

## Tutoriel Git et GitHub \_ durée 30min

Git nous évite d'avoir dans nos projets des documents du type:

*Travail.doc*  
*Travail(version2).doc*  
*Travail(version de Patrick).doc*  
*Travail(ajout conclusion).doc*  
*Travail(version corrigée).doc ...*

Entre toutes ces versions du fichiers on s'y perd.

L'idée c'est d'avoir un système qui nous dit

- quand le fichier a été modifié,
- ce qui a été modifié,
- pourquoi ça a été modifié,
- qui a effectué cette modification

**Git** = outil en lignes de commandes pour versionner nos fichiers

**GitHub** = application web qui va nous aider à partager notre code

**Installez git si vous ne l'avez pas déjà :**

(l'instruction `$ git --version` vous permet de vérifier si Git est installé ou non sur votre ordinateur)

Téléchargez et installez git depuis <http://git-scm.com/>

Dans votre interface de ligne de commande (ici : Cmder) configurez git en indiquant vos nom, prénom et e-mail :

```
git config --global user.name "prénom nom"
git config --global user.email "adresse e-mail"
git config --l
```

## Partie 1. GIT - les commandes de base.

### 1.1 Création du projet :

Avec Cmder, placez-vous dans l'espace de votre ordinateur où vous souhaitez créer votre projet et faites les commandes suivantes

```
$ mkdir monProjet
$ cd monProjet
$ git init ( = dit à git de versionner le code)

$ git status (nous donne l'état de notre dossier)
```

Vous pourrez retrouver les commandes de git en suivant ce lien:

<http://in202-uvsq.github.io/git-manual/commandes.html>

## 1.2 Créer et traiter des fichiers :

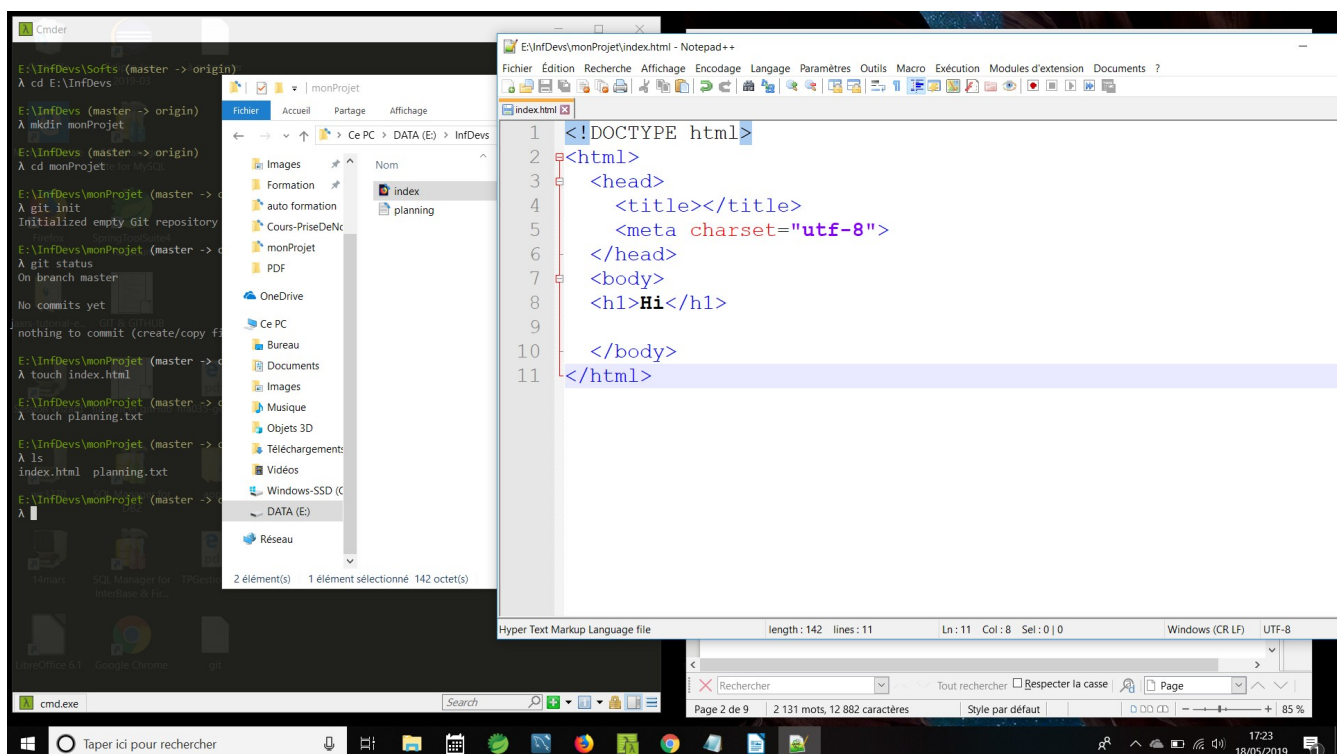
Faites les commandes suivantes:

```
$ touch index.html  
$ touch planning.txt
```

`$ls` (vous permet de voir ce que contient votre dossier *monProjet*)

Hors de Cmder, parcourez *monProjet* et ouvrez *index.html* et *planning.txt* avec un éditeur de texte.

Ajoutez du contenu à *index.html* :



Ouvrez le fichier avec le navigateur pour voir le résultat.  
(Pour le moment on ne touche pas à *planning.txt*)

`$ git status` (nous indique que ces 2 fichiers ne sont pas suivis)

Commencez à versionner ces fichiers (à surveiller leurs changements):

2 étapes :

```
1. $ git add index.html  
   $ git add planning.txt  
   ou git add . (pour sélectionner tout le dossier)
```

`$ git status` (les 2 fichiers sont prêts à être commité)

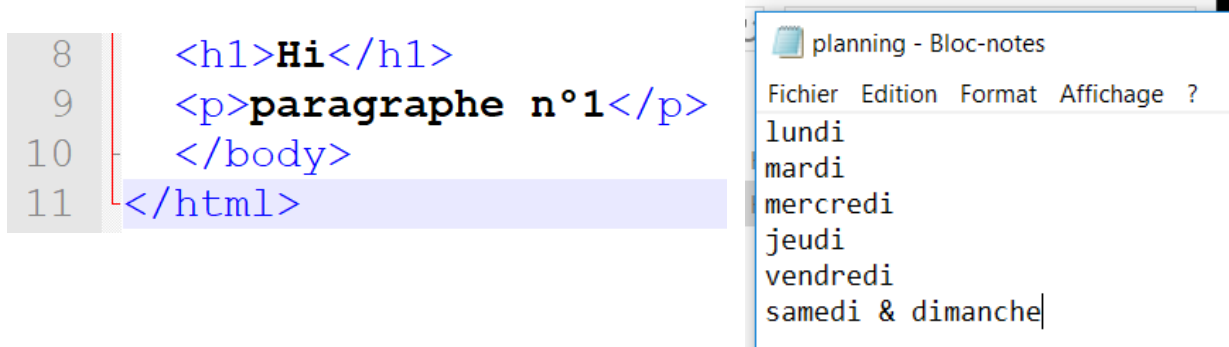
( commit = une *photo* de votre travail à un instant donné. )

```
2. $ git commit -m "Création index.html et planning.txt version 1"
```

```
$ git status (rien de plus à commiter)
```

### 1.3 Enregistrer des modifications :

Dans le fichier *index.html* ajoutez un paragraphe et enregistrez le fichier. Dans le fichier *planning.txt*, ajoutez la liste des jours de la semaine.



```
$ git status (nous signale qu'il y a eu des changements dans les fichiers html et txt et qu'ils ne sont pas prêt pour le commit)
```

Avant de commiter ces changements regardez les différences avec la version précédente de vos fichiers:

```
$ git diff
```

Ensuite faites le commit :

```
$ git add index.html
```

```
$ git add planning.txt
```

```
$ git commit -m "ajout d'un paragraphe dans index.html et des jours de la semaine dans planning.txt"
```

```
$ git status (rien de plus à commiter)
```

Obtenez l'historique du projet :

```
$ git log
```

Git log permet de visualiser tous les commits qui ont été faits et vous donne la méta-donnée autour des fichiers : identifiant de chaque commit + auteur du commit + date du commit + commentaire

```
E:\InfDevs\monProjet (master -> origin)
λ git log
commit 26e37a26340daad7aa2b8d30acf9366632038b0a (HEAD -> master)
Author: AntoineCoolen <coolen.antoine@gmail.com>
Date: Sat May 18 17:42:27 2019 +0200

    ajout d'un paragraphe dans index.html et des jours de la semaine dans planning.txt
```

## 1.4 Créer des branches

Jusque là nous avons un axe de temps qui est linéaire, les commits viennent les uns à la suite des autres sur la branche principale de notre projet: la branche MASTER.

Avec Git on travaille souvent par fonctionnalité, et pour chaque fonctionnalité on crée une branche.

**Branch** = réalité parallèle qui nous permet de manipuler les éléments du projet, de faire des expériences, sans polluer notre environnement de travail (le master).

Vous allez voir si ajouter un fond d'écran bleu à votre page html est une bonne idée :

Créez une branche:

```
$ git branch fondEcranBleu
```

```
$ git branch
```

 (affiche les branches disponibles, \* indique la branche sur laquelle vous êtes en train de travailler)

```
$ git checkout fondEcranBleu
```

 (changer de branche de travail)

Modifiez *index.html* (fond bleu + texte blanc):

```
5 <meta charset="utf-8">
6 </head>
7 <style type="text/css">
8   body {
9     background-color: #3498db ;
10    color: white;
11  }
12 </style>
```

Remarque : obtenir les réf. couleur sur le site <https://flatuicolors.com/>

```
$ git status
```

 (nous indique qu'il y a eu un changement)

Faites la sélection puis le commit de votre fichier (avec le message "ajout arrière-plan bleu index.html").

Modifiez à nouveau *index.html* en ajoutant le titre *Bonjour* (qui apparaîtra dans l'onglet de la page web):

```
<head>
  <title>Bonjour</title>
  <meta charset="utf-8">
</head>
<style type="text/css">
```

Faites la sélection puis le commit (avec le message "ajout titre index.html").

Constatez les changements depuis votre navigateur.

```
$ git status
```

 (nous indique qu'il n'y a rien à commiter)

Revenez sur la branch master : `$ git checkout master`

Sur votre navigateur, actualisez la page, votre changement a été annulé.

On a donc bien 2 versions de notre fichier.

`$ git diff master..fondEcranBleu` (affiche les différences entre les 2 branches):

```
--- a/index.html
+++ b/index.html
@@ -1,9 +1,15 @@
<!DOCTYPE html>
<html>
  <head>
-    <title></title>
+    <title>Bonjour</title>
+    <meta charset="utf-8">
  </head>
+  <style type="text/css">
+  body {
+  background-color: #3498db ;
+  color: white;
+  }
+  </style>
  <body>
  <h1>Hi</h1>
  <p>paragraphe n°1</p>
```

Avant d'importer les modifications de *fondEcranBleu* dans *master*, ouvrez le fichier *index.html* (la version du *master*, sans fond d'écran bleu).

→soulignez Hi à l'aide de ces balises : `<h1><u>Hi</u></h1>`

→mettez le 1<sup>er</sup> paragraphe en italique: `<p><i>paragraphe n°1</i></p>`

Sélectionnez et committez ces modifications (avec le message "modif' index.html, hi souligné et paragraphe n°1 en italique").

Maintenant faite le merge (la fusion) de *fondEcranBleu*:

`$ git merge fondEcranBleu`

ça n'est pas un remplacement mais bien une fusion, git a gardé à la fois les modifications de *fondEcranBleu* et les modifications du *master*, vous le constaterez en faisant `$ git log` ou en actualisant le navigateur.

Maintenant que la branche *fondEcranBleu* n'est plus utile, effacez-là avec `$ git branch -d fondEcranBleu`

## 1.5 Les conflits

Si 2 développeurs sont intervenus dans le même élément et à la même ligne que se passe-t-il?

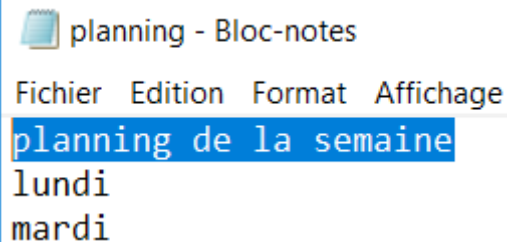
Créez une branch `editBranch`

Placez vous sur `editBranch` (`$ git checkout editBranch`)

Modifiez `index.html` : ajoutez ***everybody!*** À la suite de *Hi*.

Modifiez `planning.txt` : ajouter le titre ***planning de la semaine*** en haut de la page.

```
13 <body>
14 <h1><u>Hi everybody!</u></h1>
15 <p><i>paragraphe n°1</i></p>
16 </body>
17 </html>
```



planning - Bloc-notes

Fichier Edition Format Affichage

planning de la semaine

lundi

mardi

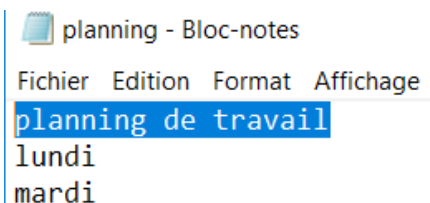
Sélectionnez les deux fichiers et commitez-les (avec le message "modification des en-têtes 1").

Remplacez-vous sur la branche Master et faites les modifications suivantes:

`Index.html` : Remplacez *Hi* par ***Hola a todos!***

`Planning.txt` : ajouter le titre ***planning de travail*** en haut de la page.

```
14 <h1><u>Hola a todos!</u></h1>
15 <p><i>paragraphe n°1</i></p>
16 </body>
17 </html>
```



planning - Bloc-notes

Fichier Edition Format Affichage

planning de travail

lundi

mardi

Après avoir commité (avec le message "modification des en-têtes 2"), faite la commande de merge de `editBranch` dans le master:

```
$ git merge editBranch
```

Vous obtenez un message d'erreur "conflit" car il y a plusieurs informations pour les mêmes lignes.

Ouvrez les fichiers `index.html` et `planning.txt` pour constater que git a gardé et "encadré" toutes les modifications.

C'est à nous de prendre la décision et d'effacer la partie qu'on ne garde pas.

Effacez les modifications effectuées dans **editBranch**, gardez **Hola a todos!** pour *index.html* et **planning de travail** pour *planning.txt*

Faites : **\$ git status** (fichiers non-mergés à cause du conflit)

Vous devez vous-même faire la manipulation mais cette fois-ci procédez de la façon suivante :

```
$ git add . (sélectionne tous les fichiers à commiter)
$ git commit
```

(nous n'avons pas indiqué de message pour ce commit, git vous ouvre une page de traitement dans Cmdr et propose un message préparé: **merge branch 'editBranch'** que l'on peut modifier si besoin. Utilisez la touche **esc**, faites **:x** puis **enter** pour sortir de cet espace.

Utilisez la commande **\$ git status** pour vérifier l'état de votre projet puis supprimez la branch **editBranch**.

Nous avons terminé de travailler sur nos branches, elles ont été fusionnées avec le master et effacées. La branch master est à jour.

Dès que l'on voudra de nouveau travailler sur le projet nous pourrons de nouveau tirer une branche, y travailler puis la merger.

La bonne pratique c'est bien de travailler dans une branche et non pas dans le master.

Merger le plus souvent possible, sinon le jour où vous voudrez merger il y aura beaucoup de conflits à gérer.

## 1.6 Retour en arrière :

Nous allons revenir dans une version antérieure de notre projet, vous allez utiliser des commandes que nous avons déjà vu :

Afficher tous les commits que vous avez effectué : **\$ git log**

Remarque : lors de l'affichage des logs : pour remonter dans les logs plus anciens utilisez la touche **enter**, puis **q** pour sortir.

Copiez l'identifiant du tout premier commit (associé au message "Création index.html et planning.txt version 1")

```
commit d61c4ac1687985f4bbf1630a42df35af6f9e90fa
Author: AntoineCoolen <coolen.antoine@gmail.com>
Date: Sat May 18 17:32:37 2019 +0200

    Création index.html et planning.txt version 1
(END)
```



Dans la ligne de commande :

```
$ git checkout + collez l'identifiant du commit
```

Consulter le fichier *index.html*: le titre, l'arrière-plan bleu et les autres modifications que vous avez apportées ont disparues. Idem pour le fichier *planning.txt*.

Créez une nouvelle branche

```
$ git branch review
```

```
$ git checkout review
```

Vous vous trouvez maintenant sur une copie de la première version du projet (qui correspondait au premier commit) sur laquelle vous pouvez travailler.

Ceci n'est pas une bonne pratique cependant car lorsque vous voudrez merger cette branche dans la version avancée du master vous risquez d'avoir plusieurs conflits.

Revenez donc sur la branche master et supprimez review.

## Partie 2 GitHub – le repository distant (remote)

GitHub va nous permettre de partager le projet.

Vous avez le dossier *monProjet* qui est le repository local et hors-ligne, vous allez le 'pousser' sur GitHub pour que vos collaborateurs puissent récupérer le projet.

### 2.1 Créer un repository distant

Rendez-vous sur <https://github.com/>

Si ça n'est pas déjà fait créez un compte github.

Sur votre espace Github créez un nouveau repository et nommez-le *notreProjet* (suivi de vos initiales). Par défaut le projet est public, Opensource (il faut donc faire attention à ce qu'on y met) mais depuis janvier 2019 Microsoft a racheté GitHub et les repository privés sont également devenus gratuits.

Le repository créé, Github vous explique comment faire le push (importer le projet). Copiez-collez les lignes de commandes suivantes :

```
$ git remote add origin + url (cela permet de lier votre repository local monProjet à votre repository github notreProjet)
```

```
$ git push -u origin master
```

Le terme *origin* correspond au repository distant (nom par convention), *master* correspond à la branche que vous voulez envoyer à *origin*.



Actualiser la page gitHub, vous y retrouvez votre projet, votre branche master ainsi que tous les commits que vous avez effectué.

Affichez les **commits**, ouvrez le **commit** correspondant au message "ajout titre index.html", gitHub vous montre la modification qui a été faite pour ce **commit**.

Rendez-vous dans l'onglet **Insights**, sélectionner **NetWork** pour afficher graphiquement l'état de votre projet.

Affichez le repository *notreProjet* et ouvrez le fichier *index.html*, modifiez-le en ligne: mettez l'arrière-plan en rouge, ajoutez un second paragraphe et mettez le premier en gras. (ces changements auraient aussi bien pu être fait pas un collaborateur).

```
8     body {
9         background-color: #e74c3c ;
10        color: white;
11    }
12    </style>
13    <body>
14        <h1><u>Hola a todos!</u></h1>
15        <p><i><b>paragraphe n°1</b></i></p>
16        <p><i>paragraphe n°2</i></p>
17    </body>
18    </html>
```

Nommer cette modification «**modification index.html sur gitHub**» et validez.

De retour sur Cmder utilisez **\$ git log** (vous constatez que ce dernier commit, présent sur gitHub, n'apparaît pas sur votre repository local.) Pour importer la dernière version de votre projet sur votre ordinateur utilisez l'instruction **\$ git pull origin master**

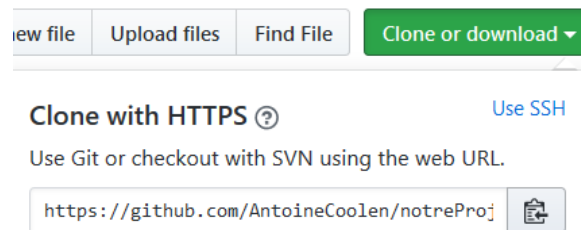
**\$ git log** le commit apparaît, votre fichier a été mis à jour.

## 2.2 Le pull request :

Sur gitHub allez dans l'onglet **settings** et ajoutez un collaborateur.

Ce collaborateur va recevoir l'invitation e-mail et l'accepter. Il aura alors accès à votre repository et pourra y opérer des changements.

Pour récupérer le projet sur son ordinateur votre collaborateur doit copier-coller l'url https de votre repository,



et effectuer sur Cmder l'instruction suivante :

```
$ git clone + url https
```

Le dossier *notreProjet*(+ initiales) est maintenant créé sur l'ordinateur de votre collègue avec les fichiers *index.html* et *planning.txt*.

Votre collaborateur va entrer dans ce dossier

```
$ cd notreProjet(+ initiales)
```

Créer une nouvelle branche avec son nom et son prénom

```
$ git branch nomPrenom
```

```
$ git checkout nomPrenom
```

et ajouter les activités dans *planning.txt*

```
lundi = Cobol
```

```
mardi = java
```

```
mercredi = Hibernate
```

```
jeudi = SQL
```

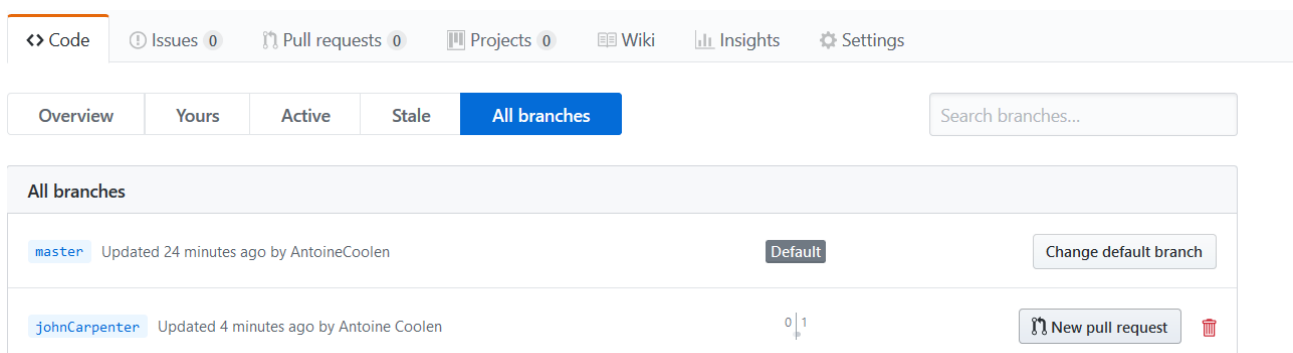
```
vendredi = XML
```

```
samedi & dimanche = PLS
```

Après avoir fait le commit (avec le message "ajout des activités dans le planning") faire **\$ git push origin nomPrenom**

La branch nomPrenom apparaît sur gitHub, vous pouvez tous les deux la consulter.

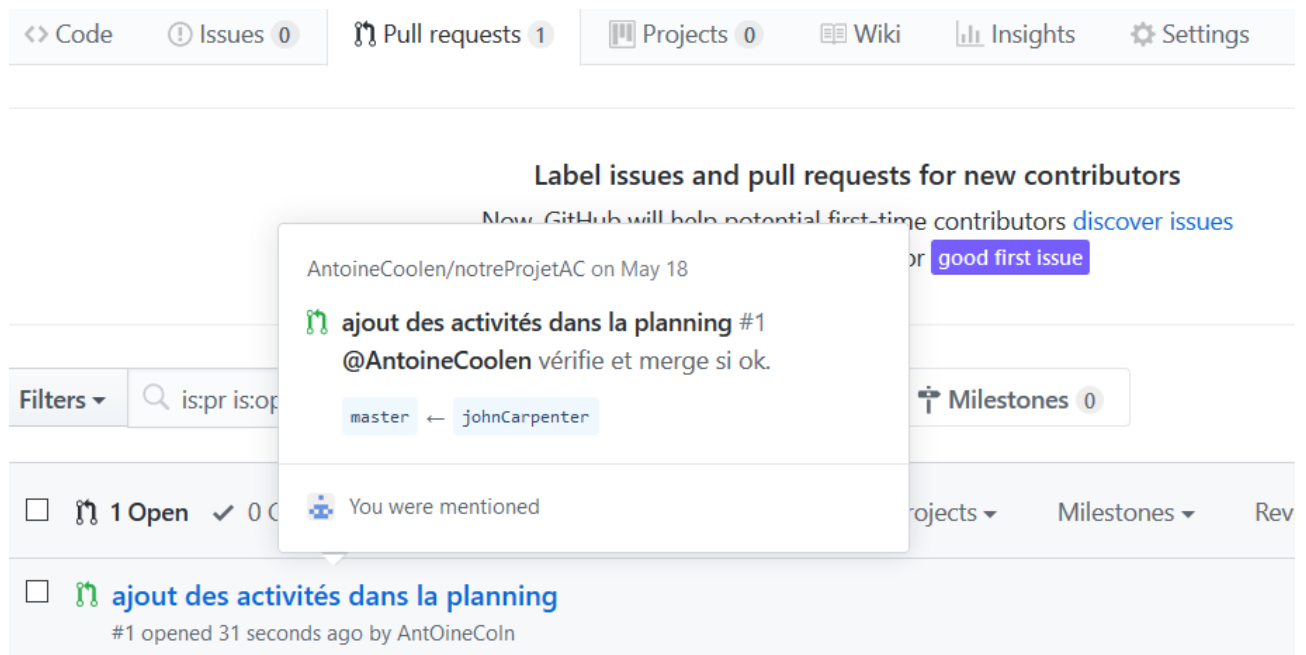
Votre collaborateur veut savoir si vous validez sa modification avant de la merger dans le master, il doit donc ouvrir l'onglet branch, sélectionner sa branch et faire une demande de **PULL REQUEST**.



Cette outil permet de signifier que cette branch est prête à être mergée après vérification de l'équipe (en effet une bonne pratique est de ne pas merger soit-même ses branches).

Votre collaborateur accompagne le pull request d'un message qui vous est adressé.

Actualisez votre page Github, consultez le Pull Request.



Puis consultez la branche proposée par votre collaborateur et faite le merge.

.. @@ -1,7 +1,7 @@	
1 planning de travail	1 planning de travail
2 -lundi	2 +lundi = cobol
3 -mardi	3 +mardi = java
4 -mercredi	4 +mercredi = hibernate
5 -jeudi	5 +jeudi = SQL
6 -vendredi	6 +vendredi = XML
7 -samedi & dimanche	7 +samedi & dimanche = PLS

Pour terminer récupérez le master mis à jour sur votre repository local.

En général on ne s'occupe que de sa **branche** et du **master**, on n'intervient pas sur les **branches** des autres sauf en cas de pull request.