Débuter avec Git et Github

Nicolas Bauwens
Conseiller pédagogique / formateur
n.bauwens@bruxellesformation.be

Introduction

Quand on travaille sur un projet, on fait continuellement des changements. Il arrive même parfois qu'on fasse des changements regrettables qui introduisent des bugs et des effets secondaires indésirables et qu'on aurait envie « d'annuler ». Il faudrait pouvoir sauvegarder l'évolution de notre projet, ou en d'autres termes les « versions » de notre projet, comme ça si le travail qu'on est en train d'effectuer n'est pas correct, on pourrait facilement revenir à une version antérieure !

Mais comment mettre ce système de versions en place ?

Un autre problème survient quand on travaille à plusieurs sur un projet. Bob travaille sur l'écran de connexion au site pendant que Patrick, lui, travaille sur la page d'un profil utilisateur. Comment vont-ils partager leur travail ? En plus, il y a de grandes chances pour qu'ils modifient en même temps la CSS ou d'autres fichiers…

Comment faire pour travailler à plusieurs sans risquer des « conflits » entre le travail de chacun des membres de l'équipe ?

La réponse à ces 2 problèmes : utiliser un système de « contrôle de source ». Ce type de système va permettre de conserver toutes les modifications apportées au projet et de « centraliser » le travail effectué à plusieurs… et il résoudra pour nous tout conflit qui pourrait subvenir dans le cas où plusieurs personnes travaillent sur le même fichier !

Pour chaque document, un système de contrôle de source va nous permettre de savoir :

- **Quand** le document a été changé
- Sur **quoi** portent les changements
- **Pourquoi** il a changé
- **Qui** a fait le changement

Il existe plusieurs systèmes de ce type: SVN, Mercurial, TFS… mais ici nous allons parler de Git, qui est bien connu des projets open-source, ainsi que de son interface web : Github.

Installer Git

Vous trouverez Git ici : https://git-scm.com/downloads

Git est un logiciel qui s'utilise en ligne de commande. Il existe aussi une application de bureau très simple à utiliser mais il est utile de prendre en main en premier la gestion sous ligne de commande pour se familiariser avec les différents termes de Git.

Ouvrez donc votre terminal pour continuer ! ☺

Commandes de base

Configurer Git avec son nom et e-mail

Pour préciser son identité lors de la publication de ses changements

```
git config –global user.name "Bob l'Eponge"
git config –global user.email "bob@bikini.io"
```

Regarder sa configuration

```
git config –l
```

Configurer un nouveau projet et initialiser Git

1. Créer un nouveau répertoire (via l'explorateur ou la console avec mkdir) et s'y placer
2. Dire à Git de « surveiller » ce répertoire. A partir de ce moment, Git va conserver l'historique de notre projet. Git va créer ce qu'on appelle un dépôt local, ou ***local repository***.

```
git init
```

Si vous vous rendez dans ce répertoire, vous verrez qu'un dossier .git y a été ajouté – si vous ne le voyez pas, c'est parce qu'il est considéré comme un répertoire caché. C'est le répertoire de travail de Git. Il faut d'ailleurs faire très attention à ne pas le mettre en ligne.

Connaître le statut de vos fichiers

```
git status
```

Rajouter un fichier « à suivre »

Il faut indiquer à Git quels fichiers à suivre, c'est-à-dire les fichiers dont on veut conserver les versions

```
git add <nom_du_fichier_ou_du_répertoire>
```

Pour ajouter tous les fichiers et dossiers du répertoire

```
git add .
```

Faire une « photographie » de notre projet

Maintenant qu'on a dit à Git quels fichiers suivre, on va vouloir lui dire « voici l'état de ces fichiers à ce moment précis », et donc justement créer une version. Cela nous permettra de simplement faire une « photographie » du contenu de nos fichiers, pour pouvoir continuer à travailler dessus sereinement et éventuellement revenir à cette version donnée ultérieurement.

```
git commit –m "Message qui indique sur quoi porte cette version"
```

Regarder les différences

Maintenant que vous avez fait votre premier **commit**, vous pouvez continuer à travailler sur vos fichiers. Evidemment, des changements vont apparaître entre la dernière version *committée* et votre travail et pour voir ces différences pour prouver utiliser :

```
git diff
```

Avoir l'historique des commit

```
git log
```

Exclure des fichiers avec .gitignore

Il arrive un moment où l'on ne veut pas *committer* certains fichiers. En effet, certains environnements de développement créent automatiquement des fichiers de travail qui leurs sont propres et qui sont propres à chaque utilisateur et qu'il serait inutile de *versionner*. Un CMS comme Wordpress possède aussi certains répertoires qui n'ont pas d'intérêt à être *committés*. OS X possède des fichiers .DS_Store dont on se passerait bien. Ou, dès que le code sera publié et accessible à tous, il serait dangereux de partager des fichiers qui contiennent des informations sensibles comme de la config, des mots de passe ou des adresses e-mails.

Heureusement, un fichier simplement nommé .gitignore et situé à la racine de votre projet va permettre de dire « Git, ne prend pas ces fichiers en compte ».

Ce fichier doit juste contenir les chemins (relatifs à la racine de votre projet) que vous souhaitez ignorer.

Pour vous faciliter la tâche, https://www.gitignore.io vous permet de générer un fichier .gitignore en fonction d'un OS, langage de programmation, outil de développement ou CMS.

Lorsque vous créerez un repository distant via Github, celui-ci vous permet aussi d'initialiser le repository avec un fichier .gitignore.

Les branches

Les branches vont nous aider à pouvoir travailler sur une « autre » version de notre projet, une sorte de « réalité parallèle », tout en maintenant le projet initial sur le « tronc », la branche originale.

Par exemple quand on ajoute une nouvelle fonctionnalité à un projet, on va créer une branche et développer la fonctionnalité dans cette branche. De telle sorte, on va pouvoir garder une version saine du projet, tel qu'il est en production dans le tronc original (appelé **master**).

On pourra, par exemple, créer une autre branche qui serait destinée à corriger et mettre à jour la production en cas de bug. Tout en gardant en parallèle son travail sur une nouvelle fonctionnalité. On peut donc avoir plusieurs branches.

Chaque nouvelle fonctionnalité = une branche

Une branche devrait « vivre » le moins longtemps possible, juste le temps nécessaire pour terminer la fonctionnalité, c'est-à-dire quelques jours. Dès la fonctionnalité terminée, le code de la branche doit être fusionné (*merged*) dans le tronc principal et la branche effacée.

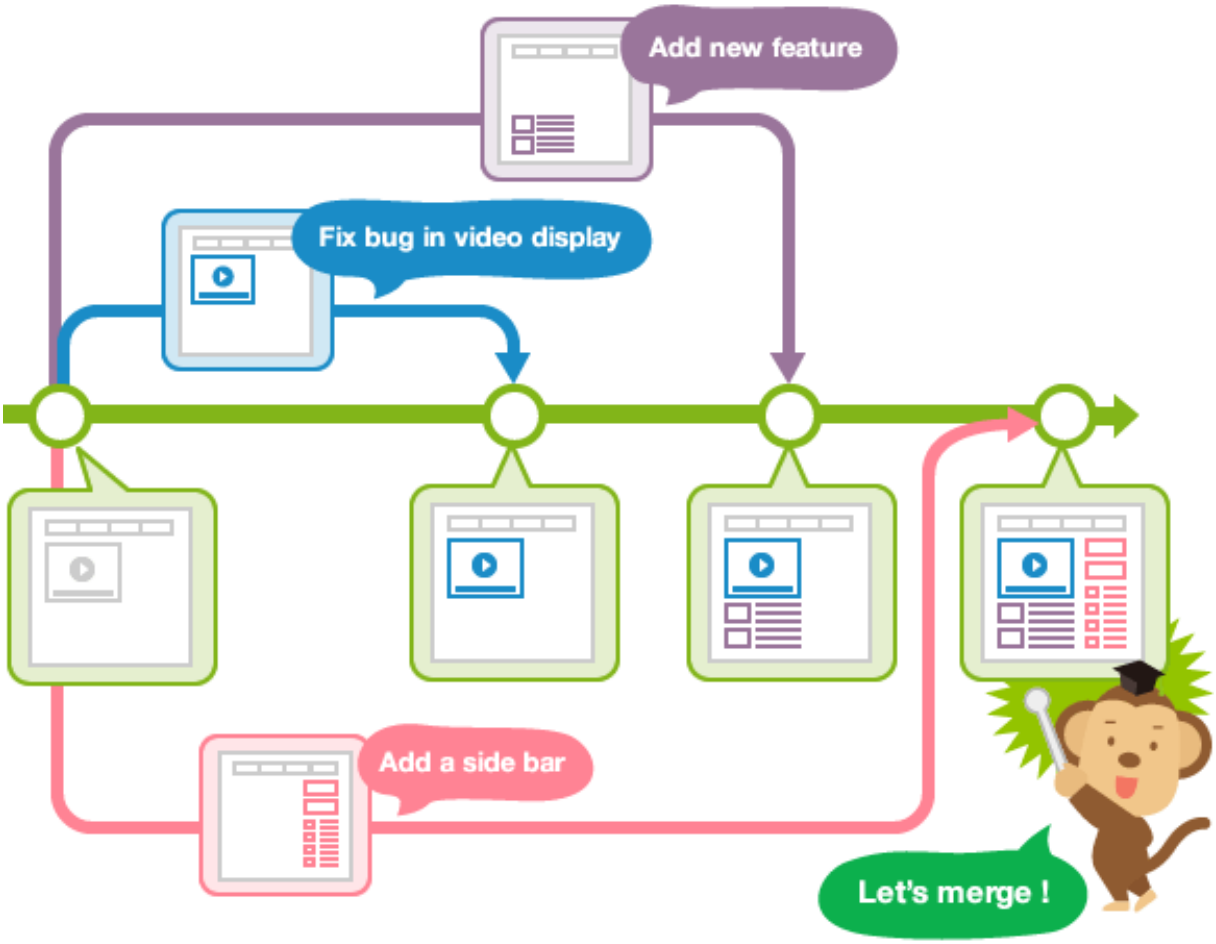


Image : https://backlogtool.com/git-guide/en/stepup/stepup1_1.html

Créer une branche

```
git branch <nom_de_la_branche>
```

Voir les branches

```
git branch
```

Se mettre sur une branche

Pour commencer à travailler sur la fonctionnalité liée à la branche, il faut se mettre dessus car par défaut on est sur la branche master.

```
git checkout <nom_de_la_branche>
```

Voir les différences entre deux branches

```
git diff master..<nom_de_la_branche>
```

Rapatrier une branche vers le tronc

```
git merge <nom_de_la_branche>
```

Si des changements sont apportés sur la même ligne de texte dans 2 branches différentes, le *merge* va donner un conflit. Il faut donc apporter une action manuelle, choisir quel est le bon contenu et refaire un `git add` suivi d'un `git commit` car le *merge* n'a pas fonctionné.

Supprimer une branche

```
git branch -d <nom_de_la_branche>
```

Partager le code avec un dépôt distant

Pour le moment, tout se passait en local. Dans le cas où l'on veut collaborer avec d'autres développeurs sur le même projet ou tout simplement sauver son travail sur un serveur, il faut travailler avec un dépôt distant (***remote repository***) qui va vous permettre de « pousser », publier votre code vers ce dépôt distant et de récupérer les changements des autres.

Ce *remote repository* va se situer sur le serveur de Github. Github c'est un service, un portail, un serveur qui va vous permettre de sauver votre code en ligne, de manière centralisée, pour le partager avec d'autres développeurs. Dès que votre code sera poussé/publié sur le serveur, tout le monde pourra le voir… c'est le principe de l'open-source ! Tout le monde pourra également y contribuer et/ou le réutiliser pour en faire quelque chose d'autre.
Github possède aussi des outils qui en font un véritable réseau social : vous pouvez suivre des développeurs, des projets, « starrer » des projets, avoir des indicateurs sur le projet et chaque développeurs, etc…

Créer un remote repository

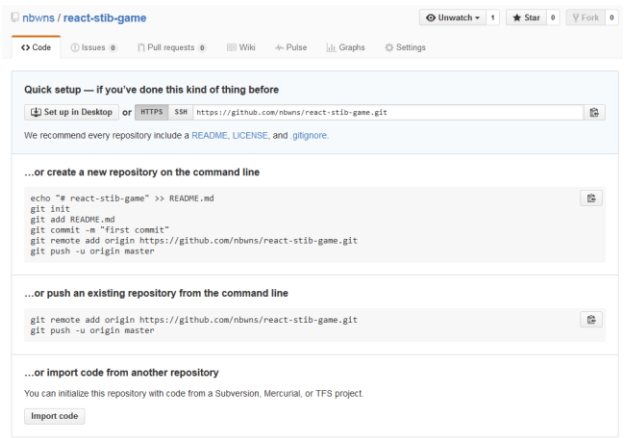
Pour cela vous devez vous créer un compte sur Github (www.github.com) en utilisant le même nom et e-mail que ceux précisé dans la configuration.

Après, allez sur https://github.com/new et indiquez:

- Un nom de repository
- Une description éventuelle
- Si le projet doit être public (open-source) ou privé (là, il faut payer Github)
- Si vous voulez initialiser le repository avec un README, une licence et un .gitignore

Une fois le *remote repository* créée, une page vous expliquant comment y ajouter votre code sera affichée.

Vous verrez notamment l'URL du projet, de la forme suivante :
https://github.com/<user>/<nom_du_repository>.git



Configurer un remote repository pour son projet

```
git remote add <nom_du_remote> <url_projet>
```

Le nom du *remote* est par défaut **origin**, et l'url du projet c'est celui qui se fini par .git, cela donnera donc par exemple :

```
git remote add origin http://github.com/nbwns/my-project.git
```

Vérifier qu'un remote repository est disponible pour le projet

```
git remote -v
```

Envoyer son code vers le remote repository

```
git push <nom_du_remote> <nom_de_la_branche>
```

Et donc, dans le cas d'un travail sur le tronc principal (*master*)

```
git push origin master
```

Récupérer du code du remote repository

Dans le cas où vous voulez récupérer (et non plus envoyer) du code de votre *remote repository*

```
git pull <nom_du_remote> <nom_de_la_branche>
```

Et donc, dans le cas d'un travail sur le tronc principal (master)

```
git pull origin master
```

Récupérer en local un projet existant

Vous pouvez récupérer sur votre ordinateur un projet existant développé par quelqu'un d'autre – pour y contribuer ou simplement pour vous en inspirer ou l'adapter.

```
git clone <url_projet>
```

Comme `git init`, cela initialisera un *local repository* mais avec tout le code du projet désiré.

Récupérer les derniers changements

Vous êtes en train de travailler sur une branche d'un projet sur lequel vous collaborez et vous souhaitez récupérer les changements des autres sans perdre vos changements ? C'est simple !

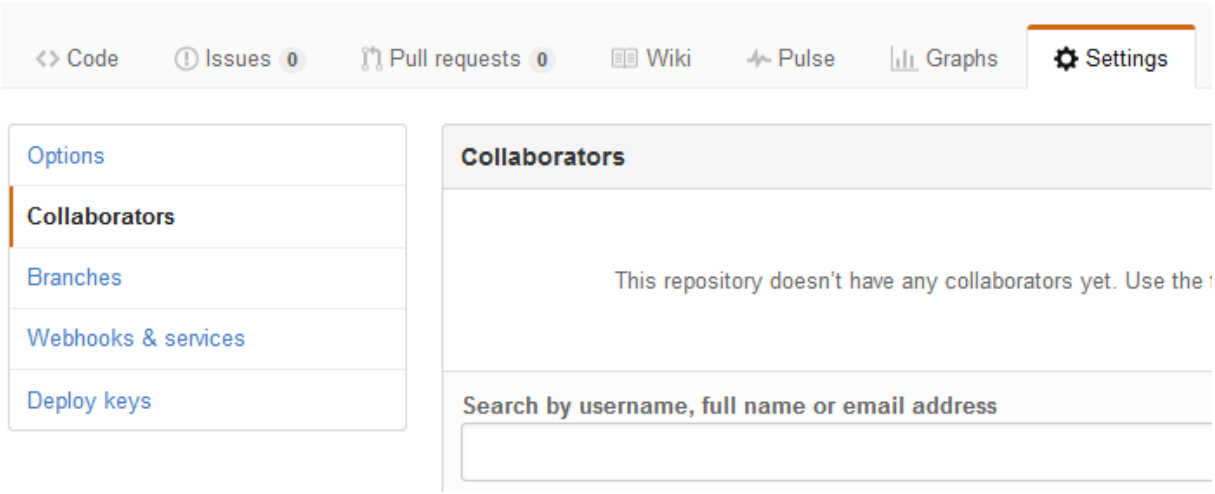
```
git fetch
```

La différence entre `git pull` et `git fetch` c'est que *pull* rapatrie la dernière version des changements mais la fusionne automatiquement avec vos changements (bref, équivaut à faire un *fetch* & *merge*)

Collaborer

Jusqu'ici nous n'avons pas encore abordé le travail collaboratif, nous étions seul sur le projet. Alors comment collaborer avec d'autres développeurs via Git et Github ?

Il faut tout d'abord ajouter des collaborateurs au projet, ceci peut être fait via les *Settings* de la page du projet sur Github.



Chaque collaborateur devrait travailler sur sa branche et non pas sur master.

Pour remettre les changements apportés par chaque collaborateur sur la branche master, la même technique vue précédemment peut être utilisée.

Pour les plus gros projets, un *code review* sera souvent d'application. C'est-à-dire qu'un ou plusieurs développeurs seront responsables de la qualité du projet et du code et chargés de relire le code pour éviter des erreurs.

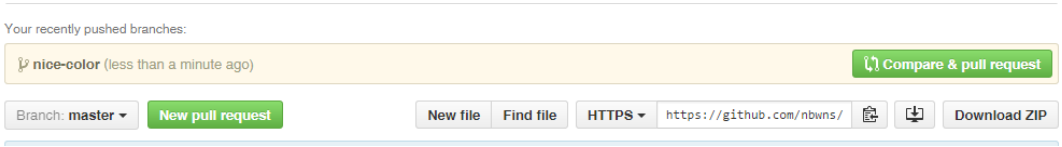
Un développeur ne valide donc pas tout seul ses changements, et la procédure pour mettre ses changements sur la branche master sera donc différente. Il faudra dire : « peux-tu vérifier mes changements, et si c'est valide, faire un merge vers master ? ». Avec git on va faire ça via ce qu'on appelle une ***pull request***.

Un autre développeur pourra donc regarder le code soumis via cette *pull request*, faire des commentaires et demandes de modifications, et si tout est OK, il pourra l'accepter. Ceci fera automatiquement un merge vers master.

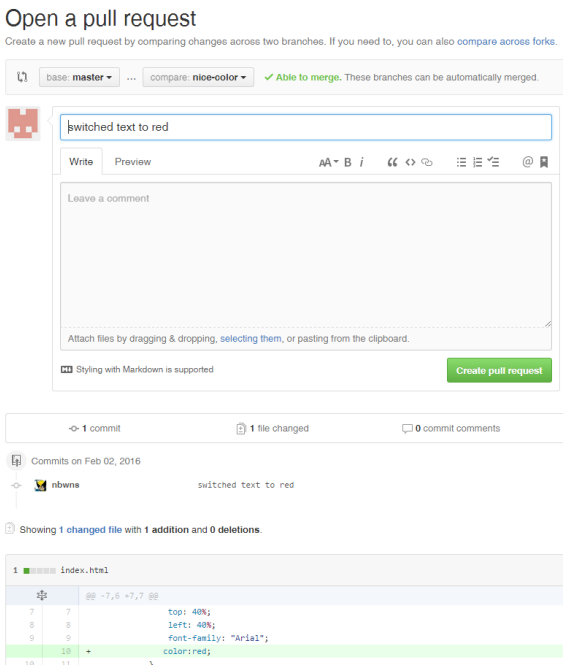
Par exemple, Bob a créé un projet et l'a publié. Il ajoute Patrick comme contributeur de son projet.

Patrick veut changer la couleur du texte dans la page HTML faite par Bob. Il pourrait faire les changements directement car il est contributeur, mais pour rappel, une bonne pratique est qu'un développeur ne valide pas tout seul ses changements. Voici ce qu'il devrait faire :

1. Faire un clone du projet de Bob via `git clone`. Cela va créer un *local repository* sur sa machine avec tout l'historique et le code du projet.
2. Créer une branche pour y apporter les changements désirés
3. Se mettre sur la branche
4. Modifier le document
5. Faire un `git add` suivi d'un `git commit`
6. Configurer le *remote repository* (vers le projet initial de Bob)
7. Faire un `git push` de la branche
8. Aller sur Github pour ouvrir une *pull request*



A ce moment-là, Bob sera informé que Patrick veut apporter des changements. Via l'interface de Github il pourra discuter avec Patrick (« je préfère vert que rouge, peux-tu modifier ?»), et une fois que Bob considère que les changements sont OK, il pourra faire un *merge* de la branche de Patrick sur *master*.



C'est cette manière de faire qui est utilisée pour les gros projets open-source sur Github. En effet, si vous souhaitez contribuer à un de ces projets, vous ne serez pas mis directement comme contributeur, ça serait trop dangereux ! Vous devrez donc faire un ***fork***, l'équivalent d'un *clone* sur un projet dont vous n'êtes pas directement contributeur, et ouvrir une *pull request*.

Les pages Github

Vous développez un site statique en HTML / CSS / JavaScript, hébergé sur Github et vous désirez l'héberger facilement et… gratuitement ?
Ne cherchez plus, hébergez-le sur Github grâce aux *Github pages*.

1. Créez une branche dans votre projet qui s'appelle gh-pages

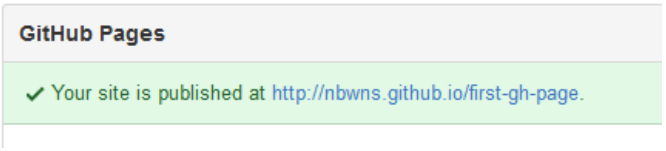
```
git branch gh-pages
```

2. Faites un push de cette branche

```
git push origin gh-pages
```

3. Votre projet est maintenant accessible depuis
   http://<nom_utilisateur>.github.io/<nom_remote_repository>

   Par exemple : http://nbwns.github.io/first-gh-page/
4. Vous pouvez vérifier cela en allant via Github dans les Settings du projet



Sources et ressources

Débuter avec Git et Github (Le Wagon)
https://www.youtube.com/watch?v=V6Zo68uQPqE

Gérer vos codes sources avec Git (OpenClassRooms)
https://openclassrooms.com/courses/gerez-vos-codes-source-avec-git

Git – petit guide
http://rogerdudler.github.io/git-guide/index.fr.html

Git Beginner's Guide for Dummies
https://backlogtool.com/git-guide/en/

Learn Git (CodeCademy)
https://www.codecademy.com/learn/learn-git

Try Git (CodeSchool)
https://www.codeschool.com/courses/try-git

La doc Github
https://git-scm.com/docs

Tutoriel de démarrage
https://guides.github.com/activities/hello-world/