Université Bretagne Sud

Systèmes de gestion de versions

Nicolas Le Sommer
Nicolas.Le-Sommer@univ-ubs.fr
Université de Bretagne Sud

Plan du cours

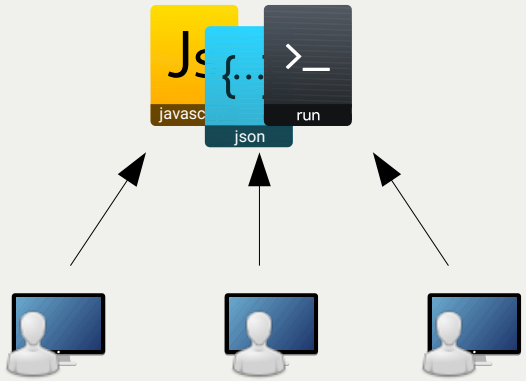
- Les différents types de systèmes de gestion de versions
- Introduction à SVN
- Introduction à GIT

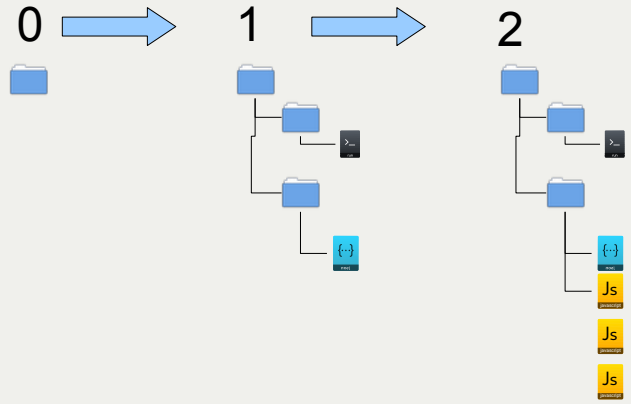
Système de contrôle de versions

- Différents termes pour désigner la même chose
  - Système de contrôle/gestion de versions (SGV)
  - En anglais on trouve la notion de SCM (System Control Manager), VCS (Version Control System) ou de RCS (Revision Control System)

Objectif et utilité

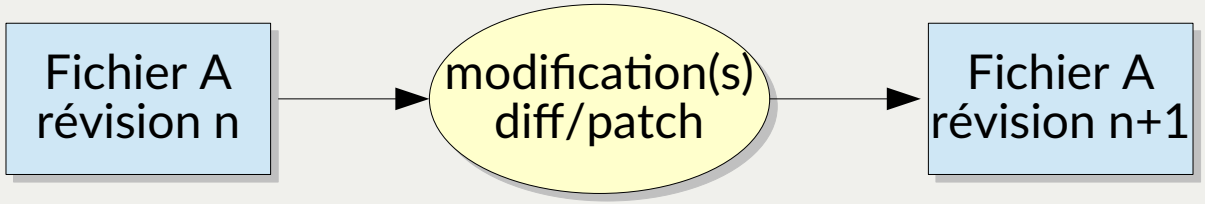
- Un SCV permet de
  - gérer plusieurs versions/révisions de documents/fichiers dans le temps
    - On peut revenir en arrière si nécessaire
  - maintenir un état cohérent d'un ensemble de fichiers (code source d'une application)
  - partager des documents/fichiers entre plusieurs utilisateurs
  - gérer les conflits d'accès en modification à un (ou des) fichier(s)



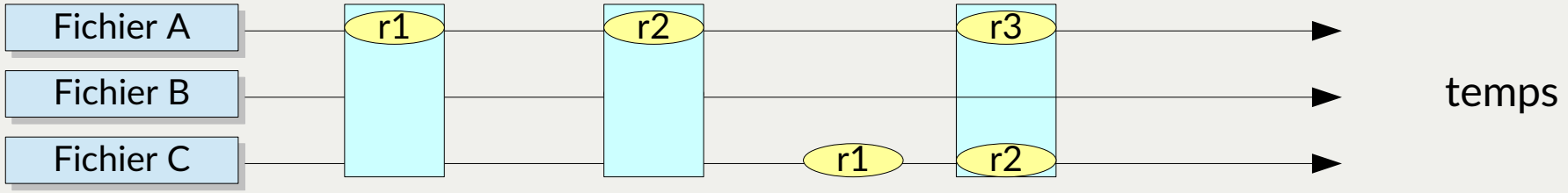


Principe de fonctionnement

- Versions/Révisions et modifications

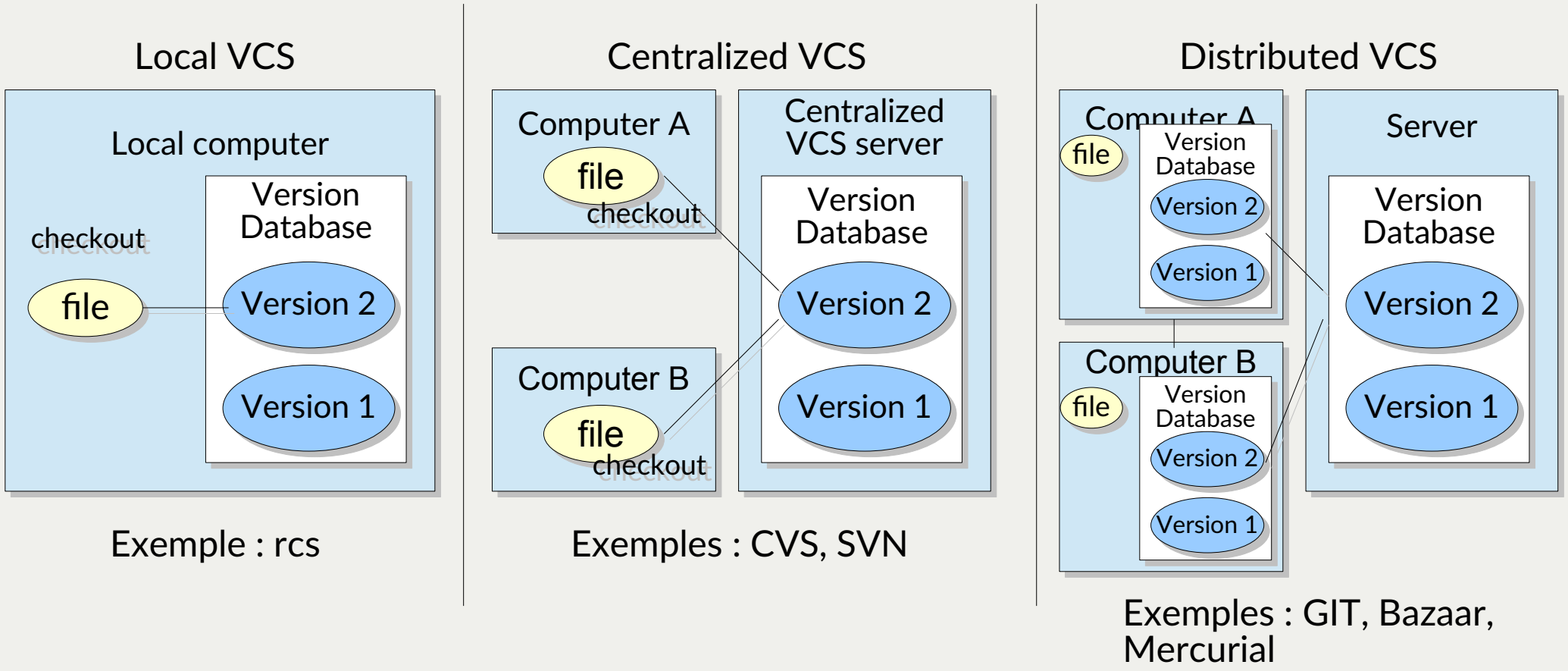


- Dans un développement logiciel, les modifications peuvent être également l'ajout, la modification et la suppression d'un fichier (ou d'un ensemble de fichiers)
- On préfère le terme révision en développement logiciel pour faire la différence avec les versions d'un logiciel
  - Une version d'un logiciel correspond à différentes révisions d'un ensemble de fichier(s) fini à une date donnée.



Les systèmes de contrôle de versions

- 3 types de systèmes de contrôle de versions(SCV)
  - Les systèmes locaux, centralisés et décentralisés



Dépôt et copies locales

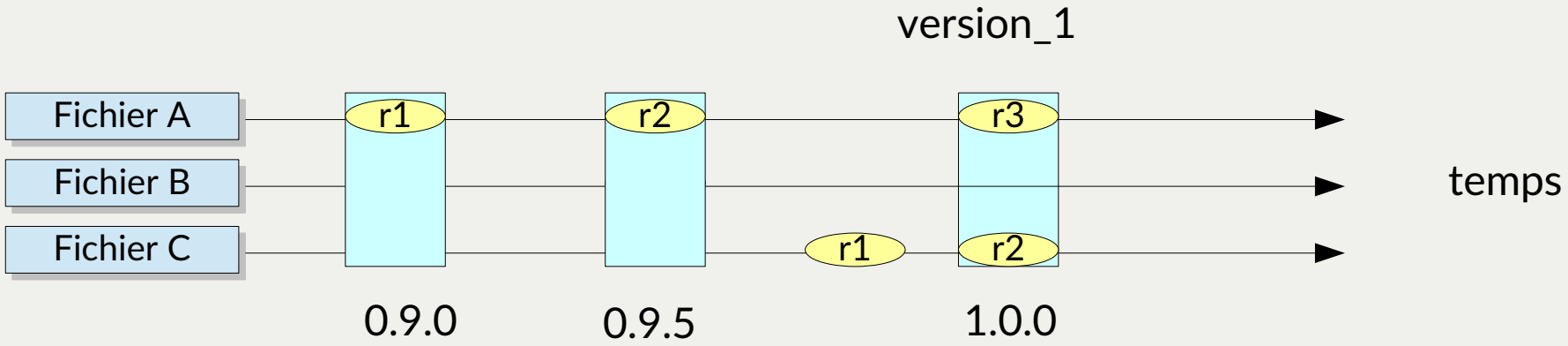
- Un dépôt est un espace de stockage (public) de fichiers gérés par un SCV
- Une copie locale est un ensemble de fichiers versionnés utilisé localement par un développeur
- À partir de sa copie locale, un développeur peut faire
  - des modifications (ajouts, suppressions, altérations de fichiers)
  - propager ses modifications sur le dépôt afin de les rendre accessibles aux autres développeurs (commit)
    - Des conflits peuvent apparaître lors de la propagation des modifications
    - Les conflits doivent être résolus pour que les modifications soient propagées

Branches

- Des branches peuvent être créées pour gérer des modifications divergentes sans conflits
- Les branches sont utilisées pour permettre :
  - la maintenance d'anciennes versions du logiciel (sur les branches) tout en continuant le développement des futures versions (sur le tronc)
  - le développement parallèle de plusieurs fonctionnalités volumineuses sans bloquer le travail quotidien sur les autres fonctionnalités.
- Le fait de vouloir rassembler deux branches est une fusion de branches.

Tags

- Des étiquettes (tags) peuvent être associées à certains états de révision pour identifier des versions du logiciel à un instant donné.
- Les étiquettes peuvent aussi être des noms donnés à un logiciel à un instant donné.



Numérotation des versions

- Schéma
  - major.mini[.micro] [-qualifier[-build_numer]]
- Incrément
  - Major : changement majeur
  - pas de retro-compatibilité (descendante) garantie
  - Mini : ajouts fonctionnels
    - retro-compatibilité garantie
  - Micro : maintenance corrective (bug fix)
- Qualificateurs
  - SNAPSHOT : version en évolution
  - alpha1 : version alpha numéro 1 (très instable et incomplète)
  - beta1 : version béta numéro 1 (instable)
  - rc2 : seconde *release candidate*
  - ...

Gestion des conflits

- Deux modes de gestion des conflits :
  - Gestion « préventive »
    - Pause d'un verrou avant modification du fichier
    - Levée du verrou explicite ou lors d'une propagation de la modification sur le dépôt
  - Gestion « optimiste »
    - « On croise les doigts pour que personne ne modifie le fichier en parallèle »
    - On doit gérer les éventuels conflits lors de la propagation des modification sur le dépôt
      - diff

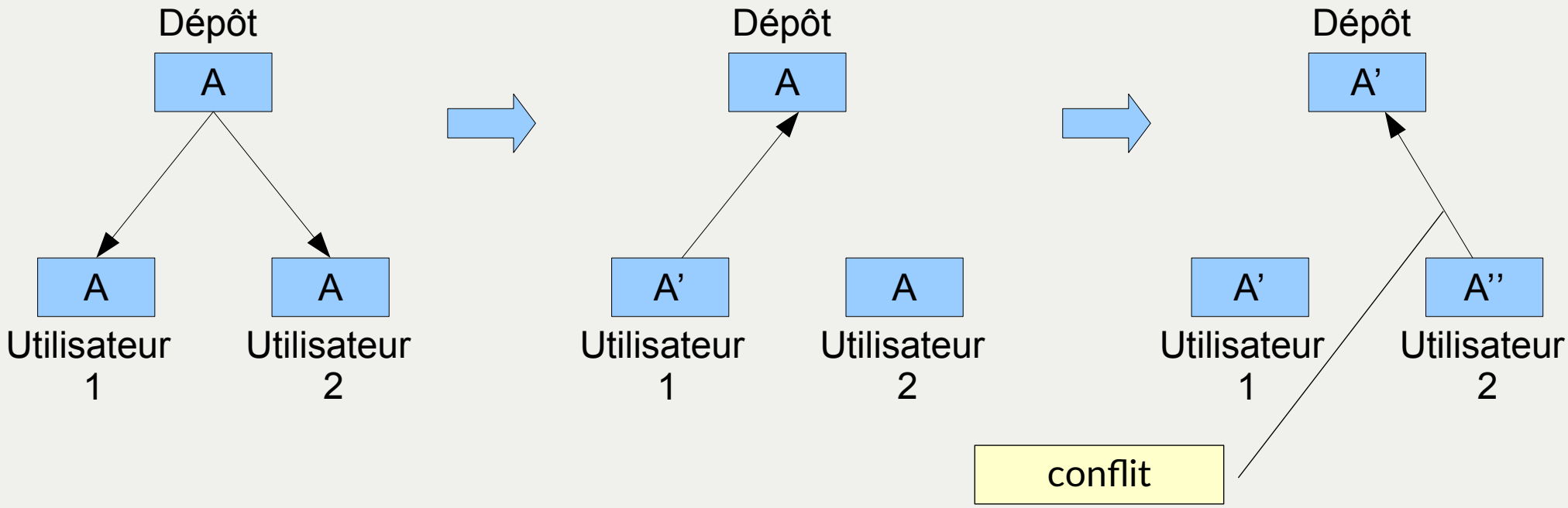
Subversion

Terminologie

- Dépôt
  - héberge tous les projets gérés sur un serveur.
  - archive toutes les versions des fichiers stockés.
- Copie locale
  - Copie du travail d'un développeur
  - Chaque modification sur la copie locale correspond à une révision

Gestion des conflits

- Il faut évitez d'altérer le travail des autres développeurs lors des conflits : 2 modèles de gestion possibles dans Subversion

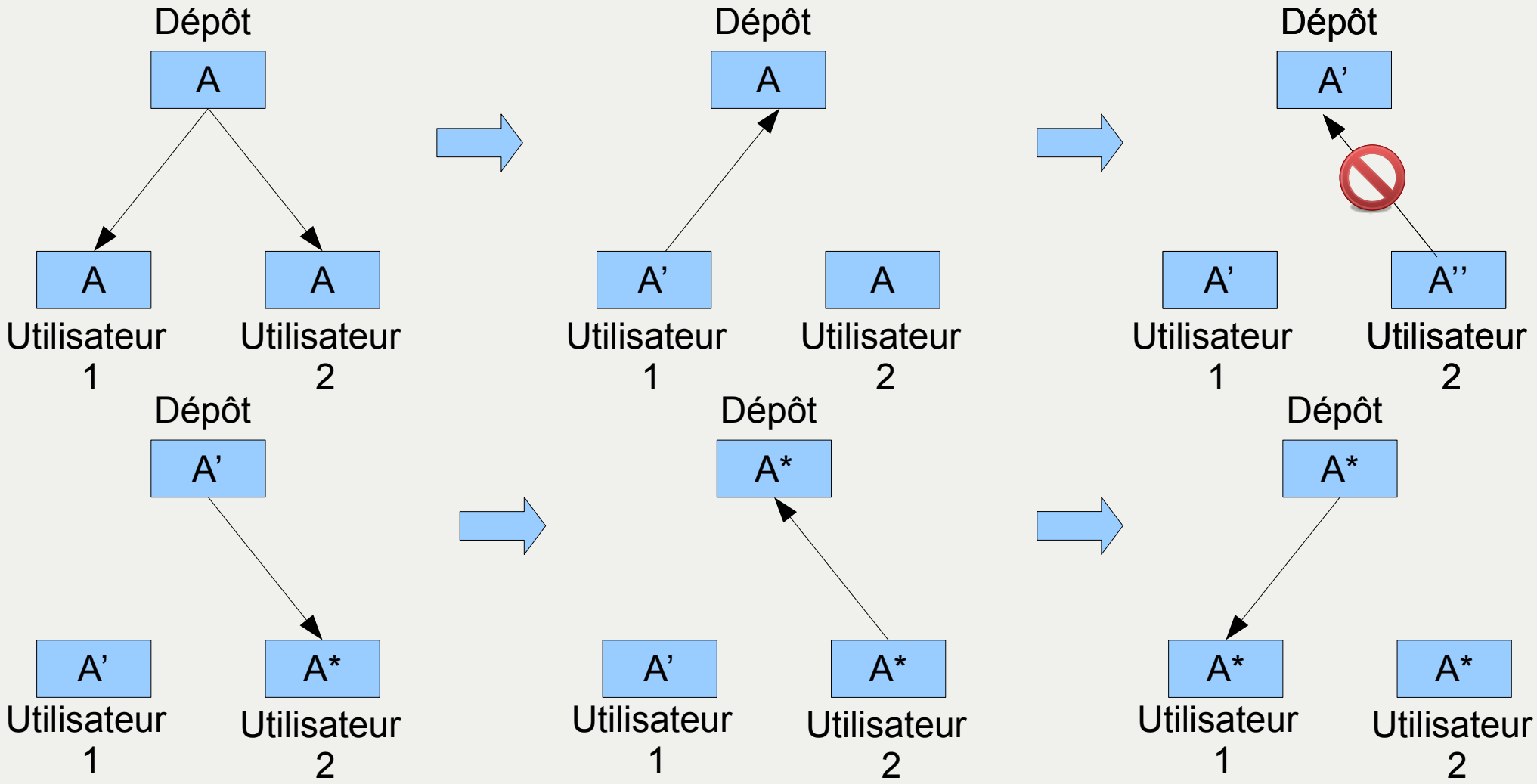


Gestion préventive des conflits : verrouillage

- Modèle verrouiller, modifier, libérer
- L'utilisateur 1 verrouille un fichier du serveur avant de mettre à jour sa copie et de travailler
- Tant que le dépôt reste verrouillé, l'utilisateur 2 ne peut pas obtenir le verrou
- Pour l'obtenir, l'utilisateur 2 doit attendre que l'utilisateur 1 libère le verrou en modifiant le fichier sur le serveur

Gestion des conflits : fusion

- Modèle copier, modifier, fusionner



Gestion des conflits : fusion

- L'efficacité du modèle copier-modifier-fusionner dépend de celle des algorithmes de fusion.
- En pratique :
  - La fusion automatique de Subversion fonctionne très bien tant que les utilisateurs ne modifient pas exactement la même ligne d'un programme
  - Ce cas est assez rare si le travail est correctement réparti au sein d'une équipe
  - Si ce cas survient, la résolution manuelle d'un conflit reste simple mais requiert une communication entre développeurs pour être bien gérée

Principales commandes

- Administration : ***svnadmin***
- Utilisation régulière pour la gestion de la copie locale : ***svn***

- Mise en place d'un dépôt

```
#> svnadmin create /path-to/repo
```

- Importation initiale des fichiers du dossier hello dans le dépôt

```
#> svn import hello file:///path-to/repo -m "import initial"
```

Principales commandes

- Création d'une copie de travail dans le dossier local
  - Plusieurs protocoles : file, http, https, svn, ...

```
#> svn checkout file:///path-to/repo /path-to/copie-locale
```

- Propagation des modifications vers le dépôt

```
#> svn commit /path-to/copie-locale -m "quelques modifications"
```

- Mise à jour de la copie locale avec la dernière version du dépôt

```
#> svn update /path-to/copie-locale
```

- Mise à jour de la copie locale d'un fichier à une révision antérieure du dépôt (exemple révision 36)

```
#> svn update -r 36 fichier
```

Principales commandes : ajout, suppression et modification

- Ajouter, supprimer, déplacer ou renommer des fichiers sur le système de fichiers (avec des commandes comme ***touch***, ***rm*** ou ***mv***) ne suffit pas.
- Un fichier local n'est pris en compte par le contrôle de version que si on le spécifie explicitement par

```
#> svn add nouveau_fichier
```

- De même, une suppression simple d'un fichier ne sera pas propagée. Pour supprimer un fichier d'une copie locale, il faut le faire par subversion :

```
#> svn rm fichier_a_supprimer
```

État des fichiers

```
#> svn status
```

- Liste des codes

`A`	Le fichier sera ajouté au dépôt
`D`	Le fichier sera supprimé du dépôt
`M`	Le fichier a été modifié localement depuis le dernier update
`C`	Le fichier est source de conflits
`X`	Le fichier est eXterne à cette copie de travail
`?`	Le fichier n'est pas géré par subversion
`!`	Le fichier est manquant, sans doute supprimé par un autre outil que subversion

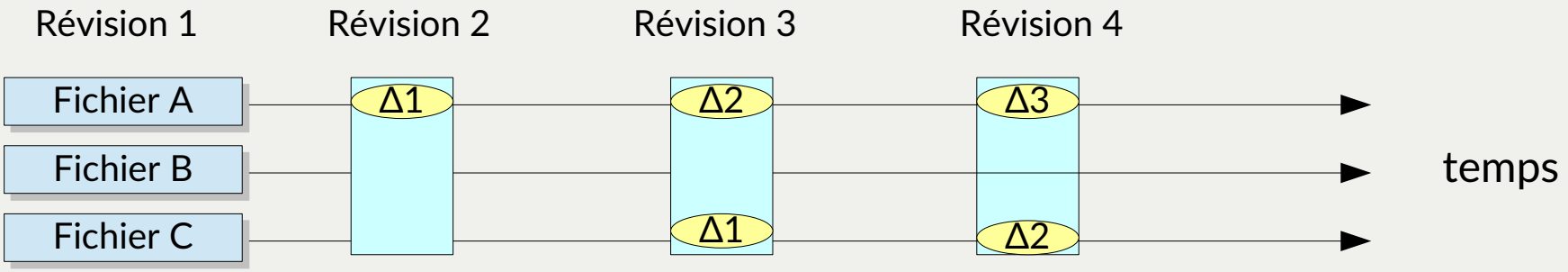

git

GIT : Introduction

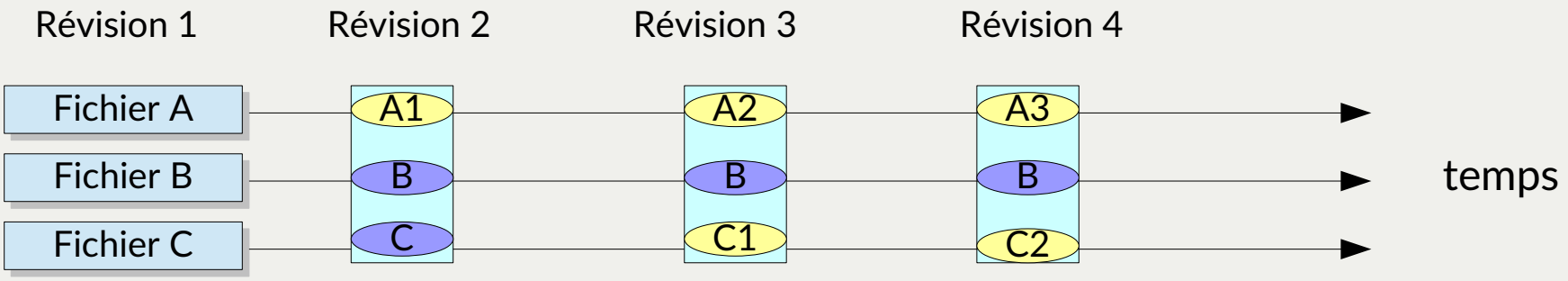
- Git est un SGV développé en 2005 par Linus Torvald pour supporter le développement du noyau Linux
- Git est un SGV distribué
  - Rapide (presque toutes les opérations sont locales),
  - supportant le développement non-linéaire (i.e., développement de plusieurs branches en parallèle)
  - SHA-1 est utilisé comme somme de contrôle pour déterminer si un fichier a été modifié ou non.

Principe de fonctionnement de Git

- Principe de fonctionnement de la plupart des SVG : gestion par différences

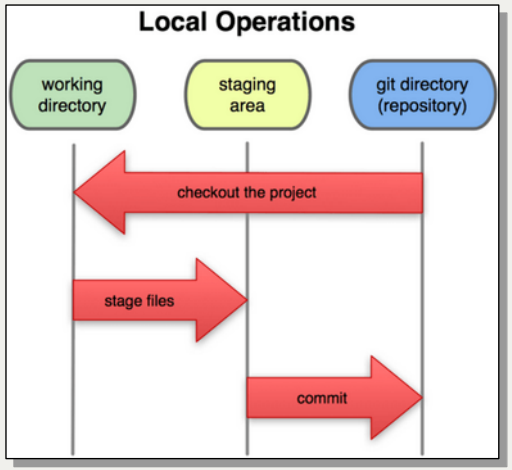


- Principe de fonctionnement de Git : gestion par instantanés



Les états locaux

- Git gère 3 états dans lesquels les fichiers peuvent résider :
  - Validé
    - Les données sont stockées en sécurité dans votre base de données locale.
  - Modifié
    - Le fichier a été modifié mais n'a pas encore été validé en base.
  - Indexé
    - Le fichier est marqué comme modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet.
- 3 zones
  - Répertoire local,
  - Zone d'index
  - Dépôt Git



Configuration de Git

- Identité

```
#> git config --global user.name "Nicolas Le Sommer"
#> git config --global user.email nicolas.le-sommer@univ-ubs.fr
```

- Éditeur et outil de différences

```
#> git config --global core.editor emacs
#> git config --global merge.tool meld
```

- Couleurs

```
#> git config –global color.ui true
```

- Fichier de configuration : ~/.gitconfig

Débuter avec Git

- Création d'un dépôt

```
#> git init
```

Créer un nouveau sous-répertoire nommé .git qui contient tous les fichiers nécessaires au dépôt

- Ajout de fichiers au dépôt

```
#> git add *.c
#> git add README
#> git commit –m 'version initiale du projet'
```

- Obtenir une copie d'un dépôt existant

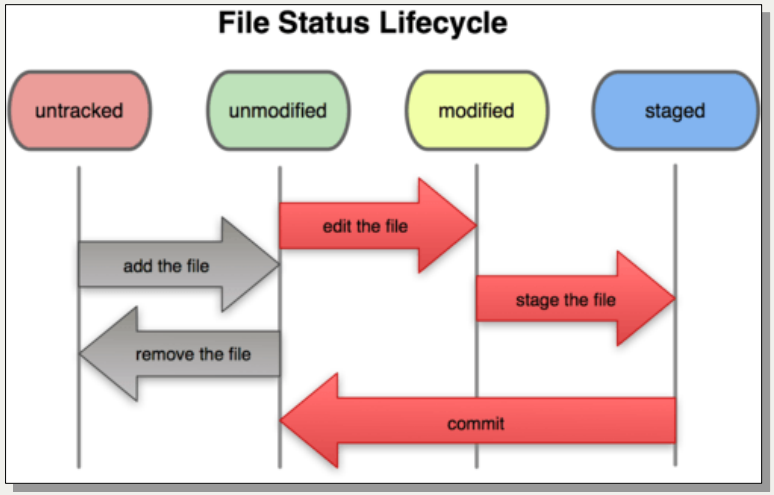
Les protocoles git, ssh, Https, http peuvent être utilisés

```
#> git clone https://git.univ-ubs.fr/mon_projet.git
#> git clone utilisateur@serveur:/chemin.git
```

Ajouter des fichiers dans un dépôt

- Vérification état

```
#> git status
```



- Placer de nouveaux fichiers sous suivi de version

```
#> git add LISEZ_MOI.txt
#> git add my_directory
```

- Indexer des fichiers modifiés

```
#> emacs LISEZ_MOI.txt
#> git add LISEZ_MOI.txt
```

La commande add peut être utilisée pour
- placer un fichier sous suivi de version,
- pour indexer un fichier ou
- pour d'autres actions telles que marquer comme résolus des conflits de fusion de fichiers

Supprimer et déplacer des fichiers dans un dépôt

- Suppression de fichiers

```
#> git rm mon_fichier
```

- Renommage/déplacement de fichiers

```
#> git mv mon_fichier mon_nouveau_fichier
```

- Validation des modifications

```
#> git commit -m “validation des modifications“
#> git commit -am “ajout nouveaux fichiers“
```

L'ajout de l'option -a permet de placer automatiquement le fichier qui est déjà en suivi de version dans la zone d'index avant de réaliser la validation, évitant ainsi d'avoir à taper la commande git add

Historique des modifications

- Visualisation de l'historique des modifications

```
#> git log
```

- De nombreuses options de formatage sont disponibles

Annuler des actions

- Modifier le dernier commit

```
#> git commit --amend
```

- Exemple :

```
#> git commit -m 'validation initiale'
#> git add fichier_oublie
#> git commit --amend
```

  - Les 3 commandes donnent lieu à la création d'un unique commit

Annuler des actions

- Désindexer un fichier déjà indexé

```
#> git reset HEAD mon_fichier.txt
```

- Réinitialiser un fichier modifié

```
#> git checkout -- mon_fichier.txt
```

Dépôts distants

- Afficher les dépôts distants

```
#> git remote
```

Si vous avez cloné un dépôt, vous devriez au moins voir l'origine origin

- Ajouter un dépôt distant

```
#> git remote add nom_court url
```

  - Exemple :

```
#> git remote add nls git://github.com/nls/a.git
```

- Récupérer des données depuis un dépôt distant

```
#> git fecth nom_du_depot
```

Dépôts distants

- Récupérer des données depuis une branche d'un dépôt distant

```
#> git pull
```

  - récupère et fusionne automatiquement une branche distante dans votre branche locale.
  - par défaut la commande git clone paramètre votre branche locale pour qu'elle suive la branche master du dépôt que vous avez cloné
- Pousser ses modifications sur un dépôt

```
#> git push nom_depot branche
```

  - Exemple :

```
#> git push origin master
```

Ressources

- http://git-scm.com/book/fr