Librairies pour les hooks

Temps de lecture : 3 minutes

Plusieurs difficultés à utiliser les `hooks` comme nous l'avons vu dans la leçon précédente se présentent.

La première est qu'il est difficile d'utiliser des librairies effectuant des actions dans certains `hooks`.

La seconde est que le dossier `hook` est dans le dossier local caché `.git`, il n'est donc pas suivi par `Git` et n'est pas partagé avec votre équipe. Cela peut poser problème si vous utilisez des `hooks` pour forcer des bonnes pratiques sur votre dépôt.

Heureusement, il existe des librairies qui vont nous aider à manipuler les `hooks Git` !

La librairie `husky`

Si vous avez fait n'importe quel cours `JavaScript` sur `Dyma` vous savez ce qu'est `npm`.

Il vous suffit d'installer `Node.js` pour l'obtenir.

Dans un projet, il faut faire :

```
npm init -y
```

Initialisez le suivi Git :

```
git init
```

Pour créer le `package.json` et permettre l'installation de `packages`.

Pous pouvez ensuite faire :

```
npx husky-init && npm install
```

Cela installera la librairie `husky` comme dépendance de développement.

`Husky` permet d'utiliser tous les `hooks Git` très facilement.

Utiliser vos `scripts` avec `Husky`

Comment faire pour utiliser vos `scripts` avec `Husky` ?

Nous allons reprendre notre exemple de la leçon précédente en le modifisant pour qu'il soit utilisable avec `sh`.

Modifiez le fichier `.husky/pre-commit` créé par Husky et mettez :

```
#!/usr/bin/env sh
. "$(dirname -- "$0")/_/husky.sh"

exec < /dev/tty
if test $(git diff --cached | grep ^+.*console.log | wc -l) != 0
then
    echo "Au moins un console.log est ajouté dans votre commit !"
    echo "Êtes-vous certain de continuer ?"
    read -p "[o/n]>" choix
    if test $choix != "o"
    then
        echo "Abandon du commit"
        exit 1
    fi
fi
```

Le `script` s'exécutera maintenant avant chaque `commit` !

A noter que ce n'est pas utilisable avec la création de `commit` dans `VS Code` car celui-ci n'utilise pas de `shell` et n'ouvre pas de terminal. Il ne peut donc pas demander d'`input` à l'utilisateur.

Une version compatible, qui ne demande pas d'`input` aux utilisateurs serait :

```
# Check
if test $(git diff --cached | grep ^+.*console.log | wc -l) != 0
then
    echo "Console.log détecté dans votre commit"
    exit 1
fi
```

Utiliser la librairie `commitlint`

La librairie `commitlint` est une librairie forçant l'utilisation de conventions pour la rédaction des messages de validation pour les `commits`.

Nous vous conseillons de conserver la convention par défaut qui est utilisée par de très nombreuses équipes et a été créée par l'équipe Angular de Google.

1 - Elle oblige à que les messages soient de la forme `type: titre`.

Les types possibles sont :

```
build
chore
ci
docs
feat
fix
perf
refactor
revert
style
test
```

`build` : `build` de l'application pour la mise en production.

`chore` : ne touche pas au code de production mais des configurations d'outiles : `Webpack` etc.

`ci` : relatif à l'intégration continu / déploiement continu.

`docs` : mise à jour de la documentation.

`feat` : nouvelle fonctionnalité (`feature`).

`fix` : correction de bug.

`perf` : optimisation des performances.

`refactor` : refactorisation, pas de nouvelle fonctionnalité.

`revert` : retour en arrière (avec `git revert`).

`style` : modifications uniquement de mise en page.

`test` : ajout / modification de tests.

2 - Elle oblige à mettre les message de validation en minuscules uniquement.

3 - Elle oblige a mettre un message de titre de 100 caractères maximum (texte passé à la première utilisation de l'option `-m`).

Pour installer la librairie, il suffit de faire :

```
npm install --save-dev @commitlint/{cli,config-conventional}
```

Ensuite ouvrez un terminal et faites :

```
echo "module.exports = { extends: ['@commitlint/config-conventional']
};" > commitlint.config.js
```

Cela va créer le fichier de configuration pour la convention utilisée par `commitlint`.

Enfin installez le `hook` avec Husky en faisant :

```
npx husky add .husky/commit-msg  'npx --no -- commitlint --edit ${1}'
```

Utiliser `prettier` dans un `hook`

Nous avons déjà vu dans tous les cours la librairie de mise en forme du code `Prettier`.

Vous pouvez utiliser un `hook` afin de reformater le code systématiquement avant un `commit`.

```
npm install --save-dev prettier pretty-quick
```

Enfin installez le `hook` avec Husky en faisant :

```
npx husky set .husky/pre-commit "npx pretty-quick --staged"
```

Vous avez maintenant une bonne base pour utiliser les `hooks Git` dans votre équipe !