
- Commandes GIT -

LO43 - Benjamin MONSERAND

Learn Git Branching

L'outil Learn Git Branching permet d'apprendre et visualiser l'effet des différentes commandes Git.

Git commit :

The screenshot shows the 'Learn Git Branching' application. On the left, a dark-themed text area contains instructions in French: 'Appuyez sur le bouton ci-dessous pour faire un nouveau commit' (Click the button below to make a new commit). Below this is a green button labeled 'git commit'. Further down, it says: 'C'est parti ! Super. Nous venons de faire des modifications sur le dépôt et de sauvegarder celles-ci dans un commit. Ce commit que nous venons de faire a un parent, C1, qui référence le commit sur lequel il est basé.' (It's off to a good start! Super. We just made modifications to the repository and saved them in a commit. This commit we just made has a parent, C1, which references the commit it is based on).

On the right, a light blue background features a commit history diagram. It shows a vertical sequence of three pink circles labeled C0, C1, and C2. Arrows point upwards from C2 to C1, and from C1 to C0. A pink arrow points from a label 'master*' to the C2 circle.

The screenshot shows the 'Learn Git Branching' application with a terminal window open. The terminal title is 'Apprenez Git Branching'. It has buttons for 'Afficher les cibles' (Show targets), 'Level Introduction aux commits avec Git', and 'Instructions'. The terminal output shows a sequence of commands and their results:

```
$ level intro1 [checked]
$ hint [checked]
Il suffit de saisir 'git commit' deux fois pour réussir !
$ delay 2000 [checked]
$ show goal [checked]
$ git commit [checked]
$ git commit [checked]
```

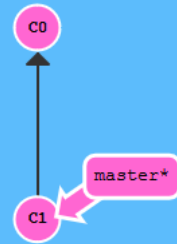
On the right, the commit history diagram is updated. It now shows four pink circles: C0, C1, C2, and C3. Arrows point upwards from C3 to C2, C2 to C1, and C1 to C0. A pink arrow points from a label 'master*' to the C3 circle.

Git branch :

Regardons à quoi ressemblent les branches en pratique.

Nous allons nous positionner (checkout) dans une nouvelle branche appelée `newImage`

```
git branch newImage
```

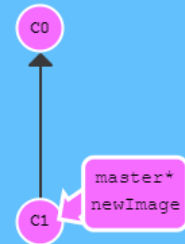


Indiquons à git que nous voulons nous positionner sur la branche avec

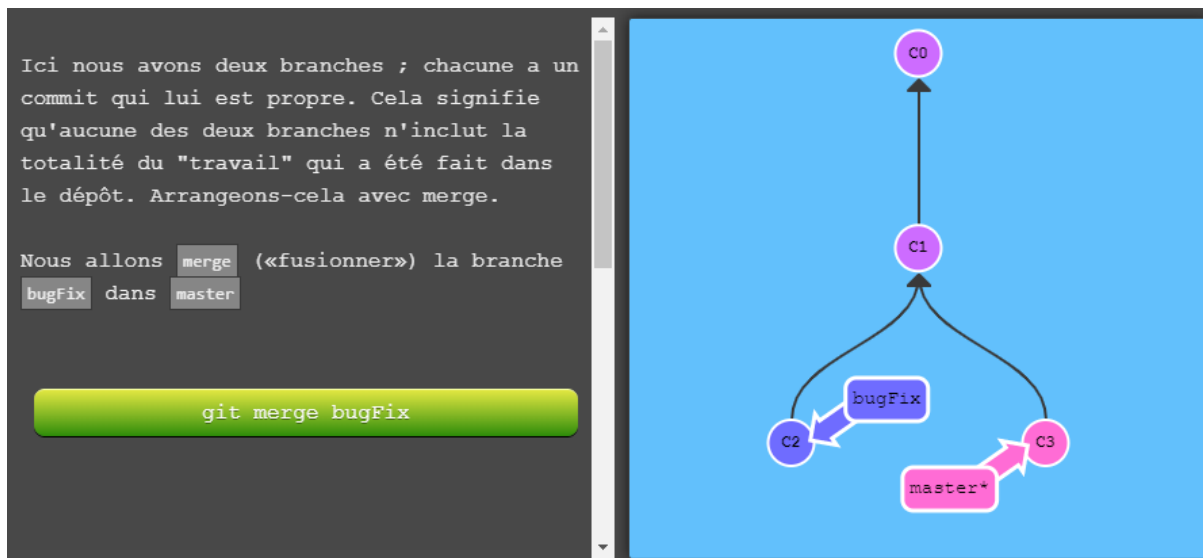
```
git checkout [nom]
```

Cela nous positionne sur la nouvelle branche avant de faire un commit avec nos modifications

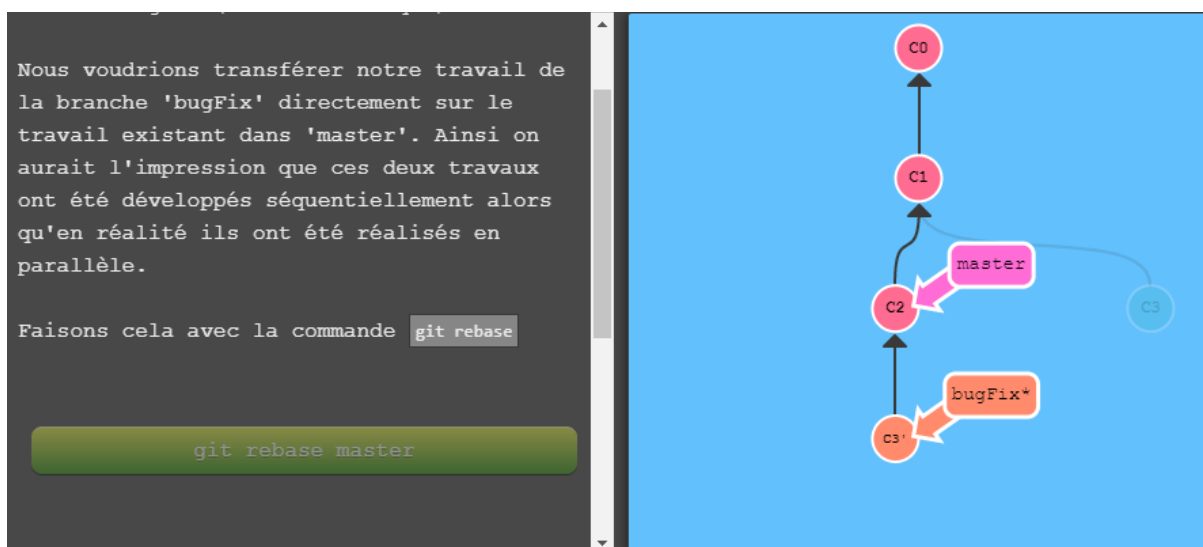
```
git checkout newImage; git commit
```



Git merge :



Git rebase :

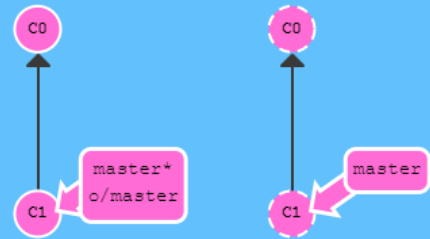


Git clone :

Commençons doucement et regardons à quoi ressemble un dépôt distant (dans notre visualisation).

```
git clone
```

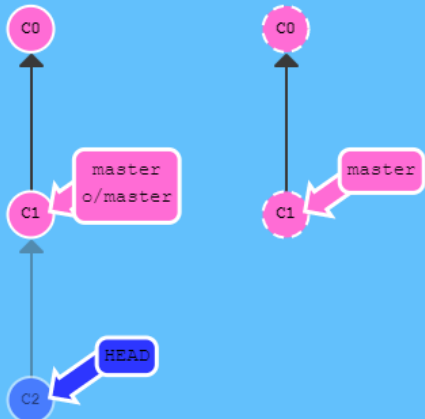
Nous y sommes ! Maintenant nous avons un dépôt distant de notre projet. Cela ressemble fortement à d'habitude, en dehors de quelques changements pour rendre compte des différences -- dans les niveaux suivants vous allez voir comment partager



Rendons-nous sur une branche et regardons ce qui se passe

```
git checkout o/master; git commit
```

Comme vous pouvez le voir, git nous a mis dans le mode "detached" (cf. `HEAD`) puis n'a pas mis à jour `o/master` quand nous avons ajouté un nouveau commit. C'est parce que `o/master` va se mettre à jour uniquement quand le dépôt distant sera mis à jour.



Git pull :

Git Pull

Maintenant que vous avez vu comment rapatrier des données depuis un dépôt distant avec `git fetch`, mettons à jour notre copie de travail pour refléter ces changements !

Il existe en fait beaucoup de façons de faire cela -- une fois que vous avez de nouveaux commits disponibles localement, vous pouvez les incorporer dans votre branche de travail comme s'ils étaient des commits normaux d'autres branches. Cela signifie que pourriez simplement exécuter des commandes comme :

- `git cherry-pick o/master`
- `git rebase o/master`
- `git merge o/master`
- etc., etc.

En fait, le principe de *rapatrier* (fetch) les branches distantes puis les *fusionner* (merge) est si commun que git a en réalité une commande pour faire les deux à la fois ! Cette commande est `git pull`.

Git push :

Git Push

Ok, donc j'ai rapatrié les changements du dépôt distant et je les ai incorporés dans mon travail local. C'est super... mais comment je partage mon travail génial avec tous les autres ?

En fait, la manière d'envoyer du travail à partager fonctionne à l'opposé du téléchargement de travail partagé. Et quel est l'opposé de `git pull` (tire) ? `git push` (pousse) !

`git push` est responsable de l'envoi de vos changements vers un dépôt distant et de la mise à jour de ce dépôt pour incorporer vos commits. Une fois `git push` terminé, tous vos amis peuvent télécharger votre travail depuis le dépôt distant.

Descriptions commandes GIT

Git config :

Git config permet de modifier ou d'accéder aux différents paramètres de git. Plusieurs paramètres peuvent être modifiés comme le nom, l'e-mail, l'éditeur de texte par défaut, l'outil de merge ainsi que l'apparence de git.

Git init :

Git init permet de créer un dépôt git dans le dossier courant. Il crée un dossier .git à l'intérieur des dossiers vers toutes les références, les têtes et les accès à distance. Il crée aussi un fichier HEAD qui renvoie vers la tête de la branche principale master.

Git status :

Git status permet d'afficher le statut des fichiers du dépôt courant. L'état est constitué de la liste des fichiers qui ont été modifiés depuis le dernier commit ou le dernier pull.

```
C:\Coding\C C++\L043-A2019>git status
On branch FB
Your branch and 'origin/FB' have diverged,
and have 3 and 5 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   .idea/.gitignore
    new file:   .idea/L043-A2019.iml
    new file:   .idea/encodings.xml
    new file:   .idea/inspectionProfiles/Project_Default.xml
    new file:   .idea/misc.xml
    new file:   .idea/modules.xml
    new file:   .idea/vcs.xml
    new file:   CMakeLists.txt
    renamed:    helloworld.cpp -> HelloWorld/helloworld.cpp
    new file:   cmake-build-debug/CMakeCache.txt
    new file:   cmake-build-debug/CMakeFiles/3.15.3/CMakeCCompiler.cmake
    new file:   cmake-build-debug/CMakeFiles/3.15.3/CMakeCXXCompiler.cmake
    new file:   cmake-build-debug/CMakeFiles/3.15.3/CMakeDetermineCompilerABI_C.bin
    new file:   cmake-build-debug/CMakeFiles/3.15.3/CMakeDetermineCompilerABI_CXX.bin
```

Git merge :

Git merge permet de combiner deux branches afin de fusionner le contenu des deux parents. Cette commande est essentielle pour intégrer de nouvelles fonctionnalités venant de deux branches parallèles.

Git diff :

Git diff permet de voir la différence de contenu entre la version locale et le dernier commit. On peut y voir les fichiers modifiés, les ajouts et suppressions, ainsi que le nombre total de modifications.

```
diff --git a/cmake-build-debug/CMakeFiles/main.dir/CXX.includecache b/cmake-build-debug/CMakeFiles/main.dir/CXX.includecache
index 162bfea..ae79b6c 100644
--- a/cmake-build-debug/CMakeFiles/main.dir/CXX.includecache
+++ b/cmake-build-debug/CMakeFiles/main.dir/CXX.includecache
@@ -6,17 +6,31 @@
 #IncludeRegexTransform:
+ C:/Coding/C++/L043-A2019/Server/Server.cpp
+ C:/Coding/C++/L043-A2019/src/Sensor/Humidity.h
+ Sensor.h
+ C:/Coding/C++/L043-A2019/src/Sensor/Sensor.h
+
+ C:/Coding/C++/L043-A2019/src/Sensor/Sensor.h
+
+ C:/Coding/C++/L043-A2019/src/Server/Server.cpp
```