Comment nommer les branches avec Git

Traduction rapide de la page :
http://stackoverflow.com/questions/273695/git-branch-naming-best-practices

Voici quelques conventions de nommage de branches :

Conventions de dénomination des branches

1. Utilisez le regroupement par mots au début de vos noms de branche.
2. Définir et utiliser des mots courts et fort de sens afin de différencier les branches d'une manière qui soit utile à votre flux de travail.
3. Utilisez des barres obliques (/) pour séparer les éléments de vos noms de branche.
4. Ne pas utiliser de numéros nues comme premier élément.
5. Évitez les longs noms descriptifs pour les branches à long terme.

Exemple de gestion de nom de branches

Utiliser des mots “de regroupement” pour chaque élément vos noms de branche.

```
group1/foo
group2/foo
group1/bar
group2/bar
group3/bar
group1/baz
```

Les groupes peuvent être nommés comme vous le souhaitez en fonction de votre flux de travail.

Jetons court bien définis

Choisissez jetons courts afin qu'ils n'ajoutent pas trop de bruit à chacun de vos noms de branche. Exemples :

- wip : (Works in progress) Travaux en cours ; choses que je sais ne sera pas bientôt fini
- feat : Fonctionnalité en cours d'ajout ou de complétion
- bug : Résolution de bug
- junk : Branche jetable pour réaliser des éxpérimentations

Chacun de ces jetons peuvent être utilisés pour vous rappeler à quelle partie de votre flux de travail chaque branche appartient.

Si plusieurs branches sont utilisées pour chaque cycle de travail, et supposons qu'ils sont nommés «new», «testing» et «verified». Il est possible de nommer les branches avec des versions abrégées de ces termes, toujours orthographié de la même façon, afin de les regrouper et de se rappeler l'étape actuelle du cycle.

```
new/frabnotz
new/foo
new/bar
test/foo
test/frabnotz
ver/foo
```

Vous pouvez rapidement dire quelles branches ont atteint quelle étape et vous pouvez les regrouper facilement en utilisant les options de correspondance de motif de Git.

```
$ git branch --list "test/*"
test/foo
test/frabnotz

$ git branch --list "*/foo"
new/foo
test/foo
ver/foo

$ gitk --branches="*/foo"
```

Utilisez des slashs pour séparer les éléments

You may use most any delimiter you like in branch names, but I find slashes to be the most flexible. You might prefer to use dashes or dots. But slashes let you do some branch renaming when pushing or fetching to/from a remote.

Vous pouvez utiliser la plupart des délimiteur que vous aimez dans les noms de branches, mais les slachs sont bien plus flexibles. Vous préférerez peut-être utiliser des tirets ou des points. Mais les slachs vous permettent de faire un peu de changement de nom de branche en les poussant ou les récupérant vers / depuis un dépôt distant.

```
$ git push origin 'refs/heads/feature/*:refs/heads/phord/feat/*'
$ git push origin 'refs/heads/bug/*:refs/heads/review/bugfix/*'
```

For me, slashes also work better for tab expansion (command completion) in my shell. The way I have it configured I can search for branches with different sub-parts by typing the first characters of the part and pressing the TAB key. Zsh then gives me a list of branches which match the part of the token I have typed. This works for preceding tokens as well as embedded ones.

Les slashs fonctionnent mieux avec la touche tab servant à l'auto-complétion dans le shell. Il est possiblede le configuré pour rechercher les branches avec les différentes sous-parties en tapant les premiers caractères de la partie et en appuyant sur la touche TAB. Zsh donne ensuite une liste de branches qui correspondent à la partie du jeton tapé. Cela fonctionne pour les jetons précédents autant que pour ceux intégrés.

```
$ git checkout new<TAB>
Menu:  new/frabnotz   new/foo   new/bar
```

```
$ git checkout foo<TAB>
Menu:  new/foo   test/foo   ver/foo
```

(Zshell permet de facilement configuré la commande d'auto-complétion et il est aussi possible de le faire pour les deux points, les underscrores ou les points.)

Il permet également de rechercher des branches dans de nombreuses commandes de Git, comme ceci:

```
git branch --list "feature/*"
git log --graph --oneline --decorate --branches="feature/*"
gitk --branches="feature/*"
```

Avertissement : Comme Slipp souligne dans les commentaires, les barres obliques peuvent causer des problèmes. Parce que les branches sont utilisées comme des chemins, vous ne pouvez pas avoir une branche nommée «foo» et une autre branche nommée «foo / bar“. Cela peut être déroutant pour les nouveaux utilisateurs.

N'utilisez pas de numéros nus

Do not use use bare numbers (or hex numbers) as part of your branch naming scheme. Inside tab-expansion of a reference name, git may decide that a number is part of a sha-1 instead of a branch name. For example, my issue tracker names bugs with decimal numbers. I name my related branches CRnnnnn rather than just nnnnn to avoid confusion.

N'utilisez pas de numéros nus (ou des nombres hexadécimaux) dans le cadre de votre convention de nommage de branches. Lors de l'auto-complétion d'un nom de référence, Git peut décider que ce nombre est une partie d'un sha-1 à la place d'un nom de branche.

```
$ git checkout CR15032<TAB>
Menu:   fix/CR15032    test/CR15032
```

En essayant d'étendre simplement 15032, Git ne pourrait pas savoir si on cherche un nom de SHA-1 ou de branche.

Évitez les longs noms descriptifs

Long branch names can be very helpful when you are looking at a list of branches. But it can get in the way when looking at decorated one-line logs as the branch names can eat up most of the single line and abbreviate the visible part of the log.

On the other hand long branch names can be more helpful in “merge commits” if you do not habitually rewrite them by hand. The default merge commit message is Merge branch 'branch-name'. You may find it more helpful to have merge messages show up as Merge branch 'fix/CR15032/crash-when-unformatted-disk-inserted' instead of just Merge branch 'fix/CR15032'. share|improve this answer

edited Mar 10 '13 at 22:18 answered May 19 '11 at 23:25 phord 2,0931916