Eliot Ragueneau
Matthieu Bouquet

Cours de programmation avancée
-
Git-Flow

Table des matières

1 Git (Rappels)

1.1 L'intérêt

Git, et par extension son hébergeur le plus connu, GitHub, vous sera utile car :

- Il permet de collaborer sur des projets
- Il est une fenêtre sur votre travail pour votre futur employeur [1]
- Il permet de sauvegarder votre dur labeur en cas de pannes matérielles ou logicielles [2]
- Il prends moins de place que ses concurrents VCS, d'où sa popularité, voir son monopole

1.2 La théorie

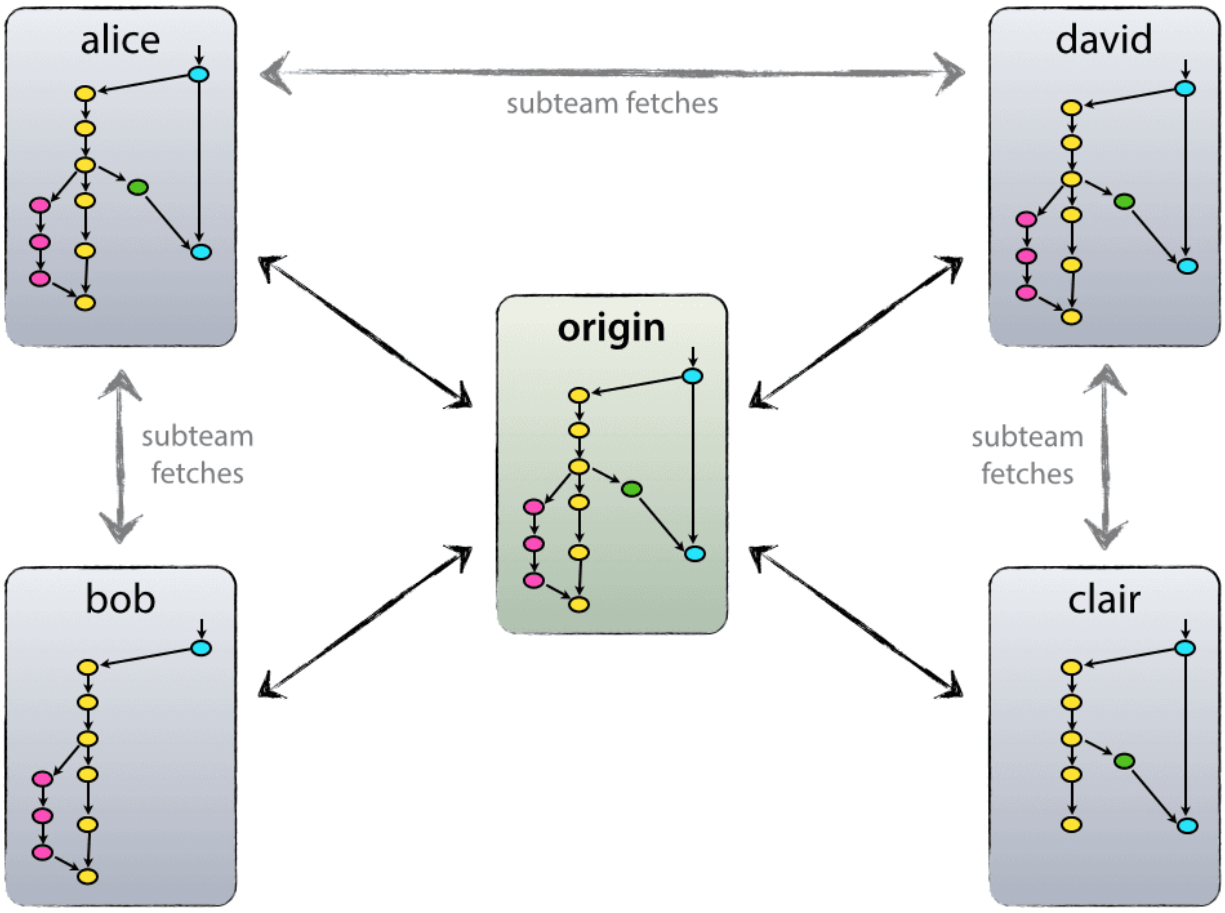

FIGURE 1 – Décentralisé & Centralisé

Git repose sur un serveur (nommé `Remote Repository` et symbolisé par le préfixe origin dans le nom des branches) qui va héberger votre code.

Chaque collaborateur du projet va alors avoir un clone du code contenu sur le serveur, et l'entreposer localement sur son working directory. Cela se fait à l'aide de la commande

```
git clone url_du_github
```

Quand un collaborateur va modifier sur son clone local du code, il va désynchroniser l'état d'avancement de sa version locale avec la version remote.

1. Veillez donc à ce qu'il ne soit pas trop dégueux ! Surtout qu'il y a une trace de tout l'historique !
2. Parce que vous n'avez pas envie de perdre tout votre projet parce que vous disque dur à crash

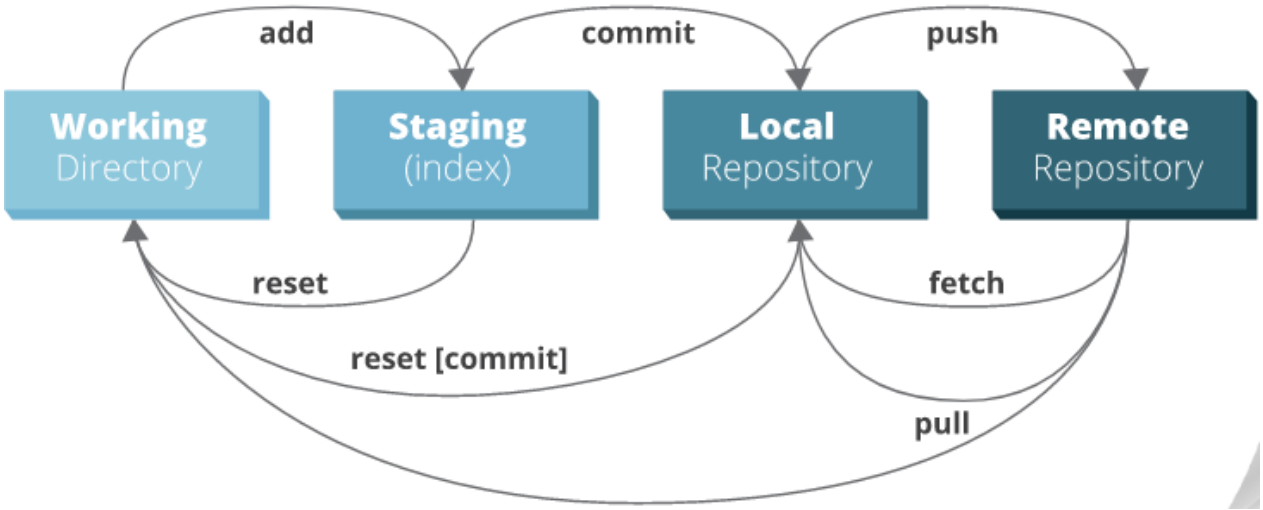

FIGURE 2 – Git theory

Afin de resynchroniser, il faut tout d'abord indexer à git tous les fichiers qu'il doit traiter, et donc toutes les modifications locales que vous souhaitez transmettre au `Remote Repository`, et ce à l'aide de la commande

```
git add fichier
```

On dit qu'un fichier est indexé quand il est "staged", l'action d'indexer peut donc se dire "stage file" ou plus simplement "add file"

La commande

```
git add .
```

permet de directement indexer tous les fichiers qui ont été modifiés dans le working directory.

Ensuite, il faut archiver tous les changements que vous avez réalisé sur les fichiers indexés, et ce à l'aide de la commande

```
git commit -m "Le message de commit"
```

Les changements sont ainsi archivés dans ce qu'on appel le `Local Repository`.

Il suffit alors de "push" tous ces commits pour qu'ils passent du `Local Repository` au `Remote Repository` et donc qu'ils soit mis à disposition de tous les autres collaborateurs mais également d'un potentiel publique. Ceci se fait grâce à la commande

```
git push
```

Si le `remote server` est en avance sur votre `Local Repository` et donc votre working directoy, c'est à dire si un autre collaborateur a push de nouveaux commit sur le serveur, alors il vous faut synchroniser votre version locale avec les changements enregistrés sur le serveur. Pour ce faire, il faut utiliser la commande

```
git pull
```

Attention cependant, si vous avez modifié les mêmes lignes de code que votre collaborateur qui a push sur le serveur, il y a alors conflit entre vos changement locaux et les changements qui proviennent du serveur. Pour résoudre ce conflit, il va donc falloir "merge", ce qui revient à dire lequel des deux changements vous voulez prendre, ou peut être un mélange des deux.

Une autre fonctionnalité de git qu'il est bon de connaître est la création et la gestion de "branches". Une branche est tout simplement un dossier dans le lequel vous allez y placer des commits. Cela sert surtout dans lorsque vous travaillez à plusieurs. voici les différentes commandes, création :

```
git branch "new-branch"
```

, se déplacer sur une branche existante

```
git checkout branch "name-branch"
```

et merge une branche dans une autre ex : merge la branche a dans la b, déplacez vous dans la branche de destination, ici la b puis

```
git merge "branch-a"
```

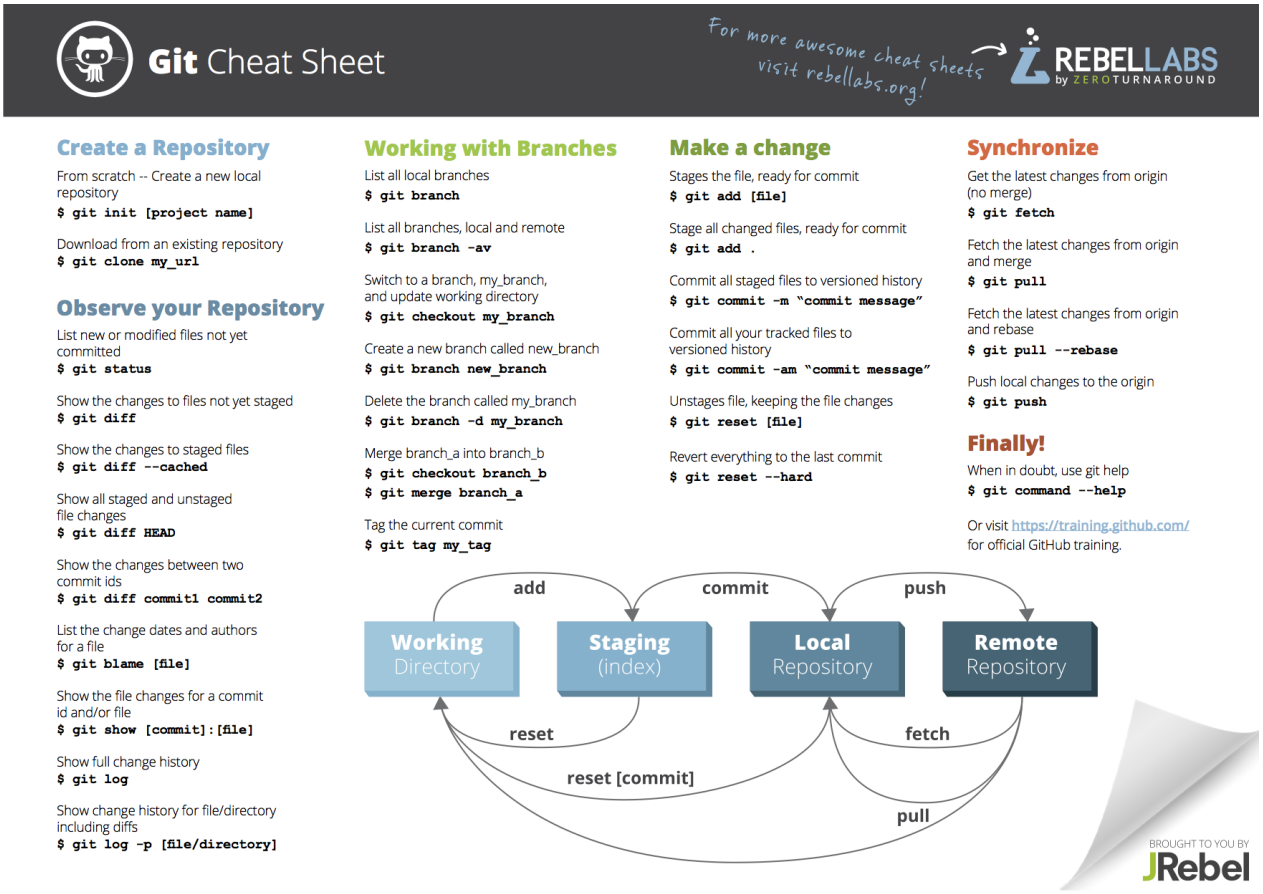

FIGURE 3 – Git Cheat

1.3 La pratique

1.3.1 En lignes de commandes

Voir le Git Cheat Sheet et le dernier tutorat avancé

1.3.2 Dans les IDEs JetBrains

Afin de récupérer un projet mis sur github pour pouvoir y participer, aller dans le menu VCS => Checkout from Version Control => Git

Il faut alors simplement copier l'url du repository GitHub sur lequel vous voulez travailler. C'est l'équivalent du **clone**.

Si vous ouvrez un projet que vous avez clone en ligne de commande, ne vous en faite pas, votre IDE saura reconnaître que le dossier est associé à un git et fera les paramétrages nécessaires normalement.

Pour pull, il suffit de cliquer sur la flèche bleue en haut à droite. Si il y a conflit, le gestionnaire de merge s'ouvrira et vous proposeras les différentes solutions.

Pour commit, il faut cliquer sur le tick vert en haut à droite. Un formulaire s'ouvrira alors à vous où il faudra préciser le message du commit et où vous pourrez spécifier d'autres paramètres.

Pour changer de branches (**checkout**), il suffit de cliquer en bas à droite de votre écran sur le **Git :origin**, vous n'aurez alors plus qu'a sélectionner la branche que vous voulez parmis toutes celles disponibles localement et sur le `remote repository`

2 Git-Flow

Git-Flow est un modèle, un ensemble de règles d'utilisation de Git. Un projet sur Git peut vite devenir une usine à gaz. L'utilisation de règles simples et connues permet alors une bonne compréhension par tous les acteurs actuels ou futurs du projet.

2.1 L'intérêt

- Connu (Google t'en dira plus)
- Simple (Quand on a compris les règles)
- Technologie éprouvé (Les grands l'utilisent)
- Meilleur maintenabilité des projets (Rangez vos commits)

2.2 La théorie

Les règles sont simples : Vos commits doivent être rangés dans des dossiers/branches. Il y a 5 branches différentes :

- master : Code stable [3], potentiellement testé et validé pour une MEP [4]
- develop : Code stable, Contient la prochaine version de l'application.
- feature : Contient le code d'une nouvelle fonctionnalité ou d'une modification. Ce code est ensuite fusionné(merge) à la branche develop.
- release : Branche qui contiendra les corrections des bugs et la préparation à une MEP. Cette branche ensuite fusionné(merge) avec la develop puis la master.
- hotfix : Correction d'un bug présent dans la master, donc après qu'elle est été mise en production.

On peut voir ces 5 branches comme les étapes de réalisation d'un projet : on définit qu'elles sont les ajouts à apporter au projet, on les réalisent à part afin de ne pas perturbé les autres ajouts et de toujours avoir un référentiel sain, puis après avoir vérifié que notre ajout était stable on merge notre travail sur la nouvelle version du projet, on effectue des tests et des corrections si nécessaire puis une fois validé on le met met en production. Si des problèmes surviennent après alors on fait un patch.

(Voir figure 4)

3. Code qui compile sans problème

4. Mise En Production : le projet/produit est diffusé, ex : un site web est mis en ligne. Une application mobile est mise sur un store.

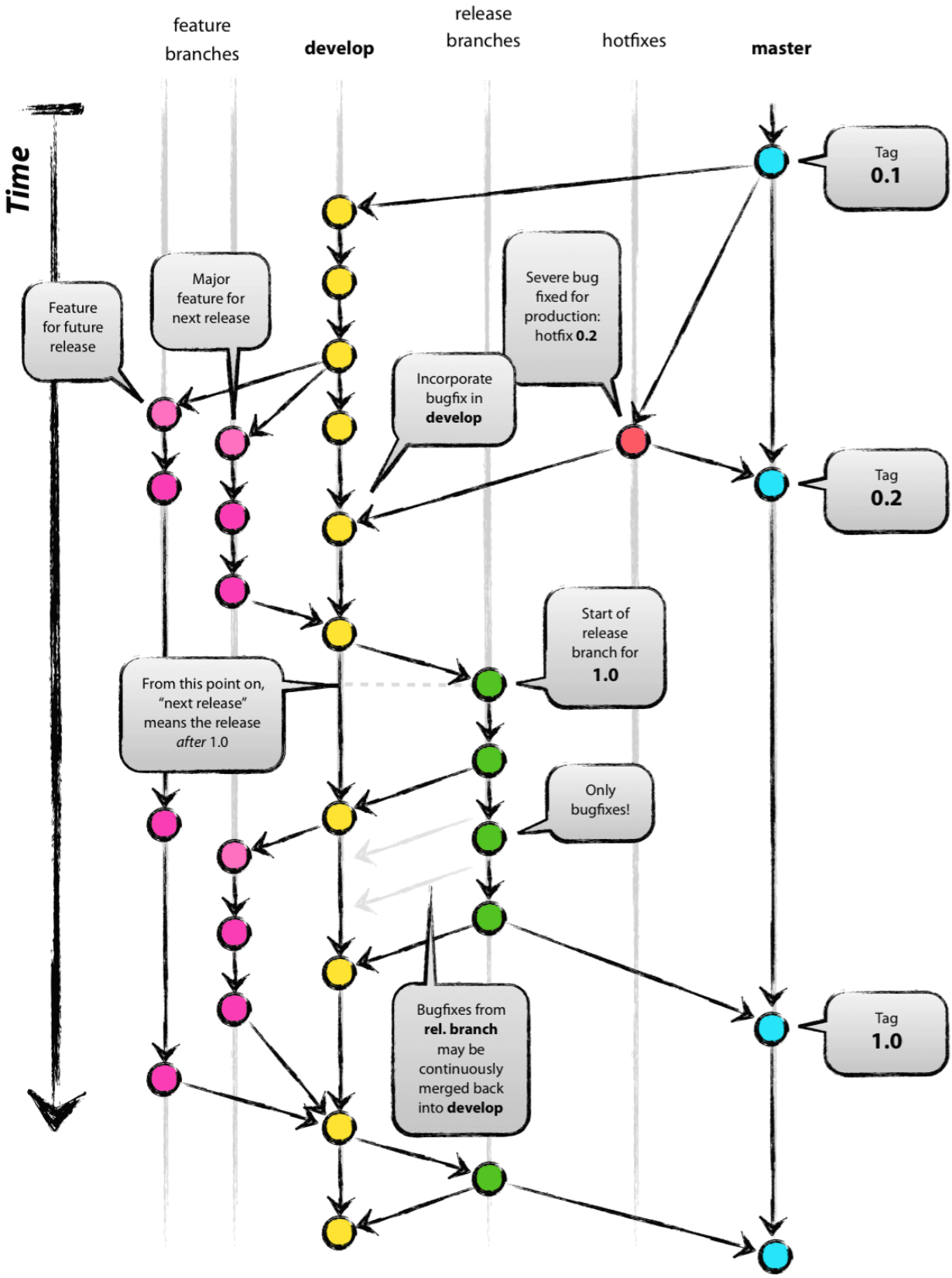

FIGURE 4 – Workflow type

2.3 La pratique

2.3.1 En lignes de commandes

Démerdez vous c'est chiant, mais sinon ce site là explique bien : le lien

2.3.2 Dans les IDEs

À notre connaissance les IDEs ne supportent pas nativement Git-Flow, il existe cependant des plugins permettant d'y remédier tel que Git Flow Integration pour les IDEs JetBrains

2.3.3 Dans les logiciels spécialisés

- Windows & Mac OS : SourceTree
- Linux & Mac OS & Windows : GitKraken