Gestion de Versions
Mise en oeuvre avec GIT

Laurent Pautet, laurent.pautet@telecom-paristech.fr
Thomas Robert, thomas.robert@telecom-paristech.fr

Définition de la Gestion de Versions

La gestion de la configuration est une discipline qui permet d'identifier les composants d'un système en évolution continuelle, d'en contrôler les évolutions durant le cycle de vie du logiciel, d'archiver chacun des états successifs, et de vérifier que chacun de ces états est complet et cohérent

AFNOR, Z61-102

Intérêts de la Gestion de Versions

- **... identifier les composants ...**
  - Ne pas versionner les objets (.class), les exécutables ...
- **... évolution continuelle ...**
  - Ne pas versionner les éléments constants (lib externes)
- **... contrôler les évolutions ...**
  - Revenir en arrière, commenter les améliorations
- **... archiver des états successifs ...**
  - Ne pas perdre les états intermédiaires
- **... vérifier ces états ...**
  - Assurer que les états de différents éléments sont cohérents

Scenario 1 : perte de pages d'un rapport

- Vous avez écrit un rapport de 40 pages. Suite à une mauvaise manipulation vous perdez 20 pages …
- **Solution 1** : vous fermez l'éditeur sans sauvegarder et rechargez le document …
- **Problème :** vous perdez le travail non sauvegardé …
- **Solution 2** : (pré-requis) l'éditeur sauvegarde à votre insu les 10 dernières actions
  Il crée un historique pour remonter dans le temps
- **Problème** : l'historique est limité en « profondeur » et perdu entre deux ouvertures de l'éditeur

Scénario 2 : mauvaise ré-écriture du texte

- **Vous souhaitez reprendre l'introduction faite initialement qui semble meilleure que l'actuelle.**
- **Solution 1 : créer des copies du document au fur et à mesure de son évolution avec des noms différents**

- **Problème :**
  - Retrouver les fichiers
  - Sauvegarder systématiquement
  - Limiter le volume sur disque
- **Solution 2 :**
  Archiver avec un système de gestion de versions

Définition d'une version

- Vos **données** sont identifiées par leur **contenant** et non par leur **contenu.**

  => il peut exister plusieurs contenus pour un même contenant au cours du temps : **les versions**

- **Exemple**
  - Votre **rapport** est stocké dans le **fichier** rapport.txt
  - Chaque jour, vous créez une **copie** dans un répertoire nommé par la date du jour de la sauvegarde
  - Chaque copie est une **version** de rapport.txt

Versions et Alternatives

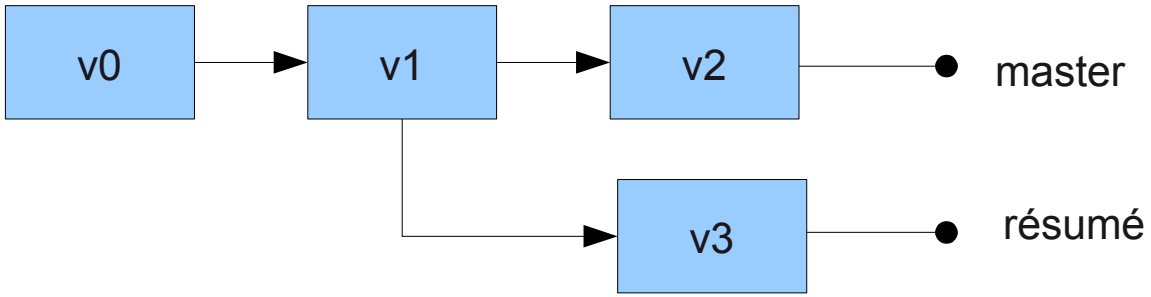
- **Deux versions d'un même contenant peuvent différer sur leur contenu pour diverses raisons**
  - V2 représente une amélioration de V1
  - V2 représente une **alternative** (concurrente) à V1 (par exemple, V0 → V1 et V0 → V2)
- **Exemple d'alternative :**
  - Vous avez écrit la V1, documentation technique détaillée de votre stage, que vous continuez à compléter.
  - La V2 inclut des sections destinées à la formation humaine mais pas toutes les sections (techniques) de V1. Vous modifiez la V2 suite aux relectures de votre resp. de stage

Historiques et Branches

- **Définition :**
  - Branche => séquence de versions d'un contenant
  - Une branche a une première et une dernière version
  - La dernière version différencie une branche d'une autre
- **Branches :**
  - V0 → V1 → V2 et V3 une alternative à V2
  - V1 → V3 est ajoutée
  - L'historique n'est plus une séquence mais un arbre.
  - V2 et V3 peuvent être améliorées indépendamment
  - Deux branches : V0 → V1 → V2 et V0 → V1 → V3

Scenario 3 : rapport et résumé

- Enregistrement Rapport V0 (intro) (master)
- Rapport V1 (intro) amélioré par chap 2-4
- Rapport V2 (intro+chap 2-4) amélioré par conclusion
- Rapport V3 (introduction+chap 2-4) alternative à V2 avec chap 2-4 résumés
- V2 se trouve sur la branche principale (master)
- V3 se trouve sur une branche particulière (résumé)



Opérations sur les améliorations

- **Problème : une amélioration s'avère erronée**
  - Conséquence d'une modification mal évaluée
  - Évolution des objectifs associés au document

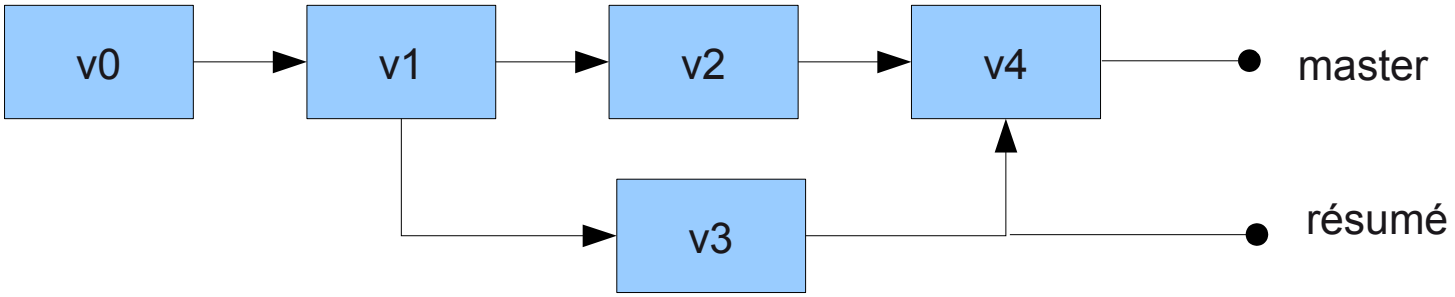
- **Solution : on rembobine**
  - Chaque version est soit la version initiale d'une branche, soit elle possède une version antérieure
  - On peut toujours remonter « le temps » jusqu'à V0

    Les versions ont des numéros uniques (id)

Opérations sur les branches

- **Problème : le meilleur de deux branches**
  - La branche principale évolue favorablement
  - La branche alternative évolue favorablement

- **Solution : fusionner les branches**
  - Rassembler les améliorations des deux branches pour produire une version unifiée



Conflit dans la fusion d'alternatives

- **Intersection entre améliorations de branches = vide**
  - Superposition des améliorations
- **Intersection entre améliorations de branches /= vide**
  - Intersection entre lignes modifiées d'un fichier texte = vide
    - Superposition des modifications de lignes pour le fichier
  - Intersection entre lignes modifiées d'un fichier texte /= vide ou conflits sur un fichier binaire
    - Résolution par l'utilisateur (conflits donnés dans un fichier intermédiaire)

V2

rien
mais vraiment rien !
c'est pour dire ...

V3

rien
mais vraiment rien !
encore que ...

Conflits entre les versions V2 et V3 d'un fichier

```
<<<<<<<
c'est pour dire ...
=======
encore que ...
>>>>>>>
```

Commenter une nouvelle version

- **Une nouvelle version se caractérise par**
  - Un ensemble de différences sur le contenu
  - Une justification de la production d'une nouvelle version
- **Le gestionnaire attache un commentaire à une version**
  - Celui-ci porte sur le *pourquoi* et non sur le *comment*
  - Le comment s'obtient par différence entre versions
    - Le comment doit être indiqué en commentaire dans le code
  - Le pourquoi doit être explicitement fourni par l'utilisateur
    - Quel problème pose l'ancienne version ?
    - Quelle amélioration apporte la nouvelle version ?

Stockage dans le gestionnaire

- **Fichiers et répertoires de travail présents en local**
- **Améliorations et commentaires stockés dans un dépôt**
  - Local (risque de perte du dépôt)
  - … ou distant (pas de travail hors ligne)
  - … ou hybride
    - 1 dépôt distant
    - 1 copie du dépôt pour le travail hors-ligne
- **Les fichiers binaires sont stockés intégralement**
- **Les fichiers texte peuvent l'être aussi**
- **Mais on peut aussi sauver la dernière version et les différences vers les versions antérieures (gain de place)**

RCS, CVS, SVN, GIT et eGIT

- **RCS (rudimentaire)**
  - Gestion de version de fichier, archivage local
- **CVS (puis SVN) (évolué)**
  - Gestion de version par fichier (par ensemble de fichiers)
  - Archivage distant centralisé
- **GIT (très évolué mais complexe)**
  - Gestion de versions par ensemble de fichiers
  - Archivage distant réparti
  - Spécialisé dans la gestion des branches

Configurer et initialiser un dépôt

- **Configurer git**
  - > git config –global user.name 'Laurent Pautet'
  - > git config –global user.email 'laurent.pautet@telecom-paristech.fr'
- **Initialiser un dépôt**
  - > git init p — crée un répertoire p dans lequel
    Initialized … *path*/p/.git/ — git crée un répertoire de dépôt p/.git
  - > ls p/.git — p/.git contient versions, commentaires, etc.

./	HEAD	ORIG_HEAD	hooks/	objects/
../	MERGE_HEAD	branches/	index	refs/
COMMIT_EDIT	MERGE_MODE	config	info/	
FETCH_HEAD	MERGE_MSG	description	logs/	

Ajouter, marquer et archiver un fichier

- **Ajouter (add) un fichier**
  - > edit README — crée le fichier README
  - > git add README — ajoute README à l'index du dépôt
- **Marquer (add) et archiver (commit) un fichier**
  - > edit README — modifie README
  - > git add README — marque README et sa version pour archivage
  - > git commit -m 'raisons' — archive dans le dépôt les versions marquées
- **add ajoute**
  - un fichier à l'index des fichiers du dépôt (la première fois) ou
  - une version dans la liste des versions pour archive ultérieure
- **Marquer et archiver un fichier**
  - > git commit README — archive explicitement un fichier et donc le marque automatiquement

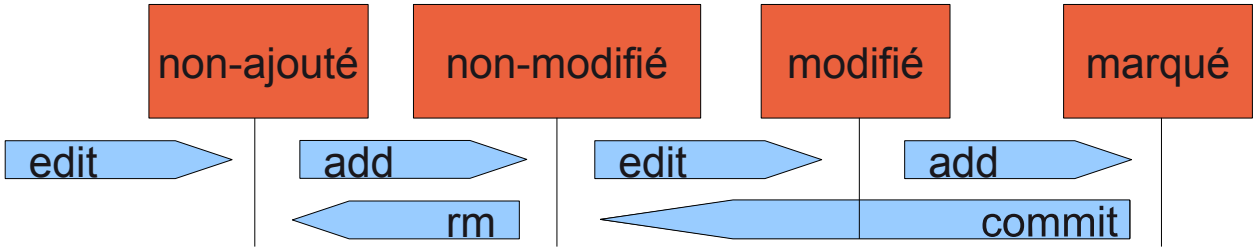
Marquer et archiver un fichier

- ***commit*** **archive ce qui est marqué au préalable**
- ***commit -a*** **marque et archive les fichiers déjà indexés**
- ***commit fichier*** **marque et archive le fichier (comme eGIT)**
  - > edit README — modifie README (V1)
  - > git add README — marque la version pour archivage
  - > edit README — modifie README (V2)

  git commit -m 'raisons' — archive V1 dans le dépôt

  git commit **-a** -m 'raisons' — archive V2 (**marque** fichiers indexés)

  git commit README -m 'raisons' archive V2 dans le dépôt



Retirer un fichier du dépôt

- **Retirer un fichier**
  - > git rm README — retire README de l'index des fichiers du dépôt ainsi que dans le répertoire courant

    ou
  - > git rm –cached README — retire README de l'index des fichiers du dépôt et le laisse dans le répertoire courant
  - > git commit -m 'raisons' — archive le retrait dans le dépôt

Connaître l'état et les différences

Connaître l'état d'un fichier

- > edit README — modifie README
- > git status — affiche l'état des fichiers

```
# Changes not staged for commit:
# * modified README …
```

- > git add README — marque README
- > git status — affiche l'état des fichiers

```
# Changes to be committed:
# * modified README …
```

Connaître les différences sur un fichier

- > git diff

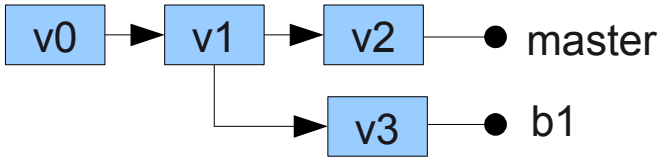
```
@@ -1,3 +1,3 @@
-rien
+en fait rien
```

Ajouter, modifier, lire les commentaires

- **Attacher un commentaire lors d'un archivage**
  - > git commit README — ouvre un éditeur, saisit commentaire
  - > git commit README -m 'raisons' — utilise l'arg de -m pour commentaire
- **Lire l'historique des commentaires**
  - > git log — liste tous les commentaires
  - > git log README — liste les commentaires de README
  - > git log -1 — liste ..... de la dernière archive
- **Modifier un commentaire après archivage**
  - > git commit README -m 'raisons' — donne un commentaire incorrect
  - > git commit –amend — edite le commentaire

Créer des branches

- **A la création, la branche principale est *master***
  - > edit f1 ; git commit -a f1 -m V1 — crée une version V1
  - > git branch -a — donne la liste des branches
    * master
- **Créer une nouvelle branche et une version par branche**
  - > git branch b1 — crée b1 en restant sur master
  - > edit f1 ; git commit -a f1 -m V2 — crée une version V2 dans master
  - > git checkout b1 — bascule sur la branche b1
  - > edit f1 ; git commit -a f1 -m V3 — crée une version V3 dans b1
  - > git branch -a — donne la liste des branches
    * b1 — b1 est la branche courante
      master



Fusionner des branches

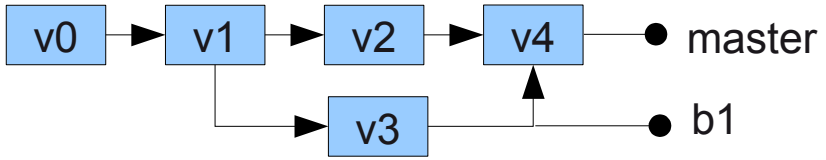
- **Fusionner la branche b1 dans la branche principale**
  - > git checkout master — bascule sur la branche master
  - > git merge b1 — ajoute les modifications de b1 dans master
  - > edit f1 — il peut y avoir des conflits

```
<<<<<<< HEAD
X
=======
Y
>>>>>>> b1
```

  - > git commit -a f1 -m V4 — crée une version V4 dans master
  - > git branch -a — donne la liste des branches

```
* master
  b1
```

master est la branche courante



Fusionner des branches

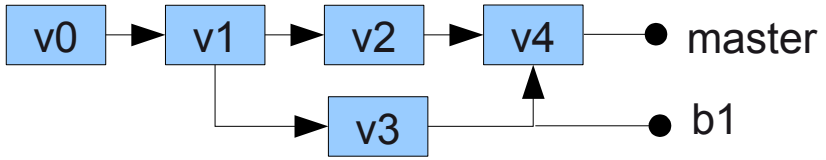
- **Fusionner la branche b1 dans la branche principale**
  - > git checkout master — bascule sur la branche master
  - > git merge b1 — ajoute les modifications de b1 dans master
  - > edit f1 — il peut y avoir des conflits

```
<<<<<<< HEAD
X
=======
Y
>>>>>>> b1
```

  - > git commit -a f1 -m V4 — crée une version V4 dans master
  - > git branch -a — donne la liste des branches

```
* master
  b1
```

master est la branche courante



Revenir en arrière

- **Identifier une version**
  - > git log liste les modifications faites

    …

    commit 928533b32dac3673c86f39aab8d2e1c9522e9e6f

    Author: Laurent Pautet <laurent.pautet@telecom-paristech.fr>

    Date: Mon Nov 19 09:13:07 2012 +0100
- **Revenir à une version pour faire une branche**
  - > git checkout -b b1 928... revient au commit 928... et crée une branche b1 à partir de là

Espace de travail – Espace de stockage

- **Jusqu'ici, espace de travail contient espace de stockage**
  - Si le répertoire p est effacé, le dépôt p/.git est perdu
- **Initialiser un dépôt indépendant (hors espace de travail)**
  - > git init p — initialise un espace de travail p dans lequel
    Initialized … *path*/p/.git/ — git crée un espace de stockage (dépôt) p/.git
  - > git clone –bare p p.git — crée un dépôt p.git à partir de p ou p/.git
    ou autrement
  - > git init –bare p.git — crée directement un espace de stockage p.git
- **On peut déplacer l'espace de stockage (p.git) vers :**
  - Un autre répertoire (en local)(par exemple avec cp)
  - Un autre serveur (à distance)(par exemple avec scp)

Accéder à un dépôt

- **Accéder à un dépôt « distant » (autre répertoire)**
  - > git clone file:///path/p.git myp
- **Accéder à un dépôt distant (par ssh)**
  - > git clone ssh://user@server:*path*/p.git myp
- ***clone* duplique le dépôt distant dans l'espace de travail**
  - > git clone ssh://user@server:*path*/p.git myp
  - > git remote -v — liste les dépôts présents
    origin ssh:///user@server/path/p.git — origin est dépôt par défaut
  - Création du répertoire myp/.git comme précédemment
    - dépôt local
    - copie de p.git pour travail hors ligne

Référencer un dépôt distant

- **Référencer en local un dépôt distant**
  - > git remote add q ssh://user@server/path/q.git ajoute un autre dépôt
  - > git remote -v liste les dépôts connus

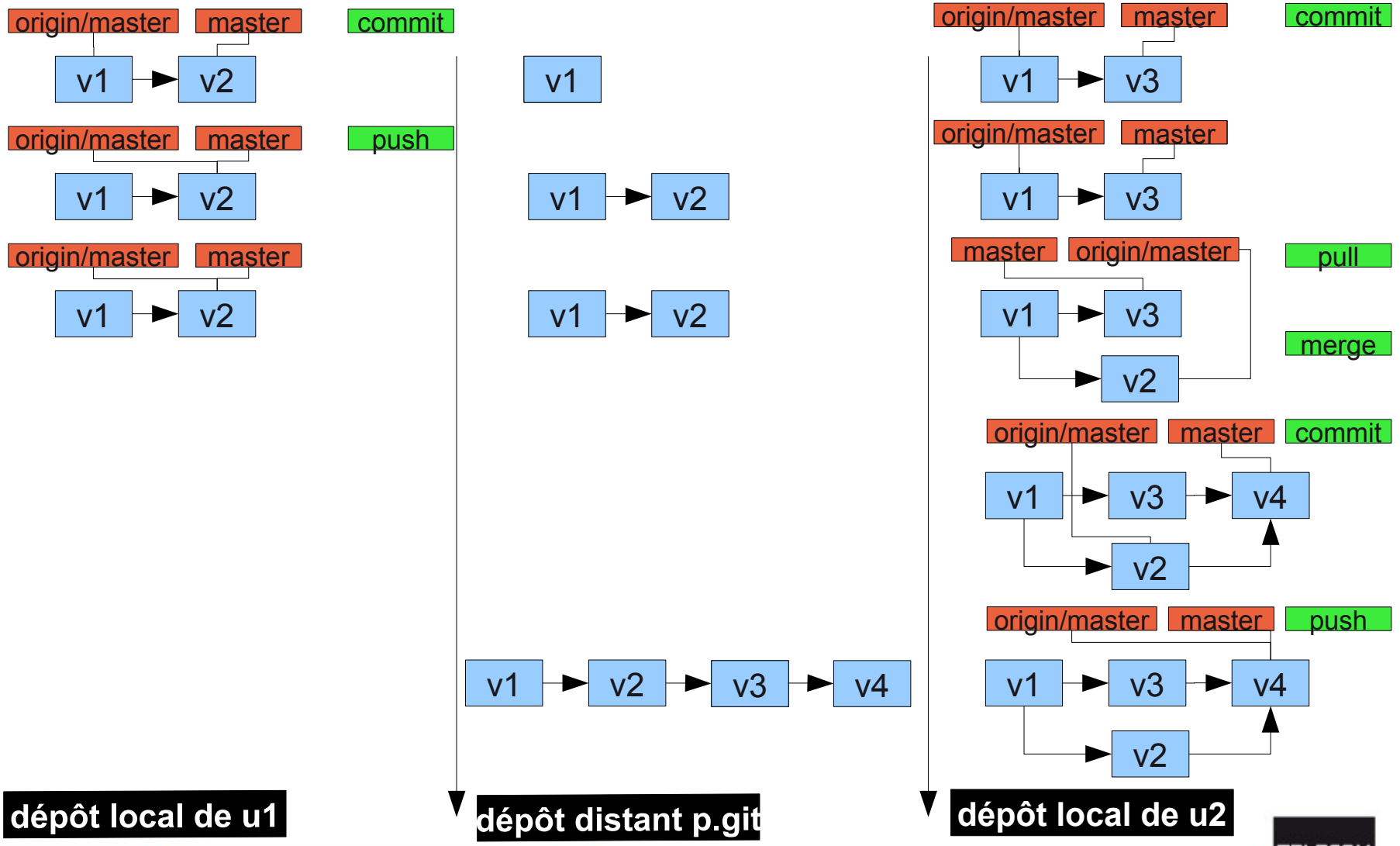
    origin ssh://user@server/path/p.git copié par clone

    q ssh://user@server/path/q.git pas copié par remote
- **Copier un dépôt référencé**
  - > git fetch q copie le dépôt q pour travail hors ligne

Synchroniser dépôt local et dépôt distant

- **Le dépôt local peut être en avance sur le dépôt distant**
  - Vos versions sont archivées dans le dépôt local myp/.git
- **Le dépôt distant peut être en avance sur le dépôt local**
  - D'autres utilisateurs modifient le dépôt distant
- **Il faut synchroniser le dépôt distant avec celui local**
  - > git push origin master reporte les modifications de la branche master du dépôt local vers le dépôt distant origin
  - > git branch -a une branche spéciale apparaît pour indiquer à quel endroit se trouve la synchronisation
    * master
    remotes/origin/master
- **Il faut synchroniser le dépôt local avec celui distant**
  - > git pull origin reporte les modifications du dépôt distant origin et les fusionne à la branche courante ce qui équivaut à
  - > git fetch origin puis git merge origin/master

Exercice simple de résolution de conflits entre dépôts locaux et dépôt « distant »



Exercice simple de résolution de conflits entre dépôts locaux et dépôt « distant »

`git init --bare p.git`	On crée un dépôt distant p.git sans espace de travail (--bare)
`git clone file://$PWD/p.git u1`	Pour un utilisateur u1, on crée un espace de travail u1 et un dépôt local u1/.git à partir du dépôt distant p.git. Pour l'interaction à distance, nous n'utilisons pas ssh mais des accès locaux au fichier p.git (file://..)

`cd u1`
`edit f`
`git add f`
	On crée un nouveau fichier f dans u1
	On l'ajoute à l'index des fichiers que gère git (il est ajouté puis marqué).

`git commit -m 'first commit'`	On archive dans u1/.git les fichiers ajoutés

`git push`
`cd ..`
`git clone file://$PWD/p.git u2`
`cd u2`
`edit f`
	On pousse les modifications faites dans u1/.git dans le dépôt distant p.git. p.git contient f.
	On crée un espace de travail et un dépôt local pour un utilisateur u2.
	Le fichier f est présent. On le modifie

Exercice simple de résolution de conflits entre dépôts locaux et dépôt « distant »

```
git commit -a -m 'commit u2'

cd ../u1
edit f
git commit -a -m 'commit u1'

git push
cd ../u2
git push
<error>
git pull
<conflict>

edit f


git -a -m 'merge u2'
```

On marque et archive dans le dépôt u2/.git les fichiers déjà indexés (-a). Les fichiers déjà ajoutés (f) sont automatiquement marqués si modifiés.

On effectue la même opération dans u1 et on archive dans u1/.git

On pousse les modifications de u1/.git vers le dépôt distant p.git.

On essaye de pousser les modifications de u2/.git dans p.git, mais p.git a été modifié précédemment au travers des modifications de u1/.git

On attire les modifications de p.git dans u2/.git. Il peut y avoir conflits, u1 et u2 ayant modifié les mêmes lignes

Le fichier contient les modifications de u1 et de u2. L'utilisateur u2 doit les résoudre à la main.

Et archiver dans u2/.git le fichier f dont les conflits sont résolus.

Exercice simple de résolution de conflits entre dépôts locaux et dépôt « distant »

```
git push

git log
   merge u2
   …
   commit u1
   …
   commit u2
   …
   first commit
```

On pousse les modifications de u2/.git dans le dépôt partagé distant p.git

On consulte les commentaires faits sur les différentes versions de f

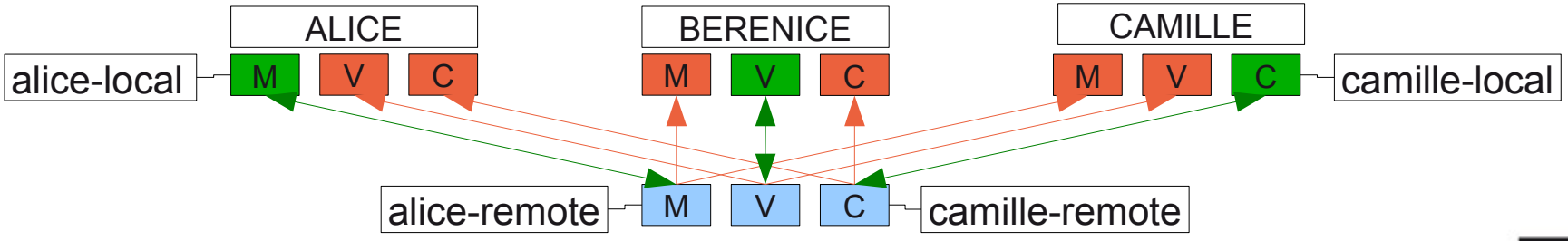
On constate que les modifications de la branche de u1 ont été intégrées à celles de u2 et que la fusion a été réalisée.

Naturellement, pour un exemple complet, il faudrait que les utilisateurs u1 et u2 se trouvent sur des répertoires différents et des machines différentes, et que le dépôt partagé soit effectivement distant sur une troisième machine. Chacun communiquant en utilisant ssh.

Cet exercice vise à simplifier le problème en utilisant le protocole file:// ou les accès à des fichiers locaux.

Un dépôt en écriture, plusieurs en lecture

- **Problème : conflits d'écrivains sur un seul dépôt**
  - écritures concurrentes sur le dépôt
  - opérations chronophages de fusion
- **Solution : découper le système en sous-systèmes**
  - un dépôt par sous-système
  - un écrivain par dépôt
  - plusieurs lecteurs par dépôt
  - pas de conflits
  - répartition des rôles



Conclusions

- Indispensable pratique d'un gestionnaire de versions
- Utilisation de gestionnaire pour un projet, un rapport, …
- Archiver l'évolution d'un travail
- Favoriser le travail en groupe
- GIT est très puissant, très flexible, très complexe
- Il rend simples des opérations sophistiquées mais …
- Il rend aussi simples des opérations erronées