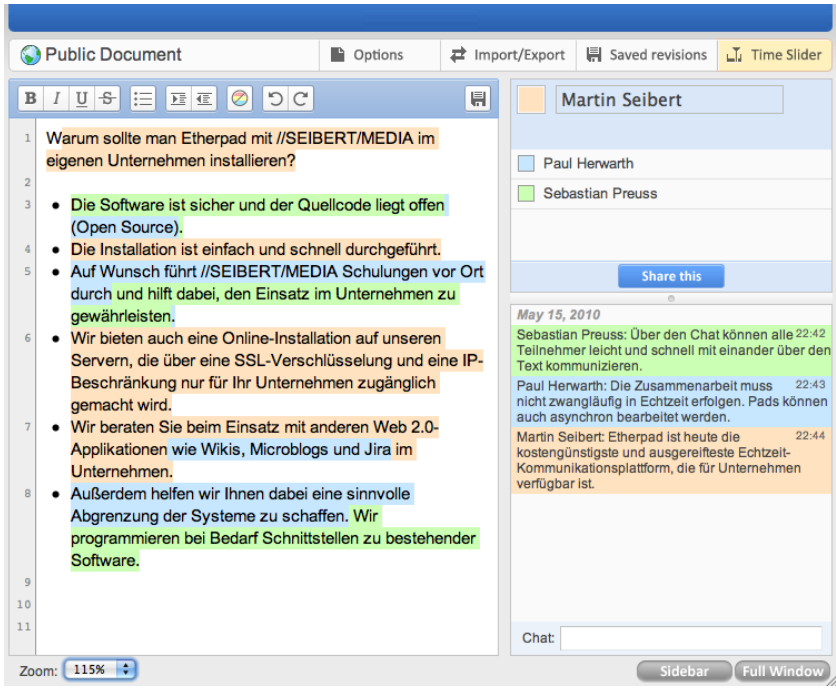
Logiciels de gestion de versions

Contexte/Problématique

- C’est pas de chance :-(:
  - Mon projet est sur mon portable … qui est tombé
  - C’est mon binôme qui a le projet sur son portable, mais il est malade
  - Mon projet marchait bien, et j’ai essayé d’implémenter une autre fonction et plus rien ne marche
- Travailler à plusieurs c’est pas facile :
  - « On s’échange le projet par mail et on travaille à tour de rôle dessus. => on n’avance pas vite »
  - « Pour que ça compile, j’ai du écraser les modifications de mon binôme… maintenant il est fâché »
  - « On a travaillé chacun dans notre coin et on n’arrive pas à fusionner notre travail. => rien ne marche à la fin »

Gestionnaire de versions et développement collaboratifs

- Gestionnaire de versions (fonction « Undo »):
  - Historique un peu évolué (annotation).
  - Exemple : les Wiki
- Outils collaboratifs :
  - Framapad (cf. cours de C2I)
  - Google docs/ * Office,… (cf. cours de C2I)
  - Dropbox, … (OK si vous êtes tout seul)
- Outils collaboratifs sans gestionnaire de version ?
  - => L'équipe explose très vite…



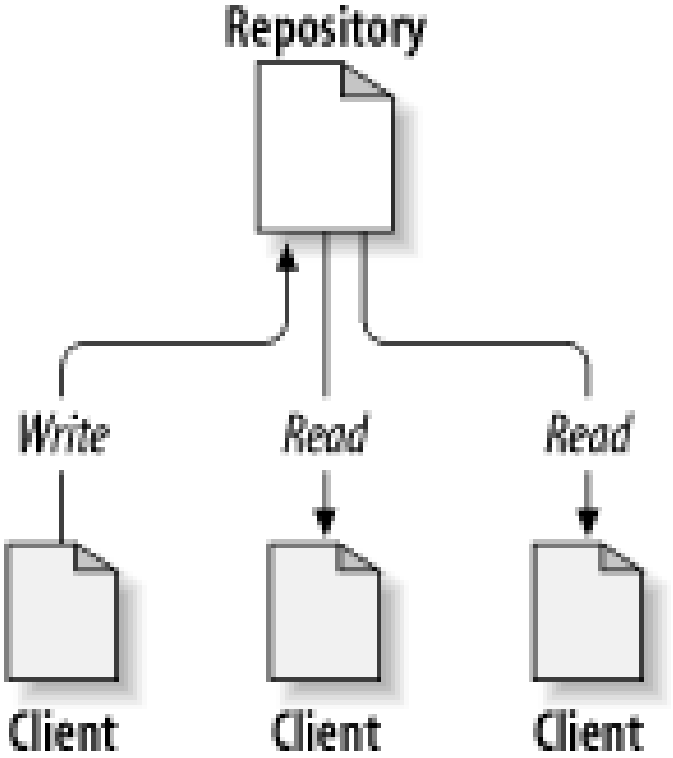
Gestionnaires de version pour le développement

- Centralisés :
  - RCS (mort ?)
  - CVS (presque mort)
  - subversion (mature, largement déployé)
  - ...
- Décentralisés :
  - git (mature, croissance ... mais plus complexe)
  - Mercurial
  - Quelques autres...

Modèle client-serveur

Les informations partagées sont stockées dans un dépôt (« repository ») sur le serveur central.

Chaque utilisateur travaille dans une copie locale.

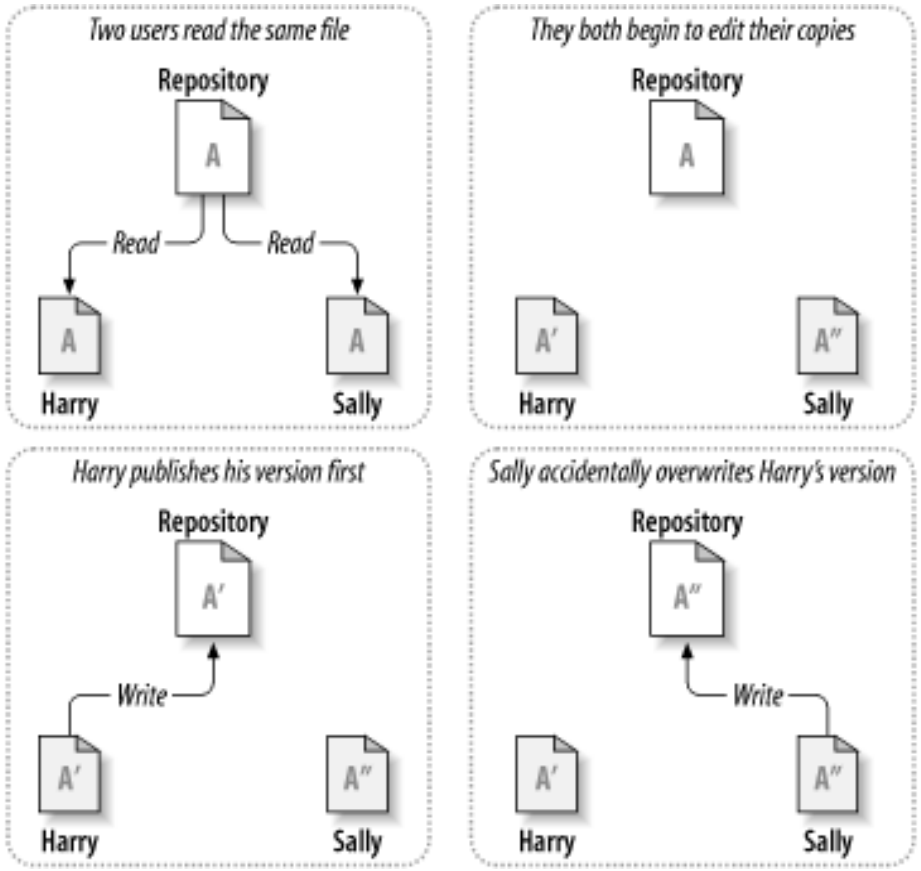


Source : "Version Control with Subversion"

Modèle client-serveur

Les problèmes arrivent vite...

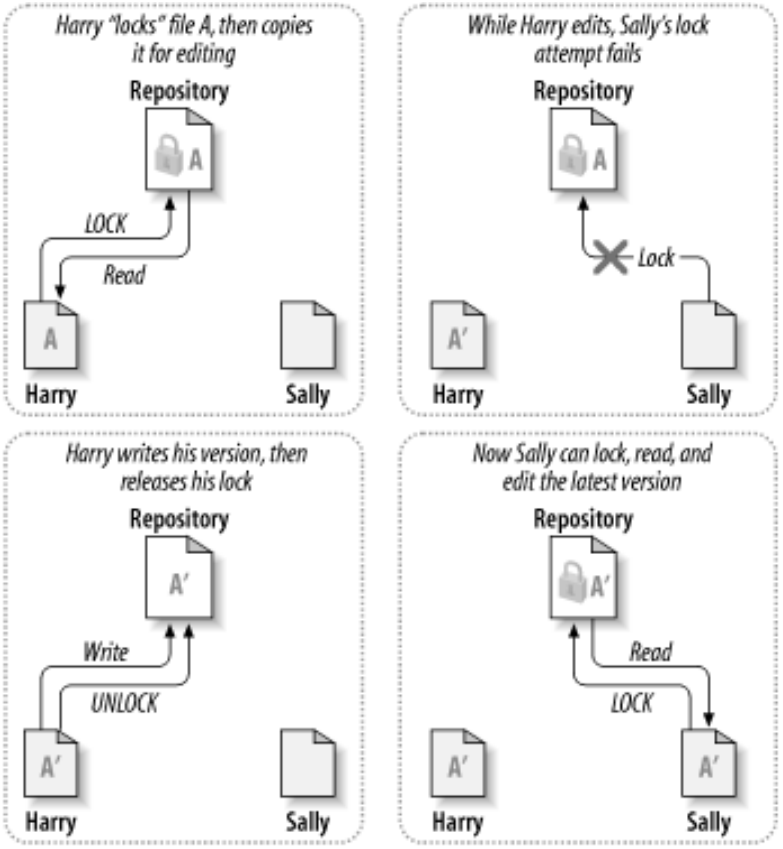
Exemple de scénario ou une partie du travail est perdue :



Source : "Version Control with Subversion"

Gestionnaire de versions

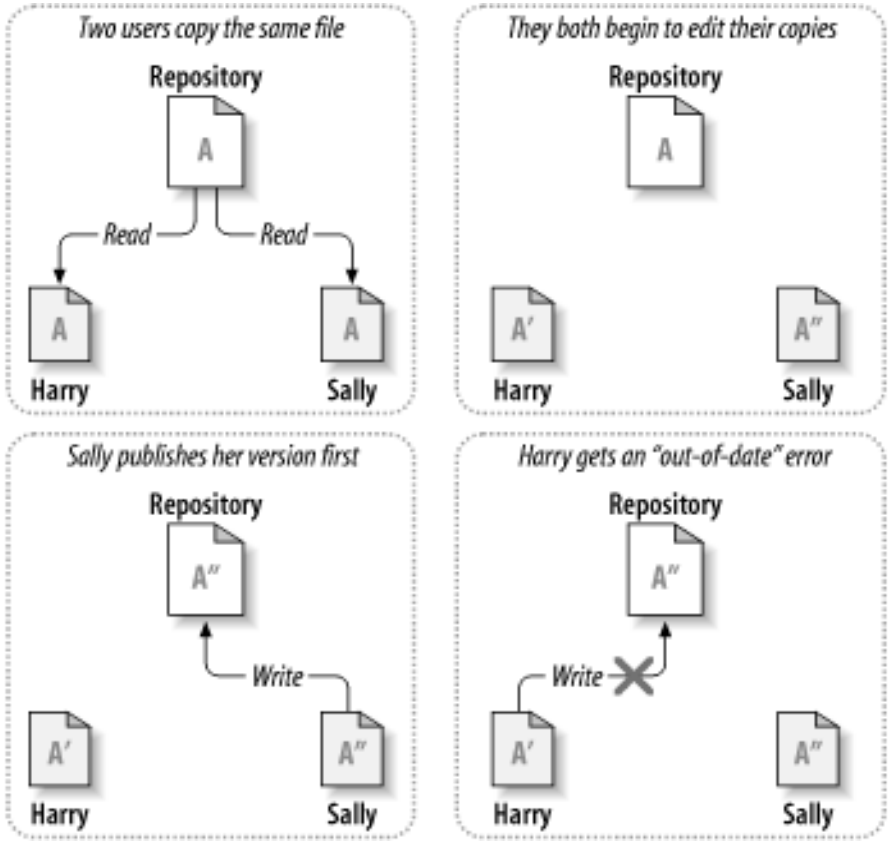
- Solution 1 : verrouiller pendant le travail (« lock-modify-unlock »)



Ca marche, mais impossible de travailler en parallèle sur un même fichier.

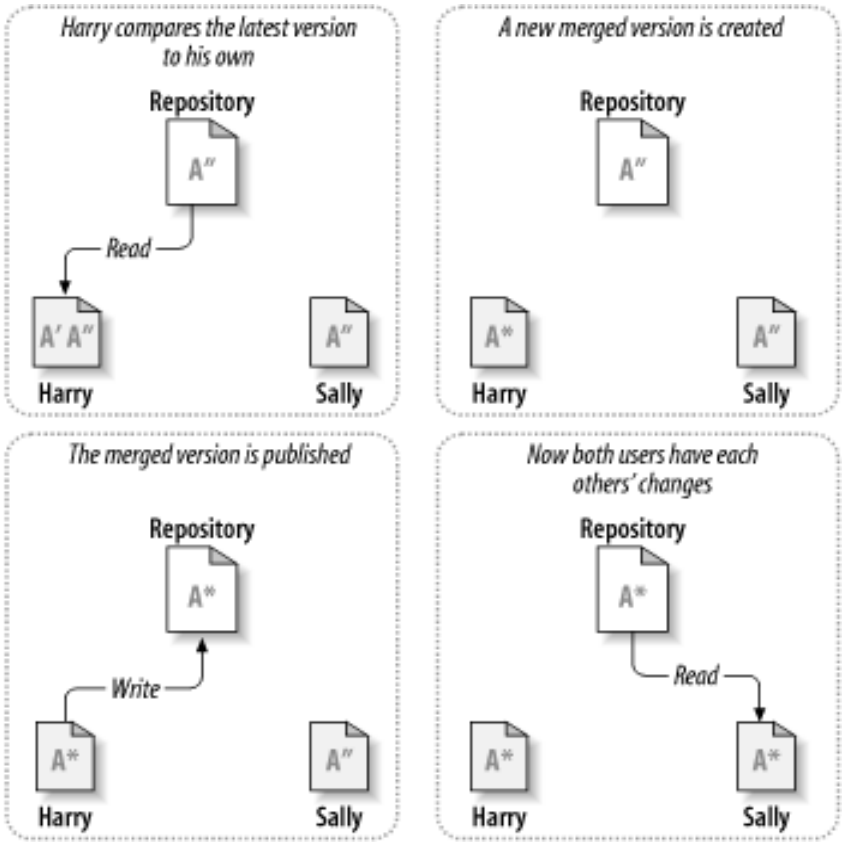
Gestionnaire de versions

- Solution 2 : travailler sur une copie (« copy-modify-merge ») (1/2)



Gestionnaire de versions

- Solution 2 : travailler sur une copie (« copy-modify-merge ») (2/2)



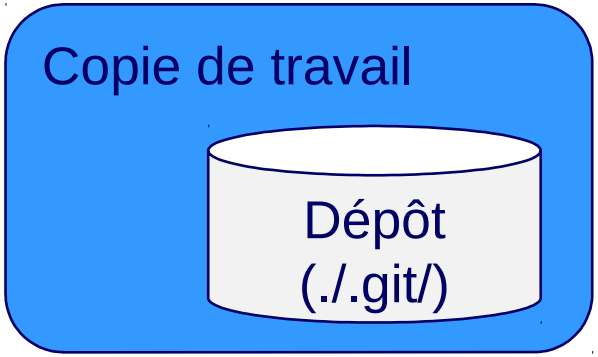
Ca marche, mais il faut gérer ensuite les conflits à la main.
C'est la solution retenue. Bonne nouvelle : en pratique les conflits se règlent automatiquement ou assez facilement à la main.

Légère introduction à Git

Git

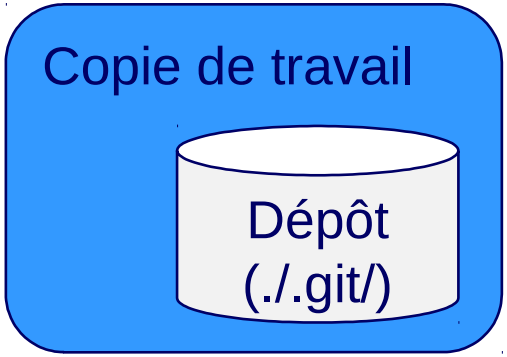
- Développé par Linus Torvalds (créateur de Linux) en 2005.
- Décentralisé
  - Chaque machine possède un dépôt local (<Monprojet>/.git/) associé à sa copie locale
    - Permet d’interroger le dépôt même sans le net
    - Beaucoup plus rapide
  - Généralement il y a 1 dépôt central et des dépôts secondaires
  - Les manipulations de base sont plus complexes qu’avec svn...

Création d'un dépôt (en local : 1 développeur)



- `cd monProjet`
- `git init`
- `git add *.c`
- `git commit –m «première version »`

« commiter » une nouvelle version



- def **Commiter :** publier une nouvelle version dans un dépôt.
- Avec Git cela se fait en 2 temps
  - 1) mettre dans l'**index** (également appelé stage) la liste des fichiers à commiter. Cela se fait avec la commande **git add**
  - 2) Envoyez ces fichiers dans le dépôt pour en faire une nouvelle version. Cela se fait avec la commande **git commit.**

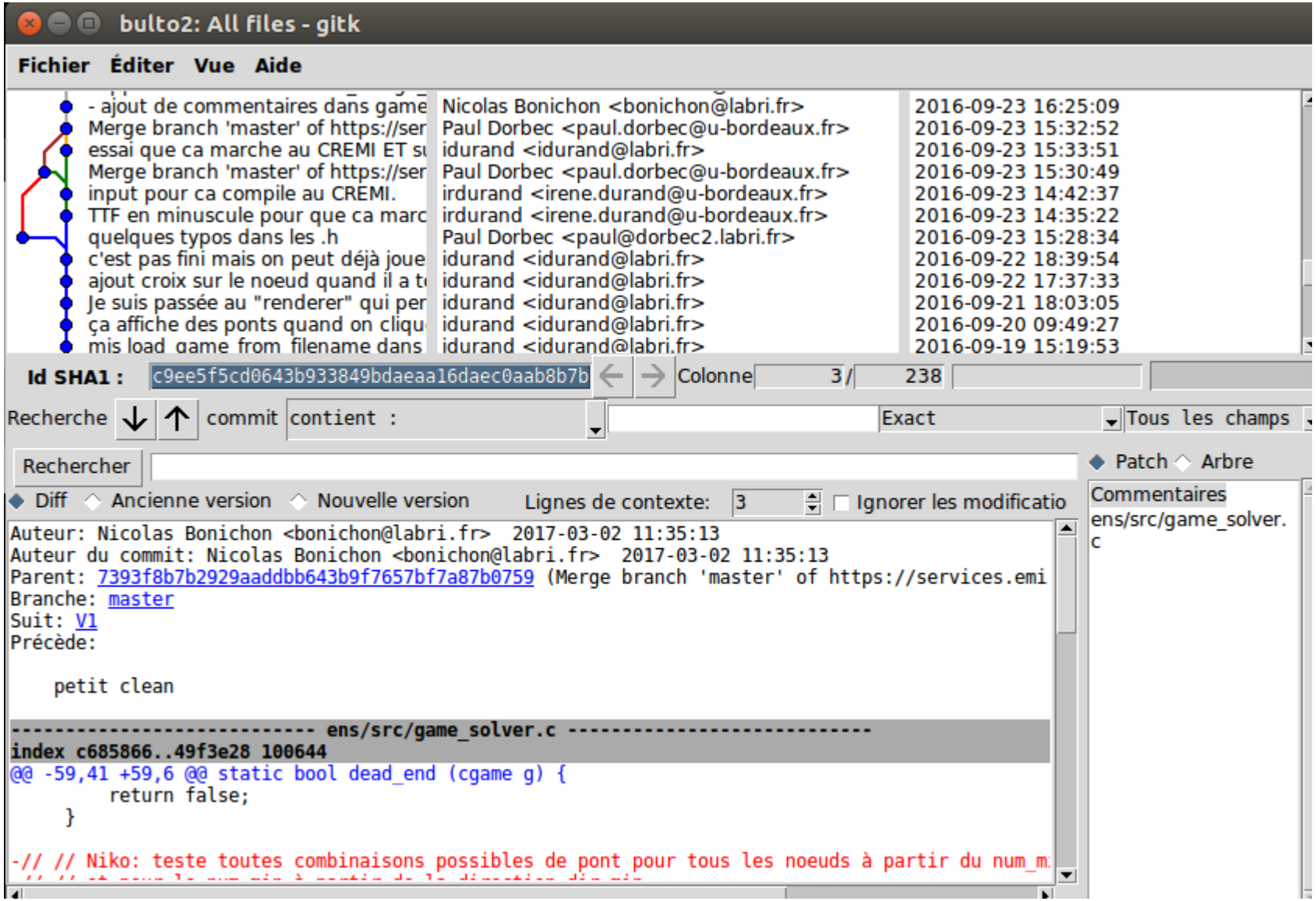
Exemple

- emacs main.c matrice2d.c
- git add main.c matrice2d.c
- git commit –m «correction du bug N° 23»

Historique et Nommage des révisions

- La commande **git log** permet de consulter l'historique des révisions.
- Chaque révision(commit) est identifié par une clé de hachage. Ex : `git log`
  - `(…)`
  - `commit cba66fbca3c4751d0a056aba2adf99ba63cb7122`
  - `Author: niko`
  - `Date:   Mon Feb 1 11:07:46 2012 +0100`
  - `(…)`
- C'est pénible, mais il y a des raccourcis
  - Préfixe : cba66fb (OK, si c'est le seul commit avec ce préfixe)
  - HEAD : dernier commit
  - HEAD~2 : avant-avant dernier commit
  - Cba66fb~2 : 2 commit avant le commit `cba66fb`…
  - …

gitk : pour visualiser l'historique

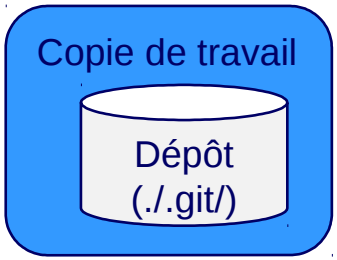


Oups !

- Récupérer la dernière version (HEAD) d’un ou plusieurs fichiers
  - git checkout -- <monfichier>
  - git checkout -- «*.c»
- Annuler le/les dernier(s) commit(s) :
  - git revert HEAD (Crée un nouveau commit qui annule le dernier commit)
  - git revert HEAD~n (crée un nouveau commit qui annule les n+1 derniers commits)

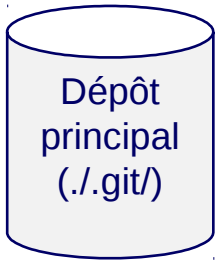
git : développer à plusieurs

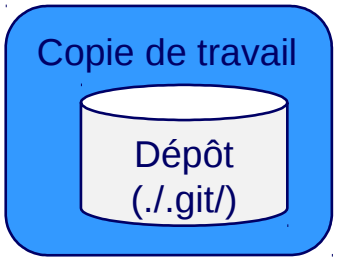
- Création du dépôt principal



git : développer à plusieurs

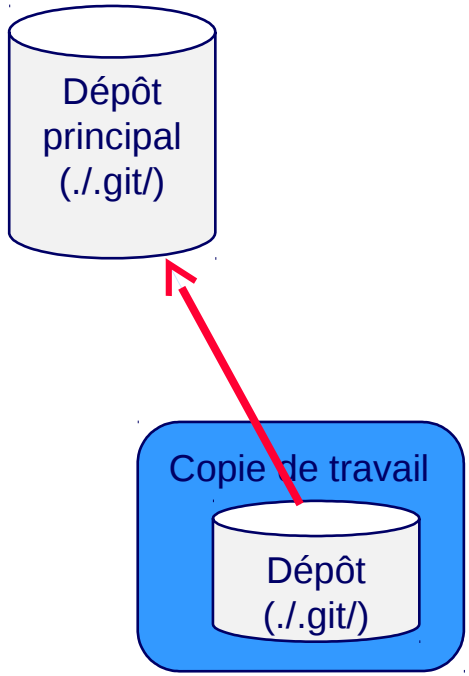
- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`





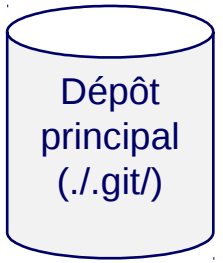
git : développer à plusieurs

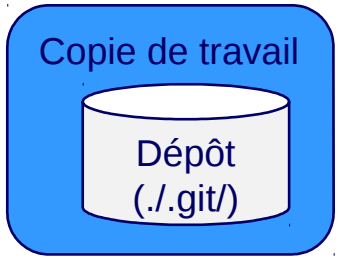
- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`

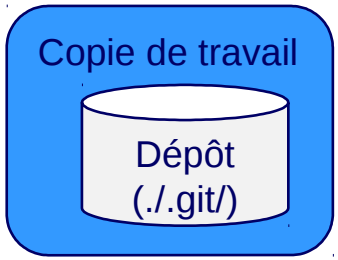


git : développer à plusieurs

- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`
- Création d'un deuxième dépôt local
  - `git clone ssh://monServeur/monProjet`

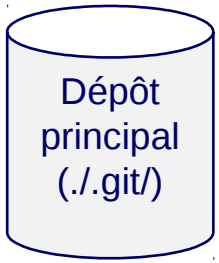


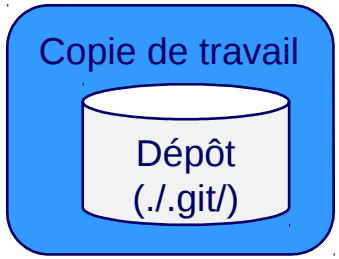


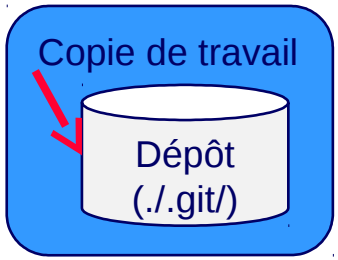


git : développer à plusieurs

- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`
- Création d'un deuxième dépôt local
  - `git clone ssh://monServeur/monProjet`
- Publication de ses modifications
  - `git add <les fichiers modifiés>`
  - `git commit –m « … »`
    (-> dépôt local)

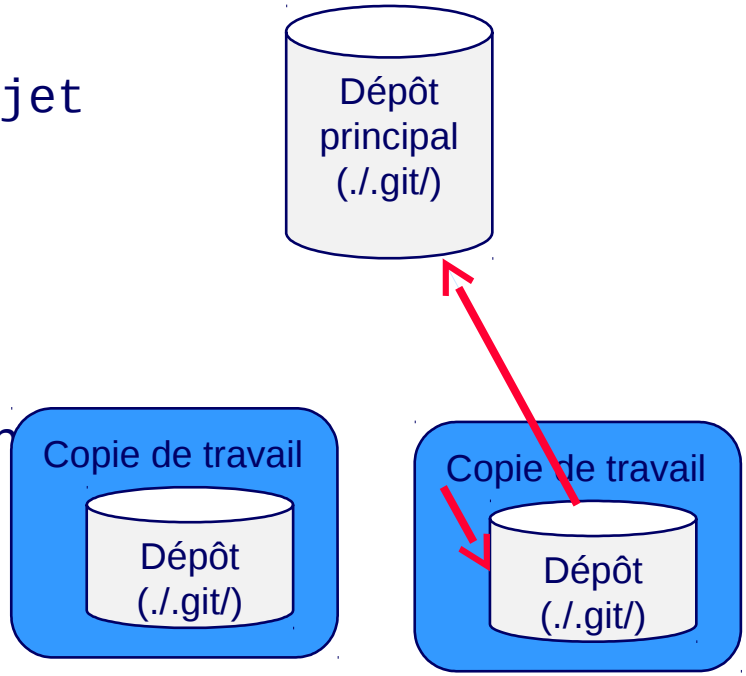






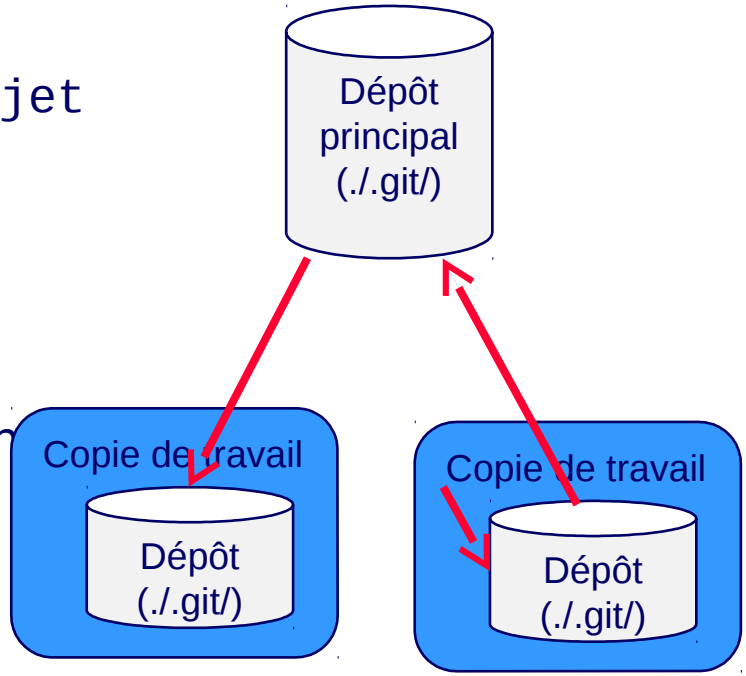
git : développer à plusieurs

- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`
- Création d'un deuxième dépôt local
  - `git clone ssh://monServeur/monProjet`
- Publication de ses modifications
  - `git add <les fichiers modifiés>`
  - `git commit –m « … »`
    (-> dépôt local)
  - `git push` (-> recopie dans le dépôt cen



git : développer à plusieurs

- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`
- Création d'un deuxième dépôt local
  - `git clone ssh://monServeur/monProjet`
- Publication de ses modifications
  - `git add <les fichiers modifiés>`
  - `git commit –m « … »` (-> dépôt local)
  - `git push` (-> recopie dans le dépôt cen
- Récupération des commits des autres
  - `git pull`



git : développer à plusieurs

- Création du dépôt principal
  - `ssh monServeur`
  - `cd monProjet`
  - `git init --bare`
- Copie du dépôt local
  - `git push ssh://monServeur/monProjet HEAD`
- Création d'un deuxième dépôt local
  - `git clone ssh://monServeur/monProjet`
- Publication de ses modifications
  - `git add <les fichiers modifiés>`
  - `git commit –m « … »` (-> dépôt local)
  - `git push` (-> recopie dans le dépôt cen
- Récupération des commits des autres
  - `git pull`
  - <résolution des conflits…>
  - `git commit –m « … »`
  - `git push`

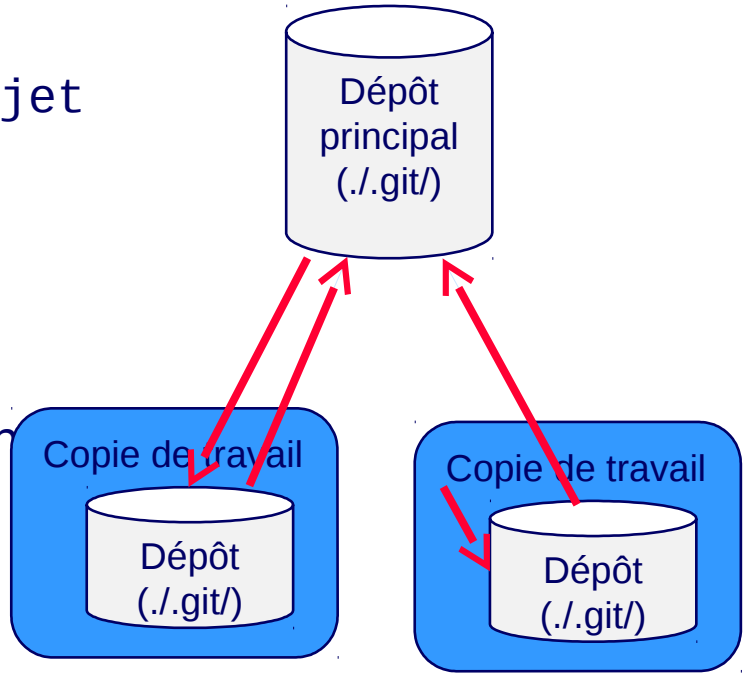
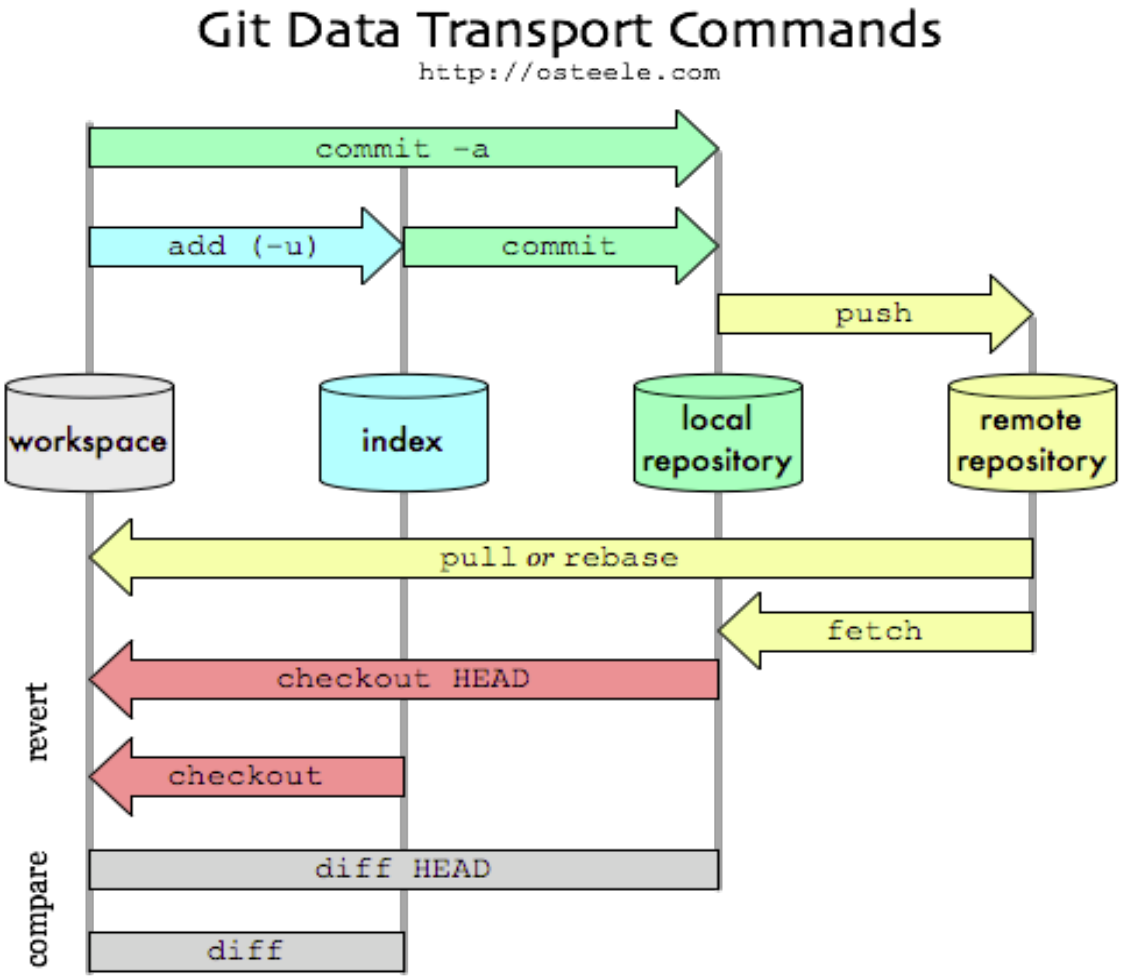


Schéma global



Bonnes pratiques

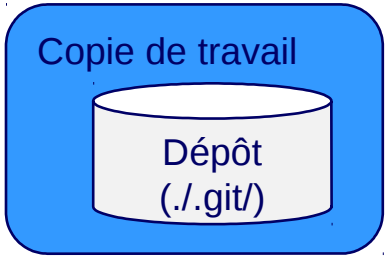
- Ne commitez que des versions qui compilent correctement (sans erreur et sans warning).
  - sinon vous risquez de bloquer les autres...
- Commentez chaque commit !
  - C’est important pour le suivi des modifications. (quel commit un corrigé quelle erreur / quel commit a introduit telle ou telle fonctionnalité / ...)
- « Commit Early, Commit Often » Commitez rapidement, commitez souvent.
  - Cela permet de mieux tracer les modifications
  - Cela permet de diminuer les conflits
  - Mais ne commitez que des choses qui compilent !

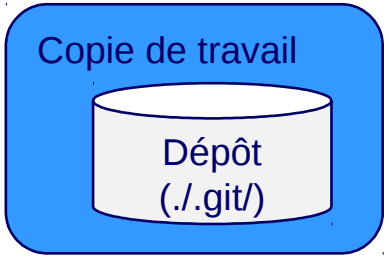
git : développer à plusieurs avec Savane

- Le dépôt principal est déjà créé sous savane.
- Il suffit que chaque développeur clone le dépôt
  - `git clone ssh://…`
- Publication de ses modifications
  - `git commit –m « … »`
    (-> dépôt local)
- Récupération des commits des autres
  - `git pull`
  - <résolution des conflits…>
  - `git commit –m « … »`
  - `git push`
  -
- Pour ne plus taper son mot de passe à chaque fois
  - `git config credential.helper store`

git : développer à plusieurs (suite)

- Généralement on utilise des branches.
- La branche principale est « master »
- Généralement on crée une branche (`git branch maBranche`) et on travaille dessus (plusieurs commits).
- Ensuite on fusionne la branche avec « master »
  - `git checkout master`
  - `git merge mabranche`





Références

- Subversion :
  - Livre "Version Control with Subversion" : disponible en ligne http://svnbook.red-bean.com/

- git :
  - http://git-scm.com/
  - Exposé de Linus Torval sur Git : https://www.youtube.com/watch?v=4XpnKHJAok8
  - Présentation du sed (Inria) : http://sed.bordeaux.inria.fr/seminars/GIT_20100511.pdf
  - Cours de Julien Sopena : http://julien.sopena.fr/enseignements/M2-SAR-Git/cours/01-Git/01-Git.pdf