gestion de sources

- Un gestionnaire de sources permet de conserver l'historique des modifications apportées à un ensemble de fichiers.
- Il permet à un ensemble d'utilisateurs d’interagir sur un même code source.
- Tous les gestionnaires de code sources sont basés sur les principes de diff et patch
- Il existe deux grandes catégories de gestionnaires :
  - les centralisés
  - les décentralisés

Les gestionnaires centralisés

- Historiquement, cvs (concurent versioning system) et son successeur svn (subversion)
- Les gestionnaires centralisés reposent sur un serveur central qui archive toutes les modifications apportées au code.
- Chaque modification incrémente un numéro de révision
- Les commandes de base de svn :
  - checkout, update, infos, status, commit, diff, revert

svn

- Chaque utilisateur interagit avec le serveur, il n'y a pas d'échange direct entre deux utilisateurs.
- Une méthodologie est associée à l'utilisation de svn, vous retrouverez cet méthodologie dans tout projet de développement (open source ou entreprise) => « svn red book »

svn : méthodologie

- Il est possible d'utiliser SVN juste pour le répertoire de développement principal.
- Seulement, comment faire si on a un « gros » développement à produire: si on transmet les modifications intermédiaires, le programme devient instable (ne compile plus par exemple...)
- Comment faire également pour gérer les versions:
  - On sort une version 1.0 qui évolue en 1.1 puis 1.2
  - On sort la version 2.0 (« casse » la compatibilité avec 1.x)
  - Un bug est trouvé dans la version 2.0: il faut le corriger dans la version courante mais également dans la version 1.x!

Un mot sur les versions...

- Il n'y a pas de « règles » universelles mais un certain nombre de pratiques.
- L'approche par numérotation est la plus répendue :

2.10.5

Major.Minor.Patch

Version.SousVersion.Patch

- Dans le cas d'une bibliothèque, on peut suivre les principes suivant :
  - Tant que l'API i.e. les fichiers d'entête et les fonctions (noms/signatures) ne changent pas, on garde le même numéro de version
  - On peut ajouter des fonctionalités : on incrémente le numéro de sous version
  - On peut corriger des bugs : on incrémente le numéro de patch

→ Un changement de version (Major) implique (sauf exception) une non compatibilité

Un mot sur les versions….

- Il existe d’autres stratégies de nommage/numérotation des versions.
- Par exemple, sur la base d’une date :
  - Ubuntu 18.04 => 2018 / Avril
- Parfois un mix :
  - Version.SousVersion.date(YYYYMMDD)
- On veille cependant à ce qu’il y ai un logique d’ordre
- Il est aussi apprécié que la logique de numérotation suit un ordre (lexicographique et/ou numérique).

svn : méthodologie

- La gestion des versions implique de maintenir plusieurs « répertoires » de développement en parallèle.
- Une méthodologie classique organise les sources ainsi:
  - / « racine »
  - /trunk : contient la version en cours des sources (dev.)
  - /branches/: des copies de trunk (« svn copy »)
  - /branches/1.0 : copie de trunk pour release (tests), retour de modif dans le trunk avec « svn merge » si compatible
  - /tags/1.0.0: version **figée** d'une branche qui est publiée, sert de référence. La branche correspondante est étiquetée (tag). **pas de commit/modifs dans ce répertoire**
  - /branches/modif_allali_155/: copie temporaire pour de « grosses » modifications avec retour dans le trunk par « merge ». Synchronisation régulière depuis le trunk (« merge »)

svn : la création d’un dépôt

- Le création d'un dépôt se fait à l'aide de la commande « svnadmin create nom_de_depot »
- possibilité d'ajouter l’exécution de script avant/pendant et après les « commits »
- possibilité d’envois de mails lors des commits
- facile à mettre en place sur son compte:
- cd ~/.depots/ ; svnadmin create SVN
- cd ~/; svn co file:///$HOME/.depots/SVN
- via ssh:
- svn co svn+ssh://allali@ssh.enseirb.fr/.depots/SVN

contrôle dans svn

- Il est possible d'avoir une approche hiérarchique dans svn en subdivisant le répertoire « branches » en répertoires et en ajustant les droits d’accès (lecture/ecriture) :
  - seuls les masters peuvent commiter dans « trunk »
  - les dev travaillent dans des branches
- Cela permet de faire de la revue de code

Les gestionnaires dé-centralisés

- Dans ce cas, il n'y a pas de dépôt central.
- Chaque utilisateur gère son propre dépôt.
- Un protocole permet l'échange de modifications (commit) entre deux utilisateurs.
- Exemple : git
- Commandes de base :
  - clone, add, commit, push, pull, fetch, merge, branch, checkout

git : les commits, pull et push

- Dans git, les commits sont locaux (il n'y a pas de dépôt central).
- On peut transmettre un ensemble (suite) de *commit* à un autre «dépôt» avec la commande *push*
- On peut réceptionner un ensemble de *commit* depuis un autre dépôt avec la commande *pull*.
- On peut ajouter/supprimer autant de dépôts connectés à notre dépôt avec « remote ». A faire dans les deux dépôts si l’on souhaite une symétrie.

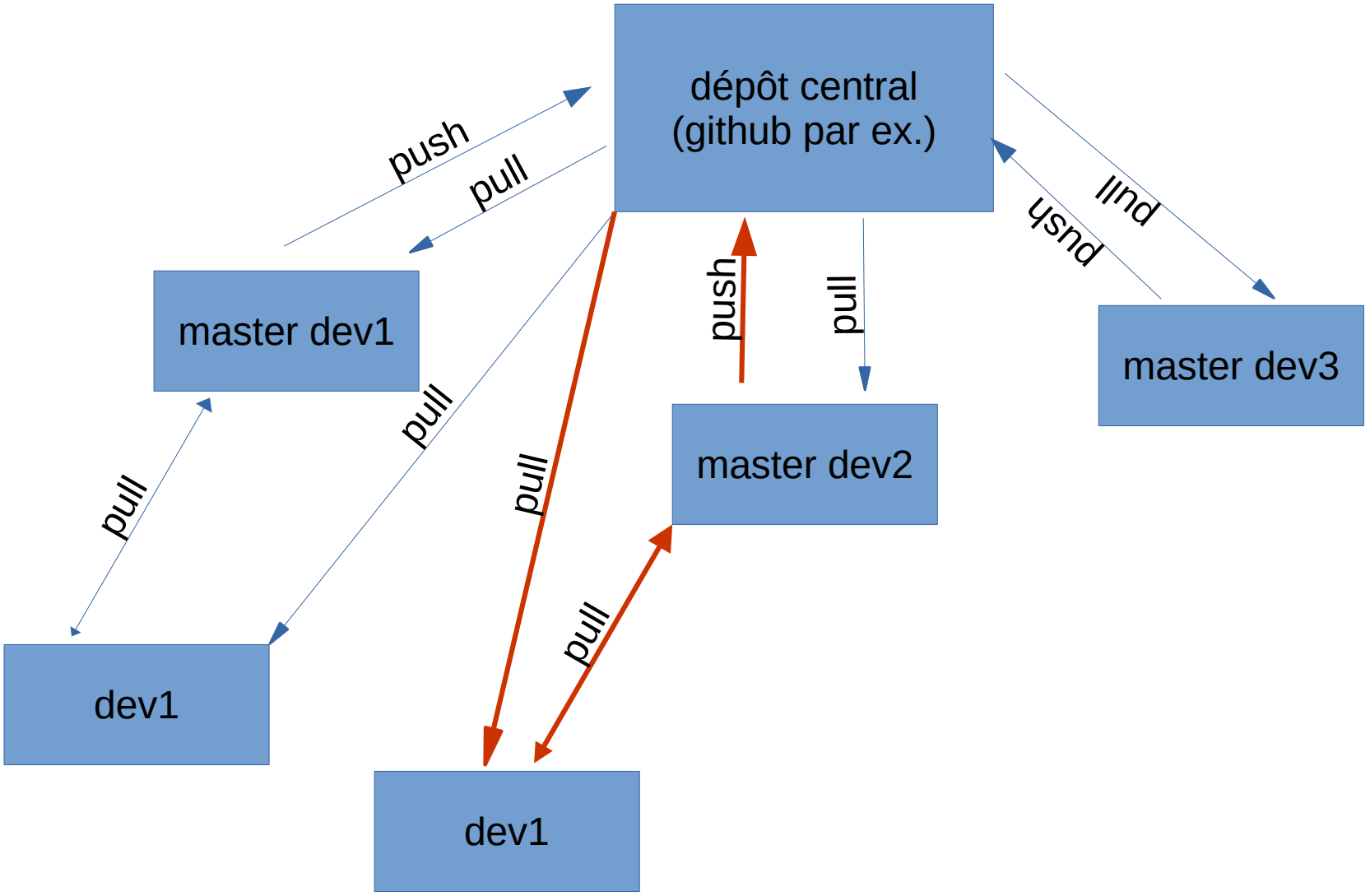
git

- git intègre une gestion native des branches :
  - création :
    - git branch b2
    - git checkout b2
    - ou bien git checkout -b b2
  - la fusion :
    - on bascule dans la branche qui doit recevoir les modifs
    - git checkout master ; git merge b2
    - Les modifications doivent avoir été *commitées* dans b2
  - la délétion : git branch -d b2
- La branche par défaut s'appelle master (paramètrable lors de la création d'un dépôt).

re-centralisation

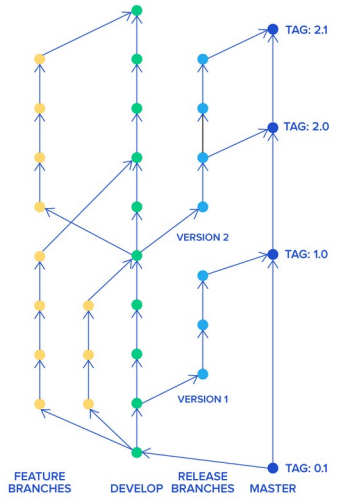
- L'avantage d'être en décentralisé et de pouvoir faire des « commit » sans connexion à un serveur.
- Pour la plupart des projets, il est cependant nécessaire d'avoir une référence : on utilise alors un dépôt git comme tel (github par exemple). Un tel dépôt s’appelle un « bare », il a la spécifité d’accepter les *push*
- On peut ensuite mettre en place un système de propagation hiérarchique des « commits ».

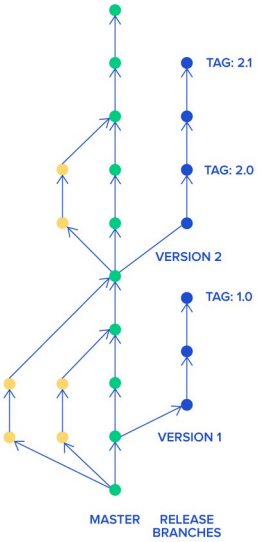
git


dépôt central
(github par ex.)
push
pull
master dev1
pull
pull
dev1
push
pull
master dev2
pull
dev1
llnd
ysnd
master dev3

git: méthodologie

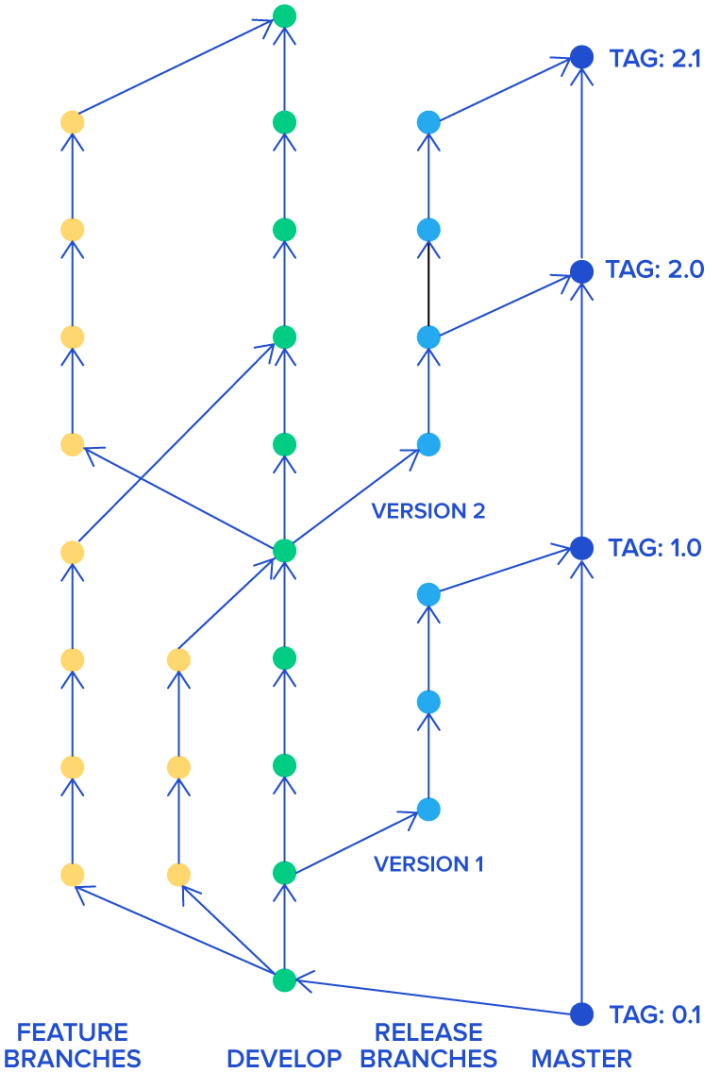
- L’utilisation de git repose nativement sur les branches
- Il y a deux approches actuellement:
  - git flow
  - trunk based





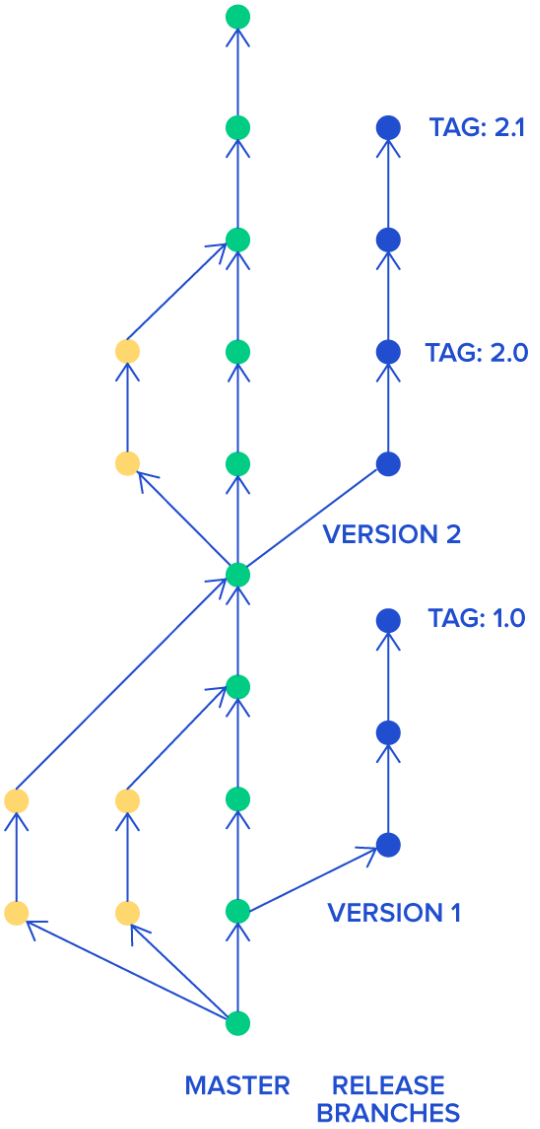
GitFlow

- *GitFlow* suppose des branches avec une durée de vie assez longue: il y a des aller-retour entre ces branches et une branches de développement (et les autres branches), elle même en synchronisation avec une branche de pré-release et la branche master de qui contient le code stable



Trunk Based

- *trunk based* privilégie les branches à durée de vie courte synchronisée avec master/trunk
- les release sont maintenues à coté.



GitFlow ou trunk based

- Les deux méthodologies ont leurs avantages et inconvénients
- L’environnement, le type de projet, le nombre de développeurs sont autant de facteurs qui plaident pour l’un ou l’autre des solutions

git ou svn?

- Différences majeures entre svn et git est :
  - centralisé / dé-centralisé
  - support natif du système de branches dans git
  - commit locaux dans git
- aujourd'hui git est de loin de le gestionnaire de sources le plus utilisé

Les autres gestionnaires

	Open Source	Centralisé	Décentralisé
CVS	●	●	
SVN	●	●	
GIT	●		●
SourceSafe		●	
Mercurial	●		●
Bazaar	●		●
BitKeeper			●
Team Foundation Server		●	

Il en existe bien d’autres (voir : https://en.wikipedia.org/wiki/List_of_version-control_software)

les plateformes

- De nombreuses plateformes proposent l'hébergement de dépôts telles que github et gitlab

  mais aussi bitbucket...

- ajout de services tels que l'intégration continue, suivi de bugs, wiki etc....