
Introduction à Git

MARIUS 'SERENDIP' LOMBARD-PLATET –

SERENDIP@VIA.ECP.FR QUENTIN 'DEGEMER' MADEC –

DEGEMER@VIA.ECP.FR



Sommaire

- I. Les bases
- II. Un peu plus loin...
- III. Encore plus loin...
- IV. Annexes

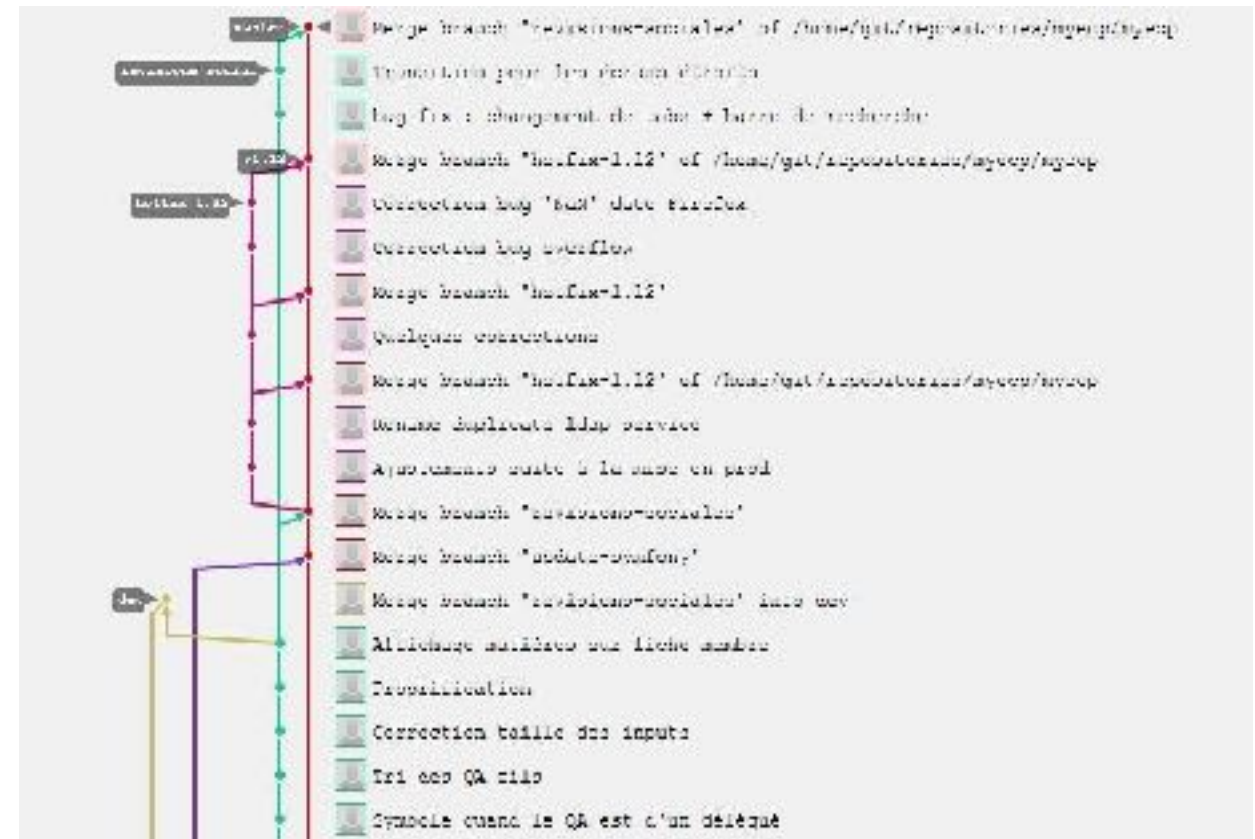
Les

AU COMMENCEMENT ÉTAIT
LE .GIT

bases

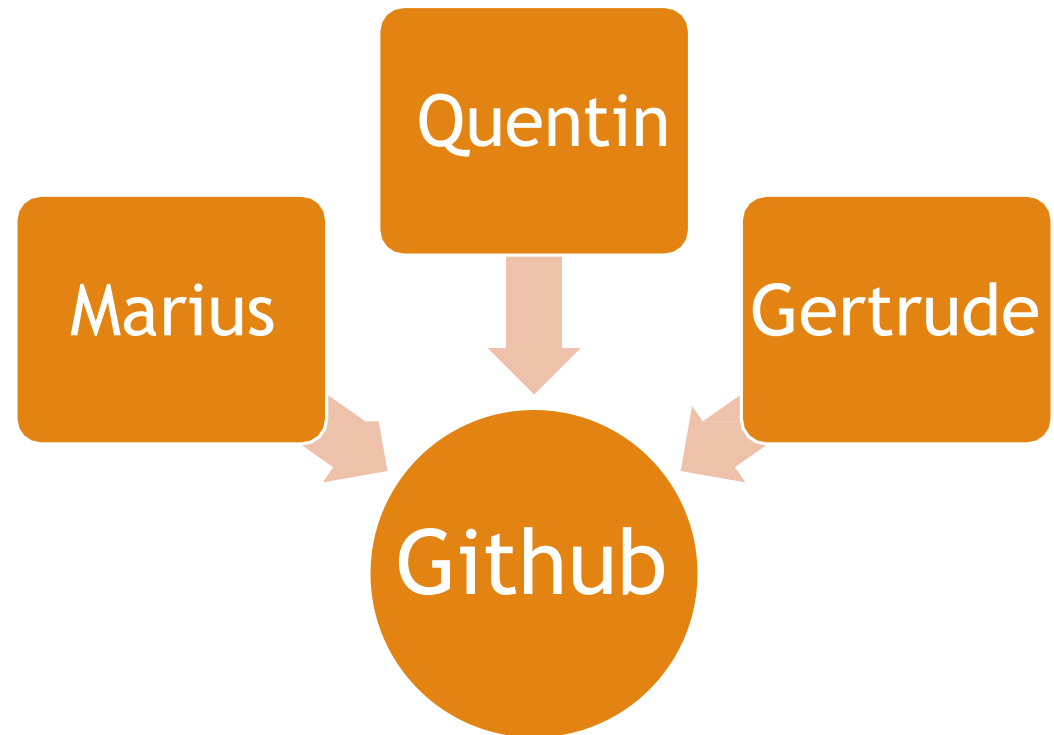
Le principe

- Travailler à plusieurs sur le même projet sans se gêner
- Garder un historique des versions (commits)



Git : un gestionnaire de version

- Dépôt central hébergé par Github ou n'importe quel système basé sur git
- Chaque développeur a son espace local (sur son ordinateur)
- Git repose sur un système de *commits*



Les commits, c'est quoi ?

- Quand un bout de code est satisfaisant, on l'enregistre dans un *commit*
- Git sauvegarde alors cette nouvelle version du code
- Un commit, c'est donc une version du code à un instant donné
- Cela permet d'annuler facilement des modifications, ou de regarder ce qu'ont fait les autres.



Gitlab

*Une plateforme pour les héberger
tous,
Une plateforme pour les forker,
Une plateforme pour les cloner
tous, Et dans les merge
requests les lier.*

- Interface web pour git
- Facilite le *code-review*
- Dépôts publics/privés
- Et quelques plugins intéressants : Issues, Network, ...

- <https://github.com>
- Identification CAS
- Attention : pour utiliser Github, il vous faudra renseigner une clé SSH
- Pour la générer : <https://help.github.com/articles/generating-ssh-keys>

Installation

(à faire une seule fois par ordinateur)

- Unix

- Linux : (sudo) apt-get install git
- Mac : <http://sourceforge.net/projects/git-osx-installer/>

- Lancer un terminal



Windows

- [Télécharger Git : http://git-scm.com/download/win](http://git-scm.com/download/win)
- Installer en gardant les options par défaut
- Lancer « Git Bash »



Utilisateurs de linux : pensez à rajouter l'autocomplétion !

Configuration

- Configuration globale

```
git config --global      user.name  
"Quentin Madec"
```

```
git config --global      user.email  
quentin.madec@student.ecp.fr
```

(à faire une seule fois par ordinateur)

- Création clé ssh

```
mkdir .ssh
```

```
cd .ssh
```

```
ssh-keygen -t rsa -C  
"votreadresse@student.ecp.fr"
```

➤ Laissez les options par défaut, sauf si vous savez ce que vous faites

➤ Mot de passe ? Plus pénible qu'autre chose si le projet n'est pas sensible

Configuration

(à faire une seule fois par ordinateur)

- Récupérer la clé publique
 - Linux/Mac : `cat id_rsa.pub`
 - Windows :
`notepad.exe id_rsa.pub`
- Rajouter la clé sur Github
 - Aller dans Profile Settings > SSH Keys > add SSH key
 - Coller la clef publique obtenue précédemment

```
16:12 serendip@zen ~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDMKTFmPyrA5LbkMWcyWZJNPmDYbeUsSXC0EZeEgDq3v
CSogX6SsHNjshowuG9wpdvWIbCRUNLV2ITPFcz3MYI+SvsX3F4ch3mns7JzzAlInrMKNW/cOEtrhYyxmT
govCfNYktZIS/eK9DXdeCmWJR/IYAAngydfxwgmwfPhlaVmdaYIg33QniKugU7NuQ+hmz9ocLCc6NSDe
8h29d3JdMuOgMr0wAzNxY0kpXjoh716ZdMpZRnxvLvlFiuzK9FEsa0ObzFa/FGDMWRzvSlq4Jg6INBWA0
TDNzSuknIOpceP+7GCtI2uBiTvQEJW45t2fpz9CSw8MTyhp5ue2cB0qH marius.lombard-platet@st
udent.ccp.fr
```

Créer un projet

(à faire pour chaque nouveau projet)

- Créer le dépôt local

```
mkdir nom-du-projet cd  
nom-du-projet git init
```

```
touch README
```

```
git add README
```

```
git commit -m "first commit"
```

- Créer le dépôt distant

Sur Gitlab, créez un nouveau projet

- Relier les deux

Sur votre ordinateur,

```
git remote add origin  
git@gitlab.my.ecp.fr:loginMyECP  
/NomDuProjet.git  
(en une ligne)
```

Récupérer un projet

- Demandez à rentrer dans le projet :

Par l'admin du projet, sur la page Gitlab du projet :
Settings > Members, au moins les droits Developer pour pouvoir contribuer

(travailler sur le projet d'un autre)

- Clonez le dépôt

`git clone git@github.com:URLduProjet.git`
(en une ligne)

```
legeme@MIA ~  
$ git clone git@gitlab.my.ecp.fr:2012madeq/nom-du-projet.git  
Cloning into 'nom du projet'...  
remote: Counting objects: 3, done.  
remote: Total 3 (delta 0), reused 0 (delta 0)  
Receiving objects: 100% (3/3), done.  
Checking connectivity... done.
```

`cd nom-du-projet`

Add : signaler une modification/création

`git add <fichier>`

- Signale à git la création/modification d'un fichier prêt à être sauvegardé
- A faire avant l'enregistrement (le commit)

```
Degemer@NUADA ~/nom-du-projet (master)
$ git add README.md

Degemer@NUADA ~/nom-du-projet (master)
$ touch test

Degemer@NUADA ~/nom-du-projet (master)
$ git add test
```

Status : vérifier l'état de son dépôt

git status

- Vérification de l'état de son dépôt local : quelle branche, quel commit, ...
- Et surtout quels fichiers ont été git add

```
Degemer@NUADA ~/nom-du-projet (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

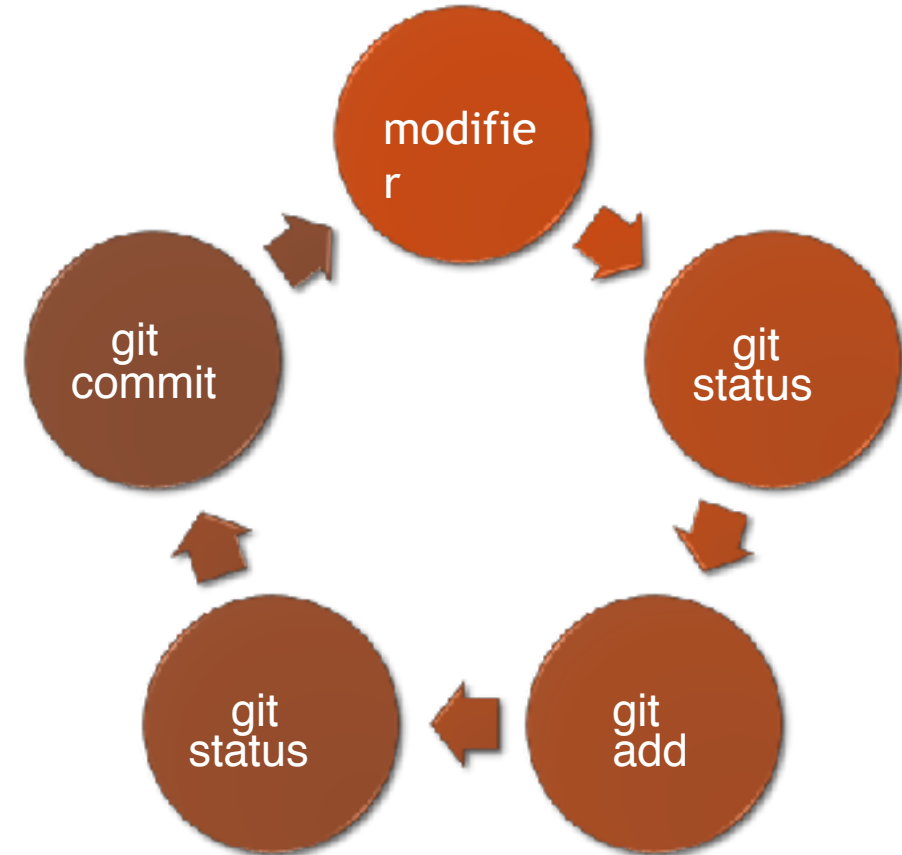
        renamed:    README -> README.md
        new file:   test
```

Commit : enregistrer une modification

`git commit -m "Mon super commit"`

- Git enregistre les modifications des fichiers ayant été « `git add` »
- Commits identifiés par un hash SHA1 :

4961019d387f86449d4d2ca8a9ec37
9ed05006c8



Commit early and often

- Git vous aide seulement si vous committez
-
-
-

Commit early and often



Git vous aide seulement si vous committez

Évite les pertes de données

Commit early and often



Git vous aide seulement si vous committez

Évite les pertes de données

Rend la progression plus lisible

Commit early and often

- Git vous aide seulement si vous committez
- Évite les pertes de données
- Rend la progression plus lisible
- Possibilité de nettoyer son historique de commits avant le push

Commit : encore et toujours

git commit

- Git ouvre alors un fichier de commit avec votre éditeur par défaut (vim)

git commit -a

- git commit tous les fichiers modifiés qu'il connaît

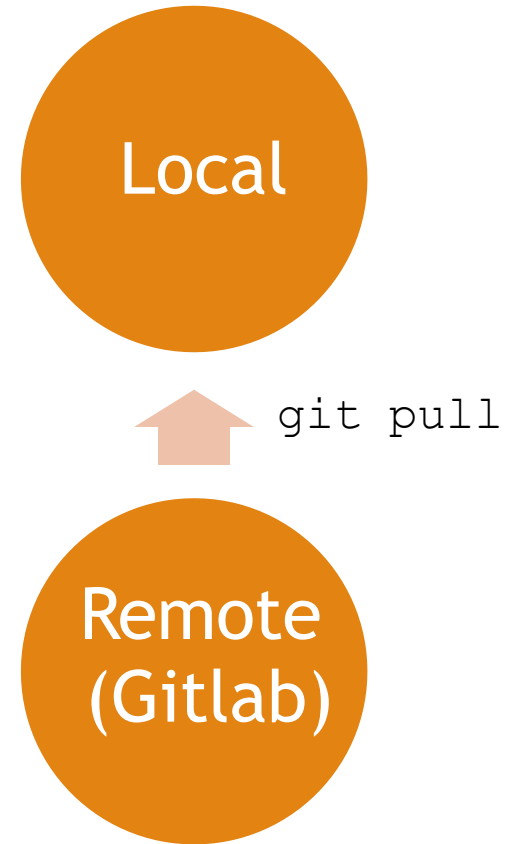
```
Degemer@NUADA ~/nom-du-projet (master)
$ git commit -m "Such upgrade"
[master 40e7ccd] Such upgrade
2 files changed, 0 insertions(+), 0 deletions(-)
rename README => README.md (100%)
create mode 100644 test
```

Pull : récupérer les modifications

`git pull <remote> <branche>`

`git pull origin master` (par défaut)

- Pull permet de récupérer les derniers commits
- **Attention** : risque de conflits avec vos modifications



Un conflit !

- Quand deux personnes modifient le même fichier en même temps, il y a conflit
- Si git ne sait pas le résoudre, il faudra le faire à la main dans les fichiers indiqués

```
Varius@MARLUS-PC ~/nom-du-projet (master)
$ git pull origin master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.my.ecp.fr:2012madecq/nom-du-projet
* branch      master      -> FETCH HEAD
   617cf13..f5fc8cf master  -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Résolution du conflit

- Dans le fichier :
 - Entre <<<<<< HEAD et===== se trouve votre code
 - Entre ===== et>>>>>>>
<hash>celui du dépôt distant
- Choisissez le bon ! (et supprimez les >>>>> et autres)

```
##### HEAD  
Ceci est le README.md du projet, il sera rédigé sous peu.  
Sesepmer je t'aime <3  
-----  
Présentation  
-----  
Ceci est un projet visant à démontrer mon amour envers didip :  
Serendip, je t'aime trop <3  
  
Déploiement  
-----  
Ce projet nécessite juste d'être cloné, et vous aurez alors déclaré votre fla-  
mme.  
  
Auteur  
-----  
Marius Lombard-Platet, aka serendip  
>>>>>> f57c8cf63360f6ec7a619f11c9b135c10b1f3079  
>  
>  
>  
>  
>
```

%\nom-du-projet\README.md [dos] (16:45 16/05/2014)

Résolution du conflit (2)

- git add <fichiers en conflit>
- git commit

```
Marius@MARIUS-PC ~/nom-du-projet (master|MERGING)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
(fix conflicts and run "git commit")

Unmerged paths:
(use "git add <file>..." to mark resolution)

        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

Marius@MARIUS-PC ~/nom-du-projet (master|MERGING)
$ git add README.md

Marius@MARIUS-PC ~/nom-du-projet (master|MERGING)
$ git commit -m 'Fix merge conflict'
[master a7e4bd4] Fix merge conflict
```


Push : envoyer ses commits à Github

git push <remote> <branche>

git push origin master (par défaut)

- Publie vos commits sur le dépôts central
- **Pas de retour en arrière !**
Les commits publiés ne changeront pas !

```
Counting objects: 4, done.  
Delta compression using up to 2 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 279 bytes | 0 bytes/s, done.  
Total 3 (delta 0), reused 1 (delta 0)  
To git@gitlab.my.ecp.fr:2012madecq/nom-du-projet.git  
4961019..40e7ccd master -> master
```

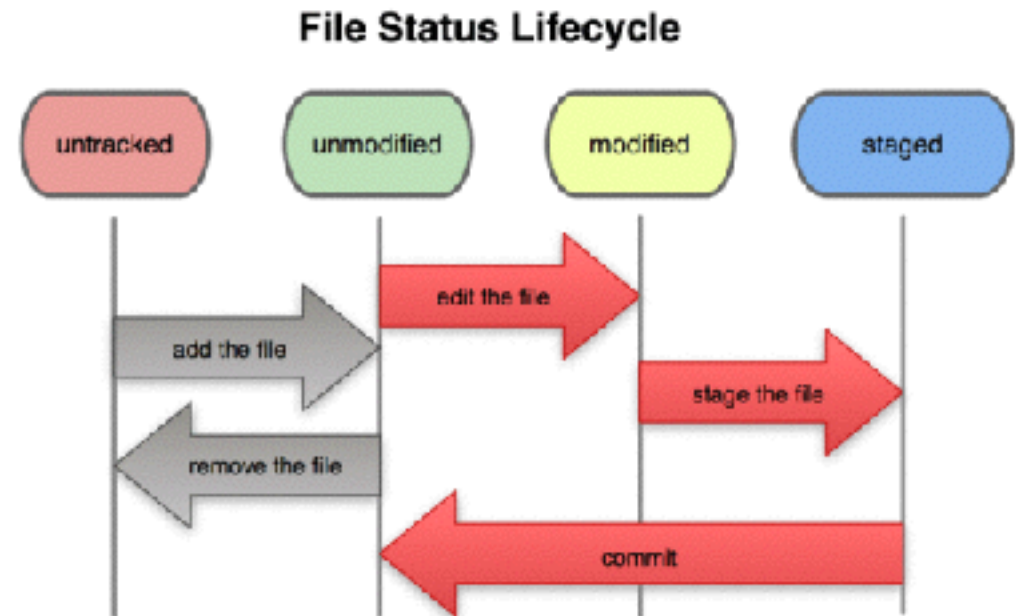
C'est quoi les <remote> ?

- Les <remote> indiquent qu'on s'adresse au dépôt distant (Gitlab)
- C'est un alias pour `git@gitlab.my.ecp.fr:Login/Projet.git`
- Par défaut, cet alias s'appelle `origin`. Là où vous lisez <remote>, vous devrez donc écrire `origin`. 😊

Un peu plus loin...

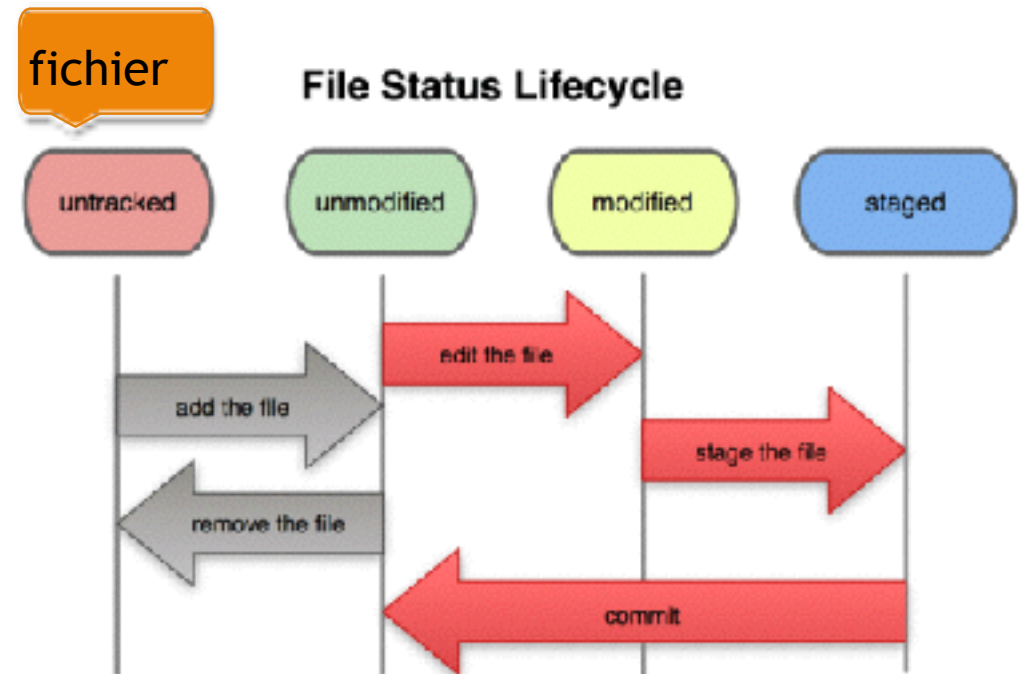
GIT BROTHER IS WATCHING
YOU

Cycle de vie d'un fichier



Cycle de vie d'un fichier

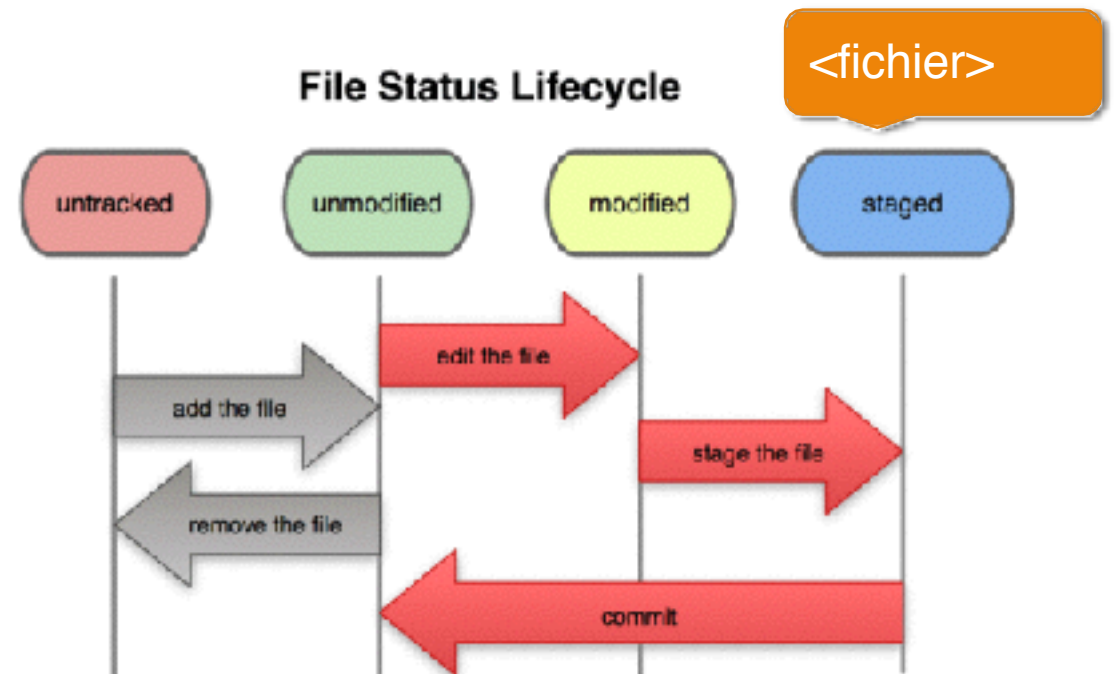
touch <fichier>



Cycle de vie d'un fichier

touch <fichier>

git add <fichier>

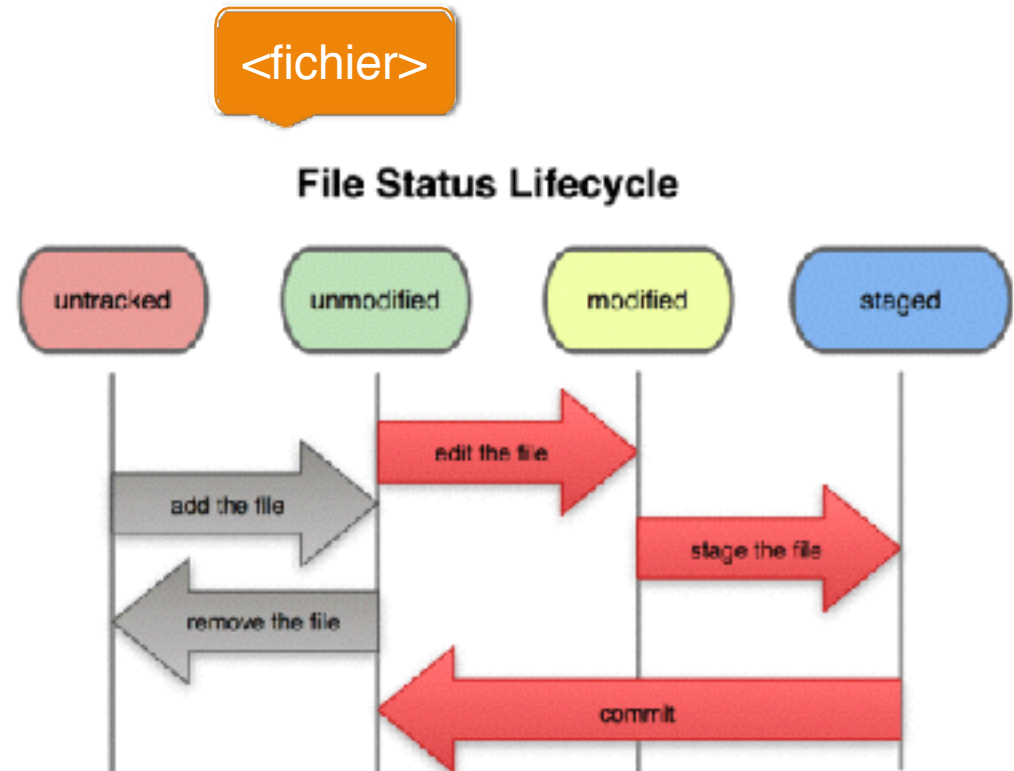


Cycle de vie d'un fichier

`touch <fichier>`

`git add <fichier>`

`git commit -m "Mon super commit"`



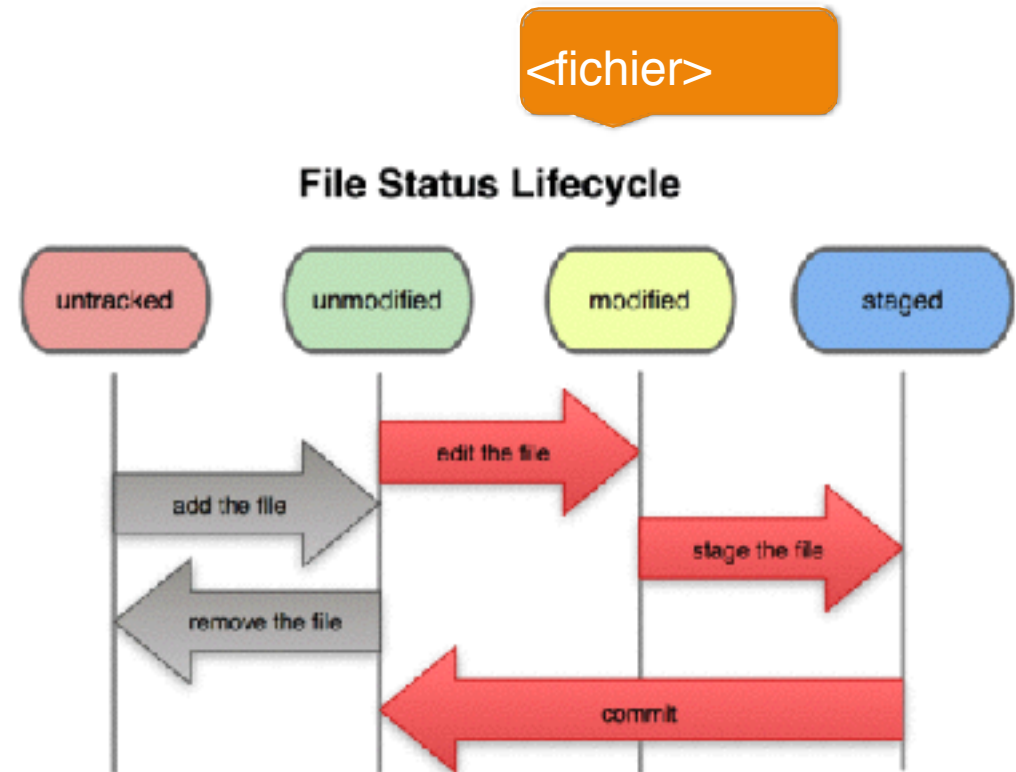
Cycle de vie d'un fichier

`touch <fichier>`

`git add <fichier>`

`git commit -m "Mon super commit"`

`vim <fichier>`



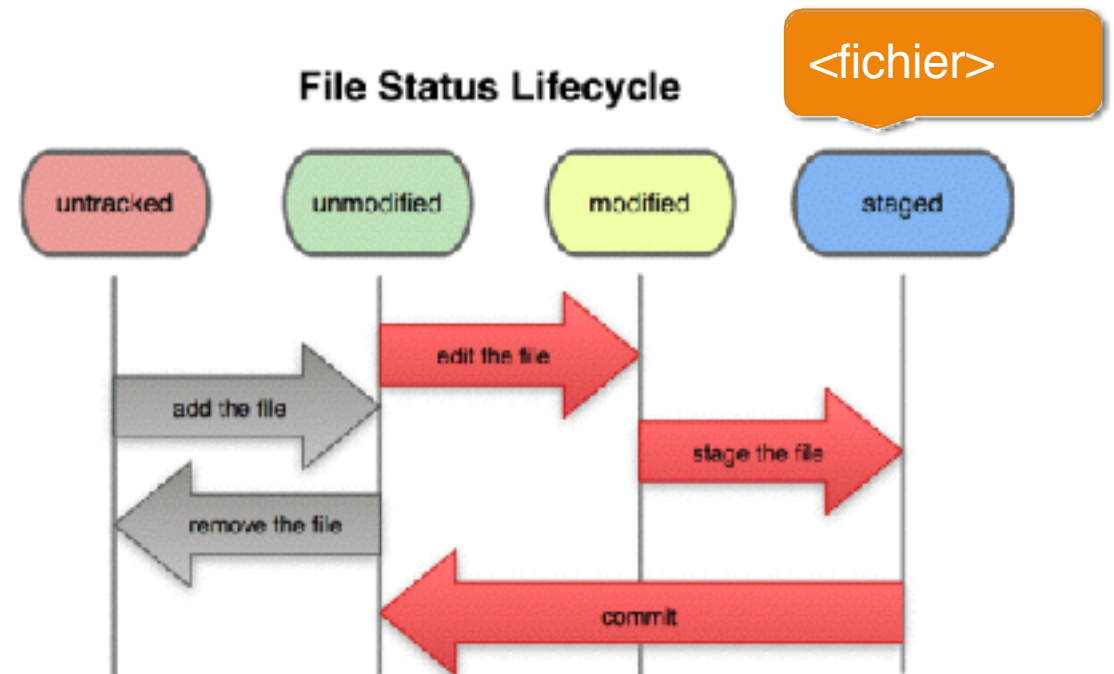
Cycle de vie d'un fichier

`touch <fichier>`

`git add <fichier>`

`git commit -m "Mon
commit"` super

`vim <fichier>` `git add
<fichier>`



Cycle de vie d'un fichier

`touch <fichier>`

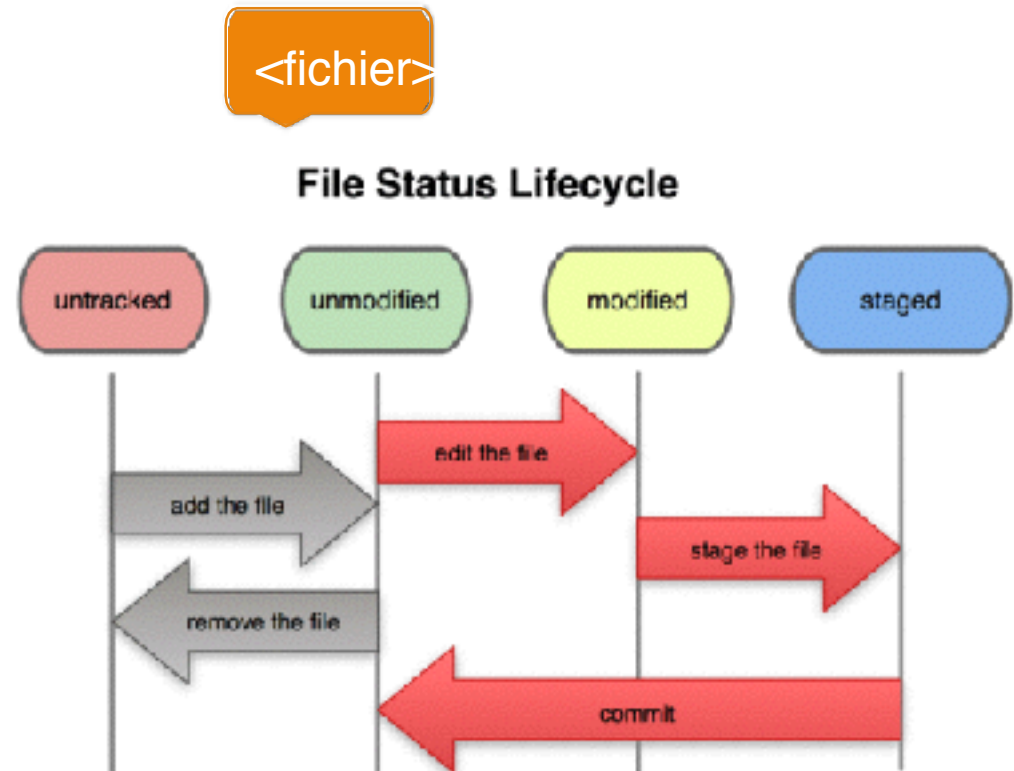
`git add <fichier>`

`git commit -m "Mon
commit"` super

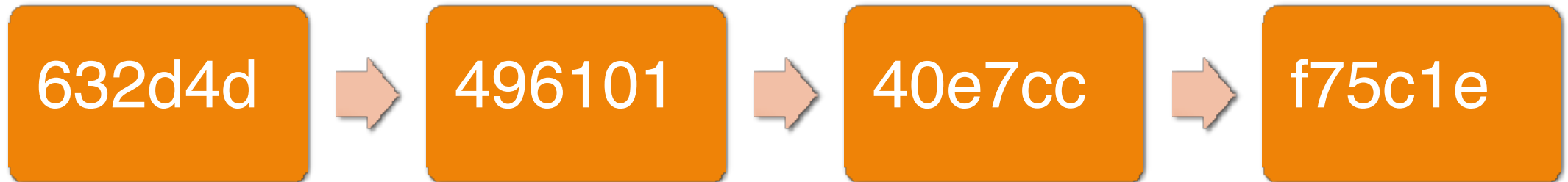
`vim <fichier>` `git add`

`<fichier>`

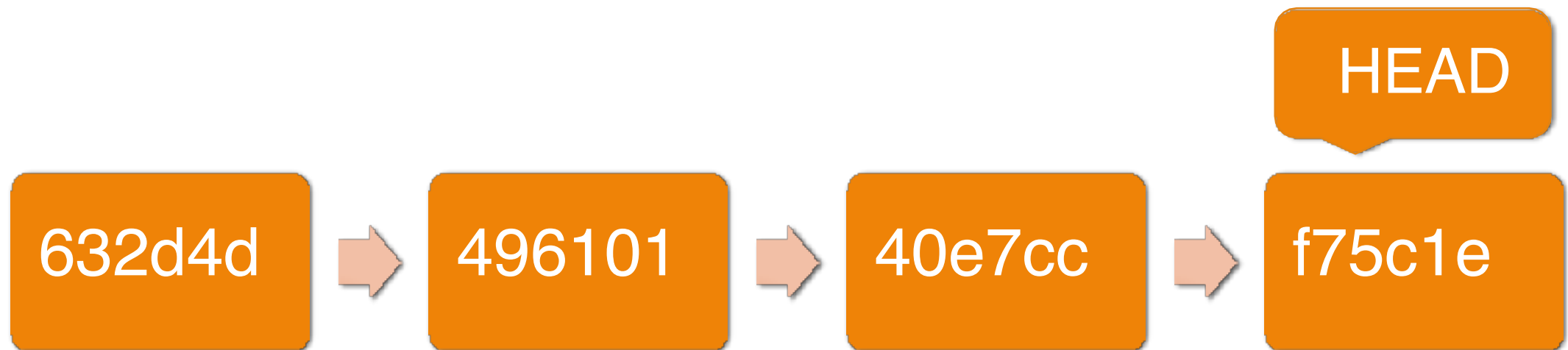
`git commit -m "Such commit"`



HEAD : mais où suis-je ?

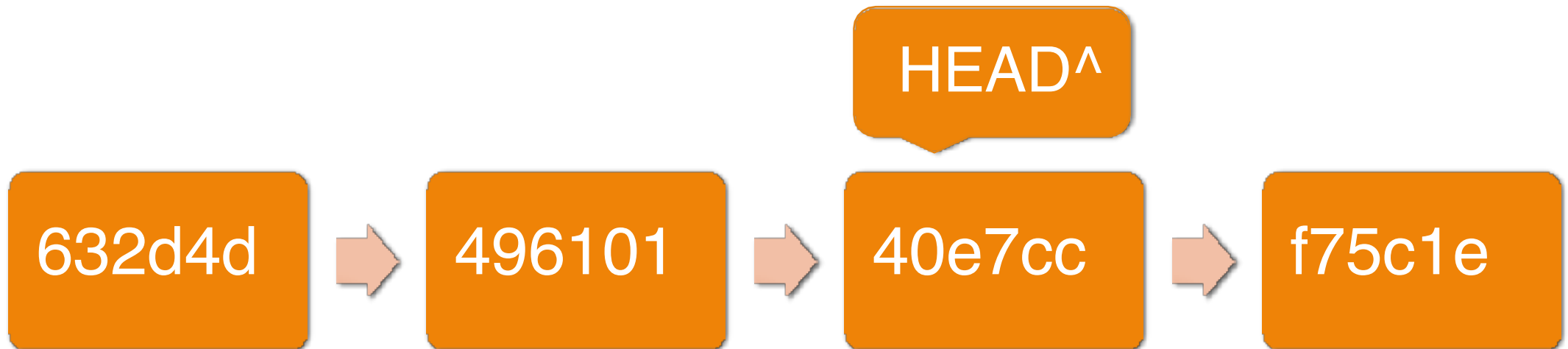


HEAD : mais où suis-je ?



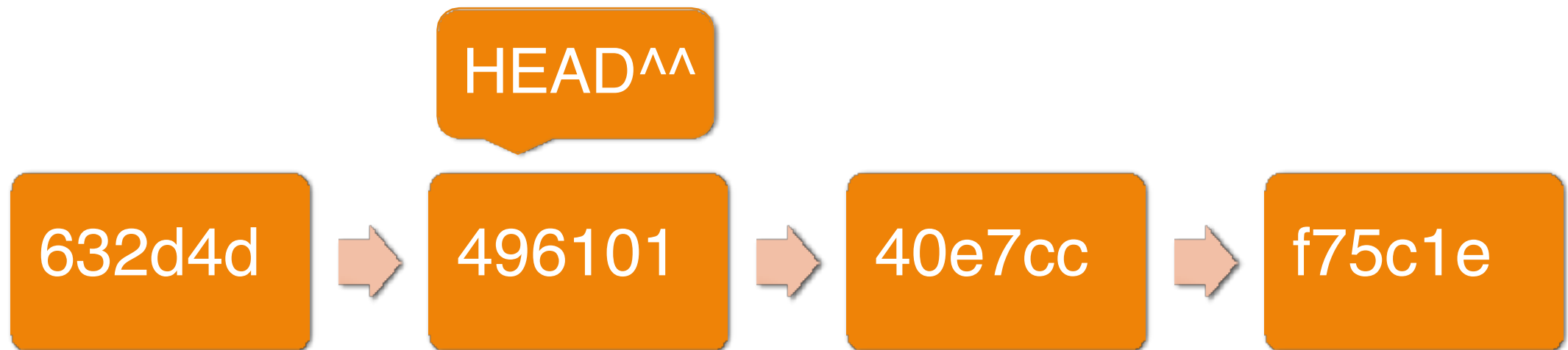
Je me trouve à l'état normal (au niveau du dernier commit)

HEAD : mais où suis-je ?



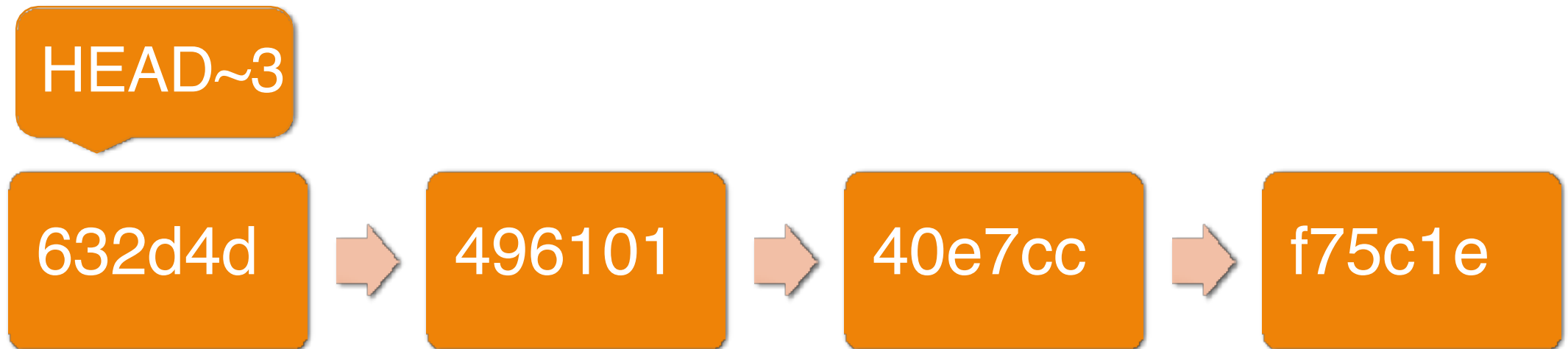
Je me trouve à l'état du commit précédent

HEAD : mais où suis-je ?



Je me trouve à l'état d'il y a 2 commits

HEAD : mais où suis-je ?



Etc. Notez le changement de notation au-delà de HEAD^^

Log : regarder les derniers commits

git log

- Historique de tous les commits

git log --pretty=oneline

- Vue plus rapide des commits

```
Degemer@NUADA ~/nom-du-projet (master)
$ git log
commit 40e7ccdca261f12d82f6b852c2226e2b5128b275
Author: Quentin Madec <quentin.madec@student.ecp.fr>
Date:   Fri May 16 14:14:17 2014 +0200

    Such upgrade

commit 4961019d387f86449d4d2ca8a9ec379ed05006c8
Author: Quentin Madec <quentin.madec@student.ecp.fr>
Date:   Thu May 15 02:14:46 2014 +0200

    Déclaration d'amour

commit 632d4d21870cc1e5de52dee04ae41db3f6a205d8
Author: Quentin Madec <degemer@via.ecp.fr>
Date:   Thu May 15 01:22:19 2014 +0200

    first commit

Degemer@NUADA ~/nom-du-projet (master)
$ git log --pretty=oneline
40e7ccdca261f12d82f6b852c2226e2b5128b275 Such upgrade
4961019d387f86449d4d2ca8a9ec379ed05006c8 Déclaration d'amour
632d4d21870cc1e5de52dee04ae41db3f6a205d8 first commit
```


Diff : comparer deux commits

git diff e78ab21 d454de

- Comparaison de deux commits (identifiés par leur SHA)
- Montre les additions/suppressions
- Mettre juste le début des SHA des commits

```
Degemer@NUADA ~/nom-du-projet (master)
$ git diff 4961 40e/
diff --git a/README b/README
deleted file mode 100644
index 1873b7e..0000000
--- a/README
+++ /dev/null
@@ -1,0 @@
-Serendip, je t'aime <3
diff --git a/README.md b/README.md
new file mode 100644
index 0000000..1873b7e
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
-Serendip, je t'aime <3
diff --git a/test b/test
new file mode 100644
index 0000000..e69de79
```

Rm/mv : déplacer/supprimer des fichiers

git rm <fichier>

- Supprime le fichier et l'untrack

git mv <ancien fichier>
<nouveau fichier>

- Git renomme le fichier (le changement apparaîtra dans le prochain commit)

```
Degemer@NUADA ~/nom-du-projet (master)
$ git rm test
rm 'test'

Degemer@NUADA ~/nom-du-projet (master)
$ ls
README.md

Degemer@NUADA ~/nom-du-projet (master)
$ git mv README.md README

Degemer@NUADA ~/nom du projet (master)
$ ls
README
```

Annuler un git add

git reset HEAD <fichier>

- Annule un git add fait précédemment
- **Attention** : annule les modifications du fichier
git reset HEAD --hard <fichier>

```
degemer@NUADA ~/nom-du-projet (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <files>..." to unstage)

        modified:   README.md

degemer@NUADA ~/nom-du-projet (master)
$ git reset HEAD README.md
Unstaged changes after reset:
M       README.md

degemer@NUADA ~/nom du projet (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

.gitignore : données sensibles ou inutiles

https://github.com/epoch/contact_form_app/blob/33f53bc1a1df88565b46e0d37d55194e05c5_be0c/.env

- Créer/modifier le .gitignore à la racine du projet
 - lib.so : le fichier lib.so sera ignoré
 - *.so : tous les fichiers en .so seront ignorés
 - conf/ : tous les fichiers du dossier conf seront ignorés
 - conf/**/*.yml : tous les fichiers .yml de tous les sous-dossiers de conf/ seront ignorés

Toujours plus loin...

GIT, ÇA VOUS
BRANCHE ?

Git et les branches : une histoire d'amour

- Les branches, de kossé ?
 - Mieux ordonner son espace de développement
 - Travailler sur plusieurs bouts de code en même temps, sans interférences

Git et les branches : une histoire d'amour

- Les branches, de kossé ?
 - Mieux ordonner son espace de développement
 - Travailler sur plusieurs bouts de code en même temps, sans interférences
- Exemple :
 - Une branche de développement (dev), une branche de production (master)
 - Plusieurs branches de développement ! Une par fonctionnalité

Exemple de branches

dev

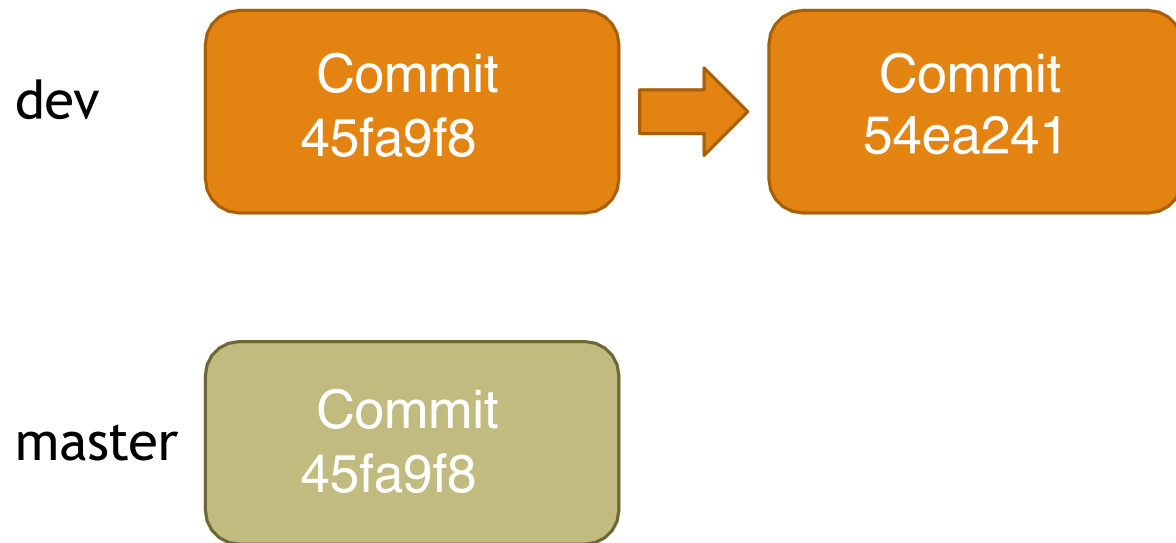
Commit
45fa9f8

master

Commit
45fa9f8

État initial : dev
et master sont
dans le même
état

Exemple de branches

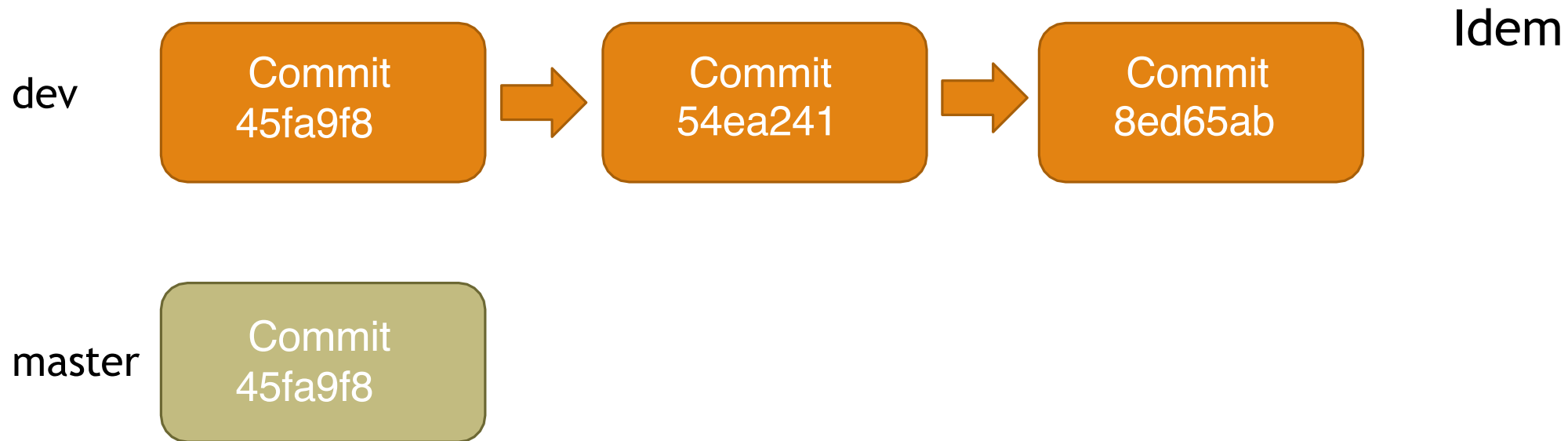


Je fais un commit sur

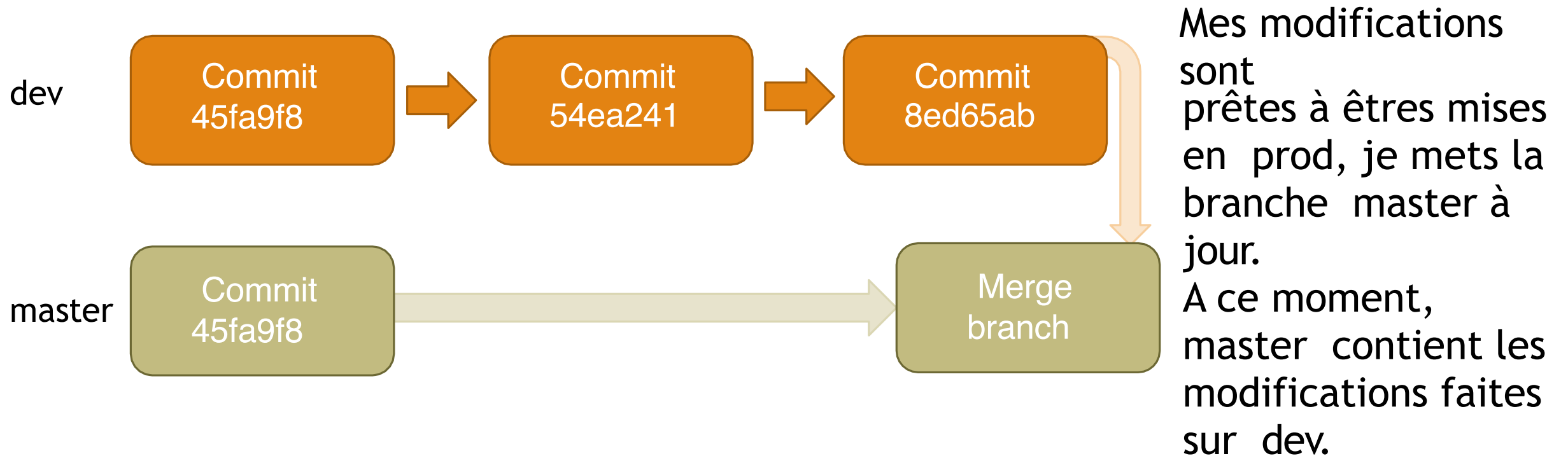
dev, je push pour que les autres puissent y avoir accès.

Notez que master ne voit aucun changement.

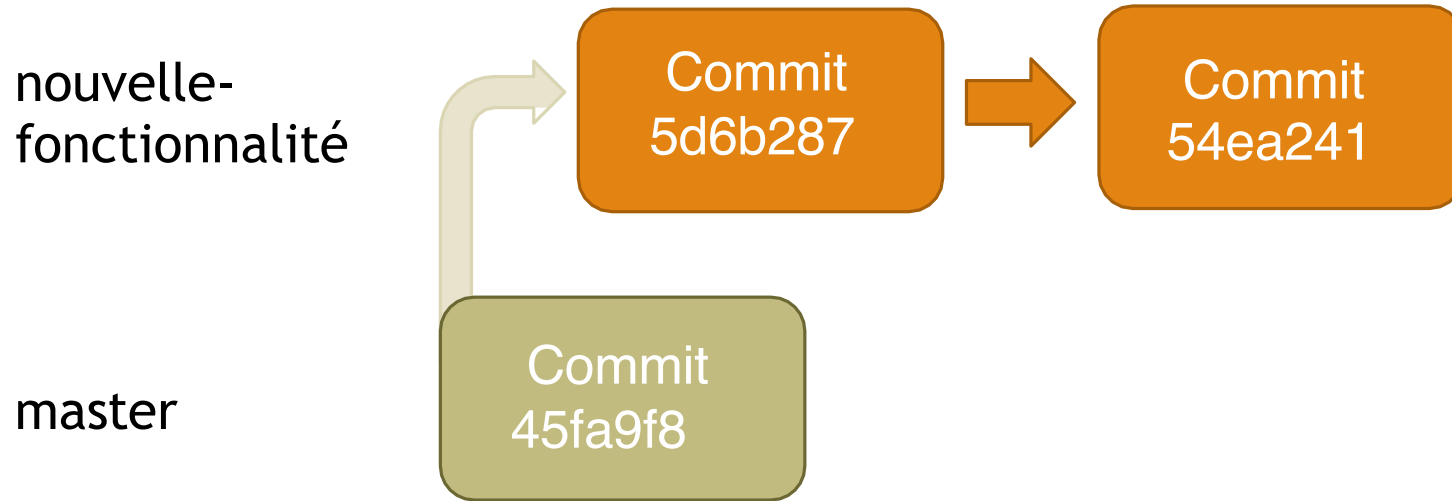
Exemple de branches



Exemple de branches

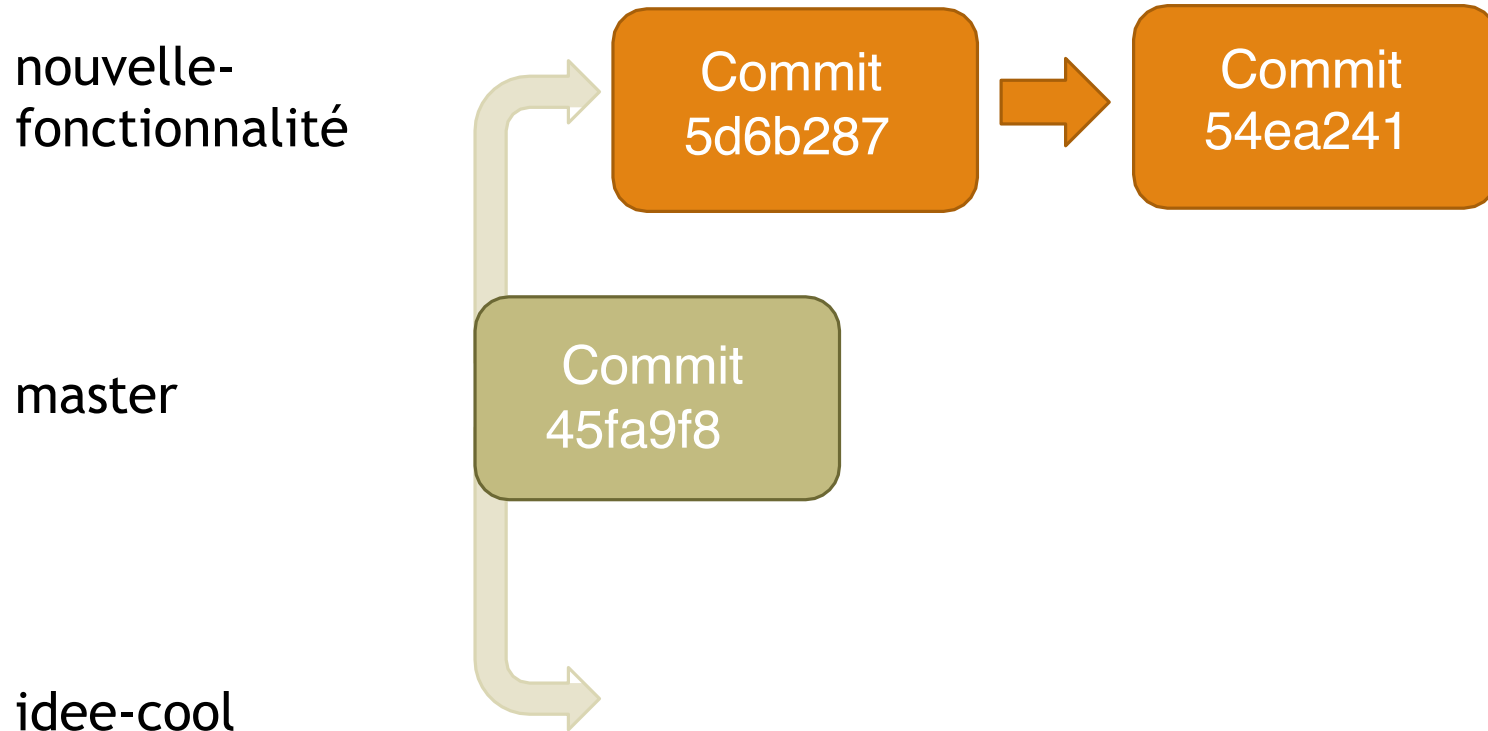


Exemple de branches



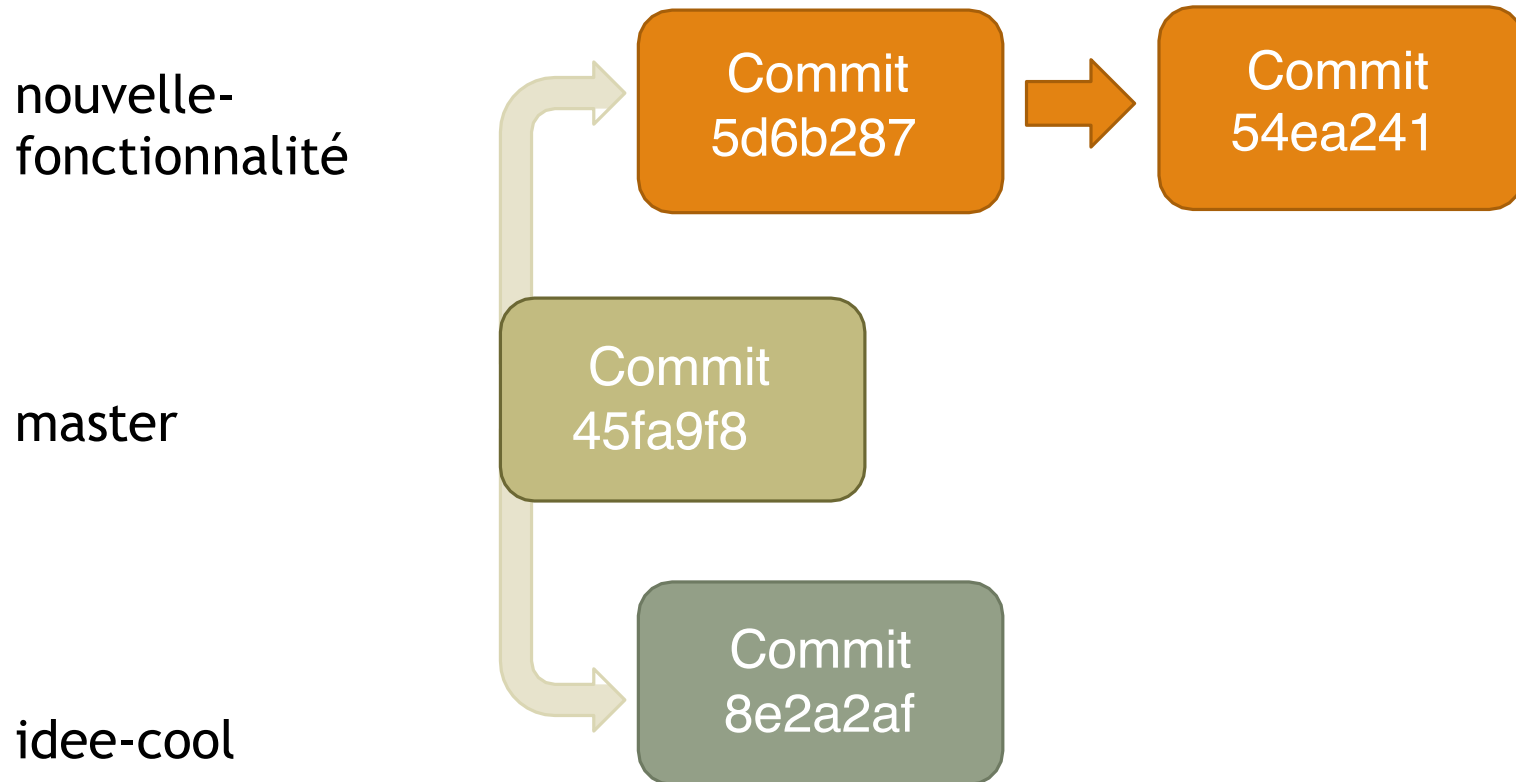
Je suis en train **développer** une nouvelle fonctionnalité, je suis sur la branche nouvelle- fonctionnalité. Mon code n'est pas encore destiné à être mis en prod.

Exemple de branches



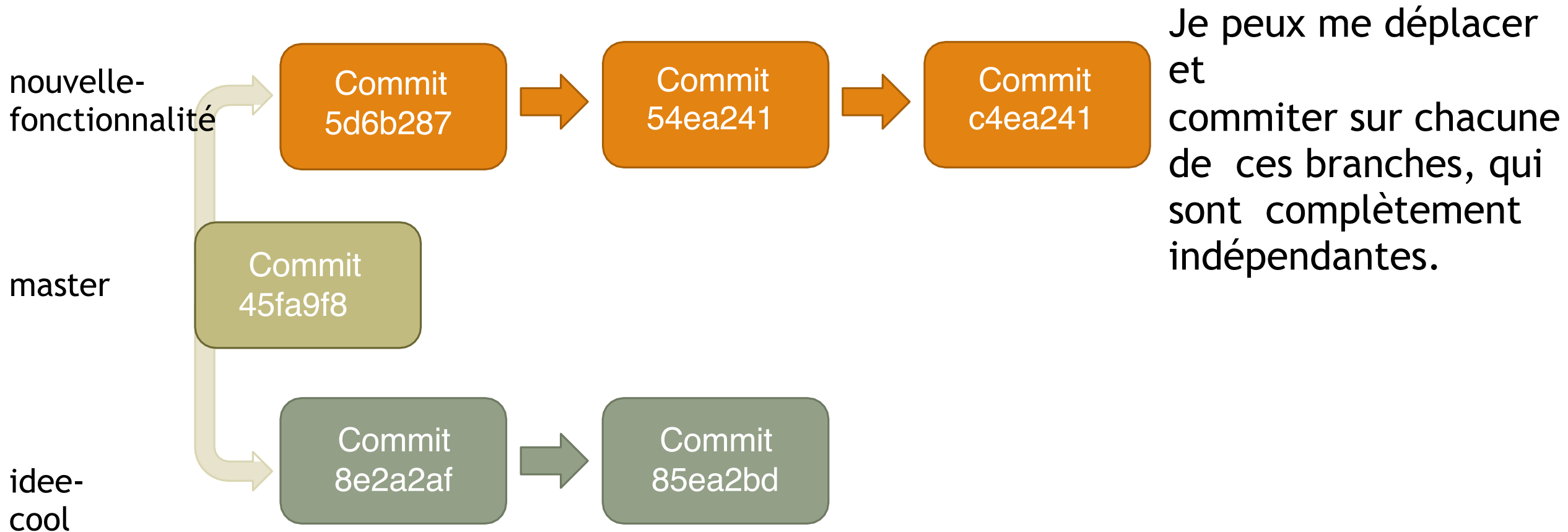
Quand soudain, me vient une idée super cool. Ni une ni deux, je crée une nouvelle branche à partir de master.

Exemple de branches

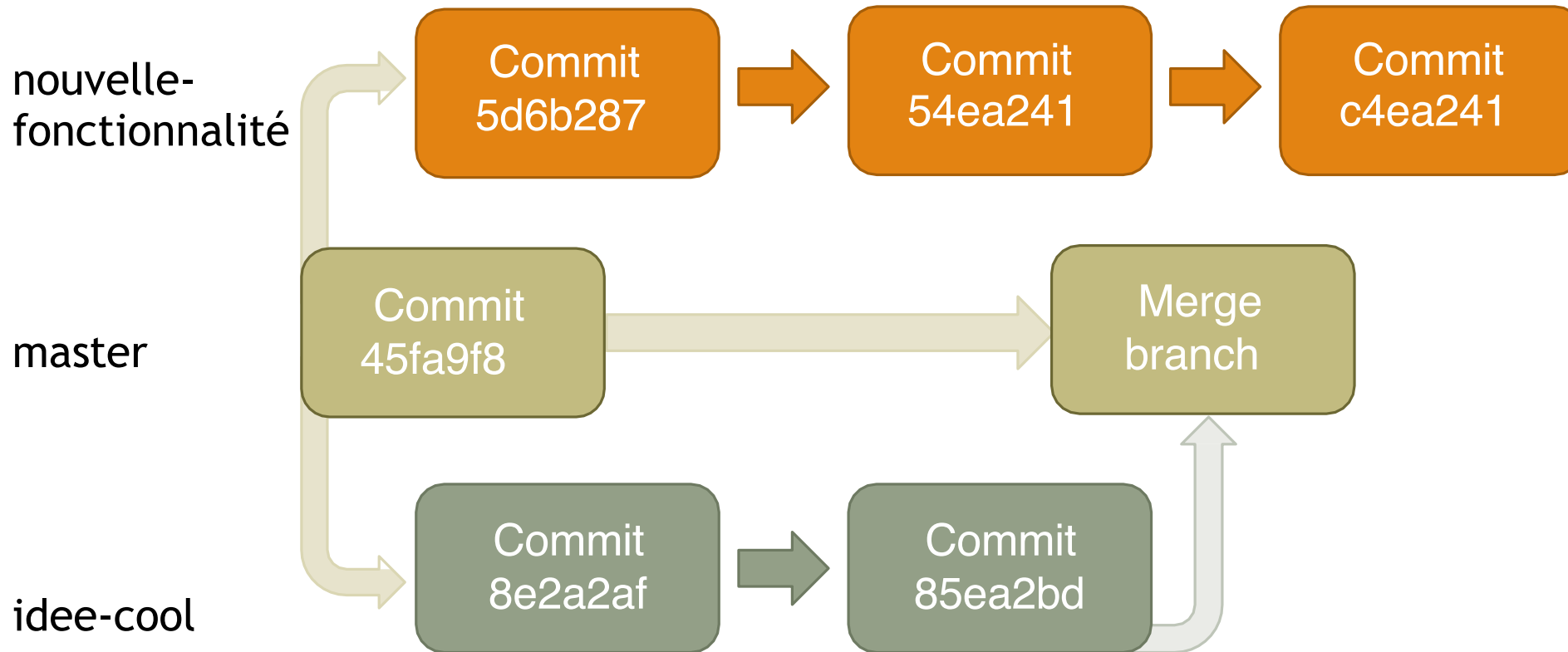


Je commence
à
développer dessus,
sans me soucier des
modifications faites
sur
nouvelle-
fonctionnalité.

Exemple de branches



Exemple de branches



Tout se passe alors comme dans le cas précédent : quand une branche est prête, je la passe sur master. Ici, master contient les modifications faites sur idee-cool mais pas nouvelle-fonctionnalite.

Les branches, en vrai

- Pour ceux qui ont trouvé cette explication un peu floue, sachez qu'en réalité, les branches sont des pointeurs vers un *commit*
- Pour en savoir plus : <http://git-scm.com/book/fr/Les-branches-avec-Git-Ce-qu-est-une-branche>
- Mais ce n'est pas primordial. 😊

Naviguer de branche en branche

- Voir les branches :

git branch

```
Marius@MARIUS-PC ~/myecp (master)
$ git branch
hotfix
logs
* master
piaulage
revisions-sociales
```

- Voir aussi les branches distantes :

git branch -a

```
$ git branch -a
hotfix
logs
* master
piaulage
revisions-sociales
remotes/origin/HEAD -> origin/master
remotes/origin/dev
remotes/origin/fusion-matieres
remotes/origin/hotfix
remotes/origin/hotfix-1.11
remotes/origin/hotfix-1.12
remotes/origin/infinite-scroll
remotes/origin/infos-pratiques
remotes/origin/logs
```

Naviguer de branche en branche

- Changer de branche :
 - Assurez-vous de tout avoir bien commité
 - `git checkout <branche>`
- Vous pouvez laisser des fichiers non commités... Mais c'est déconseillé

```
Marius@MARIUS-PC ~/myecp (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

Marius@MARIUS-PC ~/myecp (master)
$ git checkout logs
Switched to branch 'logs'
Your branch is up-to-date with 'origin/logs'.
```

Créer une nouvelle branche

- ... à partir de la branche courante :

- `git checkout -b <branche>`

```
Marius@MARIUS-PC ~/myecp (logs)
$ git checkout -b TESTEST
Switched to a new branch 'TESTEST'
```

- ... d'une branche distante (mieux) :

- `git checkout -b <branche> origin/<branche distante>`

```
$ git checkout -b update-symfony origin/update-symfony
Branch update-symfony set up to track remote branch update-symfony
Switched to a new branch 'update-symfony'
```

Supprimer des branches

- `git branch -d <branche>`
- C'est irréversible !
- Forcer la suppression :
 - Quand des commits seront perdus
- `git branch -D <branche>`

```
Marius@MARIUS-PC ~/myecp (TESTEST)
$ git checkout master
Checking out files: 100% (195/195), done.
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Marius@MARIUS-PC ~/myecp (master)
$ git branch -d TESTEST
Deleted branch TESTEST (was f456c00).
```

```
Marius@MARIUS-PC ~/myecp (AUTRETEST)
$ git commit
[AUTRETEST b10eb0f] test
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test

Marius@MARIUS-PC ~/myecp (AUTRETEST)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Marius@MARIUS-PC ~/myecp (master)
$ git branch -d AUTRETEST
error: The branch 'AUTRETEST' is not fully merged.
If you are sure you want to delete it, run 'git branch -D AUTRETEST'.

Marius@MARIUS-PC ~/myecp (master)
$ git branch -D AUTRETEST
Deleted branch AUTRETEST (was b10eb0f).
```

Synchroniser sa branche

- Se mettre à jour avec la branche distante :
 - `git pull <remote> <branche distante>`
- Mettre à jour la branche distante :
 - `git push <remote> <branche distante>`

```
Marius@MARIUS-PC ~/nom-du-projet (master)
$ git pull origin master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.my.ecp.fr:2012madecq/nom-du-projet
 * branch          master       -> FETCH_HEAD
   fb1b6eb..617cf13 master     -> origin/master
Updating fb1b6eb..617cf13
Fast-forward
 README | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

```
Marius@MARIUS-PC ~/nom-du-projet (nouvelle-branche)
$ git push origin nouvelle-branche
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 280 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@gitlab.my.ecp.fr:2012madecq/nom-du-projet.git
 * [new branch]      nouvelle-branch -> nouvelle-branch
```

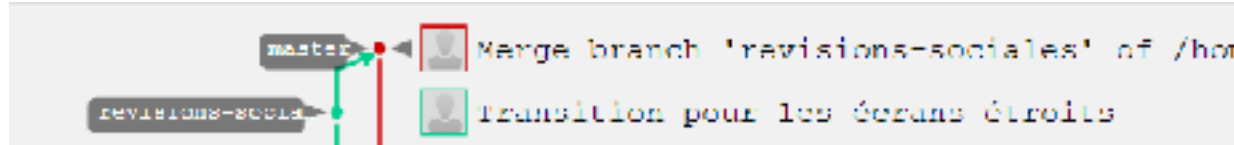
Un conflit ! (*bis*)

- Que faire ?
 - Lire le message d'erreur. 😊
- Conflit lorsque des fichiers n'ont pas été commités
- Solutions :
 - Commiter les fichiers en question ;
 - Ou annuler leur modifications (git checkout <fichier>) ;
 - Ou *stash* (utilisation plus avancée).

```
$ git pull origin master
From gitlab.my.ecp.fr:2012madecq/nom-du-projet
* branch          master      -> FETCH_HEAD
Updating G32d4d2..4961019
error: Your local changes to the following files would be overwritten by merge:
      README
Please, commit your changes or stash them before you can merge.
Aborting
```

Merger des branches

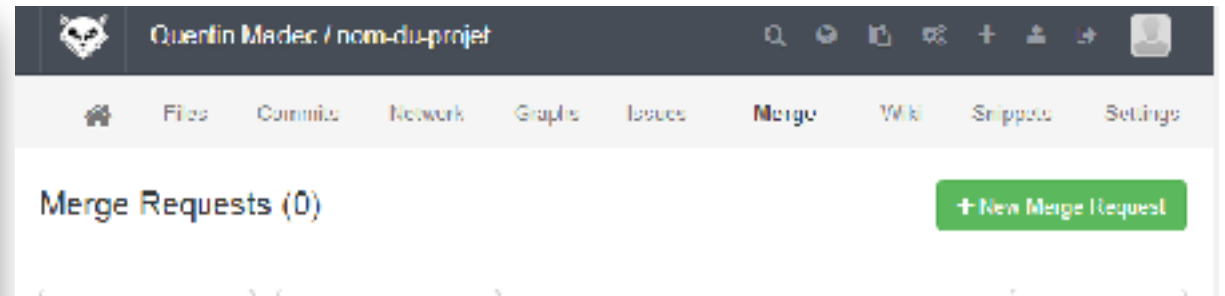
Comment appliquer les commits de ma branche de dev à ma branche de prod ?



- En ligne de commande :
 - git checkout master
 - git merge dev
- Autre solution : utiliser github et les *pull request*.

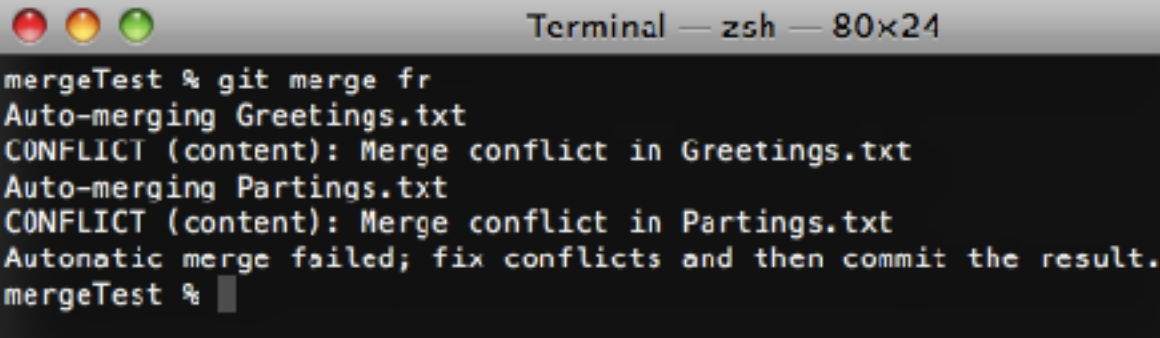
```
Marius@MARIUS-PC ~/nom du projet (nouvelle branche)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Marius@MARIUS-PC ~/nom-du-projet (master)
$ git merge nouvelle-branche
Updating 617cf13..f50Ged1
Fast-forward
 test | 1 +
 1 file changed, 1 insertion(+)
```



Un conflit ! (*ter*)

- Le git merge ne se fait pas lorsqu'un fichier est sur les deux branches

A terminal window titled "Terminal — zsh — 80x24" with three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of a git merge command. It shows "Auto-merging Greetings.txt" followed by "CONFLICT (content): Merge conflict in Greetings.txt", then "Auto-merging Partings.txt" followed by "CONFLICT (content): Merge conflict in Partings.txt". A message states "Automatic merge failed; fix conflicts and then commit the result." The prompt "mergeTest %" is visible at the bottom with a cursor.

```
mergeTest % git merge fr
Auto-merging Greetings.txt
CONFLICT (content): Merge conflict in Greetings.txt
Auto-merging Partings.txt
CONFLICT (content): Merge conflict in Partings.txt
Automatic merge failed; fix conflicts and then commit the result.
mergeTest %
```

- Il va donc falloir régler le conflit comme lors d'un git pull (ne pas oublier de commiter une fois le conflit résolu !)

git commit -m "C'est fini"

- Des questions ?

Annexes

CI-GÎT CE QUI N'A PAS PU ÊTRE PLACÉ DANS LA
FORMATION



Modifier son dernier commit

- Pour rajouter des fichiers à son dernier commit :
 - `git add <fichier>`
 - `git commit --amend -m "Message de commit"`
- **Attention !** Ne pas le faire si le commit en question a déjà été pushé. Cela pourrait créer de gros conflits d'historique.

Git stash

- Permet de sauvegarder temporairement un travail (pour passer dans une autre branche par exemple)
 - `git stash` : sauvegarde et remet la branche dans l'état du dernier commit
 - `git stash apply` : applique le dernier stash (sans le supprimer de la pile des stash)
 - `git stash pop` : applique le dernier stash (et le supprime de la pile)
 - `git stash clear` : vide la pile des stash
 - ... (regarder la doc)

Résoudre les conflits proactivement©

- `git checkout --theirs <fichier>` pour garder le fichier distant (celui qu'on pull ou qu'on merge)
- `git checkout --ours <fichier>` pour l'inverse
- Bien sûr, ne pas oublier de commiter à la fin.

RTFM !

- Plein de documentations un peu partout :
 - <http://git-scm.com/book/fr> la doc officielle, et en français ;
 - <http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/fr/index.html> un autre tuto, très complet lui aussi, en français également ;
 - <http://sethrobertson.github.io/GitBestPractices/> comment avoir un *workflow* efficace ;
- En encore plein d'autres sur Google !