

# Marvel - Version 0.2.0

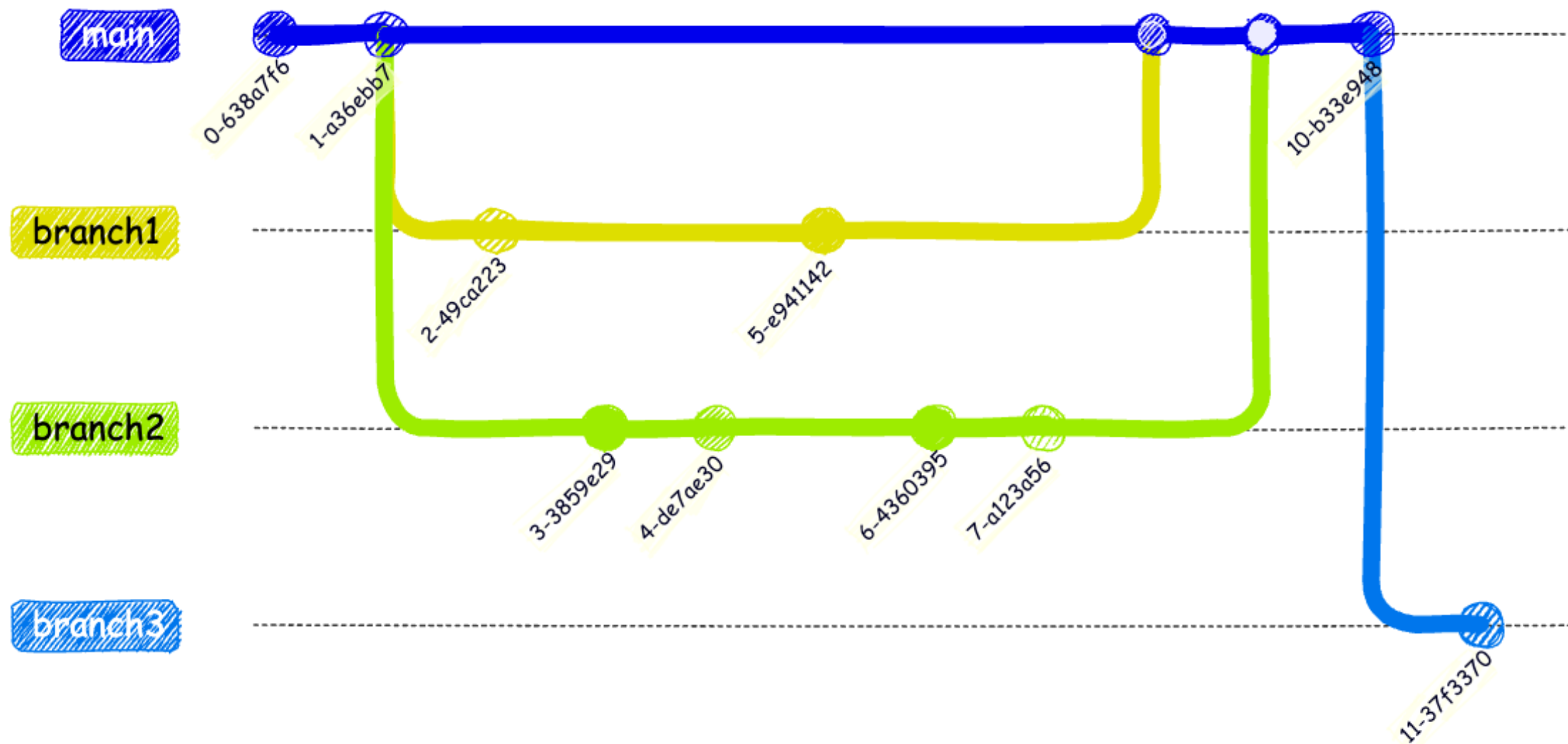
# git - Branches

- Les branches constituent un élément central de Git.
- Permettent de travailler sur plusieurs aspects d'un projet en parallèle.
- Très utiles pour travailler sur des fonctionnalités en cours de développement sans impacter le code de production.
- Permet de développer une nouvelle fonctionnalité sur une branche dédiée sans risquer d'altérer le code de la branche principale, qui reste stable à tout moment.

# git - Branches (suite)

Lors de la création d'une branche, celle-ci est dupliquée à partir de la branche courante. Les modifications apportées à cette branche n'impactent pas les autres branches. Une fois les modifications terminées, il est possible de fusionner la branche avec la branche courante.

- Les branches sont des pointeurs vers un commit.
- Créer une nouvelle branche revient à créer un nouveau pointeur sur le commit courant.
- Les branches sont très légères et peu coûteuses en ressources.
- Les branches sont locales par défaut.
- Les branches peuvent être partagées avec d'autres développeurs en les poussant sur le dépôt distant.



# gitflow

- Méthode de gestion des branches en utilisant Git.
- Permet de structurer le développement logiciel en définissant des règles pour les branches.
- Utilise des branches spécifiques pour les fonctionnalités, les correctifs, les versions, les releases...
- Permet de travailler sur plusieurs fonctionnalités en même temps sans impacter le code de production.

## gitflow - branche **develop**

La branche **develop** est la branche de développement. Elle contient les fonctionnalités en cours de développement, ainsi la branche **main** reste stable et contient le code de production.

On ne travaille jamais directement sur la branche **main**, afin de ne pas impacter involontairement le code de production (risque de bugs, de régressions...).

## gitflow - branches **feature**

Lorsque l'on souhaite ajouter une nouvelle fonctionnalité, on crée une branche spécifique **feature/nom-fonctionnalite** à partir de la branche **develop**.

Une fois la fonctionnalité terminée, on fusionne la branche avec la branche **develop**.

Le fait de travailler sur des branches **feature** permet de travailler sur plusieurs fonctionnalités en même temps pour une même version en cours de développement.

# gitflow - Mise en pratique

- Créer une nouvelle branche **develop** à partir de la branche principale
  - `git checkout -b develop`
- Créer une branche **feature/style** pour le style
  - `git checkout -b feature/style`
- Créer une branche **feature/data** pour les données
  - `git checkout -b feature/data`



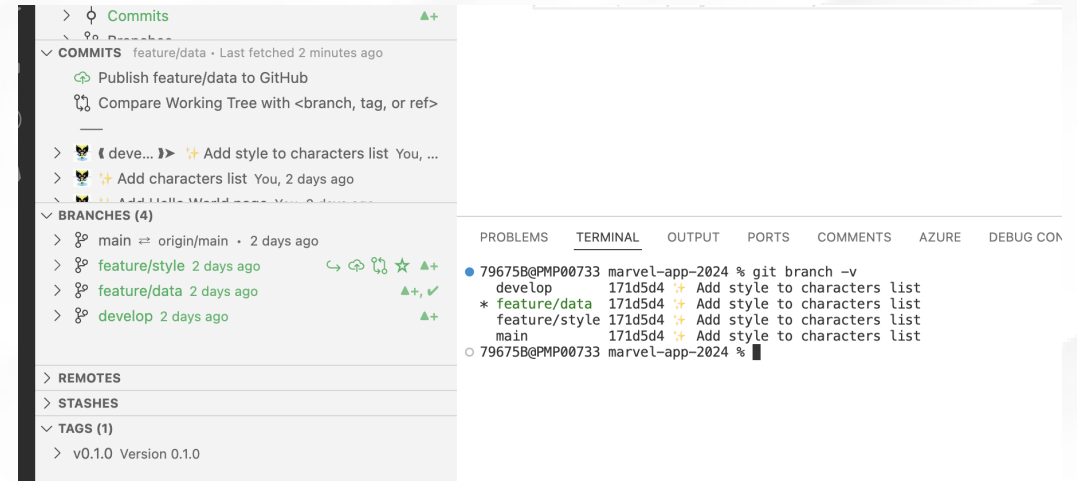
# gitflow - Mise en pratique (suite)

Nous sommes maintenant sur les branches **feature-style** et **feature-data**. Nous pouvons travailler sur ces branches en parallèle pour ajouter les fonctionnalités de style et de données.



## gitflow - Mise en pratique (suite)

- Lister les branches avec la commande: `git branch -v`
- L'option **-v** affiche le dernier commit de chaque branche. On voit ici que toutes les branches sont au même niveau (même commit)
- La branche courante est indiquée par un astérisque



# Fonctionnalités - Style

- Se positionner sur la branche **feature/style** avec la commande:  
git checkout feature/style
- Modifier le fichier **src/style.css** pour ajouter du style à la page web
- Commiter avec le message "Modify style for characters list"



# gitflow - Mise en pratique (suite)

L'état des branches après les modifications de style devrait être le suivant:



# gitflow - Mise en pratique (suite)

L'extension **Git Graph** de **Visual Studio Code** permet de visualiser les branches et les commits.

The screenshot shows the Visual Studio Code interface with the Git Graph extension. The left sidebar displays the 'SOURCE CONTROL' view, which includes a commit message field and a 'Publish Branch' button. The main area shows the 'Git Graph' view, which visualizes the Git history. The 'feature/style' branch is selected, and its commit history is shown, including 'Add style to characters list' and 'Add characters list'.

**SOURCE CONTROL**

Message (⌘Enter to commit on "feature/style")

**Publish Branch**

Outgoing

> feature/style

**Git Graph**

**Graph**

**Description**

- feature/style ✨ Modify style for characters list
- develop ✨ Add style to characters list
- feature/data ✨ Add characters list
- main ✨ Add Hello World page
- origin ✨ Prepare project for development
- origin/HEAD ✨ Initial commit
- v0.1.0 ✨ Add style to characters list

# Fonctionnalités - Données

- Se positionner sur la branche **feature/data** avec la commande: git checkout feature/data
- Créer un fichier **src/data/characters.json** pour ajouter des données sur les personnages Marvel et simuler une API
  - Le fichier characters.json contient un tableau d'objets. Chaque objet représente un personnage. Chaque personnage a un attribut name qui contient le nom du personnage.
- Les données sont ensuite accessibles <http://localhost:3000/data/characters.json>

## Fonctionnalités - Données (suite)

- Créer un fichier **src/script.js** pour ajouter du code JavaScript pour récupérer les données depuis le fichier `characters.json` dans une fonction **getCharacters**
  - Appeler l'API <http://localhost:3000/data/characters.json> pour récupérer les données
  - Afficher les données dans la console du navigateur
- Modifier le fichier **src/index.html** pour ajouter une balise **script** pour charger le fichier **script.js** et appeler la fonction pour récupérer les données.
- Vérifier que les données s'affichent dans la console du navigateur
- Commiter avec le message "Add script to get characters data"

## Fonctionnalités - Données (suite)

- Modifier l'appel de la fonction **getCharacters** pour modifier le contenu de la liste des personnages avec les données récupérées
  - Utiliser un identifiant **characters** pour récupérer l'élément **ul** de la liste des personnages
  - Ajouter un élément **li** pour chaque personnage
- Commiter avec le message "Update characters list with data"



# gitflow - Mise en pratique (suite)

L'état des branches après les modifications de données devrait être le suivant:



# gitflow - Mise en pratique (suite)



# git - Merge

- Une fois les fonctionnalités terminées, il est possible de fusionner les branches avec la branche **develop**.
- La fusion des branches se fait avec la commande **git merge**.
- La fusion des branches peut générer des conflits si les mêmes fichiers ont été modifiés sur les branches à fusionner.
- Les conflits doivent être résolus manuellement.
- Dans le cas de notre exemple, il ne devrait pas y avoir de conflits. Nous verrons comment les résoudre dans un exemple ultérieur.

## git - Merge (suite)

- Se positionner sur la branche **develop** avec la commande: `git checkout develop`
- Fusionner la branche **feature-style** avec la branche **develop** avec la commande: `git merge feature/style`

# git - Merge (suite)



## git - Merge (suite)

- Fusionner la branche **feature-data** avec la branche **develop** avec la commande:

```
git merge feature/data
```

# git - Merge (suite)

**Publish Branch**

Outgoing

✓ **develop** 4

- > **All Changes** 4 +76 -18
- > **Merge branch 'feature/data' into de...** 3 +60 -17
- > **Update characters list with data al...** 2 +8 -18
- > **Add script to get characters data ale...** 3 +53
- > **Modify style for characters list alex...** 1 +16 -1

**Graph**

**develop** Merge branch 'feature/data' into develop

- feature/data** ✨ Update characters list with data
- ✨ Add script to get characters data
- feature/style** ✨ Modify style for characters list
- main** *origin* **origin/HEAD** **v0.1.0** ✨ Add style to characters list
- ✨ Add characters list
- ✨ Add Hello World page
- ✨ Prepare project for development
- Initial commit

**Description**

## git - Merge (suite)

Les deux branches **feature/style** et **feature/data** ont été fusionnées avec la branche **develop**.

Ayant terminé les 2 fonctionnalités, nous pouvons supprimer les branches **feature/style** et **feature/data**.

- Supprimer la branche **feature/style** avec la commande: `git branch -d feature/style`
- Supprimer la branche **feature/data** avec la commande: `git branch -d feature/data`



# git - Merge (suite)

The screenshot displays a Git web interface with three main sections: 'Publish Branch', 'Graph', and 'Description'.

**Publish Branch:** A blue button labeled 'Publish Branch' is at the top. Below it, the 'Outgoing' section shows a list of changes for the 'develop' branch. The list includes 'All Changes' (4 changes, +76 -18), 'Merge branch 'feature/data' into de...' (3 changes, +60 -17), 'Update characters list with data al...' (2 changes, +8 -18), 'Add script to get characters data ale...' (3 changes, +53), and 'Modify style for characters list alex...' (1 change, +16 -1).

**Graph:** A vertical timeline of commits is shown. A blue line represents the 'main' branch, and a pink line represents the 'feature/data' branch. A merge operation is indicated by a pink line connecting the 'feature/data' branch to the 'develop' branch.

**Description:** The 'develop' branch is selected. The description shows the merge operation: 'Merge branch 'feature/data' into develop'. Below this, a list of changes is shown: 'Update characters list with data', 'Add script to get characters data', 'Modify style for characters list', 'Add style to characters list', 'Add characters list', 'Add Hello World page', 'Prepare project for development', and 'Initial commit'.

## git - Merge (suite)

Nous avons atteint l'objectif de la version 0.2.0 de l'application Marvel.

Nous avons ajouté du contenu JavaScript pour récupérer des données depuis un fichier JSON et du style CSS pour améliorer l'affichage.

Nous sommes maintenant prêts à fusionner la branche **develop** avec la branche principale **main**.

## git - Merge (suite)

- Avant de fusionner la branche **develop** avec la branche **main**, il est recommandé de vérifier que tout est fonctionnel.
- Tester l'application en local pour vérifier que les fonctionnalités ajoutées fonctionnent correctement.
- Vérifier que les modifications sont cohérentes avec les objectifs de la version 0.2.0.
- Modifier la version dans le fichier **package.json** pour refléter la nouvelle version de l'application et commiter la modification.

# git - Merge (suite)

The screenshot displays a Git web interface with two main panels. The left panel, titled 'Publish Branch', shows the 'Outgoing' section for the 'develop' branch. It lists a series of commits with their respective file changes (additions and deletions). The right panel, titled 'Graph', shows a commit graph with a vertical line of blue dots representing commits. A pink line branches off from the main line, indicating a merge or a new branch creation. Below the graph, the 'develop' branch is highlighted, and a list of commits is shown, including 'prepare version 0.2.0' and 'Merge branch 'feature/data' into develop'.

**Publish Branch**

Outgoing

- ✓ **develop** (5)
- > **All Changes** (5 +77 -19)
- > **prepare version 0.2.0** alexandre-girard (1 +1 -1)
- > **Merge branch 'feature/data' into de...** (3 +60 -17)
- > **Update characters list with data al...** (2 +8 -18)
- > **Add script to get characters data ale...** (3 +53)
- > **Modify style for characters list alex...** (1 +16 -1)

**Graph**

**develop** **prepare version 0.2.0**

Merge branch 'feature/data' into develop

- ✦ Update characters list with data
- ✦ Add script to get characters data
- ✦ Modify style for characters list

**main** **origin** **origin/HEAD** **v0.1.0** ✦ Add style to characters list

- ✦ Add characters list
- ✦ Add Hello World page
- ✦ Prepare project for development

Initial commit

## git - Merge (suite)

- Se positionner sur la branche **main** avec la commande: `git checkout main`
- Fusionner la branche **develop** avec la branche **main** avec la commande: `git merge develop`

# git - Merge (suite)

The screenshot displays a Git GUI interface during a merge operation. On the left, the 'Outgoing' pane shows the 'main' branch selected. The right pane is divided into two sections: a commit graph and a list of changes.

**Commit Graph:** The graph shows a vertical line of blue dots representing commits. A pink line branches off from the main line, indicating a merge. The top of the graph is labeled 'main', 'develop', and 'prepare version 0.2.0'.

**Changes List:** The list of changes includes the following items:

- Merge branch 'feature/data' into develop
- ✦ Update characters list with data
- ✦ Add script to get characters data
- ✦ Modify style for characters list
- origin/HEAD origin/main v0.1.0 ✦ Add style to characters list
- ✦ Add characters list
- ✦ Add Hello World page
- ✦ Prepare project for development
- Initial commit

# git - Merge (suite)



## git - push

- Une fois la branche **main** fusionnée avec la branche **develop**, il est possible de pousser les modifications sur le dépôt distant.
- Pousser les modifications sur le dépôt distant avec la commande: `git push origin main`
- Créer un tag pour la version 0.2.0 avec la commande: `git tag -a v0.2.0 -m "Version 0.2.0"`
- Pousser le tag sur le dépôt distant avec la commande: `git push origin v0.2.0`