Gestion du code Source avec Git

Webinaire

Cognitic Sprl

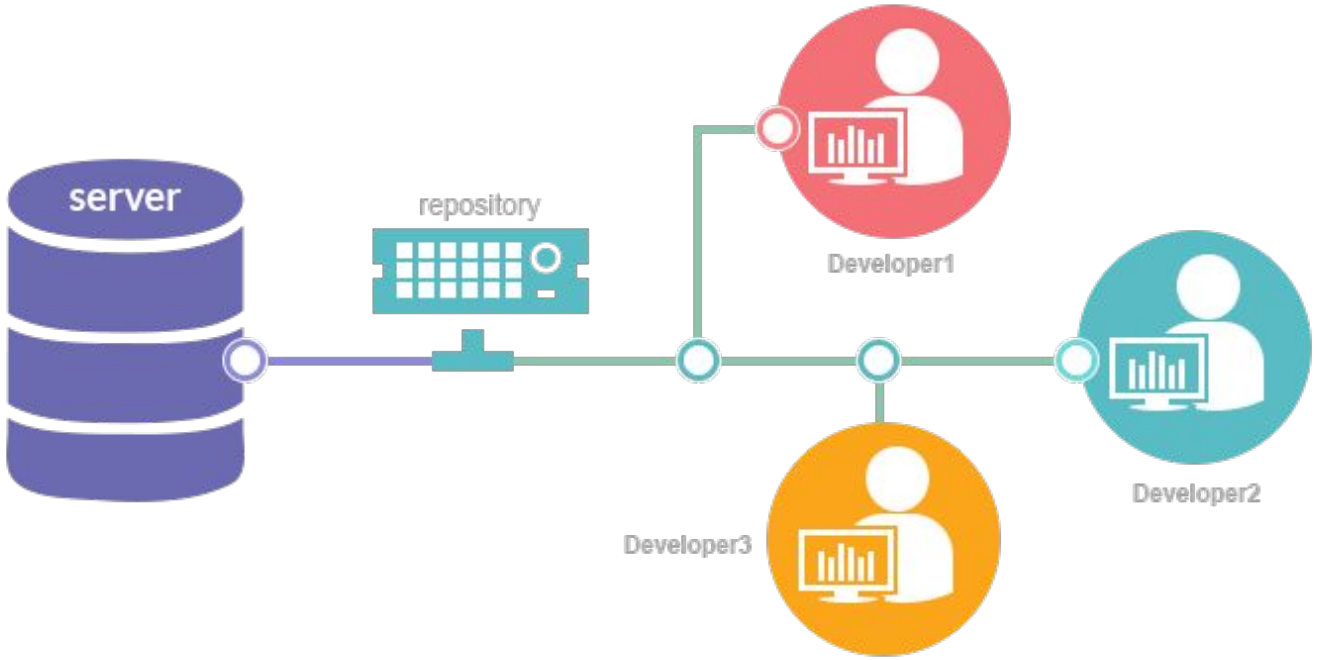
Git : Un système de contrôle de version

•Un VCS (Version Control System) est un logiciel pour stocker, suivre et partager vos données .

•BUTS :

- Sauvegarde et restauration (time travel)
- Travail isolé (branching)
- Collaboration entre les members d'une même équipe
- Intégration continue et développement
- Tracing

- GIT est un **logiciel de gestion de versions**, conçu par Linus Torvalds, l'auteur du noyau Linux.
  Gratuit et open-source, il est actuellement distribué par Software Freedom Conservancy.



Git : Un système de contrôle de version

- Qu’est-ce que la gestion de version et pourquoi devriez-vous vous en soucier ?

Un gestionnaire de version est un système qui enregistre l’évolution d’un fichier ou d’un ensemble de fichiers au cours du temps de manière à ce qu’on puisse rappeler une version antérieure d’un fichier à tout moment.

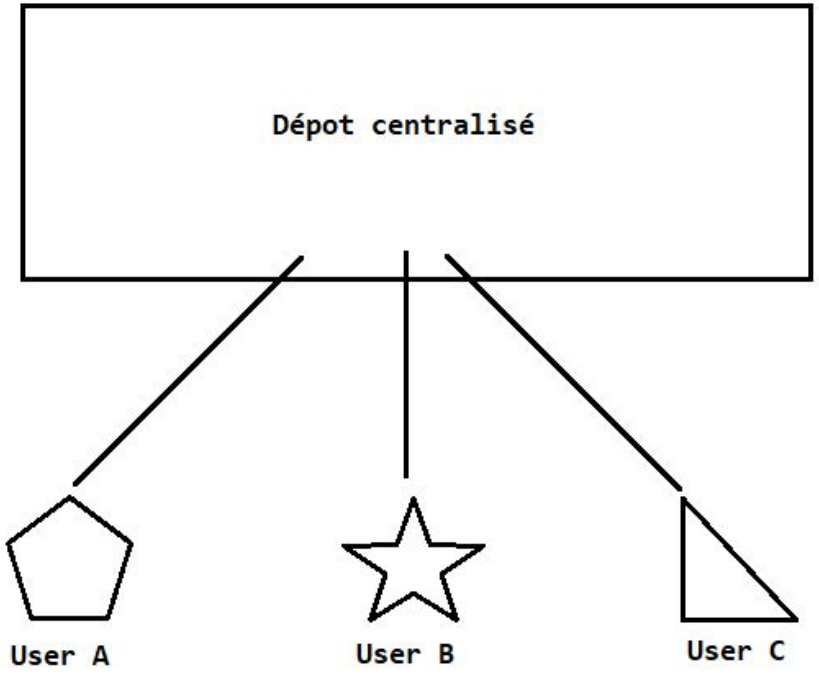
Il permet donc de :

- ramener un fichier à un état précédent
- ramener le projet complet à un état précédent
- comparer les changements au cours du temps
- voir qui a modifié quelque chose qui pourrait causer un problème
- …

Centralisé VS Décentralisé

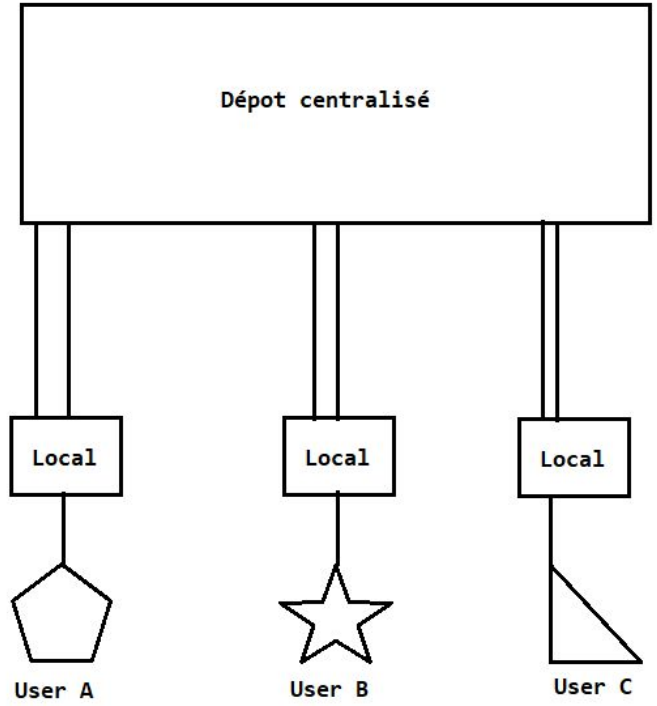
Centralisé

Tous les users lisent et écrivent dans le même dépôt distant…



Décentralisé

Chaque user a sont dépôt local ou il valide ses changements avant de synchroniser avec le serveur distant. Limite le risque



Git, Github , GitLab,...

Git, Github , GitLab,…

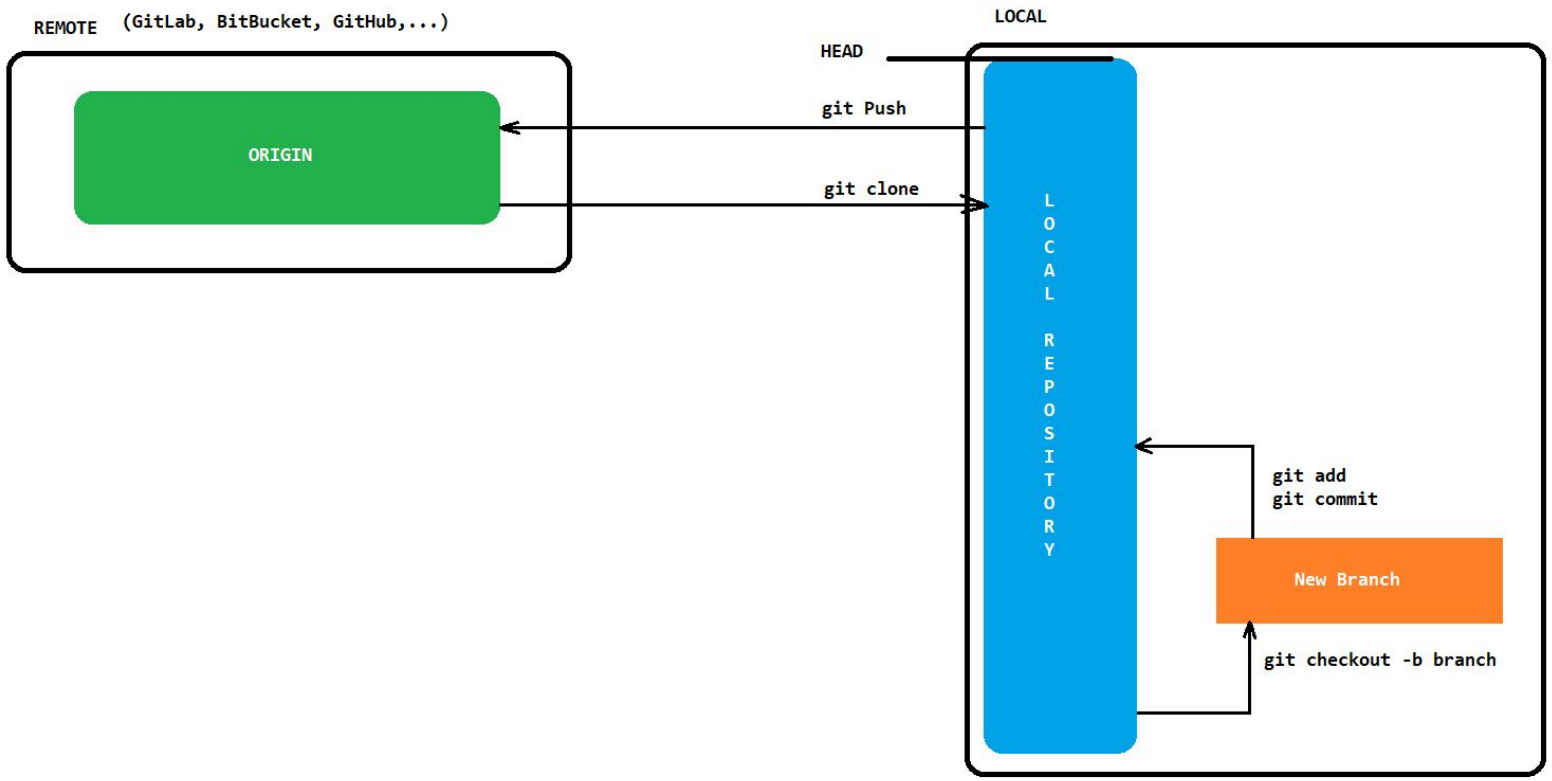
Git = **outil** qui permet de gérer différents projets en les envoyant sur un serveur permettant le partage et la collaboration ainsi que le suivi des modifications, des versions

Github, GitLab,…sont des **plateformes** dont le rôle est d'héberger les projets GIT et de proposer une interface graphique permettant de faciliter la gestion du dépôt GIT.

Principes

Principes

- Origin
- Local Repository
- Clone
- Branch
- Checkout
- add
- commit



Les commandes Git de base pour tout développeur

- **Git pull**

  permet de récupérer tous les changements sur la branche distante

- **Git fetch**

  permet de rechercher et afficher les changements sur un remote passé en argument, qui ne sont pas présent en local, sans **aucun transfert de fichiers.**

- **Git rebase**

  permet de transférer les changements d'une branche à une autre

- **Git checkout**

  permet de se déplacer d'une branche à une autre

- **Git add**

  permet d'ajouter les changements que nous avons fait dans nos fichiers sur la branche courante

- **Git commit**

  permet commiter les modifications que nous avons en local sur la branche courante

- **Git reset**

  permet de supprimer tous les changements que nous avons fait sur la branche courante

Les commandes Git de base pour tout développeur

- **Git push**

  permet d'envoyer les modifications sur un remote.

- **Git status**

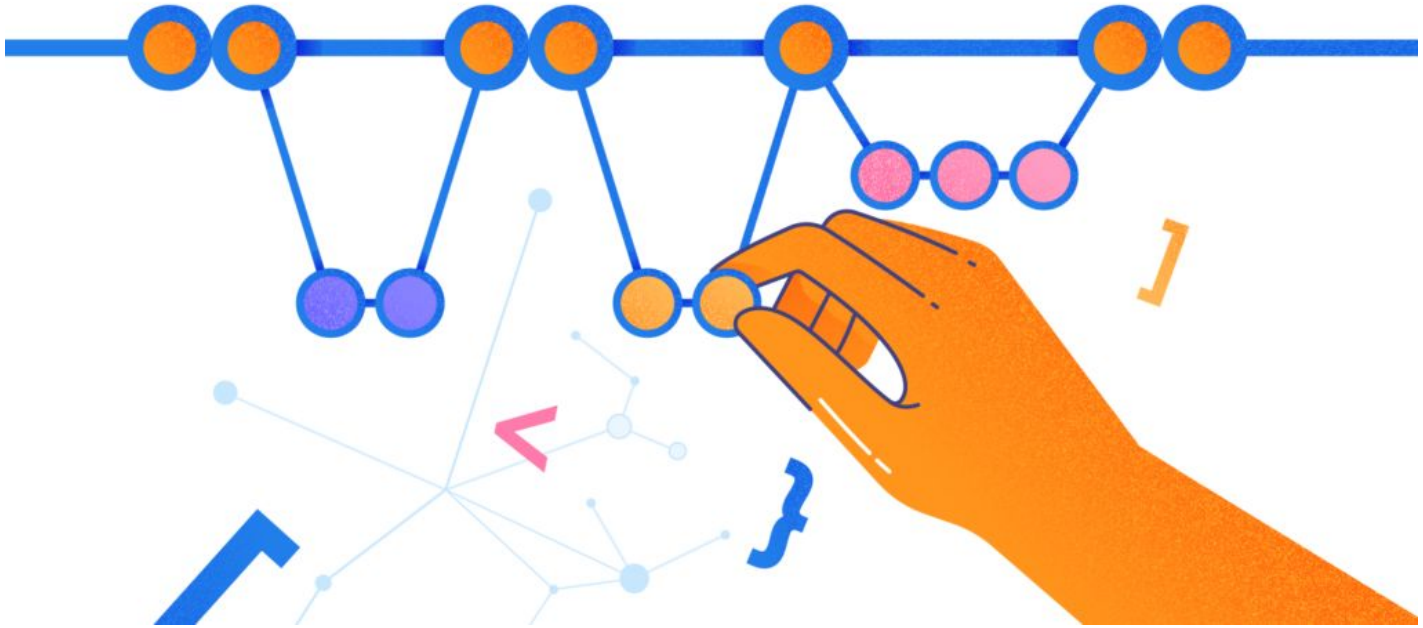
  permet d'afficher toutes les modifications non commités sur la branche courante

Toutes ses commandes sont essentielles et même si il existe des outils graphiques comme Kraken, smartGit, … il est important de pouvoir les écrire et les connaitre

Les branches

C’est un concept HYPER IMPORTANT lorsque nous travaillllons avec GIT…

Une bonne organisation des branches nous fait gagner du temps et nous facilite la livraison des produits développés

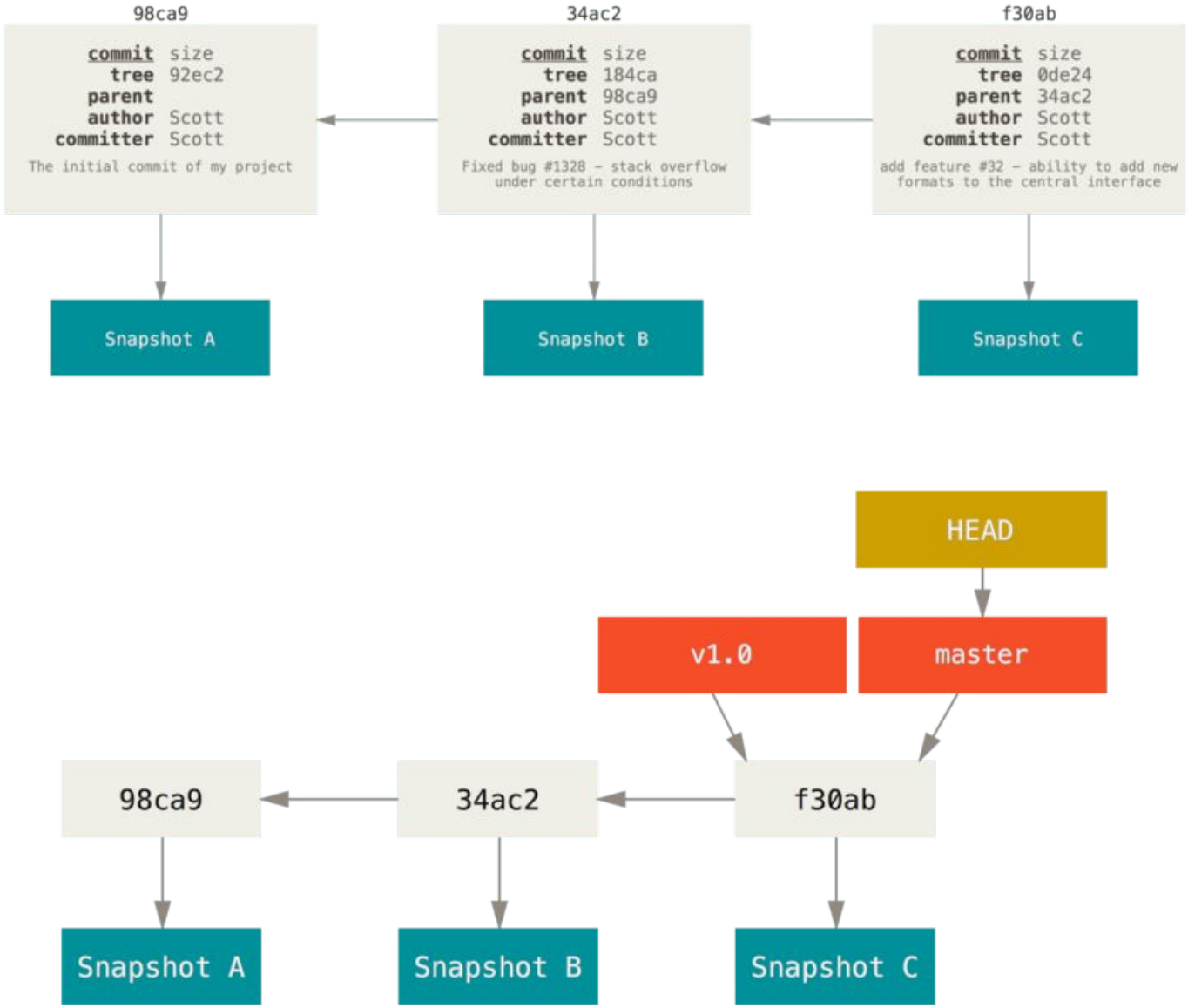


Les branches

Git ne stocke pas les données sous formes de modifications mais sous la forme d'instantané (snapshot)

Lors d'un commit, Git enregistre les fichiers dans un arbre et créé un objet commit (contenant le message, l'auteur, etc..) pointant sur cet arbre.

Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces **commits**



Les branches

La gestion des branches est le principe de base pour avoir un GIT utilisable, exploitable et partageable

Organisation des branches

- Master ou Main

  La branche par défaut.

  Elle doit être stable et ne doit pas permettre de commit directe. Seul les merges devraient être autorisés après un code review

- Dev

  La branche principale de développement.

  Elle doit contenir le code qui sera changé, adapté, testé par les développeurs avant d'être fusionnée vers la Main

C'est le minimum de branches que nous devrions avoir sur notre dépôt.

Model de branches

Afin de permettre le développement parallèle , il est plus que conseillé de mettre en place un modèle plus complet pour notre gestion de branches.

Dans ce modèle, en plus des branches Master/Main et Dev, nous aurons des branches « temporaire » pouvant être supprimées après utilisation et ayant des noms évoquant leur utilité :

- Features
- Release
- Bugs
- HotFix

De cette façon, vous pouvez organiser le travail d'équipe et vous définissez des règles précises concernant ces branches (origine -> Merge)

Remarques : Même le modèle le plus poussé de branching ne remplace pas la communication dans l'équipe et la rigueur des développeurs

Exemple de Modèle de Branching

Modèle de Branching

Quelques règles :

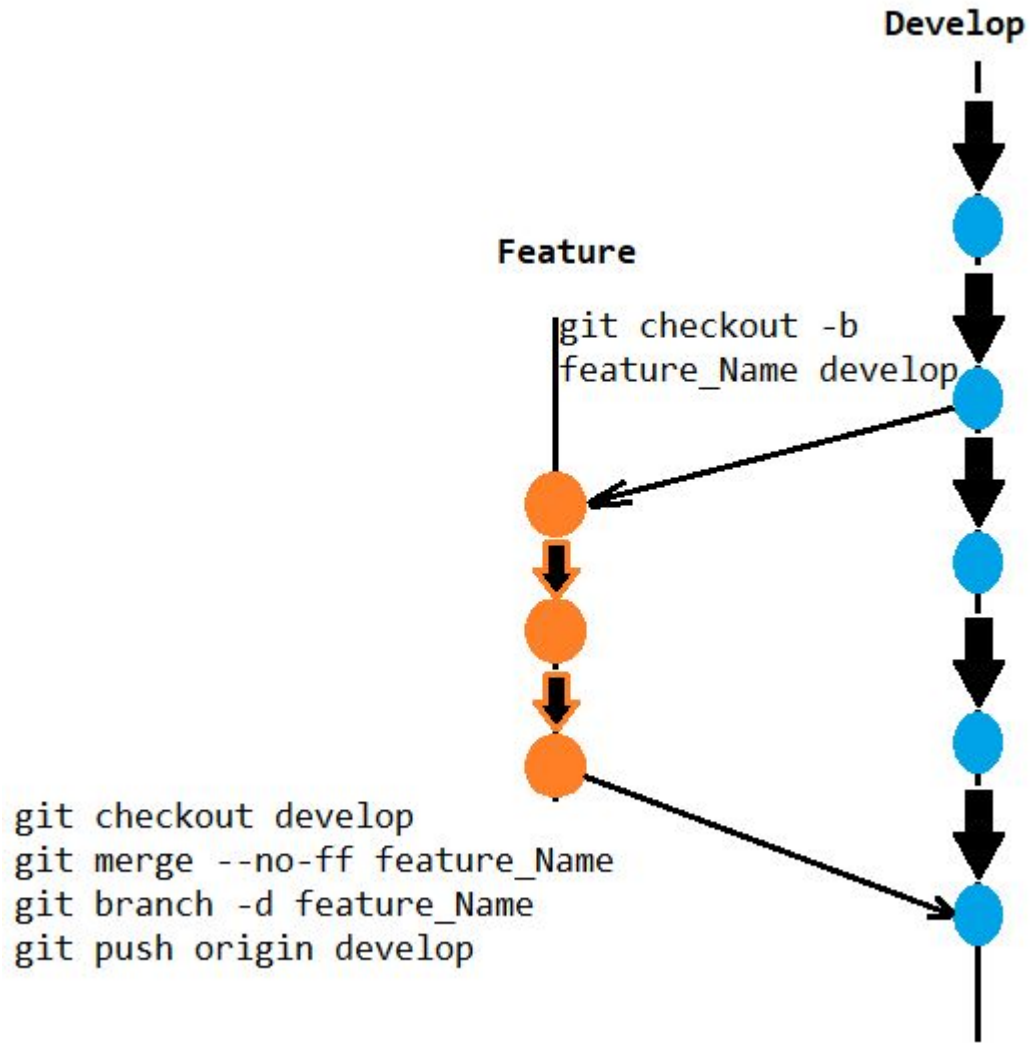
- Commencer le nom de vos branches par un mot ciblant son but
  - Bug
  - Feature
  - …
- Si vous utiliser un outil de gestion de projet (Jira, Azure devops…), incorporez le trakingID utilisé dans cet outil
  - Jira : IT-2001 Login Process
  - Branche : Feature_IT-2001_OpenIdImplementation
- Utilisez l'underscore ou la notation camelCase/pascalCase

Modèle Branching

- **Premier Type de Branche : Feature**
  - **Branche originale :** Dev/Develop
  - **Merge To :** Dev/Develop
  - **Convention de Nommage :** **Feature**_*TrackId*_*Nom*

Utilisée pour dévelloper les fonctionnalités de l'application finale.

Lorsque la feature est terminée, le merge vers Dev se fait et elle disparait



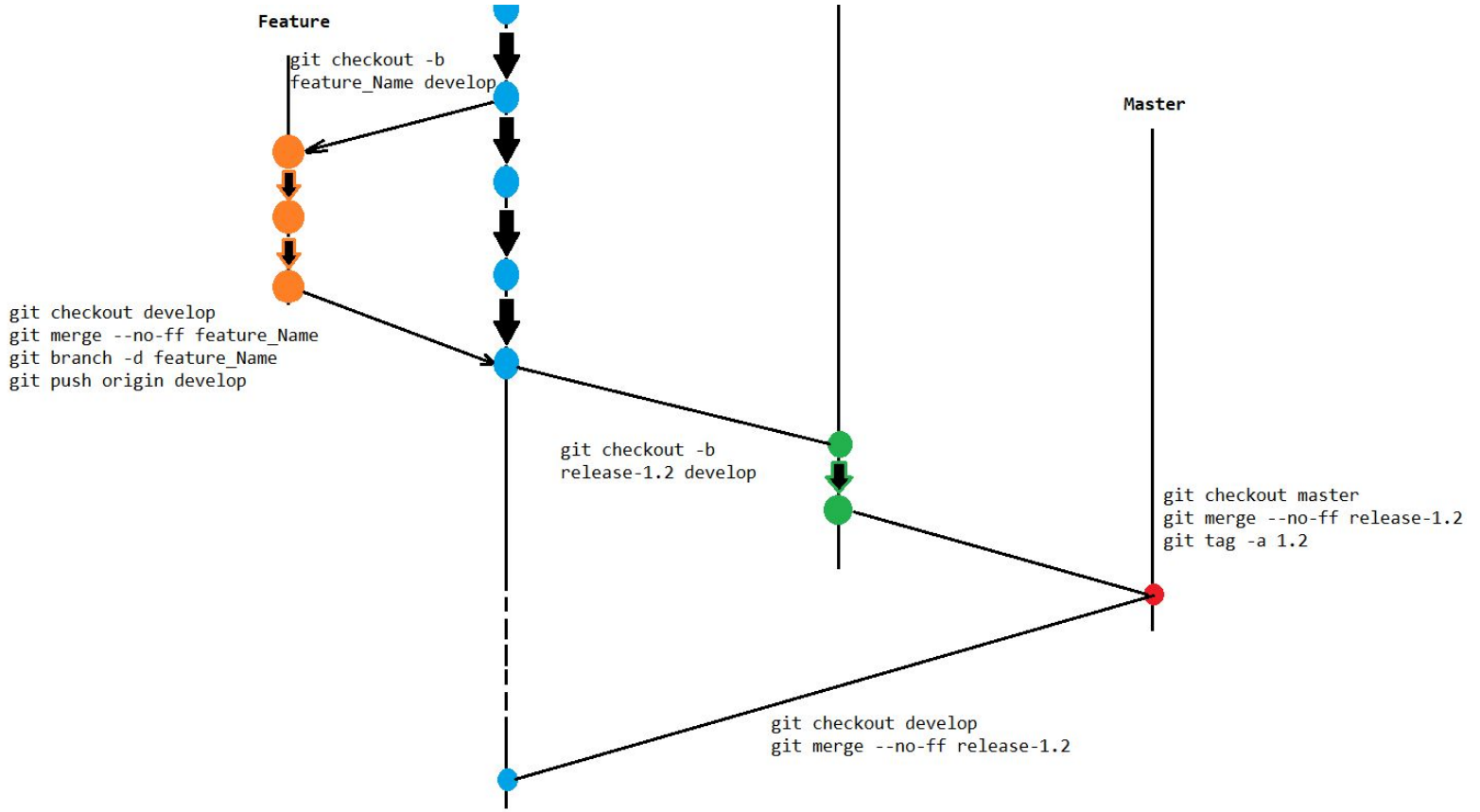
Modèle Branching

•Deuxième Type : Release

- **Branche originale :** Dev/Develop
- **Merge To :** Dev/Develop ou Master
- **Convention de Nommage : Release-***

•Dernier rempart avant la mise en production, elle autorise des corrections de bugs mineurs ou l'application de Hotfixes.

•Cette branche recevra les modifications de la branche Dev lorsque celle-ci reflètera au mieux le résultat attendu



Modèle Branching

- **Les HotFixes:**
  - **Branche originale :** Main/Master
  - **Merge To :** Dev/Develop et Master
  - **Convention de Nommage : HotFix-***

Cette branche contiendra les modifications concernant des fonctionnalités « oubliées » ou « Buggées » devant être livrée en même temps que la release actuelle ou portant sur une release précédente si l'actuelle n'est pas encore en production

Elle permet a l'équipe de continuer a avancer sur les développements actuels en permettant à 1 ou plusieurs développeurs de préparer les fix pour la production