCS)

Attention : ne pas confondre VCS et CVS...

Pourquoi ?

Les **VCS** en questions





VCS : qu'est-ce que c'est ?

Des **protocoles** pour la sauvegarde (**backup**), le suivi (**tracking**) et la synchronisation (**merge**) de *fichiers*

Des **outils** permettant d'utiliser ces protocoles

Pour quels fichiers ?

Tous !

Le code source, mais aussi la documentation, les fichiers de configuration, les pages HTML, les compte-rendu de TP, les rapports, votre CV, des images, le cahier des charges de votre projet tuteuré, etc.

Par l'exemple

Protocoles	En ligne de commande	Outils « standalone »	Outils intégrés
CVS	cvs	SmartCVS, etc.	Plugins dans de nombreuses applications (Eclipse, Netbeans, etc.)
Subversion	svn	Tortoise, etc.	
GIT	git	SmartGIT, TortoiseGIT, etc.	
Mercurial	hg	TortoiseHG, Diffuse, SCM, etc.	



Mais à quoi ça sert ?

À sauvegarder des fichiers

À conserver un historique des modifications
apportées aux fichiers

À visualiser la différence entre deux « versions »
d'un fichier

À revenir facilement en arrière

À faciliter le travail à plusieurs (travail collaboratif)

Et aussi à ... ?



Pourquoi utiliser un VCS ?

Parce que faire ses sauvegardes sur une clé USB ou en s'envoyant les fichiers par mails **n'est pas une bonne idée**

Parce qu'il n'existe pas de meilleure solution pour **travailler en équipe** sur un projet commun !



Comment ça marche ?

Deux approches : **centralisées** et **distribuées**

Dans les deux cas, les principes suivants s'appliquent :

Les **modifications** apportées à un fichier sont enregistrées de sorte à ce que l'on puisse à tout moment revenir à n'importe quelle « version » antérieure du fichier

Les modifications sont partagées entre les différents membres du projet selon une **politique** définie au préalable

Quel est l'intérêt d'enregistrer les modifications ?



À votre avis, quelle est la différence entre
« enregistrer les modifications d'un fichier »
et
« enregistrer chaque version du fichier » ?

Comment travailler à plusieurs sur un même fichier ?



À votre avis ?

Politiques d'édition concurrente de fichiers

Opération atomique : une opération atomique est une opération qui garantit que le système sera dans un état stable après son exécution (comme une transaction en BD)

Politique de « lock » : placer un verrou sur un fichier lorsqu'on l'édite

Politique de « merge » : plusieurs utilisateurs peuvent éditer un fichier en même temps. Au moment de la sauvegarde, on fusionne les modifications et on résout les conflits.

Les + et - de lock et merge

Politique	Avantages	Inconvénients
Lock	<p>Pas de conflits</p> <p>Simple à mettre en oeuvre</p>	<p>Nuit à la productivité</p> <p>Si le lock reste trop longtemps, risque d'édition « en parallèle »</p>
Merge	<p>Permet une véritable édition concurrente</p> <p>Gain de temps et de productivité (on peut travailler « en même temps »)</p>	<p>Les conflits doivent être résolus avant la sauvegarde</p> <p>Les conflits sont parfois difficiles à résoudre</p>



Et en pratique ?

(exemple d'une approche centralisée)

Chaque utilisateur travaille sur sa copie locale (**working copy**)

Lorsqu'il le souhaite, il envoie (**commit**) ses modifications au dépôt (**repository**)

À tout moment, il peut récupérer (**update**) les modifications récentes effectuées par les autres utilisateurs et disponibles sur le dépôt

À vous de faire le schéma !

Un peu de vocabulaire sur la gestion de version



Différences entre **Version** et **Révision**

Révision : à chaque fois qu'un utilisateur notifie au système de gestion de version qu'un fichier a été modifié, le système crée une nouvelle **révision** de ce fichier et lui affecte **automatiquement** un numéro.

Version : elle est définie par l'utilisateur pour repérer un état particulier de l'ensemble des fichiers (par exemple, un état stable). Une version indique, pour chaque fichier, quel numéro de révision il faut considérer.

À retenir : les révisions sont numérotées automatiquement, les versions sont définies par les utilisateurs.

Pour aller plus loin avec les versions

En général, on numérote les versions avec deux nombres :

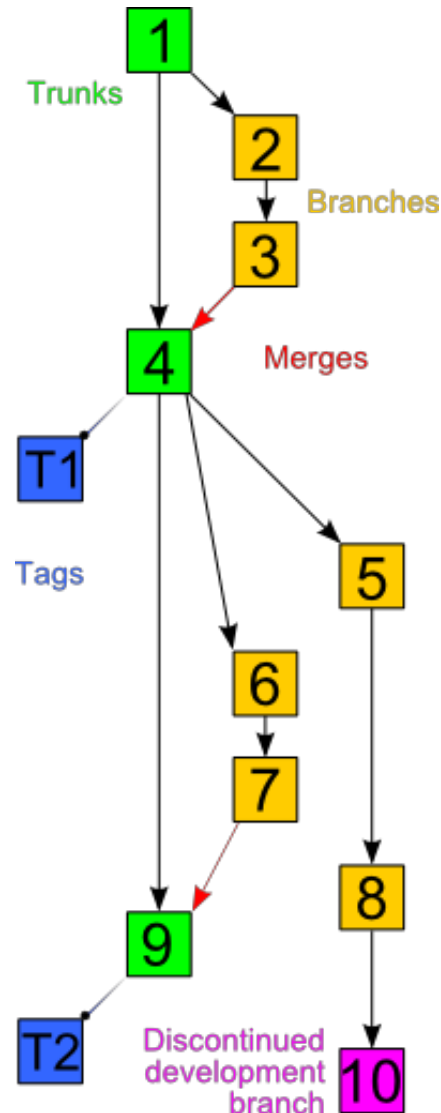
MonProjet v6.12

Le premier correspond à la **majeure**, et le second à la **mineure**

Version majeure : incompatibilités avec la version précédente

Version mineure : ajout de nouvelles fonctionnalités, mais pas d'incompatibilités majeures à signaler

Trunk, Branch, Tags, Merge ?



Les tags permettent de donner des noms symboliques à des versions.

Exemple : la version 11.10 d'Ubuntu s'appelle « Oneiric Ocelot »

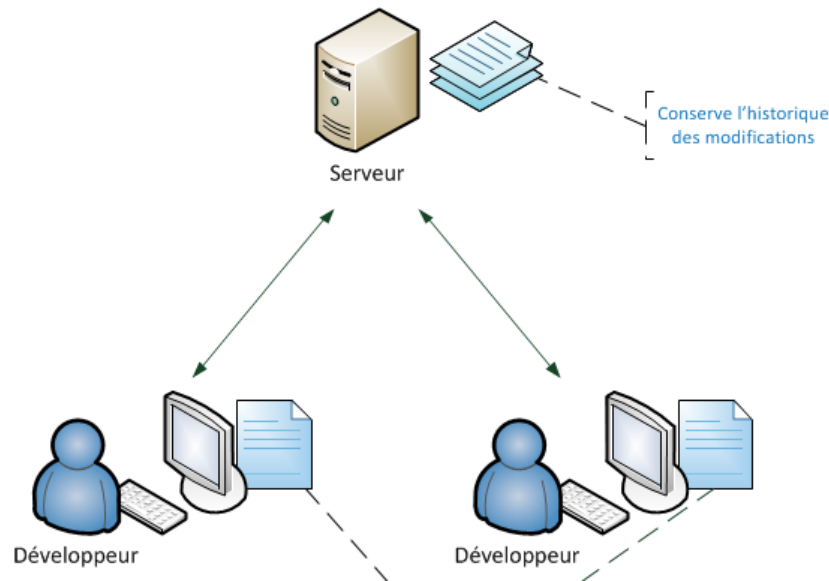
Pourquoi faire des branches ?

Pour mieux organiser le travail, et être plus efficaces !

Bonne pratique

- Sur le trunk : correction des bugs
- Sur les branches : développement de nouvelles fonctionnalités
- Quand des versions stables sont atteintes : fusion (**merge**) d'une ou plusieurs branches avec le tronc.

Comprendre le fonctionnement d'un VCS centralisé



VCS Centralisé

Principes :

- Un serveur central
- Des clients se connectent au serveur

*L'ensemble des illustrations qui suivent sont copiées de l'**excellent** guide « A Visual Guide to Version Control »*

<http://betterexplained.com/articles/a-visual-guide-to-version-control/>

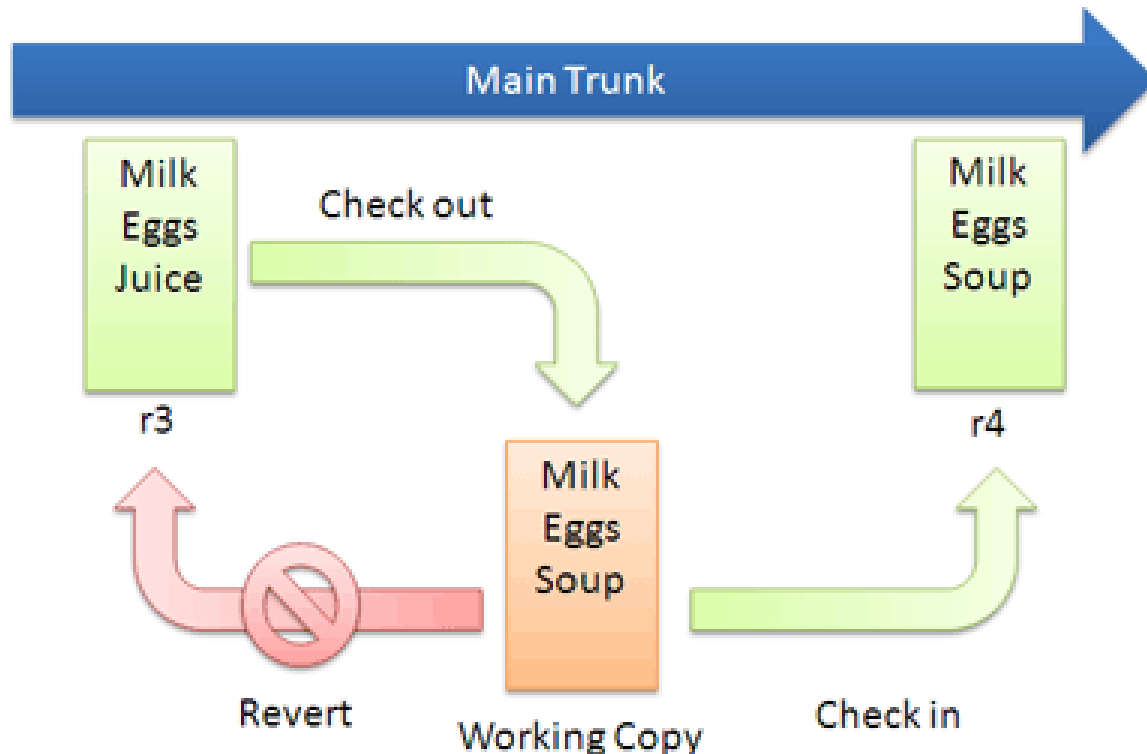
Les révisions



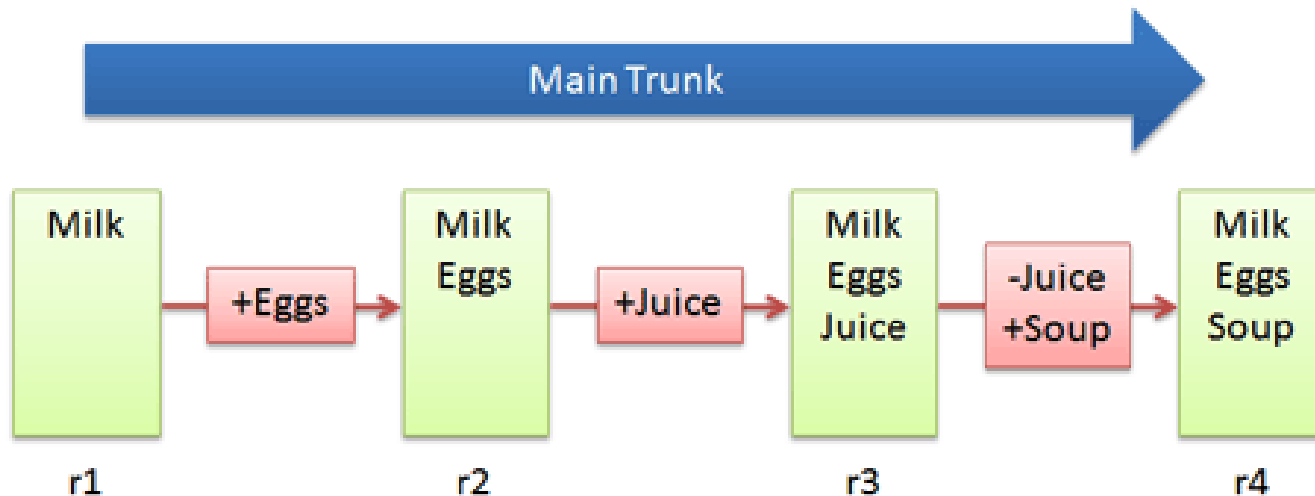
Chaque ajout ([commit](#)) d'un fichier par un utilisateur produit une révision, qui porte un numéro unique

Copie de travail

L'utilisateur peut récupérer la version du serveur (**checkout**) pour l'éditer. Il peut aussi décider de revenir à une version antérieure du fichier (**revert**)



La commande diff



Grâce à la commande **diff**, on peut voir à tout moment les différences entre deux versions d'un même fichier

Un exemple de diff avec SVN

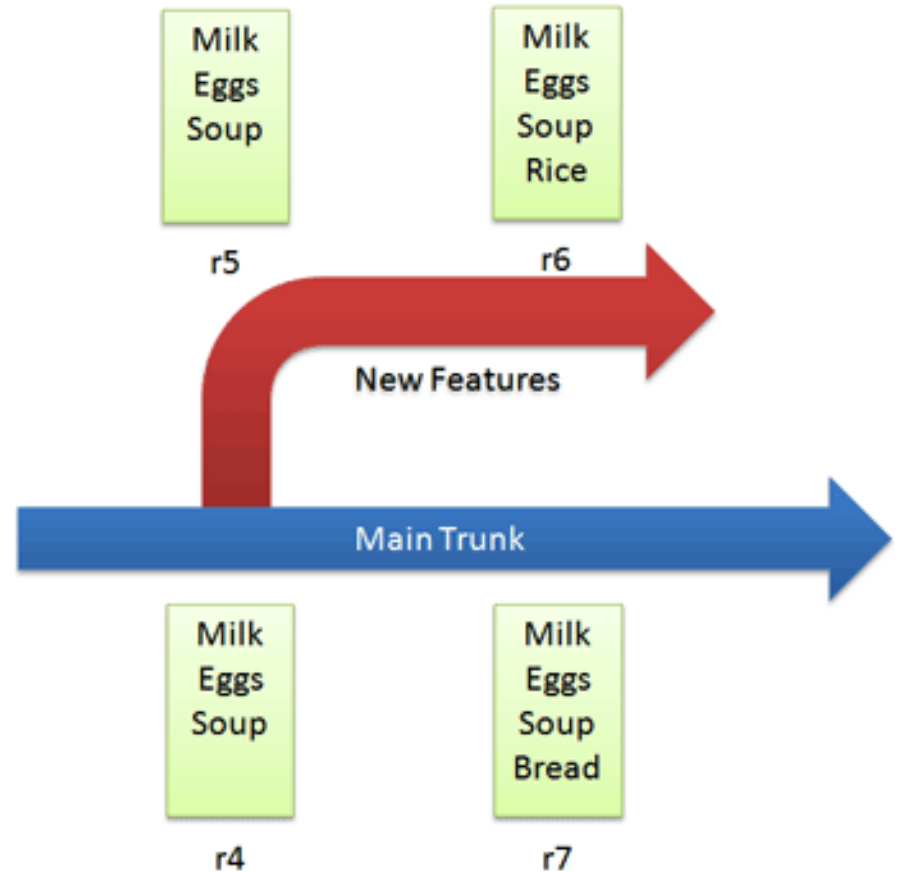
```
$ svn diff helloYou
Index: helloYou
=====
--- helloYou (revision 134)
+++ helloYou (working copy)
@@ -1,1 @@
-avant +après
```

Dans le fichier
« helloYou », j'ai retiré le
mot « avant » et je l'ai
remplacé par « après »

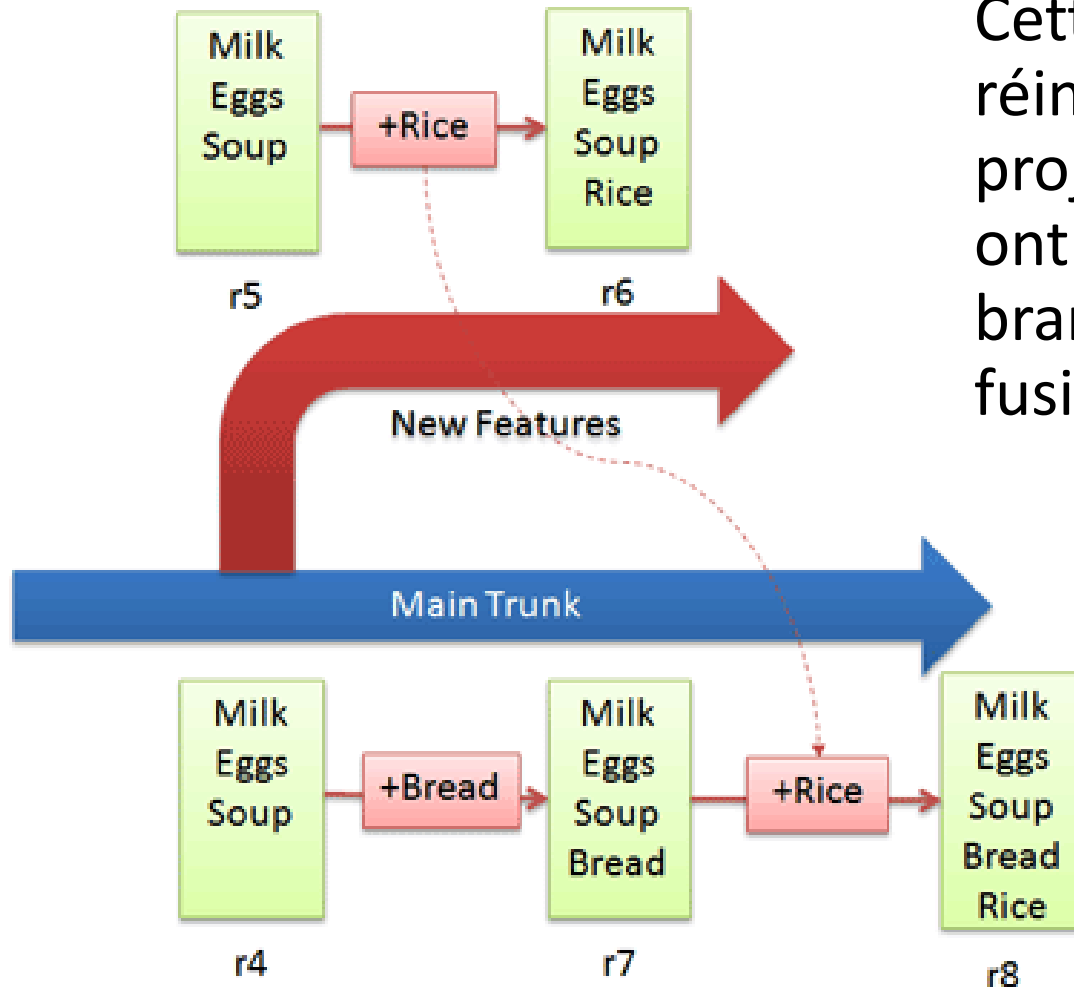
Que va-t-il se passer si
je fais un commit de ce
fichier ?

La notion de branche en images

Attention : la numérotation des révisions ne dépend pas de la branche dans laquelle on se trouve !

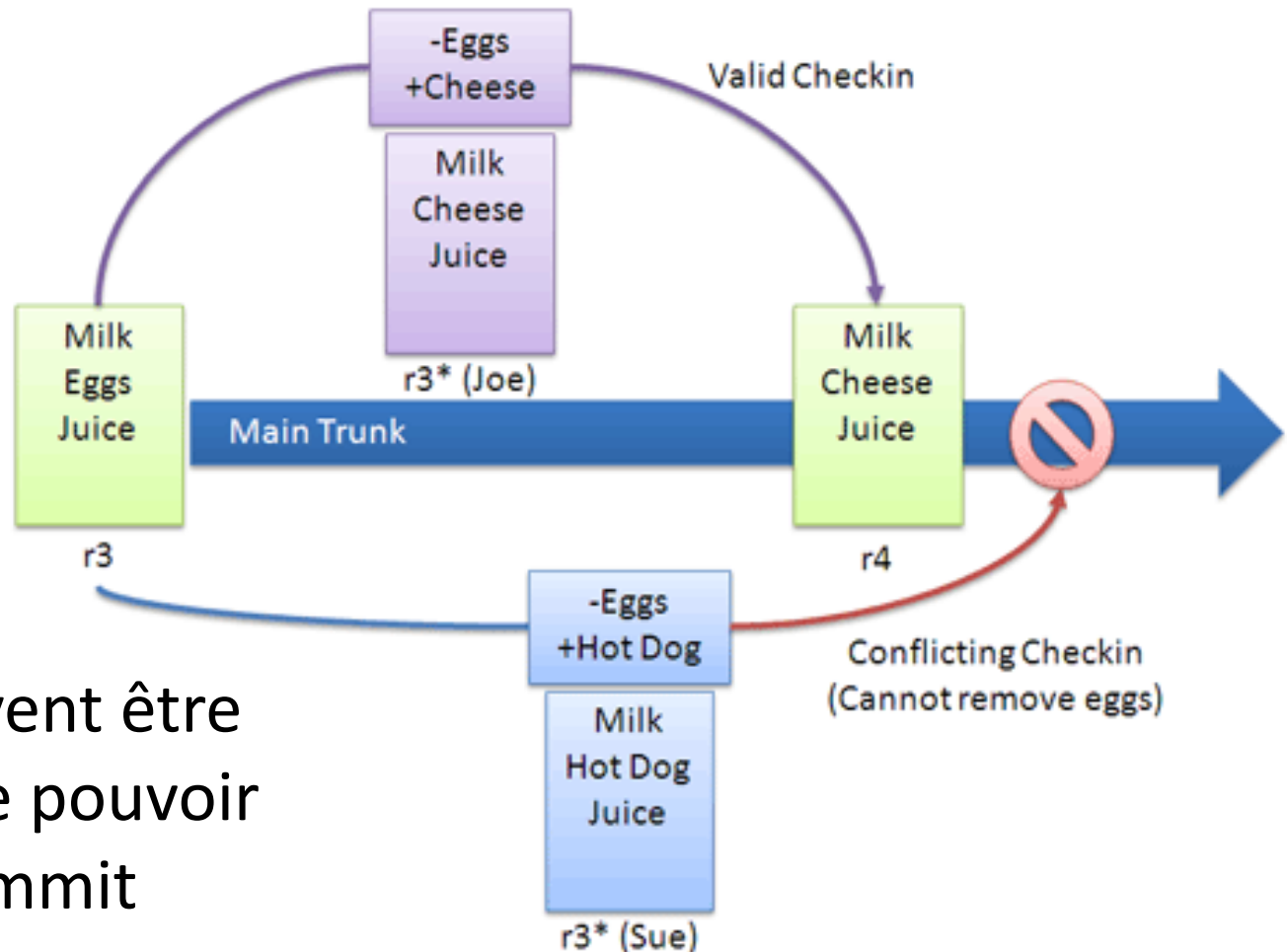


Fusion de plusieurs versions



Cette opération consiste à réintégrer dans le tronc du projet des modifications qui ont été faites dans une branche : on parle alors de fusion ([merge](#))

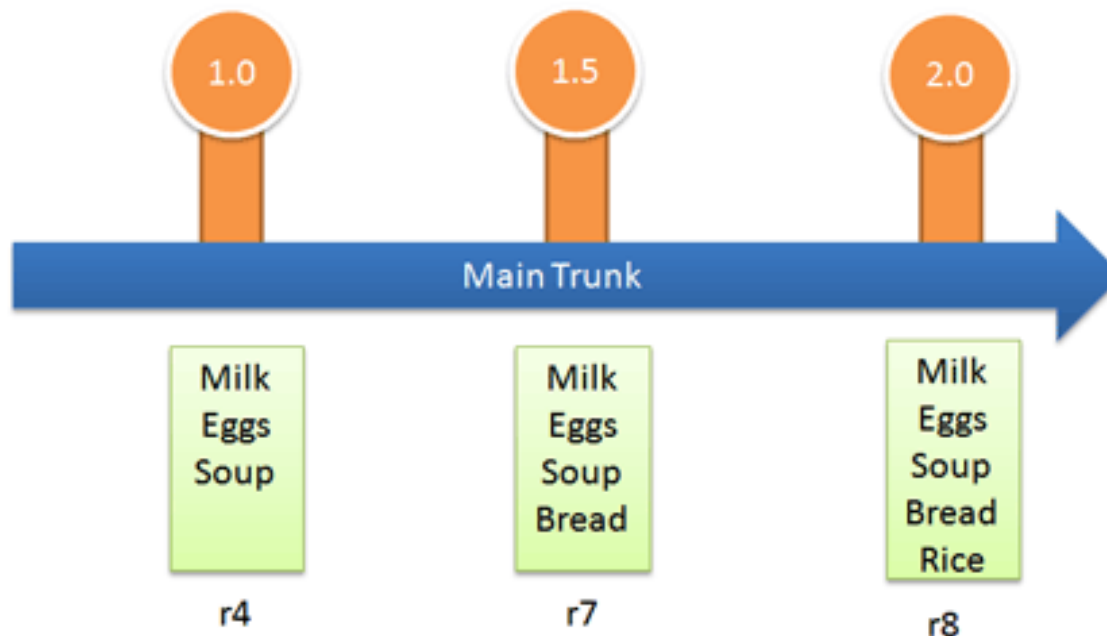
Gestion des conflits



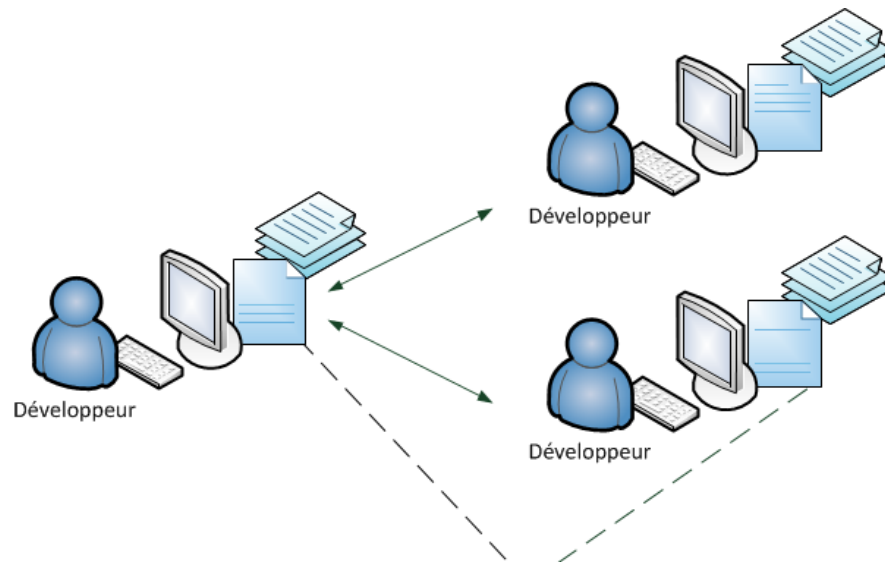
Les conflits doivent être résolus avant de pouvoir effectuer un commit

Utilisation des tags



Les tags (ou labels) permettent de retrouver facilement des versions que l'on souhaite retenir



Comprendre le fonctionnement d'un DVCS (VCS décentralisé ou distribué)



Les + et les - des outils centralisés ?

- + Simple d'utilisation
- + Sauvegarde systématique sur « une autre machine »
- Manque de souplesse pour gérer des « gros » projets
- Processus de fusion (**merge**) délicat 
- Connexion au serveur nécessaire pour chaque **commit** 

Pourquoi devrais-je utiliser un DVCS ?

Parce que c'est de plus en plus répandu

Parce que qui peut le plus peut le moins

Parce que c'est très geek 😊

Et parce que c'est ce que l'on va utiliser en TP...

VCS Distribué (DVCS)

Principes :

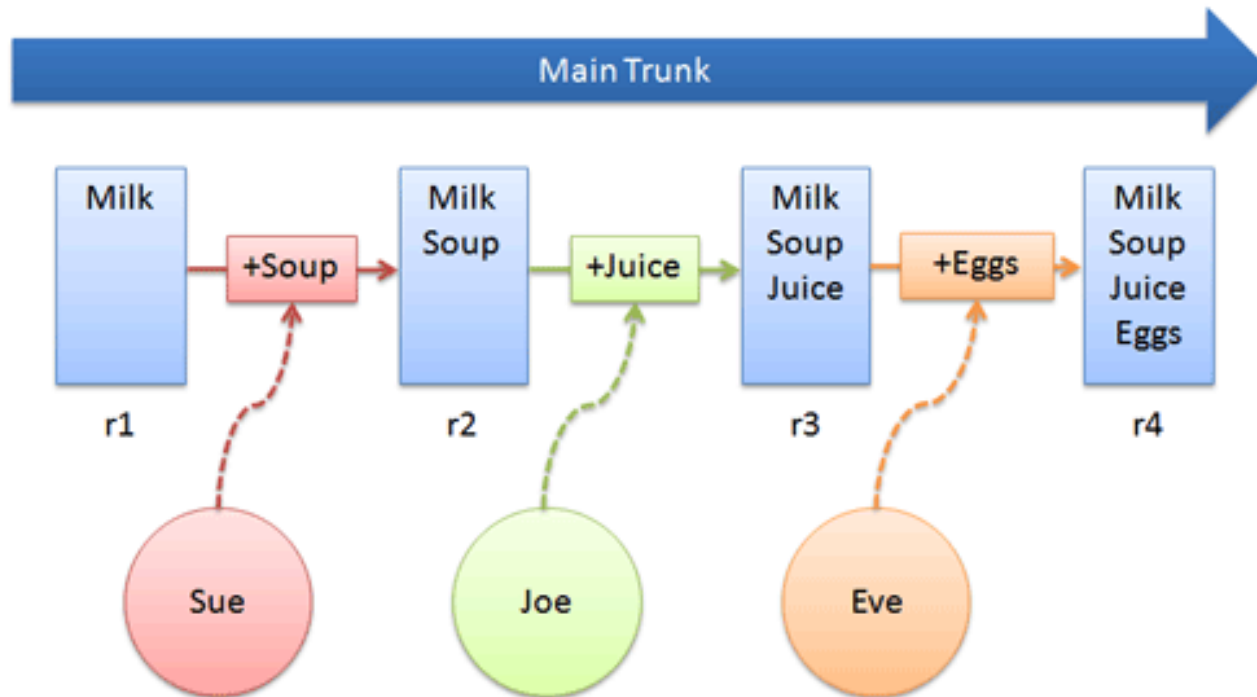
- Pas de serveur central
- Chaque utilisateur dispose de son propre dépôt (**repository**)
- On peut néanmoins mettre en place un serveur...

*L'ensemble des illustrations qui suivent sont copiées de l'**excellent** guide
« A Visual Guide to Distributed Version Control »*

<http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/>

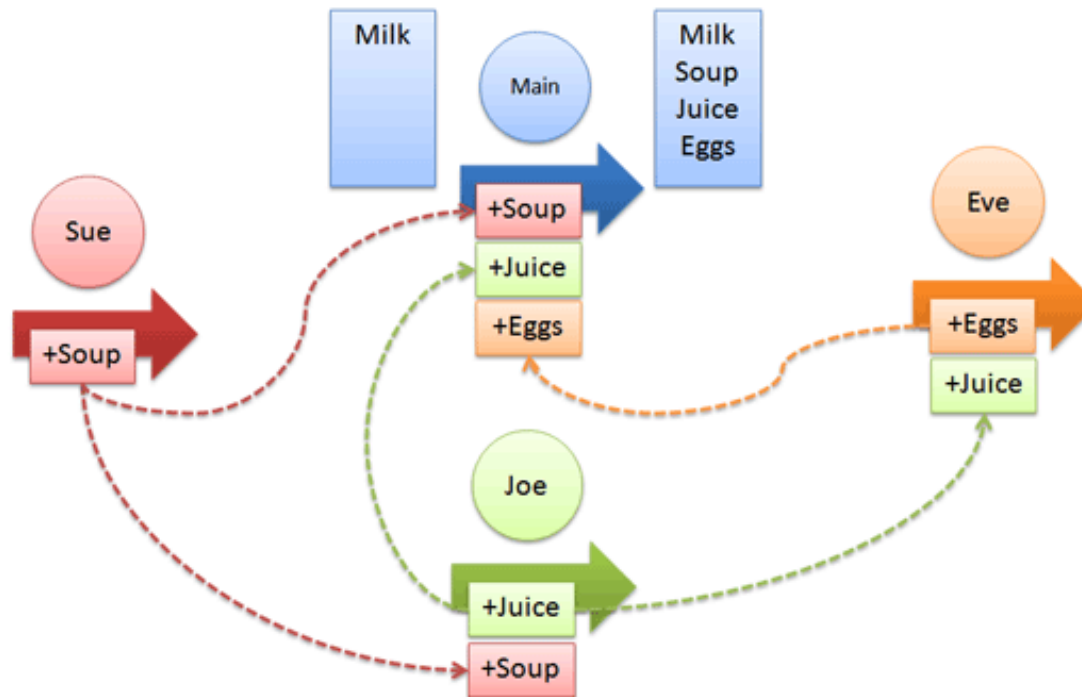
Rappel : VCS

Dans les approches centralisées, tous les utilisateurs se réfèrent au même dépôt (repository) distant



DVCS : partage !

Chaque utilisateur dispose de son propre dépôt, en local

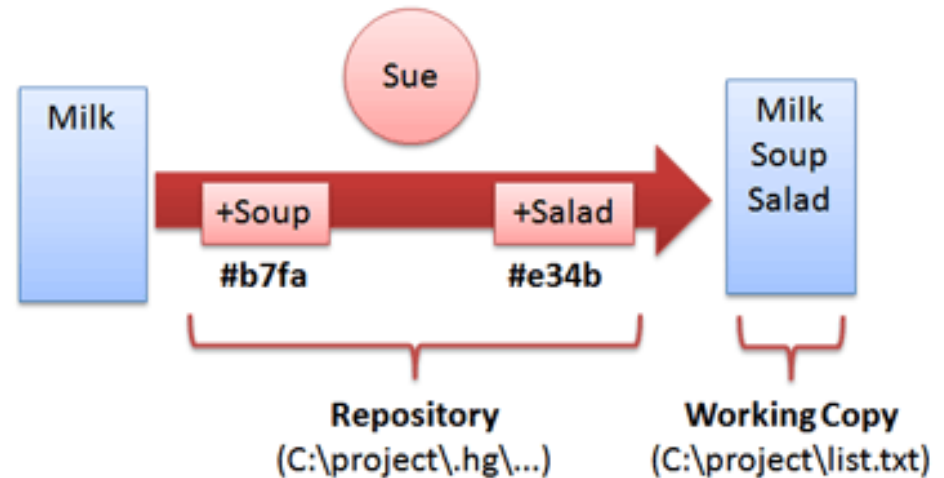


Le système repose sur le partage : chacun partage ce qu'il veut avec qui il veut

DVCS du point de vue d'un utilisateur

Chaque utilisateur dispose de son propre dépôt (**repository**) et de sa propre numérotation des versions (affreuse)

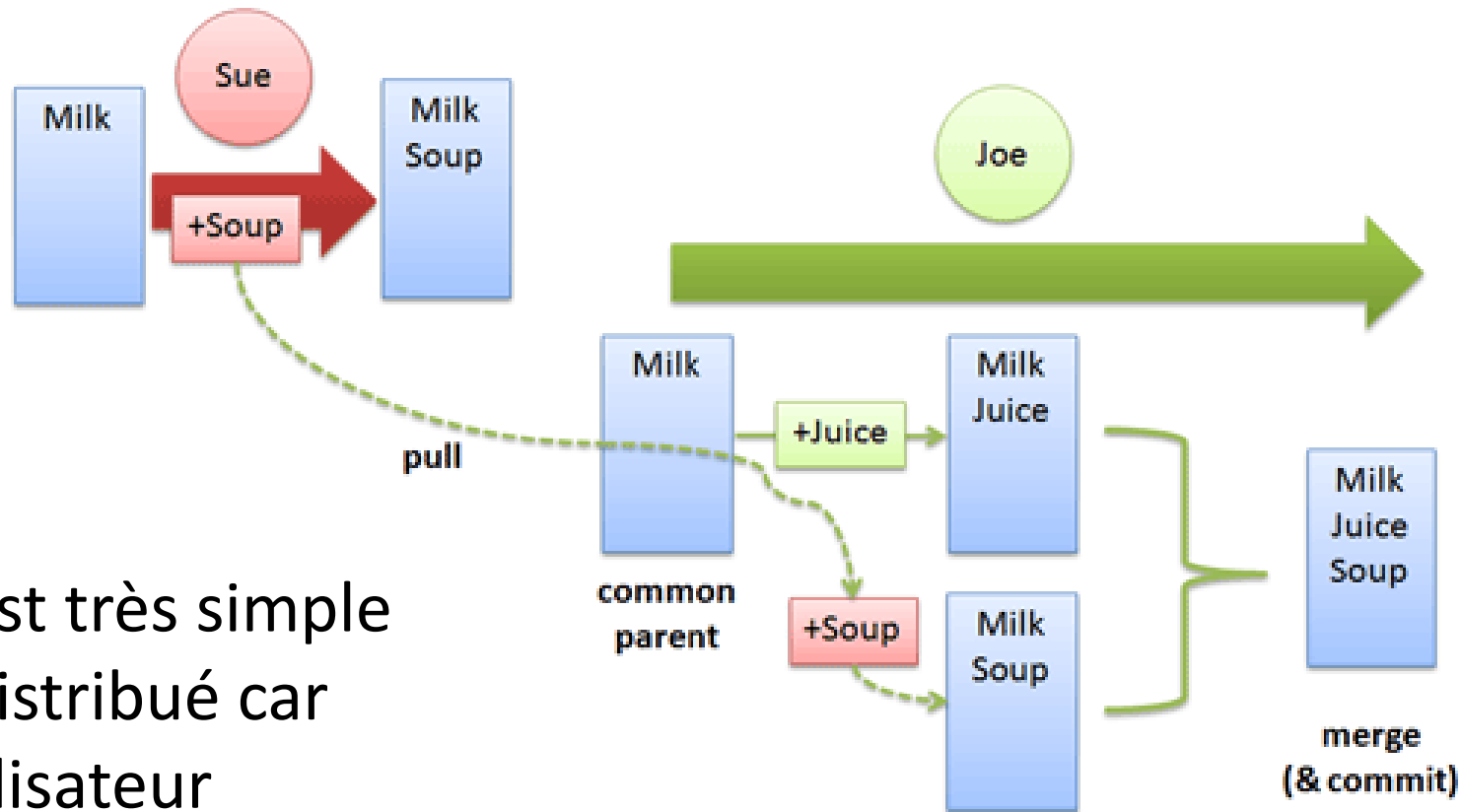
Le **repository** enregistre en local toutes les modifications des fichiers



Nouvelles instructions en mode distribué

- **Push** : envoyer ses modifications aux autres repositories
- **Pull** : récupérer des modifications d'un autre repository
- Attention : les mises à jour se font en plusieurs étapes :
 - Récupérer les modifications : avec un **push** ou un **pull**
 - Sauvegarder les modifications (en faisant un **commit**)
 - Un **update** ne peut avoir lieu que s'il n'y a pas de conflit.

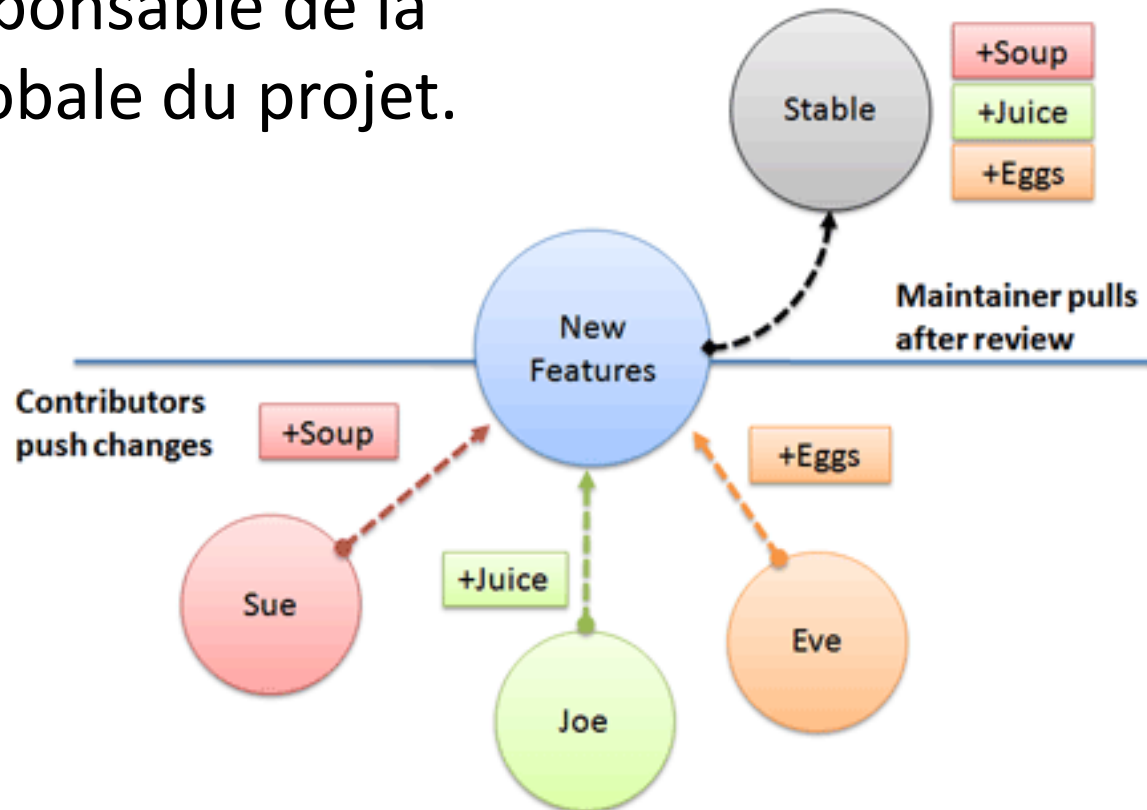
Comment fusionner les changements ?



La fusion est très simple
en mode distribué car
chaque utilisateur
choisit ce qu'il souhaite
intégrer

Comment organiser un projet ?

Une organisation possible avec un « Maintainer » (project manager) responsable de la cohérence globale du projet.



Les + et les – des DVCS

- + Chacun a son dépôt (**repository**), et donc son bac à sable sous contrôle de version
- + Chaque utilisateur peut faire des **commit** offline
- + La gestion des branches et des **merge** est plus simple
- + Il n'est pas nécessaire d'installer un serveur central
- S'il n'y a pas de serveur, il n'y a pas de **backup** distant
- La numérotation des révisions est plus compliquée (propre à chaque utilisateur)
- S'il n'y a pas de **project manager**, il n'y a pas vraiment de **latest version**... Chacun a la sienne.

Quelques (D)VCS

Quatre systèmes choisis pour leur réputation et leur popularité... Mais il en existe beaucoup d'autres !

Qu'est-ce qui différencie les VCS ?

Support des opérations atomiques (CVS ne le fait pas par exemple)

Différentes politiques pour l'édition concurrente de fichiers : **lock** / **merge** / autre

Approches centralisées vs. distribuées

Différentes fonctionnalités proposées (gestion des répertoires, **diff** amélioré, etc.)

CVS



Centralisé

Assez ancien

Fait toujours office de référence

```
$ cvs commit backend.c
```

```
$ cvs release -d tc
```

```
$ cvs diff driver.c
```

Subversion (SVN)

Centralisé

Assez récent

```
$ svn commit -m 'Use a class to print  
hello world'
```

```
$ svn add src/class.h src/class.cpp
```

```
$ svn checkout -r 4  
file:///home/user/svn project
```

```
$ svn info trunk/doc/index.html
```

GIT



Distribué
Récent

```
$ git clone git://github.com/git/hello-  
world.git  
$ cd hello-world  
$ (edit files)  
$ git add (files)  
$ git commit -m 'Explain what I changed'  
$ git format-patch origin/master  
  
$ cd (project-directory)  
$ git init  
$ (add some files)  
$ git add .  
$ git commit -m 'Initial commit'
```

Mercurial

Distribué

Récent

Une idée sur la
signification de
« hg » ?



```
$ hg clone http://selenic.com/repo/hello
$ cd hello
$ (edit files)
$ hg add (new files)
$ hg commit -m 'My changes'
$ hg push
```

```
$ hg init (project-directory)
$ cd (project-directory)
$ (add some files)
$ hg add
$ hg commit -m 'Initial commit'
```

Avez-vous bien suivi le cours ?

Savez-vous traduire et définir les mots suivants ?

Branch

Change

Checkout

Commit

Conflict

Diff

Label

Locking

Merge

Repository

Resolve

Revert

Revision

Tag

Trunk

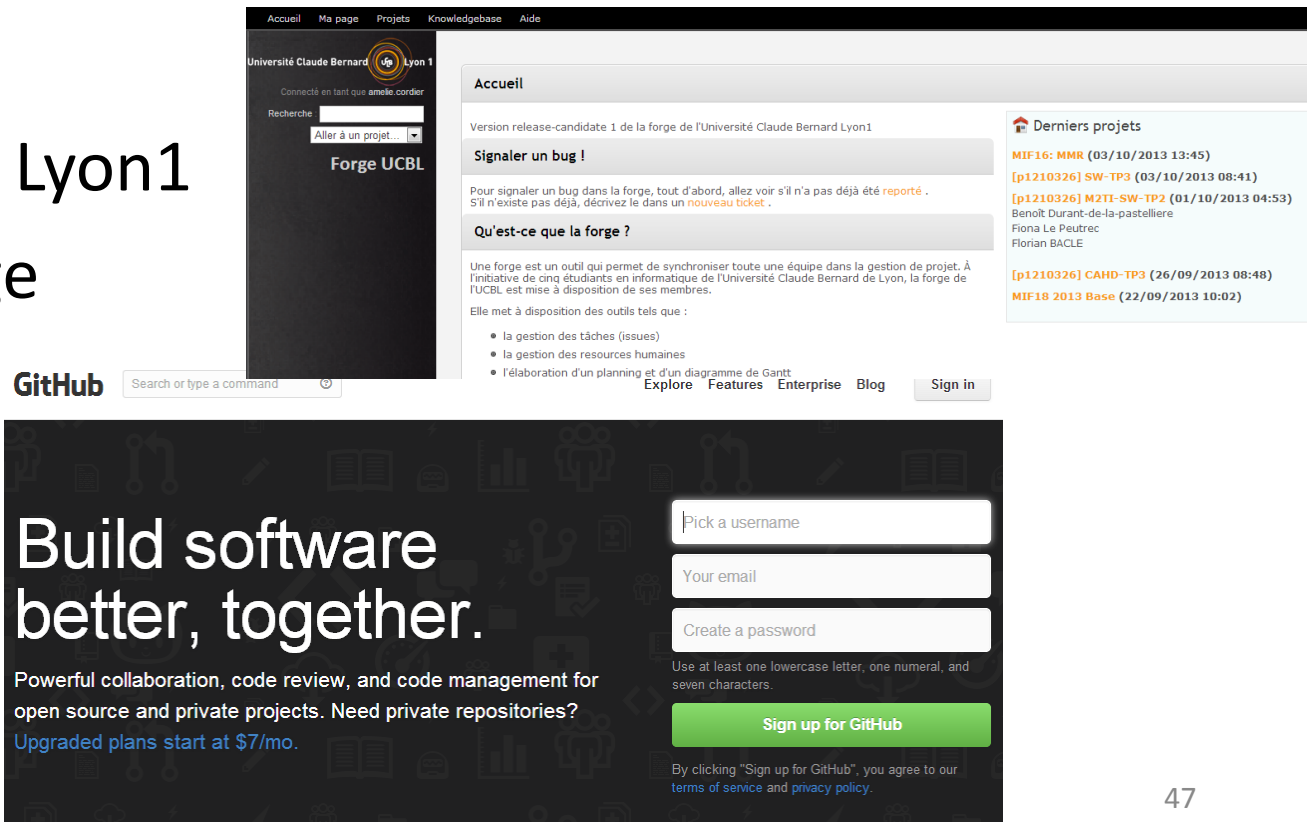
Update

Version

Working Copy

Et maintenant ?

- Choisir votre système (centralisé, décentralisé)
- Choisir votre environnement
 - GitHub
 - La forge de Lyon1
 - SourceForge
 - Bitbucket
 - ...



Sources

- Un bon point d'entrée pour tout savoir sur la gestion de versions :
 - http://en.wikipedia.org/wiki/Revision_control
- Un comparatif des outils de gestion de version distribués :
 - <http://code.google.com/p/support/wiki/DVCSAnalysis>
- Deux bons tutoriels en images :
 - <http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/>
 - <http://betterexplained.com/articles/a-visual-guide-to-version-control>
- Les sites des outils cités dans ce cours :
 - <http://mercurial.selenic.com/>
 - <http://git-scm.com/>
 - <http://subversion.tigris.org/>
 - <http://cvs.nongnu.org/>
- Source de certaines illustrations :
 - <http://www.siteduzero.com>