Guide des bonnes pratiques pour l'utilisation de Git

Ce guide a pour but de fournir des principes simples pour une utilisation efficiente de Git. Les recommandations qu'on y trouve vous permettront d'assurer une saine gestion de votre projet et faciliteront grandement la gestion de votre code.

Principes fondamentaux

Commençons par cinq principes fondamentaux:

1) **Faites des commits propres qui ne concernent qu'une seule chose.**

> Très souvent, on fait plusieurs changements de natures différentes dans une session de travail. Il ne faut pas les regrouper en un seul commit, mais plutôt répartir en plusieurs commits distincts. On saura ainsi plus facilement ce qui a été fait, puisque la description associée à chaque commit sera simple (et ne concernera qu'une chose), et si on a besoin de revenir en arrière, on pourra choisir de défaire le commit qui concerne le problème spécifique, plutôt que de tout défaire ce qu'on a fait dans la session de travail.

2) **Pour chaque commit, écrivez une description concise qui décrit exactement les changements apportés.**

> Ce principe est très important et souvent négligé par le développeur trop pressé ou insouciant. Comment pensez-vous qu'on peut se retrouver en cas de problème si la description du commit est quelque chose comme "du ménage pour corriger les conneries de Paul"? Il est suggéré d'écrire une description de la forme suivante:
>
> ```
> type: description du changement apporté
> ```
>
> Voici des exemples de types souvent utilisés:
>
> | | |
> |---|---|
> | tâche/chore: | pour un travail de routine |
> | doc: | ajout de documentation (commentaires) |
> | fonction/feature: | travail sur une fonctionnalité |
> | test: | ajout ou modification de test |
> | style: | modifications cosmétiques au code |
> | réusinage/refactor: | amélioration du code (sans changer la fonctionnalité) |
> | débogage/fix: | correction de bogue |

3) Faites des commits très fréquemment.

Git est un excellent système de gestion des versions, profitez-en! N'attendez pas d'avoir écrit beaucoup de code avant de faire un commit. Dès que vous avez quelque chose qui fonctionne raisonnablement (mais pas nécessairement complètement), faites un commit. Décomposez votre session de travail en très petites étapes significatives, et faites un commit après chacune. Par exemple, chaque fois qu'on avance un peu dans le développement d'une fonctionnalité, on s'assure qu'un nouveau test passe, et on fait un commit.

4) Ne modifiez pas l'historique des commits qui ont été publiés

Ce principe vous évitera de gros maux de tête. Si vous changez l'historique des commits qui sont visibles par tous, vous risquez de causer des problèmes lorsqu'un développeur voudra faire une fusion avec son entrepôt local. Donc, pensez-y bien avant d'utiliser une commande qui change l'historique des commits (comme rebase), et si vous le faites, faites-le seulement sur une branche locale.

5) Ne faites pas de commit d'un fichier qui est généré automatiquement

Nous disposons aujourd'hui de beaucoup d'espace mémoire, mais il ne faut pas exagérer! On veut conserver ce qui demande beaucoup d'effort à produire (le code tout particulièrement). Il n'y a absolument aucun intérêt à stocker un fichier qui est généré automatiquement par l'exécution d'un programme. Cela occupe inutilement de l'espace (surtout que ces fichiers sont parfois très gros).

Utilisation des branches

1) Créez une nouvelle branche dans les cas suivants:

- Vous corrigez un problème majeur
- Vous développez une nouvelle fonctionnalité
- Vous faites du code expérimental

Ainsi, la branche `master` ne contiendra essentiellement que des commits qui sont le résultat d'une fusion avec une branche secondaire.

2) Lorsque vous travaillez sur une branche secondaire, faites fréquemment un `rebase` avec la branche `master`. En gros, exécutez souvent la séquence de commandes suivante:

```
git checkout master
git pull
```

```
git checkout <branche secondaire>
git rebase master
```

De cette manière, vous vous assurez que votre travail ne sera pas désynchronisé avec celui de vos collègues, et vous éviterez une fusion cauchemardesque lorsque votre travail sur la branche secondaire sera terminé.

3) Tous les commits sur la branche `master` correspondent à du code propre qui fonctionne bien.

4) Donnez des noms significatifs à vos branches.

5) Pour une gestion simple et efficace des branches, inspirez-vous de celle proposée par GitHub.

Autres bonnes pratiques

1) Si vous déplacez ou renommez un fichier, assurez-vous de faire un commit avant de recommencer à modifier ce fichier.

2) Ne faites pas de commit d'un fichier de configuration.

3) Ne faites pas de commit d'un gros fichier binaire.

4) Soyez très prudent avec la commande `reset`, qui peut détruire des fichiers de manière définitive.