Industrialisation et Continuous Delivery

GÉRER SES SOURCES ET LES LIER À SON INTÉGRATION CONTINUE
AVEC L'UN DES OUTILS LES PLUS POPULAIRES

The world is how we shape it*

sopra steria

* Le monde est tel que nous le façonnons.

Vos intervenants

Benoit POIRIER
Architecte Solution
Sopra Steria
benoit.poirier@soprasteria.com

Fabrice ROULAND
Expert Technique
Sopra Steria
fabrice.rouland@soprasteria.com

Ce que nous allons aborder

- Les bases de GIT
- GitLab en tant que Git Server
- Les workflows Git
- Les principes de GitFlow

01

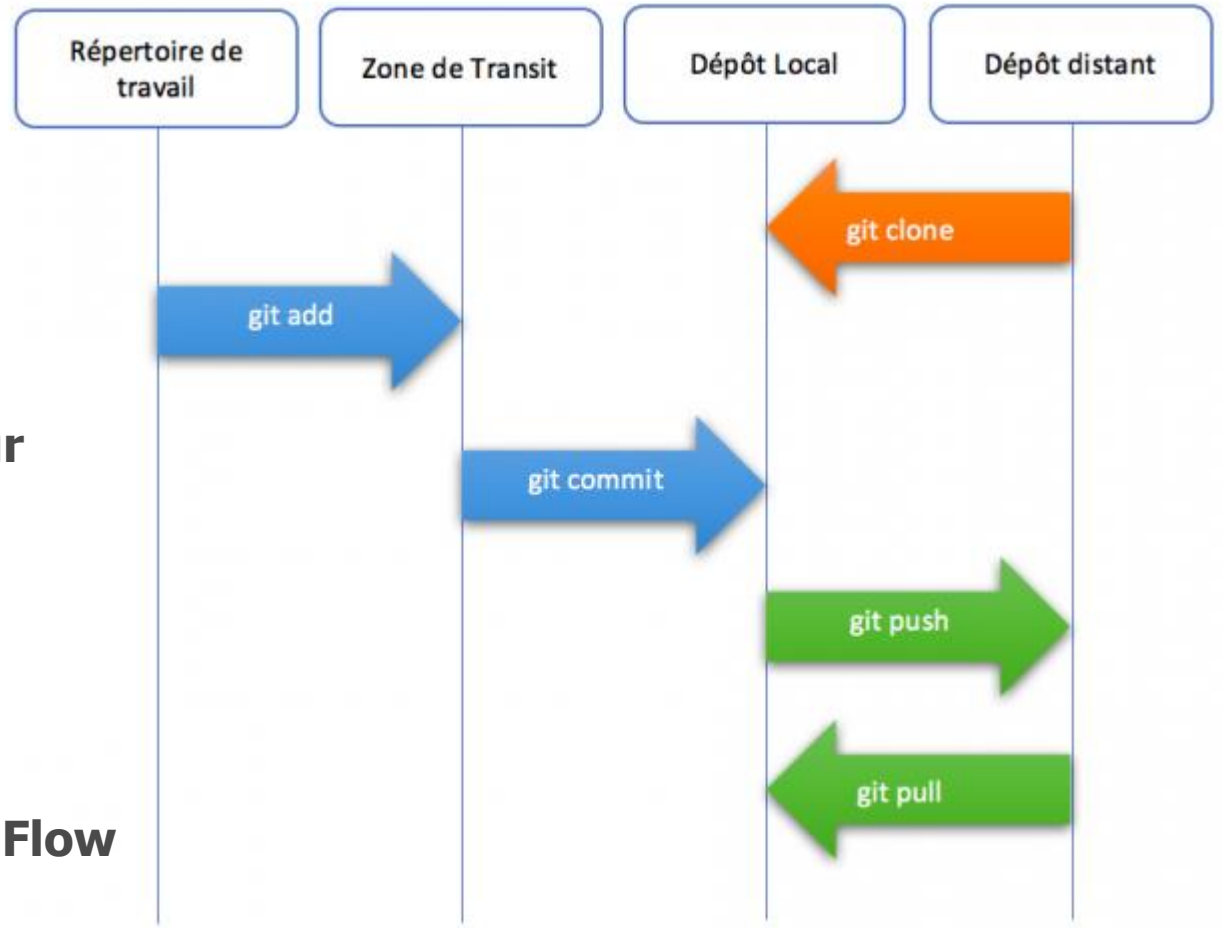
Les bases de GIT

VOYONS TOUT CELA PLUS EN DETAILS

C’est quoi Git

Le gestionnaire de sources

- **Gestionnaire de sources décentralisées**
- **Très populaire et performant**
- **Souplesse au niveau du poste du développeur**
- **Permet de travailler en mode déconnecté**
- **Favorise l’expérimentation et le test**
- **Méthode de gestion des sources proposé Git Flow**

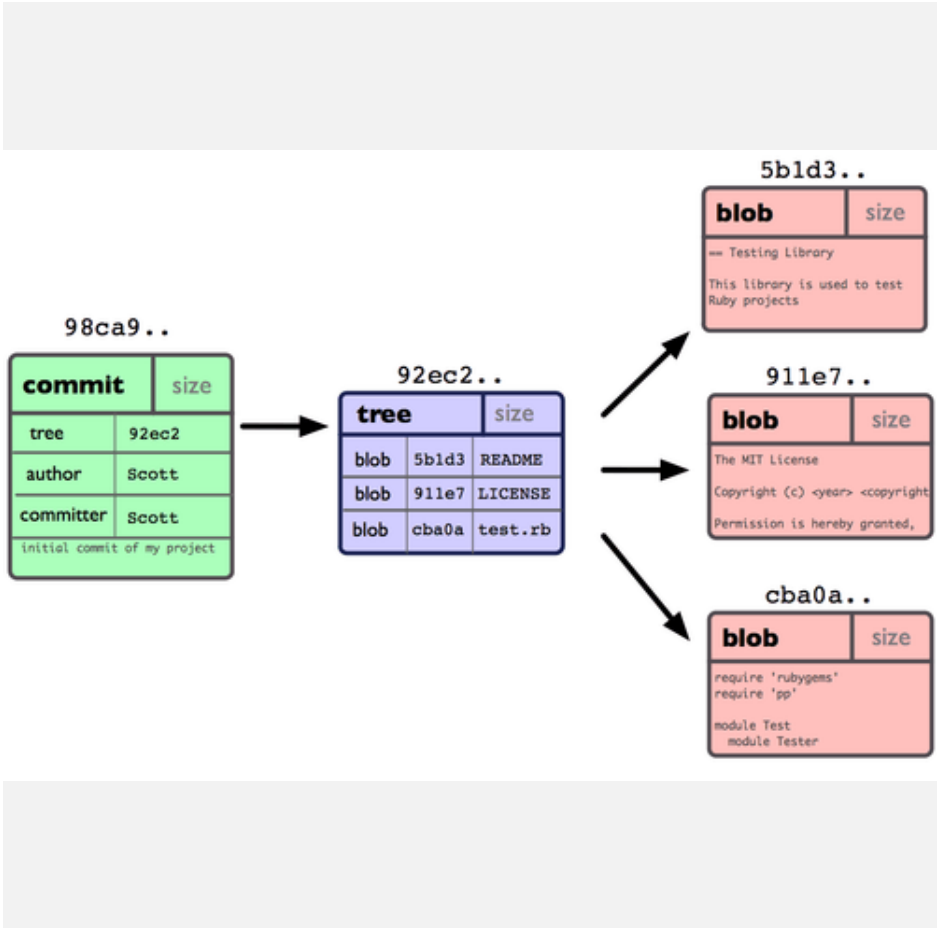


sopra steria

Les bases de Git

Stockage des données

- **Pas une série d'ensembles de modifications ou différences mais une série d'instantanés**
- **Lorsqu'on soumet du code dans git on enregistre un objet commit qui contient**
  - un pointeur vers l'instantané du contenu indexé
  - les méta-données d'auteur et de message
  - le(s) pointeur(s) vers le(s) **commit(s)** qui sont les parents directs de ce **commit**
- **Lorsqu'on commit, Git :**
  - fait la somme de contrôle pour chaque répertoire (**un tree**)
  - stocke ce **tree** contenant des pointeurs vers les fichiers, impacté par le **commit**, dans le dépôt Git
  - Stocke un **blob** par fichier



sopra steria

Les bases de Git

Les branches

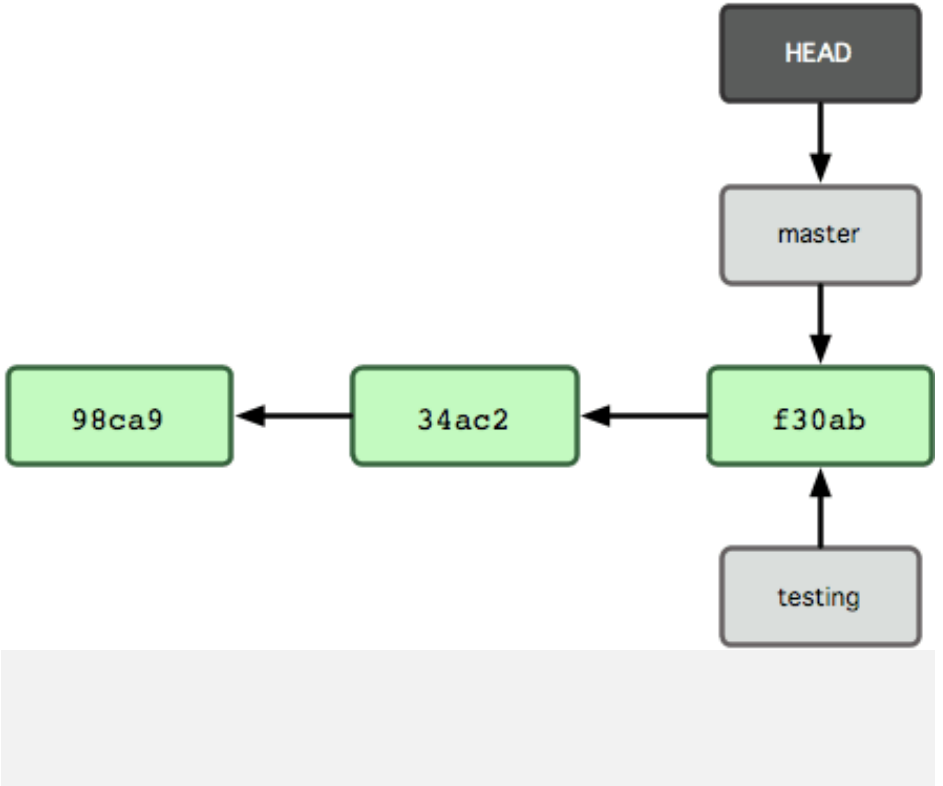
- Le modèle de branche représente la vrai différence de Git par rapport aux autres SCM
- Une branche est en réalité un pointeur vers un des commits évoqués plus tôt
- Pour que Git sache sur quelle branche vous vous trouvez, il utilise un pointeur spécial HEAD qui pointe sur votre branche en cours

SCM : Source Control Management

Commit : Objet contenant l'ensemble des informations sur un instantané de code

Branche : Pointeur vers un commit

HEAD : Pointeur indiquant sur quel branche nous nous trouvons



sopra steria

Les bases de Git

Avantages des branches

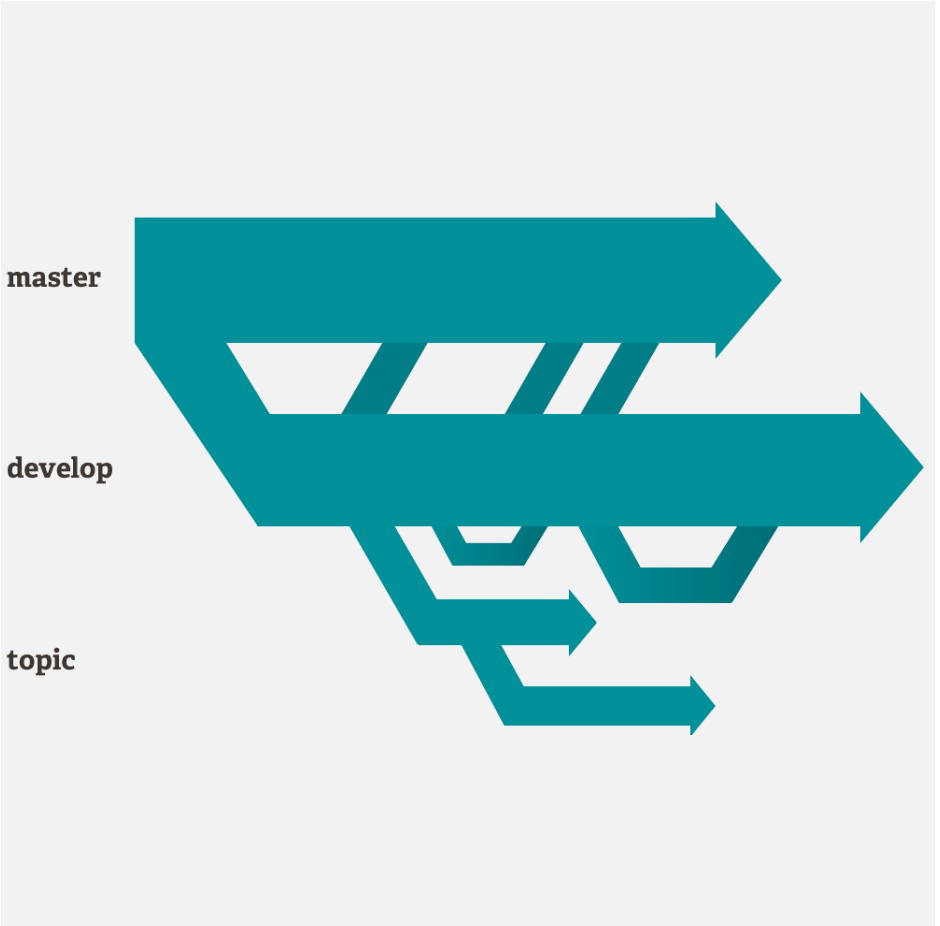
- **Git encourage le fait d'avoir plusieurs branches locales complètements indépendantes :**
  - Changement de contexte avec peu de conflits

    Vous pouvez créer une branche pour commencer à développer une fonctionnalité, faire quelques commits, revenir à la branche d'origine, appliquer un patch et revenir sur la branche que vous venez de créer pour la merger
  - Une branche par rôle

    Une branche dédiée à la production une autre pour le test qui pourra être mergée vers la production, une autre pour le dev, …
  - Workflow basé sur les fonctionnalités

    Créer une branche pour chaque nouvelle fonctionnalité, switcher de branche quand on le souhaite et les supprimer une fois qu'elles auront été mergées dans la branche principale
  - Expérimentations jetables

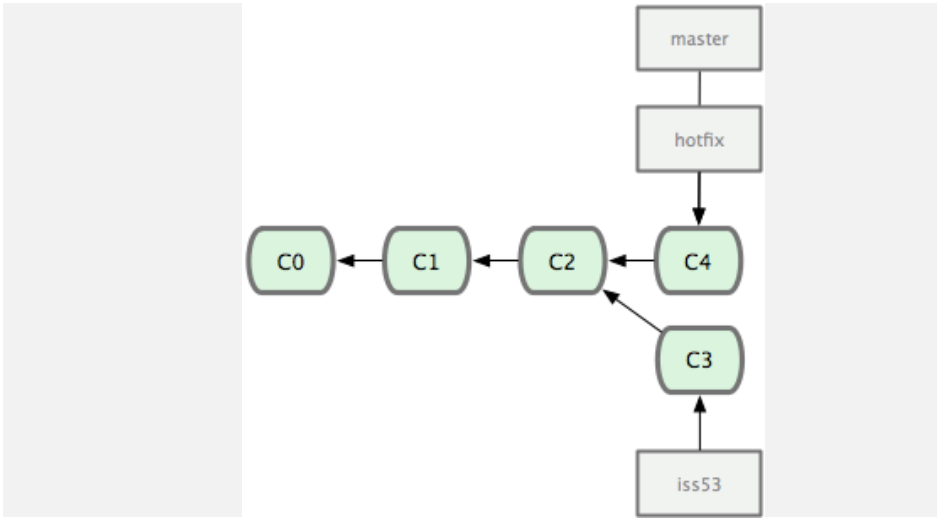
    Créer une branche pour expérimenter une fonctionnalité, la supprimer car non concluante sans que personne ne l'ai jamais vu

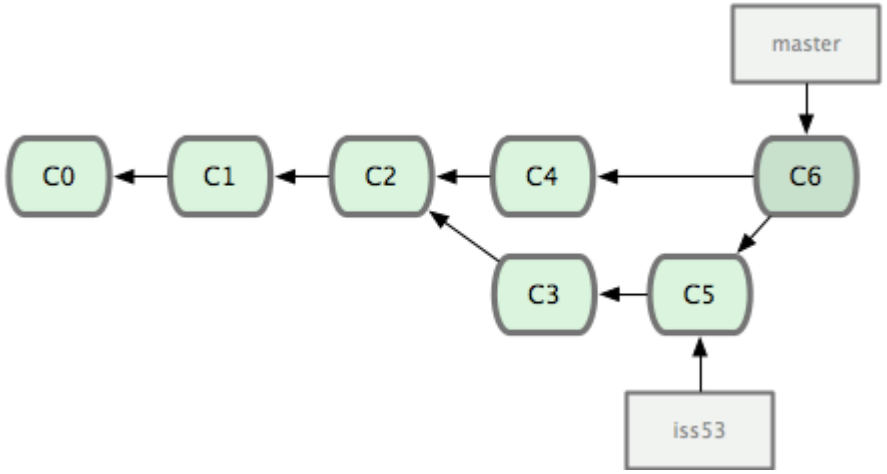


Les bases de Git

Les merges

- **Fast-forward**
  - Si nous souhaitons fusionner une branche contenant des modifications sur sa branche d'origine et que **la branche d'origine n'a pas avancé de son côté**
  - Git va **repositionner le pointeur** de la branche d'origine sur le commit où se trouve la branche contenant les modifications
- **Merge**
  - Si nous souhaitons fusionner une branche contenant des modifications sur sa branche d'origine et que **la branche d'origine a évolué indépendamment**
  - Git va **créer un commit de fusion** (ayant comme commits précédents le dernier commit de chacune des 2 branches) et repositionner le pointeur de la branche d'origine sur ce commit => Gestion de conflits





sopra steria

Les bases de Git

Petit et rapide

- **Toutes les opérations sont faites localement => plus rapide qu'un système centralisé nécessitant de communiquer constamment avec un serveur**
- **Git a été créé à l'origine pour gérer des répertoires très volumineux**
- **Il est écrit en C ce qui réduit le temps d'exécution comparé à des langages de haut niveau**
- **Le volume de transfert de données est très faible pour chaque opération (comparé à SVN)**
- **L'opération de clone est plus volumineuse ainsi que la taille du stockage côté client**

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push 1000 1k images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c - 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x
Clone	Clone in Git vs checkout in SVN	107.5	14.0	x7
Size (M)	Size of total client side data and files after clone/checkout (in M)	181.0	132.0	x1

Les bases de Git

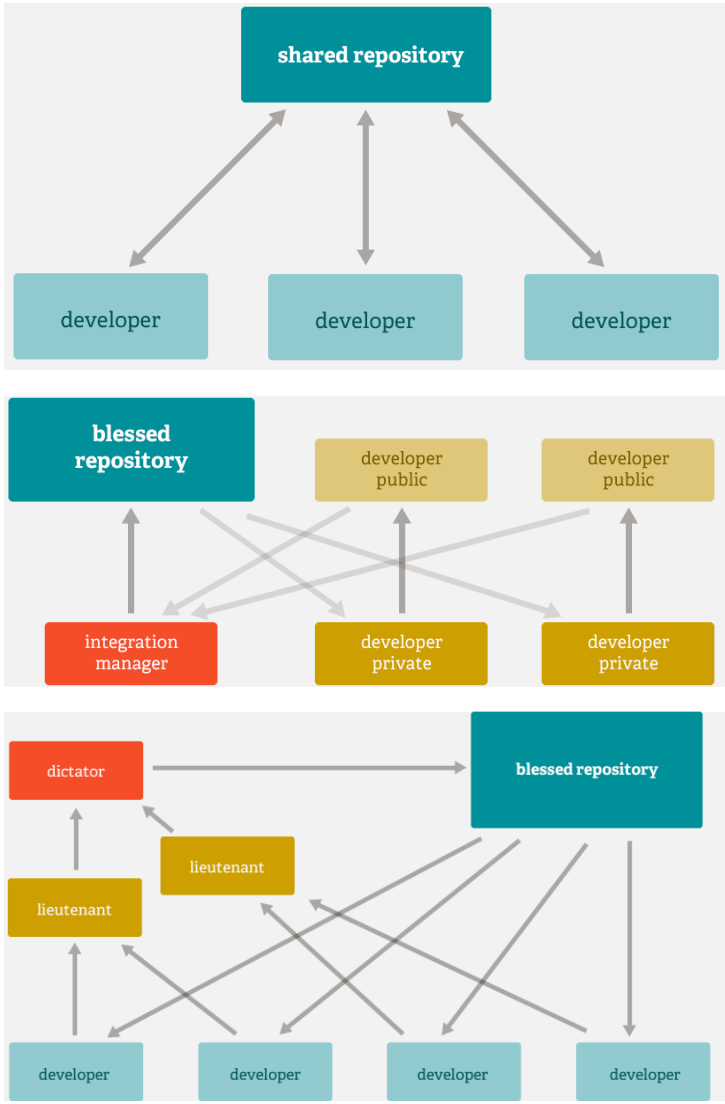
Distribué

- **Pour contribuer à un dépôt Git, il faut le cloner**
- **De cette façon, chaque utilisateur possède un backup complet du dépôt du server Git => Facilité de restauration si crash ou corruption**
- **Supporte n'importe quel type de workflow**
  - Subversif

    Tous les développeur travail sur un système centralisé. Git n'autorisera pas de push à un développeur si quelqu'un a push avant
  - Manager d'intégration

    Les développeur possède leur branches et demande à un manager d'intégration de pouvoir merger leur code dans le dépôt
  - Dictateur et lieutenants

    Dans de plus grosses équipes, les lieutenants sont les managers d'intégration de chaque équipe et le dictateur fait les merges de packages de chaque lieutenant dans le dépôt



sopra steria

02

GitLab en tant que Git Server

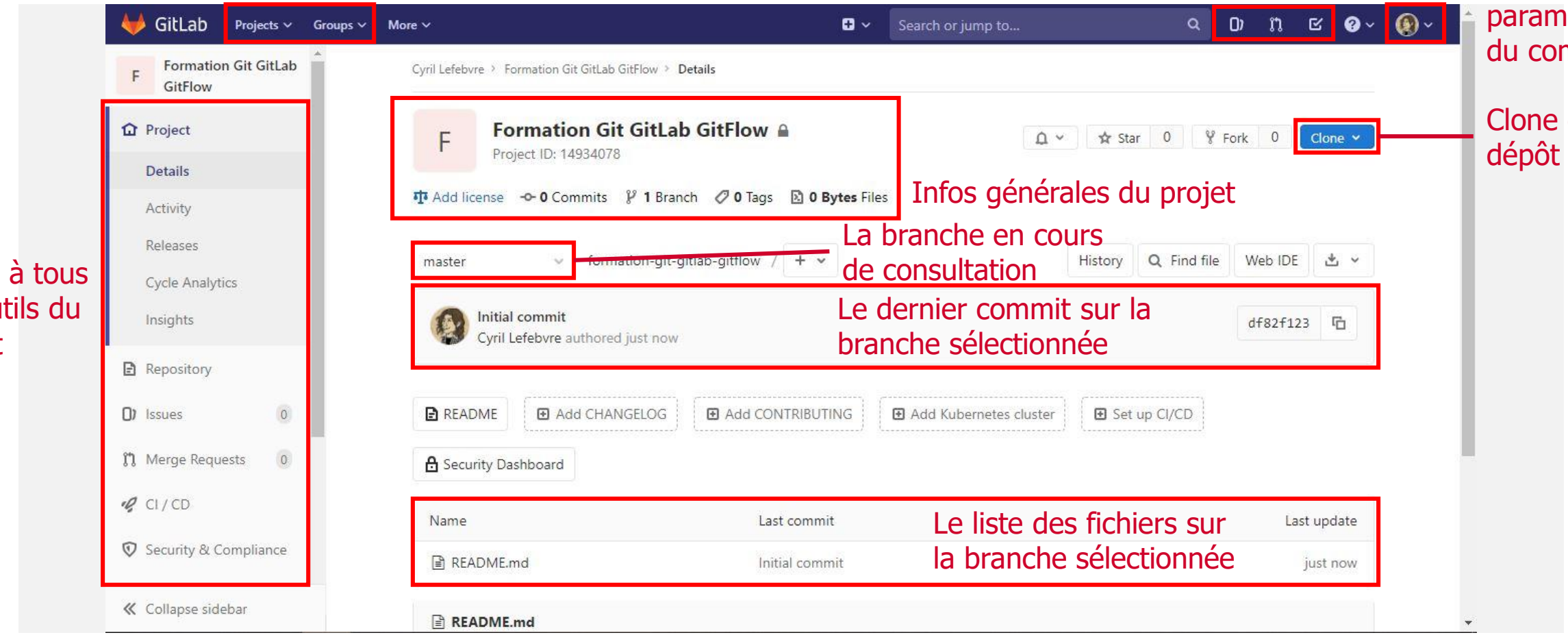
GitLab en tant que Git Server

Fonctionnalités

- **GitLab est une plate-forme basée sur Git qui intègre un grand nombre d'outils essentiels pour le dev, le déploiement et la gestion de projet :**
  - Hébergement de code source avec gestion de version
  - Suivi des propositions pour les nouvelles implémentations, des retours de bugs et des feedbacks
  - Organisation et priorisation avec le tableau de tâches
  - Revue de code sur les demandes de fusion
  - Build, test et déploiement avec Gitlab CI/CD
  - Déploiement de sites webs statiques avec GitLab Pages
  - Intégration avec Docker en utilisant GitLab Container Registry

GitLab en tant que Git Server

Overview



sopra steria

GitLab en tant que Git Server

Le code source, les commits, les branches, …

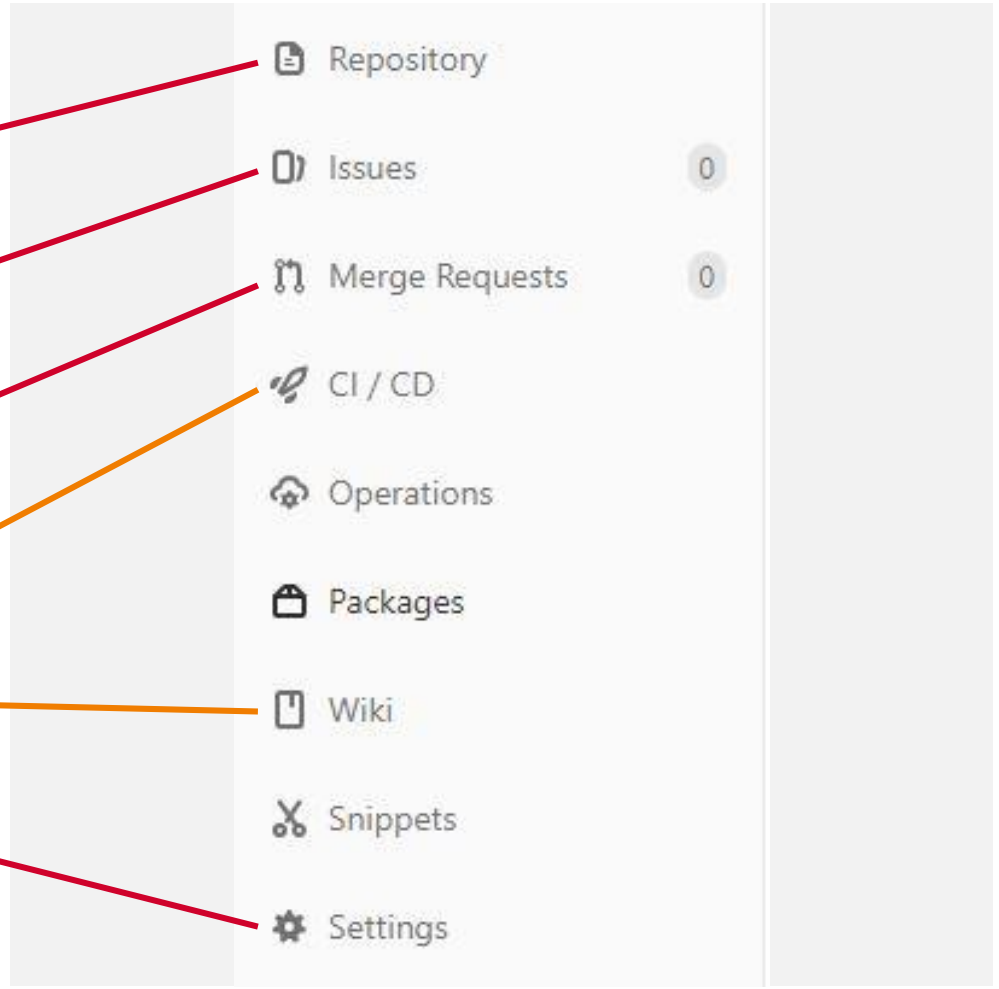
Board des développements en cours, le backlog, …

Gestion des fusions de branches

Outils d'intégration et de déploiement continu

Documentation sur tout le projet

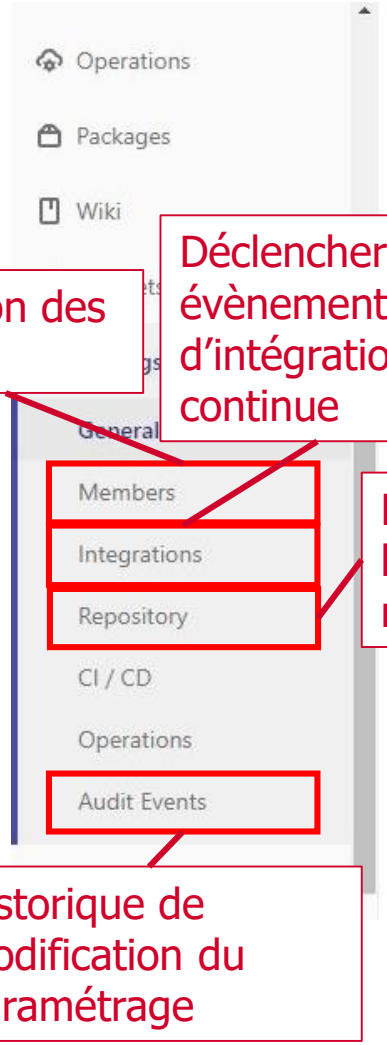
Paramétrage de tout le projet (membres, branches, …)



sopra steria

GitLab en tant que Git Server

Paramétrage d'un projet



Gestion des accès

Déclencher un évènement d'intégration continue

Branche par défaut, branches protégées, règles pour le push, …

Historique de modification du paramétrage

Qui peut voir le dépôt, les issues, …

Quelle méthode de merge utiliser, comment gérer les conflits, les prérequis à la validation

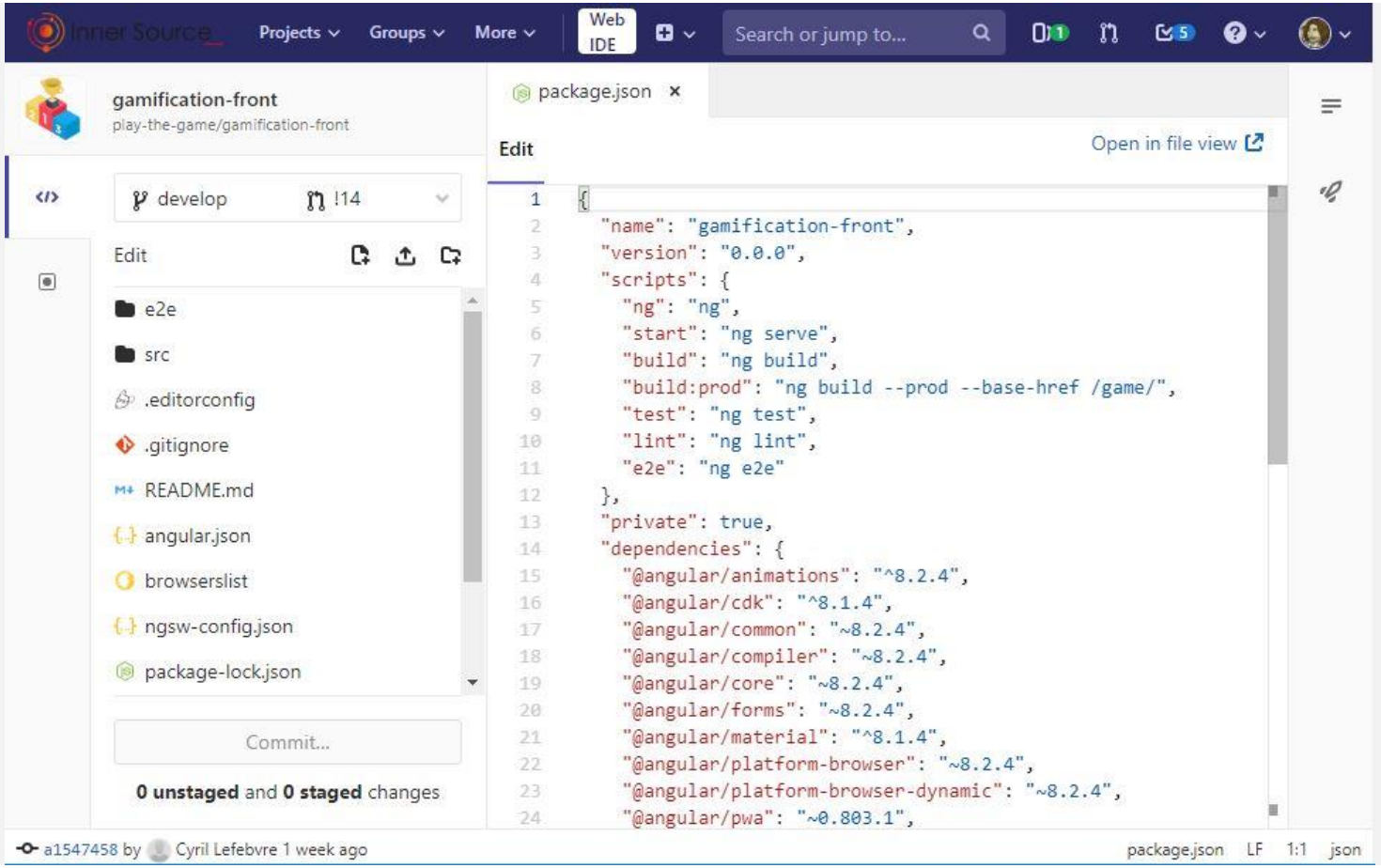
Qui peut approuver une merge request

Template par défaut des ticket de tâche (en markdown)

Transférer un projet, le supprimer, l'exporter, …

GitLab en tant que Git Server

Web IDE

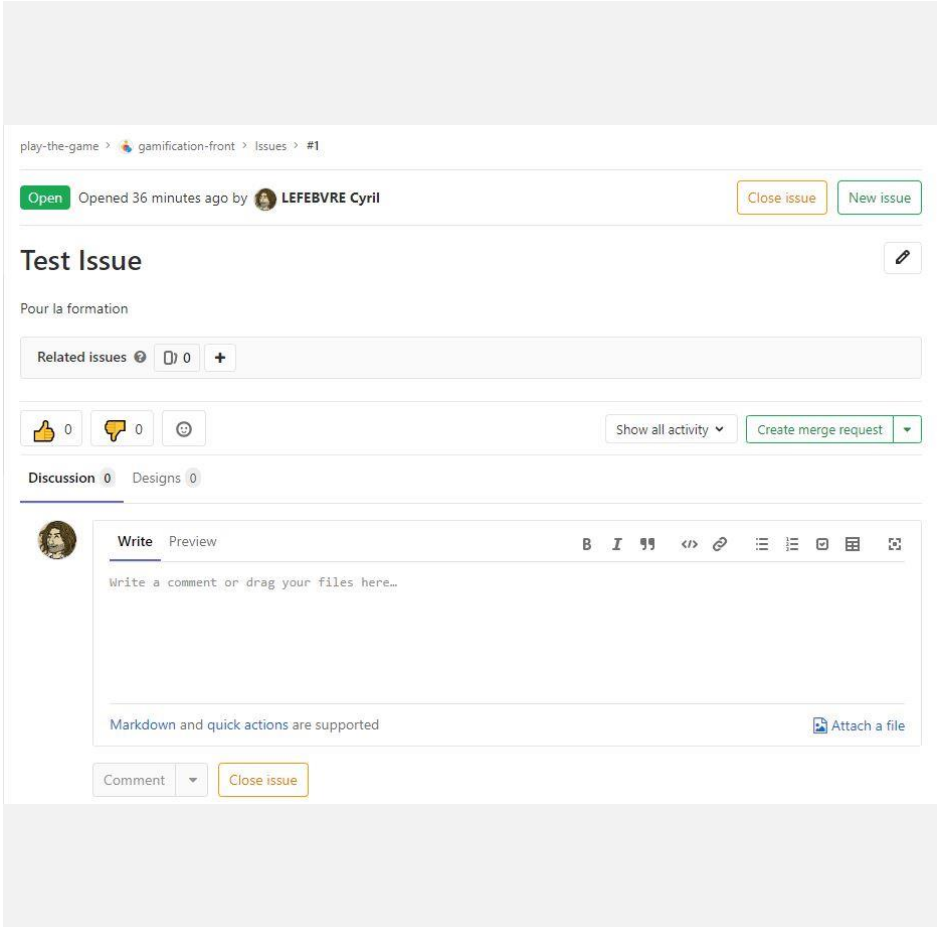


sopra steria

GitLab en tant que Git Server

Une issue

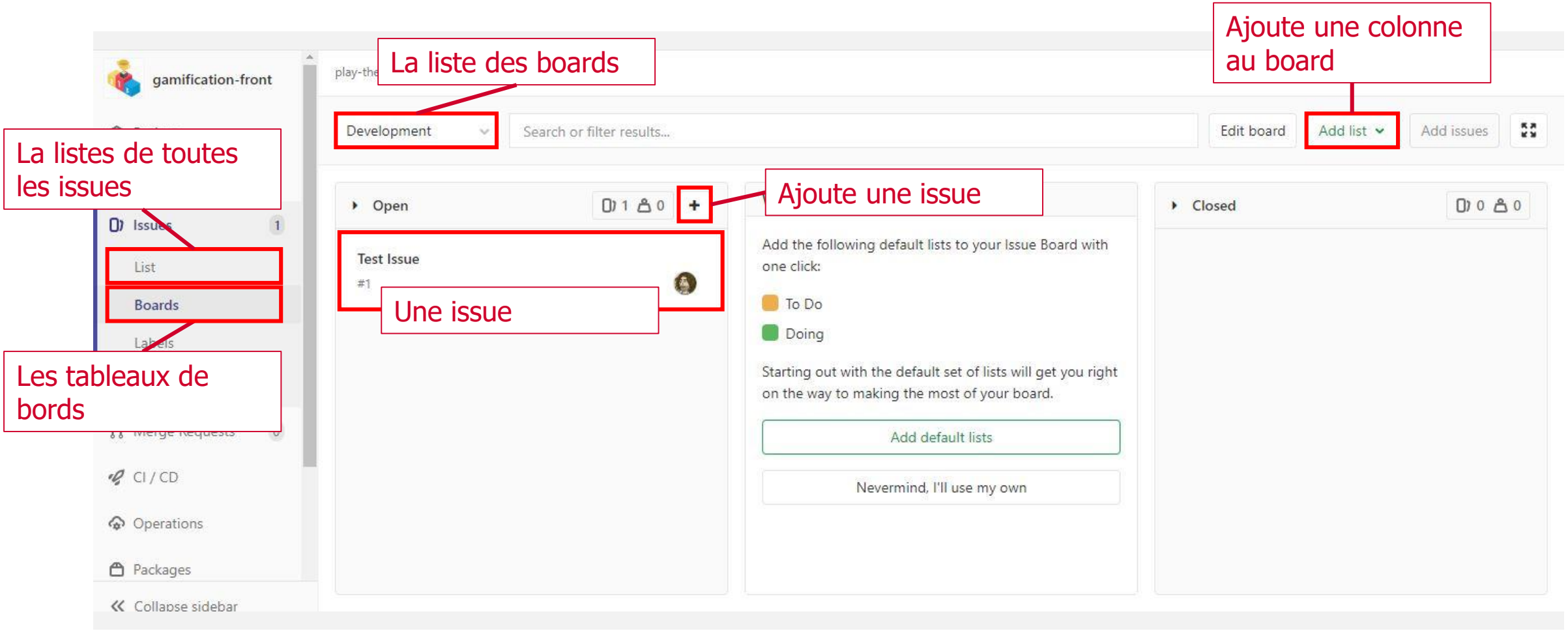
- **Une issue constitue un bug ou une fonctionnalité à réaliser**
- **Une issue dans GitLab contient :**
  - Une titre
  - Une description
  - Les issues liées à cette issue
  - Les discussions autour de l'issue
- **Les actions possibles sont :**
  - La clôture de l'issue une fois terminée ou non
  - La création d'une merge request à partir de cette issue pour créer directement une liaison entre l'issue et la merge request



sopra steria

GitLab en tant que Git Server

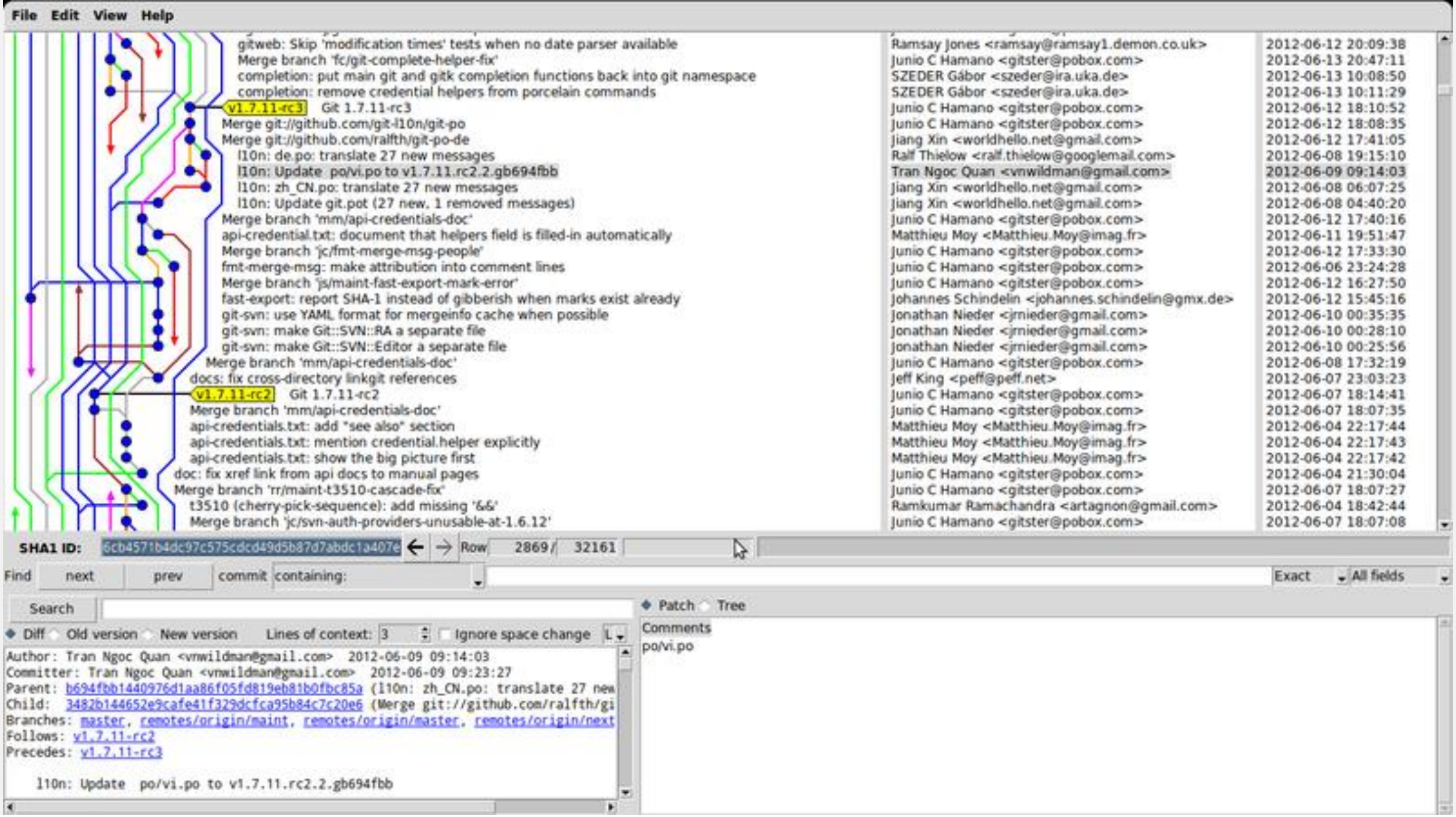
La gestion des issues



sopra steria

03

Les workflows Git



sopra steria

Les workflows Git

Un workflow pour les gouverner tous

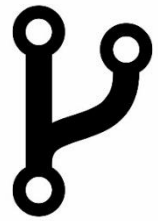
- **Un dépôt Git peut vite se transformer en arbre de Noël avec des guirlandes dans tous les sens**
- **Solution : un workflow de développement**
  - Pour unifier les pratiques au sein d'une même équipe
  - Pour simplifier la gestion du dépôt
  - Pour connaitre en temps réel l'état de son dépôt (les fonctionnalités en cours de développement, les branches pouvant être supprimées, …)

Feature Branch Workflow

La simplicité, penser petit et synchronisation régulière

GitFlow

Une solution robuste s'adaptant à tous les contextes



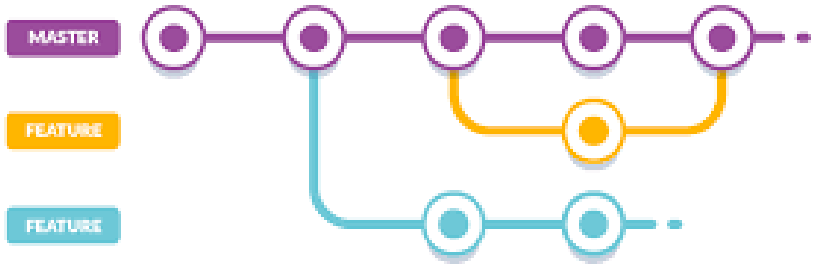
Fork & Merge

Chacun son dépôt distant, utilisé pour l'open-source

Les workflows Git

Feature Branch Workflow

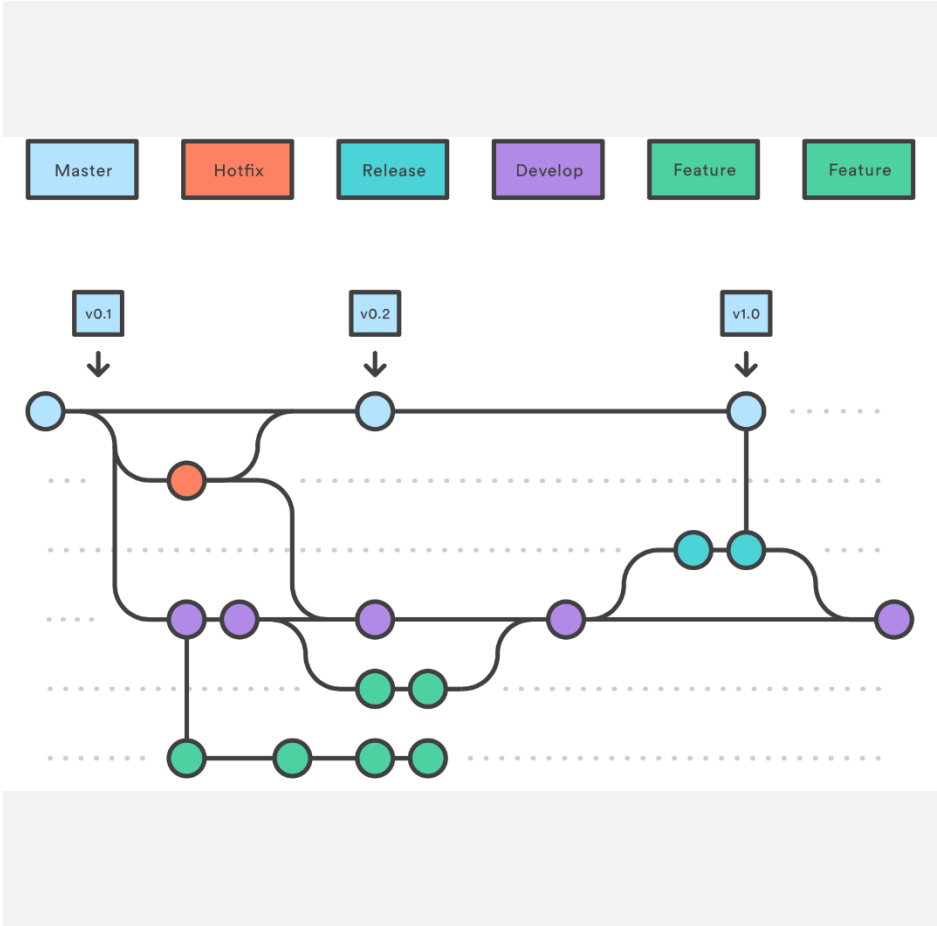
- **Workflow idéal lorsqu'il n'y a pas besoin de gérer des releases ou des versions**
- **Fonctionne surtout en équipe réduite**
- **Concept**
  - Une seule branche principale : master
  - Chaque fonctionnalité fait l'objet d'une branche feature tirée de la master
- **Principes**
  - Tout ce qui est sur master est déployable
  - Chaque branche doit avoir un nom significatif
  - Commit et push réguliers
  - Pull request à la fin d'une feature
  - Déployer directement après merge



Les workflows Git

GitFlow

- **Pour les contextes d'équipes grandissantes, de méthode agile et de livraison continue**
- **Concept**
  - Une branche pour la production : **master**
  - Une branche pour le développement : **develop**
  - Des branches pour la préparation des mises en production et la gestion des versions : **release**
  - Des branches pour le dev des fonctionnalités : **feature**
  - Des branches pour les corrections en prod : **hotfix**
- **Principes**
  - Tout ce qui est sur master est déployable
  - Qualification continue sur la develop ou les features
  - Qualification totale/recette sur les branches de release

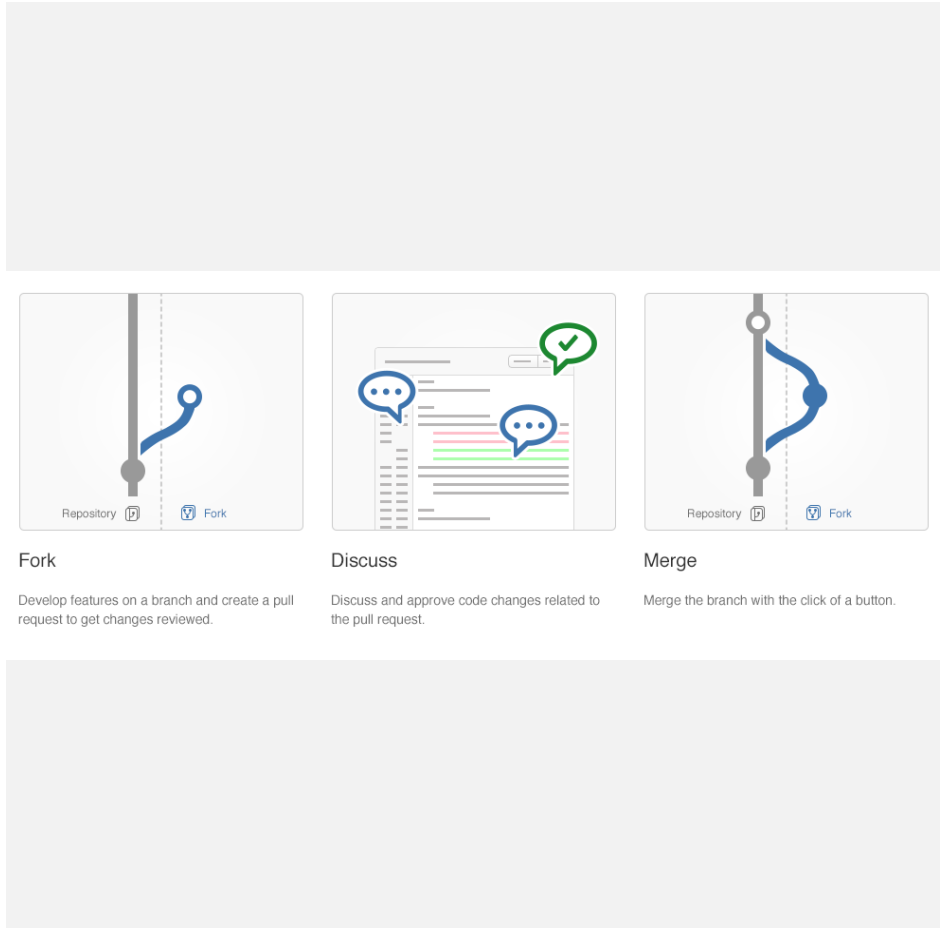


sopra steria

Les workflows Git

Fork & Merge

- **Idéal pour l'open-source afin que n'importe qui puisse contribuer à un projet sans le corrompre**
- **Plus complexe à mettre en place et à maintenir**
- **Concept**
  - Aucun travaille sur le dépôt principal
  - Fork du dépôt distant (git clone vers un autre dépôt distant)
  - Travail sur le fork qui peut régulièrement récupérer les modifications du dépôt principal
  - Pull request du fork à la fin avec discussions et merge par les maintainers du dépôt principal
- **Principe**
  - Tout ce qui est sur le dépôt principal est déployable et revu de tous



sopra steria

04

Les principes de GitFlow

VOYONS TOUT CELA PLUS EN DETAILS

Les principes de GitFlow

GitFlow c'est quoi ?

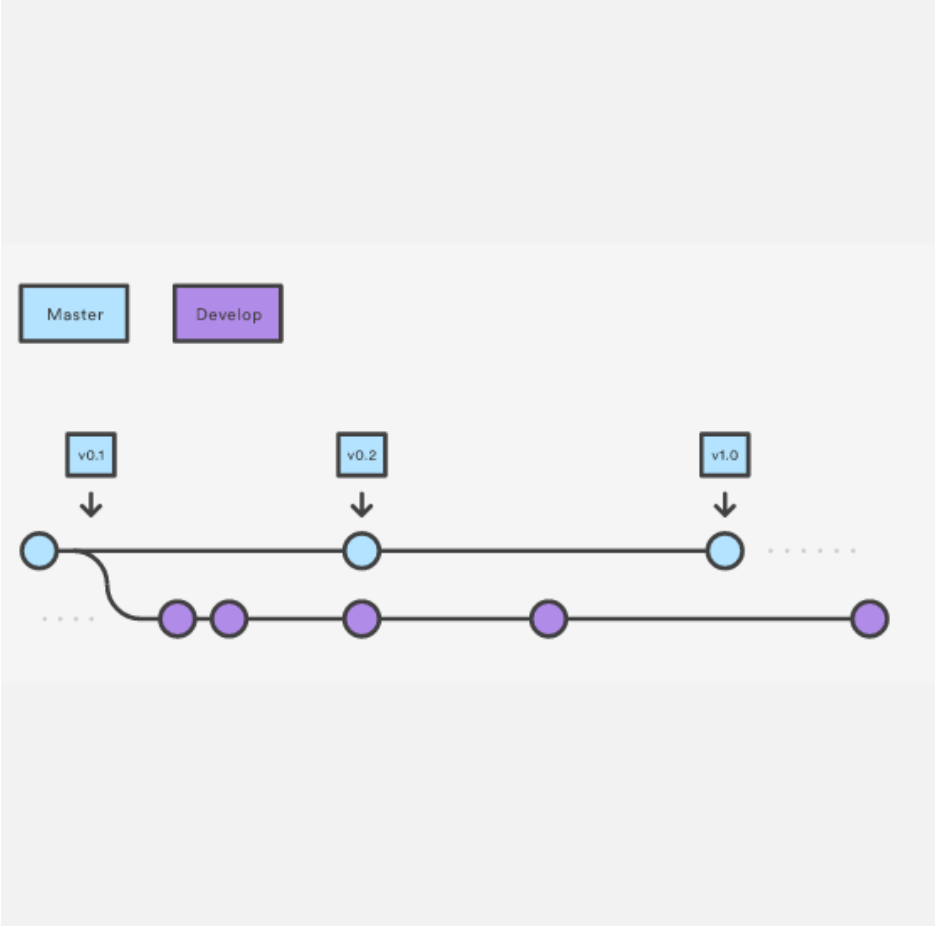
- **GitFlow se revendique seulement de définir quelle branche créer, à quel moment et comment les fusionner entre elles**
- **GitFlow prend en compte tous les bénéfices du Feature Branch Workflow …**
  - Pull request
  - Expérimentations isolées
  - Facilités de collaboration
- **… et y ajoute de nouveaux concepts pour plus de robustesse et de stabilité**
  - Définition des rôles spécifiques de chaque branche (comment et quand elle doivent interagir)
  - Préparation, maintenance et suivi de versions

- **A retenir :**
  - Basé sur le Feature Branch Workflow
  - Pull Request : Demande de fusion suite à un développement
  - Définit le rôle de chaque branche
  - Gestion des versions

Les principes de GitFlow

Master / Develop

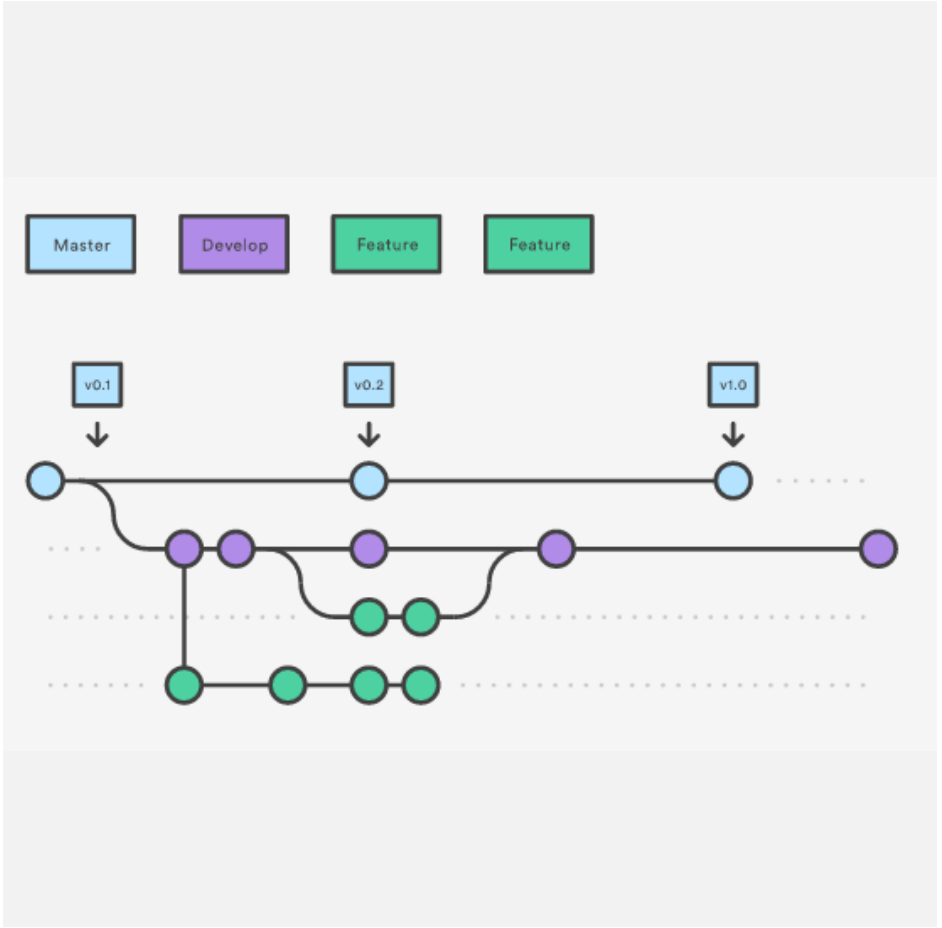
- **A la différence du Feature Branch Workflow, nous n'utilisons pas une mais deux branches principales**
  - Master qui contient la dernière release en date et correspond à la version pouvant être en production la plus récente
  - Develop qui contient les dernières fonctionnalités développées en intégralités
- **Cette organisation permet d'avoir la main sur la sortie des nouvelles versions (plus de merge en direct sur la master à la fin du développement d'une fonctionnalité**
- **Par convention, la branche master possède des tag pour chaque version mise en production**



Les principes de GitFlow

Feature & Bugfix

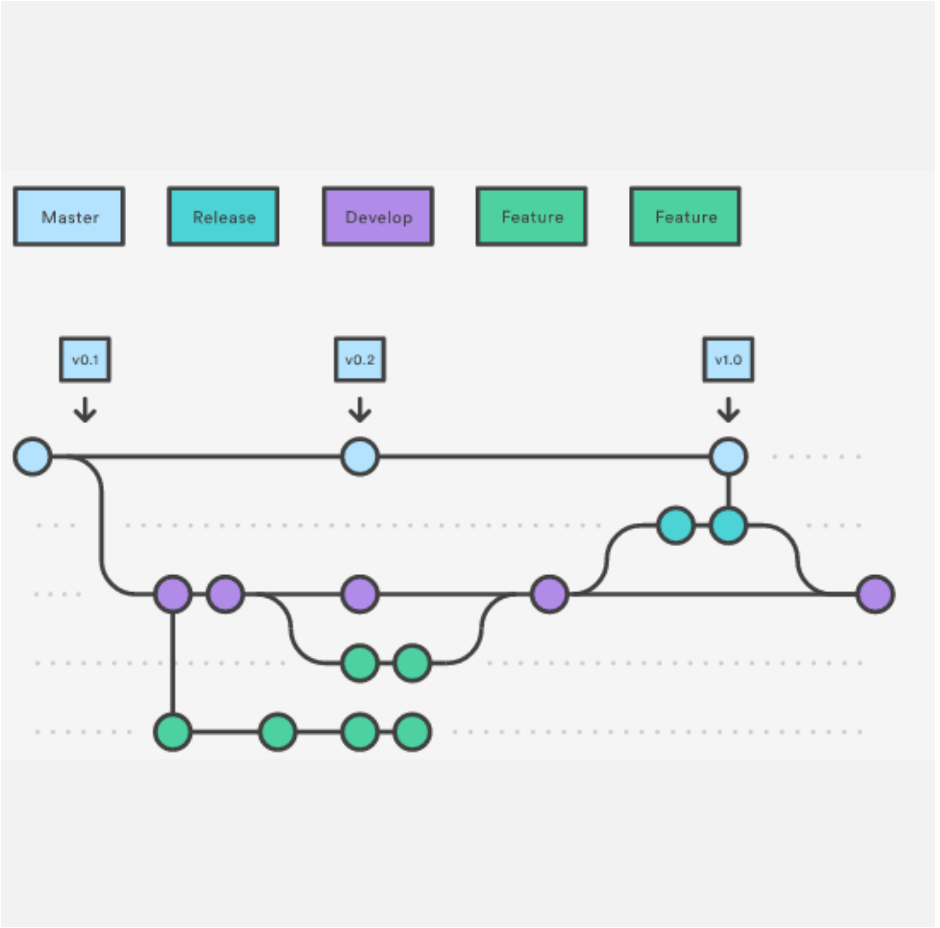
- Les branches feature correspondent au développement en cours de nouvelles fonctionnalités
- Les branches bugfix correspondent à la correction de bugs mineures qui impactent peu la production
- Ces branches commencent depuis la branche develop et seront fusionnées dans cette dernière lorsque le développement sera achevé
- Après leur fusion, elles n'ont plus d'utilité et peuvent être supprimées (la branche develop conservant l'historique suite à la fusion)
- Les fusions des branches feature dans la branche develop passent par des pull request ou merge request selon les outils (afin de permettre une revue de code des modifications)



Les principes de GitFlow

Release

- Les branches release correspondent aux versions de l'application en préparation pour une mise en production future
- Une release démarre de la branche develop et sera fusionnée dans la branche master ainsi que dans la branche develop (si correction de bug suite à qualification)
- Une branche release peut être supprimée à partir du moment qu'on ne maintient plus une version de l'application (l'historique étant conservé dans la branche master et la branche develop)



Les principes de GitFlow

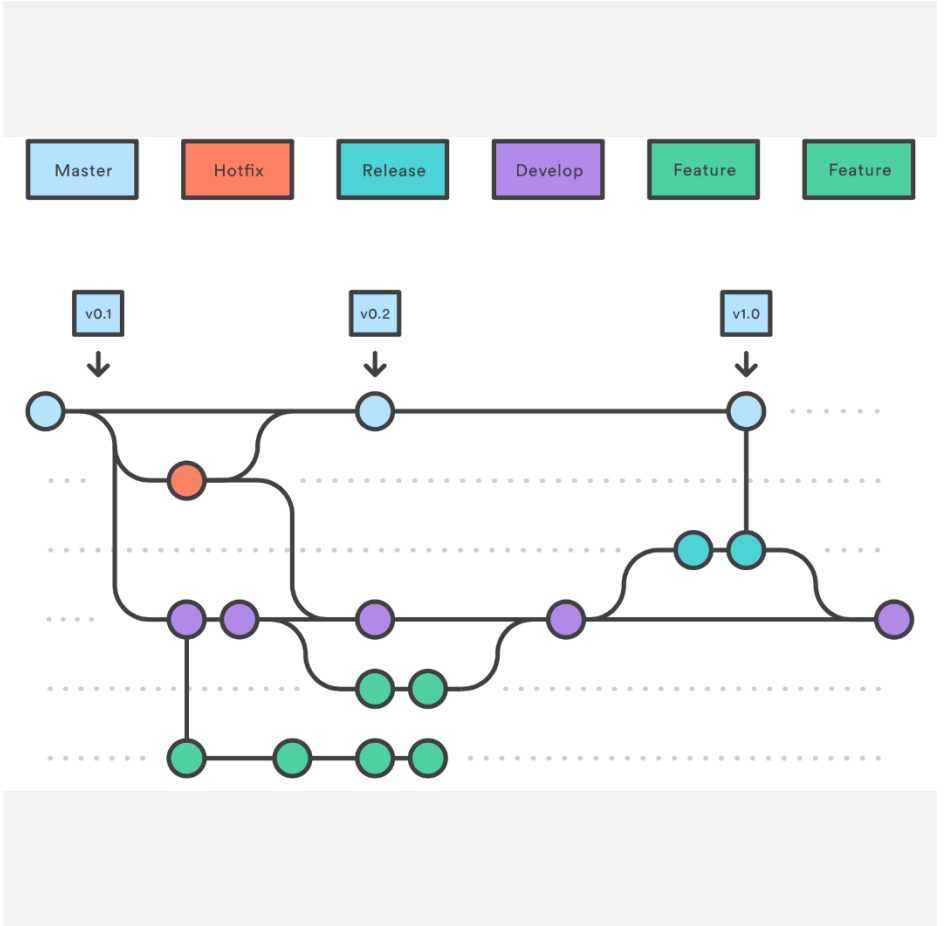
Support

- **Les branches support correspondent aux branches de maintenance d'anciennes versions de l'application**
- **Elles commencent d'une branche de release ou de la branche master et ne sont jamais fusionnées**
- **Leur nommage correspond au nommage des versions : support/{major_version}.{minor_version}.0**

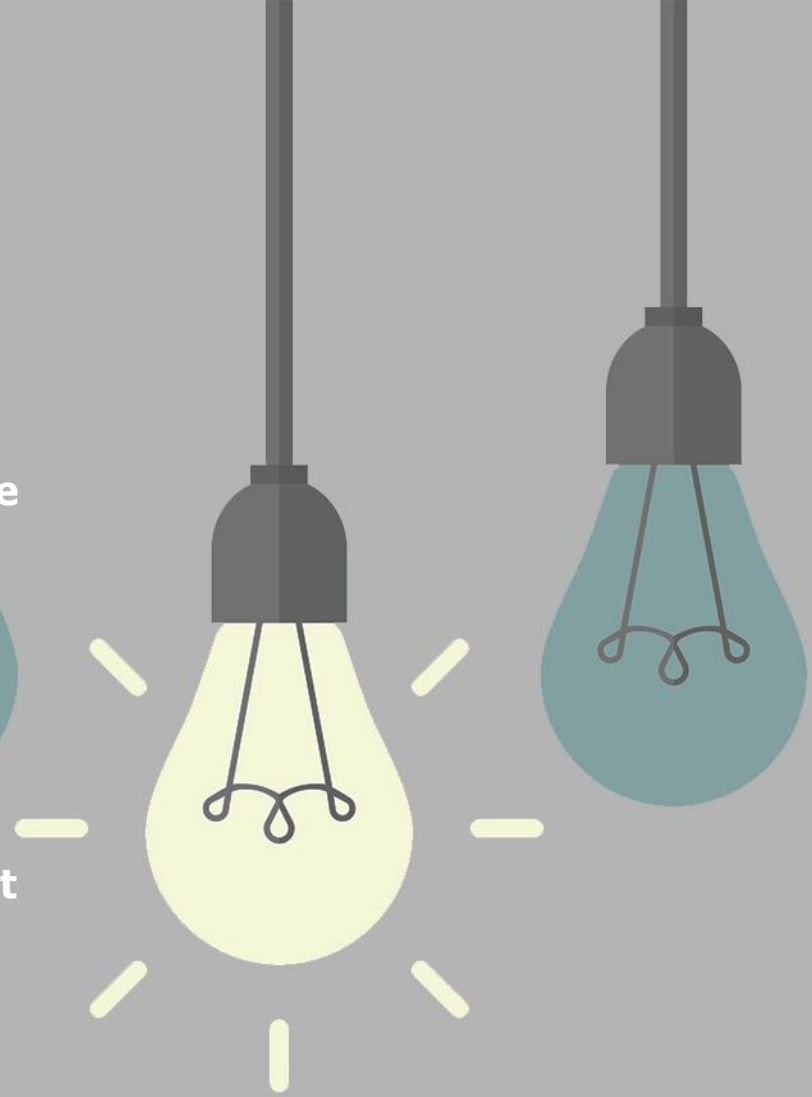
Les principes de GitFlow

Hotfix

- Les branches hotfix permettent d'opérer un correctif en production sur la dernière version déployable
- Ces branches partent de la branche master et seront fusionnées dans la branche master ainsi que dans la branche develop (afin de faire remonter les corrections dans la branche de développement et éviter les conflits lors de la prochaine release)
- Elle n'ont de durée de vie que celle de la correction. Elle peuvent être supprimées directement après la fusion



Qu’avons-nous appris ?



01 GitLab est une plate-forme Git server

02 GitLab permet la gestion des tâches, des revues de code, du paramétrage des droits sur le dépôt, …

03 Un workflow Git permet de normaliser les pratiques et d’éviter de ne plus s’y retrouver

04 Il existe des workflow adaptés à des besoins diverses : Feature Workflow, GitFlow, …

05 GitFlow donne un rôle à chaque branche et permet un meilleur suivi des versions

06 Les branches de GitFlow sont : develop, master, feature/, bugfix/, release/, hotfix/ et support/

Lien et documents utiles

- **https://git-scm.com/ pour toute les commandes de Git**
- **https://docs.gitlab.com/ pour tout ce qui concerne GitLab**
- **https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow pour vous poser les bonnes questions sur GitFlow**

- **La Cheat Sheet Git pour avoir les commandes Git toujours à portée de main**
- **La Cheat Sheet GitFlow qui est son équivalent GitFlow**

- **Des outils :**
  - Git graph sous Visual Studio Code
  - Git UI / bash sous Windows
  - GitKraken (sous licence pour les pro) / Git Sourcetree (Opensource)