



Formation Git, Drush et Console

Animée par Romain JARRAUD

Stagiaires et formateur

- **Stagiaires**

- Nom et profil ?
- Comment avez-vous découvert Drupal ?
- Qu'attendez-vous de cette formation ?

- **Formateur Romain JARRAUD**

- Développeur web depuis 1998.
- A commencé par faire du **développement Drupal**, et aujourd'hui fait de l'**animation de formations et du consulting Drupal**.
- Contact : romain.jarraud@drupalfrance.com.

Objectifs de la formation

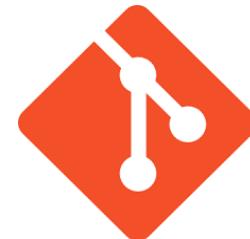
- Comprendre l'intérêt d'un système de version de code comme ***Git***.
- Savoir utiliser ***Git*** en connaissant les commandes de base.
- Mettre en place un Workflow de travail.
- Les outils en ligne de commande (CLI) : ***Drush*** et ***Drupal Console***.



Qu'est-ce que Git ?

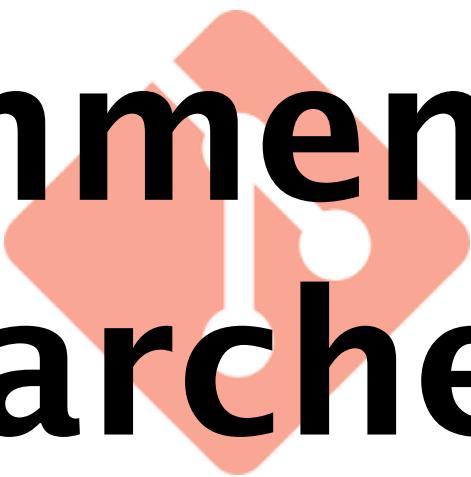
Qu'est-ce que Git?

- Git est un logiciel de gestion de versions décentralisé.
- Git a été créé par **Linus TORSVALDS** (créateur de Linux).



Objectif de Git

- Améliorer le travail collaboratif.
- Paralléliser les développements.
- Garder un historique.
- Déployer
- Pouvoir revenir à une version précédente.
- Créer de l'openSource.
- Pouvoir travailler hors ligne.



Comment ça marche ?

Comment ça marche?

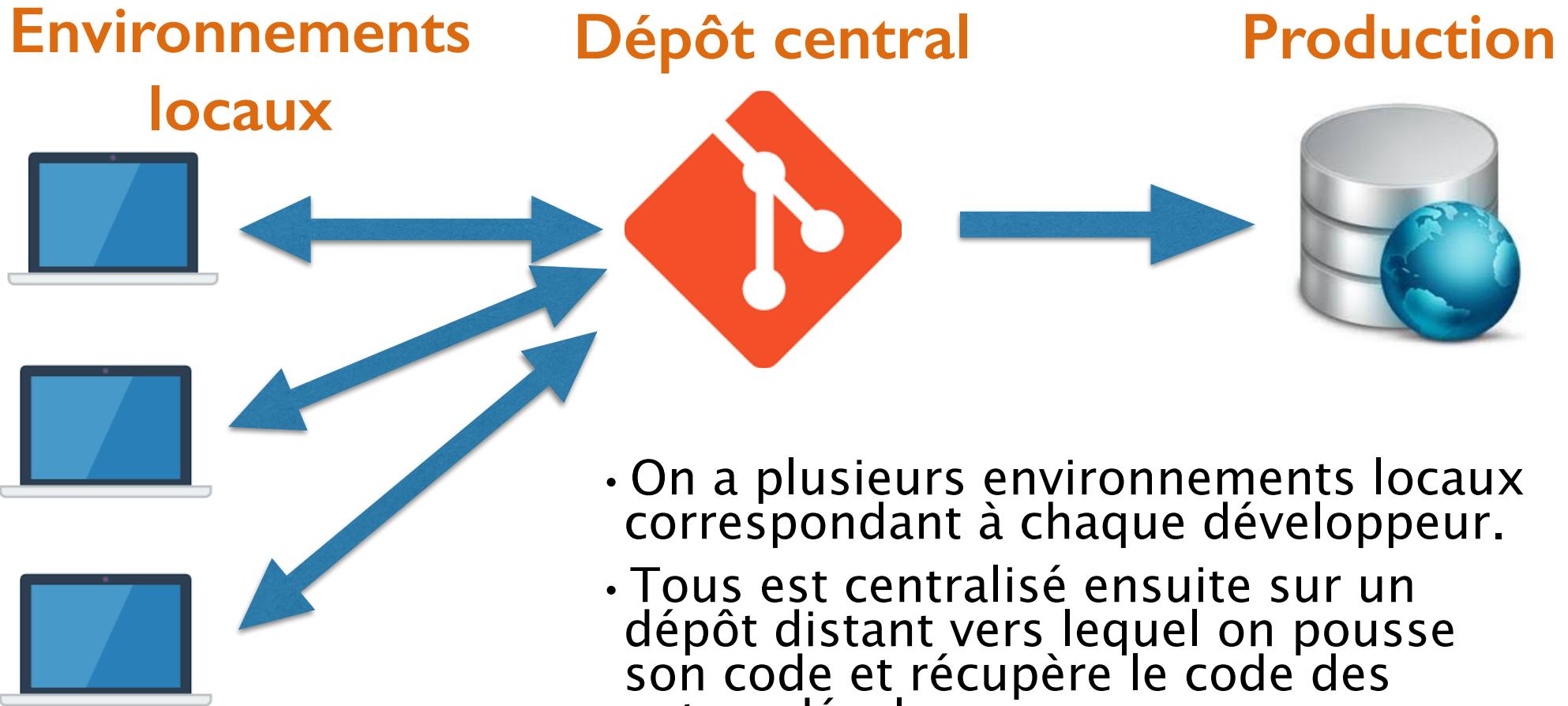
Travail

Index

Stage

- Localement on travaille toujours dans le Workspace (espace de travail). Cet espace correspond à l'état actuel des fichiers.
- L'index regroupe les fichiers que l'on a ajoutés à *Git*. L'état de ces fichiers correspond à l'instant où ils ont été ajoutés. Ainsi les fichiers de l'index sont la plupart du temps dans un état différent de ceux dans l'espace de travail.
- Le stage regroupe les fichiers commités. Ce sont ceux qui sont prêts à être déployés.

Comment ça marche?



Les commandes de base

Git - environnement local

- Lorsque l'on a un dossier suivi par *Git*, on dispose de 3 espaces distincts :
 - **Travail** (workspace) : ensemble des fichiers.
 - **Index** : ensemble des fichiers prêts à être commités.
 - **Stage** : regroupe les commits.
- Pour connaître l'état dans lequel sont les fichiers, on utilise la commande *Git status*.

```
MacBook-Air-de-Romain:git_test romanjarraud$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  fichier.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    fichier2.txt

MacBook-Air-de-Romain:git_test romanjarraud$ █
```

Git - initialisation

Git init

Travail

Index

Stage

Espace de travail

Index

Commits

Git status

(montre les fichiers de l'index non commités,
les fichiers de l'espace de travail non ajoutés à l'index
et les fichiers qui ne sont pas encore suivis par *Git*)

Git - ajout d'un nouveau fichier à l'index

- Pour ajouter un nouveau fichier (*Untracked files*) à l'index on utilise la commande *git add nom_du_fichier*.
- Les fichiers prêts à être commis sont listés sous *Changes to be committed*.

```
[MacBook-Air-de-Romain:git_test romanjarraud$ git add fichier2.txt
[MacBook-Air-de-Romain:git_test romanjarraud$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  fichier.txt
    new file:  fichier2.txt

MacBook-Air-de-Romain:git_test romanjarraud$ ]
```

Git - ajout d'un fichier modifié à l'index

- Pour ajouter un fichier qui a été modifié (*Changes not staged for commit*) à l'index on utilise la commande *git add nom_du_fichier*.

```
new file:  fichier2.txt

[MacBook-Air-de-Romain:git_test romanjarraud$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  fichier.txt
    new file:  fichier2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   fichier2.txt

MacBook-Air-de-Romain:git_test romanjarraud$
```

Git - ajout d'un fichier modifié à l'index

```
[MacBook-Air-de-Romain:git_test romainjarraud$ git add fichier2.txt
[MacBook-Air-de-Romain:git_test romainjarraud$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  fichier.txt
    new file:  fichier2.txt

MacBook-Air-de-Romain:git_test romainjarraud$ ]
```

Git - commits

- Pour créer un nouveau commit, on dispose de la commande ***git commit -m 'Message du commit'***. Il est fortement conseillé de toujours renseigner un message expliquant en quoi consiste le commit. On peut par exemple rappeler un numéro de ticket, qui a participé au code...

```
[MacBook-Air-de-Romain:git_test romainjarraud$ git add fichier2.txt
[MacBook-Air-de-Romain:git_test romainjarraud$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  fichier.txt
    new file:  fichier2.txt

[MacBook-Air-de-Romain:git_test romainjarraud$ git commit -m 'Ajout de fichier.txt et fichier2.txt'
[master (root-commit) c90cee8] Ajout de fichier.txt et fichier2.txt
 2 files changed, 1 insertion(+)
 create mode 100644 fichier.txt
 create mode 100644 fichier2.txt
MacBook-Air-de-Romain:git_test romainjarraud$ ]
```

Git - sortir un fichier de l'index

- Il est possible de sortir un fichier de l'index, afin qu'il ne fasse pas parti du prochain commit. On utilise pour ce faire la commande *git reset HEAD nom_du_fichier*.

```
[MacBook-Air-de-Romain:git_test romanjarraud$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

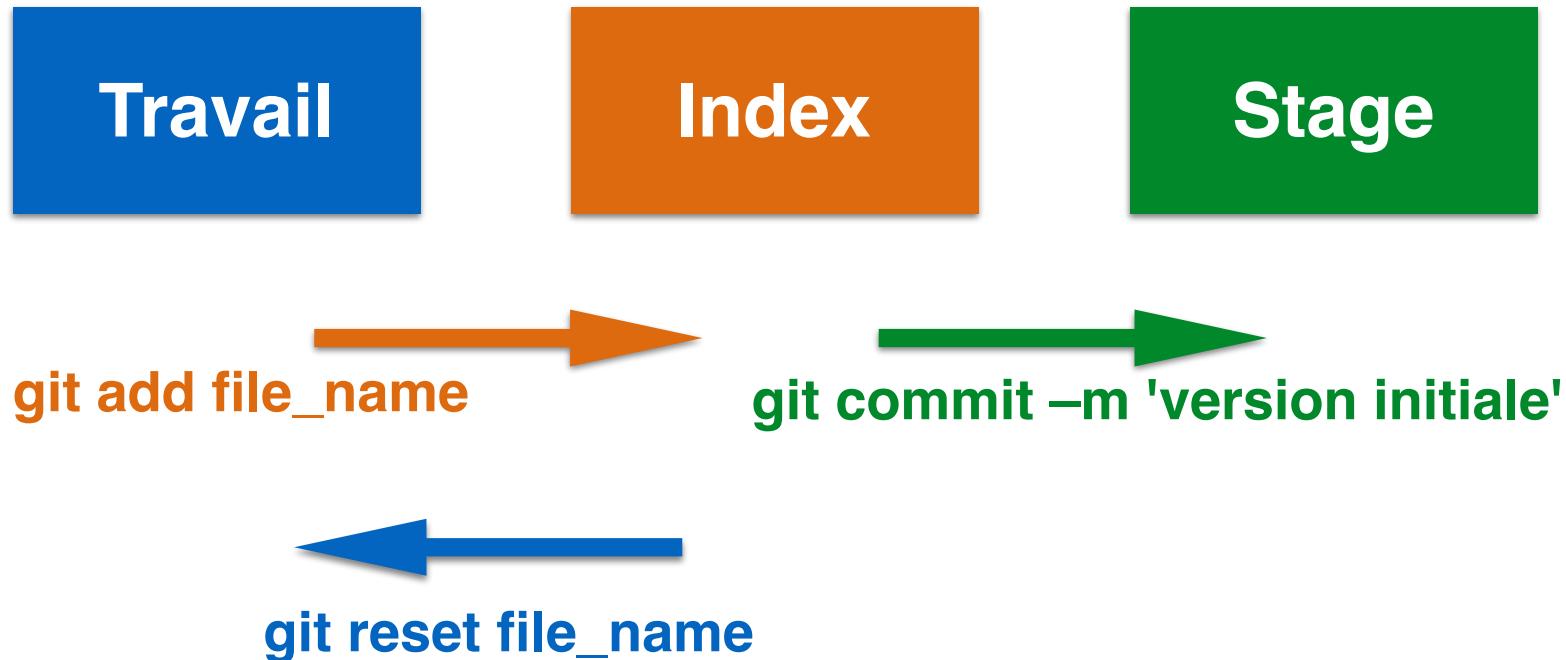
    modified:  fichier2.txt

[MacBook-Air-de-Romain:git_test romanjarraud$ git reset HEAD fichier2.txt
Unstaged changes after reset:
 M      fichier2.txt
[MacBook-Air-de-Romain:git_test romanjarraud$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:  fichier2.txt

no changes added to commit (use "git add" and/or "git commit -a")
MacBook-Air-de-Romain:git_test romanjarraud$ █
```

Git - ajout et commit de fichiers



Commande *git diff*

- La commande ***git diff*** permet de comparer un ou plusieurs fichiers entre les différents états (travail, index et stage).
- Par défaut ***git diff fichier*** compare l'index et l'espace de travail. Cela permet de suivre les modifications d'un fichier depuis son dernier ajout à l'index.



Commande *git diff*

```
[MacBook-Air-de-Romain:git_test romainjarraud$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   fichier2.txt

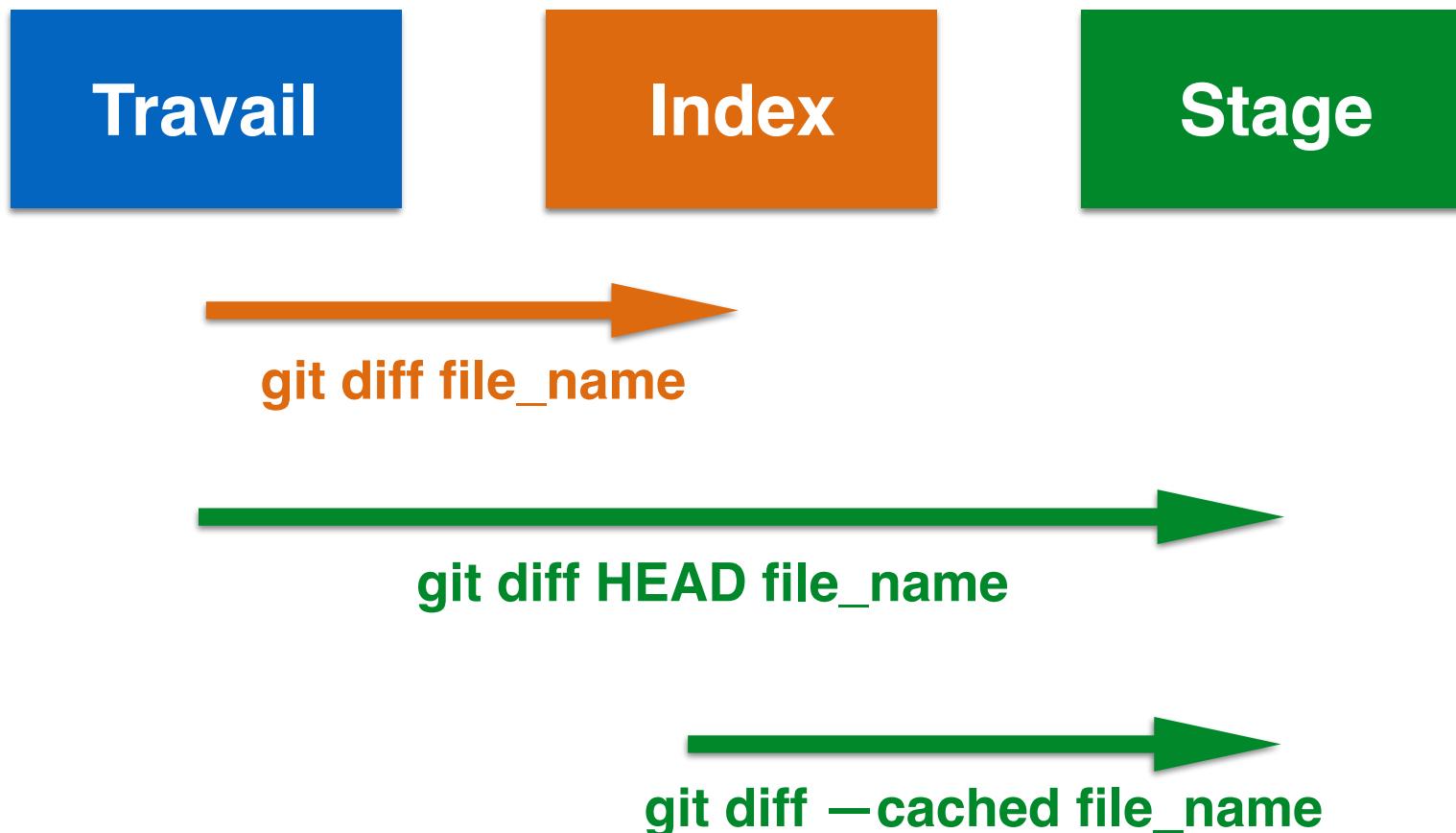
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   fichier2.txt

[MacBook-Air-de-Romain:git_test romainjarraud$ git diff
diff --git a/fichier2.txt b/fichier2.txt
index 9015a7a..1ff7003 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -1 +1 @@
-index
+travail
MacBook-Air-de-Romain:git_test romainjarraud$ ]
```

Commande *git diff*

La commande *git diff* dispose d'options permettant d'indiquer entre quels espaces on souhaite comparer un fichier :



Commande *git diff*

```
modified:   fichier2.txt

[MacBook-Air-de-Romain:git_test romainjarraud$ git diff fichier2.txt
diff --git a/fichier2.txt b/fichier2.txt
index 9015a7a..1ff7003 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -1 +1 @@
-index
+travail
[MacBook-Air-de-Romain:git_test romainjarraud$ git diff HEAD fichier2.txt
diff --git a/fichier2.txt b/fichier2.txt
index 0e26ecd..1ff7003 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -1 +1 @@
-stage
+travail
[MacBook-Air-de-Romain:git_test romainjarraud$ git diff --cached fichier2.txt
diff --git a/fichier2.txt b/fichier2.txt
index 0e26ecd..9015a7a 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -1 +1 @@
-stage
+index
MacBook-Air-de-Romain:git_test romainjarraud$
```

Commande *git diff*

- Il est possible de comparer également un fichier entre différents commits (123 et 456) :

git diff 123 456 fichier

- *Rappel : la commande **git log** liste les commits et leurs identifiants (hash).*

```
MacBook-Air-de-Romain:git_test romainjarraud$ git log
commit e2c7bc269c4bba7495e9abcb7b48805cf6fde16f
Author: romainj <rjarraud@gmail.com>
Date:   Wed May 24 12:18:09 2017 +0200

    Fichier2.txt stage

commit c90cee8b108c4be4a9abee8648cbbe77a56a781e
Author: romainj <rjarraud@gmail.com>
Date:   Fri Mar 31 18:23:22 2017 +0200

    Ajout de fichier.txt et fichier2.txt
MacBook-Air-de-Romain:git_test romainjarraud$ git diff e2c7bc269c4bba7495e9abcb7b48805cf6fde16f c90cee8b108c4be4a9abee8648cbbe77a56a781e fichier2.txt
diff --git a/fichier2.txt b/fichier2.txt
index 0e26ecd..0807e8f 100644
--- a/fichier2.txt
+++ b/fichier2.txt
@@ -1 +1 @@
-stage
+nouveau contenu
MacBook-Air-de-Romain:git_test romainjarraud$
```

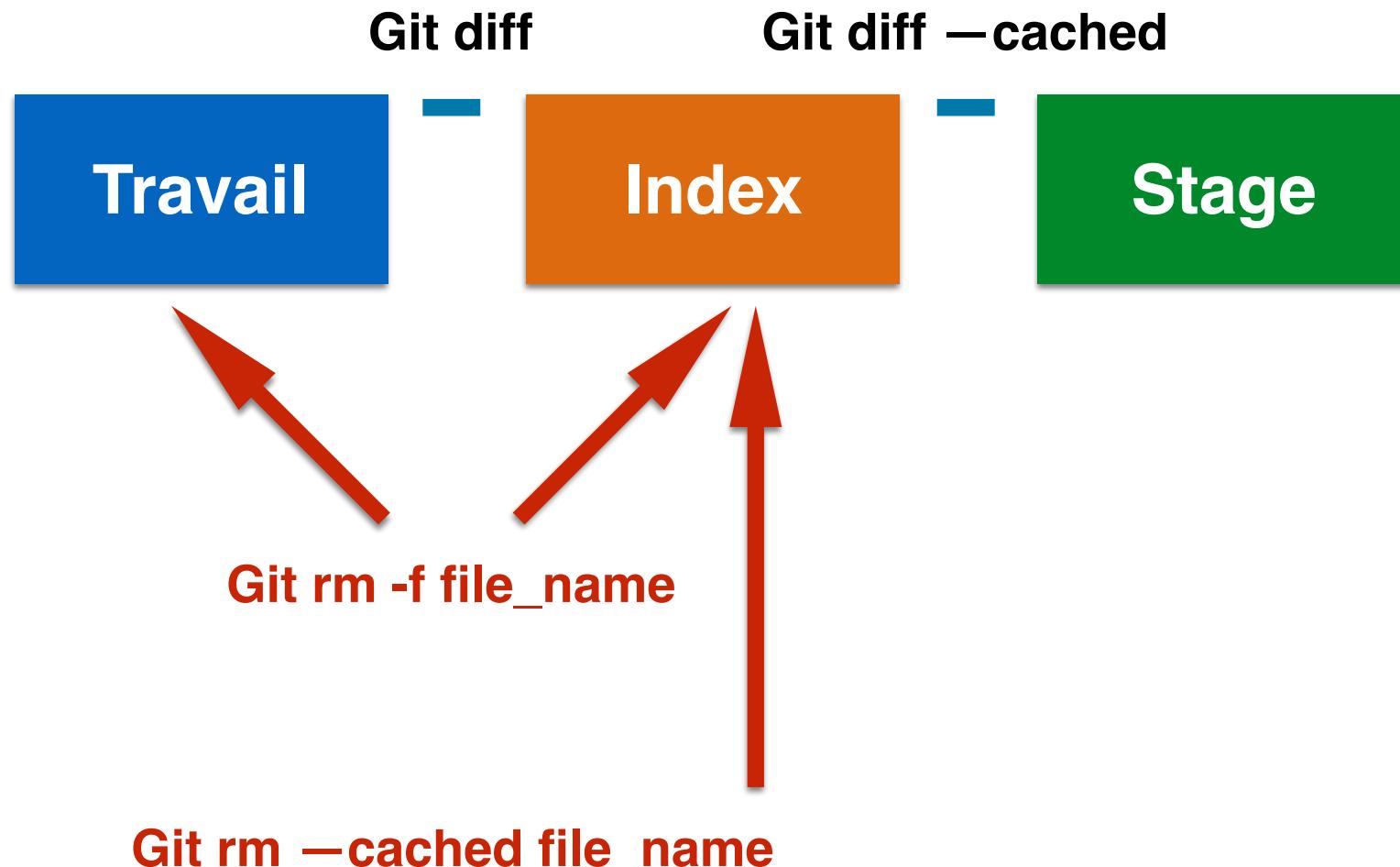
Commande *git add*

- **Git add .** : ajoute à l'index tous les fichiers modifiés ou nouveaux mais pas les fichiers ignorés (*.gitignore*) ou supprimés (rm).
- **Git add -u** : ajoute à l'index tous les fichiers modifiés ou supprimés mais pas nouveaux.
- **Git add *** : ajoute à l'index tous les fichiers même ceux ignorés (*.gitignore*).

Initialiser un repository

- Installer **Git** en local (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>).
- Initialiser un repository local.
- Ajouter un fichier (par exemple */fichier.txt*) à ce repository et lancer la commande *git status*.
- Ajouter ce fichier à l'index et lancer la commande *git status*.
- Commiter ce fichier et lancer la commande *git status*.
- Faire des changements dans le fichier et lancer la commande *git status*.
- Ajouter ce fichier à l'index et lancer la commande *git status*.
- Commiter ce fichier et lancer la commande *git status*.

Git - suppression



Sortir un fichier de *Git*

- Ajouter un nouveau fichier (par exemple */nouveau-fichier.txt*) et lancer un *git status*.
- Ajouter ce fichier à l'index et lancer un *git status*.
- Faites-en sorte de sortir ce fichier de l'index, sans le supprimer de l'espace de travail.
- Lancer un *git status*.

Comande *git log*

Git log

(liste des commits)

Git log -p

(liste des commits avec les différences)

Git log -p – pretty=oneline

(une ligne par commit)

Git - annulation



Git commit —amend
(recommit avec l'index et propose un nouveau message)



Git checkout file_name
(récupère depuis le dernier commit vers la zone de travail)

Annulation

Commande *git remote*

- Lorsque l'on travail en local sur sa machine, il nous faut partager le code avec les autres développeurs.
- On crée pour ce faire un repo accessible de tous, qui est la référence.
- Ensuite chacun peut récupérer le travail des autres et pousser son code vers ce repo.
- Les instances de TEST, PREPROD et PROD ne font que tirer (*pull*) le code depuis ce repo. **On ne devrait jamais avoir à pousser le code depuis l'instance en production vers le repo de partage.**
- La commande *git remote* permet de gérer les repos distants.

Commande *git remote* ajouter un repo distant

git remote add name url_repo

(ajoute le repo *name* accessible via *url_repo*)



Commande *git remote*

Affiche les repos :

git remote

Ajoute le repo NOM hébergé sur URL :

git remote add NOM URL

Affiche les repos avec URL et fetch/push :

git remote -v

Supprime un repo :

git remote rm NOM_REPO

Commande *git clone*

- Lorsqu'un nouveau développeur souhaite récupérer le code d'un projet, on utilise la commande ***git clone*** :

git clone URL

- Pour cloner un repo dans un dossier FOLDER :

git clone URL FOLDER

Commandes *git push/pull*

- Lorsque l'on souhaite partager son travail avec les autres développeurs, il faut pousser son code vers le repo de référence. On utilise la commande :

git push origin master

- De leurs côtés les développeurs peuvent récupérer ce qui est partagé avec la commande :

git pull origin master

- Ainsi chacun travaille en local avec ses propres fichiers, fait des commits et ensuite partage sur le repo de référence.

Commandes *git push/pull*



git push name master
(pousse vers le repo *name* la branche *master*)



git pull origin branche_distante
(récupère une branche distante et fusionne avec une branche locale)

git fetch origin branche_distante:branche_locale
(récupère une branche distante)

Ajouter un repository distant

- Créer un compte GitHub.
- Ajouter la clé SSH.
- Ajouter ce repo comme remote de votre projet local.
- Déployer le repository local sur le remote.

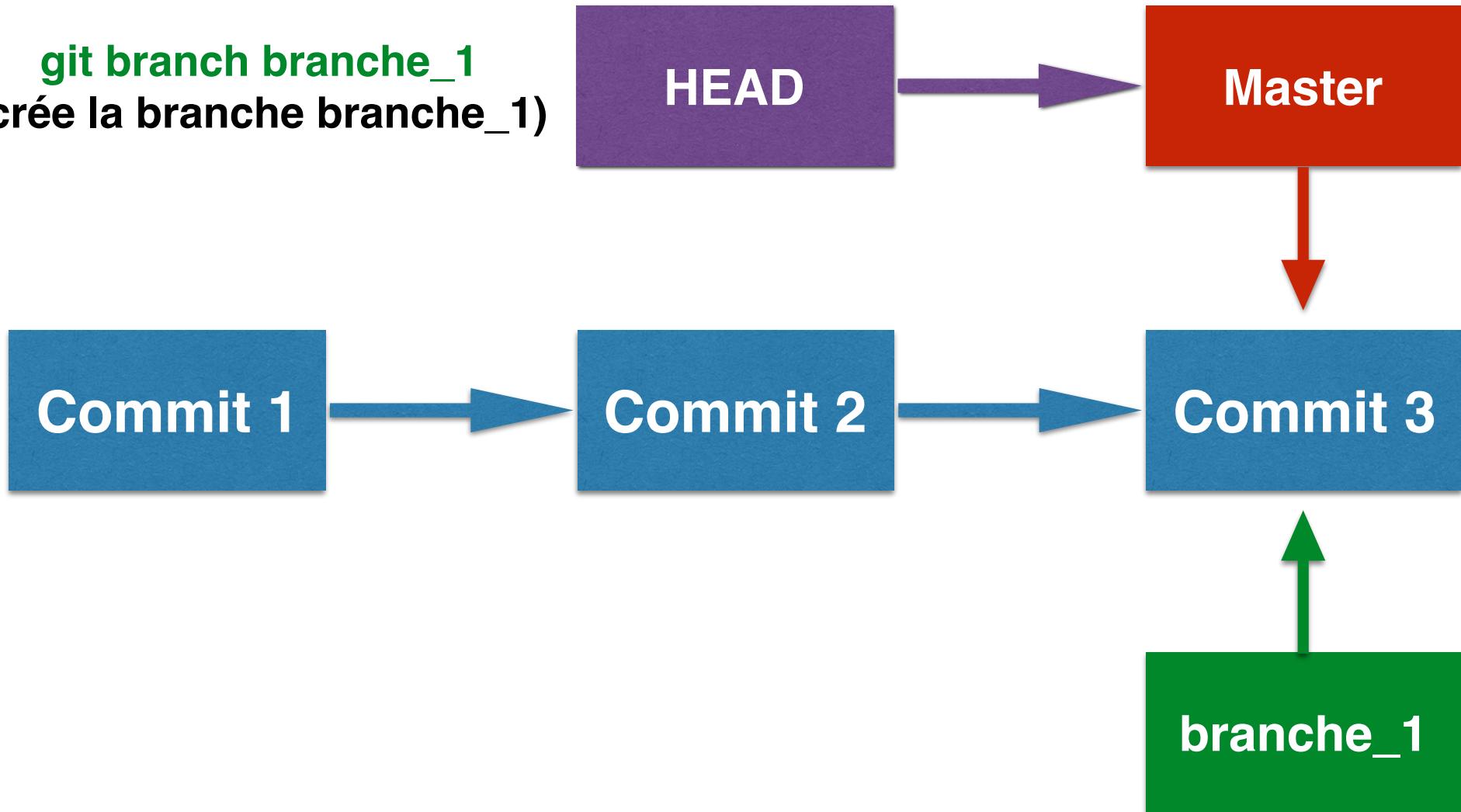
Les branches

Les branches

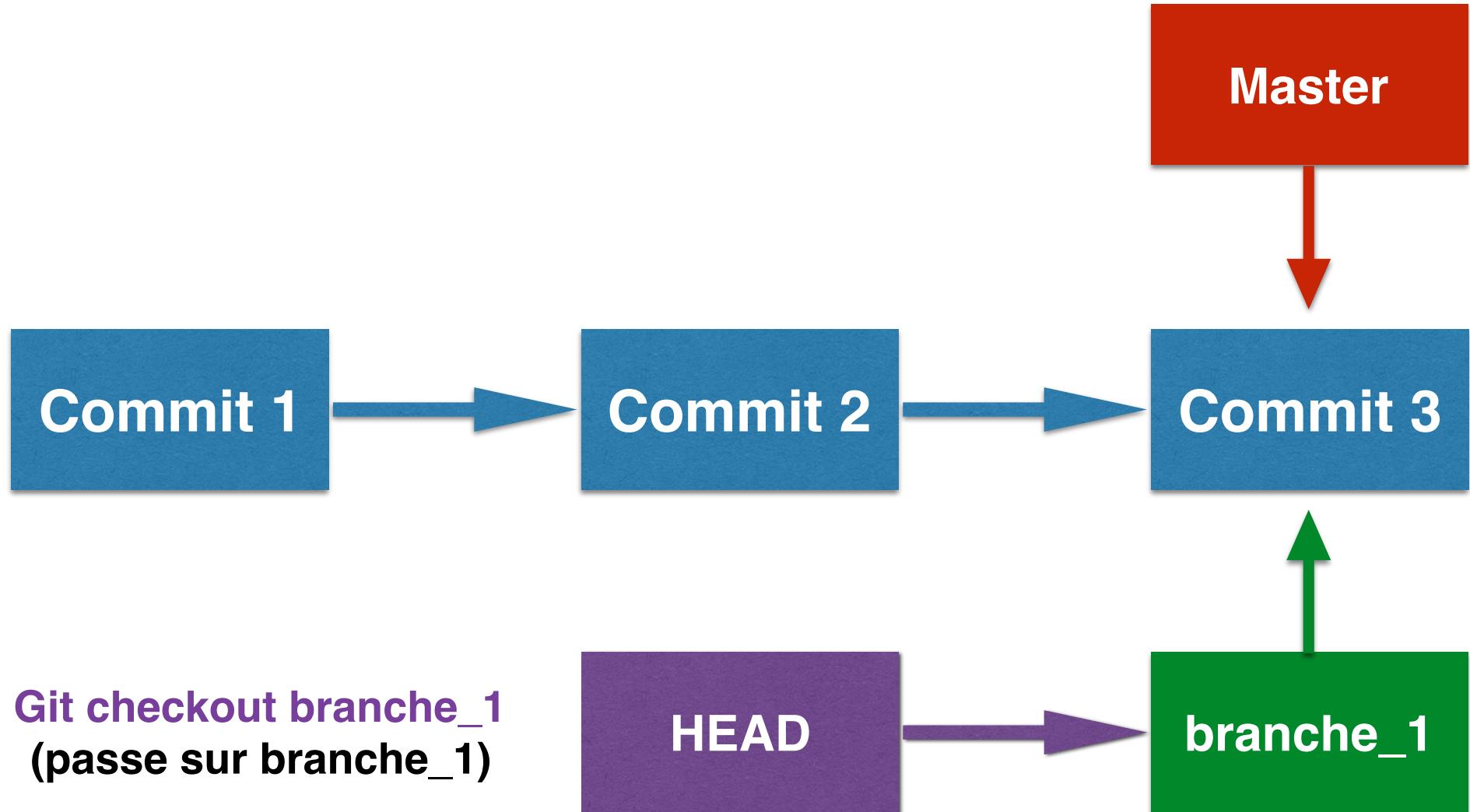
- Les branches permettent d'avoir différentes versions du code.
- On peut ainsi organiser des développements en parallèle, en créant autant de branches que nécessaire.
- Chaque branche correspond généralement à une fonctionnalité bien précise. On isole ainsi chaque fonctionnalité les unes des autres.
- Une fois que le développement d'une fonctionnalité est terminé, la branche correspondante doit être fusionner avec la branche principale (master commun).

Création d'une branche *branche_1*

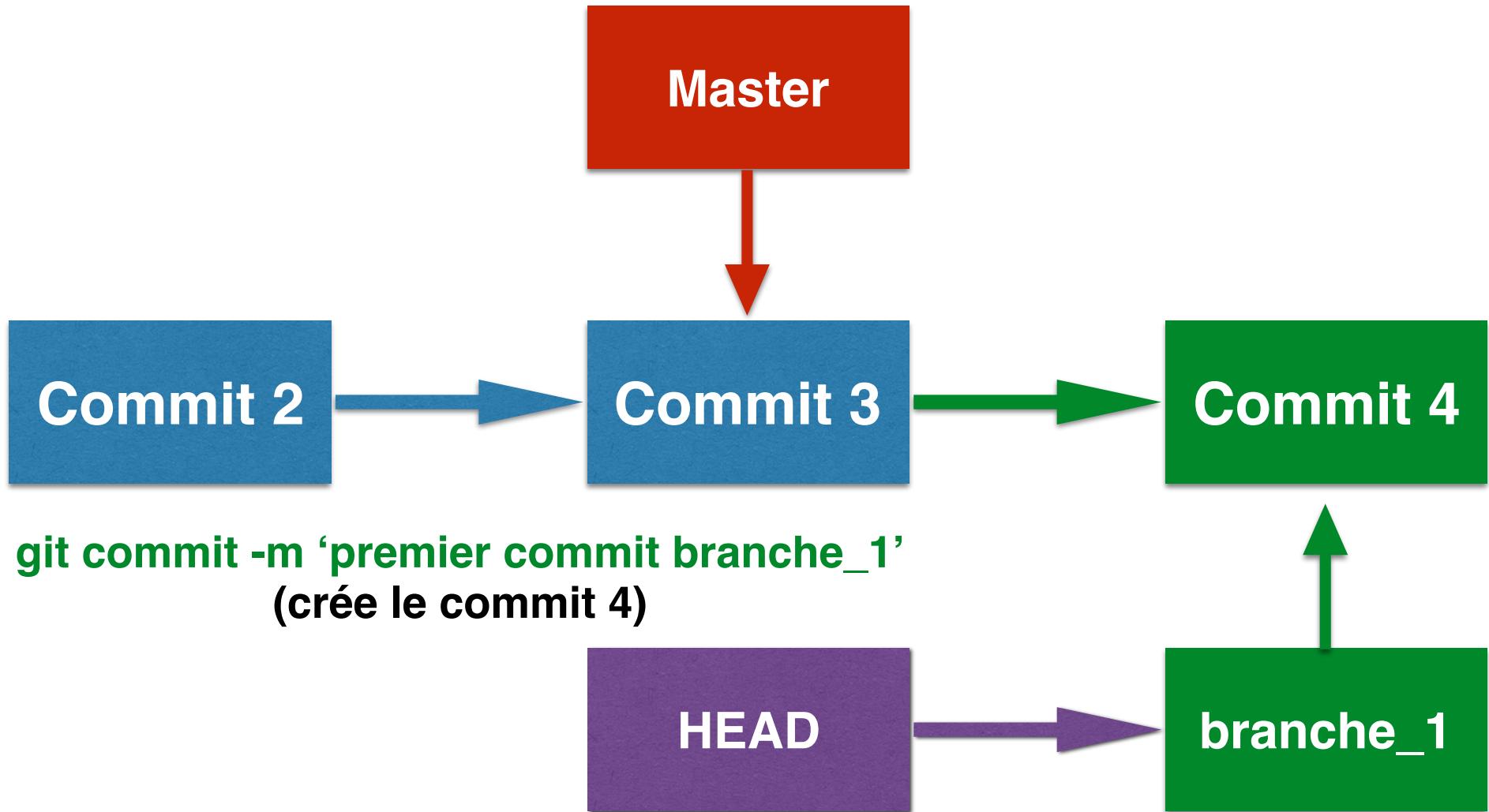
`git branch branche_1`
(crée la branche `branche_1`)



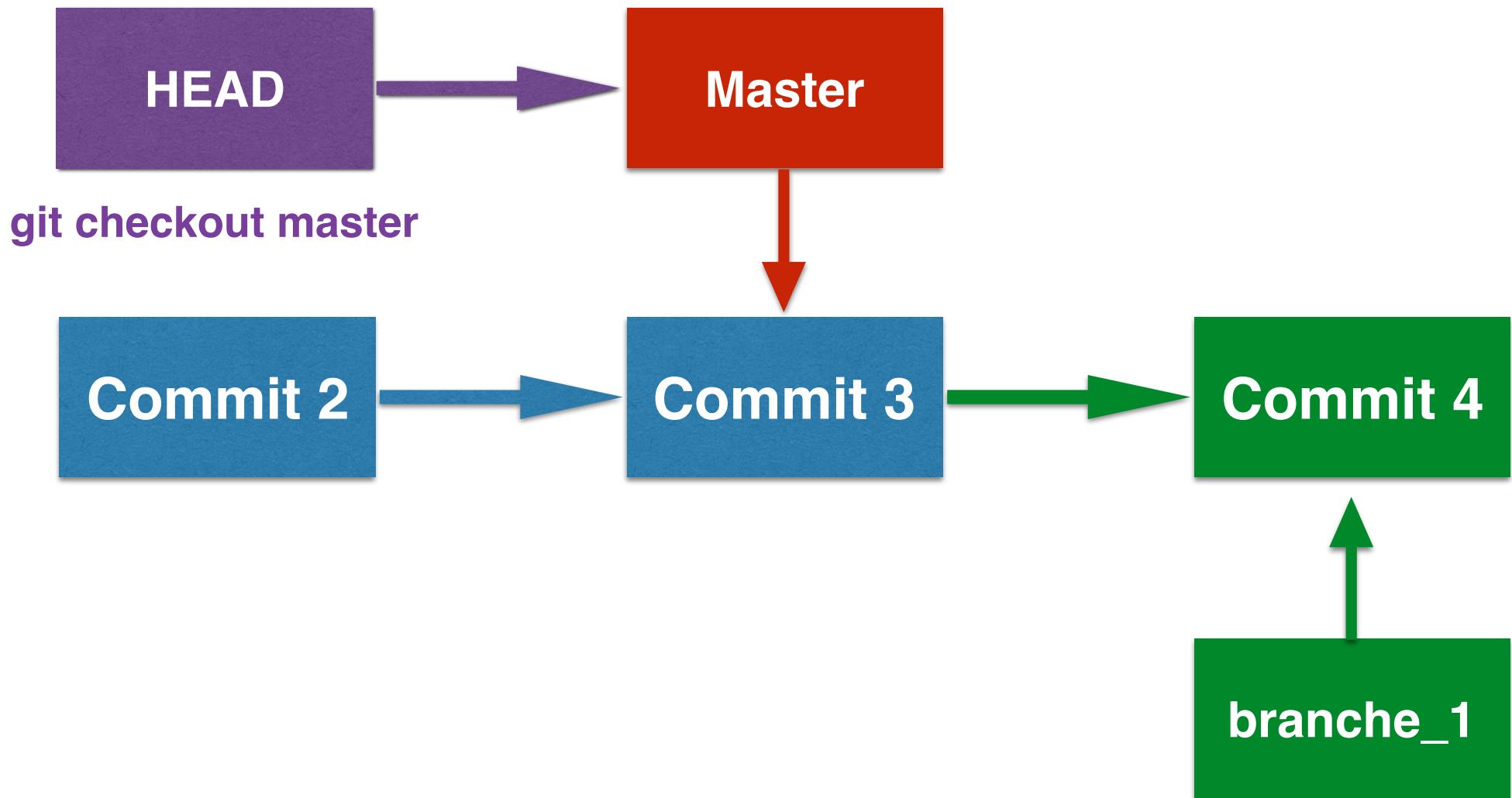
Passage sur la nouvelle branche *branche_1*



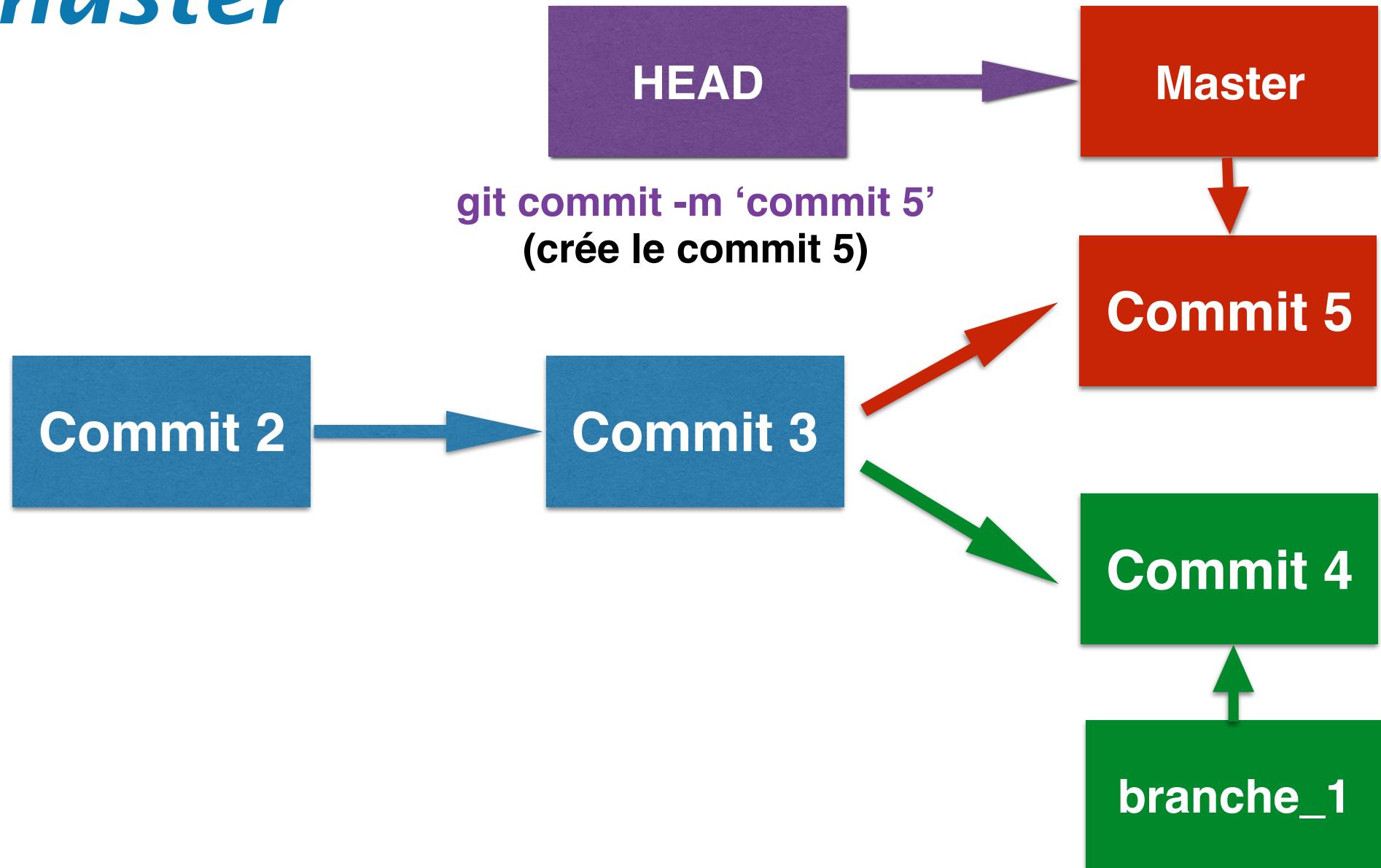
Ajout de commits à la branche *branche_1*



Passage sur la branche de référence (*master*)

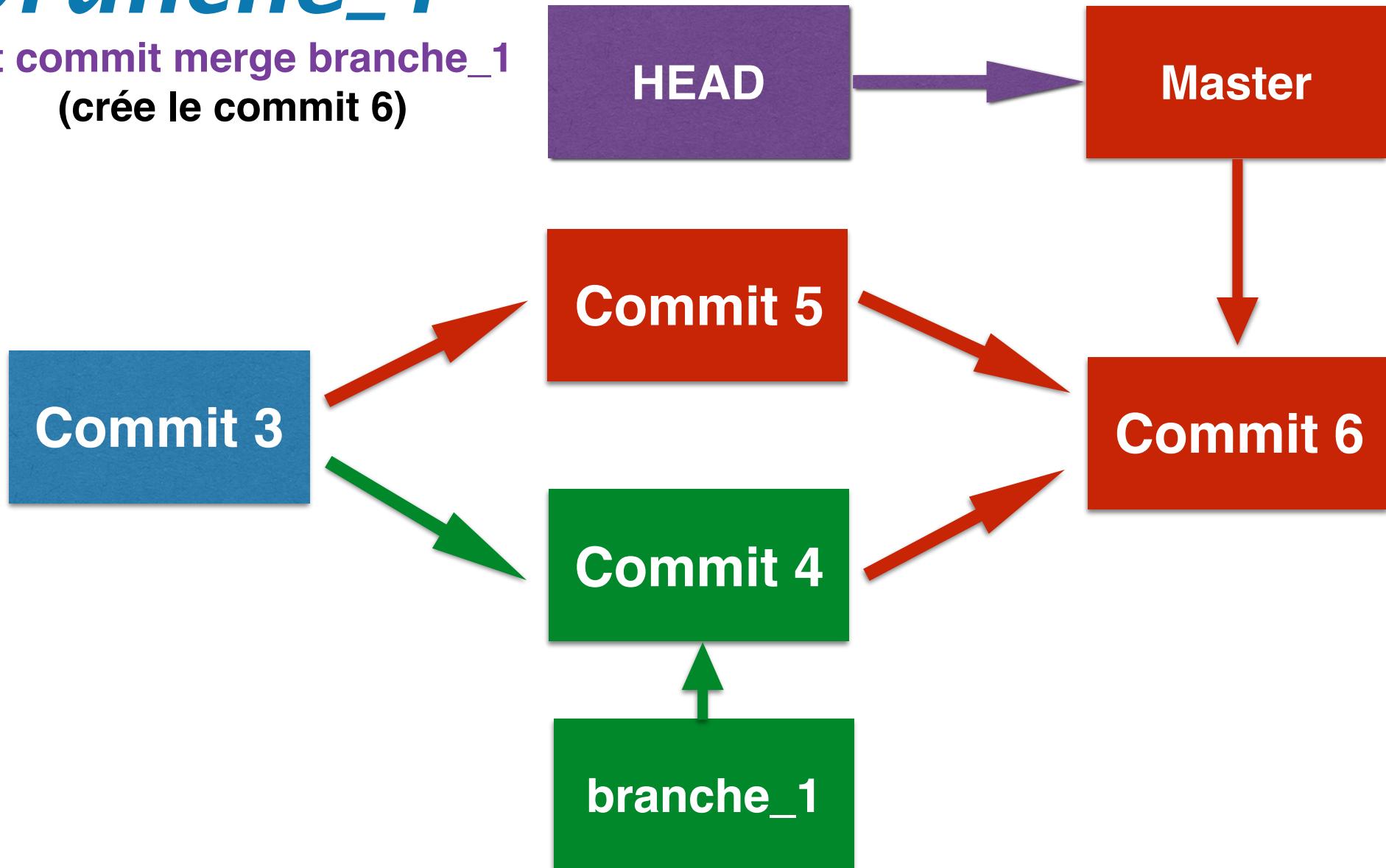


Ajout de commits à la branche *master*



Fusion des branches *master* et *branche_1*

git commit merge branche_1
(crée le commit 6)



Créer une branche

- Créer une nouvelle branche *test*.
- Passer sur cette dernière.
- Modifier le premier fichier que vous avez créé.
- Lancer un *git status*.
- Ajouter le fichier à l'index puis créer le commit correspondant.
- Rapatrier ce commit sur la branche *master*.
- Déployer les changements sur le repository distant.

Les patches

Les patches

- Git permet de créer/appliquer des patches entre des versions différentes d'un ou plusieurs fichiers et/ou entre différentes branches.
- Les commandes à connaître sont : *git apply* et *git diff*.

git apply nom_fichier_patch
(applique le patch nom_fichier_patch)

git diff > nom_fichier_patch
(crée le patch nom_fichier_patch)

Les patches

- la commande **git diff** indique les modifications apportées entre les fichiers de travail et ceux dans l'index.
- On peut ajouter en argument le nom du fichier afin de n'indiquer que les modifications apportées au fichier en question :

git diff NOM_DU_FICHIER

- Il est aussi possible d'indiquer en argument le nom d'une branche afin de la comparer avec la branche de travail :

git diff NOM_DE_LA_BRANCHE

- En ajoutant le nom d'un fichier :

git diff BRANCHE FICHIER

```
macbook-air-de-romain:git_test romainjarraud$ git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   fichier.txt

macbook-air-de-romain:git_test romainjarraud$ git diff
diff --git a/fichier.txt b/fichier.txt
index a67a846..4af3043 100644
--- a/fichier.txt
+++ b/fichier.txt
@@ -1 +1,2 @@
  branche de test
+addition
macbook-air-de-romain:git_test romainjarraud$
```

```
macbook-air-de-romain:git_test romainjarraud$ git diff master fichier.txt
diff --git a/fichier.txt b/fichier.txt
index e69de29..4af3043 100644
--- a/fichier.txt
+++ b/fichier.txt
@@ -0,0 +1,2 @@
+branche de test
+addition
macbook-air-de-romain:git_test romainjarraud$
```

Les Tags

Les tags

- Un *tag* permet de définir une version stable du code.
- Par exemple lors du développement d'un module on crée un tag pour chaque version du module.
- On dispose de tags légers et de tags annotés. Un tag annoté contient le nom du tagger, la date du tag et éventuellement un message de description. Un tag léger n'est qu'un pointeur vers un commit.
- Pour lister les tags, on utilise la commande :

git tag

Les tags

- Pour créer un tag léger :

git tag 1.0

- Pour créer un tag annoté :

git tag -a 1.0 -m 'Description du tag.'

- Pour lister les tags :

git tag

- Pour afficher un tag :

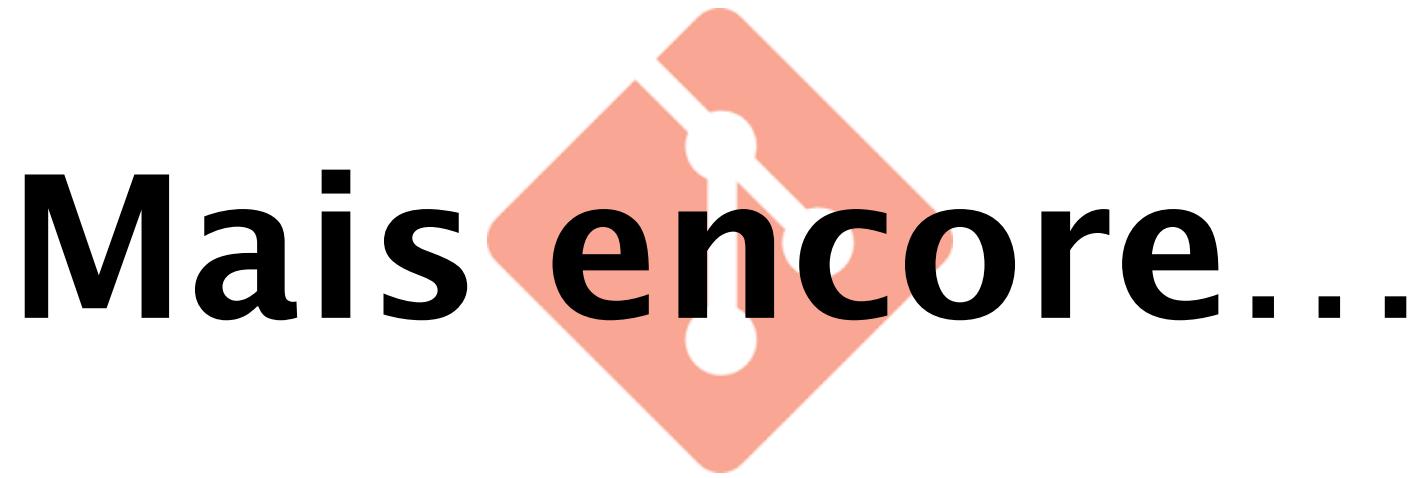
git show TAG

- Pour pousser un tag vers le repo de référence :

git push origin TAG

- Pour pousser tous les nouveaux tags vers le repo de référence :

git push origin --tags



Mais encore...

Fichiers */.Gitignore*

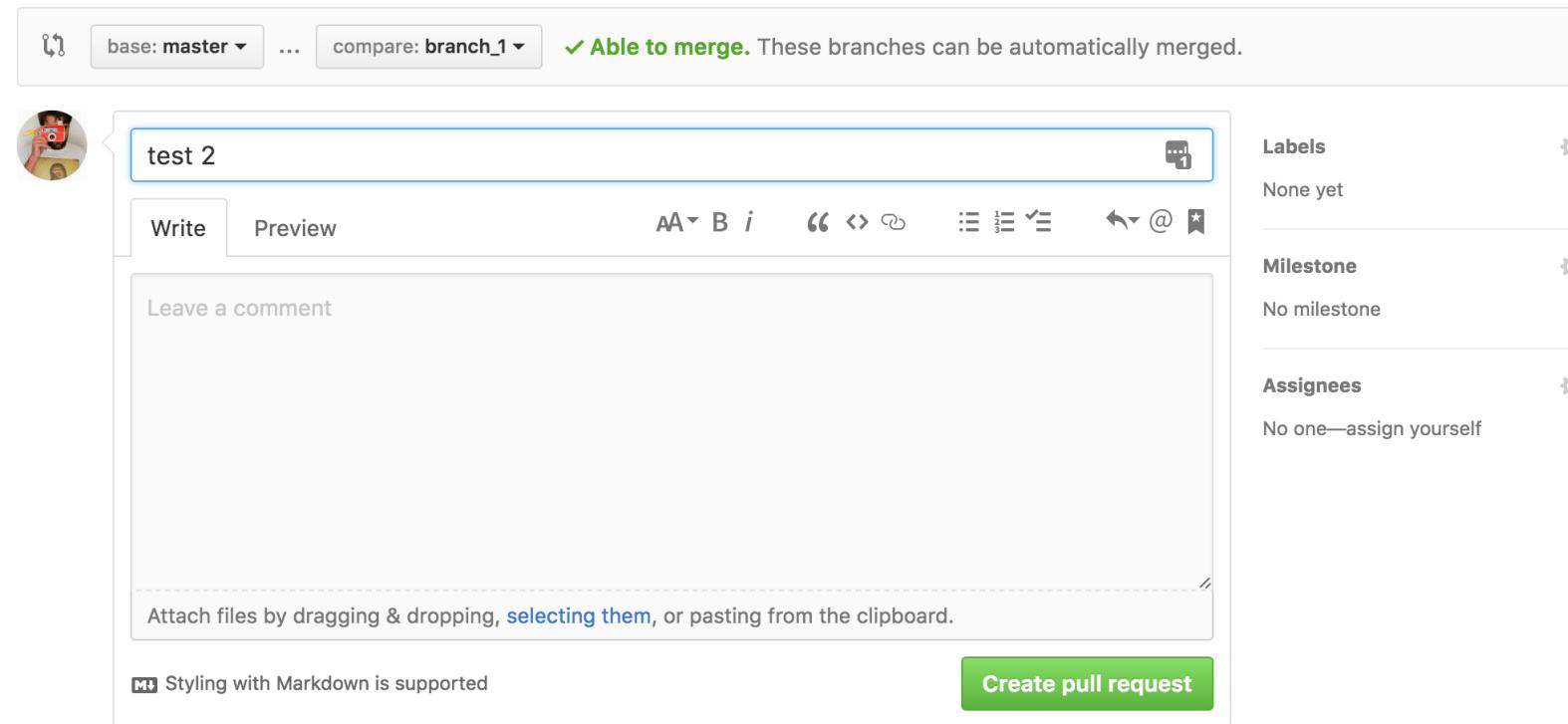
- Le fichier */.gitignore* permet d'indiquer les fichiers à ne pas inclure dans le dépôt Git.
- C'est à vous de définir ces fichiers en fonction du projet et de votre environnement local (attention par exemple aux fichiers *.DS_Store* sous Mac).
- La commande *git add .* tient compte du fichier */.gitignore* mais pas la commande *git add **.

```
1 # This file contains default .gitignore rules. To use it, copy it to .gitignore,
2 # and it will cause files like your settings.php and user-uploaded files to be
3 # excluded from Git version control. This is a common strategy to avoid
4 # accidentally including private information in public repositories and patch
5 # files.
6 #
7 # Because .gitignore can be specific to your site, this file has a different
8 # name; updating Drupal core will not override your custom .gitignore file.
9 #
10 # Ignore core when managing all of a project's dependencies with Composer
11 # including Drupal core.
12 # core
13 #
14 # Core's dependencies are managed with Composer.
15 vendor
16 #
17 # Ignore configuration files that may contain sensitive information.
18 sites/*/settings*.php
19 sites/*/services*.yml
20 #
21 # Ignore paths that contain user-generated content.
22 sites/*/files
23 sites/*/private
24 #
25 # Ignore SimpleTest multi-site environment.
26 sites/simpletest
27 #
28 # If you prefer to store your .gitignore file in the sites/ folder, comment
29 # or delete the previous settings and uncomment the following ones, instead.
30 #
31 # Ignore configuration files that may contain sensitive information.
32 # */settings*.php
33 #
34 # Ignore paths that contain user-generated content.
35 # */files
36 # */private
37 #
38 # Ignore SimpleTest multi-site environment.
39 # simpletest
40 #
41 # Ignore core phpcs.xml and phpunit.xml.
42 core/phpcs.xml
43 core/phpunit.xml
```

Merge request

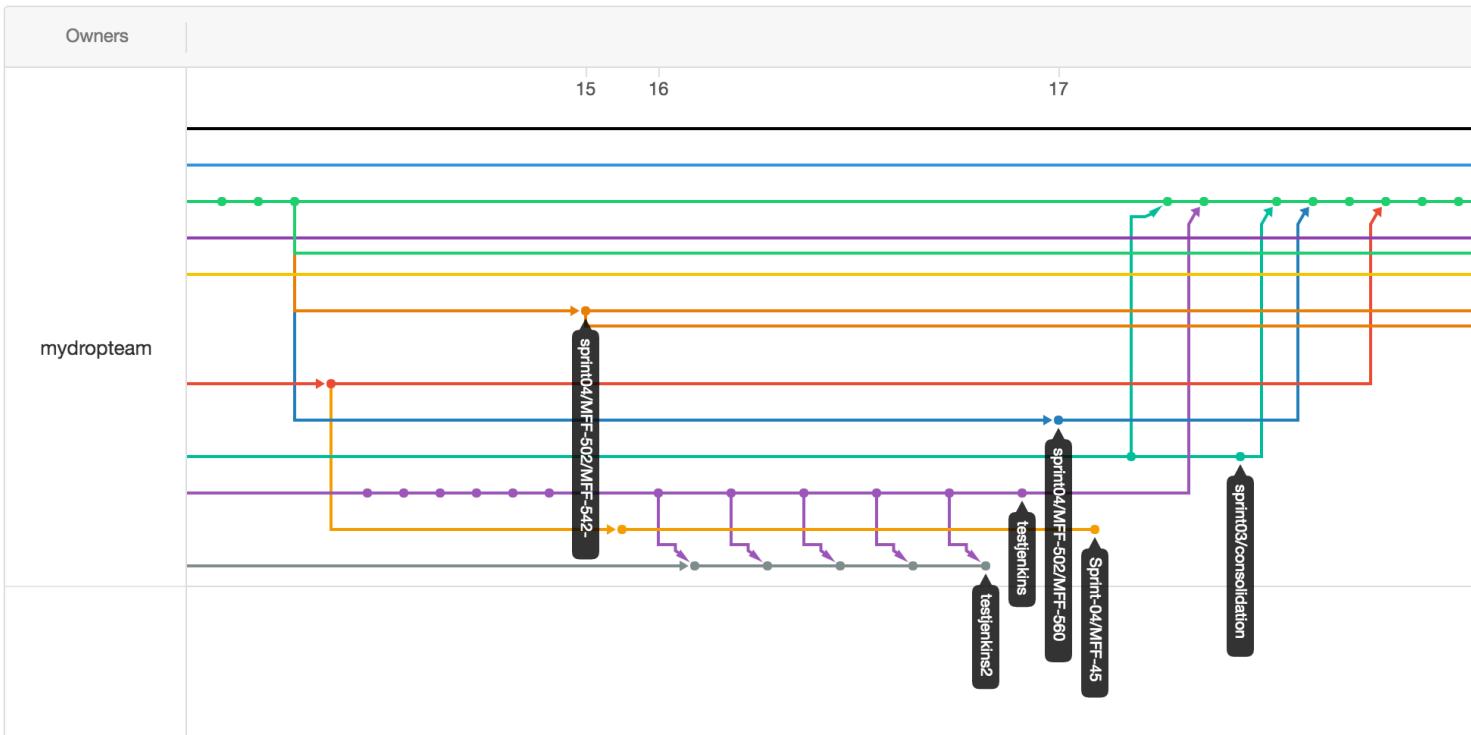
Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

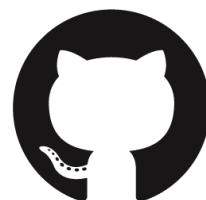


The screenshot shows a pull request creation interface. At the top, there are dropdown menus for 'base: master' and 'compare: branch_1'. A green success message indicates 'Able to merge. These branches can be automatically merged.' Below this, a title 'test 2' is entered. The interface includes a rich text editor toolbar with options like bold, italic, and code blocks. A large text area for comments is labeled 'Leave a comment'. Below it, a dashed line indicates where files can be attached. A note says 'Styling with Markdown is supported'. On the right, there are sections for 'Labels' (None yet), 'Milestone' (No milestone), and 'Assignees' (No one—assign yourself). A prominent green button at the bottom right says 'Create pull request'.

Graph network



Git servers



GitHub

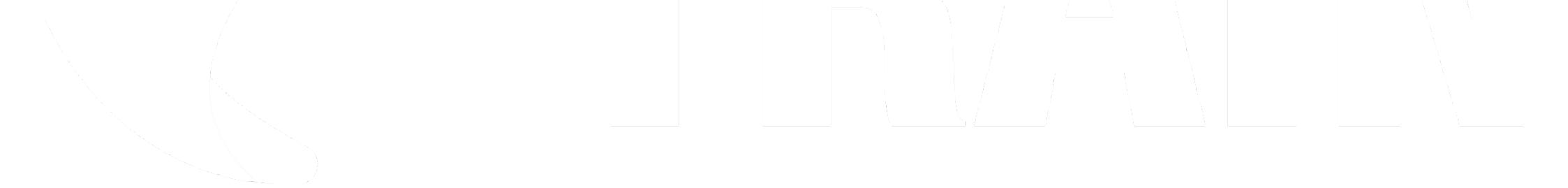


GitLab



Les interfaces

- Certains logiciels proposent une interface graphique afin de s'y retrouver plus facilement entre les commits et les branches :
 - SourceTree
 - Tower
 - GitHub Desktop
 - ...
- Plus d'informations sur [*https://Git-scm.com/download/gui/linux*](https://Git-scm.com/download/gui/linux).



Drush et Console



Qu'est-ce qu'on peut faire?

- On dispose de 2 outils en ligne de commande permettant d'administrer un site sous *Drupal* :
 - **Drush** = **D**rupal **S**hell.
 - **Console** = composant Symfony.
- Chacun a des commandes propres et ne permet pas forcément de faire la même chose qu'avec l'autre. **Drush** se destine davantage à l'administration du site, tandis que **Console** se révèle être un bon générateur de code.

Commandes à connaître

- **Drush :**

- Vider les caches (avec des options) : *drush cr* (cache rebuild).
- Télécharger un module : *drush dl NOM_DU_MODULE*.

- **Console :**

- Vider les caches : *drupal cache:rebuild*.
- Télécharger un module : *drupal module:download*.
- Générer du contenu : *drupal generate:nodes* (fonctionne aussi pour les utilisateurs, termes de taxonomie, vocabulaires de taxonomie et les commentaires).
- Lister les commandes : *drupal list*.
- Mettre à jour Console : *drupal self-update*.
- Lancer les tâches planifiées : *drupal cron:execute*.
- Générer du code : *drupal generate:module* (par exemple).

Drush et Console

- Utiliser *Drush* et/ou *Console* afin de :
 - Faire une sauvegarde de la base de donnée.
 - Vider le cache de rendu de **Drupal**.
 - Faire la mise à jour d'un module.
 - Faire la mise à jour de la base données.
 - Télécharger/installer un nouveau module depuis le site *drupal.org*.
 - Lister les messages loggués (watchdog).
 - Générer le squelette d'un module.

Merci !