

# Gestion de versions GPI

Jean-Marie Mottu (Arnaud Lanoix, Gerson Sunyé)  
IUT de Nantes – Département Informatique

# Un produit à gérer

---

- ▶ Le produit final est construit progressivement
  - ▶ différents acteurs travaillent sur un produit (et ses constituants)
  - ▶ la construction est progressive
    - ▶ et la livraison sera suivie de versions ultérieures
- ▶ La Gestion de Versions permet de
  - ▶ gérer l'évolution du produit,
    - ▶ Versionner
    - ▶ Sauvegarder
  - ▶ gérer les activités des différents acteurs sur le produit
    - ▶ Partager

# Gérer l'évolution du produit

---

- ▶ Pendant le développement le produit évolue
  - ▶ Dans un projet informatique, il s'agira
    - ▶ du cahier des charges,
    - ▶ du code, des fichiers de configurations
    - ▶ des contenus (multimédia, etc.)
    - ▶ de la documentation, etc.
  - ▶ Le produit doit être sauvegardé
  - ▶ L'évaluation du produit doit être « versionnées »
    - ▶ On sauvegarde aussi l'historique
      - retour arrière,
      - traçabilité (évolution dans le temps, localisation des problèmes),
      - apprentissage pour les projets futurs.
    - ▶ numéro de version,
    - ▶ version de développement, alpha, beta testeur, stable, finale, distribution, LTS.





# Gérer le partage du produit et les activités des différents acteurs

---







- ▶ Les acteurs travaillent sur le même produit en même temps
  - ▶ Une stricte organisation du projet pourrait leur éviter de travailler simultanément sur le même fichier mais ce n'est
    - ▶ ni suffisant (dépendances entre fichiers, etc.),
    - ▶ ni praticable (cela imposerait trop de communication)
    - ▶ ni possible parfois (plusieurs méthodes indépendantes mais dans une même classe, etc.),
  - ▶ La gestion de versions permet
    - ▶ de partager les avancées de chacun,
    - ▶ de résoudre les conflits.

# Solutions trop limitées pour versionner et partager

## ► Numérotation manuelle

-  GPI - TD3 - a - Taches partie 2 - 1.3.doc
-  GPI - TD3 - a - Taches partie 2 - 2.0.doc
-  GPI - TD3 - a - Taches partie 2 - 2.1.doc
-  GPI - TD3 - a - Taches partie 2 - 2.2.doc
-  GPI - TD3 - a - Taches partie 2 - 2.3.doc

## ► Considérer le produit comme un singleton

| Documents > Recherche > collaboration > 2016 - CancerRegistry - fuzzy - PaperAccepted > submission                               |                  |                    |          |
|--|------------------|--------------------|----------|
| Nom  | Modifié le       | Type               | Taille   |
|  Fuzzyfactor_08122016_clean.docx                | 14/12/2016 16:25 | Document Micros... | 1 525 Ko |
|  Fuzzyfactor_08122016_clean-JM14122016.docx     | 14/12/2016 19:39 | Document Micros... | 1 731 Ko |
|  Fuzzyfactor_08122016_clean-JM14122016_gu.docx | 10/01/2017 18:34 | Document Micros... | 1 736 Ko |
|  Fuzzyfactor_v0.5.docx                        | 11/01/2017 13:23 | Document Micros... | 1 737 Ko |
|  Fuzzyfactor_v0.6.docx                        | 17/01/2017 17:21 | Document Micros... | 1 737 Ko |
|  Fuzzyfactor_v0.7.docx                        | 17/01/2017 01:02 | Document Micros... | 1 873 Ko |

## ► Utiliser des jetons

# Bonne pratique de sauvegarde : 3-2-1

- ▶ 3 copies
- ▶ 2 supports
- ▶ 1 sauvegarde « hors site »



C'est un minimum, il vaut mieux tout tripler.  
Mais gérer cela à la main ne sera pas efficaces.

# Confronté à la réalité

---

- ▶ En informatique le produit est duplicable
- ▶ 2 personnes peuvent travailler sur la même chose en même temps
- ▶ Des dépendances peuvent être cassées
  - ▶ « Mon code marchait hier, j'ai rien touché pourtant »
- ▶ Un logiciel c'est une multitude d'éléments
  - ▶ « Qui a la version originale de ce code ? »
- ▶ Effet tunnel au sein même d'une équipe
  - ▶ « Comment on fait marcher ensemble ces classes maintenant qu'on a mis des semaines à les programmer séparément ? »

# Gestion de Versions

---

- ▶ Utile pour éviter une gestion de version manuelle
  - ▶ sauvegarde à distance, régulièrement, sans perte, ni régression
    - ▶ De toutes les versions et tout l'historique
  - ▶ Permet de gérer le partage des contributions
    - ▶ Résout ou aide à résoudre les conflits
    - ▶ Gère les variantes du projet
- ▶ Un système de Gestion de Versions indispensable aux projets informatiques
  - ▶ Particulièrement adapté pour versionner du texte (e.g. du code)  
*(ce que nous considérons dans ce cours)*
  - ▶ Les autres produits d'un projet informatique peuvent aussi être versionnés avec ces systèmes mais pour gérer les conflits (nécessitant de comparer leur contenu) ils ont souvent leur propre système :
    - ▶ SGBD,
    - ▶ Contenu Multimédia,
    - ▶ Gestion de versions intégrée aux suites bureautiques.



# Emplacement client et dépôt

---

- ▶ Chaque acteur travaille en un **emplacement client** sur sa **copie de travail**
- ▶ Le produit est stocké sur un **dépôt** qui est le **référentiel**
  - ▶ Ce dépôt peut être local, distant, centralisé, décentralisé.
  - ▶ Chaque acteur en récupère sa copie de travail
    - ▶ Plusieurs versions si besoin, en un ou plusieurs emplacement(s) client
  - ▶ Chaque acteur met à jour le dépôt
    - ▶ en fonction de ses progrès,
    - ▶ en fonction du cycle de développement.

# Quelques outils de Gestions de Versions

---

## ▶ Dépôt centralisé

- ▶ CVS
  - ▶ <http://www.nongnu.org/cvs/>
- ▶ SVN (Subversion)
  - ▶ <http://subversion.apache.org/>

## ▶ Dépôt décentralisé

- ▶ GNU Arch
  - ▶ <http://www.gnu.org/software/gnu-arch/>
- ▶ Mercurial
  - ▶ <http://mercurial.selenic.com/>
- ▶ Git
  - ▶ <https://git-scm.com/>
- ▶ ...



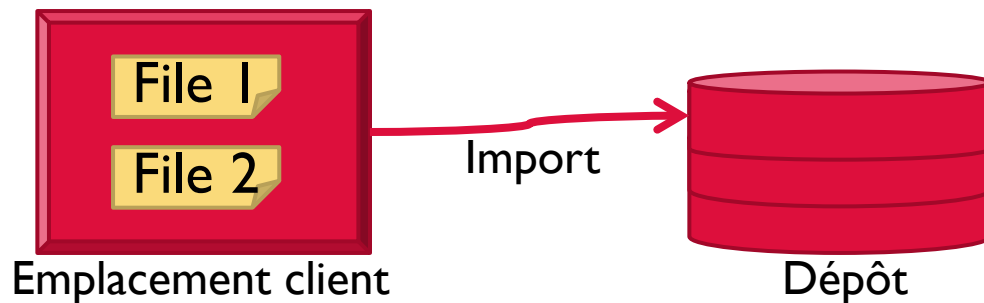
Dépôt centralisé - SVN



# Dépôt (ou référentiel/repository)

---

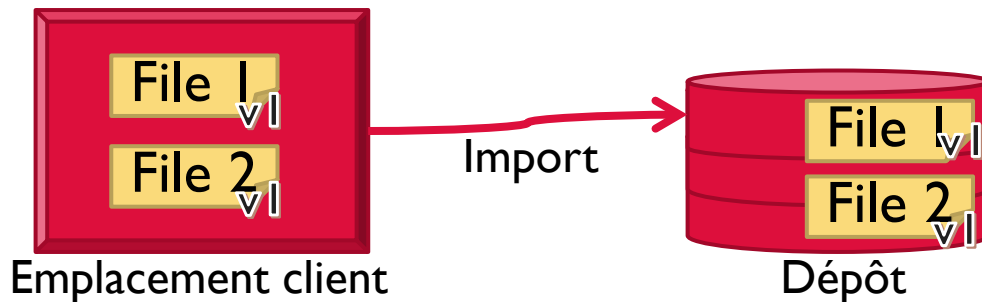
- ▶ Le dépôt est
  - ▶ créé par un administrateur,
  - ▶ accessible aux acteurs autorisés,
  - ▶ différemment hébergé.
- ▶ Une première version du projet est déposée
  - ▶ On importe sur le dépôt les prémices du produit



# Dépôt (ou référentiel/repository)

---

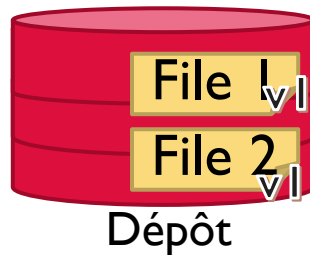
- ▶ Le dépôt est
  - ▶ créé par un administrateur,
  - ▶ accessible aux acteurs autorisés,
  - ▶ différemment hébergé.
- ▶ Une première version du projet est déposée
  - ▶ On importe sur le dépôt les prémices du produit
  - ▶ Attention tout étant automatiquement versionné, on ne parle pas ici d'une version 1.0 mais du premier dépôt.



# Récupération

---

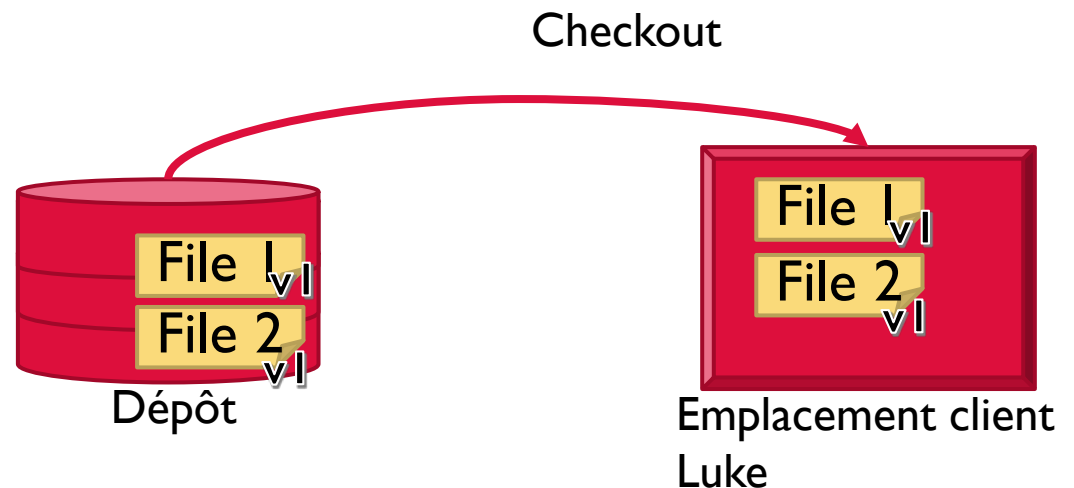
- ▶ Récupérer une version du dépôt
  - ▶ « checkout »
  - ▶ Typiquement pour travailler sur un nouvel emplacement client



# Récupération

---

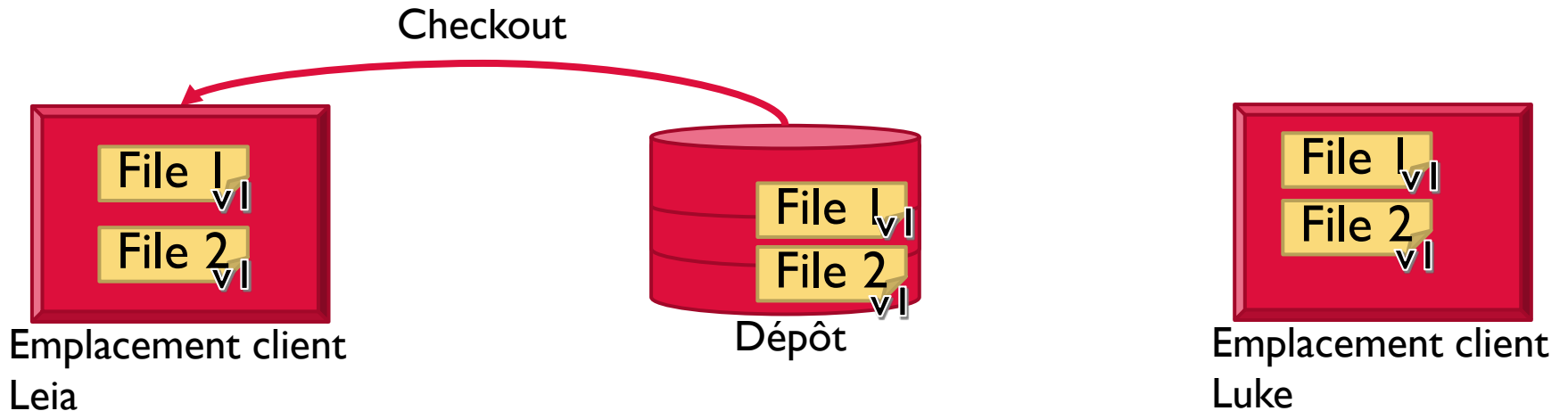
- ▶ Récupérer une version du dépôt
  - ▶ « checkout »
  - ▶ Typiquement pour travailler sur un nouvel emplacement client



# Récupération

---

- ▶ Récupérer une version du dépôt
  - ▶ « checkout »
  - ▶ Typiquement pour travailler sur un nouvel emplacement client

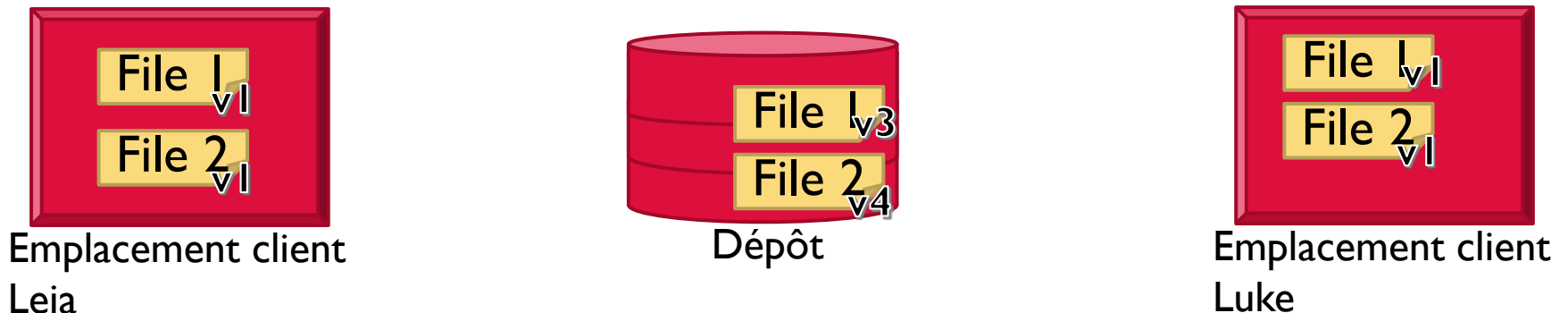




# Mise à jour

---

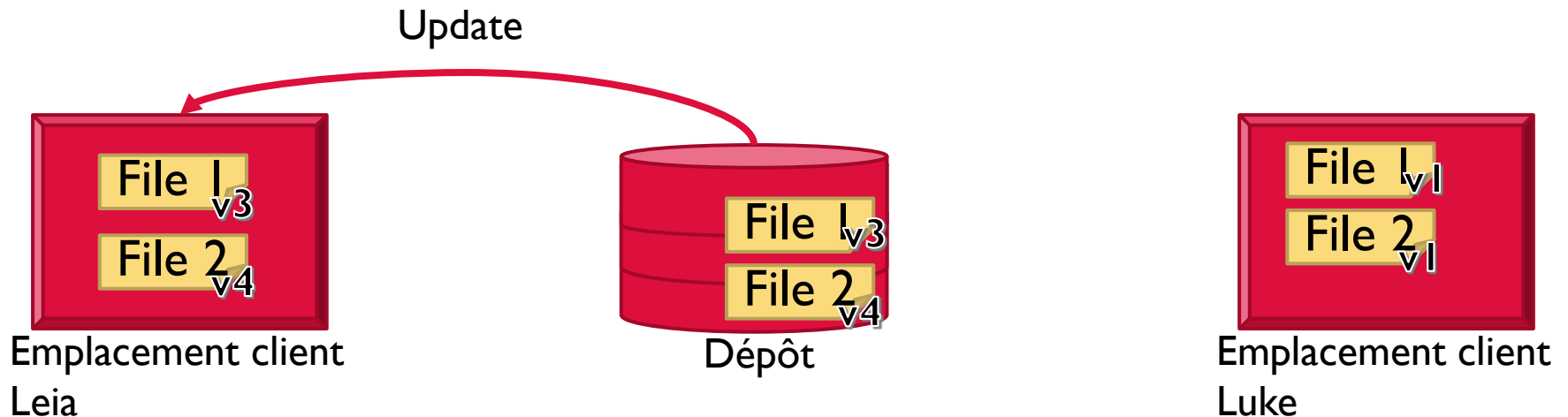
- ▶ **Mettre à jour l'emplacement client**
  - ▶ Par fichier(s), dossier(s)
  - ▶ Quand le dépôt a évolué, il faut faire évoluer sa version de travail



# Mise à jour

---

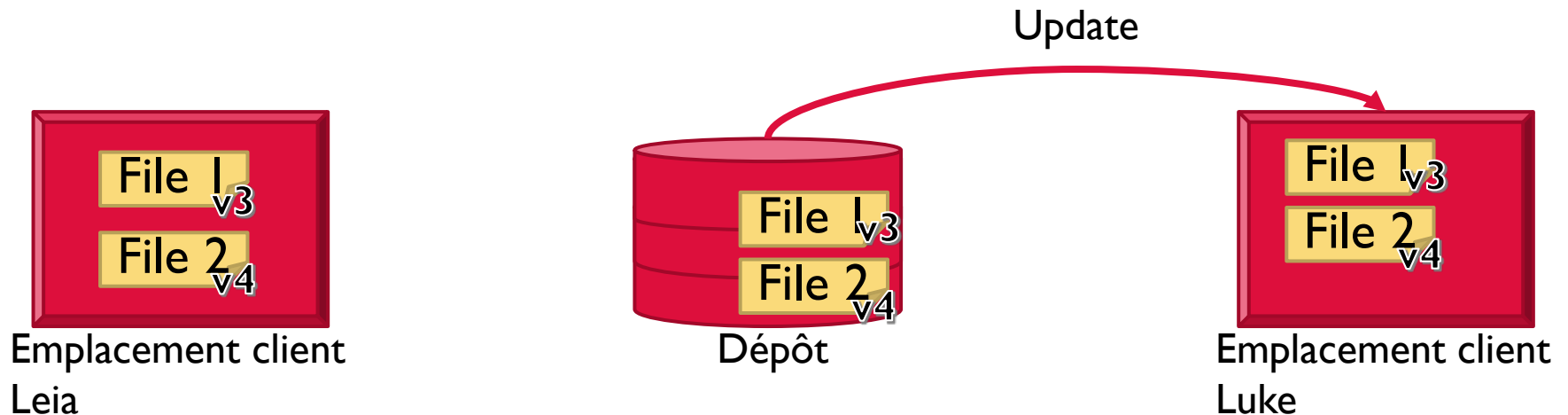
- ▶ **Mettre à jour l'emplacement client**
  - ▶ Par fichier(s), dossier(s)
  - ▶ Quand le dépôt a évolué, il faut faire évoluer sa version de travail



# Mise à jour

---

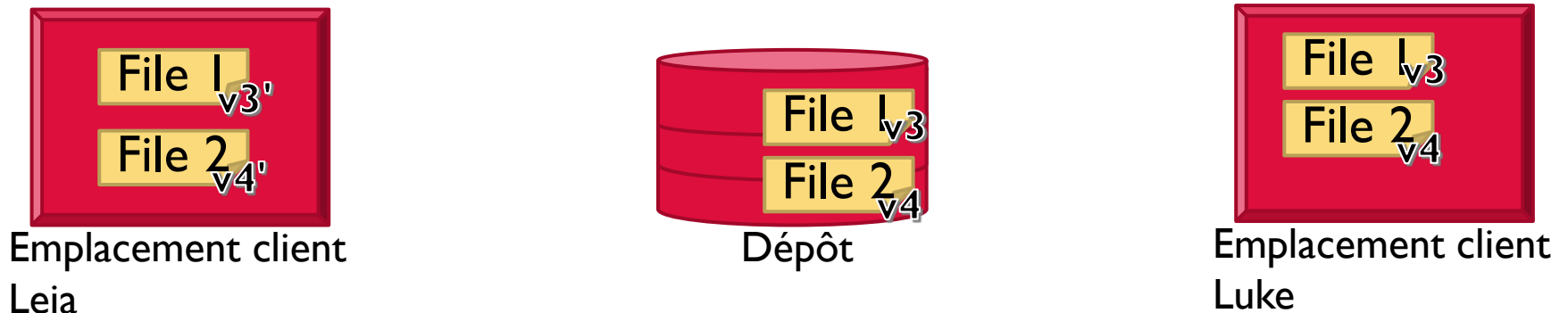
- ▶ **Mettre à jour l'emplacement client**
  - ▶ Par fichier(s), dossier(s)
  - ▶ Quand le dépôt a évolué, il faut faire évoluer sa version de travail



# Soumission

---

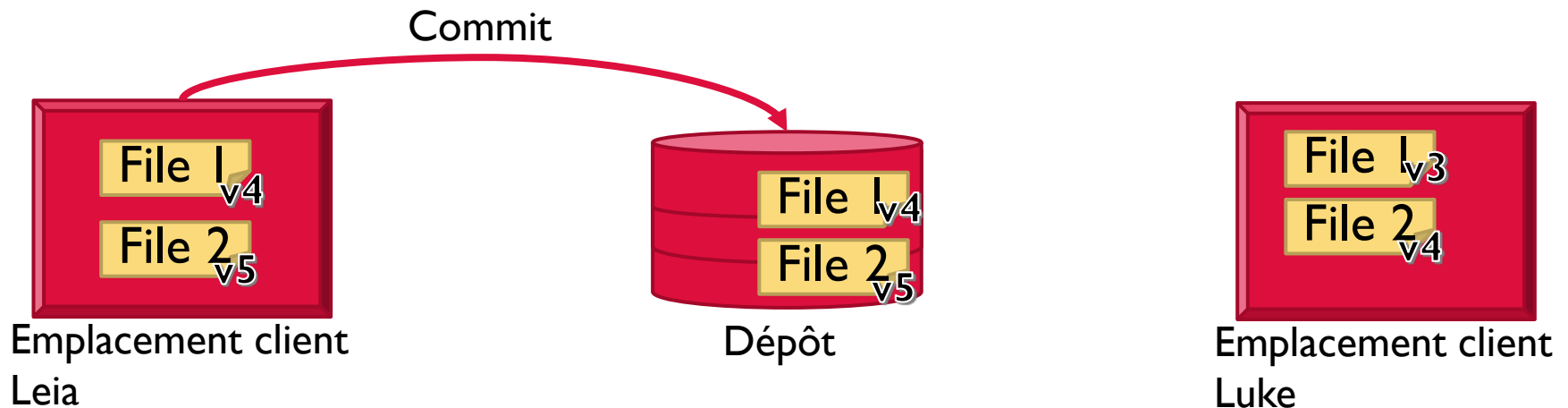
- ▶ Les modifications de la version de travail sont soumises au dépôt qui est mis à jour.
- ▶ Avant de soumettre sur le dépôt, il faut mettre à jour sa version de travail.



# Soumission

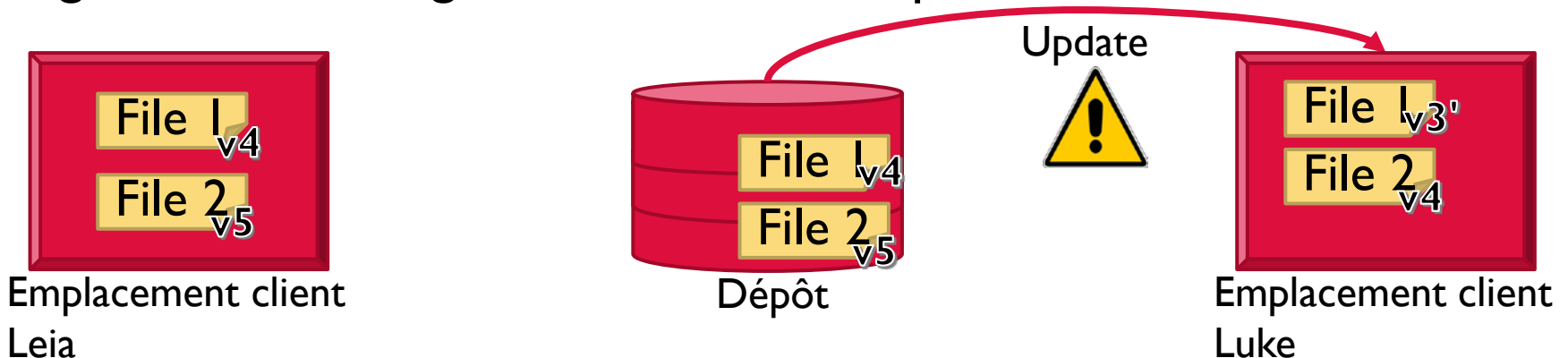
---

- ▶ Les modifications de la version de travail sont soumises au dépôt qui est mis à jour.
  - ▶ Avant de soumettre sur le dépôt, il faut mettre à jour sa version de travail.
  - ▶ En faisant un commit, on versionne, sauvegarde, partage son travail



# Conflits

- ▶ A la mise à jour de sa version de travail des conflits peuvent apparaître :
  - ▶ Les mêmes lignes d'un fichier ont été modifiées à la fois sur la version de travail et sur le dépôt
- ▶ Quand on tente une soumission alors que le dépôt a évolué depuis la précédente mise à jour :
  - ▶ Il faut remettre à jour sa version de travail
- ▶ Les conflits peuvent être résolus automatiquement par le gestionnaire ou gérés manuellement par l'utilisateur.



# Commandes SVN en ligne de commande

---

- ▶ **Création d'un référentiel**
  - ▶ `svnadmin create <repository>`
- ▶ **Importation des fichiers initiaux au référentiel**
  - ▶ `svn import -m "message" <source> protocole>/<repository>`
- ▶ **Récupération d'une copie locale**
  - ▶ `svn checkout <protocole>/<repository>[/<repertory>]`
- ▶ **Mise à jour de la copie locale**
  - ▶ `svn update [<repertory>/<file>]`
  - ▶ Peut entraîner des conflits
- ▶ **Soumission des modifications locales vers le référentiel**
  - ▶ `svn commit -m "message" [<repertory>/<file>]`
  - ▶ Peut nécessiter un update



# Dépôt décentralisé - GIT

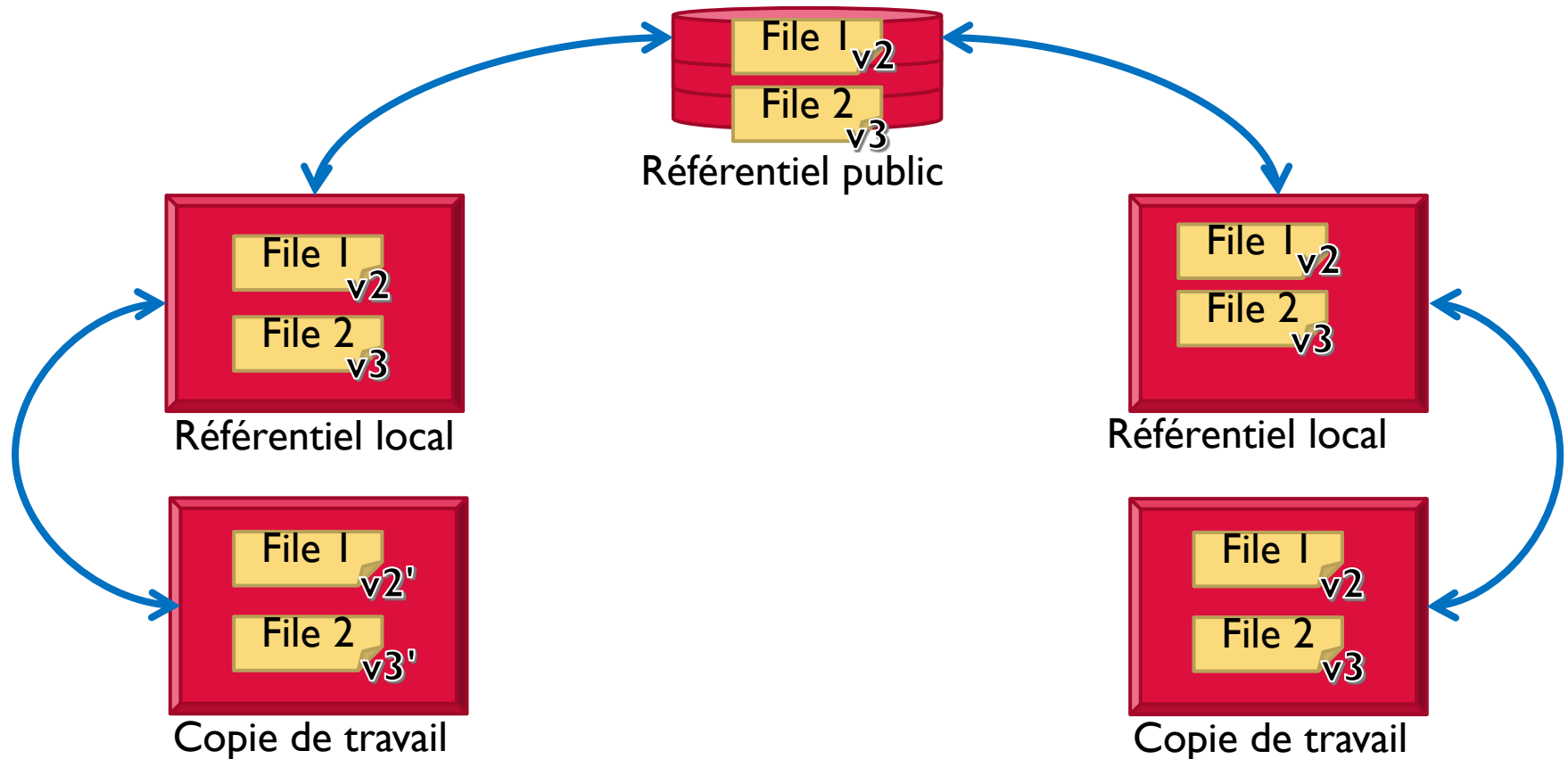




# Référentiels local et public

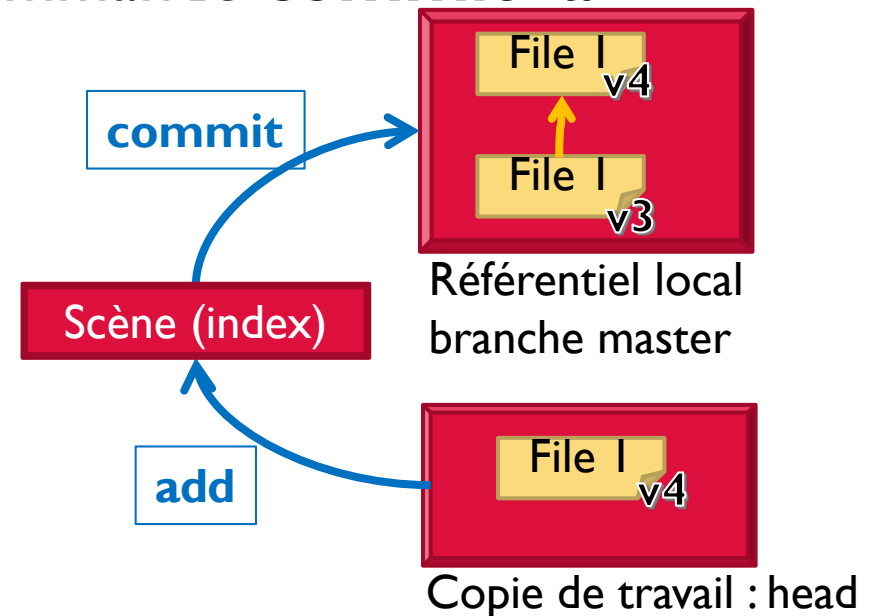
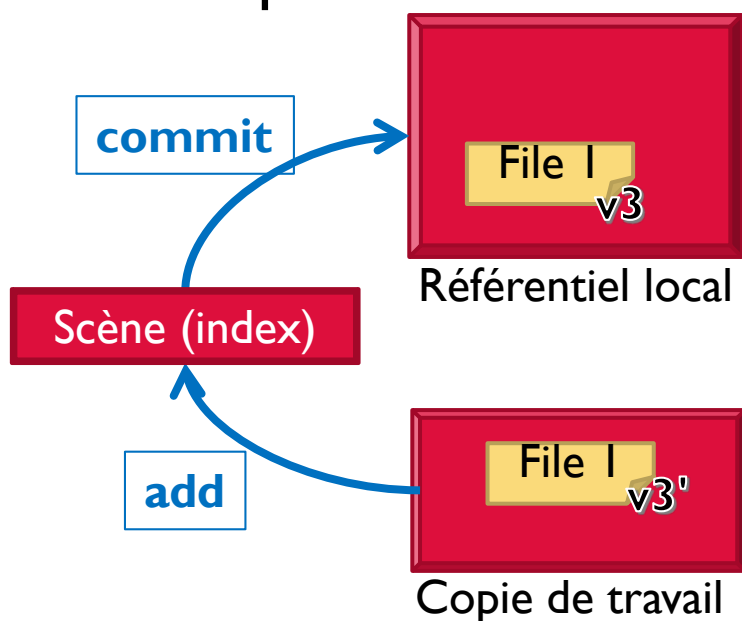
## Vue simplifiée (i.e. sans branches)

- Chacun versionne et sauvegarde son travail localement avant d'en faire le partage public



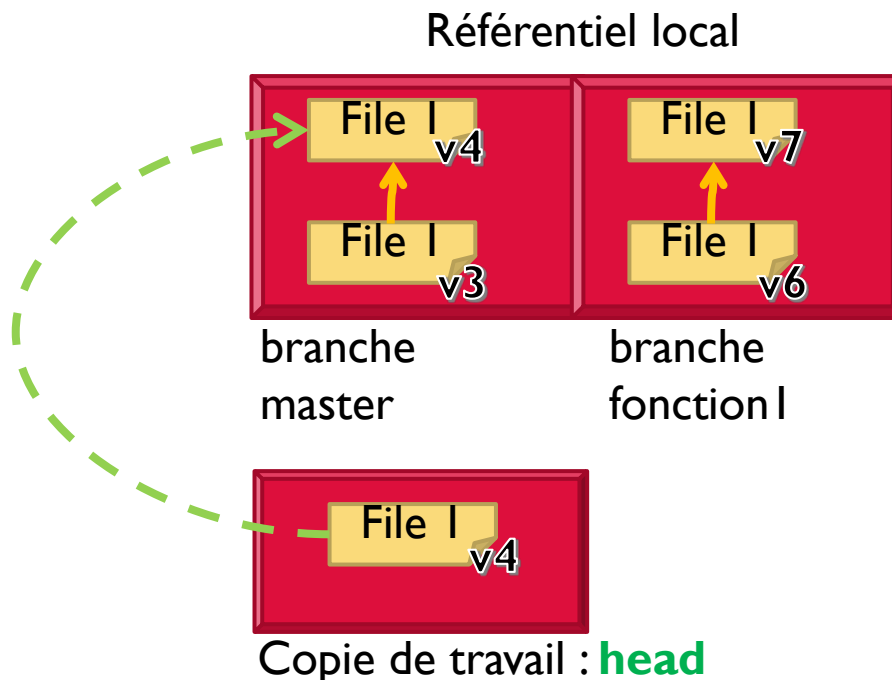
# Gestion localement entre la version de travail et le référentiel local

- ▶ Quand le nouveau contenu d'un fichier doit être versionné, on le met en scène « stage » en le mettant dans l'index : **commande add**
- ▶ Quand un fichier en scène a évolué, on le sauvegarde/versionne : **commande commit**
- ▶ On peut raccourcir avec la commande **commit -a**



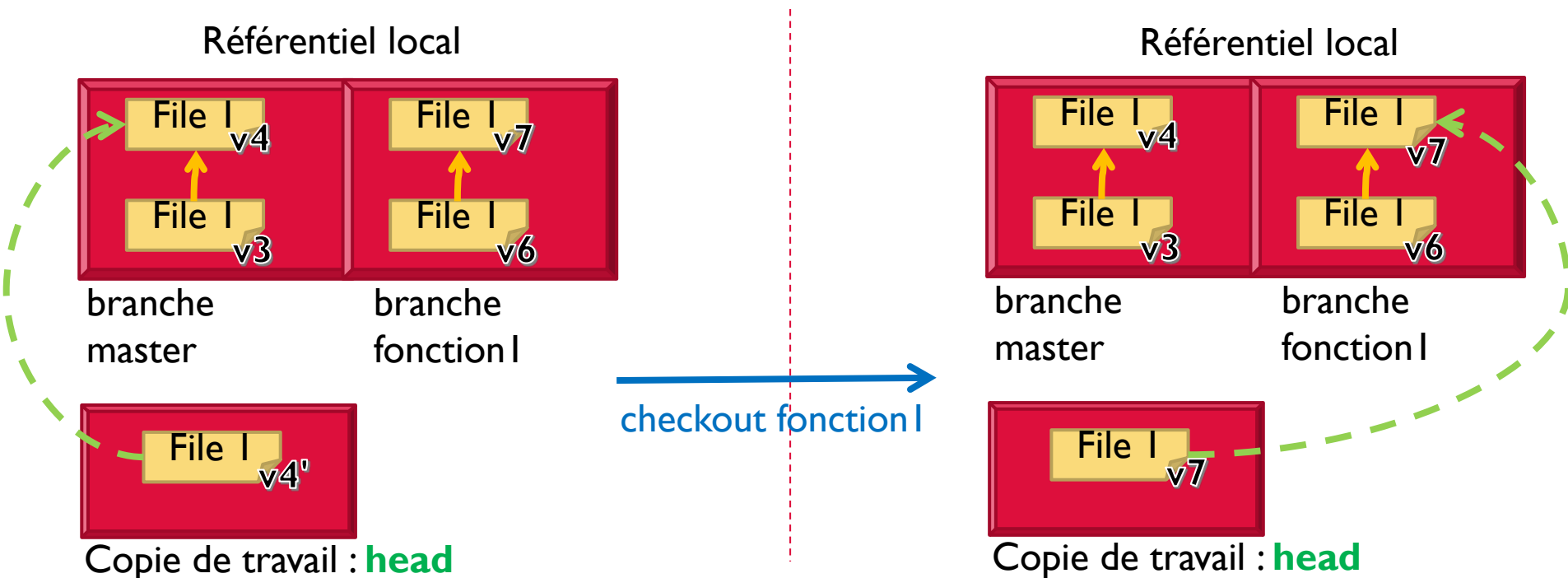
# Branche et états

- ▶ L'état HEAD de la copie de travail désigne un état d'une branche du référentiel local
- ▶ Le référentiel local peut sauvegarder plusieurs branches



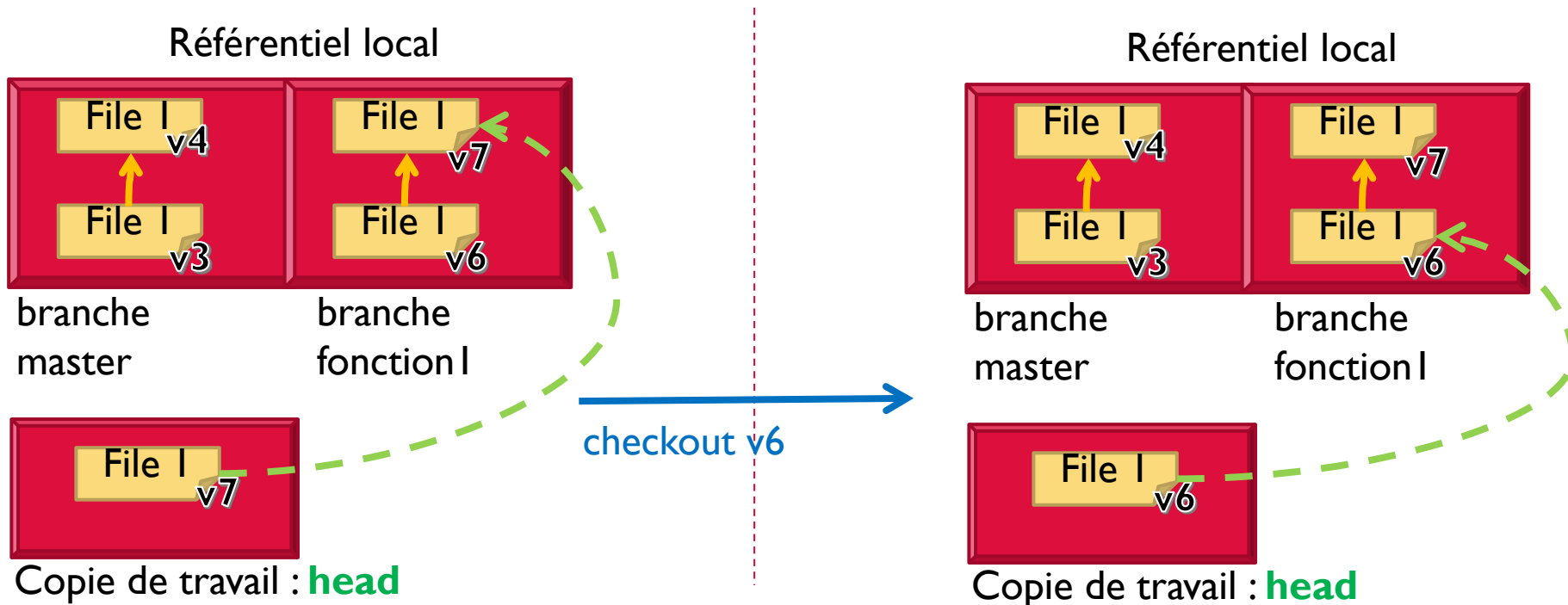
# Changement d'état

- ▶ La commande checkout permet de changer d'état
  - ▶ D'une branche à l'autre
    - ▶ Pour revenir à la version du référentiel (annulant les changements de la copie de travail)
  - ▶ D'un état d'une branche à un autre
    - ▶ Pour revenir en arrière (annulant des commits)



# Changement d'état

- ▶ La commande checkout permet de changer d'état
  - ▶ D'une branche à l'autre
    - ▶ Pour revenir à la version du référentiel (annulant les changements de la copie de travail)
  - ▶ D'un état d'une branche à un autre
    - ▶ Pour revenir en arrière (annulant des commits)



# Suivi du travail local

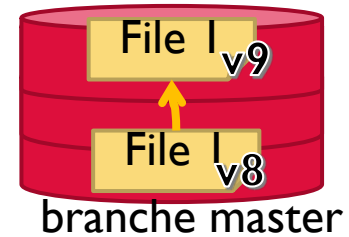
---

- ▶ Savoir où on en est entre la version de travail et le référentiel local : **commande status**
- ▶ Consulté l'historique : **commande log**
- ▶ Marquée une version : **commande tag**

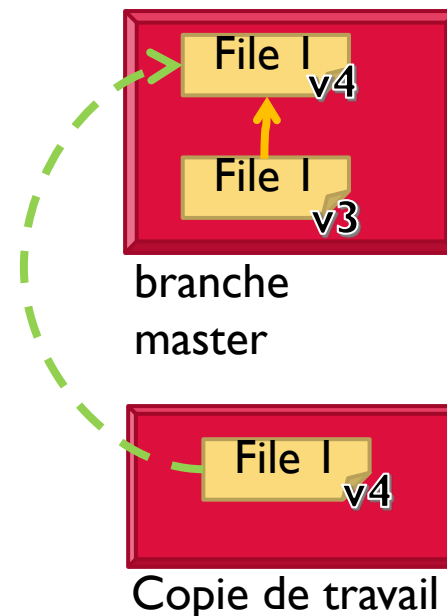
# Récupération (pull ) depuis le référentiel public : Télécharger (fetch) puis Fusionner (merge)

- ▶ Télécharger la version du référentiel public en local : **commande fetch**
- ▶ Cela récupère les nouveautés mais dans une branche séparée pour pouvoir maîtriser leur intégration

Référentiel public : origin

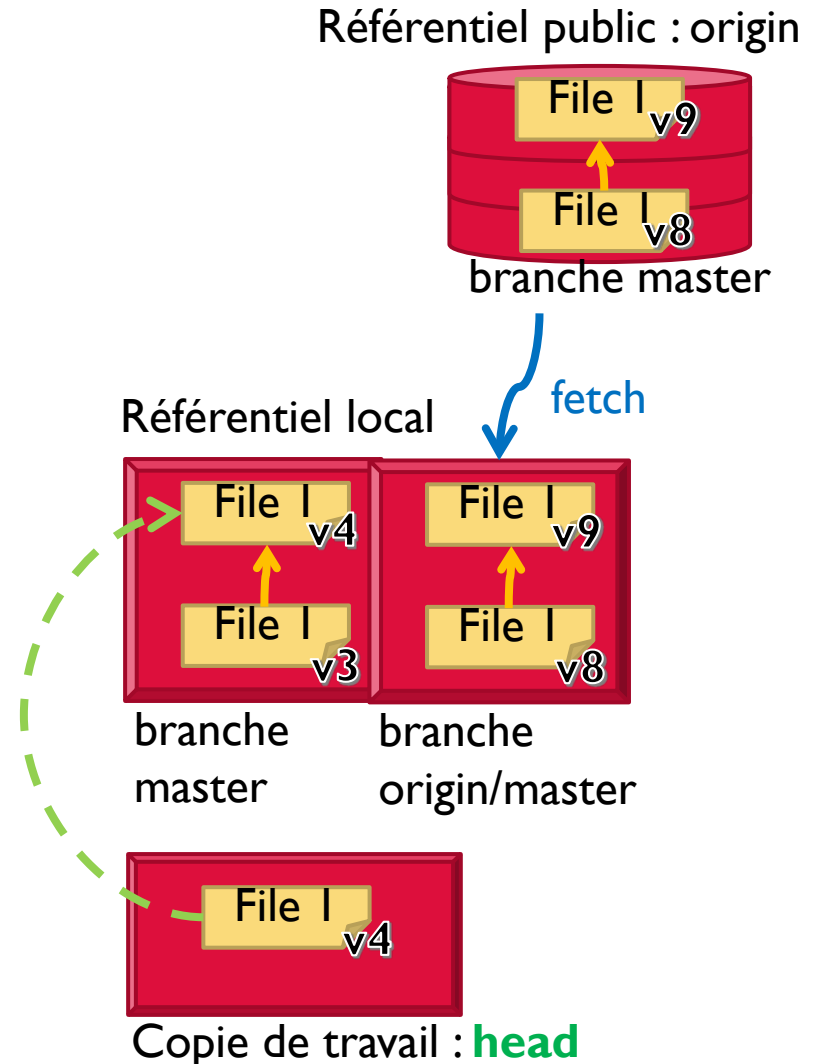


Référentiel local



# Récupération (pull ) depuis le référentiel public : Télécharger (fetch) puis Fusionner (merge)

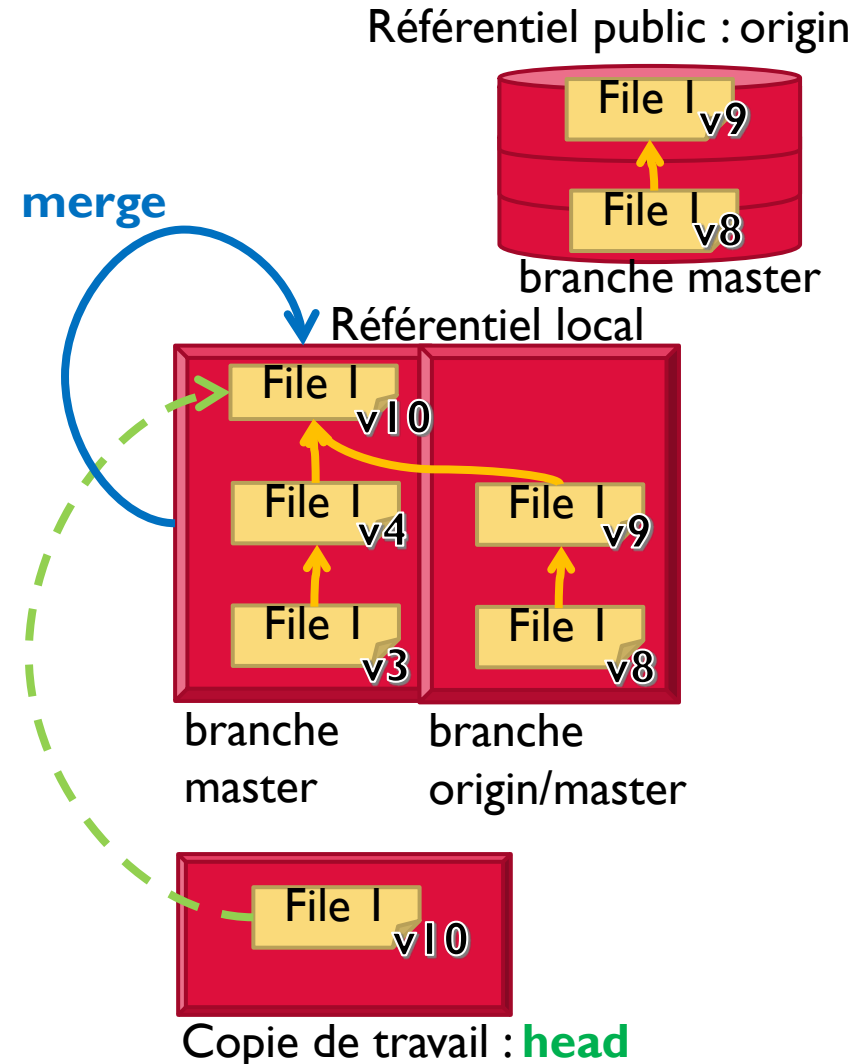
- ▶ Télécharger la version du référentiel public en local : **commande fetch**
- ▶ Cela récupère les nouveautés mais dans une branche séparée pour pouvoir maîtriser leur intégration





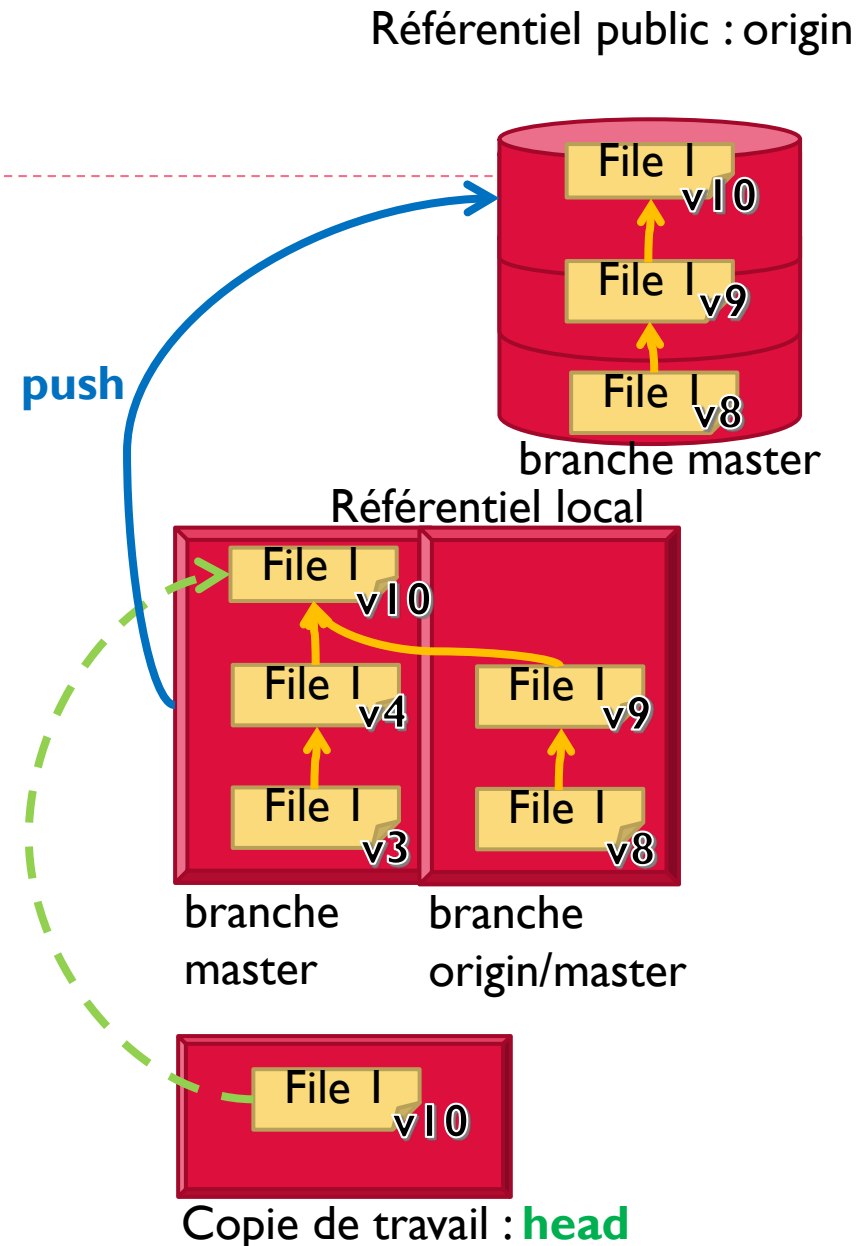
# Récupération (pull ) depuis le référentiel public : Télécharger (fetch) puis Fusionner (merge)

- ▶ Fusionner les branches pour intégrer les nouveautés du référentiel public : **commande merge**
- ▶ On peut faire **fetch** et **merge** en même temps avec la **commande pull** mais au risque de difficulté dans le merge



# Partage (push) avec le référentiel public

- Envoi de ces contributions au référentiel public : **commande push**



# Cycle habituel

---

- ▶ Récupérer une version publique
  - ▶ `git pull`
- ▶ Travailler (sur le code)
  - ▶ codage
- ▶ Versionner (Sauvegarder)
  - ▶ `git add`
  - ▶ `git commit`
- ▶ Récupérer la dernière version publique
  - ▶ `git pull`
- ▶ S'il y a des conflits, les résoudre (dans le code), les versionner
  - ▶ codage
  - ▶ `git commit -a`
- ▶ Partager son travail (Sauvegarder à distance)
  - ▶ `git push`

# Forge

---

- ▶ On appelle une «Forge» un serveur web dédié à l'hébergement de projets informatiques.
- ▶ Généralement, une forge fournit les services suivants:
  - ▶ un ou des systèmes de gestion des versions (par exemple, Git ou Mercurial);
  - ▶ un gestionnaire de listes de discussion (et/ou des forums);
  - ▶ un outil de suivi des bugs (eg, Jira, Redmine, Bugzilla);
  - ▶ gestionnaire de documentation (wiki);
  - ▶ gestionnaire des tâches.

# Exemples de forge

---

- ▶ GitHub : Plus de 100 millions de projets hébergés.
- ▶ SourceForge : Plus de 500.000 projets.
- ▶ Bitbucket : Plus de 6 millions de projets ouverts en 2017.
- ▶ GitLab : Logiciel libre permettant de déployer ses forges
  - ▶ comme GitLab à l'Université de Nantes
    - ▶ <https://gitlab.univ-nantes.fr/>

# gitlab.univ-nantes.fr

---

- ▶ Intègre l'outil de gestion de versions git
  - ▶ Avec en plus une interface web de modification et d'analyse de code
- ▶ Gère la documentation
- ▶ Gère des tâches et leur suivi (tableau kanban)
- ▶ S'intègre avec d'autres outils de gestion de projet, en particulier mattermost pour la discussion instantannée
  - ▶ [mattermost.univ-nantes.fr](https://mattermost.univ-nantes.fr)
- ▶ Gère l'intégration continue, etc.

# Travailler avec une forge

---

- ▶ Récupérez un dépôt en faisant un clone
  - ▶ copie un projet d'un référentiel publique dans un nouveau référentiel à part
    - ▶ Soit pour démarrer un nouveau projet
    - ▶ Soit pour contribuer au projet original mais en travaillant à part
- ▶ Partager ses contributions
  - ▶ On fera le versionnage dans des branches (à moins d'être le propriétaire)
  - ▶ On n'enverra pas ses contributions au projet original mais on lui proposera de les récupérer : pull request