

Gestion de projet logiciel avec Jira

Sommaire

1. Les outils	1
1.1. Git	1
1.2. Atlassian : Jira, Bitbucket,	2
1.3. GitHub	2
2. Le projet BTS SNIR à LaSalle Avignon	3
2.1. Processus de développement logiciel	3
2.2. Le développement itératif	4
2.3. Le développement incrémental	4
2.4. Versions	4
2.5. Planification	5
2.6. Kanban	5
2.7. Le développement collaboratif	10
2.7.1. Pull Request et Révision de code	10
2.7.2. Gitflow	11
2.7.3. Branche de suivi	13
2.8. Cycle de travail	14
2.9. Jira et GitHub	16
2.10. Jira et Bitbucket	36

Thierry Vaira - <tvaira@free.fr> - version v0.2 - 23/08/2021 - btssn-lasalle84.github.io

Objectif : développer un projet logiciel BTS SNIR avec Jira

1. Les outils

1.1. Git

Git est un logiciel de **gestion de versions décentralisé** (DVCS). C'est un logiciel libre créé par **Linus Torvalds** en 2005. Il s'agit maintenant du logiciel de gestion de versions le plus populaire devant **Subversion** (**svn**) qu'il a remplacé avantageusement.



Site officiel : <https://git-scm.com/>

Ressources Git :

- [Manuel de référence](#)
- [Livre Pro Git en français](#)
- [Livre Git Community Book en français](#)
- [Wikilivre Git en français](#)

1.2. Atlassian : Jira, Bitbucket, ...

À l'origine, [Jira](#) est un système de suivi de *bugs* et de gestion des incidents (*tickets*). Il est maintenant un système de gestion de projets développé par **Atlassian**.



[Atlassian](#) est un éditeur de logiciels, basé en Australie, qui développe des produits pour la gestion de développement et de projets. Ses logiciels les plus connus sont Jira, Confluence, Bitbucket et Trello :

- [Jira Software](#) constitue la plate-forme centrale pour les phases de programmation, de collaboration et de livraison. Jira Software est gratuit jusqu'à 10 utilisateurs, avec 2 Go de stockage.



- [Confluence](#) est un logiciel de wiki, utilisé comme logiciel de travail collaboratif.
- [Bitbucket](#) est un service web d'hébergement et de gestion de développement logiciel utilisant le logiciel de gestion de versions [Git](#). Bitbucket est gratuit pour les particuliers et les petites équipes comptant jusqu'à 5 utilisateurs, avec des référentiels publics et privés illimités.



- [Trello](#) est un outil de gestion de projet en ligne, inspiré par la méthode Kanban de Toyota.

Ressources Atlassian :

- [Les guides Jira](#)
- [À quoi sert Jira ?](#)
- [Découvrez Agile grâce à Jira](#)
- [Tutoriels](#)

1.3. GitHub

[GitHub](#) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git.



Site officiel : <https://github.com/>

Ressources GitHub :

- [Git Handbook sur Github](#)
- [Collaborating with pull requests](#)

2. Le projet BTS SNIR à LaSalle Avignon

Un projet (ou mini-projet) BTS SNIR sera mené avec :

- un [processus de développement itératif](#) et [incrémental](#)
- la méthode [Kanban](#) dans [Jira](#)
- le gestionnaire de versions [Git](#) (et le *workflow* [Gitflow](#)) dans un dépôt hébergé sur [GitHub](#) (ou Bitbucket) en lien avec [Jira](#)

2.1. Processus de développement logiciel

Un processus de développement décrit une méthode qui permet de construire, déployer et éventuellement maintenir un logiciel.

Un processus de développement définit une séquence d'étapes, partiellement ordonnées, qui permettent d'obtenir un système logiciel ou faire évoluer un système existant.

Exemples d'étapes :

- Exigences, Analyse, Conception, Mise en œuvre (implémentation), Test
- Besoin/Faisabilité, Élaboration, Fabrication, Transition/Test

On utilisera un développement [itératif](#) et [incrémental](#).

Liens :

- [Cycle de développement](#)
- [Processus Unifié](#)
- [Méthode agile : extreme programming \(XP\), Scrum](#)

2.2. Le développement itératif

Le développement itératif s'organise en une série de développement très courts de durée fixe nommée itérations.

Dans une itération, on répète les mêmes activités (de la spécification jusqu'au test).

Le résultat de chaque itération est un système partiel exécutable, testé et intégré mais incomplet.

Une nouvelle itération écrase la précédente.

Le résultat d'une itération n'est pas un prototype expérimental ou « jetable ».

Le projet ou mini-projet sera mené à son terme en plusieurs itérations successives (2 ou 3 maximum).

2.3. Le développement incrémental

Le développement incrémental consiste à réaliser successivement des éléments fonctionnels utilisables.

Un incrément est une avancée dans le développement en terme de fonctionnalités.

Dans un développement incrémental, on planifie donc par fonctionnalités.

Chaque développement s'ajoute et enrichit l'existant.

Chaque incrément produira une version.

2.4. Versions

Une [version d'un logiciel](#) correspond à un état donné de l'évolution d'un projet logiciel. Une version de logiciel est le plus souvent associée à une numérotation.

Il faut différencier les évolutions dans un logiciel :

- Les évolutions majeures apportent de nouvelles fonctionnalités, voire restructurent complètement l'application.
- Les évolutions mineures apportent principalement des corrections de bugs ou des ajouts de fonctionnalités secondaires.

Convention utilisée pour les numéros de versions (tags) **X.Y** :

- **X** le numéro de version majeur de l'application (**1** pour la première version/itération)
- **Y** le numéro de version mineur de l'application (**0** par défaut, ensuite incrémenté pour chaque correction)

2.5. Planification

La planification est essentielle dans le projet.

La plupart des méthodes encourage une planification itérative pilotée à la fois par les risques et par le client.

Cela signifie que les objectifs des premières itérations sont choisis afin de d'identifier les risques les plus importants et de construire les fonctionnalités visibles qui comptent le plus pour le client.

Pour chaque itération, on choisira des cas d'utilisation présentant ces trois qualités :

- significatifs du point de vue de l'architecture
- de grande valeur pour le client (les fonctionnalités qui comptent vraiment pour lui)
- à haut risque (technique)

On classera en distinguant trois niveaux de priorité (Haut, Moyen et Bas).

2.6. Kanban

Issue d'un mot japonais signifiant tableau, la méthode [Kanban](#) a vu le jour dans les usines Toyota au milieu du XXème siècle. Cette image de tableau vient de l'utilisation des porte-étiquettes permettant d'organiser le travail dans les usines.

Sa reprise pour la gestion de projet date de l'apparition des [méthodes dites agile](#).

La méthode Kanban tire sa force de sa simplicité. En effet, toujours dans un esprit agile, cette méthode simplifie au maximum le concept d'organisation des tâches.

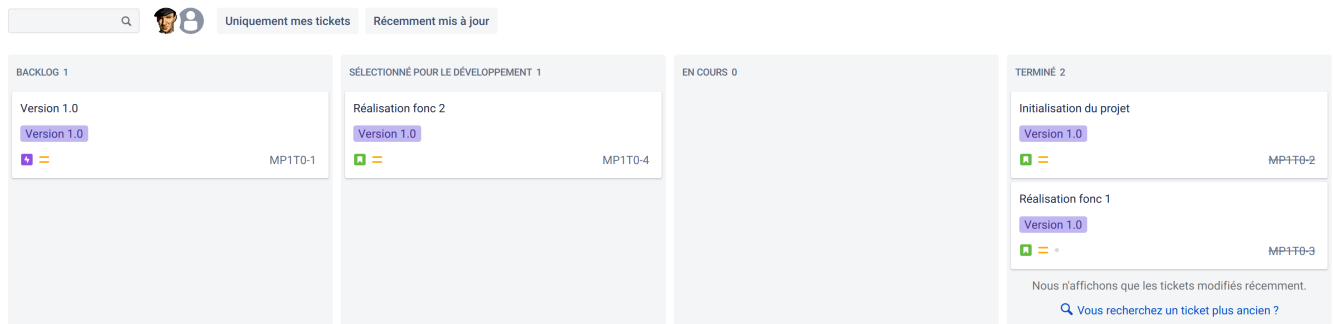
Elle repose sur quatre étapes principales aboutissant à un tableau de quatre colonnes :

- à faire (*backlog* ou *todo*) : la liste des tâches à effectuer pour l'itération ;
- prêt (*ready* ou *selected*) : la liste des tâches que vous souhaitez effectuer dans l'itération ;
- en cours (*in progress*) : les tâches en cours de réalisation (en général par développeur une à la fois, voire deux) ;
- terminé (*done*) : Les tâches terminées.



On peut ajouter, supprimer et/ou renommer les colonnes (Par exemple : à faire, en cours et terminé).

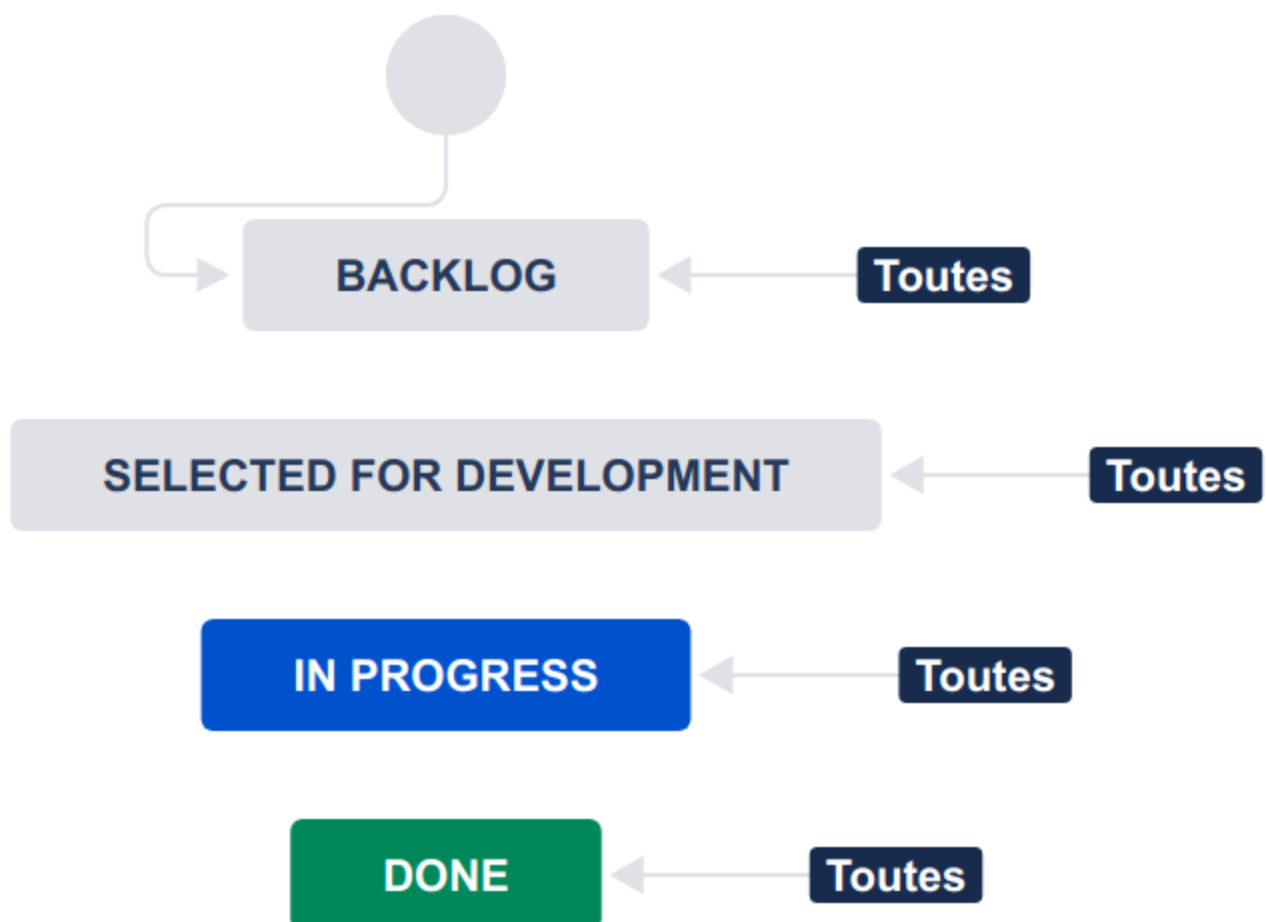
Dans Jira, un tableau Kanban permet à l'équipe de visualiser le flux de travail (*workflow*) :



Backlog

Un *backlog* contient des tickets en suspens sur lesquels une équipe devra travailler. Au démarrage d'une itération, on sélectionne les tickets dans le *backlog* pour les basculer dans "Sélectionné pour le développement" ou dans "En cours".

Le flux de travail (*workflow*) par défaut :



Il est possible d'ajouter des règles pour gérer automatiquement le flux de travail (*workflow*) :

MP1-Team0

Projet logiciel

Revenir au projet

Paramètres du projet

Détails

Personnes

Automation

Fonctionnalités

Résumé

Types de tickets

Disposition des tickets

Workflows

Écrans

Champs

Composants

Opsgenie

Autorisations

Sécurité des tickets

Vous faites partie d'un projet géré par l'entreprise

En savoir plus

Administration globale

Règles

Journal d'audit

Bibliothèque

Automation

Logiciels

Lorsque toutes les stories sont terminées → fermer l'epic

Lorsqu'une epic est terminée → fermer toutes les stories présentes

Lorsqu'un ticket est transitionné → l'assigner automatiquement

Fermer les tickets en double

Lorsque toutes les sous-tâches sont terminées → déplacer le parent vers « Terminé »

DevOps

Lorsqu'un parent est terminé → déplacer toutes les sous-tâches vers « Terminé »

Lorsqu'un ticket est résolu → commenter tous les tickets associés

Lier les tickets mentionnés dans les commentaires

Par exemple, pour faire le lien avec Git :

Automation

[Règles](#)
[Journal d'audit](#)
[Bibliothèque](#)

Filtrer les règles

Toutes les règles

Règles globales

Règles de projet

Nom

Quand un commit est effectué → puis déplacer le ticket vers « En cours »

Quand une branche est créée → puis déplacer le ticket vers « En cours »

Quand une pull request est mergée → puis déplacer le ticket vers « Terminé »

7

Kanban vs Scrum

Kanban et Scrum sont des *frameworks* Agile populaires auprès des développeurs de logiciels.

Il y a cependant des différences clés :



- Les sprints Scrum ont des dates de début et de fin, alors que Kanban est un processus continu.
- Dans Scrum, les rôles de l'équipe sont clairement définis (*Product Owner*, *Scrum Master*, équipe de développement), contrairement à Kanban où il n'y a pas de rôles formels. Les deux équipes sont auto-organisées.
- Un tableau Kanban est utilisé tout au long du cycle de vie d'un projet, alors qu'un tableau Scrum est nettoyé et recyclé après chaque sprint.
- Un tableau Scrum possède un nombre de tâches définies ainsi que des échéances strictes pour les effectuer.
- Les tableaux Kanban sont plus flexibles en termes de tâches et d'échéances. Les tâches peuvent être hiérarchisées à nouveau, réassignées ou mises à jour si besoin.

Les tâches (un **ticket** dans Jira) sont représentées visuellement sur le tableau Kanban, ce qui permet à l'équipe de suivre l'état du travail à tout moment. Les colonnes du tableau représentent chaque étape du *workflow*, des tâches à faire à celles qui sont terminées.

Les tickets Jira, également appelés « tâches », suivent chaque travail qui doit passer par les différentes étapes du workflow jusqu'à son achèvement. Les tickets sont des éléments de **travail individuel** qui sont assignés aux membres de l'équipe.

Les différents types de ticket dans Jira :

Type de ticket	Description
 Story	Une fonctionnalité exprimée sous la forme d'un objectif utilisateur.
 Bug	Un problème ou une erreur.
 Epic	Une collection de bugs, stories et tâches connexes.
 Tâche	Une tâche distincte.
 Sous-tâche SOUS-TÂCHE	Une petite unité de travail qui fait partie d'une tâche plus importante.

Epics

Une « epic » (épopée) est un vaste ensemble de tâches qui peuvent être subdivisées en plus petites unités. Ces unités, appelées « stories » (histoire) ou « user stories », représentent les exigences ou besoins du point de vue de l'utilisateur.

Il est possible de créer des *epics* de trois manières dans Jira : la feuille de route, le *backlog* et le bouton global "Créer un ticket". Après avoir créé une *epic*, on peut y ajouter des *stories* ou des

tickets enfant.

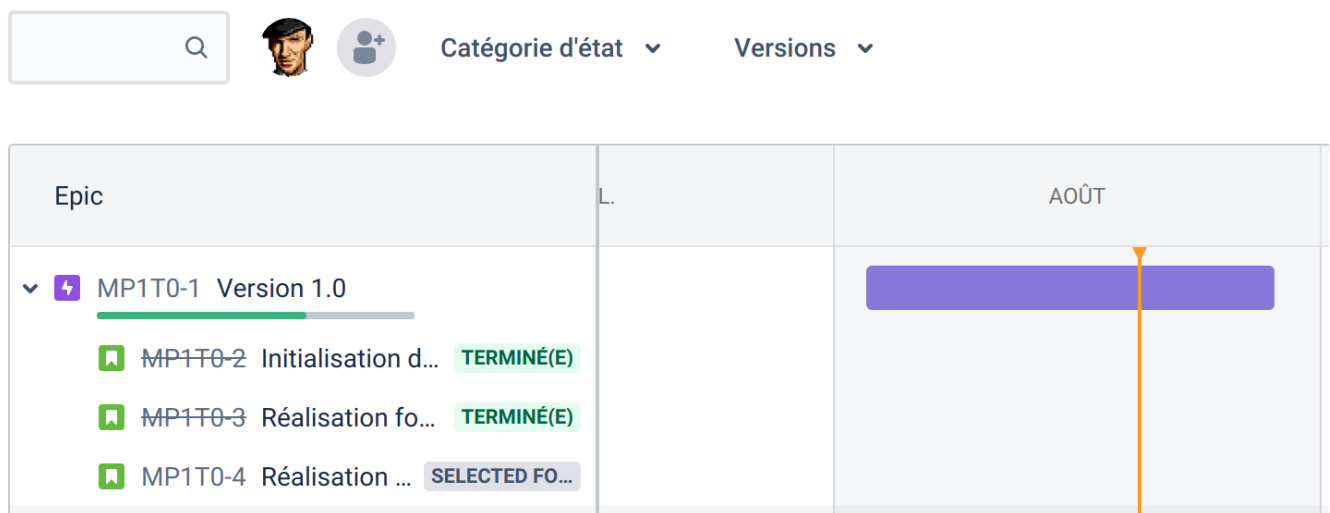


Les *stories*, les bugs et les tâches décrivent un bloc de travail, alors que les *epics* sont utilisées pour décrire un groupe de tickets.

Feuilles de route

Les feuilles de route créent une représentation visuelle de toutes les *epics* sur lesquelles une équipe travaille. Une feuille de route est une source de référence partagée qui décrit la vision, l'orientation, les priorités et l'avancement d'un produit ou d'une équipe au fil du temps.

Feuille de route



Versions

Dans Jira, les versions représentent des points dans le temps pour un projet. Elles aident à organiser les tâches grâce à des étapes importantes à suivre. Il est possible d'assigner les tickets à une version spécifique et organiser les itérations en fonction des tâches réalisées dans cette version. Les noms de version correspondent généralement à des chiffres, par exemple, **1.0** ou **2.1.1**.

Versions

Version ↕	État	Avancement
1.0	NON LIVRÉE(S)	

1

Avertissements

4

Tickets dans la version

2

Tickets terminés

0

Tickets en cours

2

Tickets à faire

1-2 sur 2

[Afficher dans le navigateur de tickets](#)

Pr T Clé Résumé

MP1T0-2 Initialisation du projet

Responsable

État

Développement



Non assigné

TERMINÉ(E)

4 commits

MP1T0-3 Réalisation fonc 1



Non assigné

TERMINÉ(E)

FUSIONNÉE

2.7. Le développement collaboratif

2.7.1. Pull Request et Révision de code

Les *Pull Requests* sont une fonctionnalité facilitant la collaboration des développeurs sur un projet. Cela permet à un développeur d'informer les membres de l'équipe qu'il a terminé un « travail » (une fonctionnalité, une version livrable, un correctif, ...) et de proposer sa contribution au dépôt central.

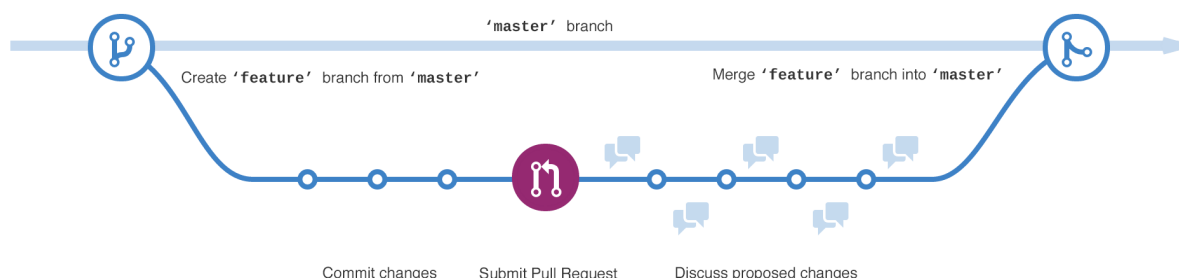


Pull Request peut être traduit par « Proposition de révision » (PR) : c'est-à-dire une demande de modification ou de contribution.

Le principe est le suivant :

- Une fois que sa branche de suivi est prête, le développeur crée ou ouvre (*Open*) une *Pull Request*.
- Tous les développeurs du projet seront informés du fait qu'ils doivent **réviser le code** puis le **fusionner** (*merge*) dans la branche principale (*main* ou *master*) ou dans une autre branche.

Pendant cette révision de code, les développeurs peuvent discuter de la fonctionnalité (commenter le code, poser des questions, ...) et proposer des adaptations de la fonctionnalité en publiant des *commits* de suivi.



Les *Pull Requests* offrent cette fonctionnalité dans une **interface Web** à côté des dépôts GitHub ou Bitbucket. Cette interface affiche une comparaison des changements, permet l'échange entre développeurs et fournit une méthode simple pour réaliser la fusion (*merge*) du code quand il est prêt.

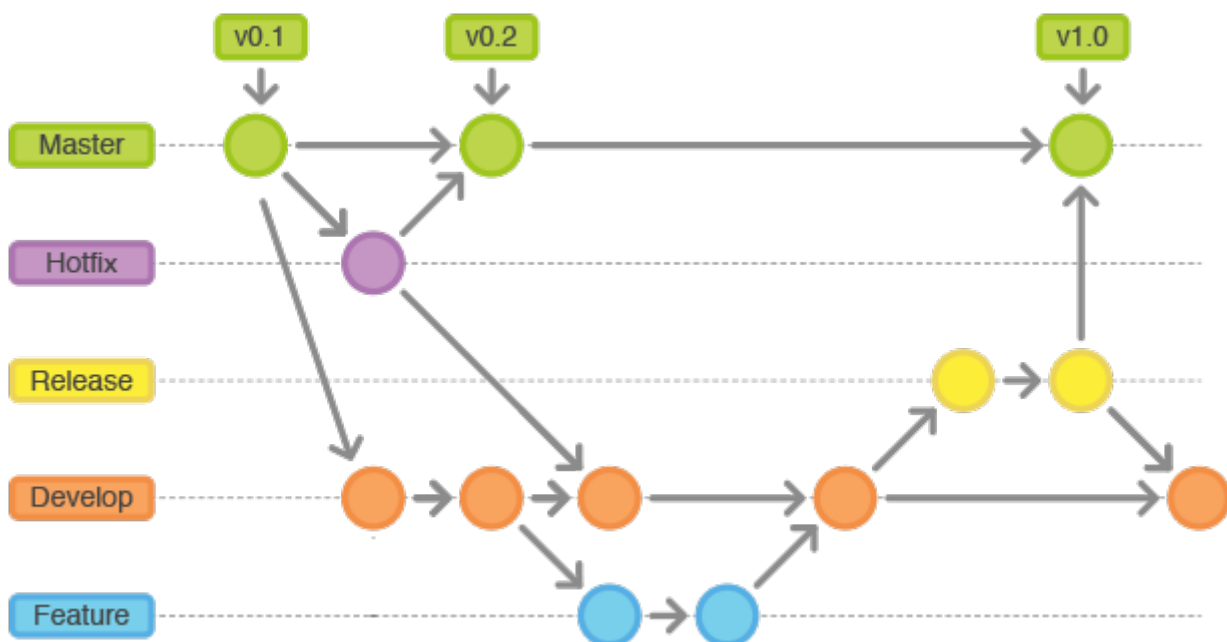
2.7.2. Gitflow



Un *workflow git* est une méthode, un processus de travail, une recette ou une recommandation sur la façon d'utiliser **git** pour accomplir un travail de manière cohérente et productive. Il n'existe pas de processus standardisé sur la façon d'interagir avec **git**. Il est important de s'assurer que l'équipe de projet est d'accord sur la façon dont le flux de modifications sera appliqué. Un *workflow git* doit donc être défini.

Lien : [Comparaison des workflow git](#)

Le *workflow Gitflow* définit un modèle de branchement strict conçu autour de la version du projet. Ce *workflow* n'ajoute pas de nouveaux concepts ou commandes. Gitflow permet de gérer les bugs (*issues*), les nouvelles fonctionnalités (*features*) et les versions (*releases*) en attribuant des rôles très spécifiques à différentes branches et définit comment et quand elles doivent interagir.



Les rôles des branches sont les suivants :

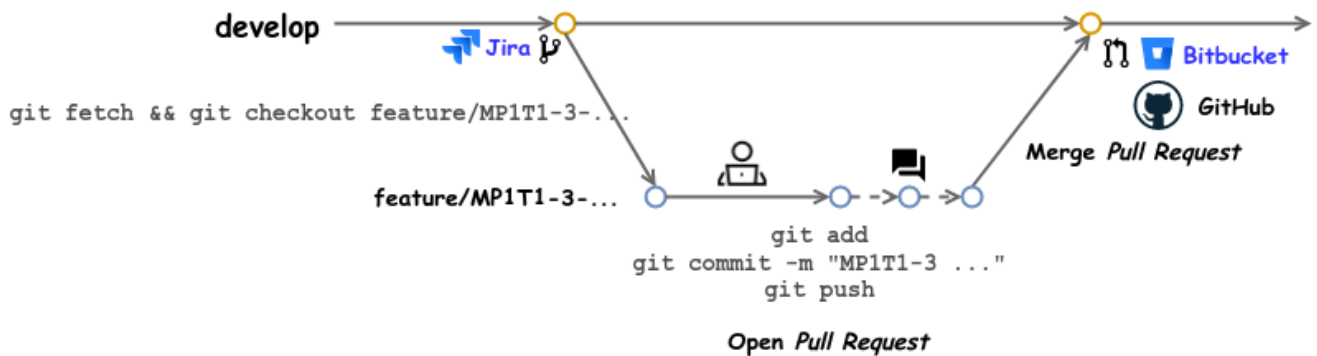
- pour les branches permanentes :
 - La branche **master** stocke l'historique des versions officielles. Tous les *commits* de cette branche sont étiquetés avec un numéro de version (*tags*).
 - La branche **develop** est créée à partir de la branche **master**. Elle sert de branche d'intégration pour les fonctionnalités. Cette branche contiendra l'historique complet du projet.
- pour les branches temporaires :
 - Les branches **features-xxxx** permettent de travailler sur des nouvelles fonctionnalités. Elles sont créées directement à partir de la branche **develop** et une fois le travail fini, fusionnées vers la branche **develop**.
 - Les branches **release-xxxx** permettent de travailler sur une livraison (généralement des tâches dédiées à la documentation). On les crée à partir de **develop** puis on les fusionne dans **master** en leur attribuant un numéro de version (*tag*).

- Les branches **hotfix-xxxx** permettent de publier rapidement (*hot*) une correction (*fix*) depuis la branche **master**. Ces branches seront ensuite fusionnées vers la branche **master** et **develop**.

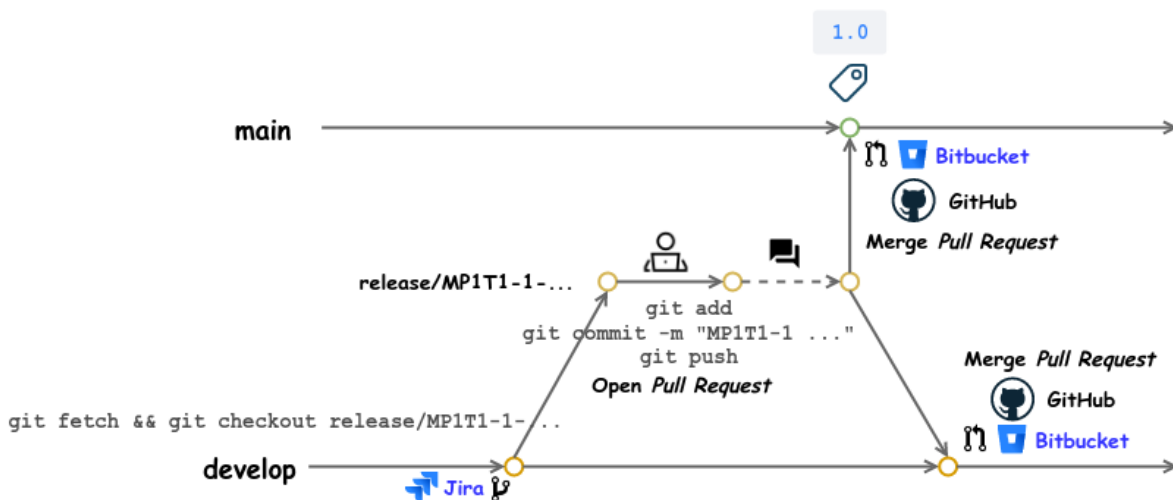


En projet BTS SN, les branches (*feature*, *release* et *hotfix*) seront créées dans Jira à partir d'un ticket. Les fusions seront réalisées lors d'une revue de code en utilisant les *Pull Requests* dans GitHub ou Bitbucket.

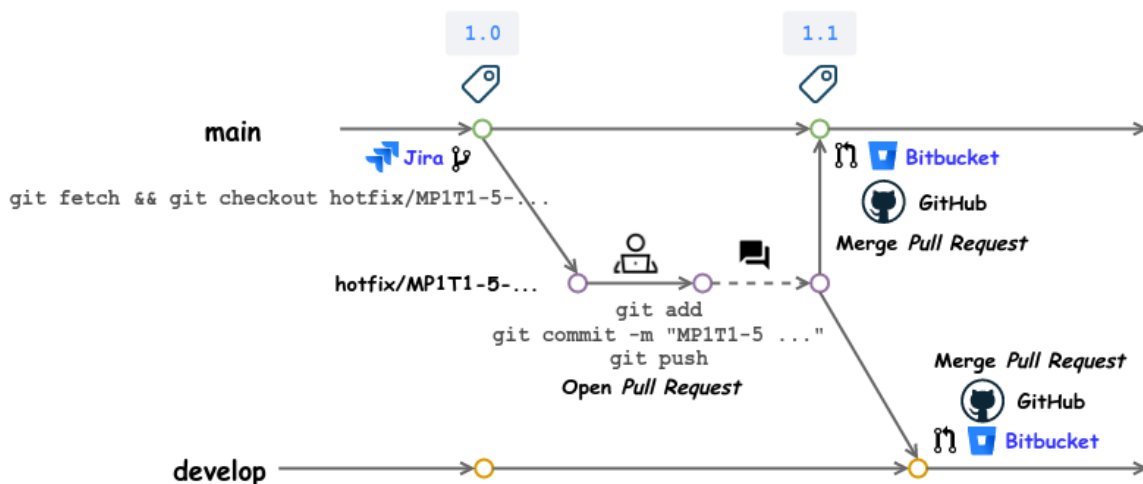
Réalisation d'une fonctionnalité :



Réalisation d'une *release* :



Correction d'un *bug* :





Une branche représente une ligne de développement indépendante. Lorsqu'elle désigne un travail bien identifié du projet (une fonctionnalité, une *release* ou un correctif), il est préférable (obligatoire) que cela reste visible dans le graphe d'historique, même lorsque la branche est supprimée. Pour éviter que Git utilise par défaut une avance rapide (*Fast Forward*) si c'est possible, il faudra réaliser un *commit* de fusion avec l'option `--no-ff`.

2.7.3. Branche de suivi

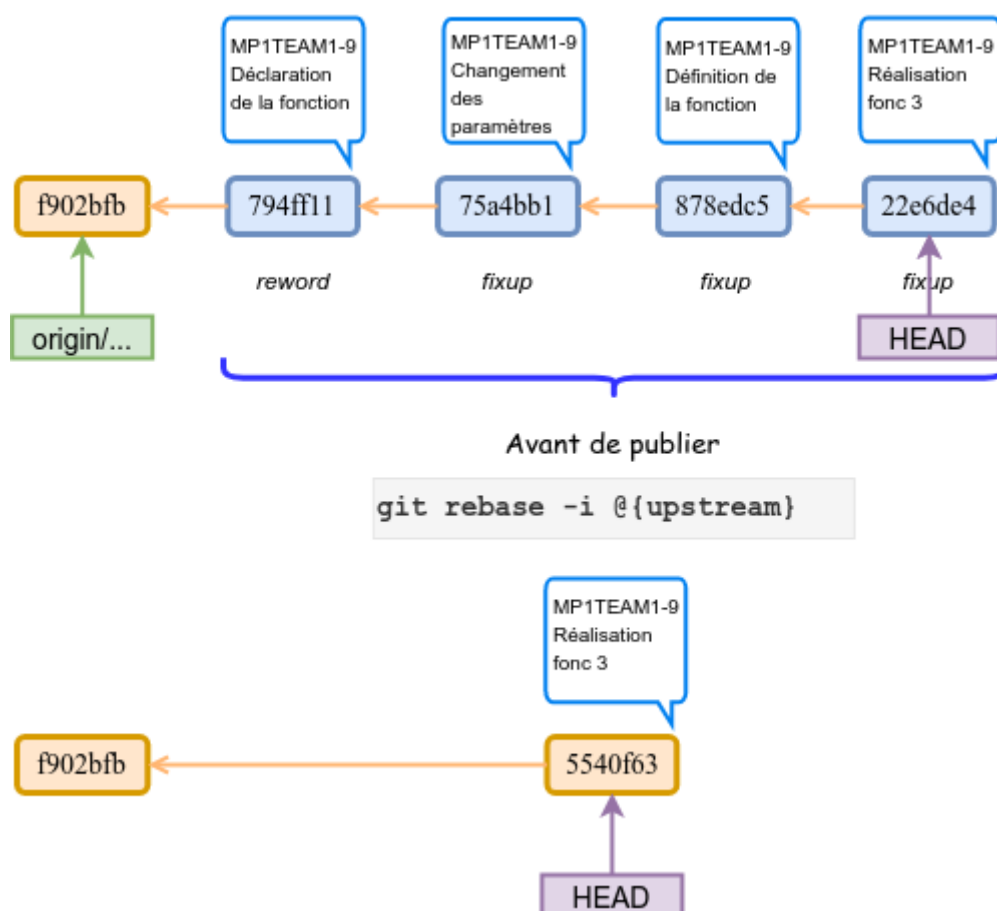
Une branche de suivi (*tracking branch*) est une branche locale qui est en relation directe avec une branche distante (*upstream branch*).

1 . Nettoyer son historique local avant de publier

Avant de faire un `git push` sur une branche de suivi, il faut eut être nettoyer son historique local (une série de *commits* dans la branche) afin de pouvoir proposer quelque chose de propre et d'utilisable. Avant de publier la branche, il est conseillé d'effectuer une rebase interactif avec `git rebase -i`. On a alors une totale liberté pour nettoyer, réécrire, annuler, regrouper les *commits* locaux avant de les partager (`git push`) sur le dépôt distant.

Sur la branche actuelle depuis la dernière synchronisation : `git rebase -i @{upstream}` (ou `git rebase -i origin/feature` ou `git rebase -i HEAD~n`).

Par exemple :





Lorsque l'on développe seul, une branche de suivi peut servir de sauvegarde sur un dépôt distant. Dans le cadre d'un travail collaboratif, cela devient une branche de partage.

2 . Travailler à plusieurs sur une branche de fonctionnalité

Il est possible que le `git push` soit refusé en raison d'une branche de suivi obsolète (un travail a été poussé entre-temps) : entre la dernière synchronisation entrante (`git pull`) et le moment où on souhaite effectuer un `git push`, un autre développeur a publié des changements (des *commits*). La branche distante (par exemple `origin/feature`) est donc maintenant plus avancée que sa copie locale.

Un `git pull` provoquerait une fusion avec une divergence mais on souhaite conserver un historique linéaire au sein d'une branche : car ce n'est réalité qu'un problème de séquençement dans le travail sur la branche.

On va demander à `git pull` de faire un *rebase* au lieu d'une fusion (*merge*) en utilisant `git pull --rebase`.



La commande `git rebase` permet de changer la « base » d'une branche, c'est-à-dire son *commit* d'origine. Elle rejoue une série de *commits* sur une nouvelle base.

2.8. Cycle de travail

Cycle de travail

- Cloner le dépôt

À chaque itération (produisant une nouvelle version) :

- Créer une version dans Jira
- Mettre à jour la Feuille de route dans Jira
 - Créer une *Epic* pour l'itération et l'associer à la version
 - Créer les tâches (tickets enfants) dans cette *Epic* et les associer à la version
- Pour chaque fonctionnalité de l'itération :
 - Sélectionner une tâche dans Jira
 - Créer une branche *feature* à partir de *develop*
 - Implémenter la fonctionnalité sur la branche
 - Créer une *Pull Request* vers *develop* lorsque la fonctionnalité est prête pour validation
 - Approuver la *Pull Request* si la fonctionnalité passe la révision de code (notamment respect des règles de codage et bonnes pratiques)
 - Fusionner la *Pull Request* dans *develop* avec l'option *--no-ff*
 - Supprimer la branche locale et distante
 - Vérifier que la tâche est marquée "Terminée" dans Jira
- Finaliser une version :
 - Sélectionner l'*Epic* pour l'itération dans Jira
 - Créer une branche *release* à partir de *develop*
 - Finaliser la *release* sur la branche
 - Créer une *Pull Request* vers *main* lorsque la *release* est prête pour validation
 - Approuver la *Pull Request* si la *release* passe la révision de code (notamment respect des règles de codage et bonnes pratiques)
 - Fusionner la *Pull Request* dans *main* avec l'option *--no-ff*
 - Créer un *tag* et le publier sur la branche *main*
 - Créer une *Pull Request* vers *develop*
 - Fusionner la *Pull Request* dans *develop* avec l'option *--no-ff*
 - Supprimer la branche locale et distante
 - Vérifier que l'*Epic* est marquée "Terminée" dans Jira
 - Livrer la version dans Jira

- Corriger un défaut dans une version :
 - Créer un ticket *bug* dans l'*Epic* concernée et l'associer à une nouvelle version **X.Y.Z** dans Jira
 - Sélectionner le ticket *bug* dans Jira
 - Créer une branche **hotfix** à partir de **main**
 - Implémenter la correction sur la branche
 - Créer une *Pull Request* vers **main** lorsque le correctif est prêt pour validation
 - Approuver la *Pull Request* si le correctif passe la révision de code
 - Fusionner la *Pull Request* dans **main** avec l'option **--no-ff**
 - Créer un *tag* et le publier sur la branche **main**
 - Créer une *Pull Request* vers **develop**
 - Fusionner la *Pull Request* dans **develop** avec l'option **--no-ff**
 - Supprimer la branche locale et distante
 - Vérifier que l'*bug* est marqué "Terminé" dans Jira
 - Livrer la version corrigée

2.9. Jira et GitHub

Cette partie décrit un exemple de développement de projet avec Jira connecté à un dépôt GitHub. Il reprend le cycle de travail défini précédemment.

Au démarrage, il faut **cloner le dépôt** GitHub :

GitHub :

```
$ git clone git@github.com:btssn-lasalle84/mp1-teamX.git  
  
$ cd mp1-teamX/
```

À chaque **itération** (produisant une nouvelle version) :

- **Création d'une version dans Jira**

Versions



Commencer à utiliser une version

Les versions vous aident à préparer et à planifier vos livraisons de projets. Ajoutez une version pour commencer à regrouper et à livrer votre travail.

[Créer une version](#)

[En savoir plus](#)

Créer une version

Nom *

1.0

Date de début

2/18/1993



Date de livraison

2/18/1993



Description

Enregistrer

Annuler

Projets / miniprojet1-team1

Versions



Non livrée(s) ▼

Version ↕

État

Avancement

1.0

NON LIVRÉE(S)

Aucun ticket

- Mise à jour de la Feuille de route
 - Création d'une *Epic* (l'itération) : Réalisation de la version x.x

Feuille de route



Catégorie d'état ▾

Epic

 Version 1.0

Il faut l'associer à la version créée :

Feuille de route

 Partager

 Exporter



Catégorie d'état ▾

Versions ▾

Aujourd'hui

 Afficher les paramètres

Epic	L.	
 MP1T1-1 Version 1.0		
+ Créer Epic		

 MP1T1-1

 1   ... 

Étiquettes

Aucun

Priorité

 Medium

Epic Name

Version 1.0

Automation

 Rule executions

Plus de champs

^

Story Points

Aucun

Estimation originale

0min

Suivi temporel

Aucun temps enregistré

Composants

Aucun

Versions corrigées

1.0

- **Création des tâches** (tickets enfants) pour chaque fonctionnalité associée à la version

Projets / miniprojet1-team1 / Tableau MP1T1

Feuille de route







Catégorie d'état 

Epic	L.
 MP1T1-1 Version 1.0	 Créer un ticket enfant
+ Créer Epic	

Par exemple :

Feuille de route



Catégorie d'état ▾

Epic

▼  MP1T1-1 Version 1.0

 MP1T1-2 Initialisation du projet

BACKLOG

 MP1T1-3 Réalisation fonc 1

BACKLOG

 MP1T1-4 Réalisation fonc 2

BACKLOG

- Sélection d'une tâche puis création de la branche (Type **feature** à partir de **develop**)

MP1T0-1 /  MP1T0-2

Initialisation du projet

 Joindre  Créer une sous-tâche  Associer un ticket ▾ ...

Description

Ajouter une description...

Activité

Afficher : Tout Commentaires Historique Journal du travail Git Diff Les plus récents d'abord ↕



Conseil de pro : appuyez sur  pour commenter

Donnez votre avis  1   ... ✕

← GitHub

Create Branches PRs Tags

Repository *

btssn-lasalle84/mp1-team0 ▾

Base branch *

 develop ▾

Branch name *

feature/MP1T0-2-initialisation-du-projet

 Branch Name Template Create branch

On récupère (copier) la commande Git :



Branch **feature/MP1T0-2-initialisation-du-projet** has been successfully created.

Use this command to check out your branch:

```
git fetch && git checkout feature/MP1T0-2-initialis
```



- **Implémentation de la fonctionnalité sur la branche**

```
$ git fetch && git checkout feature/MP1T0-2-initialisation-du-projet
Depuis github.com:btssn-lasalle84/mp1-team0
* [nouvelle branche] feature/MP1T0-2-initialisation-du-projet ->
origin/feature/MP1T0-2-initialisation-du-projet
La branche 'feature/MP1T0-2-initialisation-du-projet' est paramétrée pour suivre la
branche distante 'feature/MP1T0-2-initialisation-du-projet' depuis 'origin'.
Basculement sur la nouvelle branche 'feature/MP1T0-2-initialisation-du-projet'

$ vim projet.txt
fonc 1 : vide
fonc 2 : vide
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide

$ git add projet.txt
```

Au minimum, il faut inclure la clé du ticket au début du message de *commit* pour faire lien avec Jira :

```
$ git commit -m "MP1T0-2 Ajout du fichier projet.txt"
```



Avant de faire le **git push**, il est possible de ré-organiser ses *commits* locaux avec **git rebase -i @{upstream}**.

Dans le cadre d'un travail collaboratif, on "publie" la branche sur le dépôt distant :

```
$ git push
Décompte des objets: 3, fait.
Delta compression using up to 12 threads.
Compression des objets: 100% (3/3), fait.
Écriture des objets: 100% (3/3), 322 bytes | 322.00 KiB/s, fait.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:btssn-lasalle84/mp1-team0.git
 5401f1a..9be3d62  feature/MP1T0-2-initialisation-du-projet -> feature/MP1T0-2-
initialisation-du-projet
```

- **Création d'une *Pull Request*** lorsque la fonctionnalité est prête pour validation

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: develop ← compare: feature/MP1T0-3-realisation-fo... ✓ Able to merge. These branches can be automatically merged.

MP1T0-3 Fonc 1 ok

Write Preview H B I

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone



Il est possible de continuer à travailler sur la branche. Dans le cadre d'un travail collaboratif, il sera peut-être nécessaire de mettre à jour la branche avec un `git pull --rebase`.

Lorsque la *Pull Request* sera approuvée, on pourra faire la fusion (avec l'option `--no-ff`) :

MP1T0-3 Fonc 1 ok #3

 Open

tvaira wants to merge 1 commit into `develop` from `feature/MP1T0-3-realisation-fonc-1` 



Conversation 0



Commits 1



Checks 0



Files changed 1



tvaira commented 1 minute ago



No description provided.



MP1T0-3 Fonc 1 ok

2450bd8

Add more commits by pushing to the `feature/MP1T0-3-realisation-fonc-1` branch on `btssn-lasalle84/mp1-team0`.



Continuous integration has not been set up

[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



or view [command line instructions](#).

Merging via command line

If you do not want to use the merge button or an automatic merge cannot be performed, you can perform a manual merge on the command line. However, the following steps are not applicable if the base branch is protected.

HTTPS

SSH

Patch

`git@github.com:btssn-lasalle84/mp1-team0.git`



Step 1: From your project repository, bring in the changes and test.

```
git fetch origin
git checkout -b feature/MP1T0-3-realisation-fonc-1 origin/feature/MP1T0-3-realisation-fonc-1
git merge develop
```



Step 2: Merge the changes and update on GitHub.

```
git checkout develop
git merge --no-ff feature/MP1T0-3-realisation-fonc-1
git push origin develop
```



On obtient :

MP1T0-3 Fonc 1 ok #3

 Merged tvaira merged 1 commit into `develop` from `feature/MP1T0-3-realisation-fonc-1` 10 seconds ago

Conversation 0 Commits 1 Checks 0 Files changed 1

 tvaira commented 3 minutes ago

No description provided.

  MP1T0-3 Fonc 1 ok 2450bd8

  tvaira merged commit 750f1bb into `develop` 10 seconds ago [Revert](#)

 **Pull request successfully merged and closed**

You're all set—the `feature/MP1T0-3-re...` branch can be safely deleted. [Delete branch](#)

Une fois la fusion réalisée, on peut faire la mise à jour de la branche `develop` :

```
$ git checkout develop
$ git pull
```

Avec Gitflow, seules les branches `main` et `develop` sont permanentes. Les branches `feature` peuvent être supprimées :

```
$ git branch -d feature/MP1T0-2-initialisation-du-projet
$ git push origin --delete feature/MP1T0-2-initialisation-du-projet
```

Dans Jira, le ticket est passé automatiquement à l'état "Terminé" (sinon le faire manuellement) :

TERMINÉ 1

Réalisation fonc 1

Version 1.0



MP1T0-3

Nous n'affichons que les tickets modifiés récemment.

Vous recherchez un ticket plus ancien ?

Avec des informations sur le développement :

MP1T0-1 / MP1T0-3

Réalisation fonc 1

Joindre Créer une sous-tâche Associer un ticket

Description

Ajouter une description...

Activité

Afficher : Tout Commentaires Historique Journal du travail Git Diff Les plus récents d'abord



Ajouter un commentaire...

Conseil de pro : appuyez sur pour commenter

Donnez votre avis 1

Terminé(e)

✓ Terminé

Champs épinglés

Cliquez sur à côté des étiquettes de champ pour épingler.

Détails

Responsable Non assigné

Rapporteur Thierry VAIRA

Développement 1 branche
2 commits il y a 4 heures
1 pull request

Étiquettes Néant

Epic Link Version 1.0

Versions corrigées 1.0

Priorité Medium

Automation Rule executions

GitHub GitHub toolkit

- Finalisation d'une **release** puis création de la branche (Type **release** à partir de **develop**)

Lorsque toutes les tâches de l'itération sont terminées :

TERMINÉ 3

Initialisation du projet

Version 1.0



MP1T0-2

Réalisation fonc 1

Version 1.0



MP1T0-3

Réalisation fonc 2

Version 1.0



MP1T0-4

Nous n'affichons que les tickets modifiés récemment.

 [Vous recherchez un ticket plus ancien ?](#)

On sélectionne le ticket *Epic* (représentant l'itération réalisée) et on crée une branche **release** à partir de la branche **develop** :

MP1T0-1

Version 1.0

Joindre
Créer un ticket dans l'epic
Associer un ticket

Description
Ajouter une description...

Tickets dans cette epic
Organiser par
Progression : 100%

MP1T0-2 Initialisation du projet
MP1T0-3 Réalisation fonc 1
MP1T0-4 Réalisation fonc 2

Donnez votre avis
1

GitHub

Create
Branches
PRs
Tags

Repository *
btssn-lasalle84/mp1-team0

Base branch *
develop

Branch name *
release/MP1T0-1-version-1-0

Branch Name Template
Create branch

• Implémentation de la *release* sur la branche

On récupère la branche :

```
$ git fetch && git checkout release/MP1T0-1-version-1-0
Depuis github.com:btssn-lasalle84/mp1-team0
* [nouvelle branche] release/MP1T0-1-version-1-0 -> origin/release/MP1T0-1-version-1-0
La branche 'release/MP1T0-1-version-1-0' est paramétrée pour suivre la branche distante 'release/MP1T0-1-version-1-0' depuis 'origin'.
Basculement sur la nouvelle branche 'release/MP1T0-1-version-1-0'
```

On prépare une version livrable, par exemple la mise à jour du **README.md** :

```
$ vim README.md
$ git add README.md
$ git commit -m "MP1T0-1 Finalisation de la version 1.0"
[release/MP1T0-1-version-1-0 136f392] MP1T0-1 Finalisation de la version 1.0
1 file changed, 3 insertions(+)

$ git push
Décompte des objets: 3, fait.
Delta compression using up to 12 threads.
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 331 bytes | 331.00 KiB/s, fait.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:btssn-lasalle84/mp1-team0.git
92ea1cc..136f392 release/MP1T0-1-version-1-0 -> release/MP1T0-1-version-1-0
```

• Création d'une *Pull Request* lorsque la *release* est prête pour validation dans la branche **main**

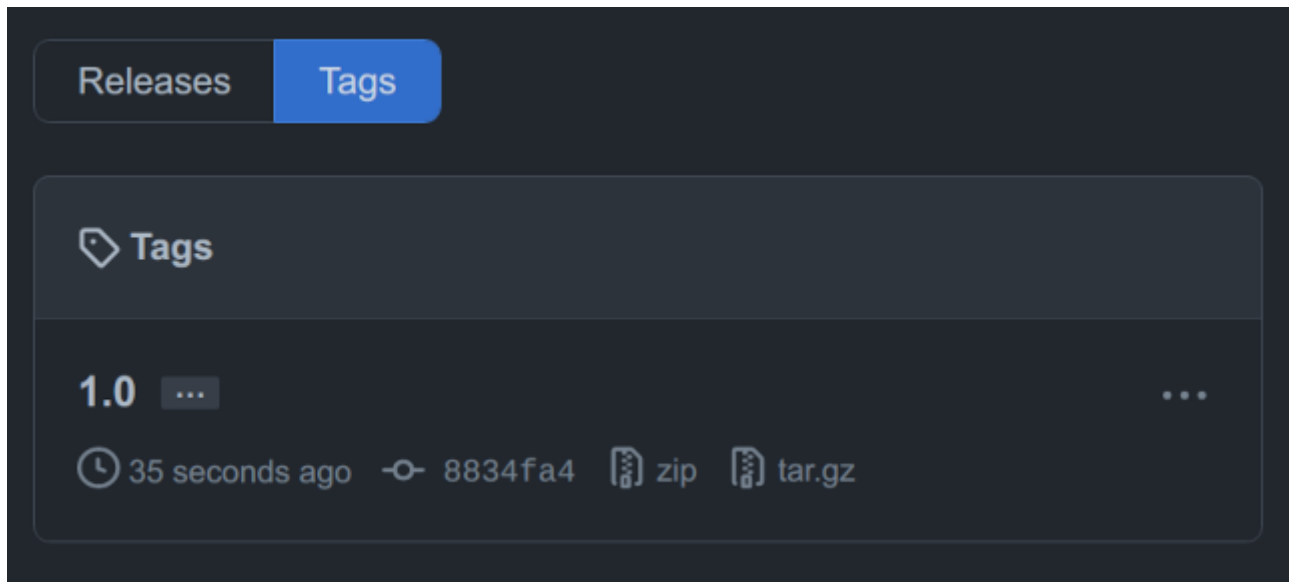
Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

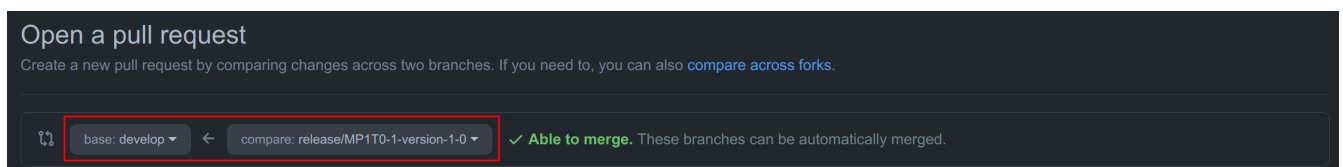
base: main
compare: release/MP1T0-1-version-1-0
Able to merge. These branches can be automatically merged.

Une fois la fusion réalisée, il faut créer un *tag* dans *main* pour cette *release* et le publier sur le dépôt distant :

```
$ git checkout main
$ git pull
$ git tag -a -m "Version 1.0" 1.0
$ git push --tags
```



Maintenant, il faut intégrer les changements éventuels à la branche *develop* pour qu'elle soit à jour pour continuer le développement. On crée une autre *Pull Request* :



On valide la fusion et on met à jour le dépôt local :

```
$ git checkout develop
$ git pull
```

Pour finir, on supprime la branche *release* :

```
$ git branch -d release/MP1T0-1-version-1-0
$ git push origin --delete release/MP1T0-1-version-1-0
```

Dans Jira, la version est automatiquement "Terminé" :

TERMINÉ 4

Version 1.0

Version 1.0



MP1T0-1

Initialisation du projet

Version 1.0



MP1T0-2

Réalisation fonc 1

Version 1.0



MP1T0-3

Réalisation fonc 2

Version 1.0



MP1T0-4

Nous n'affichons que les tickets modifiés récemment.

 [Vous recherchez un ticket plus ancien ?](#)

Feuille de route

  Catégorie d'état ▾ Versions ▾

Epic		AOÛT
>  MP1-T0-1 Version 1.0 TERMINÉ(E)		
+ Créer Epic		

On peut donc livrer la version :

Projets / MP1-Team0

Versions

Créer une version

Q	Non livrée(s) ▾					
Version ▾	État	Avancement	Date de début ▾	Date de livraison ▾	Description ▾	
1.0	NON LIVRÉE(S)					Livrer ...

Lancer 1.0

4 tickets sera/seront livré(s).

Date de livraison

8/20/2021



Lancer

Annuler

La version est maintenant à l'état "LIVRÉE" :

Versions

Q	Livrée(s), Non livrée(s) ▾			
Version ▾	État	Avancement	Date de début ▾	Date de livraison ▾
1.0	LIVRÉE(S)			20 août 2021

- **Correction d'un *bug*** (Type *hotfix* à partir de *main*)

Malheureusement, il est fort possible que le client remonte des dysfonctionnalités. Dans ce cas, il faudra réaliser un correctif qui aboutira à une nouvelle version (1.1 par exemple).

Il faut créer un ticket *bug* dans (l'Epic de) la version 1.0 :

Version 1.0

[Joindre](#)
[Créez un ticket dans l'epic](#)
[Associer un ticket](#) ▾

...

Description

Ajouter une description...

Tickets dans cette epic

Organiser par ▾ ... +

Progression : 100%

	MP1T0-2 Initialisation du projet			TERMINÉ(E) ▾
	MP1T0-3 Réalisation fonc 1			TERMINÉ(E) ▾
	MP1T0-4 Réalisation fonc 2			TERMINÉ(E) ▾

 Bug ▾

Créer

Annuler

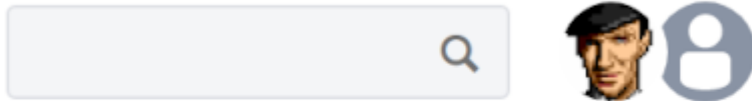
Il faut maintenant créer et associer une nouvelle version (ici 1.1) :

Epic Link

[Version 1.0](#)
Versions corrigées
1.1

Le ticket apparaît dans le tableau Kanban :

Tableau Kanban



BACKLOG 1

Défaut dans fonc 2

Version 1.0



MP1T0-5

On le sélectionne pour créer une branche de type **hotfix** à partir de **main** :

MP1T0-1 /  MP1T0-5

Défaut dans fonc 2

[Joindre](#) [Créer une sous-tâche](#) [Associer un ticket](#) [...](#)

Description

Ajouter une description...

Environnement

Aucun

Activité

Afficher : [Tout](#) [Commentaires](#) [Historique](#) [Journal du travail](#) [Git Diff](#) Les plus récents d'abord ↕



Ajouter un commentaire...

Conseil de pro : appuyez sur  pour commenter

 Donnez votre avis  1    

 GitHub

[Create](#) [Branches](#) [PRs](#) [Tags](#)

Repository *

btssn-lasalle84/mp1-team0

Base branch *

 main

Branch name *

hotfix/MP1T0-5-defaut-dans-fonc-2

 Branch Name Template

Create branch

Le ticket *bug* apparaît à l'état en cours :

0

Avertissements

1

Tickets dans la version

0

Tickets terminés

1

Tickets en cours

0

Tickets à faire

1-1 sur 1

[Afficher dans le navigateur de tickets](#)

Pr T Clé

Résumé

Responsable

État

Développement

🇫🇷 MP1T0-5 Défaut dans fonc 2



Non assigné

EN COURS

1 branche

On récupère la branche :

```
$ git fetch && git checkout hotfix/MP1T0-5-defaut-dans-fonc-2
```

On travaille sur le correctif :

```
$ vim projet.txt
$ git add projet.txt
$ git commit -m "MP1T0-5 Correction fonc 2"
$ git push
```

Création d'une *Pull Request* lorsque le correctif est prêt pour validation dans la branche **main** :

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base: main



compare: hotfix/MP1T0-5-defaut-dans-fo...



Able to merge. These branches can be automatically merged.

Une fois la fusion réalisée, il faut créer un *tag* dans **main** pour cette *release* et le publier :

```
$ git checkout main
$ git pull
$ git tag -a -m "Version 1.1" 1.1
$ git push --tags
```

Tags

1.1

18 seconds ago

7212c1a

zip

tar.gz

1.0

1 hour ago

8834fa4

zip

tar.gz

Pour finir, il faut intégrer le correctif à la branche **develop** en créant une *Pull Request* :

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: develop ← compare: hotfix/MP1T0-5-defaut-dans-fo... ✓ **Able to merge.** These branches can be automatically merged.

On valide la fusion et on met à jour le dépôt local :

```
$ git checkout develop
$ git pull
```

Pour finir, on supprime la branche **hotfix** :

```
$ git branch -d hotfix/MP1T0-5-defaut-dans-fonc-2
$ git push origin --delete hotfix/MP1T0-5-defaut-dans-fonc-2
```

Le dépôt distant :

The screenshot shows a GitHub pull request interface. At the top, it says 'develop' and 'mp1-team0 / projet.txt'. Below that, the title is 'tvaïra MP1T0-5 Correction fonc 2' with the latest commit '1e77668 5 minutes ago'. It shows '1 contributor'. The file content is displayed with line numbers 1 to 6. The content is: 1 fonc 1 : ok, 2 fonc 2 : ok, correction 1, 3 fonc 3 : vide, 4 fonc 4 : vide, 5 fonc 5 : vide, 6. There are buttons for 'Edit on HackMD', 'Raw', 'Blame', and a trash icon.

Dans Jira, on peut livrer la version **1.1** :

Versions

Version ↕	État	Avancement
1.1	LIVRÉE(S)	
1.0	LIVRÉE(S)	

L'historique du projet en mode graphique :

Graph	Description	Date	Author	Commit
	develop <i>origin</i> Fusion effectuée hotfix/MP1T1-5-défaut-dans-fonc-2 (pull request #6)	21 Aug 2021 08:04	Thierry Vaira	606712b3
	master <i>origin</i> 1.1 Fusion effectuée hotfix/MP1T1-5-défaut-dans-fonc-2 (pull request #5)	21 Aug 2021 08:02	Thierry Vaira	78efe21f
	hotfix/MP1T1-5-défaut-dans-fonc-2 <i>origin</i> MP1T1-5 Correction fonc 2	21 Aug 2021 08:01	vaira	f9e053d2
	Fusion effectuée release/MP1T1-1-version-1.0 (pull request #4)	21 Aug 2021 07:55	Thierry Vaira	0e6a3662
	1.0 Fusion effectuée release/MP1T1-1-version-1.0 (pull request #3)	21 Aug 2021 07:54	Thierry Vaira	f76c8ec9
	MP1T1-1 Livraison 1.0	21 Aug 2021 07:52	vaira	f4651a16
	Fusion effectuée feature/MP1T1-4-réalisation-fonc-2 (pull request #2)	21 Aug 2021 07:46	Thierry Vaira	d81e31ed
	MP1T1-4 Réalisation fonc 2 ok	21 Aug 2021 07:45	vaira	3302cea2
	Fusion effectuée feature/MP1T1-3-réalisation-fonc-1 (pull request #1)	19 Aug 2021 16:12	Thierry Vaira	45a8c769
	MP1T1-3 Réalisation fonc 1 ok	19 Aug 2021 16:08	vaira	61ee1d0b
	Fusion de la branche 'feature/MP1T1-2-initialisation-du-projet' dans develop	19 Aug 2021 15:56	vaira	0499c796
	MP1T1-2 Ajout du fichier projet.txt	19 Aug 2021 15:54	vaira	40cb4b10
	Initial commit	18 Aug 2021 15:16	vaira	ddc49adc

2.10. Jira et Bitbucket

On sélectionne un ticket (ici une fonctionnalité) :

MP1T1-1 / MP1T1-2

Donnez votre avis 1

Initialisation du projet

Joindre Créer une sous-tâche Associer un ticket ...

Description
Ajouter une description...

Activité
Afficher : Tout Commentaires Historique Journal du travail Les plus récents d'abord

Ajouter un commentaire...
Conseil de pro : appuyez sur **M** pour commenter

Détails

Responsable Non assigné

Rapporteur Thierry Vaira

Développement **Créer une branche**

Étiquettes Aucun

Epic Link Version 1.0

Versions corrigées 1.0

Priorité Medium

Automation Rule executions

Plus de champs

Story Points Aucun

Estimation originale 0min

Suivi temporel Aucun temps enregistré

Composants Aucun

On crée la branche (Gitflow) :

Create branch

Repository

btssn-avignon/miniprojet1-team1

Type i

Feature

From branch

develop

Branch name

feature/ MP1T1-2-initialisation-du-projet



 develop

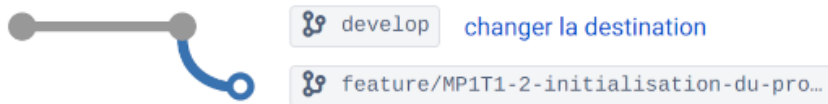
 feature/MP1T1-2-initialisation-du-proj...

Create

Cancel

On récupère la branche (copier/coller) :

feature/MP1T1-2-initialisation-du-projet



Consultez cette branche

Cette branche ne contient aucune modification pour l'instant. Consultez-la depuis votre ordinateur local pour la modifier.

[Consulter via Sourcetree](#)

You can also use this command to check out your branch:

```
git fetch && git checkout feature/MP1T1-2-initialisation-du-projet
```

```
$ git fetch && git checkout feature/MP1T1-2-initialisation-du-projet
Password for 'https://btssn-avignon-admin@bitbucket.org':
Depuis https://bitbucket.org/btssn-avignon/miniprojet1-team1
 * [nouvelle branche] feature/MP1T1-2-initialisation-du-projet ->
origin/feature/MP1T1-2-initialisation-du-projet
La branche 'feature/MP1T1-2-initialisation-du-projet' est paramétrée pour suivre la
branche distante 'feature/MP1T1-2-initialisation-du-projet' depuis 'origin'.
Basculement sur la nouvelle branche 'feature/MP1T1-2-initialisation-du-projet'

$ vim projet.txt

$ git add projet.txt
```

Au minimum, il faut inclure la clé du ticket au début du message de *commit* pour faire lien avec Jira :

```
$ git commit -m "MP1T1-2 Ajout du fichier projet.txt"

$ git push
Décompte des objets: 3, fait.
Delta compression using up to 12 threads.
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 292 bytes | 292.00 KiB/s, fait.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for feature/MP1T1-2-initialisation-du-projet:
remote:  https://bitbucket.org/btssn-avignon/miniprojet1-team1/pull-
requests/new?source=feature/MP1T1-2-initialisation-du-projet&t=1
remote:
To https://bitbucket.org/btssn-avignon/miniprojet1-team1.git
   ddc49ad..40cb4b1  feature/MP1T1-2-initialisation-du-projet -> feature/MP1T1-2-
initialisation-du-projet
```

Deux situations possibles :

- fusion locale dans la branche **develop** puis on publie sur le dépôt central

```
$ git checkout develop

$ git pull

$ git merge --no-ff feature/MP1T1-2-initialisation-du-projet
Merge made by the 'recursive' strategy.
 projet.txt | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 projet.txt

$ git push
```

- création d'une *Pull Request* dans Bitbucket (avec le lien fourni après le **push**)

Créer une pull request

 btssn-avignon / miniprojet1-team1
Created yesterday, updated 49 seconds ago

 feature/MP1T1-3-réalisation-fonc-1

→

 btssn-avignon/miniprojet1-team1

 develop

Title* MP1T1-3 Réalisation fonc 1 ok

Description

Aa **B** *I* ...     @   <> ” + **Feedback** ?

Attachments [Browse to upload](#)

Relecteurs Add reviewers...

☐ Supprimer la branche Delete feature/MP1T1-3-réalisation-fonc-1 after the pull request is merged

Créer une pull request

Différences Commits

Auteur	Committer	Message
 Thierry Vaira	61ee1d0	MP1T1-3 Réalisation fonc 1 ok

La *Pull Request* dans Bitbucket :

MP1T1-3 Réalisation fonc 1 ok

 feature/MP1T1-3-réalisation-fo... → develop **OPEN**
#1 · Created il y a 40 secondes · Last updated il y a 40 secondes

Modifier

✓ Approuver

Fusionner

...

Settings 

Description

Add a description...

> 0 attachments

0 comments



Add a comment

> 1 commit

1 file

FILTER BY COMMENTS



SORT BY

File tree



▼  projet.txt 									
@@ -1,4 +1,4 @@									
1	- fonc 1 : vide			1	+ fonc 1 : ok				
2	fonc 2 : vide			2	fonc 2 : vide				
3	fonc 3 : vide			3	fonc 3 : vide				
4	fonc 4 : vide			4	fonc 4 : vide				
↓									

On fusionne :

Fusionner la pull request

Source

feature/MP1T1-3-réalisation-fonc-1

Destination

develop

Commit message

Fusion effectuée feature/MP1T1-3-réalisation-fonc-1 (pull request #1)

MP1T1-3 Réalisation fonc 1 ok

Merge strategy

Merge commit

Transition issue

☒  MP1T1-3 Réalisation fonc 1 → **TERMINÉ** ▼

☐ Fermer la branche source

Fusionner

Annuler

L'historique :

btssn-avignon / miniprojet1-team1 / miniprojet1-team1

Commits

Search commits		All branches	
Author	Commit	Message	Date
  Thierry Vaira	45a8c76	MERGED Fusion effectuée feature/MP1T1-3-réalisation-fonc-1 (pull ... develop)	il y a 23 secon...
 Thierry Vaira	61ee1d0	MP1T1-3 Réalisation fonc 1 ok 2 branches	il y a 4 minutes
 Thierry Vaira	0499c79	MERGED Fusion de la branche 'feature/MP1T1-2-initialisation-du-... 2 branches	il y a 16 minutes
 Thierry Vaira	40cb4b1	MP1T1-2 Ajout du fichier projet.txt 3 branches	il y a 17 minutes
 Thierry Vaira	ddc49ad	Initial commit	hier

À la fin, mise à jour :

```
$ git checkout develop
$ git pull
$ cat projet.txt
```

Avec Gitflow, seules les branches **main** et **develop** sont permanentes. Les autres branches peuvent être supprimées :

```
$ git branch -d feature/MP1T1-2-initialisation-du-projet
$ git push origin --delete feature/MP1T1-2-initialisation-du-projet
```

Dans Jira, le ticket est passé automatiquement à l'état "Terminé" :

TERMINÉ 1

Initialisation du projet

Version 1.0



MP1T1-2

Nous n'affichons que les tickets modifiés récemment.

 Vous recherchez un ticket plus ancien ?

Avec des informations sur le développement :

Réalisation fonc 1

 Joindre  Créer une sous-tâche  Associer un ticket  ...

Description

Ajouter une description...

Activité

Afficher : Tout Commentaires Historique Journal du travail

Les plus récents d'abord ↕



Ajouter un commentaire...

Conseil de pro : appuyez sur  pour commenter

Terminé(e)  Terminé

Détails

Responsable  Non assigné

Rapporteur  Thierry Vaira

Développement  1 branche
2 commits il y a 54 secondes
1 pull request MERGED

Étiquettes Aucun

Epic Link Version 1.0

Versions corrigées 1.0

Priorité Medium

Automation  Rule executions

Thierry Vaira - <tvaira@free.fr> - version v0.2 - 23/08/2021 - btssn-lasalle84.github.io