



# Workflows et Process

🕒 60 minutes 🖨️ Normal



DataScientest • com

GitHub

## Machine status



Ubuntu  
Server  
18.04 LTS  
SSD  
Volume  
Type  
64-bit x86

Online

34.245.135.23

➡ Connect

Reset



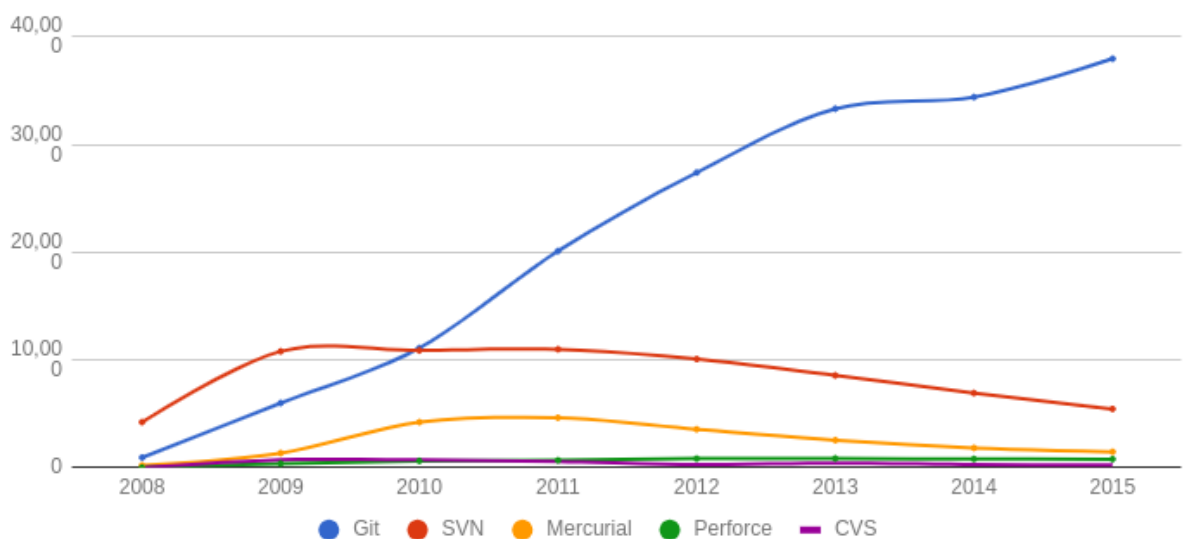
Stop

## III. Workflows et Process

Vous êtes désormais familier avec les principales commandes Git et la plateforme GitHub. Gardez en tête que Git et GitHub ne sont que des outils, pour lesquels il existe des alternatives.

SVN, CVS et Mercurial sont des exemples d'alternatives à Git, même si ce dernier est aujourd'hui le système de gestion de version (*version control system* ou VCS) le plus populaire, comme le montre ce graphique qui recense le nombre de questions par an liées à chaque VCS sur Stack Overflow :

Questions on Stack Overflow, by Year



De même, il existe de nombreuses plateformes proposant des services similaires à GitHub, dont certaines ont été évoquées précédemment : GitLab, Azure DevOps, Google Cloud Source, AWS CodeCommit. Le choix d'une plateforme pour une

Robin  
BIRON

Mais pour bien utiliser un système de gestion de versions, en connaître les commandes ne suffit pas. Il faudra également **définir des *process* et les respecter**.

Git en particulier est un outil très complet permettant de très nombreuses opérations, certaines d'entre elles étant plus ou moins équivalentes. Si chaque membre de l'organisation utilise les outils différemment, sans suivre un *process* commun à l'ensemble de l'équipe de développement, il en résultera de très nombreux conflits. Ceux-ci peuvent être résolus, mais c'est en général une opération chronophage, et il vaut donc mieux éviter les conflits que les résoudre.

C'est pourquoi des *process* standardisés (*workflows*) ont été définis quant à l'utilisation de Git. Il en existe plusieurs, et l'adoption d'un workflow plutôt qu'un autre sera en général la responsabilité du *technical leader* de l'équipe de développement. Les membres de l'équipe doivent cependant connaître et comprendre les différents workflows, et respecter le workflow choisi.

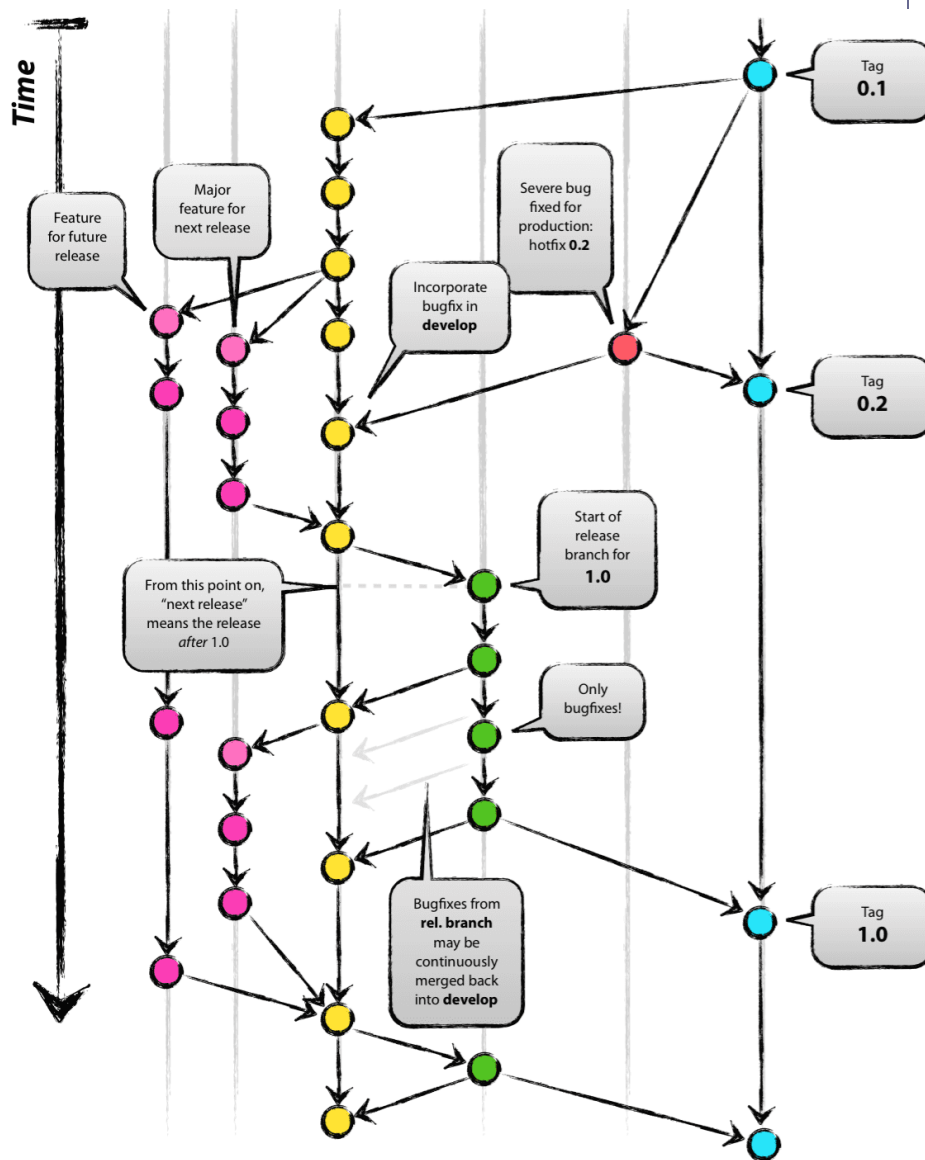
Nous détaillerons les deux workflows les plus connus : Git Flow et GitHub Flow. Il existe de nombreuses variantes de ces workflows et chaque équipe de développement peut adapter l'un de ces workflows à ses contraintes propres. L'important est de partager la même manière de travailler au sein de l'équipe.

## Git Flow

Git Flow repose sur un *branching model* (modèle de branche) relativement complexe. Il comporte les branches suivantes :

- **main** (ou **master**) : C'est la branche stable, dont les différents *commits* reflètent les versions du code qui ont été déployées en production. Pour chaque commit de cette branche, on crée en général un tag qui porte le nom de la version (par exemple **1.0.1**)
- **develop** : C'est la branche de développement, à partir de laquelle les développeurs vont créer les **feature branches**
- **feature branches** : ce sont les branches créées à partir de **develop** pour travailler sur une nouvelle fonctionnalité (*feature*). Par exemple, si vous êtes chargé d'implémenter l'authentification sur votre application, vous devrez créer une branche **feature/auth** à partir de **develop**. Une fois l'authentification implémentée, vous fusionnerez vos changements dans la branche **develop** (via une pull request).
- **release branches** : Une fois que les features que l'on souhaite déployer dans la prochaine *release* (sortie de la prochaine version de l'application) ont été développées et fusionnées dans **develop**, on peut créer une branche de release à partir de **develop**, dont le nom contient en général le numéro de la version (par exemple **release/1.0.1**). Ce code sera ensuite testé extensivement, et on pourra effectuer quelques corrections de bugs (*bug fix*) si nécessaire. Par la suite, on fusionnera la branche de release à la fois dans la branche **main / master** et dans la branche **develop**, et on pourra déployer la nouvelle version en production.
- **hotfixes** : ces branches sont créées à partir de **main / master** afin de corriger les bugs importants en production. Ces branches sont par la suite fusionnées à la fois dans la branche **main / master** et dans la branche **develop**.

Toutes ces notions sont plus simples à appréhender avec un schéma :

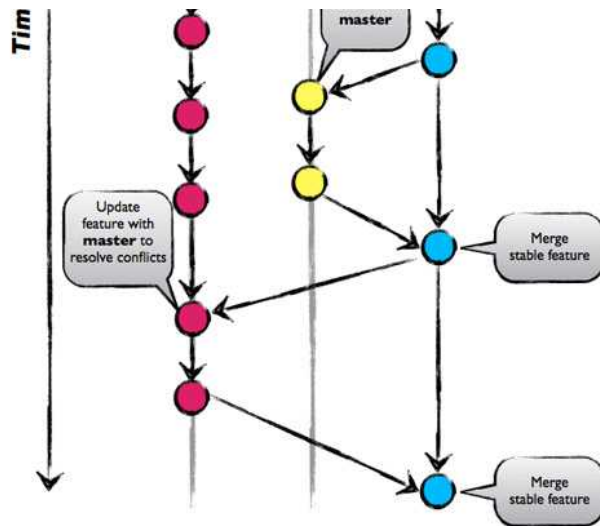


## GitHub Flow

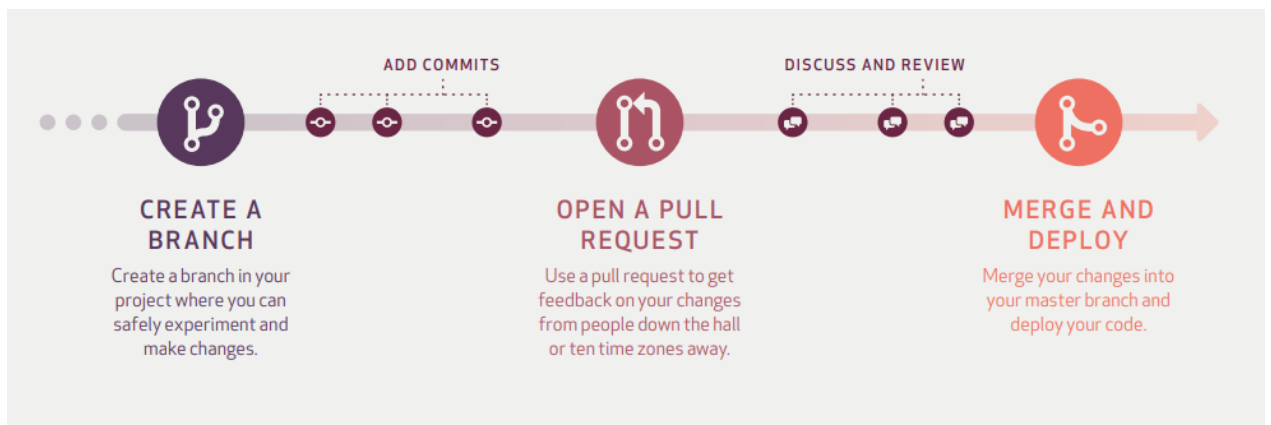
Comme nous l'avons vu, Git Flow est un workflow avec un *branching model* relativement complexe. Si ce modèle a des avantages, il peut néanmoins s'avérer lourd dans certains cas, notamment lorsque les *releases* sont très fréquentes (par exemple hebdomadaire voire quotidienne).

Certains workflows ont été défini dans le but de proposer un modèle plus simple et plus adapté dans le cas où l'on souhaite déployer les modifications dès qu'elles ont été développées, sans les regrouper au sein d'une release. C'est notamment le cas de GitHub Flow.

Le *branching model* est donc beaucoup plus simple : il comporte une branche **main** (ou **master**) qui contient les versions "déployables" de l'application. A partir de cette branche, les développeurs créeront de nouvelles branches pour travailler sur des *features* ou des *bug fix*. Après avoir implémenté la nouvelle *feature* ou corrigé le bug, la branche sera directement "mergée" dans **main** / **master** via une Pull Request :



Comme les changements sont intégrés directement dans `main / master`, il faudrait bien faire attention à prendre le temps de valider ces changements dans la Pull Request : celle-ci permet en effet aux différents contributeurs du projet de discuter quant aux changements apportés. En particulier, un administrateur du projet devra vérifier que ces changements sont conformes aux attentes, qu'ils sont bien documentés et testés, et qu'ils n'introduisent pas de bugs.



## Process complet


Les deux workflows vus précédemment définissent des *branching models* qu'une équipe de développement pourra adopter, en fonction de ses contraintes et de son organisation. Mais d'autres aspects devront être couverts pour définir le process complet :


- Le process de review des Pull requests : On a insisté sur l'importance de valider les changements proposés via les pull requests. Sur GitHub (ainsi que sur de nombreuses plateformes similaires), on peut notamment imposer qu'un certain nombre de *reviewers* (relecteurs) valident les changements avant de pouvoir la fusionner. Ceci peut être utile dans certaines équipes, afin de s'assurer de la qualité des modifications apportées. Pour d'autres, notamment des petites équipes, imposer plusieurs reviewers peut s'avérer lourd. Encore une fois, chaque équipe doit mettre en place les process qui lui conviennent.
- Mise en place de test : On peut (et même on doit) écrire des tests pour l'application développée, les automatiser, et par exemple imposer que les tests se passent avec succès pour pouvoir fusionner une Pull Request.
- Les conditions de déploiement : il faudra que l'équipe décide quand et comment les nouvelles versions devront être déployées. On peut notamment automatiser les déploiements à partir d'évènements sur un dépôt (création d'un tag, *merge* d'une PR) : c'est ce que l'on appelle le **Continuous Delivery** (Livraison continue).











Robin  
BIRON

