Introduction à Git

Club Robotronik Phelma

10 Décembre 2015

1 Qu'est-ce que Git ?
- Un gestionnaire de versions
- À quoi peut ressembler un historique sur Git ?

2 Commandes basiques
- Passons à la pratique !

3 Le fichier .gitignore

4 La cachette (stash)
- La cachette (stash)

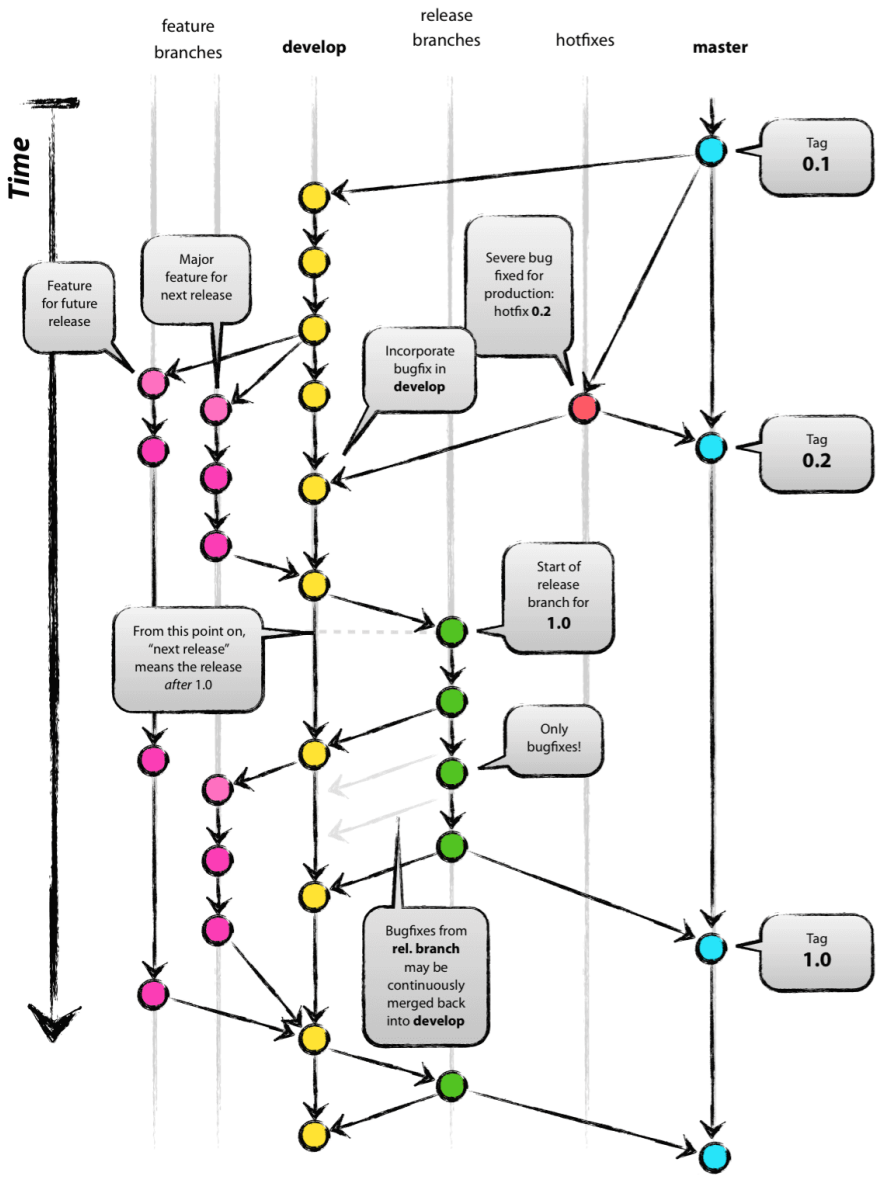
Qu'est-ce que Git ?

Un gestionnaire de versions

- Faire des test sans crainte
- Conserver une version toujours fonctionnelle
- Travailler en commun
- **Fusionner des modifications automatiquement**
- Travailler chacun dans son coin (train. . .) sans risque de conflits à l'arrivée

- Utilisation basique très simple à apprendre
  - `git clone https://github.com/robotronik/git-initiation`
  - `git add / rm <fichiers>`
  - `git commit -m "message"`
  - `git pull / push`
  - Création de branches
- Utilisation avancée plus complexe mais très puissante
  - `git merge / rebase`
  - `git filter-branch`
  - Retourner dans le passé et réutiliser d'anciennes modifications
  - `git --prune` (soyons fous)

À quoi peut ressembler un historique sur Git ?



Passons à la pratique !

- Récupérons le dépôt
  ```
  git clone https://github.com/robotronik/git-initiation
  ```
- Regardons son état et son historique
  ```
  git status
  git log --graph --all
  ```

Personnalisation de Git

```
# Nécessaires :
git config --global user.name "<votre nom>"
git config --global user.email <votre_adresse@mail>
# Non nécessaires :
git config --global core.editor <éditeur_préféré>
git config color.ui auto
# (permet d'avoir une coloration dans le terminal, pas de base à
phelma)
```

Création d'une branche

Cela permet de travailler sur une fonctionnalité sans se soucier des autres.

- Basculer sur une branche déjà existante :
  ```
  git checkout felix/localisation
  ```
- Créer une nouvelle branche :
  ```
  git checkout -b moi1A/ma_branche
  ```
- Lister les branches :
  ```
  git branch -a
  ```

Création et modification de fichiers

On va ajouter ou modifier des fichiers du dépôt.

- Signaler à git les modifications :
  ```
  git add monfichier
  ```
  on peut faire "`git add *`" mais peu recommandé
  on peut faire "`git add .`" si le fichier n'est pas nouveau mais modifié
- On "commit" les modifications :
  ```
  git commit -m "j'ai ajouté un fichier"
  ```
  on peut faire "`git commit -am`", si il n'y a pas de modifications
  (et alors pas besoin de git add)
- On peut corriger le précédent commit (tant qu'il est en local) :
  ```
  git commit --amend
  ```

Les messages de commit

Ils sont d'une importance capitale !
Ils nous permettent de retrouver 6 mois plus tard... Comment on a résolu ce bug déjà ?
Ils doivent être courts (= une ligne = 80 caractères), mais si il faut détailler, on peut dépasser :D

Travailler avec les autres

- Envoyer ses commits :
  ```
  git push
  ```
- Récupérer les commits distants :
  ```
  git pull
  ```
- Ceci va merger (fusionner), il est possible de juste récupérer sans merger :
  ```
  git fetch
  ```
- On peut ensuite merger à la main :
  ```
  git merge
  ```

Le fichier .gitignore

- Permet d'ignorer des fichiers
- Très utile pour les fichiers générés ou la configuration des logiciels
- Est présent à la racine de chaque dépôt

La cachette (stash)

Elle permet de "cacher" les modifications de git (les déplacer dans un coin). Comme ça on peut pull, merge, etc, puis *ensuite* les ressortir et gérer les conflits (eh oui :(. . .)

- "Cacher" les modifications
  ```
  git stash
  ```
- Ressortir les modifications
  ```
  git stash pop
  ```
- Détruire le stash
  ```
  git stash drop
  ```

La cachette (stash)

Elle permet de "cacher" les modifications de git (les déplacer dans un coin). Comme ça on peut pull, merge, etc, puis *ensuite* les ressortir et gérer les conflits (eh oui :(. . .)

- "Cacher" les modifications
  ```
  git stash
  ```
- Ressortir les modifications
  ```
  git stash pop
  ```
- Détruire le stash
  ```
  git stash drop
  ```

À vous de jouer !

C'est tout pour aujourd'hui !
N'hésitez pas à poser vos questions aux anciens ;)