GitHub cheat sheet

GitHub Cheat Sheet

🚀 Structure d’un bon message de commit :

Voici le format standard pour structurer un message de commit clair et utile :

```
<emoji>(<scope>): <sujet>
```

✨ **Détails :**

1. **Emoji** : Identifie rapidement le type de changement (visuellement impactant).
2. **Scope (portée)** : La partie du projet concernée (facultatif, mais recommandé).
3. **Sujet** : Une phrase concise (< 50 caractères) au présent impératif.

🔑 Principaux emojis Gitmoji :

Emoji	Type	Usage
✨	**feat**	Ajout d'une nouvelle fonctionnalité.
🐛	**fix**	Correction d'un bug.
📝	**docs**	Mise à jour ou ajout de documentation.
💄	**style**	Modifications de style sans impact fonctionnel (indentation, espaces).
♻️	**refactor**	Refactorisation du code sans ajout de fonctionnalité ni correction de bug.
⚡	**perf**	Amélioration des performances.
✅	**test**	Ajout ou modification de tests.
📦	**build**	Changements affectant le système de build ou les dépendances.
👷	**ci**	Modifications liées au déploiement ou à l'intégration continue.
⏪	**revert**	Annulation d’un précédent commit.

🔗 Exemples pratiques :

- **Ajout de fonctionnalité :**

  ```
  ✨(product): add product search functionality
  ```

- **Correction de bug :**

```
🐛(cart): fix discount calculation issue
```

- **Amélioration des performances :**

```
⚡(database): optimize query execution time
```

- **Mise à jour de la documentation :**

```
📝(README.md): update installation guide
```

- **Annulation d’un commit précédent :**

```
⏪ revert add caching for better performance
```

- Aout de tests pour un fichier :

```
✅(utils/helpers.js): add unit tests for formatDate function
```

- Mise à jour d'un fichier de configuration :

```
📦(config.yaml): update server configuration for staging
```

🛠️ Commandes de base Git

Voir les fichiers modifiés :

- **Liste des fichiers modifiés** :

```
git status
```

- **Voir les différences dans le code** :

```
git diff
```

Ajouter un fichier modifié à un commit :

- **Ajout du fichier**

```
git add chemin/vers/le/fichier
```

- **Suppression d'un fichier du commit (ajouté par erreur) :**

```
git reset chemin/vers/le/fichier
```

Effectuer et pousser un commit :

- **Création du commit :**

```
git commit -m "<emoji>(<scope>): <sujet>"
```

- **Pousser le commit:**

```
git push
```

Modifier le dernier message de commit :

```
git commit --amend -m "Nouveau message du commit"
```

Si le commit a déjà été poussé :

```
git push --force
```

> **Important**
>
> Un commit doit être spécifique. Il faut éviter, dans un même commit, d'effectuer des changements sur la gestion du panier et corriger un problème d'affichage sur la page principale, par exemple.
> Dès que vous ajoutez ou modifiez quelque chose, faites un commit. Cela permet de créer une sauvegarde de votre travail et de garder une trace de ce que vous avez fait.

🔄 Fonctionnement des Pull Requests (PR)

🚧À mettre à jour 🚧

Création d’une Pull Request (PR)

1. **Créer une branche locale :**

```
git checkout main          # Assurez-vous d’être sur la branche principale
git checkout -b feature/new-feature  # Créer une nouvelle branche
```

2. **Apportez vos modifications et poussez la branche :**

```
git add .
git commit -m "✨(feature): description de la fonctionnalité"
git push origin feature/new-feature
```

3. **Créer une Pull Request sur GitHub :**
   - Allez sur le dépôt GitHub.
   - Cliquez sur **"Compare & pull request"**.
   - Remplissez le titre et la description, puis assignez des reviewers.

Une fois la PR fusionnée :

1. Retournez sur la branche principale :

```
git checkout main
```

2. Mettez à jour votre branche principale :

```
git pull origin main
```

3. Supprimez la branche locale :

```
git branch -d feature/new-feature
```

4. Supprimez la branche distante :

```
git push origin --delete feature/new-feature
```
