Les systèmes de gestion de version

Matthieu Herrb

LAAS-CNRS

Envol 2010

http://homepages.laas.fr/matthieu/talks/envol10-sgv.pdf

Systèmes de gestion de version - kesako?

Logiciel permettant de gérer l'historique des modifications d'un ensemble de documents.
Typiquement : les codes source d'un logiciel.
Mais aussi :

- documentation
- site web
- fichiers de configuration
- etc.

Fonctions de base

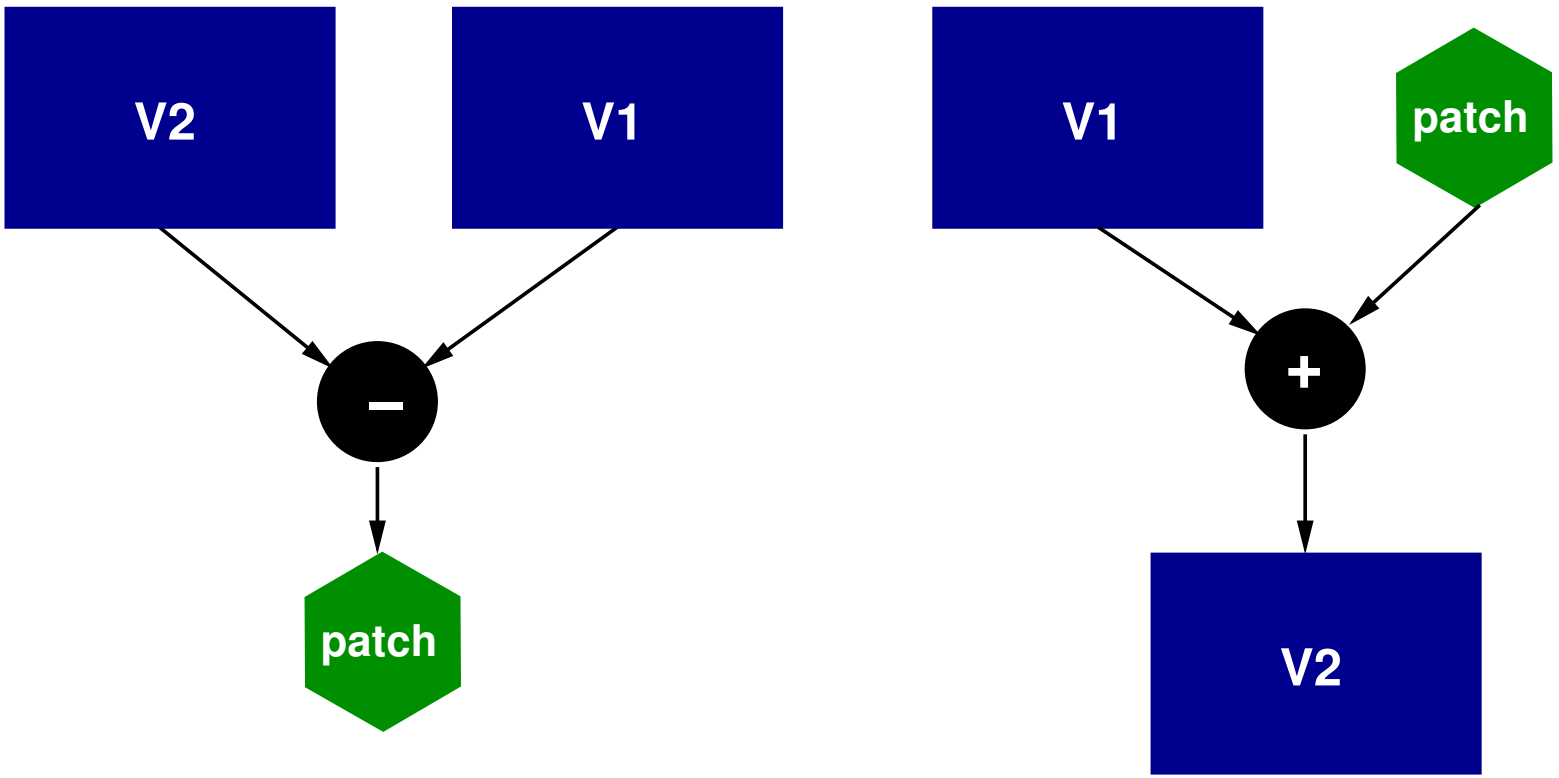
- conserver un historique des modifications
- permettre travailler à plusieurs (verrous, gestion des conflits)
- permettre les modifications en parallèle (branches)
- garantir la sécurité (intégrité, disponibilité, confidentialité)

Comment faire?

- gestion manuelle de plusieurs copies des fichiers
- logiciels dédiés

Agenda

Diff et patch



diff texte

Représentation des différences entre 2 versions d'un fichier.

`diff(1)` produit un diff de 2 fichiers texte :

```
--- a/src/server.c
+++ b/src/server.c
@@ -222,7 +222,9 @@ reset_log(void)
 #ifdef HAVE_SS_LEN
 #define sockaddr_len(s) s.ss_len
 #else
-#define sockaddr_len(s) sizeof(s)
+#define sockaddr_len(s) (s.ss_family == AF_INET6 ? \
+                         sizeof(struct sockaddr_in6) \
+                         : sizeof(struct sockaddr_in))
 #endif

 void
```

commande patch

- *patch :* Pièce qui permet de passer d'une version à la suivante.
- `patch(1)` commande Unix qui utilise un diff comme entrée.

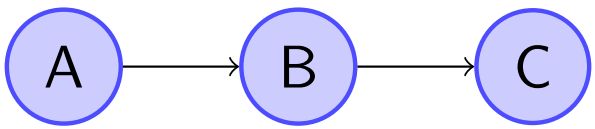
Exemple :

```
# patch -p1 -E < diff
```

`patch` peut gérer des petites incohérences grâce au contexte

Concepts de base (1)

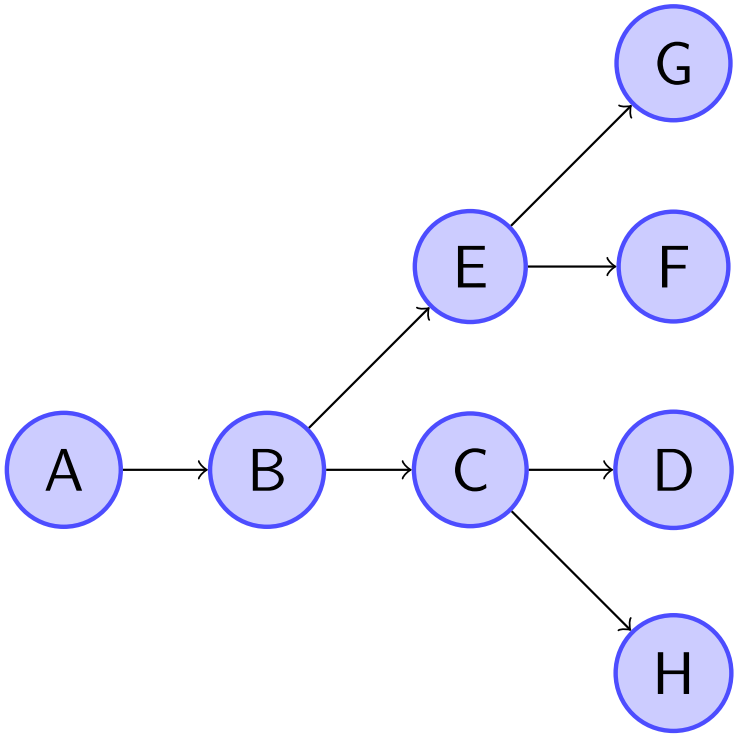
- **Dépot** (*repository*)
  Répertoire ou espace de stockage quelquonque: conserve l'historique des modifications
- **Révision**
  Chaque état des données a un identificateur unique
  $\rightarrow$ **révision**.
  Également appelée **commit** par abus de langage.



  Séquence ordonnée.
- avec CVS: révision par fichier
  $\rightarrow$ **version du projet** $\neq$ révision d'un fichier !

Concepts de base (2)

- **Branches**



Gestion des branches

Pour :

- corriger un problème sur une ancienne version
- développer plusieurs idées en parallèle
- gérer sa propre version d'un logiciel
- fusionner après une divergence.
- ...

Concepts de base (3)

- **Tags**
  Marques symboliques sur une révision.
  Permettent de définir les versions du projet.
  Permettent de nommer des branches.

Travailler à plusieurs

- Pas de verrou sur les sources.
  Chacun a sa propre copie.
- Gestion des conflits:
  - d'abord intégrer les modifications des autres
  - fusion automatique
  - détection des conflits $\rightarrow$ résolution à la main
  - pas de nouveau `commit` avant résolution du conflit

Autres fonctions d'un SGV

- Visualisation de l'historique sous diverses formes
- Exécution automatique de scripts avant/après commit
  - Tests de validation,
  - Envoi d'e-mail après commit.
- Annotation du code avec les contributions
- Recherche dichotomique de regressions
- Import/export vers d'autres SGV
- ...

Trois modèles de fonctionnement

Local

Fonctionne dans un système de fichiers local. Pas de réseau.

- SCCS, RCS,...

Client/Serveur (ou centralisé)

Un serveur centralise le dépot, accessible à distance.

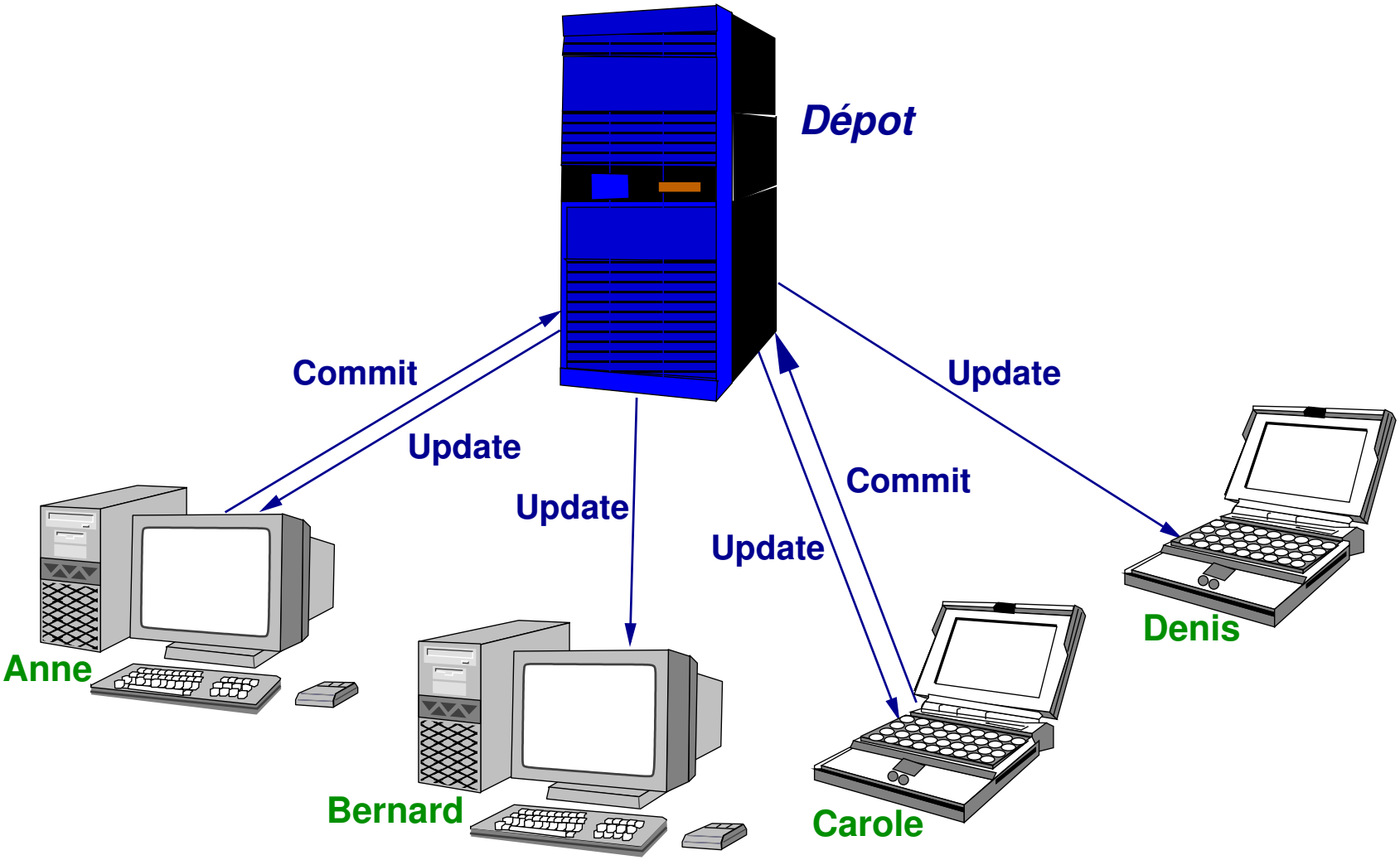
- CVS
- Subversion

Distribué

Multiples copies du dépots, branches locales.

- bitkeeper, monotone, arch, darcs
- mercurial, git, bazaar

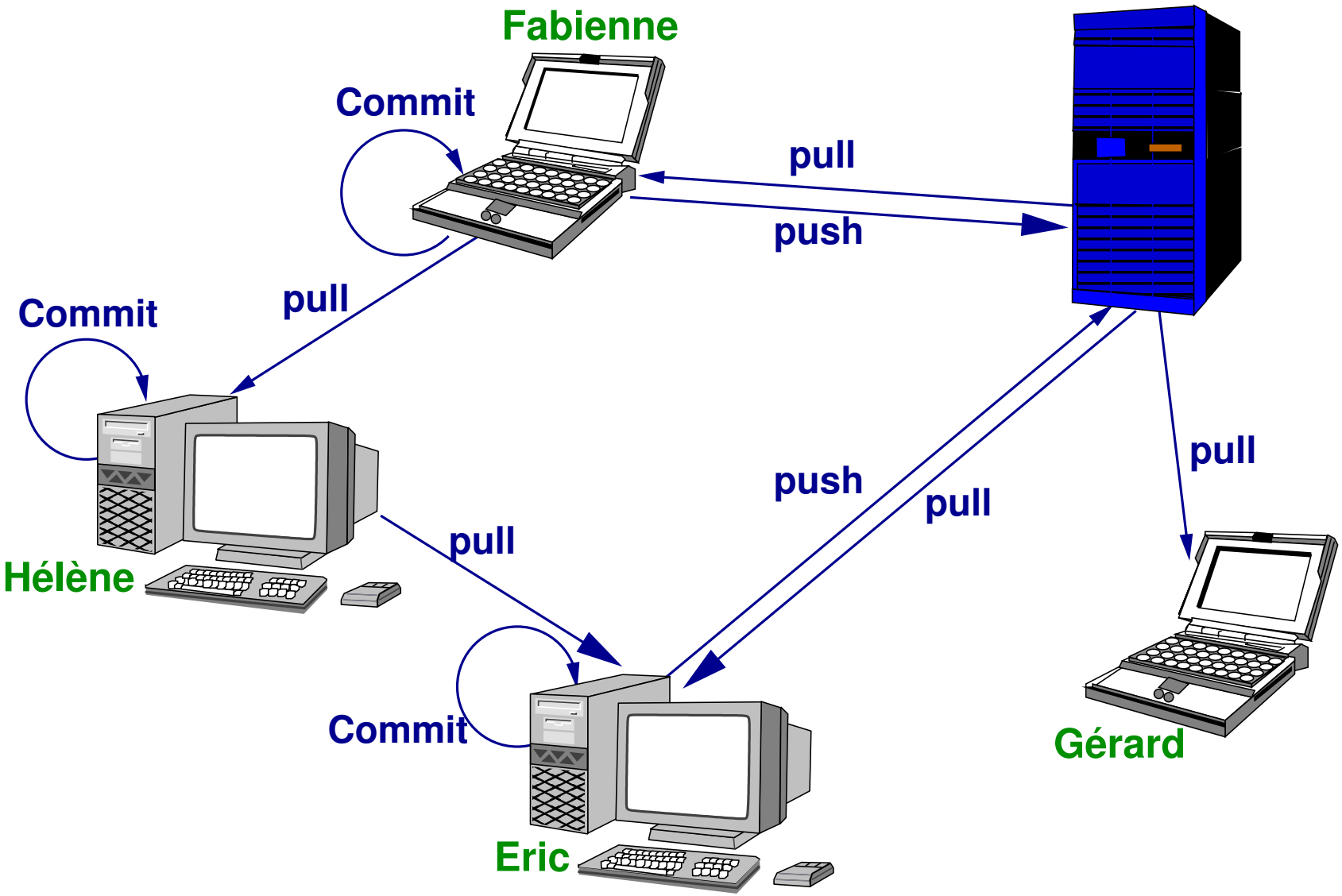
Client-serveur - principe



Client-serveur - principe

- Dépot stocké dans un endroit partagé
  - par le système de fichiers
  - par un mécanisme réseau (rsh/ssh ou protocole dédié)
- Plusieurs copies de travail en parallèle : opérations de fusion.
- Nécessaire d'avoir la connexion au dépot pour committer.
- Le ≪tronc≫ a une importance particulière : modèle très centralisé.

Systèmes distribués - principe



Systèmes distribués - principe

- Plus de dépot centralisé
- Chaque développeur a sa copie avec ses branches privées
- Opérations push/pull : synchronisation avec les autres dépots.
- Simplification de la fusion de branches en gardant l'historique des fusions.
- Influence sur la philosophie de développement : plus de liberté, mais risque de dispersion...

Politiques de management dans les systèmes distribués

Un système distribué peut devenir anarchique :

- pas de notion de branche “principale” ou “de référence”
- chacun résoud les conflits à sa manière...

$\Rightarrow$ nécéssite une politique :

- définir une branche de référence et nommer un responsable
- définir une nomenclature pour les branches partagées
- inciter les développeurs à merger leurs travaux

Agenda

Les éléments de base

- le dépot est dans un sous-répertoire du projet (exemple: `.git/`).
- les commits sont identifiés par un hash (généralement SHA-1) plutôt que par un numéro $\rightarrow$ ordre pas clair...
- **master** désigne la branche principale.
- **HEAD** désigne le commit le plus récent de la branche courante

Utilisation locale

Opérations élémentaires:

init	initialisation du dépot (une seule fois)
add	ajouter des fichiers
commit	enregistrer des modifications
branch	créer ou changer de branche active
merge	fusionner une branche
cherry-pick	applique un commit d'une autre branche
rebase	ré-ordonner des commits d'une autre branche

Commandes de visualisation de l'état

status	affichage d'un résumé de l'état
diff	affichage des diffs des fichiers non commités
log	affichage de l'historique des modifications
show	affichage d'un commit particulier

L'index de git

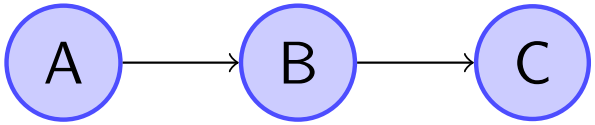
Représente les modifications en attente de commit.
2 étapes :

1. ajouter les fichiers modifiés à l'index (`add`,`rm`)
2. ≪ vider ≫ l'index dans le dépot

Il existe des raccourcis pour enchaîner les 2 pour aller vite.
Pas d'équivalent direct dans mercurial ou baazar.

Commits

Ajoute un noeud en bout de la branche courante.

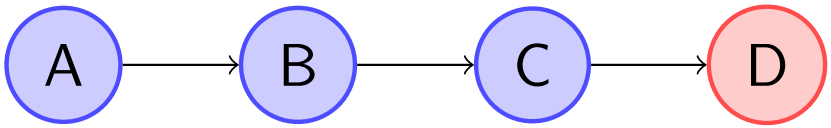


Contient:

- le diff avec la révision précédente pour les fichiers texte
- la nouvelle version complète pour les fichiers binaires
- des infos sur les attributs des fichiers commités (droits d'accès)
- le nom et l'adresse e-mail du committeur
- un message de journal
- en option, une signature numérique

Commits

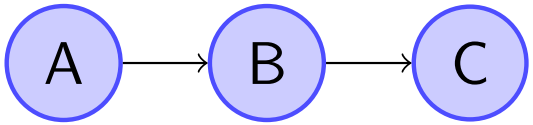
Ajoute un noeud en bout de la branche courante.



Contient:

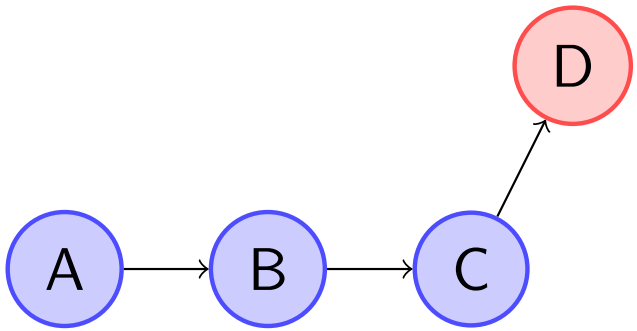
- le diff avec la révision précédente pour les fichiers texte
- la nouvelle version complète pour les fichiers binaires
- des infos sur les attributs des fichiers commités (droits d'accès)
- le nom et l'adresse e-mail du committeur
- un message de journal
- en option, une signature numérique

Branches



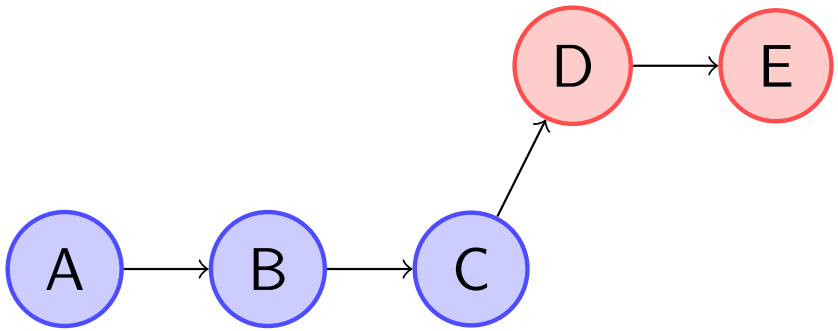
- historique existant

Branches



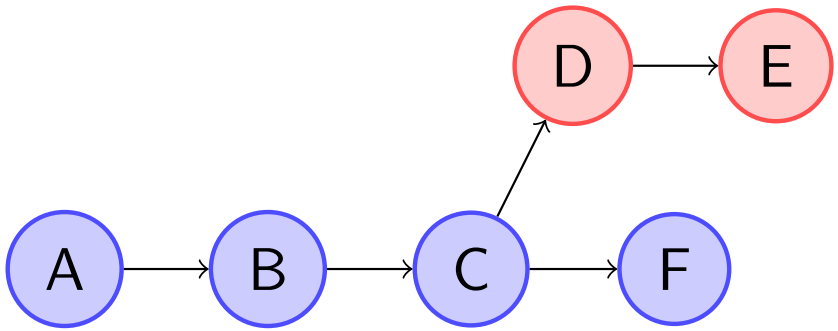
- historique existant
- création d'une branche

Branches



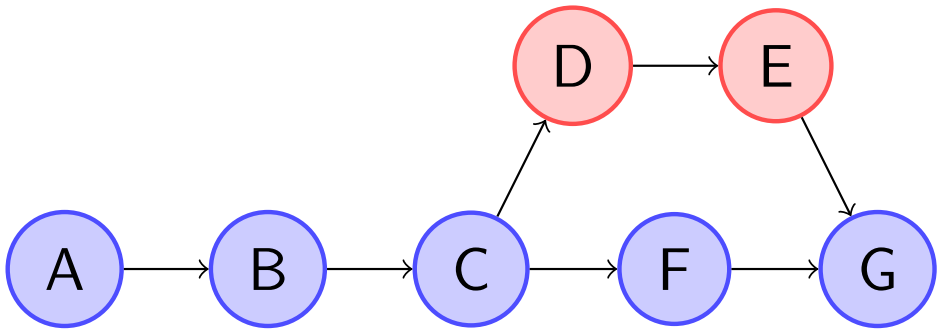
- historique existant
- création d'une branche
- commits dans la branche

Branches



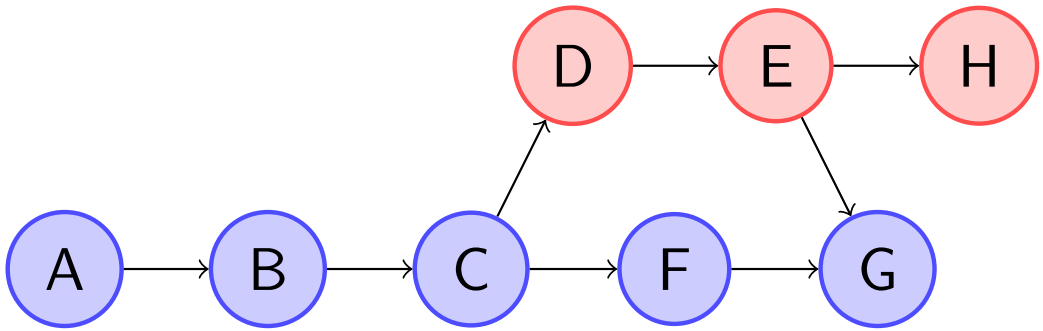
- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*

Branches



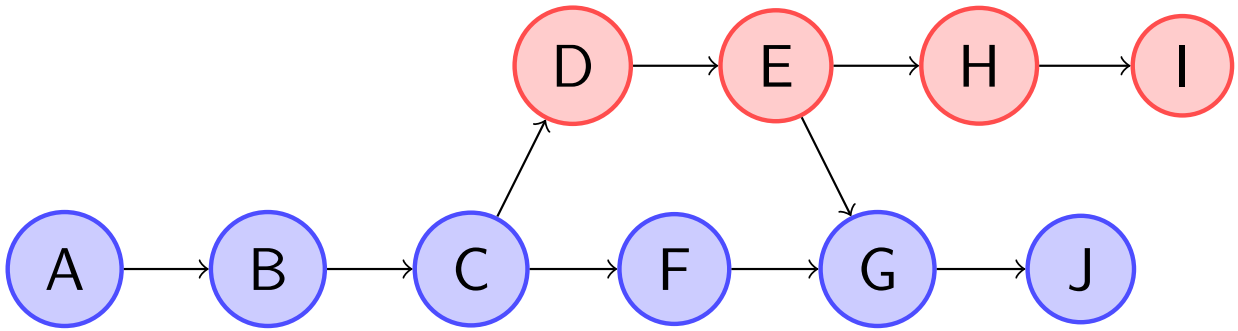
- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*
- fusion de la branche dans *master*

Branches



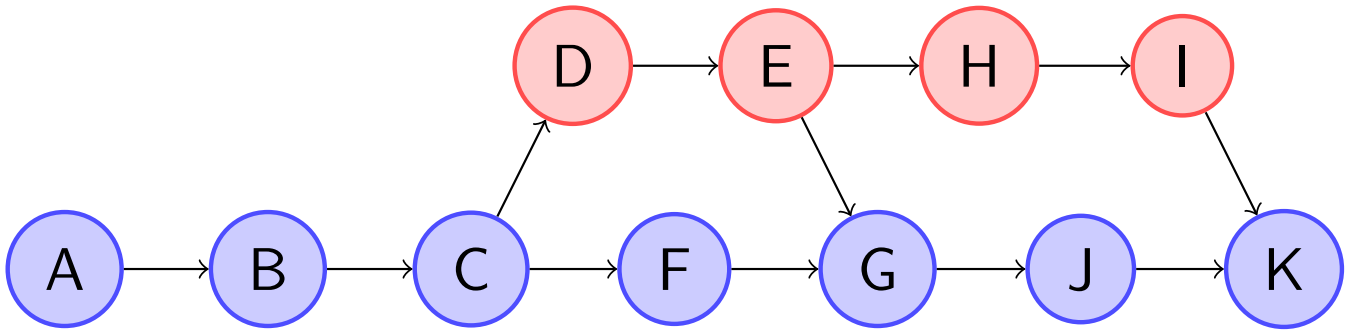
- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*
- fusion de la branche dans *master*
- suite commit dans la branche

Branches



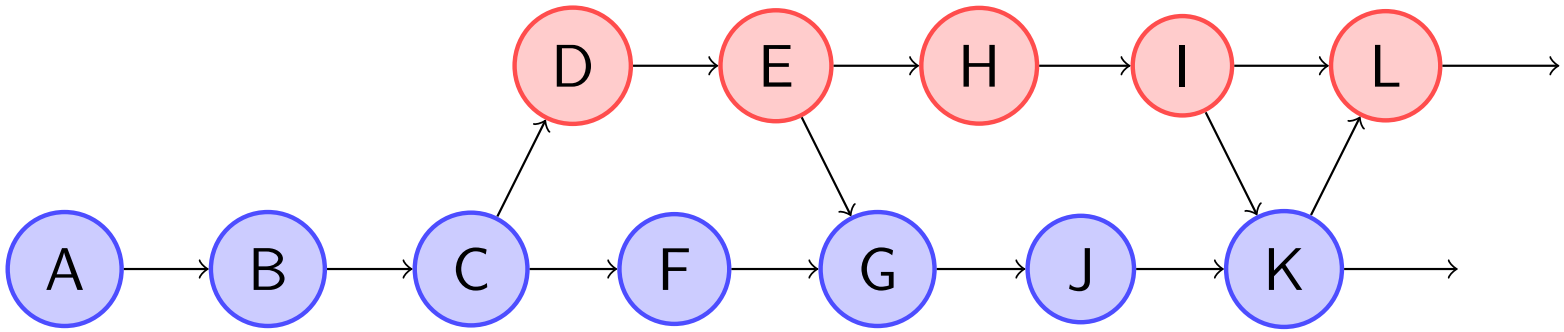
- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*
- fusion de la branche dans *master*
- suite commit dans la branche
- etc...

Branches



- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*
- fusion de la branche dans *master*
- suite commit dans la branche
- etc...

Branches



- historique existant
- création d'une branche
- commits dans la branche
- commit dans *master*
- fusion de la branche dans *master*
- suite commit dans la branche
- etc...

Fusion de branches

Deux types de fusion:

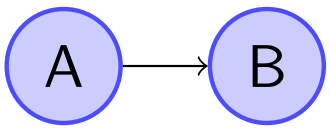
- *Fast-Forward* si aucun recouvrement entre les 2 branches.
- *True Merge* si recouvrement, avec conflits potentiels.

Si vraie fusion, plusieurs stratégies: par défaut *recursive* :

1. recherche récursive d'un ancêtre commun,
2. fusion à partir de cet ancêtre.

Rebase

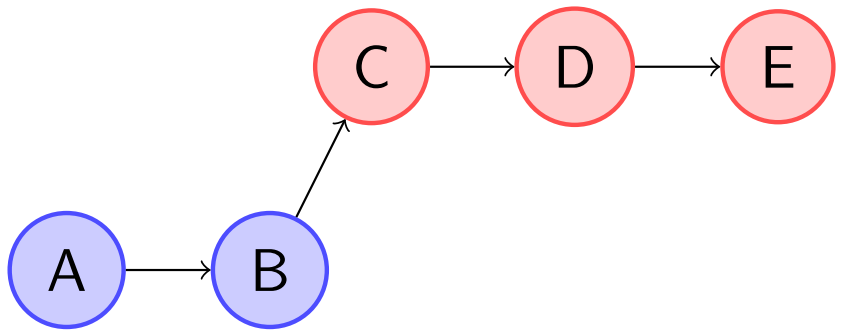
Permet d'éviter trop de merges pour une branche simple non partagée



- Commits existants

Rebase

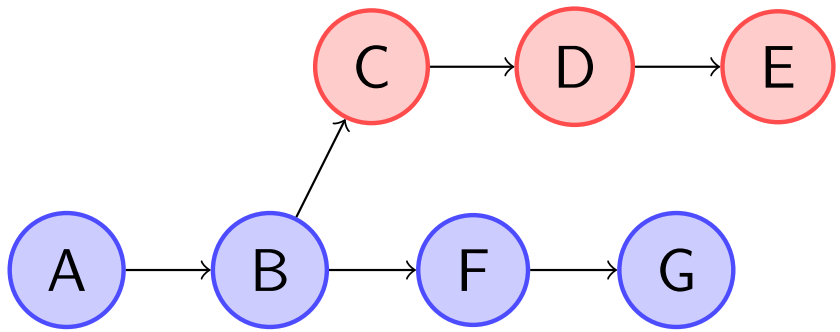
Permet d'éviter trop de merges pour une branche simple non partagée



- Commits existants
- Branche de développement

Rebase

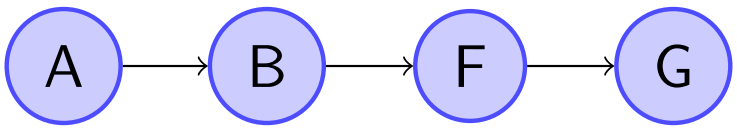
Permet d'éviter trop de merges pour une branche simple non partagée



- Commits existants
- Branche de développement
- Commits dans *master*

Rebase

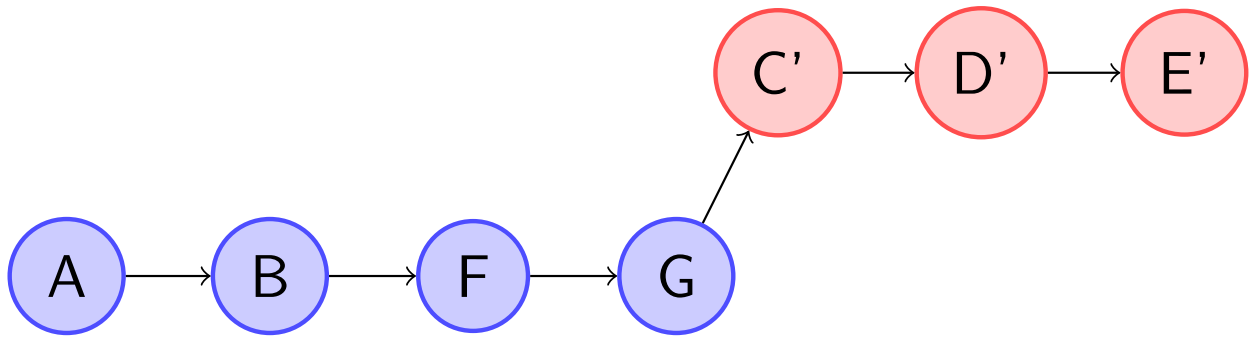
Permet d'éviter trop de merges pour une branche simple non partagée



- Commits existants
- Branche de développement
- Commits dans *master*
- Début rebase : enlève les commits de la branche

Rebase

Permet d'éviter trop de merges pour une branche simple non partagée



- Commits existants
- Branche de développement
- Commits dans *master*
- Début rebase : enlève les commits de la branche
- Fin rebase : recrée les commits à partir du *HEAD* de *master*

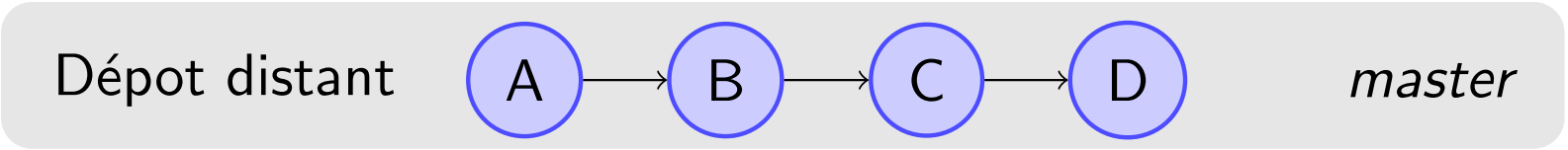
Agenda

Opérations Dépot distant

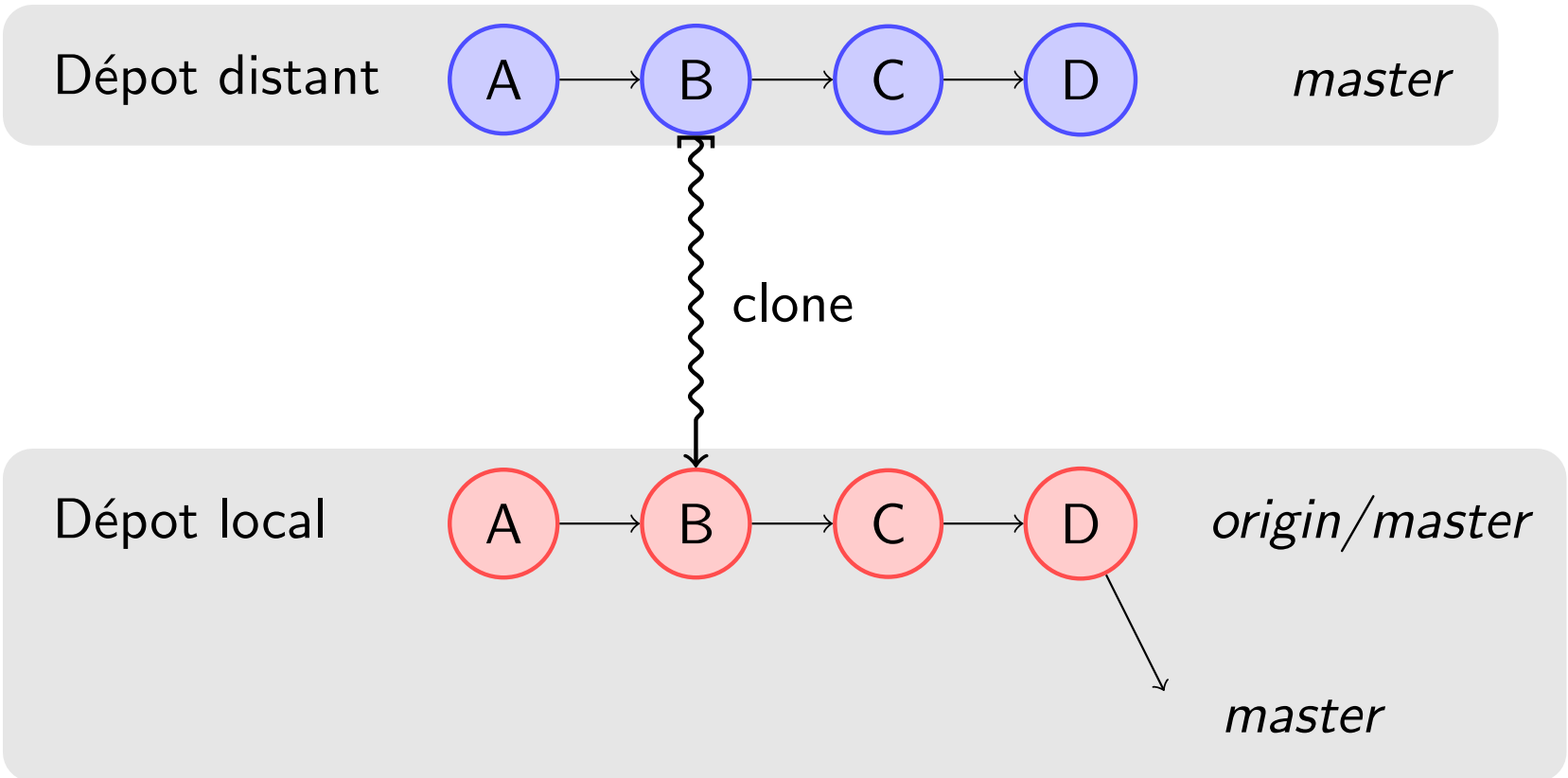
Principe: suivi des dépots distants dans des branches spécifiques.

clone	copie initiale d'un dépot distant
fetch	récupération des commits du dépot distant
pull	**fetch** + merge de la branche courante
push	transferts de commits vers un dépot distant

Dépot distant



Dépot distant



Dépot distant - suivi des branches

- les branches du dépot distant apparaissent avec le préfixe *origin/* par défaut (configurable)
- après un **clone** la branche *master* suit automatiquement *origin/master*
- la création d'une branche avec le même nom qu'une branche distante configure automatiquement le suivi
- lors d'une opération de **push** ou **pull** toutes les branches suivies sont fusionnées et synchronisées.

Dépot distant - pull

Enchaîne 2 opérations:

1. **fetch** récupère les commits du dépot distant et les applique aux branches *origin/* (Fast Forward).
2. **merge origin/branch** fusionne la branche distante suivie dans la branche locale.

Alternative: pull –rebase : après le **fetch**, **rebase** les commits de la branche locale

Dépot distant - push

- seuls les push qui se fusionnent en Fast Forward sont autorisés par défaut (sinon: perte de l'historique).
- si Fast Forward pas possible : faire d'abord un pull (–rebase) : $\rightarrow$ rend le Fast Forward possible.
- spécifier le nom d'une branche nouvellement créee localement pour la créer aussi dans le dépot distant.

Dépots distants - stratégies

Dépot central

- dépot nu (pas de copie de travail des fichiers)
- similaire à SVN ou CVS
- chacun pousse ses commits

Modèle distribué

- pas de push
- chacum ≪ pioche ≫ chez les autres les commits qui l'intéressent
- un ≪ intégrateur ≫ qui centralise dans son dépot ≪ La ≫ version officielle.

Agenda

Conclusion

- Ne pas utiliser un SGV est une faute professionnelle...
- Choix d'un système « moderne »: système distribué: git, mercurial.
- Commencer par une utilisation simple
- Committer souvent
- Importance des messages de commit
- Ne remplace pas le chef de projet

Questions ?