Introduction à Git

Sommaire

+ GIT : C’est quoi ? un peu d’histoire, les concepts…

+ Les commandes GIT

+ GITFLOW

1 Git

C’est quoi ? Un peu d’histoire, les concepts

Organisation des sources

- Git est un logiciel libre créé par Linus Torvalds en 2005
- Git est un logiciel de gestion de version (VCS en anglais)
- Git est un logiciel de gestion de version distribué / décentralisé.

© SQLI Group 2015 - Git : Organisation des sources

- Git propose une solution de « commit » local puis de « push » sur un repository central

- Git propose une gestion de branche rapide et simple*

© SQLI Group 2015 - Git : Organisation des sources

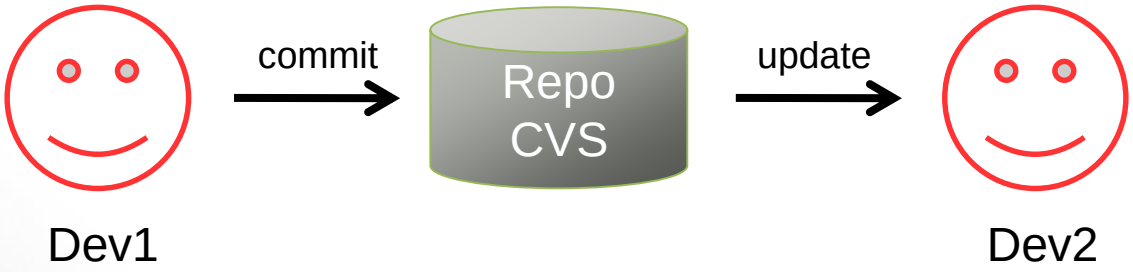
- CVS (Concurrent versions system) : 1990
- SVN (Subversion) : 2000

- CVS et SVN travaillent sur un concept client/serveur.

Le dépôt est forcement une instance spécifique sur un serveur centralisé.

© SQLI Group 2015 - Git : Organisation des sources

Modèle CVS



- Git travaille sur un concept distribué

Il est possible d'avoir un dépôt central mais ce n'est pas une obligation.

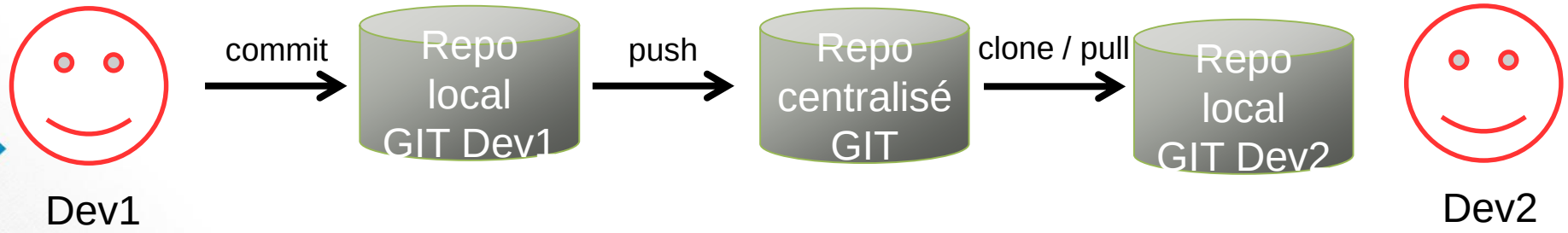
- Git permet donc de « commiter » sur son dépôt local.

L'action de mise à disposition sur un dépôt centralisé est une action supplémentaire à CVS.

C'est le « push ».

© SQLI Group 2015 - Git : Organisation des sources

Modèle GIT.



- Clone : permet de cloner un dépôt vers un dépôt local
- Pull : permet de récupérer les modifications d'un dépôt central

© SQLI Group 2015 - Git : Organisation des sources

- Quelques avantages de GIT par rapport à CVS
  - Historique conservé sur les fichiers renommés
  - Rapidité
  - Facilité de création et de gestion des branches

2 Les commandes Git

La base…

Organisation des sources

Initialiser son dépôt distant :

```
mkdir monProjet
cd monProjet
git init
touch README
git add README
git commit -m "Initialisation du dépôt"
git remote add origin https://urlDeMonRepoDistant/...
git push -u origin master
```

- Cloner un dépôt distant existant :

```
git clone https://urlDeMonRepoDistant/...
```

Etat du repository :

```
# Etat de la branche courante du dépôt local
git status

# Liste les branches
git branch

# Télécharge les informations de mon dépôt distant
git fetch monRemote

# Liste les remotes disponible sur mon dépôt local
git remote -v
```

© SQLI Group 2015 - Git : Organisation des sources

Créer sa branche :

```
# Je créé la branche nouvelleBranche
git branch nouvelleBranche

# Je me positionne sur la branche nouvelle branche
git checkout nouvelleBranche
```

```
# Je créé et me position sur la branche nouvelleBranche
git checkout -b nouvelleBranche
```

© SQLI Group 2015 - Git : Organisation des sources

Mes premiers commits :

```
# Ajout de monFichierACommiter au “Stagging”
git add monFichierACommiter

# Commit du “Stagging”
git commit -m "Message de commit"

# Si le fichier monFichierACommiter à déjà référencer
#    dans le dépôts, on peux le faire en une commande
git commit -am "Message de commit"

# Mon message de commit ne me plais pas
git commit --amend -m "Mon joli message de commit"
```

Pousser mes commits sur les dépots distants :

```
# Push mes commits sur mon dépot distant
git push origin maBranche
```

Mettre à jour ma branche depuis le dépôt distant :

```
# Pull les commits de maBranche du dépot distant
git pull origin maBranche
```

Créer un tag :

```
# Je me positionne sur la branche Master
git checkout master

# Je créé un tag 1.0
git tag -a 1.0 -m "Version stable 1.0"

# Je pousse mon tag sur mon repository distant
git push origin 1.0
```

© SQLI Group 2015 - Git : Organisation des sources

Supprimer branche/tag local :

```
# Suppression Tag
git tag -d tagASupprimer

# Suppression Branche
git branch -d brancheASupprimer
```

Supprimer branche/tag distant :

```
#
git push origin :tagASupprimer

#
git push origin :brancheASupprimer
```

Autres commandes :

```
# Supprime un fichier du dépôt
git rm

# Annulation de commmit
git reset
# ou
git revert monHashDeCommit


# Application d'un commit
git cherry-pick monHashDeCommit
```

© SQLI Group 2015 - Git : Organisation des sources

Autres commandes :

```
# Différence [de monFichier] depuis le dernier commit
git diff [monFichier]

# Listing des derniers commits
git log

# Listing des derniers commits ( avec l'affichage du diff )
git log -p

# Listing des derniers commits ( avec un résumé court )
git log --stat
```

Mettre à jour les sources de sa branche :

```
# J'ai développé et commité sur ma branche locale
# Mais l'origine de ma branche (le master par exemple)
#   a également évolué !

# Je me position sur la branche dont je viens
#   et qui n'est plus a jour
git checkout master
git pull # ou # git pull origin master

git checkout brancheAMettreAJour
git rebase master brancheAMettreAJour
```

© SQLI Group 2015 - Git : Organisation des sources

Existe également : merge.

Le Merge va créer un commit supplémentaire de merge.

Alors que le rebase conserve les commits existants en repartant de l'initialisation de la branche puis en appliquant chaque commit un par un.

© SQLI Group 2015 - Git : Organisation des sources

Mettre de coté le code en cours :

```
git checkout brancheDeRecette
# Je dev quelques corrections mineures...
…
# Urgence absolue !
# Correction à faire et à pousser sur le serveur de
recette.
git stash
# Je fais mon correctif urgent
…
git commit -am "Correction urgente ! "
git push
git stash apply
# Je continue des corrections mineures.
…
```

Créer une branche à partir de modifications (1/3):

```
# Je dev sur le master (alors que j'aurai du créer une branche)
git checkout master
git commit -am "Je commit sur le master"
…
# Je push mon commit sur le dépôts distant

git push
# => Erreur ma branche master est protégée en écriture !
```

© SQLI Group 2015 - Git : Organisation des sources

Créer une branche à partir de modifications (2/3) :

```
# Si je veux revenir au commit précédent
#   et conserver les modifications de fichiers
git reset --soft HEAD^

# Si je veux revenir à l'avant dernier commit
#   et que je ne veux pas conserver les modifications de fichiers
git reset --hard HEAD~1 # HEAD~2 pour avant-avant-dernier …

# Si je veux revenir a un commit particulier
#   et que je ne veux pas conserver les modifications de fichiers
git reset --hard monHashDeCommit
```

© SQLI Group 2015 - Git : Organisation des sources

Créer une branche à partir de modifications (3/3) :

```
# Si j'ai conservé mes fichiers
# Je créé ma nouvelle branche et me positionne dessus
# Et je re-commite sur ma nouvelle branche
git checkout -b newFeature

git commit -am "Je commit sur la branche newFeature"
...
```

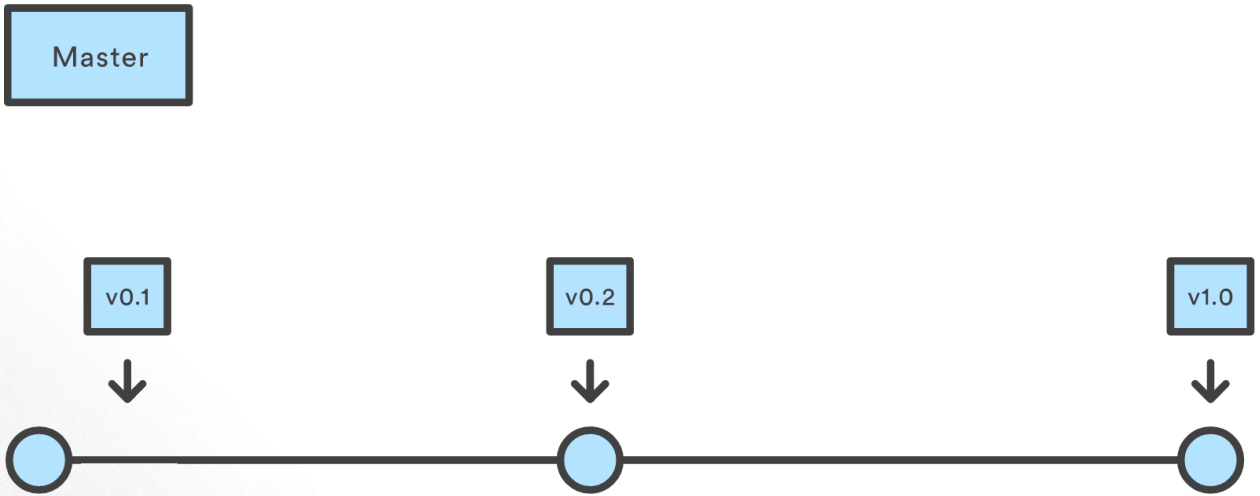
© SQLI Group 2015 - Git : Organisation des sources

3 GitFlow

Les bonnes pratiques d’utilisation de GIT

Organisation des sources

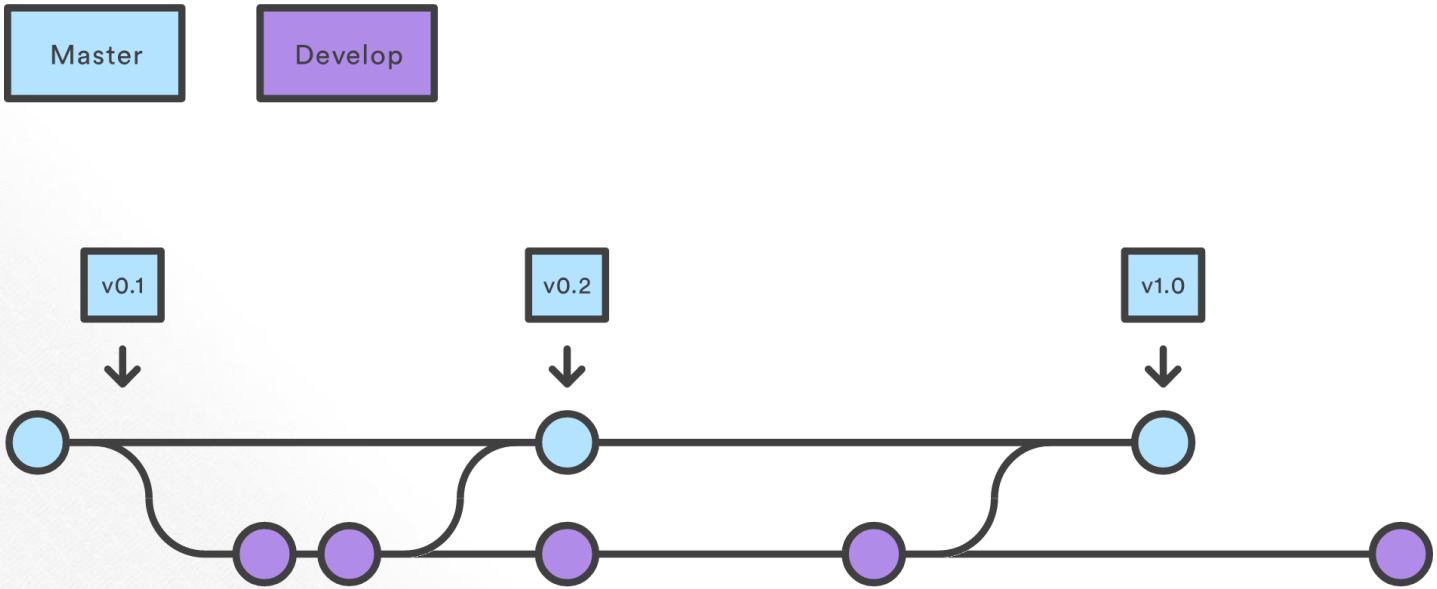
- GitFlow est une proposition de workflow pour la gestion des branches proposée en 2010 par Vincent Driessen.
- Cette organisation est utilisable quelle que soit la taille de l'équipe et l'avancée du projet.
- Dans un projet Git Traditionnel, il n'y a qu'une branche master qui porte l'ensemble des développements.



GitFlow

Et si on gardait une branche à l'identique de la production ?

- Dès qu'on veut passer un projet en production, se pose la question de pouvoir intervenir rapidement alors qu'on a des nouveaux développements en cours.
- GitFlow répond à ce problème en ajoutant une branche « develop » qui portera les nouveaux développements jusqu'au jour de la mise en production, où elle sera « mergée » dans master.

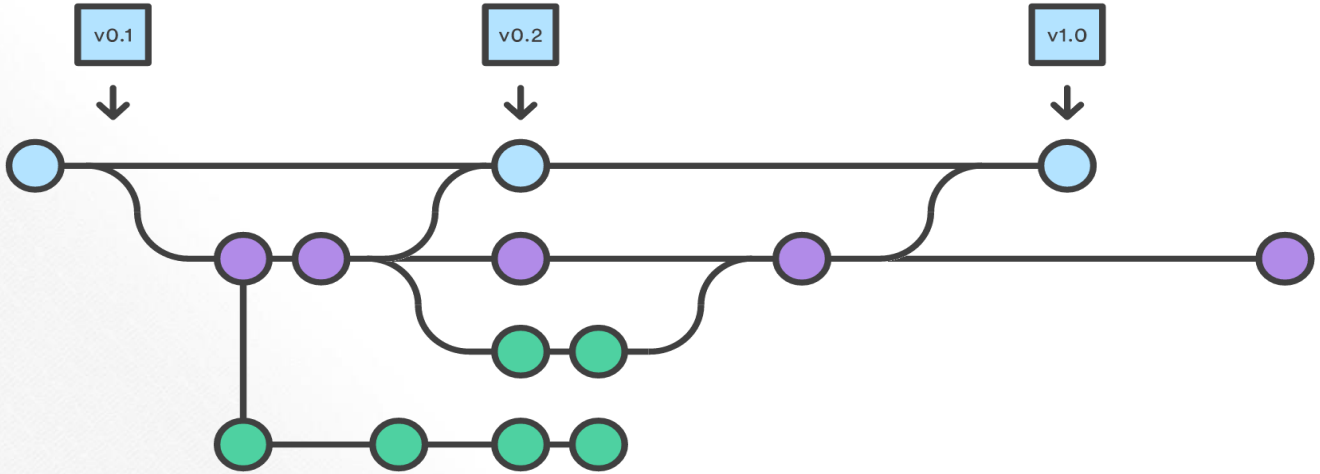


© SQLI Group 2015 - Git : Organisation des sources

GitFlow

Et si on travaille sur plusieurs sujets en même temps ?

- Pour pouvoir passer d'un sujet à l'autre sans laisser des développements non finalisés sur develop, chaque nouvelle fonctionnalité aura sa propre branche feature et ne sera « mergée » dans develop que lorsque la fonctionnalité est prête à être testée.

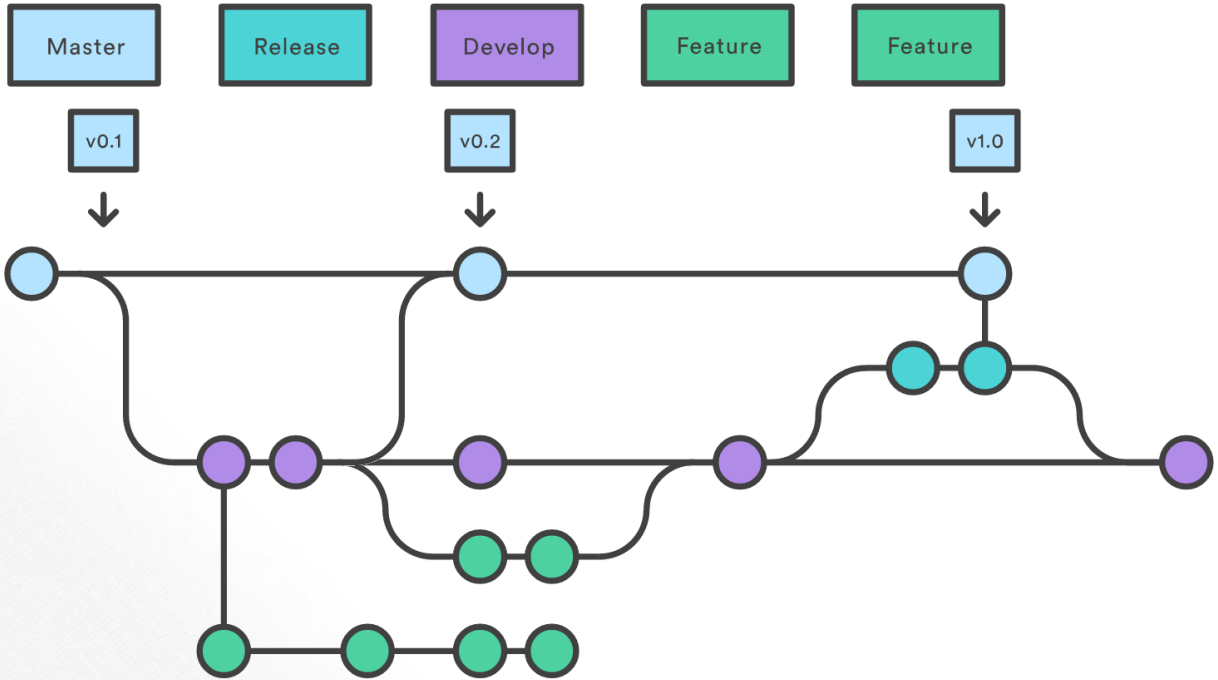


© SQLI Group 2015 - Git : Organisation des sources

GitFlow

Et si on stabilisait avant de passer les nouveautés en production ?

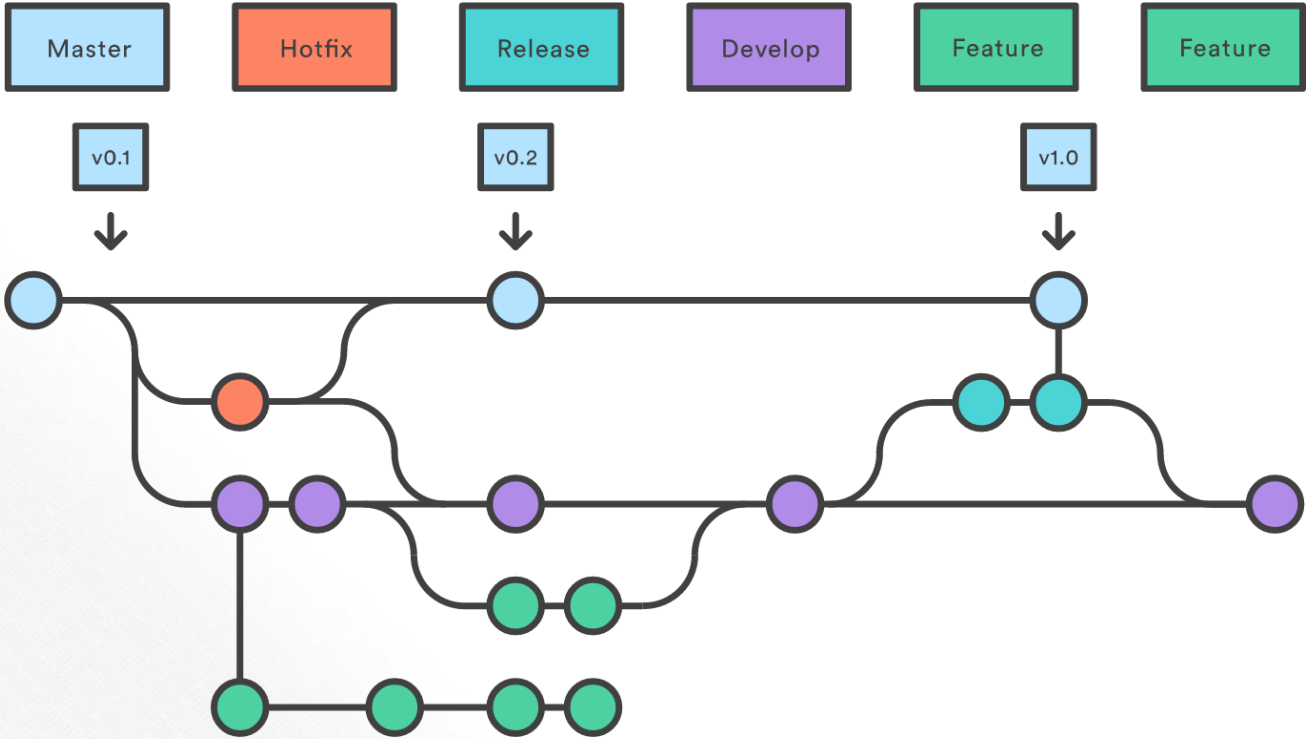
- Pour pouvoir faire la recette des nouvelles fonctionnalités par les key-users avant de les passer en production et accepter leurs corrections, on va tirer une branche release depuis develop pour pouvoir traiter les anomalies tout en n'empêchant pas les développeurs de publier de nouvelles releases dans develop.



© SQLI Group 2015 - Git : Organisation des sources

Et si on a une correction urgente à faire en production ?

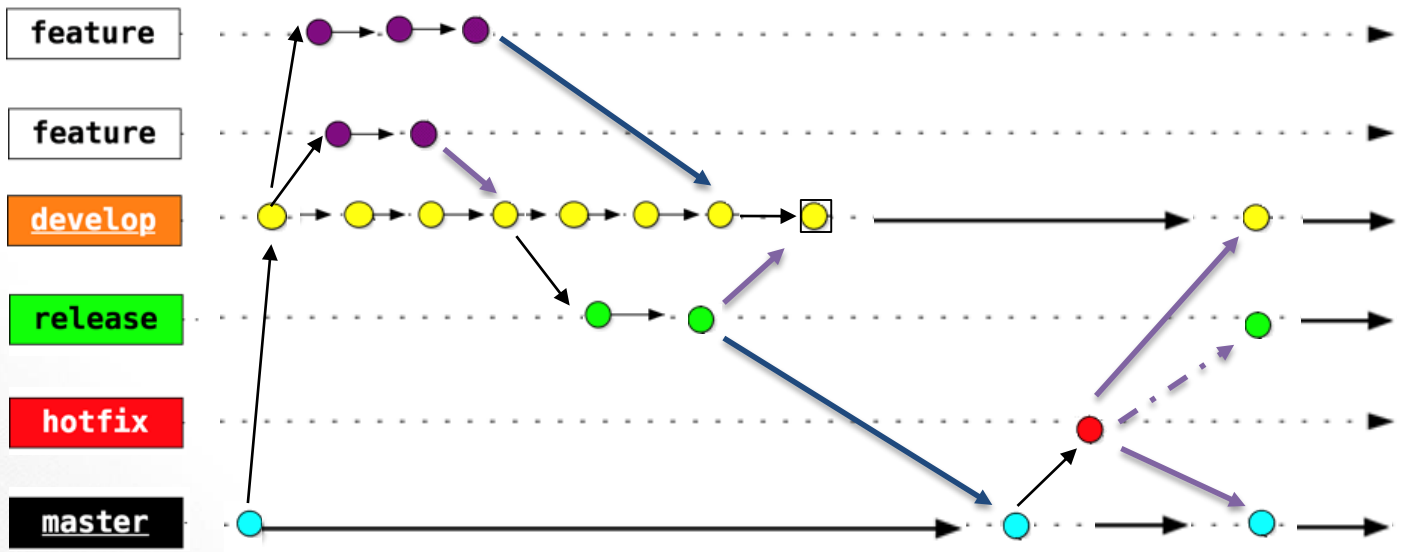
- Les corrections urgentes de la production auront un cycle de vie court et sur une branche unique hotfix.
  La branche sera tirée depuis master et réintégrée dès la validation de la correction.



Heu, vraiment ? Tout ça ? On a pas besoin de tout ça nous !

- GitFlow est un modèle qui est flexible.
- Il n'est pas forcement nécessaire d'utiliser toutes ces typologies de branches.
- Il faut savoir ce que vous voulez utiliser et ce qui semble pertinent pour votre projet.

© SQLI Group 2015 - Git : Organisation des sources



© SQLI Group 2015 - Git : Organisation des sources