Guide Complet pour Configurer et Tester Ton Projet Git avec GitHub

1. Initialiser et Configurer Git Localement

Assure-toi d'avoir installé Git. Si ce n'est pas encore fait, télécharge et installe Git.

Ensuite, configure ton nom d'utilisateur et ton email pour que tes commits soient identifiés correctement.

```
git config --global user.name "TonNom"
git config --global user.email "ton_email@example.com"
```

Ces informations seront liées à chaque commit, permettant de savoir qui a effectué des changements.

2. Cloner Ton Dépôt GitHub

Si ton dépôt GitHub existe déjà (par exemple, `https://github.com/userr320/football_manager.git`), clone-le pour créer une copie locale :

```
git clone https://github.com/userr320/football_manager.git
```

Ensuite, navigue dans le répertoire du projet cloné :

```
cd football_manager
```

Pour vérifier que le clonage s'est bien passé, utilise :

```
git status
```

3. Structurer le Projet avec des Branches

Pour un développement organisé, crée une nouvelle branche de développement (`develop`). Cette branche te permet de tester et de modifier sans affecter la branche principale (`main`).

```
git checkout -b develop
```

🔹 **Astuce** : Utilise des branches spécifiques aux fonctionnalités que tu développes (par ex., `feature/authentication`, `feature/match-stats`) pour garder chaque tâche isolée. Cela rend la gestion et le suivi des changements plus simple.

Pour vérifier la branche active, utilise :

```
git branch
```

La branche en cours sera marquée par un astérisque `*`.

4. Ajouter et Committer des Changements

Pour tester la configuration, crée un fichier `README.md` avec des informations de base sur le projet. Ajoute ensuite ce fichier pour que Git le suive :

```
echo "# Football Manager Project" >> README.md
git add README.md
```

Vérifie que le fichier est prêt à être commit :

```
git status
```

Puis fais un commit :

```
git commit -m "Ajout d'un fichier README avec description du projet"
```

5. Pousser les Modifications vers GitHub

Pour envoyer les changements de la branche `develop` vers le dépôt distant :

```
git push -u origin develop
```

Le `-u` lie ta branche locale `develop` à la branche distante `develop`, facilitant les prochains `git push`.

Pour les changements futurs, il suffira de faire simplement :

```
git push
```

6. Fusionner une Branche sur la Branche Principale (main)

Une fois ta fonctionnalité terminée et testée sur `develop`, fusionne-la dans `main` :

Basculer sur la branche principale :

```
git checkout main
```

1.

Fusionner les changements de `develop` dans `main` :

```
git merge develop
```

2.

Pousser les modifications de `main` vers GitHub :

```
git push origin main
```

3.

🔹 **Astuce** : Avant chaque fusion, utilise `git pull` pour t'assurer que ta branche `main` est à jour avec le dépôt distant.

7. Récupérer les Modifications d'Autres Contributeurs

Pour synchroniser ta branche locale avec les changements effectués par d'autres contributeurs, utilise `pull` sur la branche `main` :

```
git pull origin main
```

🔹 **Conseil** : Avant d'implémenter des changements majeurs, toujours `pull` les modifications distantes pour éviter les conflits.

8. Utiliser GitHub Desktop pour une Interface Graphique

GitHub Desktop est un outil pratique qui simplifie certaines commandes Git. Télécharge GitHub Desktop et connecte ton compte GitHub. Voici les opérations de base avec GitHub Desktop :

1. **Cloner un dépôt** : Choisis `File > Clone repository` et ajoute le lien de ton dépôt GitHub.
2. **Créer des branches** : Utilise le menu `Branch > New Branch` pour créer une nouvelle branche.

3. **Committer des modifications** : Les changements sont affichés dans la section `Changes`. Ajoute un message de commit et clique sur `Commit to [nom de la branche]`.
4. **Pousser vers GitHub** : Après un commit, clique sur `Push origin` pour synchroniser avec le dépôt distant.
5. **Fusionner des branches** : Dans GitHub Desktop, utilise `Branch > Merge into Current Branch` pour fusionner.

9. Utiliser des Branches Collaboratives et Résoudre des Conflits

Si plusieurs personnes travaillent sur le projet, des conflits peuvent survenir. Pour éviter des conflits majeurs :

1. **Commits fréquents** : Commits réguliers avec des messages descriptifs permettent une meilleure compréhension de chaque modification.
2. **Pull avant un push** : Toujours récupérer les changements du dépôt distant avec `git pull` pour s'assurer de la mise à jour avant d'envoyer les modifications locales.

En cas de conflit, Git marquera les sections du fichier en conflit avec des indications `<<<<< HEAD` et `>>>>>`. Édite les lignes pour conserver les changements souhaités, puis enregistre et committe.

10. Suivre l'Historique des Modifications et Afficher les Logs

Pour voir un historique des commits et comprendre les modifications passées, utilise :

```
git log --oneline
```

Cela affiche un historique simplifié des commits avec leur identifiant unique (SHA) et leur message de commit.

11. Organiser la Structure des Fichiers du Projet

Pour une organisation propre, adopte une architecture MVC (Modèle-Vue-Contrôleur) pour structurer les fichiers :

- **model/** : Contient les fichiers de connexion et de requêtes vers la base de données.
- **view/** : Contient les fichiers HTML/CSS et les vues de l'application.
- **controller/** : Contient les fichiers qui récupèrent les données des modèles et les envoient aux vues.
- **config/** : Stocke les configurations globales (ex : base de données, constantes).
- **public/** : Contient les fichiers accessibles publiquement (assets, images, CSS).
- **.env** : Stocke les informations de configuration privées (ne jamais l'ajouter au dépôt).

Crée des sous-dossiers au besoin pour chaque type de fichier.

12. Gestion des Variables Sensibles avec `.env`

Ajoute un fichier `.env` pour les configurations sensibles (comme les identifiants de base de données) et utilise des bibliothèques comme `dotenv` pour les charger.

Dans `.gitignore`, ajoute `.env` pour éviter qu'il ne soit poussé sur GitHub.