

Présentation des VCS/DVCS

Alexandre Ancel

alexandre.ancel@ihu-strasbourg.eu

IHU Strasbourg - Institut de Chirurgie Guidée par l'Image / IRCAD

06/07/2017



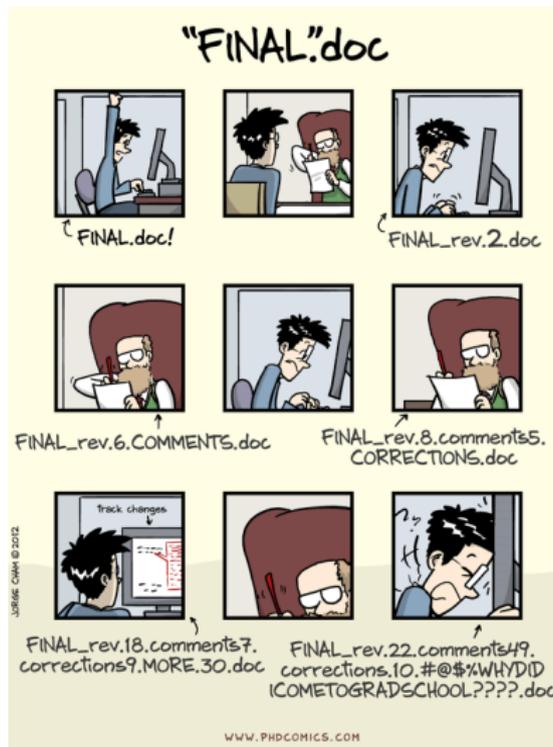
Outline

- 1 Gestion de version
- 2 VCS/DVCS
- 3 Conclusion

Plan

- 1 Gestion de version
- 2 VCS/DVCS
- 3 Conclusion

Pourquoi la gestion de version ?



"Piled Higher and Deeper" by Jorge Cham, www.phdcomics.com

Pourquoi la gestion de version ?

- 1 Votre code a un comportement étrange ?
- 2 Vous avez supprimé du code, mais c'était une erreur ?
- 3 Vous voulez exposez votre travail du mois dernier à vos collaborateurs ?
- 4 Vous voulez expérimenter une nouvelle fonctionnalité ou faire un refactoring sans casser le code de votre projet ?
- 5 Vous voulez tester différentes approches pour résoudre un problème sans savoir laquelle est la plus efficace ?

Pourquoi la gestion de version ?

But : Conserver l'historique d'un ensemble de fichiers au cours du temps

- Partager une base commune de travail
- Versions : Travail sur les modifications (groupes de diffs)
- Retourner à des versions antérieures (détection de bugs, ...)
⇒ Role correcteur critique
- Documenter les modifications, forme de programmation lettrée (auteurs, ...)
- Reproductibilité

Exemples : fichiers textes

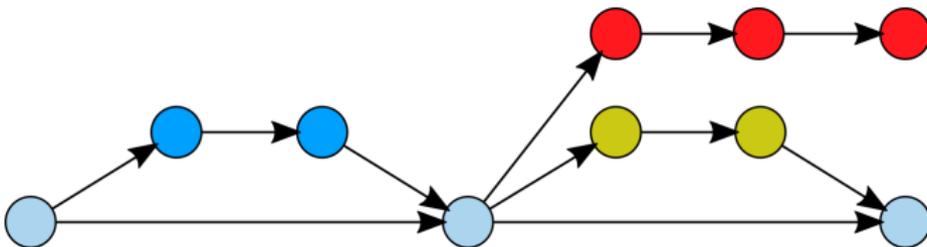
- Code source d'une application (C/C++, scripts ...)
- Documentation (html ...)
- Fichiers de configuration (dotfiles ...)
- Articles au format T_EX
- ...

Fonctionnalités de bases

- Dépôts (création, gestion, stockage)
- Récupérer du code (checkout)
- Proposer ses modifications (commit)
- Récupérer les modifications des autres (update)
- Révisions/Tags
- Branches
- Gestion des conflits
- Connexion à des services externes :
 - Forges,
 - Intégration continue,
 - ...

Représentation des données sous-jacentes

- Graphe orienté acyclique (DAG)
 - Noeud du graphe = *commit*
 - Changement depuis le noeud N-1
 - Identifiant unique global, ex : SHA-1 (Git)
 - Nombre d'arêtes sortantes > 1 = Création de branches
 - Nombre d'arêtes entrantes > 1 = Fusion de branches
 - Généralement :
 - le bout de la branche courante est marqué (HEAD),
 - la branche principale constitue le tronc (*trunk*)



Branches ¹

Primordial : Développement en parallèle de la branche principale

Modèles de branches :

- master, dev(evelop)
- Feature branch :
 - Faciliter la revue de code avant fusion
 - Pas d'impact sur la base de code initiale
 - Complexification de l'historique
- Conventions de nommage : feat/newExporter, fix/bug ...



¹Source: <https://git-scm.com>, Licence: <https://creativecommons.org/licenses/by/3.0>, non modifié

Collaboration

Aspect collaboratif :

- Possibilité de n'utiliser le gestionnaire de version qu'en mode mono-utilisateur
- Permet l'édition collaborative de documents avec gestion des conflits de fusion avancée

Aspect organisationnel / Gestion de projet :

- Plateformes construites autour (web)
- Tickets/Issues/Milestones
- Boards (Scrum, Kanban, ...)

Aspect participatif / Communautaire :

- Merge/Pull requests (Github/Gitlab/Rhodecode)

Plan

- 1 Gestion de version
- 2 VCS/DVCS
- 3 Conclusion

Deux modèles : VCS/DVCS

Deux modèles pour la gestion de version :

- Version Control Systems
- Centralisée : (C)VCS, ou décentralisée : DVCS

VCS : Approche Client-server

- Modèle historique
- Une seule copie centralisée d'un dépôt
- Commiter
 - Enregistrer ses changements dans le système central
- Besoin d'accéder au serveur central pour envoyer chaque changements
 - Développement à grande échelle, possibilité de goulot d'étranglement
- Modes de fonctionnement : Verrouillage de fichier ou Fusion

Deux modèles : VCS/DVCS

DVCS : Approche Pair à pair

- Aucun dépôt de référence par défaut
- Sur-ensemble des VCS
- Copie de travail = un bac à sable
 - Historique complet du dépôt
 - Mode déconnecté : Les changements peuvent rester locaux
 - Rapidité d'accès aux modifications : Disque
 - Pas de nécessité d'avoir l'autorisation de commiter
 - Création de sauvegardes multiples
- Envoyer des changements indifféremment vers plusieurs dépôts distants
 - Chaque copie de travail peut récupérer du code d'une autre copie de travail
- Potentiellement moins de gestion requise
 - Pas de serveur central à configurer

(C)VCS/DVCS Principaux

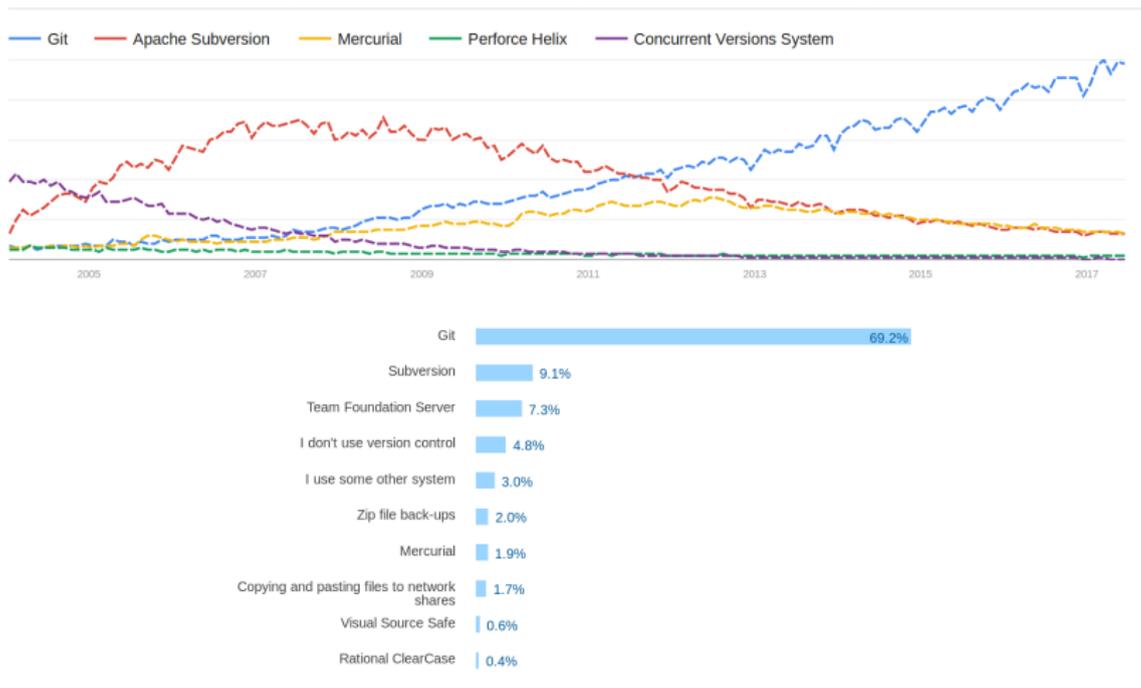
VCS :

- CVS (The CVS Team, Open Source, 1990),
- Subversion (SVN) (Apache, Open Source, 2000),
- Perforce Helix (Perforce software, Propriétaire, 1995),
- ...

DVCS :

- Mercurial, hg (Open Source, 2005),
- bazaar, bzt (Canonical/Communauté, Open Source, 2005),
- Git (Linus Torvalds/Junio Hamano/Communauté, Open Source, 2005)
- ...

Evolution et tendances^{2, 3, 4}



²<https://rhodecode.com/insights/version-control-systems-2016>

³Data source : Google Trends, 2004-present (www.google.com/trends)

⁴<https://insights.stackoverflow.com/survey/2017>, 30 730 réponses de développeurs

Subversion (SVN), Mercurial & Git

	Subversion	Mercurial	Git
1ère release	20/10/2000	19/04/2005	07/04/2005
Dernière version	29/11/2016	04/06/2017	25/06/2017
Mainteneur	Fond. Apache	Matt Mackall	Junio Hamano
Langage principal	C	Python/C	C/Shell
Type	Centralisé	Distribué	Distribué
Modèle	Verrou/Fusion	Fusion	Fusion

- Utilisateurs :
 - SVN : Clang, GCC, Apache OpenOffice, ...
 - Mercurial : Python⁵, OpenJDK, Nginx, ...
 - Git : Linux kernel, Android, MySQL, Libreoffice, ...

⁵<https://www.python.org/dev/peps/pep-0374/#why-mercurial-over-other-dvcss>

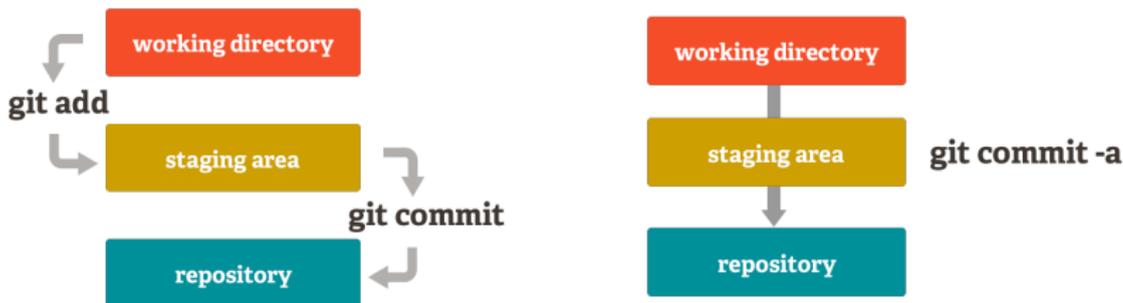
Points communs

- Tous multi-OS
- Implémentation de l'ensemble des fonctionnalités de bases d'un VCS

Subversion (SVN)	Mercurial (Hg)	Git
svn checkout	hg clone	git clone
svn add	hg add	git add
svn delete/remove	hg remove	git rm
svn move/rename	hg move/rename	git mv
svn commit	hg commit ; hg push	git commit -a ; git push
svn diff	hg diff	git diff, git diff -cached
svn log	hg log	git log
svn revert	hg revert	git checkout -f
svn status	hg status	git status
svn update	hg pull -update	git pull

Quelques différences : add⁶

- SVN & Mercurial : Marque un fichier comme versionné
- Particularité de Git : "Staging area"
 - Fonctionnalité cachée dans les autres VCS
 - Etape de "soft commit" dans un espace intermédiaire
 - Permet de façonner ses commits au préalable



⁶Source: <https://git-scm.com>, Licence: <https://creativecommons.org/licenses/by/3.0>, non modifié

Quelques différences : branches

- * SVN : branche = copie d'un repertoire du dépôt
 - Fait physiquement partie du dépôt
 - Intégration dans certaines workflows
 - ⇒ lancer des tests sur l'ensemble des branches en même temps
 - Branches temporaires ?
- ✓ Git : branche = référence vers un commit
- ✓ Mercurial :
 - hg branch : Meta data enregistré dans les commits
 - Marquage des commits de la branche
 - hg bookmark : équivalent aux branches Git
- Bonus: Git : fusion de plusieurs branches : octopus merge

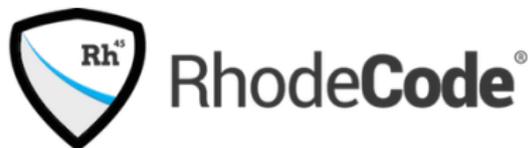
Quelques différences : Contrôle d'accès

VCS

- La présence d'un serveur permet de spécifier accès en lecture-écriture

DVCS

- Tous les contributeurs ont les mêmes permissions
- Possible avec les forges



Particularités : Modification de l'historique

"Une grande responsabilité est la suite inséparable d'un grand pouvoir"⁷

- Pourquoi modifier l'historique ?
 - Mots de passes committés ...
 - Modification d'un commit
 - Fusion de commits
- ✗ Subversion
- * Mercurial de base (sauf hg rollback)
 - Extensions : Patch queues (MQ) : Permet un niveau de réécriture
 - Phases : Différencier public et local (draft)
- ✓ Git : rebase
 - Ne faire les modifications que sur des commits locaux

⁷Collection générale des décrets rendus par la convention nationale, Volume 3

Cas particulier : Gestion des artefacts/Fichiers binaires

- Artefacts : contenu graphique, données binaires ...
- ✓ SVN : uniquement la dernière version
 - Mieux adapté : beaucoup de ressources graphiques
- ✗ Mercurial & Git
 - Commit d'un fichier binaire : tout le contenu dans l'historique
- Solutions pour les DVCS
 - Ignorer des fichiers par regexp : Fichier .gitignore (Git)
 - "Shallow clones"
 - Extensions :
 - Mercurial : extension LargeFiles (ou hg > 2.0)
 - git lfs (Gitlab, Github), git annex
 - Alternatives : ownCloud / nextcloud / Seafile
 - Possibilité de versionner ses fichiers aussi
 - Edition collaborative

Interfaces graphiques

- GUI pour chaque VCS :
 - Tortoise*,
 - GitKraken,
 - SourceTree,
 - ...

- Plugins pour environnements de développement :
 - Eclipse, Visual Studio, TextMate, ...
 - Vim, Emacs



Plan

- 1 Gestion de version
- 2 VCS/DVCS
- 3 Conclusion**

Conclusion

Choisir un VCS/DVCS :

- Tous sont matures, différences de workflow/implémentation
- DVCS offrent plus de flexibilité qu'un VCS classique en gardant les mêmes fonctionnalités
- Question d'organisation :
 - Votre entreprise/laboratoire a déjà des services en places autour d'un certain VCS ?
 - L'investissement pour le passage d'un VCS à un autre est-il important ?
 - Quelles seront les bénéfices que vous allez en tirer ?
 - Considérer l'écosystème à mettre en place (forges et fonctionnalités ...)