

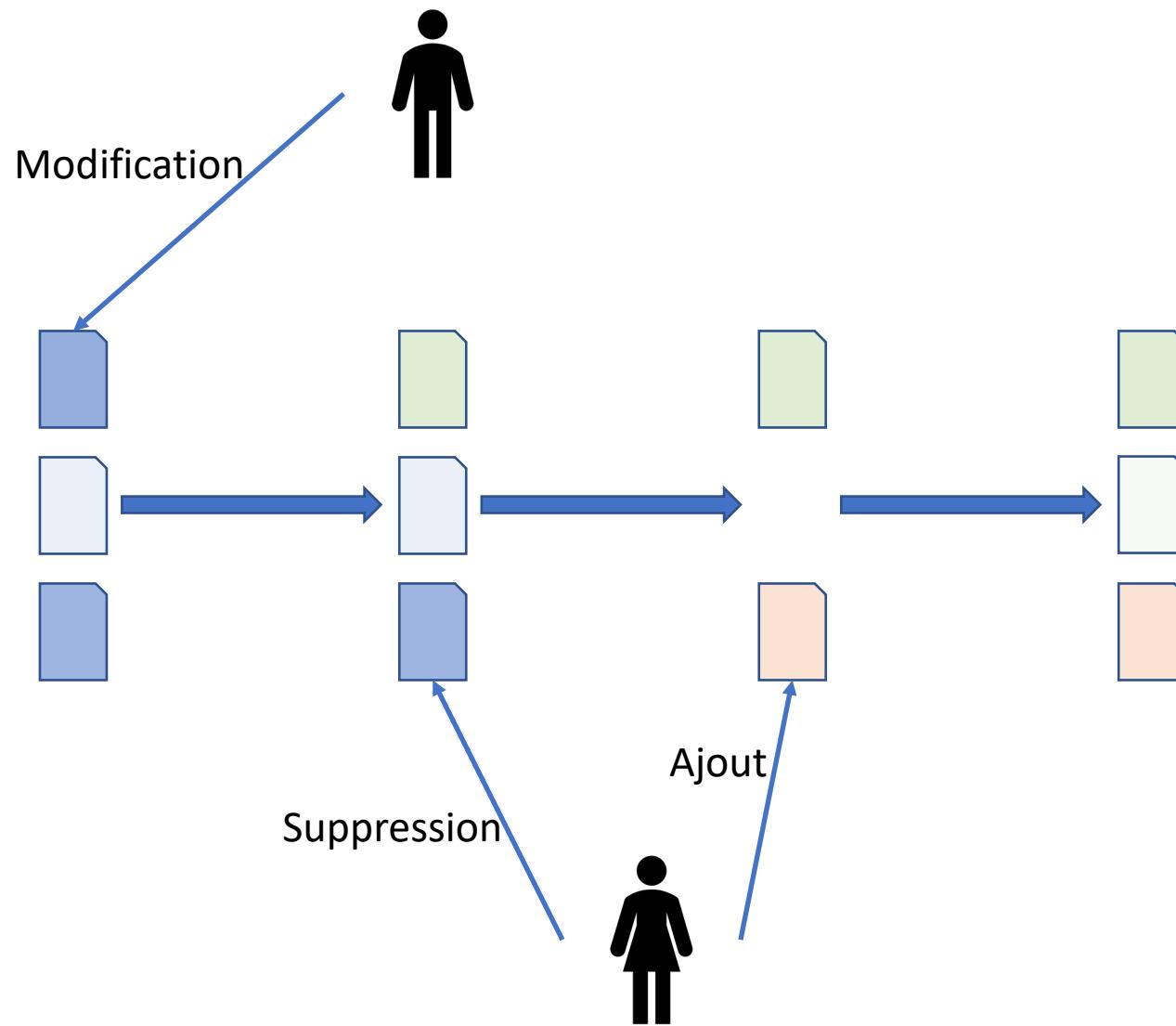
Gestion de source

principes

Introduction

- Suivi des modifications d'un ensemble de fichiers
 - Code source
 - Documentation
 - Fichiers de configuration...
- Objectif
 - Garder un historique des modifications
 - Permettre un développement collaboratif
 - Casser la linéarité du développement

Exemple idéalisé



Architectures

- Utilisation d'une *Repository* (*dépôt*) pour stocker
 - Fichiers
 - Modifications
- Le programmeur travaille sur une copie locale
 - Divergence avec le *repository*
 - Doit régulièrement être réconcilié
- Localisation et contenu du *repository* définissent l'architecture

Architecture centralisée

- Copie locale et *repository* sur une unique machine
- Pros :
 - Simple à déployer et utiliser
 - Permet la gestion de l'historique
- Cons :
 - Pas de redondance
 - Pas de collaboration possible

Architecture distribuée : *repository* distant

- *Repository* sur une machine distante (aka serveur)
- Le serveur est un oracle
 - Il dicte quelles sont les versions officielles du logiciel
- Pros :
 - Pas trop compliqué à mettre en place
 - Collaboration à plusieurs possible
 - Gestion des *release* simplifié
- Cons :
 - Pas de redondance
 - Difficile de travailler sur de gros changements
 - Les modifications sont forcément envoyées sur le serveur, donc visibles

Architecture distribuée : *repository* distribué

- Chaque développeur a une copie complète du *repository*
 - Le développement est local
 - Les modifications sont *poussées* aux autres
- Pro :
 - Redondance
 - Meilleure répartition possible du travail
 - Maintient d'un historique sans forcer les autres à se mettre à jour
- Cons :
 - Pas de version officielle naturelle
 - Synchronisation compliquée entre tous
- Souvent approche hybride distant-distribué
 - Un *repository* est spécial
 - Utilisé par GitHub ...

GIT

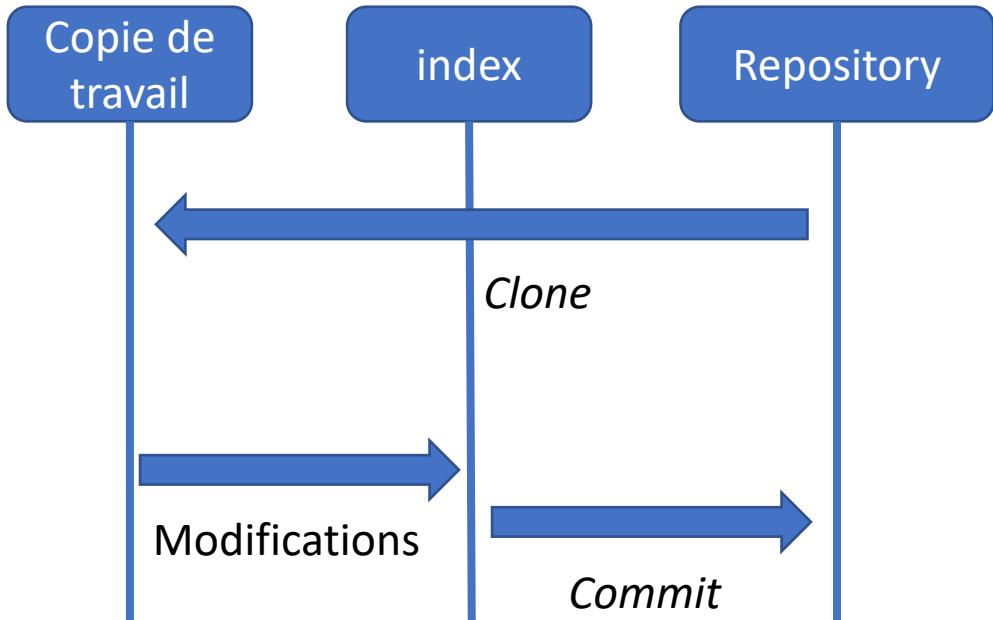
Introduction

Git

- Git est un gestionnaire de source local et distribué
 - Repository local
 - Repository distant possible
 - Utilisation d'un index local
- Crée en 2005 par Linus Torvalds
 - Dev commenté le 3 avril 2005
 - Git utilisé pour développer Git à partir du 7 avril 2005
- Caractéristiques
 - Adapté aux gros projets avec beaucoup de développeurs
 - Facilite le développement non linéaire (branches)

Structure d'un projet GIT

- Repository
 - Dans le répertoire .git
- Copie de travail
 - Fichiers gérés ou non par GIT
- Index
 - Fichier contenant les opérations à faire lors du prochain commit
 - Purement local, peut être modifié facilement



Principes

- Git gère des contenus, pas des fichiers
 - Les répertoires ne sont qu'une information
 - Les répertoires vides sont ignorés
- Chaque version d'un fichier est entièrement sauvegardée
- Le SHA1 d'un fichier est utilisé comme indexe unique
 - 160 bits (40 hex) uniques pour un contenu donné
 - Fichiers identiques stockés une unique fois par GIT, même si chemin différent

```
c672d78380e5d0bb29e5e9faeab1c77e8ca6dd38  test/fichierTest.txt
```

Classification des fichiers

- Git a 3 groupes pour gérer les fichiers
- *Untracked*
 - Non géré par GIT (état par défaut)
- *Tracked*
 - Gérés par GIT (ajout explicite)
- *Ignored*
 - Fichiers qui ne doivent PAS être gérés par GIT
 - Liste par défaut + ajouts possible (`.gitignore`)
 - Typiquement : fichiers générés, infos sensibles...

GIT

Solo

Exemple – git solo

- Création d'un projet git dans un répertoire existant

```
GIT $mkdir test
GIT $cd test
test $git init
Initialized empty Git repository in /Users/fhuet/Documents/Unice/0utilsGenieLogiciel/GIT/test/.git/
```

```
test $git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

- 2 commandes importantes
 - Git init
 - Git status

Commandes

- La commande de base est git
 - On ajoute l'action à effectuer
- git config
 - Lors de la première utilisation de git

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

- Git init
 - Crée un repository vide
 - Le répository et l'indexe seront dans le répertoire *.git*

Commandes

- `git status`
 - Affiche les différences entre
 - l'index et le dernier commit (HEAD)
 - La copie de travail et l'index
 - Les fichiers qui ne sont pas *tracked*

```
test $git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Commandes

- git add
 - Met à jour l'index avec la version de travail
 - Indiquer un chemin ou un *fileGlob* (*, toto/*.c ...)

```
test $git add test.txt
test $
test $git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   test.txt
```

Commandes

- **git commit**
 - Stock le contenu de l'index dans le repository sous forme de nouveau commit
 - Doit avoir un message (*commit log*)
 - Opération atomique

```
test $git commit -m "premier commit"
[master (root-commit) 4dc3739] premier commit
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

- Chaque commit a un SHA1
 - Le dernier commit est HEAD
- **git log**
 - Affiche la liste des commits et les messages associés

```
test $git log
commit 4dc3739e7fdbf193d202037957ab3bc8ab571d97 (HEAD -> master)
Author: Fabrice Huet <fabrice.huet@gmail.com>
Date:   Mon Sep 17 16:02:23 2018 +0200

  premier commit
```

Suppression de fichier

- Commande *git rm <fichier>*
- Supprimer un fichier ne supprime pas son historique
 - git supprime le fichier localement avant le commit

```
test $  
test $rm test.txt  
test $git status  
On branch master  
Changes not staged for commit:  
  (use "git add/rm <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
        deleted:    test.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")  
test $
```

Suppression de fichier

```
test $git rm test.txt
rm 'test.txt'
test $git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    test.txt
```

Suppression de fichier

- Sans rm explicite

```
test $  
test $git rm test2.txt  
rm 'test2.txt'  
test $git status  
On branch master  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
          deleted:    test2.txt  
  
test $ls
```

Annuler un rm

- Avant commit, c'est « facile » Il faut
 - annuler la modification dans l'index (pas encore dans le repo)
 - Récupérer le fichier supprimé (depuis le repo)
- Annuler modifications
 - git reset HEAD
 - Remet l'indexe dans l'état correspondant au dernier commit (HEAD)
- Récupérer le fichier supprimé
 - git checkout -- <fichier>

```
test $git rm test.txt
rm 'test.txt'
test $git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    test.txt

test $git reset HEAD
Unstaged changes after reset:
D      test.txt
test $git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    test.txt

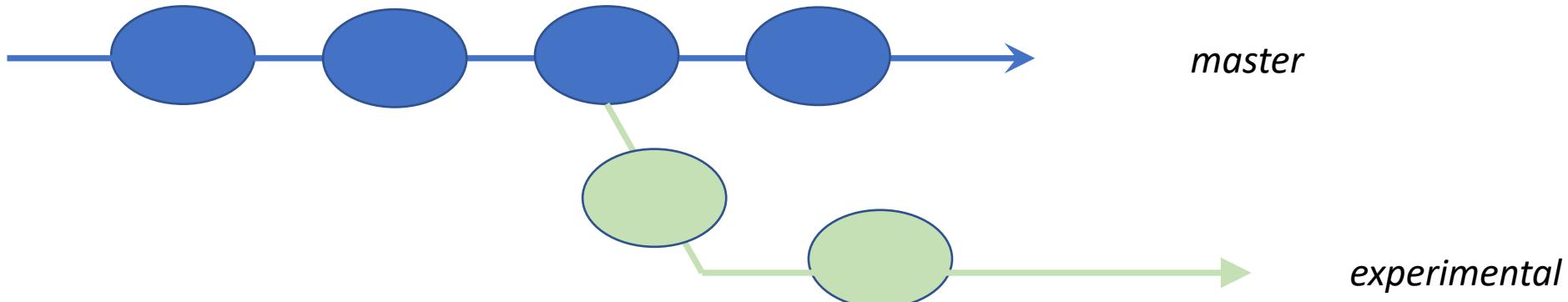
no changes added to commit (use "git add" and/or "git commit -a")
test $git checkout -- test.txt
test $ls
test.txt
test $
```

GIT

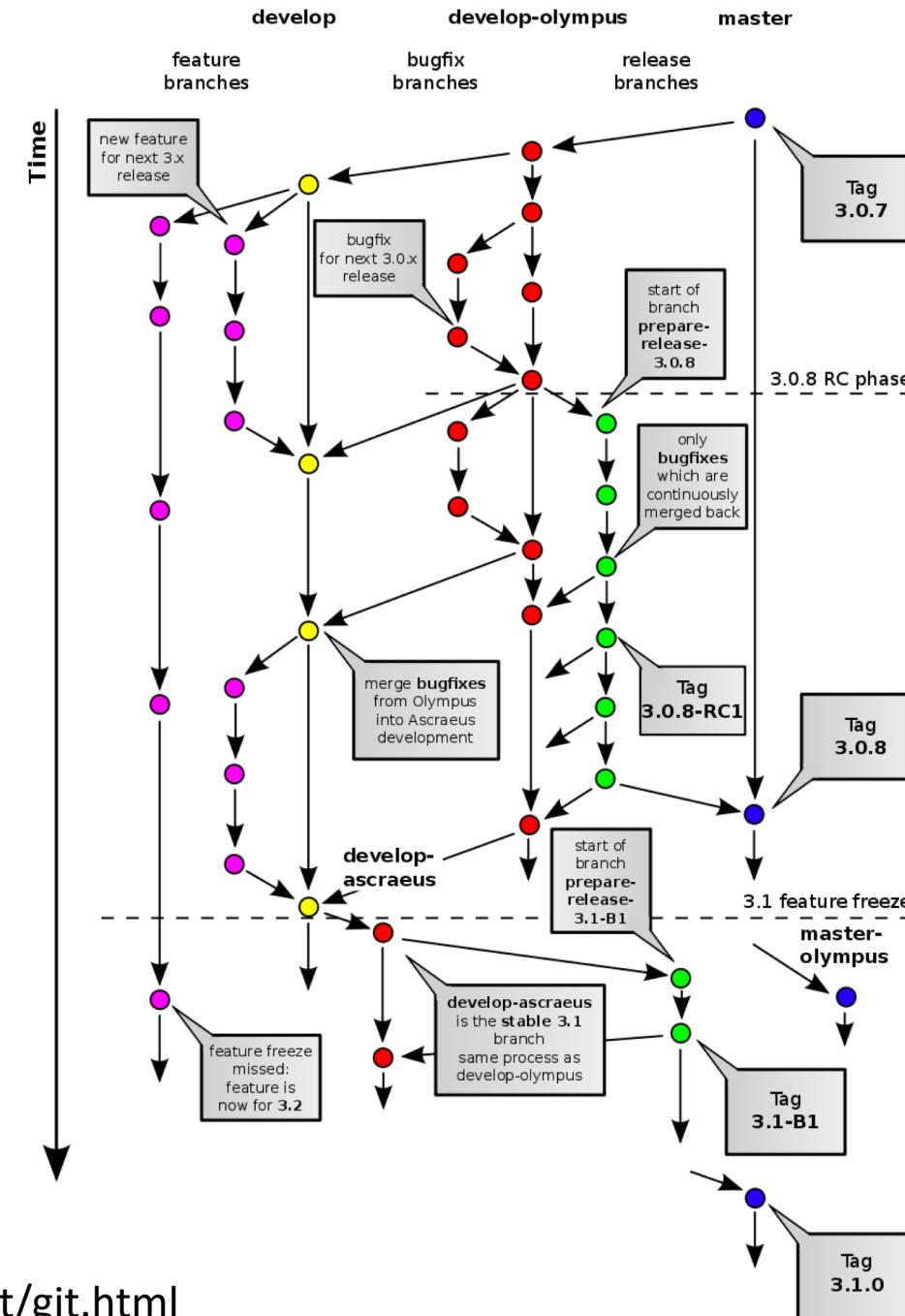
Branches

Branches

- Branche == ligne de développement séparé
 - Plus une unique version du projet
 - Très utilisé dans Git
- Branche par défaut : *master*
- La copie de travail ne peut correspondre qu'à une branche



Exemple : phpBB



Création de branches

- Creation par *git branch <name> [commit]*
 - Crée depuis un commit spécifié ou HEAD

```
test $git branch maPremiereBranche  
test $
```

- Permet aussi de lister les branches
 - Branche active indiquée par *

```
test $git branch  
maPremiereBranche  
* master
```

Changement de branches

- Commande *git checkout <branche>*

```
test $git checkout maPremiereBranche  
Switched to branch 'maPremiereBranche'
```

```
test $git branch  
* maPremiereBranche  
  master
```

- Exemple

- ajout fichier dans branche

```
test $git add testBranche.txt  
test $git commit -m "Fichier dans branche"  
[maPremiereBranche 94b8f3a] Fichier dans branche  
 1 file changed, 2 insertions(+)  
  create mode 100644 testBranche.txt
```

```
test $ls  
test.txt          testBranche.txt
```

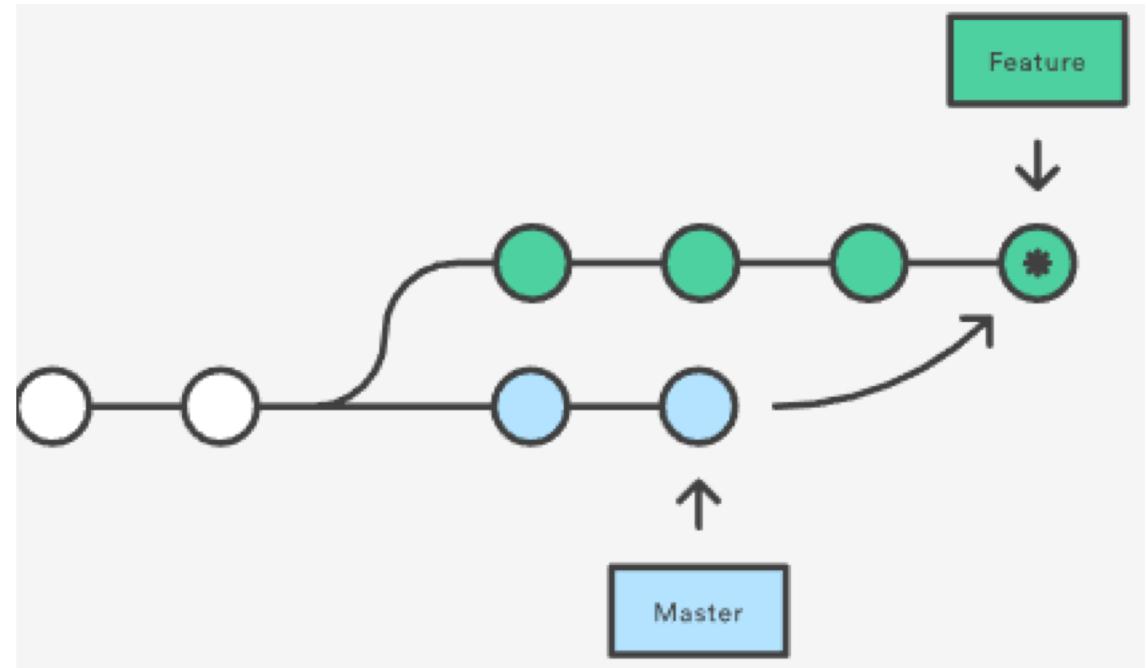
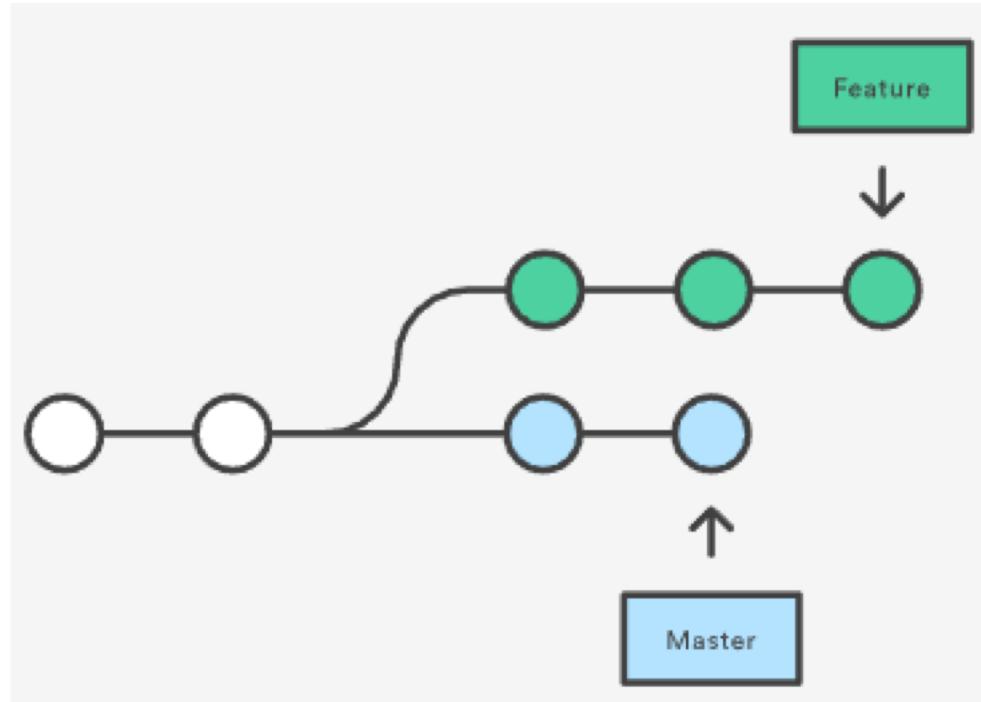
- Retour dans master

```
test $git checkout master  
Switched to branch 'master'  
test $ls  
test.txt
```

Fusion de branches

- Incorporer les changements d'une branche dans la branche courante
 - *Merge ou Rebase*
- Pas les même conséquences
- La fusion ne fait pas disparaître une branche
 - On peut continuer un développement divergent

Merge



Merge

- Crée un nouveau commit : *commit merge*
- Git calcule la fusion des fichiers et crée un nouveau fichier résultant
 - Facile si aucun conflit
- Merge une autre branche **dans** la branche courante

```
test $git merge master
Merge made by the 'recursive' strategy.
nouveauFichier.txt | 2 ++
1 file changed, 2 insertions(+)
create mode 100644 nouveauFichier.txt
test $
test $
test $git log --graph --pretty=oneline --abbrev-commit
*   2c2bf69 (HEAD -> maPremiereBranche) Merge branch 'master' into maPremiereBranche
|\ 
| * c97198b (master) nouveauFichier dans Master
* | 94b8f3a Fichier dans branche
|/
* 27c0223 delete test3.txt
* 4c423a4 added test.txt again
* 6cb5a66 add test3.txt
* dcb2574 modified test.txt
* 320547c suppression de test.txt
* 4dc3739 premier commit
```

Merge

- Conflit
 - Si au moins un fichier a été modifié dans chacune des branches

```
test $git merge master
Auto-merging futurConflit.txt
CONFLICT (content): Merge conflict in futurConflit.txt
Automatic merge failed; fix conflicts and then commit the result.
test $git status
On branch maPremiereBranche
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
:
Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  futurConflit.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Résolution de conflit

- La zone de conflit est indiquée par <<<< HEAD et >>> branche
 - Deux versions séparées par =====
 - Version actuelle en premier, version de branche après
- Il faut
 - Editer le fichier pour résoudre le conflit
 - Ajouter le fichier à l'index et commiter

```
<<<<<< HEAD
4
5
6
=====
7
8
9
>>>>> master
```

Merge : cas particulier

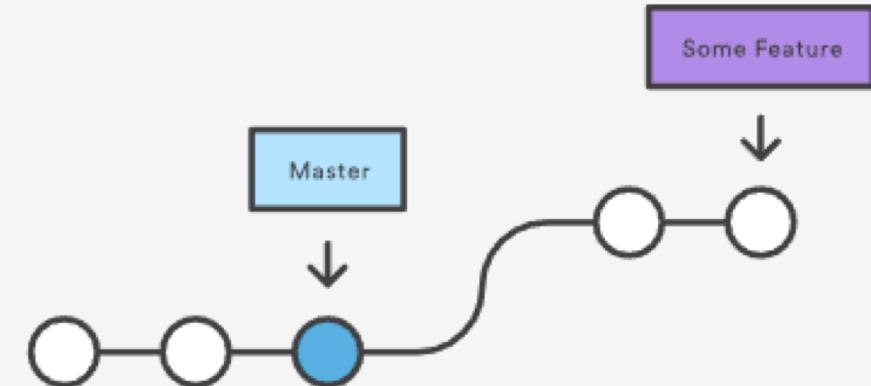
- *Already up-to-date*

- Tous les commits de l'autre branche sont déjà présents (ex: 2 merge de suite)
- Ou le commit de l'autre branche est un ancêtre du commit courant
- Pas de création de *commit-merge*

```
test $git merge master  
Already up-to-date.
```

- *Fast forward*

- Le commit de l'autre branche est un descendant du commit courant
- On déplace juste le pointeur



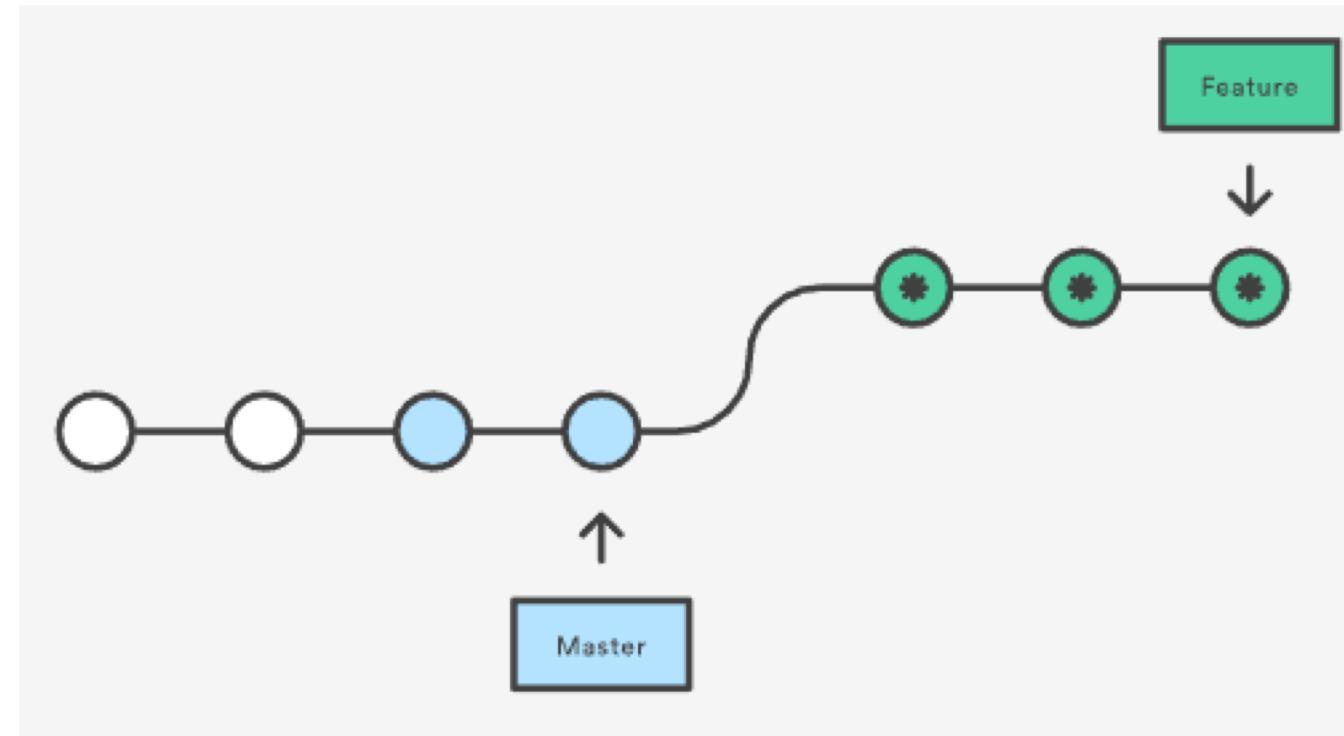
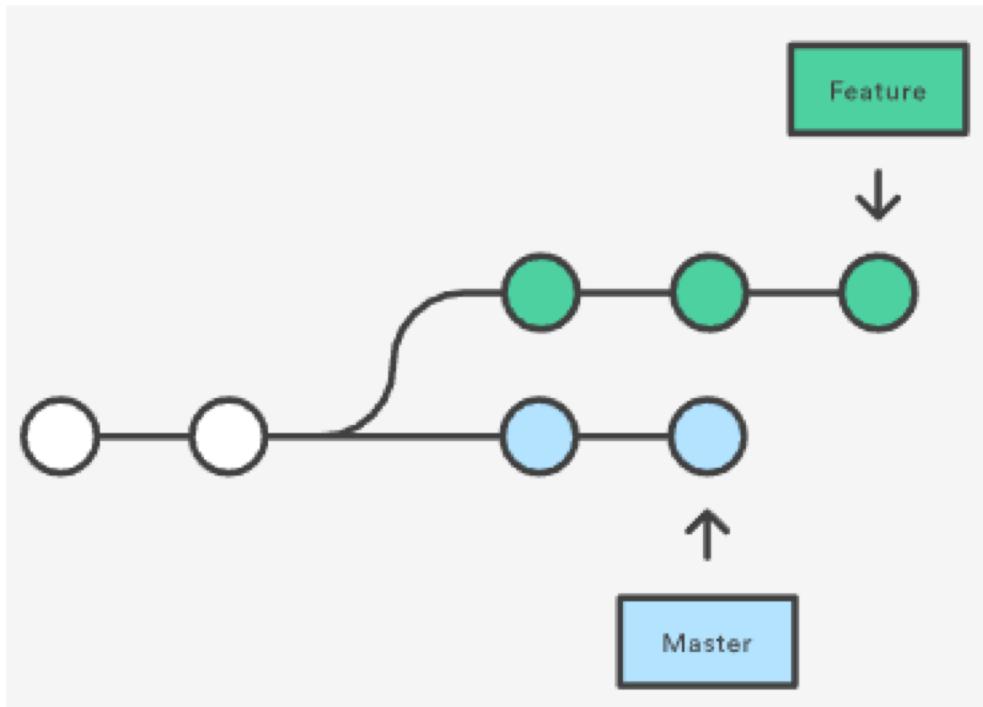
- Fast forward
 - On est dans « Master »
 - *git merge « Some Feature »*

```
| test $git merge maPremiereBranche
| Updating 8bf67c7..fe2091b
| Fast-forward
|   futurConflit.txt | 3 +++
|   testBranche.txt  | 2 ++
|   truc.txt         | 2 ++
| 3 files changed, 7 insertions(+)
| create mode 100644 testBranche.txt
| create mode 100644 truc.txt
```

After a Fast-Forward Merge



Rebase



```
git checkout feature  
git rebase master
```

Rebase

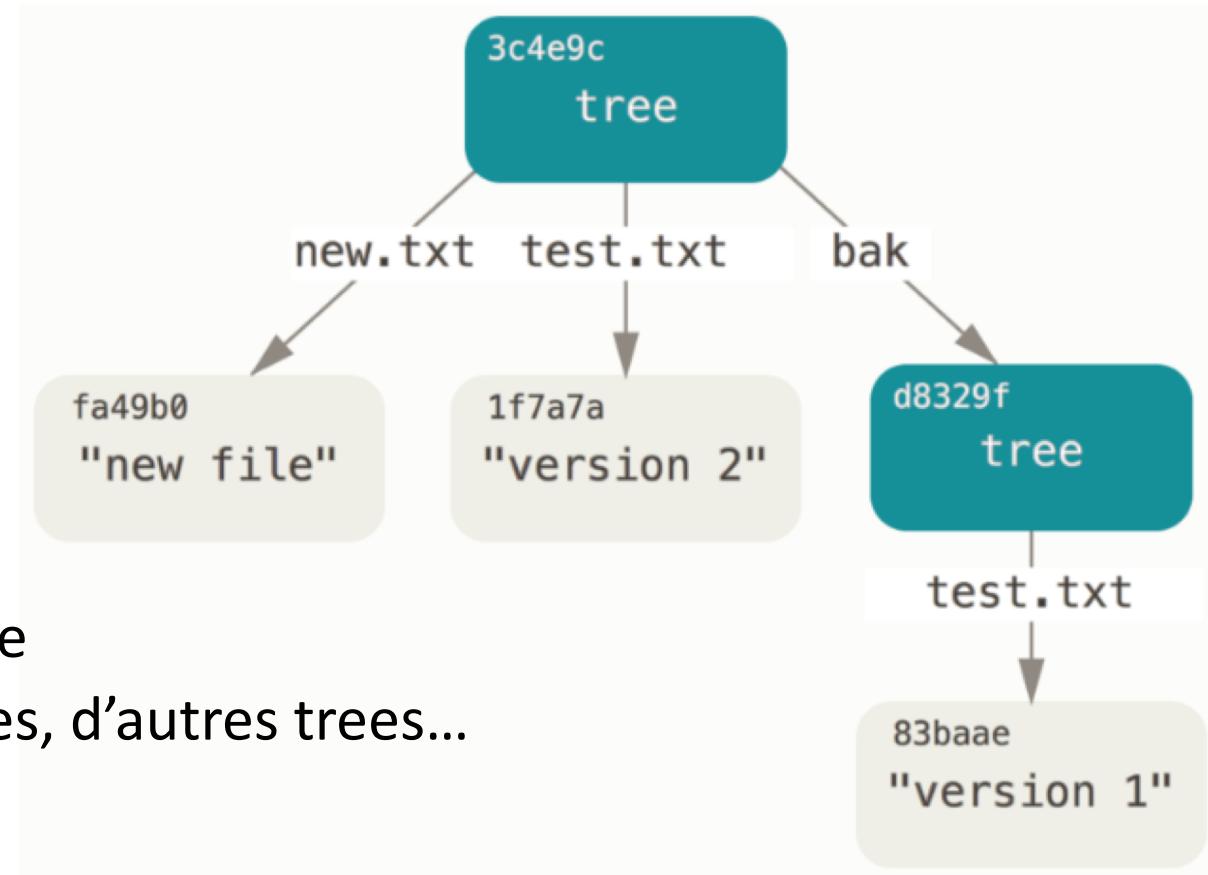
- Change le commit parent d'une série de commit
 - Déplace une branche de l'arbre
- Change l'histoire !
- Effets complexe si plusieurs branches

GIT

interne

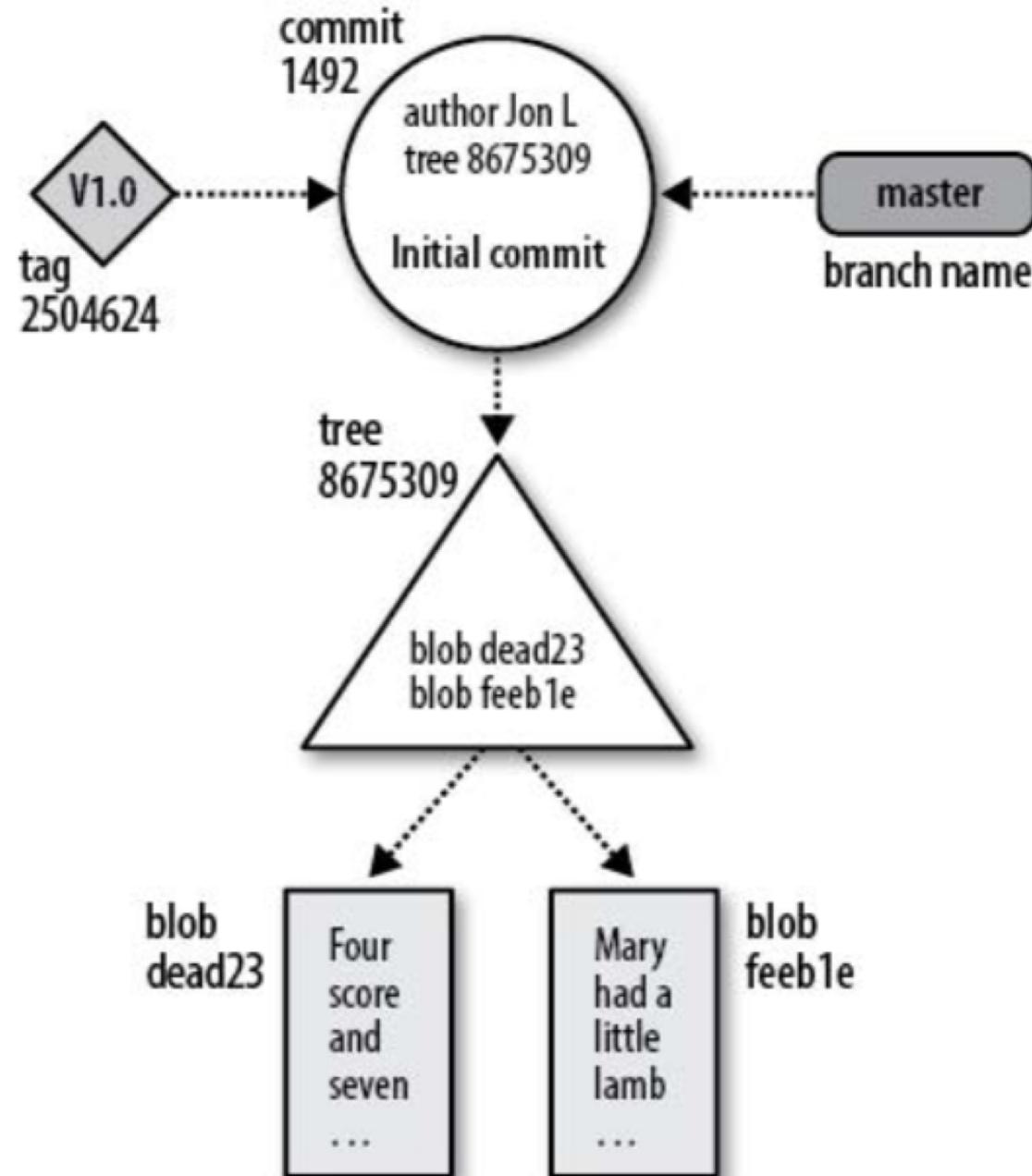
Objets

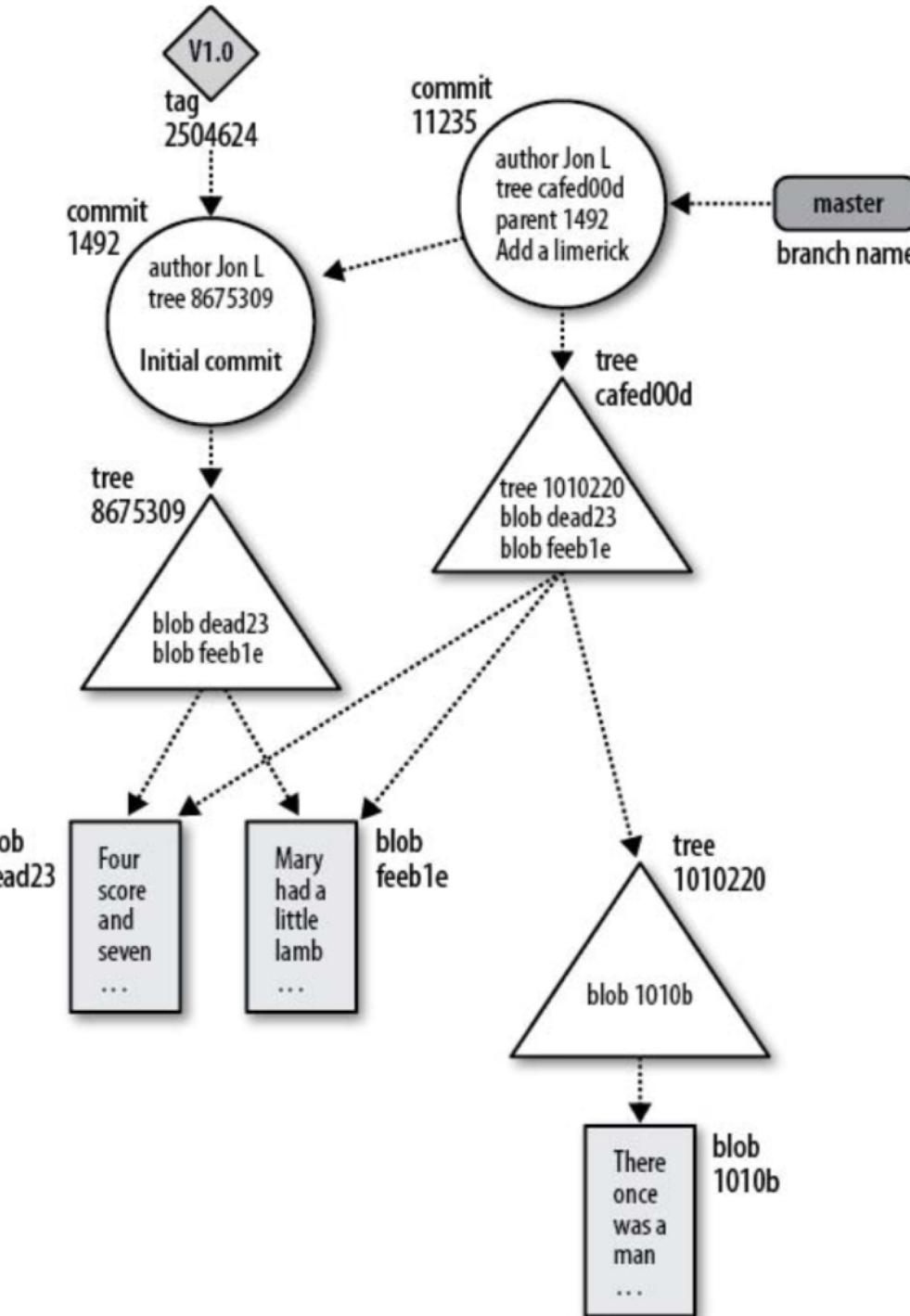
- Blobs (*Binary Large OBjectS*)
 - Contenu d'un fichier
- Trees
 - Équivalent à un système de répertoire
 - Contient des blobs, des méta-données, d'autres trees...
 - Possède lui même un SHA1



Objets

- Commits
 - Contient les meta-données pour chaque changement (auteur, date...)
 - Pointe vers un objet *tree* qui capture l'état du repository
 - Chaque commit a 0/1/plusieurs commit parents
- Tags
 - Nom symbolique permettant de retrouver un commit

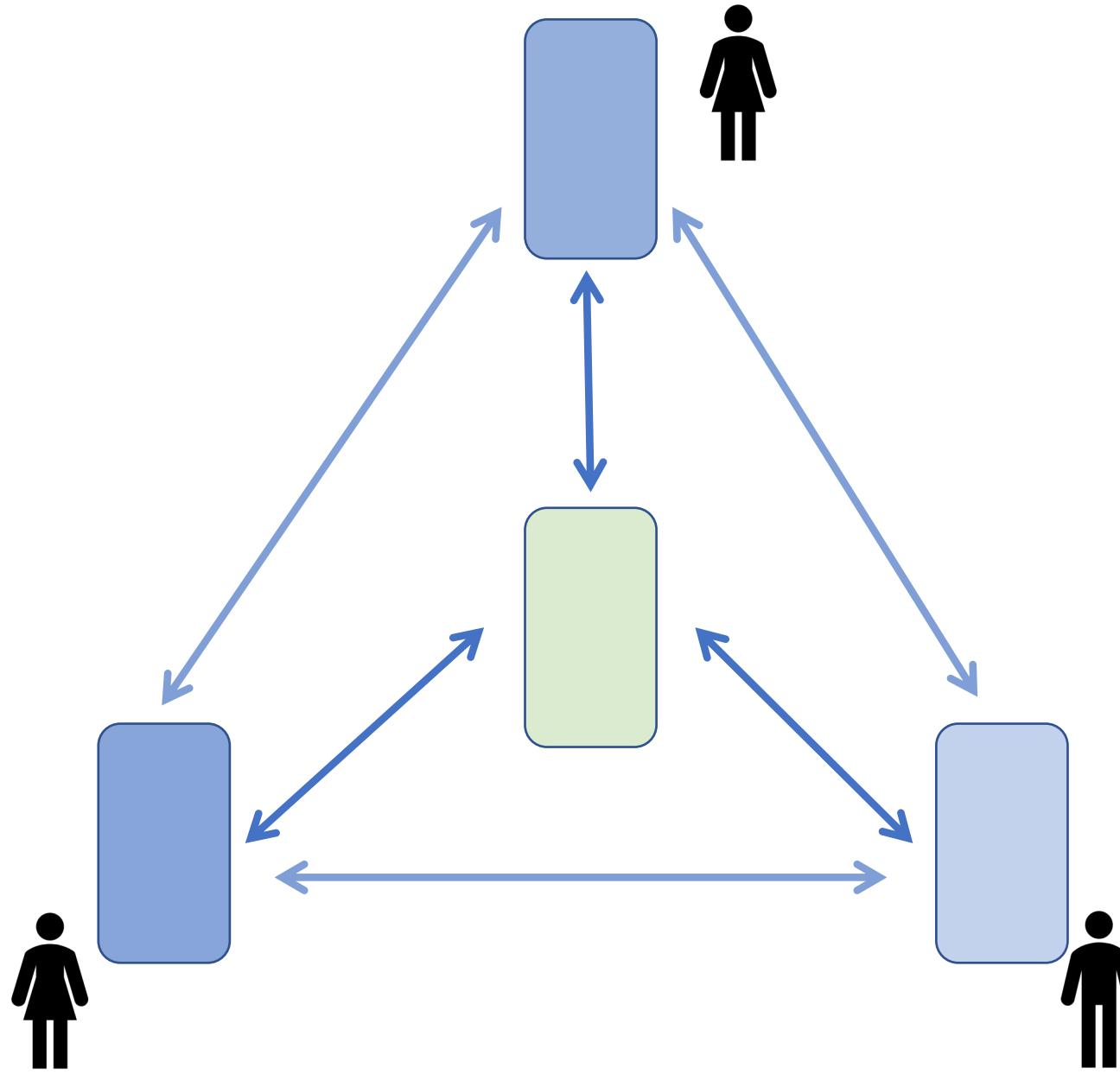




GIT

À plusieurs

Principe



Repository distants

- Development repository
 - Version standard d'un répo
 - Notion de branche active
- Bare repository
 - Version « light » d'un répo
 - Non utilisé pour le développement
 - Sert de point central pour dev collaboratif
 - *git init --bare ...*
- Clone repository
 - Copie d'un repo existant
 - Maintient un lien interne vers l'original : *origin*
 - Vocabulaire : *remote* ou *upstream* repository

Repository distants

- Addressage distant
 - Utilisation d'urls (*protocol://adresse...*)
- Exemples
 - *file:///home/bob/bareRepo.git*
 - <git@github.com:fabricehuet/programmationAvancee.git>
 - *ssh://git@github.com:fabricehuet/programmationAvancee.git*
- Clonage

```
GIT $git clone git@github.com:fabricehuet/programmationAvancee.git
Cloning into 'programmationAvancee'...
Warning: untrusted X11 forwarding setup failed: xauth key data not generated
remote: Counting objects: 97, done.
remote: Total 97 (delta 0), reused 0 (delta 0), pack-reused 97
Receiving objects: 100% (97/97), 19.97 KiB | 4.99 MiB/s, done.
Resolving deltas: 100% (7/7), done.
```

Workflow

- On travaille toujours sur sa copie locale
- Il faut explicitement se synchroniser
 - Récupérer les modifications (*pull*)
 - Pousser ses modifications (*push*)

Git Data Transport Commands

<http://osteelle.com>

