Marvel - Version 0.2.0

Alexandre GIRARD - Conseiller en Nouvelles Technologies - alexandre.girard@maif.fr

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 fonctionalité 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.

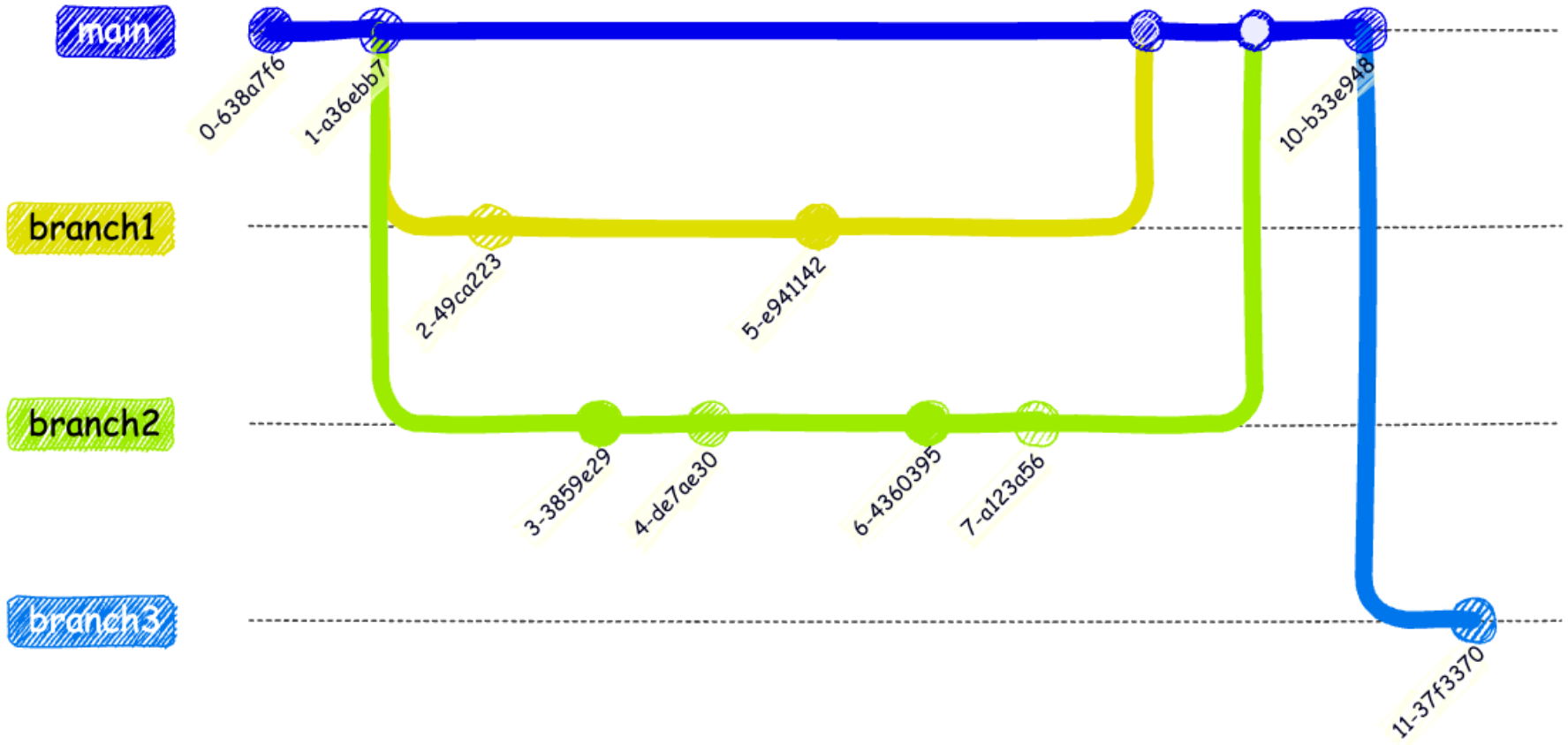
Les modifications apportées à cette branche n'impactent pas les autres branches.

Une fois les modifications terminées, il est possible de fusionner la branche avec la branche courante.

git - Branches (suite)

- 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.

git - Branches (exemple)



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**.

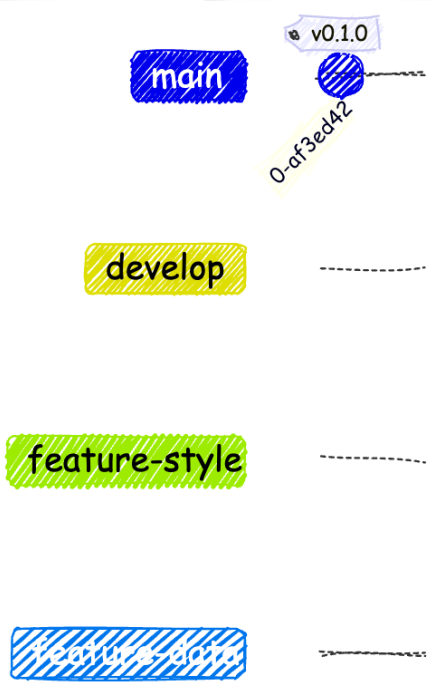
Le fait de travailler sur des branches **feature** permet de travailler sur plusieurs fonctionnalités en même temps pour une même version en cours de développement.

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`

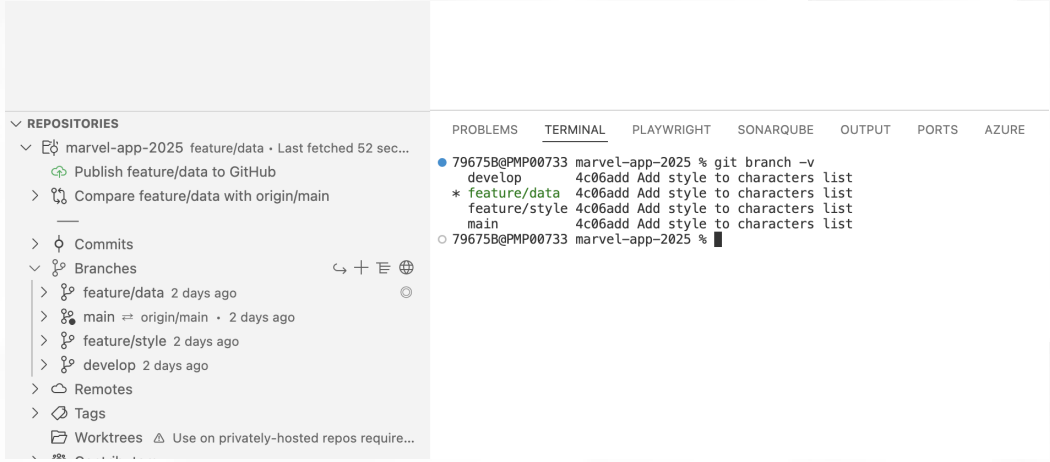
gitflow - Mise en pratique (suite)

Nous venons de créer deux branches de type **feature**: **feature/style** et **feature/data** et une branche de développement **develop**.



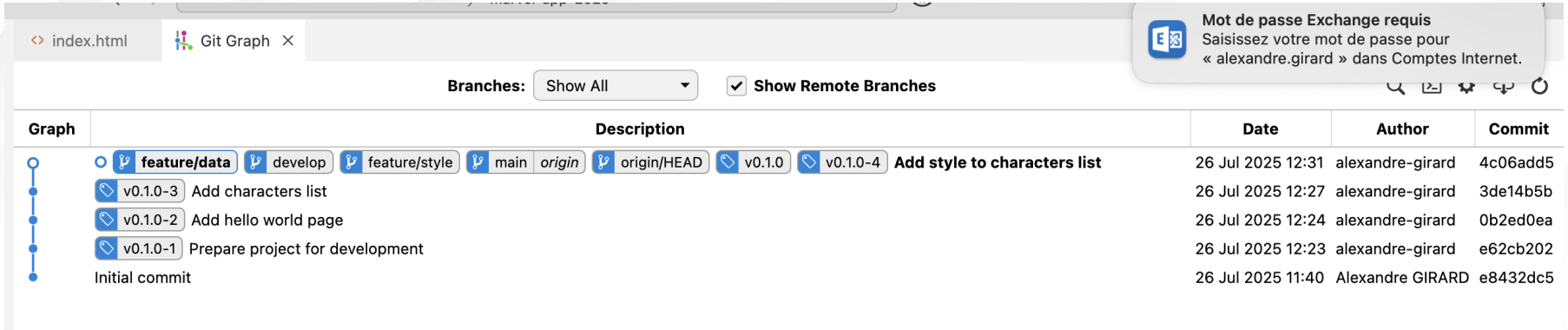
gitflow - Mise en pratique (suite)

- 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



gitflow - Mise en pratique (suite)

La vue **Git Graph** de **Visual Studio Code** permet aussi de visualiser les branches et les commits de manière graphique.

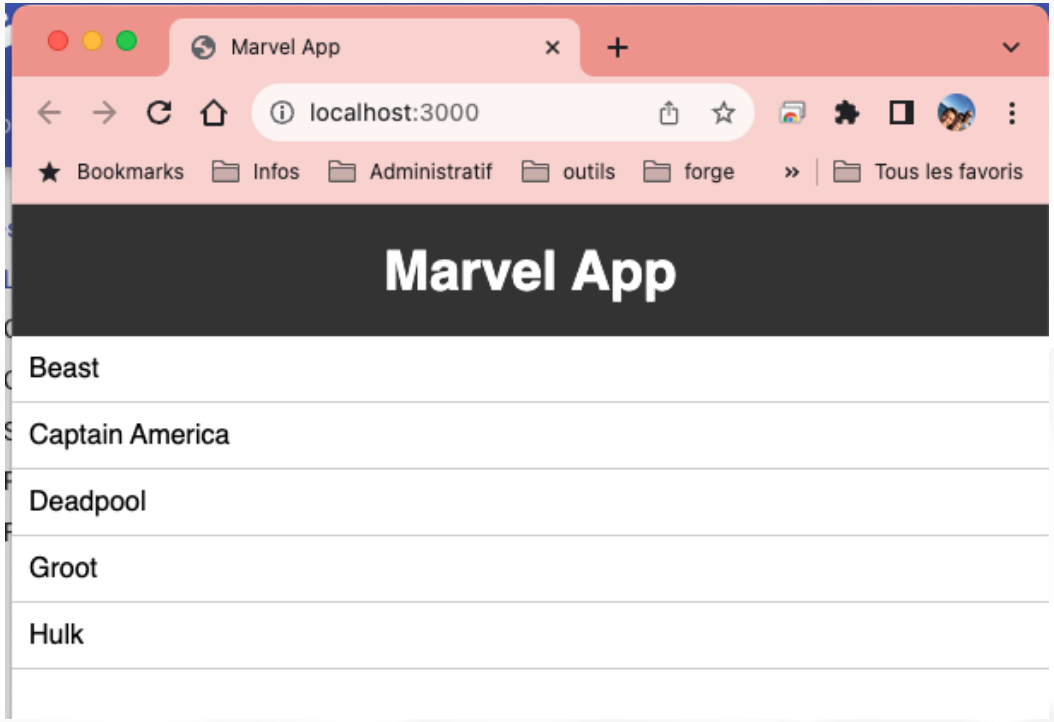


Ainsi pour l'instant nous voyons que nous avons plusieurs branches, mais qu'elles sont toutes au même niveau (même commit).

Nous voyons aussi les précédents commits de la branche principale **main**.

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



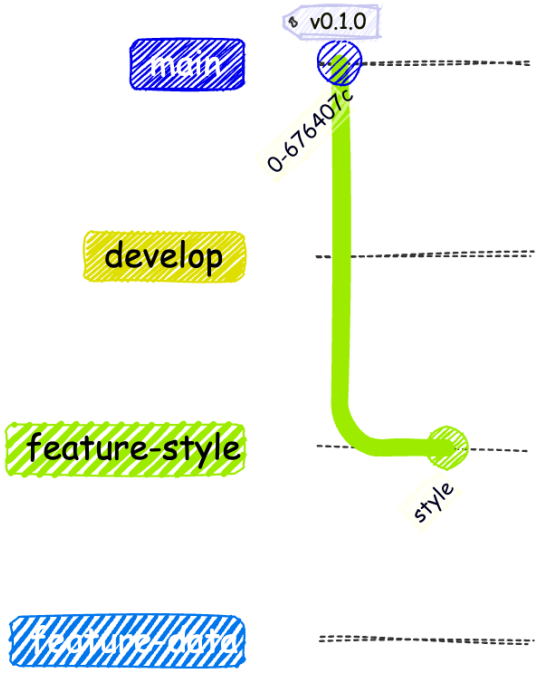
```css
body {
    font-family: sans-serif;
    margin: 0;
    padding: 0;
}

h1 {
    margin: 0;
    padding: 20px;
    background: #333;
    color: #fff;
    text-align: center;
}

ul {
    list-style: none;
    margin: 0;
    padding: 0;
}

li {
    padding: 10px;
    border-bottom: 1px solid #ccc;
}
```

Fonctionnalités - Style (suite)

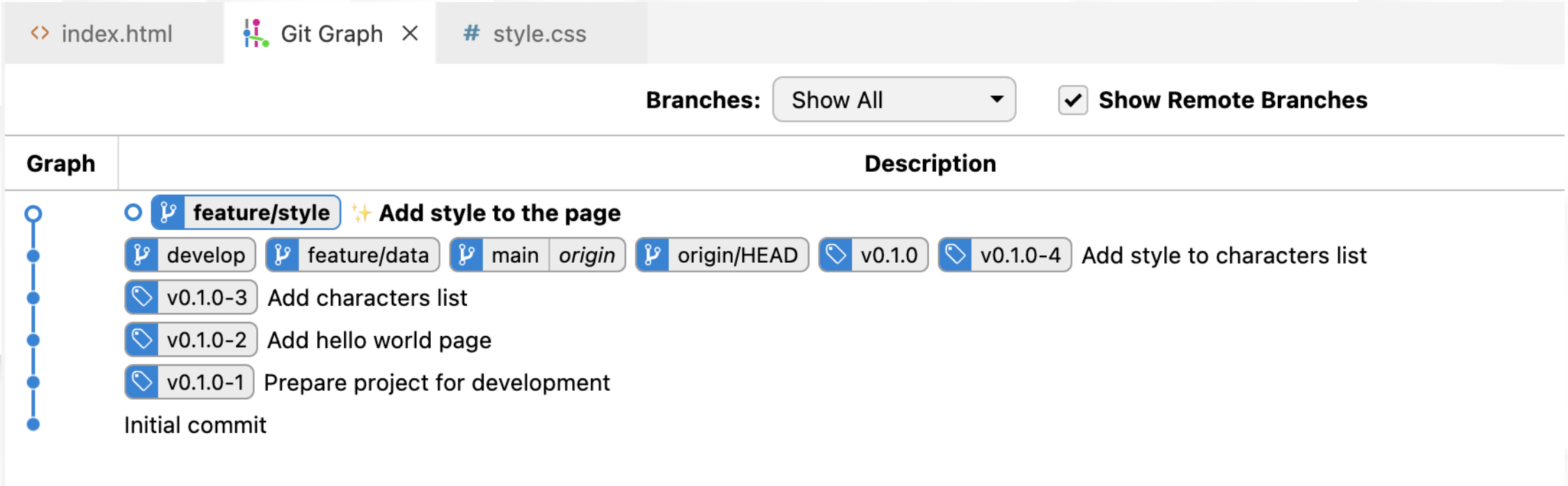
Commiter les modifications avec le message "Add style to the page".

gitflow - Mise en pratique (suite)

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



gitflow - Mise en pratique (suite)



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.
- Vous pouvez récupérer les données des personnages ici
- Les données sont ensuite accessibles http://localhost:3000/data/characters.json

Fonctionnalités - Données (suite)

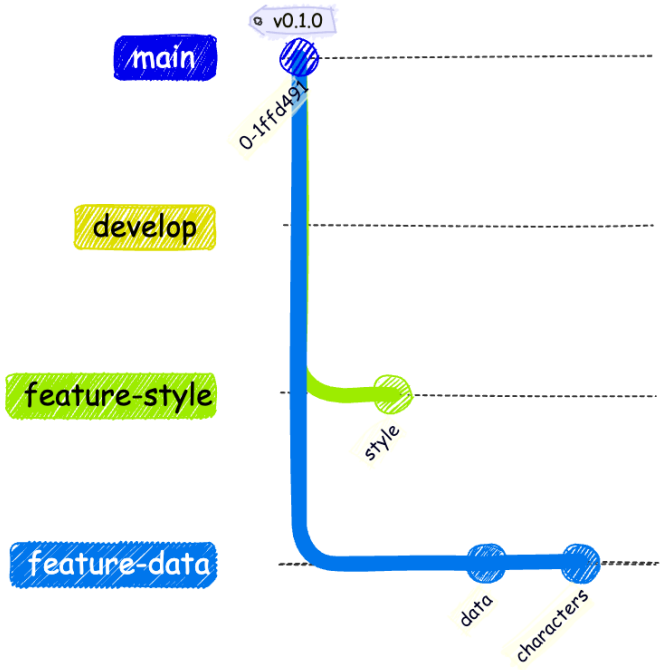
- Créer un fichier **src/script.js** pour ajouter du code JavaScript pour récupérer les données depuis le fichier characters.json dans une fonction **getCharacters**
  - Appeler l'API http://localhost:3000/data/characters.json pour récupérer les données
  - Afficher les données dans la console du navigateur
- 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)

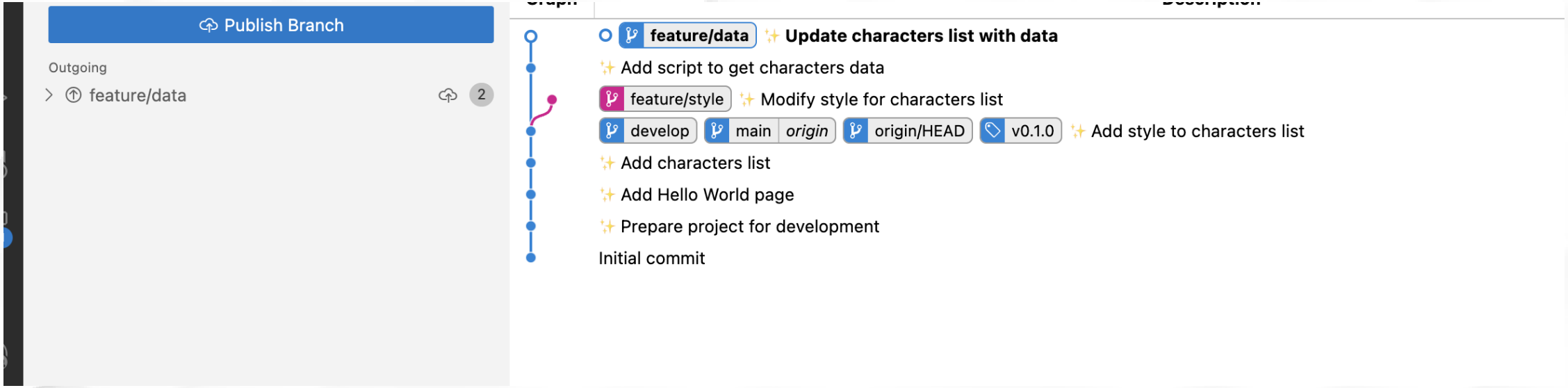
- Modifier l'appel de la fonction **getCharacters** pour modifier le contenu de la liste des personnages avec les données récupérées
  - Utiliser un identifiant **characters** pour récupérer l'élément **ul** de la liste des personnages
  - Ajouter un élément **li** pour chaque personnage
- Commiter avec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



En basculant entre les branches **feature/style** et **feature/data**, on peut voir que les modifications apportées à chaque branche sont indépendantes.

Les modifications de style n'ont pas impacté les données et vice versa.

git - Merge

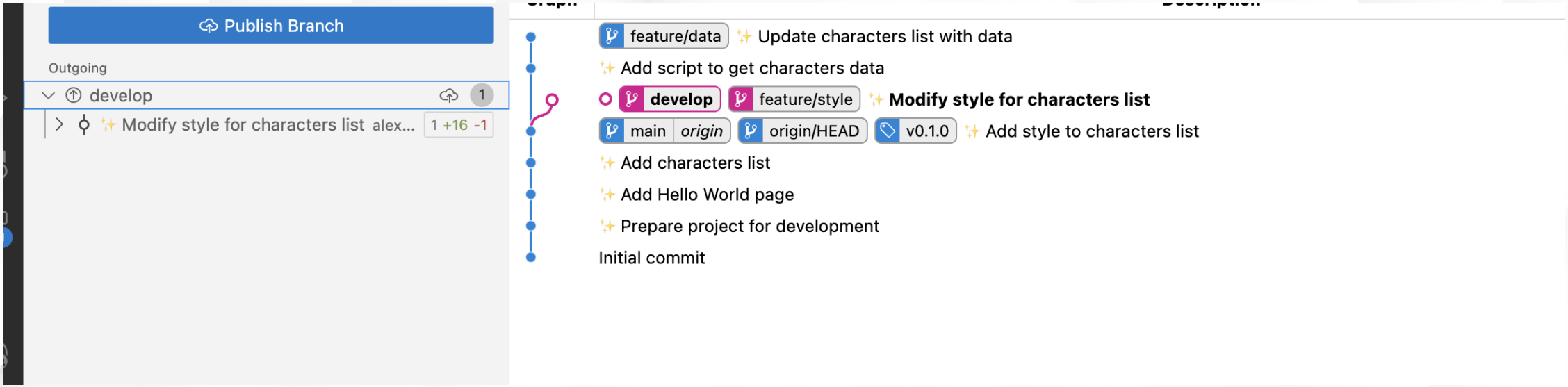
Une fois les fonctionnalités terminées, il est possible de fusionner les branches avec la branche **develop**.

- 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)



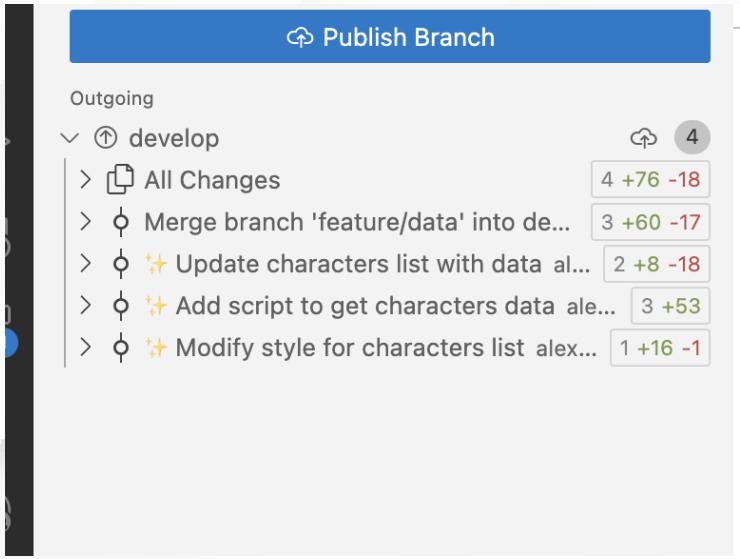
On voit que la branche **feature/style** a été fusionnée avec la branche **develop**, elles sont maintenant au même niveau.

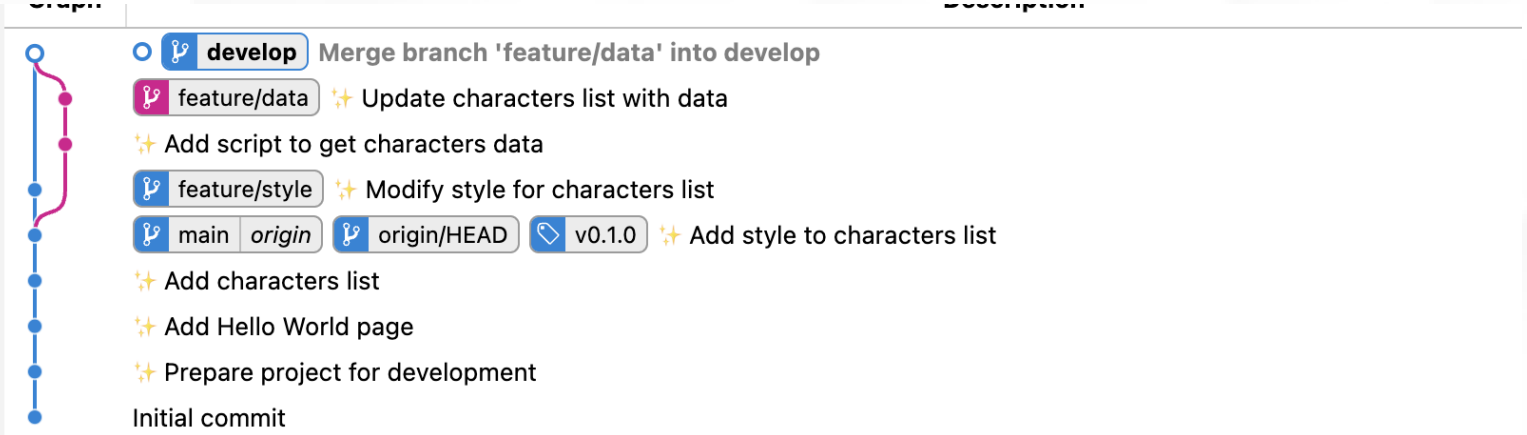
git - Merge (suite)

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

```
git merge feature/data
```

git - Merge (suite)





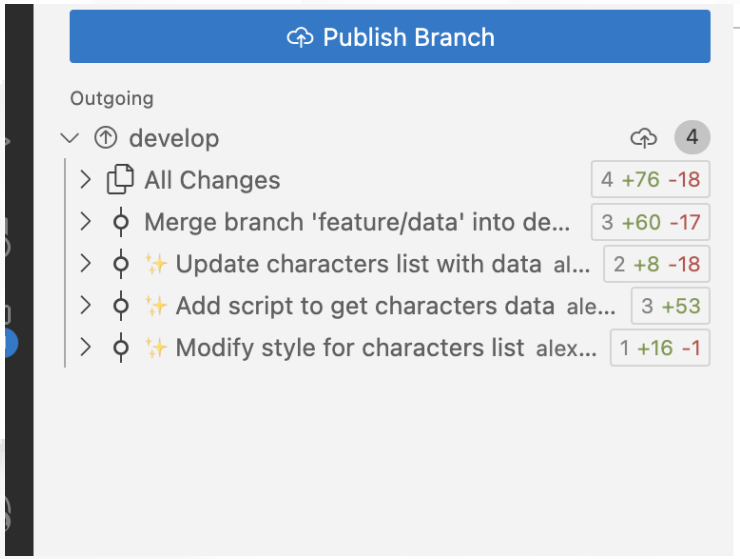
git - Merge (suite)

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)



git - Merge (suite)

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

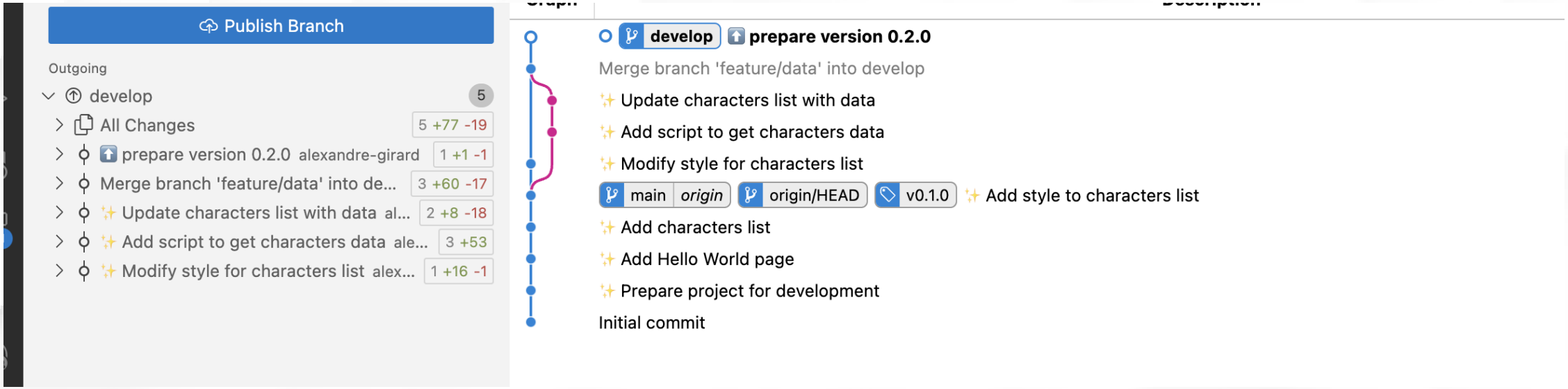
Nous avons ajouté du contenu JavaScript pour récupérer des données depuis un fichier JSON et du style CSS pour améliorer l'affichage.

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
- Commiter la modification avec le message "Update version to 0.2.0" `git commit -m "prepare version to 0.2.0"`

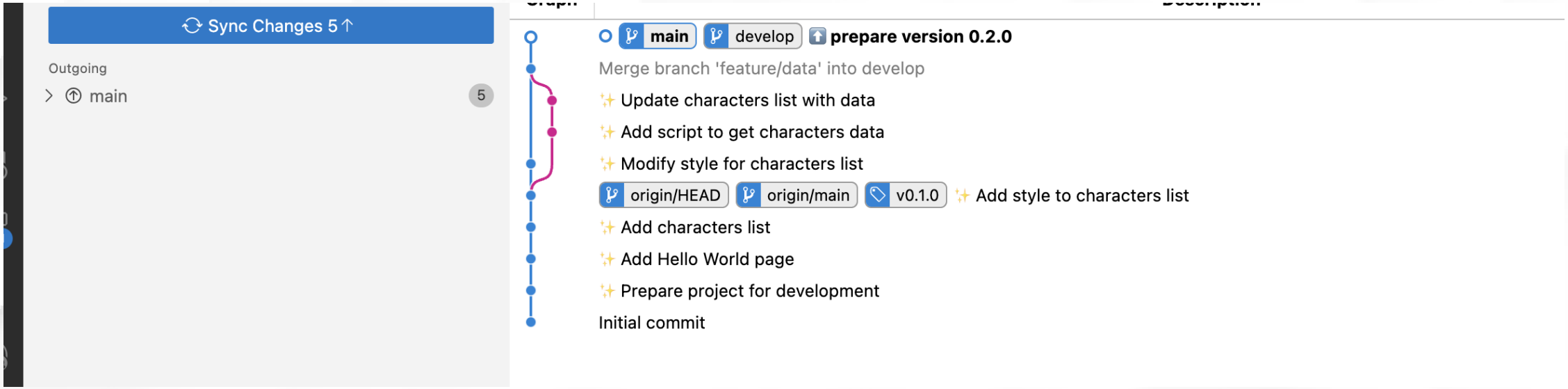
git - Merge (suite)



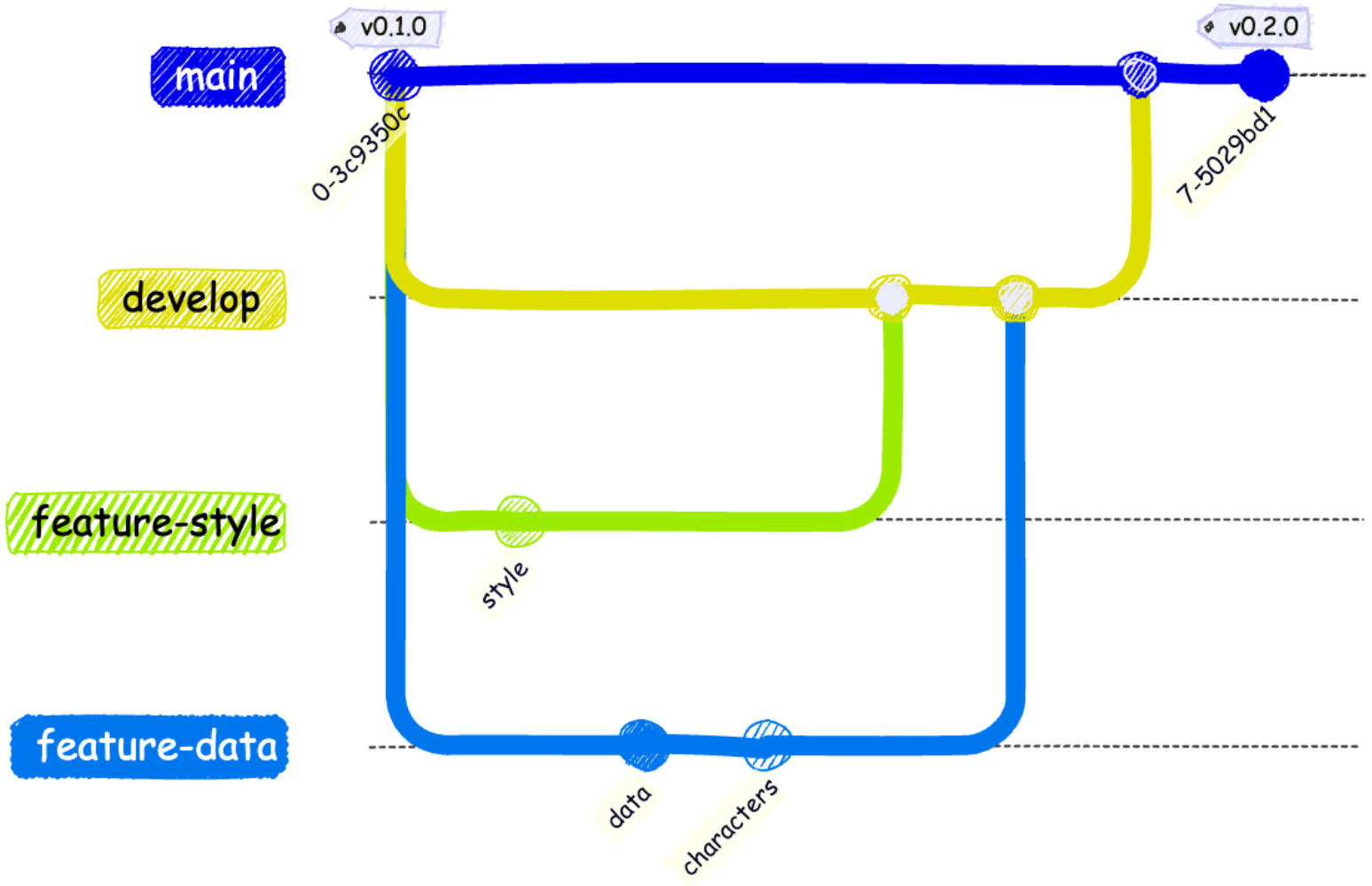
git - Merge (suite)

- 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)



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` et `git push origin develop`
- 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`

Version 0.2.0 - Objectif

L'objectif de cette version était d'améliorer l'affichage de l'application en ajoutant du style CSS et de récupérer des données depuis un fichier JSON.

Ces modifications ont été réalisées en utilisant Git pour travailler sur plusieurs fonctionnalités en même temps, en créant des branches spécifiques pour chaque fonctionnalité.

On pourrait imaginer que ces 2 fonctionnalités ont été développées par 2 développeurs différents , ce qui permet de travailler en parallèle sans impacter le code de production.

Version 0.2.0 - Objectif (suite)

L'utilisation de **Git** permet de gérer les branches et de fusionner les modifications de manière efficace.

Le choix de **Git Flow** permet de structurer le développement en définissant des règles pour les branches, ce qui facilitera la gestion des versions et des fonctionnalités, des correctifs et des releases à l'avenir.