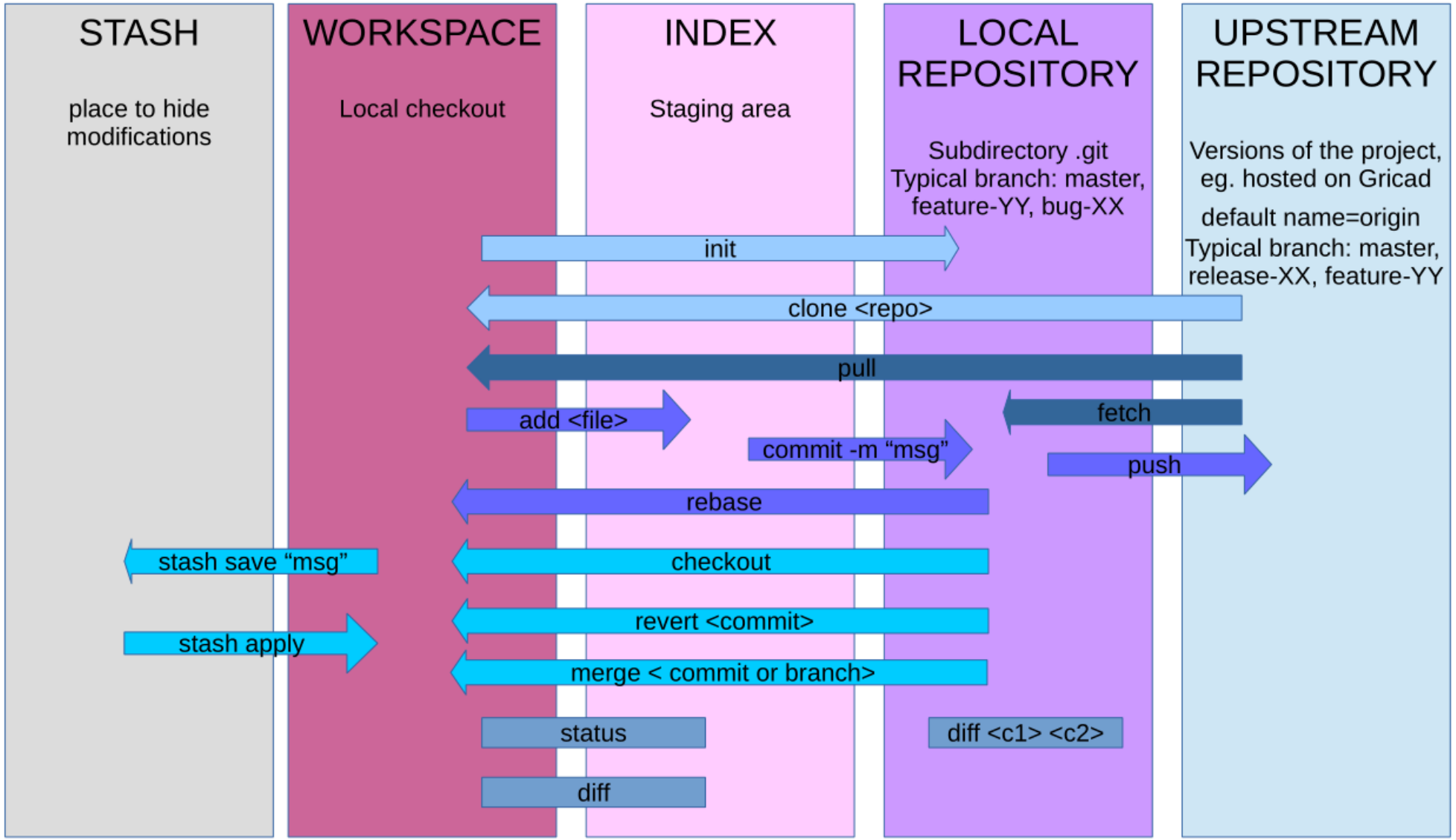
# Prise en main de GIT/GITLab

David HARBINE

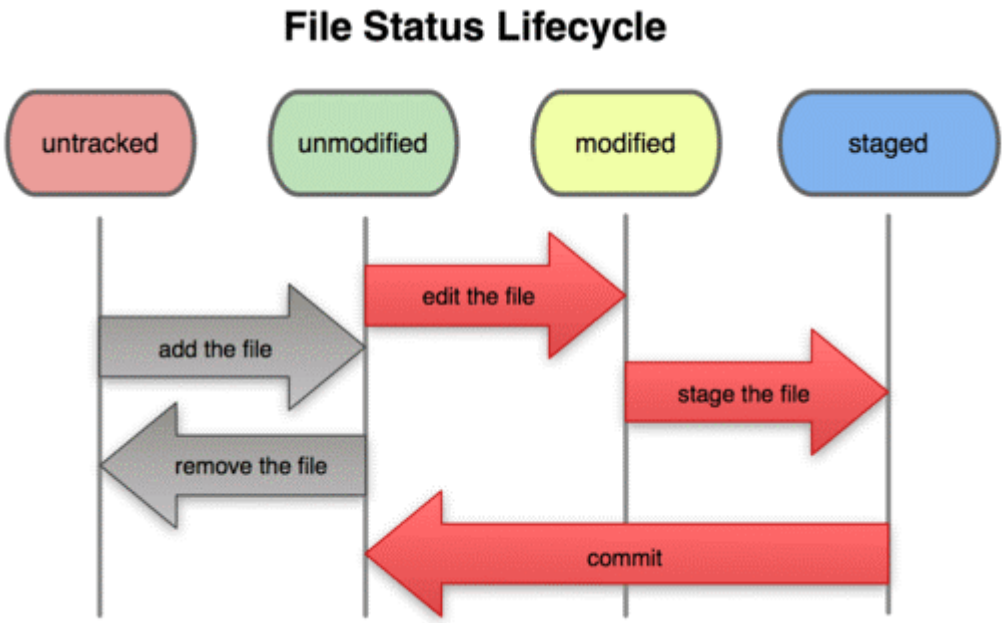
Projet MIA2

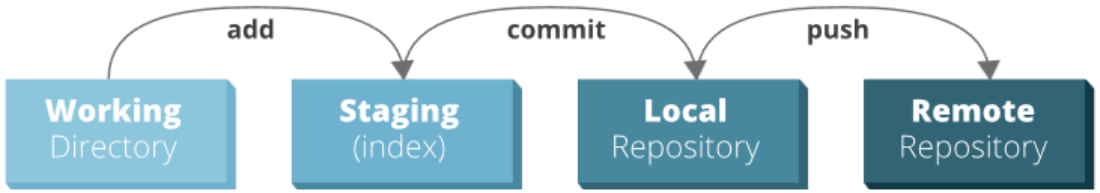
12/12/2017

# Git concepts



# L’état des fichiers



# L’indexation

**<u>Définition :</u> marquer un fichier modifié dans sa version actuelle pour qu’il fasse partie du prochain instantané du projet**

**<u>Quelques commandes utiles :</u>**

- **$ git add *<fichier(s) ou dossier(s)>***
  Indexer des fichiers ou des dossiers
- **$ git status**
  Connaître l’état du projet et des fichiers
- **$ git diff**
  Voir les différences entre les fichiers modifiés et indexés
- **$ git rm *<fichier(s)>***
  Supprimer des fichiers de l’espace de travail et de l’index
- **$ git mv *<fichier(s)>***
  Déplacer des fichiers de l’espace de travail et de l’index

# L’indexation : exemple

```
harbined@harbined-VirtualBox:~/mon_projet$ git status
Sur la branche branch1
rien à valider, la copie de travail est propre
harbined@harbined-VirtualBox:~/mon_projet$ vim fichier1
harbined@harbined-VirtualBox:~/mon_projet$ git add fichier1
harbined@harbined-VirtualBox:~/mon_projet$ git status
Sur la branche branch1
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        modifié :         fichier1

harbined@harbined-VirtualBox:~/mon_projet$ vim fichier1
harbined@harbined-VirtualBox:~/mon_projet$ git status
Sur la branche branch1
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        modifié :         fichier1

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

        modifié :         fichier1
```

# L’indexation : exemple

```
harbined@harbined-VirtualBox:~/mon_projet$ git diff
diff --git a/Capture du 2017-12-06 11-12-48.png b/Capture du 2017-12-06 11-12-48.png
deleted file mode 100644
index 6ab0175..0000000
Binary files a/Capture du 2017-12-06 11-12-48.png and /dev/null differ
diff --git a/fichier1 b/fichier1
index 30e0800..ce12e49 100644
--- a/fichier1
+++ b/fichier1
@@ -1 +1 @@
-Ceci est un fichier test appelé fichier1.
+Ceci est un fichier test appelé fichier1 !
```

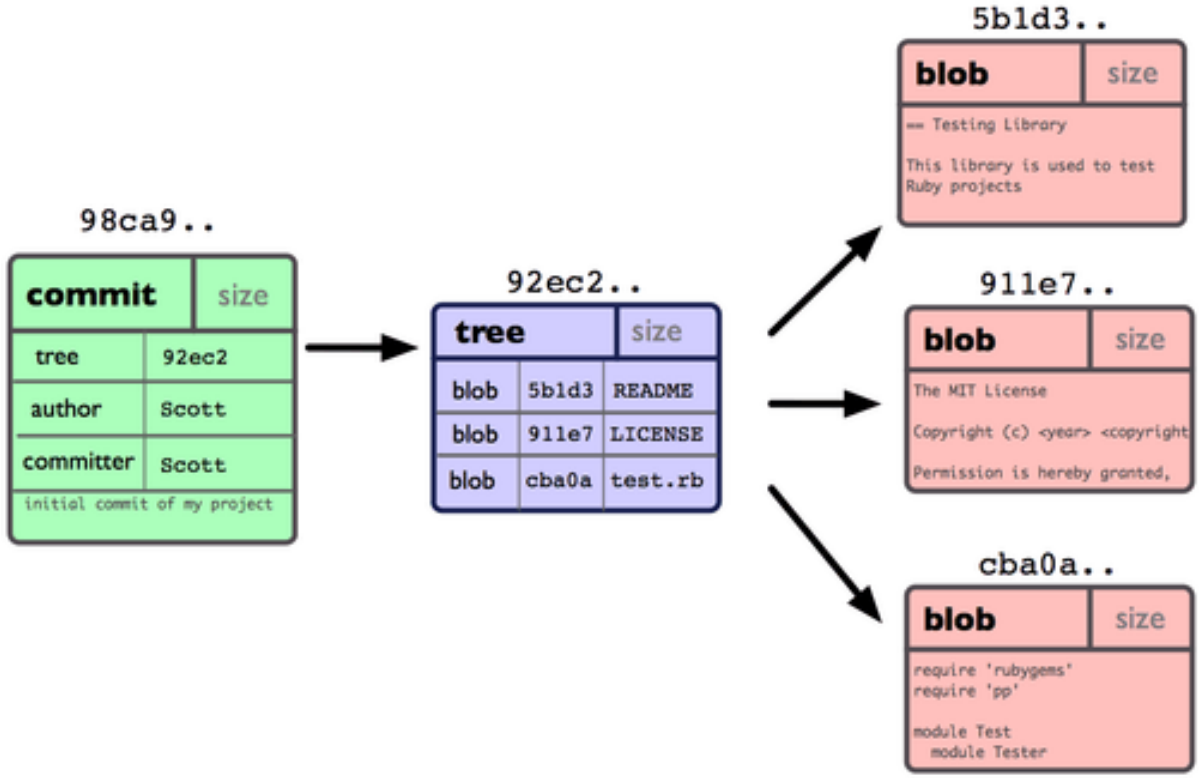
# La validation

**<u>Définition :</u> stocker les données en sécurité dans le dépôt local**

**<u>Quelques commandes utiles :</u>**

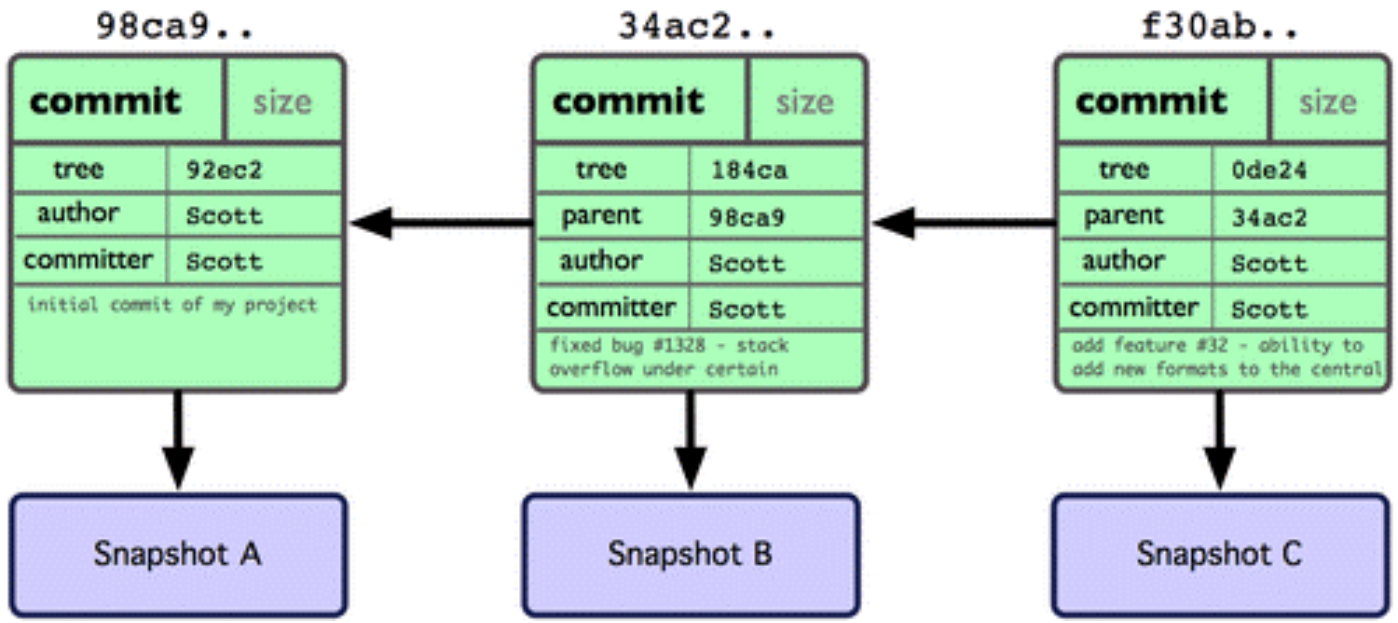
- **$ git commit –m « message »**
  Enregistrer le contenu de l'index dans un nouveau commit en y associant un message utilisateur décrivant les modifications effectuées
- **$ git commit –a –m « message »**
  Effectuer un commit de tous les fichiers qui ont changé depuis le dernier commit, à l'exception des fichiers non-suivis.

# La validation : l'objet commit




*Données d'un commit unique*

# La validation : l'objet commit




*Données et objets Git pour des validations multiples*

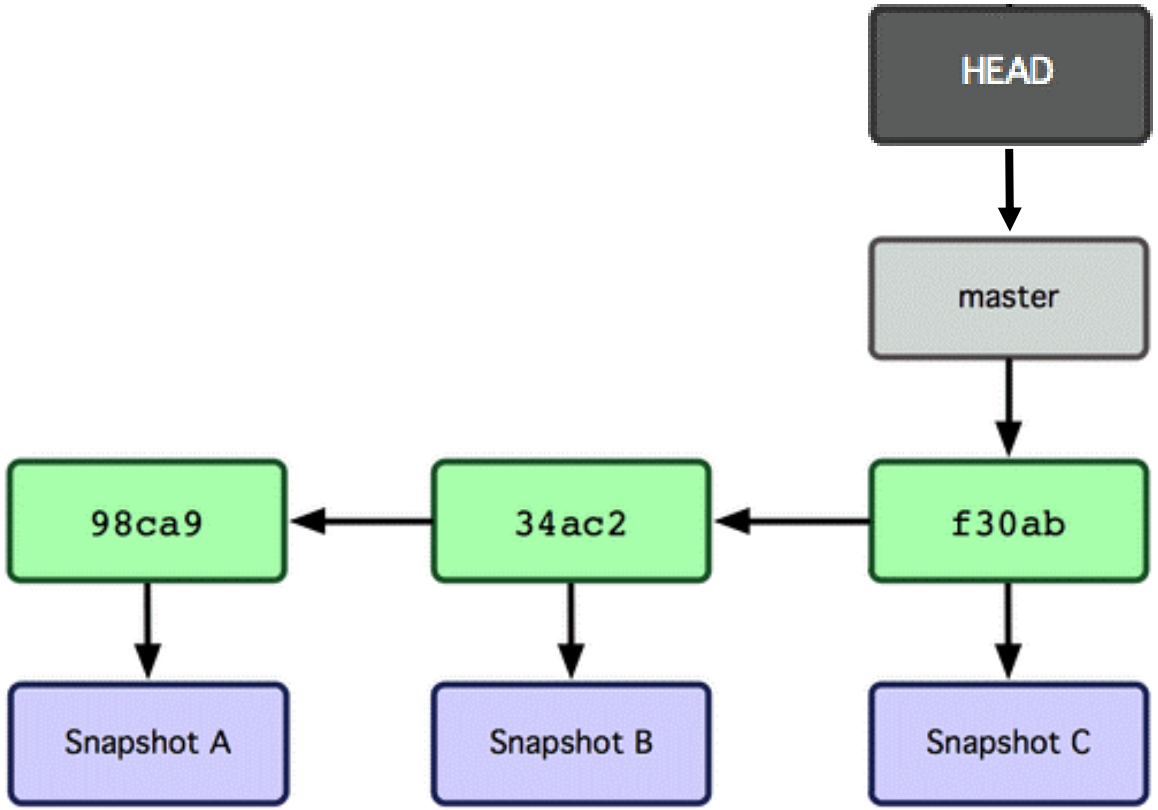
# Les branches

**<u>Définition :</u> pointeur mobile léger vers un objet commit**

**<u>Quelques commandes utiles :</u>**

- **$ git checkout –b *<branche>***
  Créer une nouvelle branche *<branche>* et se positionner dessus
- **$ git checkout *<branche>***
  Se placer sur la branche *<branche>*
- **$ git branch**
  Lister les branches locales existantes
- **$ git branch –d *<branche>***
  Supprimer la branche *<branche>*
- **$ git merge *<commit ou branche>***
  Fusionner les modifications du *<commit>* ou de la *<branche>* dans la branche courante.

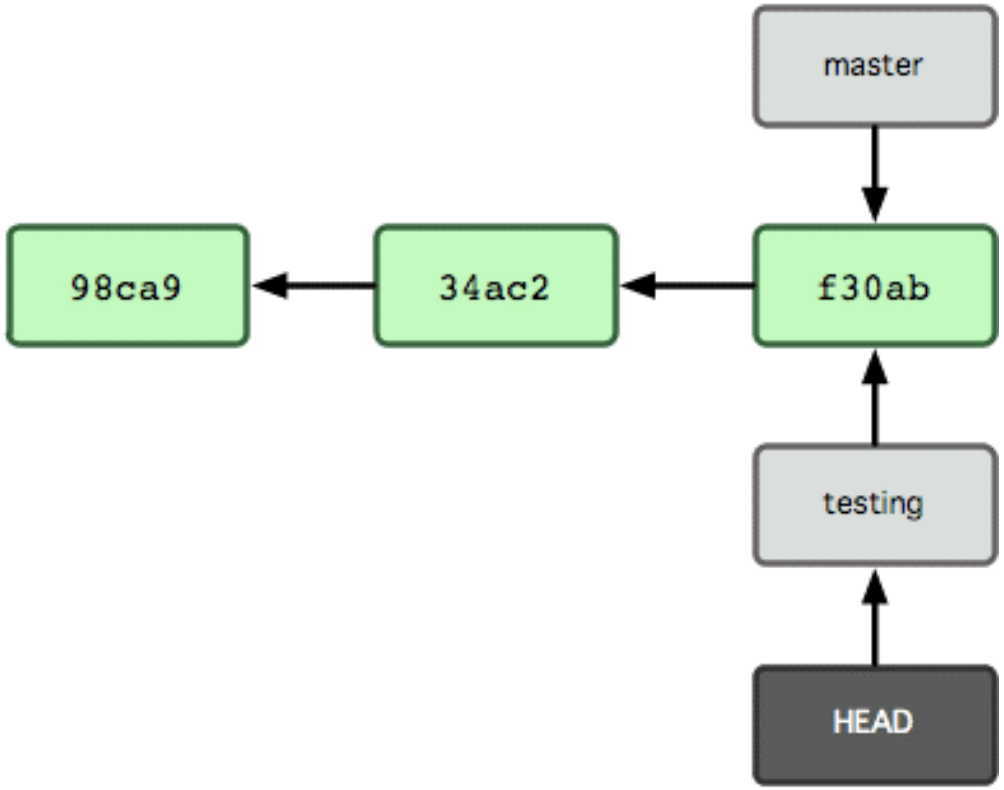
# Les branches : illustration




*Branche pointant dans l'historique de données de commit et fichier HEAD pointant sur la branche active*

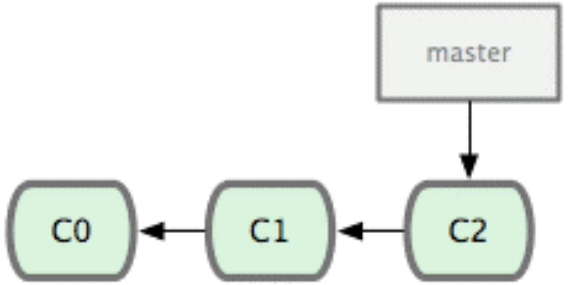
# Les branches : illustration

$ git checkout -b testing




*Création d'une nouvelle branche avec git checkout –b : basculement sur la nouvelle branche*

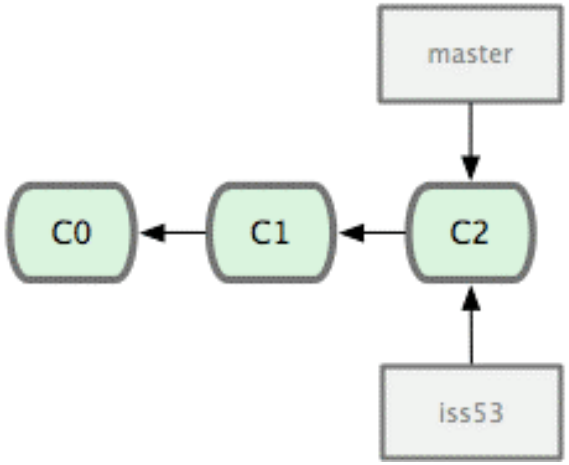
# Les branches : exemple




*Branche master*

```
harbined@harbined-VirtualBox:~/mon_projet$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
rien à valider, la copie de travail est propre
```

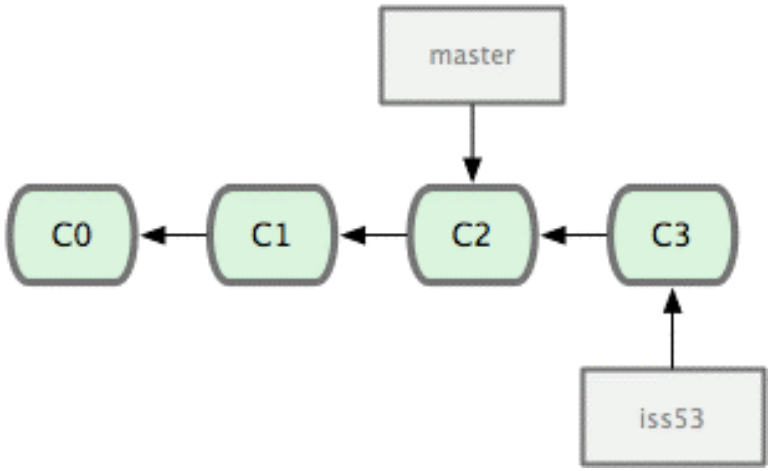
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ git checkout -b iss53
Basculement sur la nouvelle branche 'iss53'
```

*Création d'une nouvelle branche iss53*

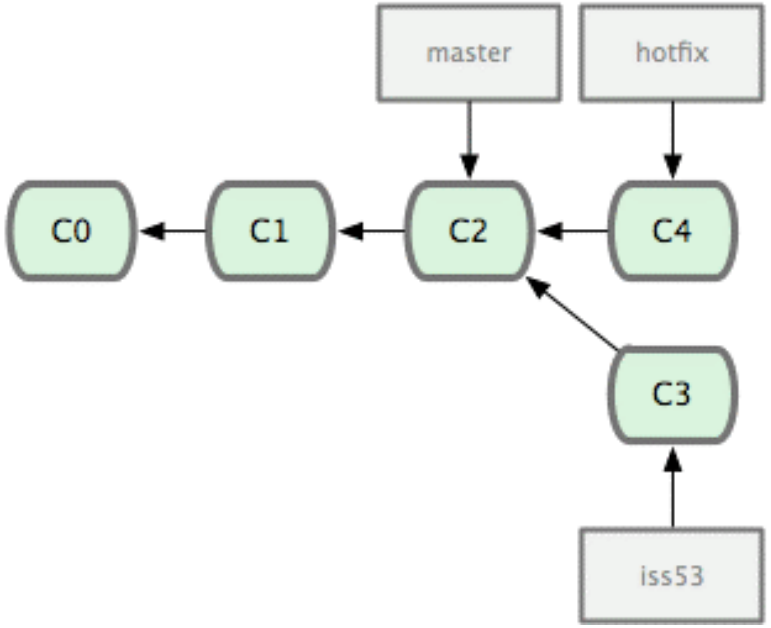
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ vim index.html
harbined@harbined-VirtualBox:~/mon_projet$ git commit -a -m "Modif pour pb53"
[iss53 ec1dad4] Modif pour pb53
 1 file changed, 1 insertion(+), 1 deletion(-)
```

*Création d'une nouvelle branche iss53*

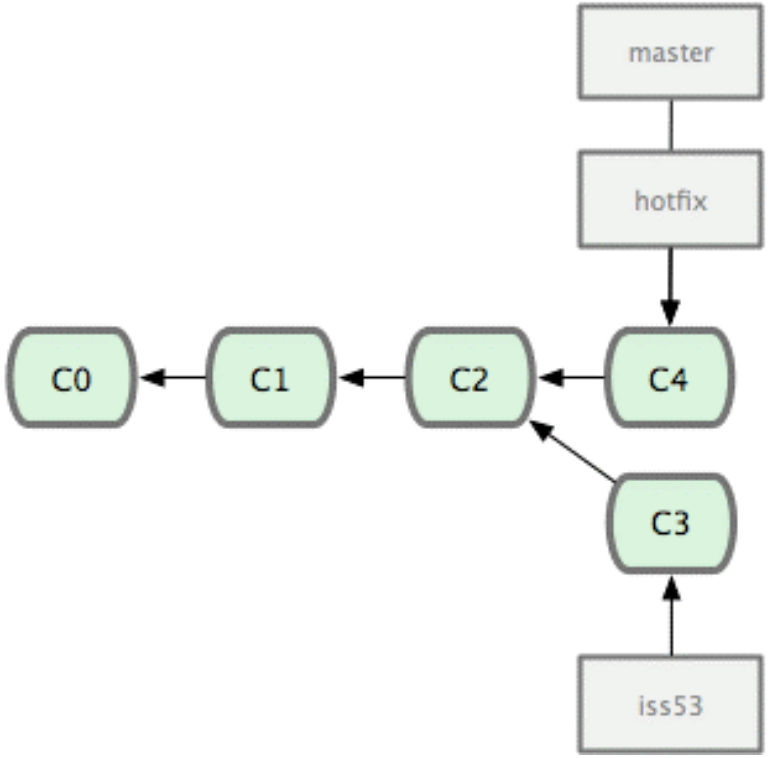
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ git checkout master
Basculement sur la branche 'master'
Votre branche est à jour avec 'origin/master'.
harbined@harbined-VirtualBox:~/mon_projet$ git checkout -b hotfix
Basculement sur la nouvelle branche 'hotfix'
harbined@harbined-VirtualBox:~/mon_projet$ vim index.html
harbined@harbined-VirtualBox:~/mon_projet$ git commit -a -m "Correction rapide"
[hotfix 75a1ca1] Correction rapide
 1 file changed, 1 insertion(+), 1 deletion(-)
```

*Création d'une nouvelle branche hotfix*

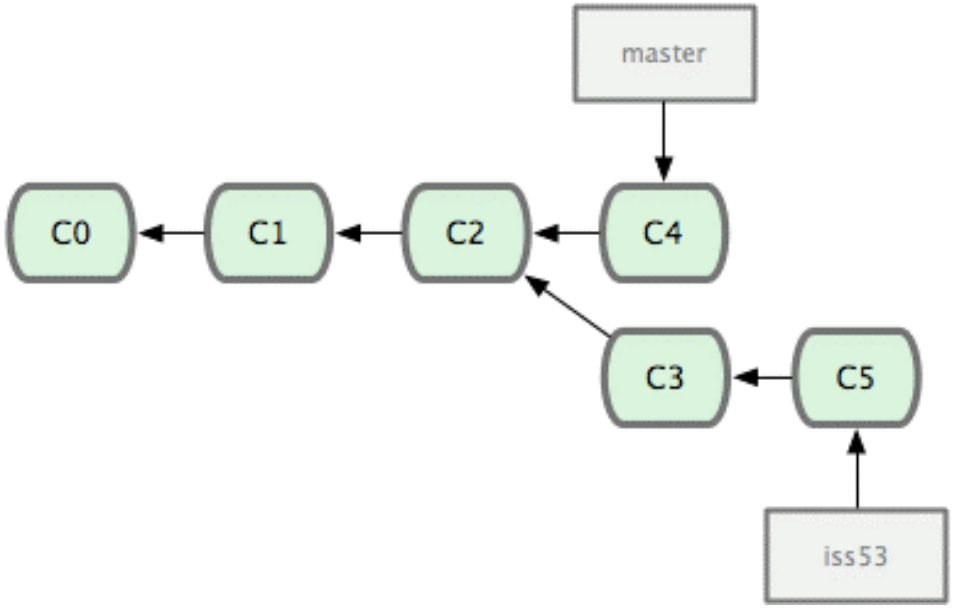
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ git checkout master
Basculement sur la branche 'master'
Votre branche est à jour avec 'origin/master'.
harbined@harbined-VirtualBox:~/mon_projet$ git merge hotfix
Mise à jour aea9d03..75a1ca1
Fast-forward
 index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

*Merge de la branche hotfix avec la branche master*

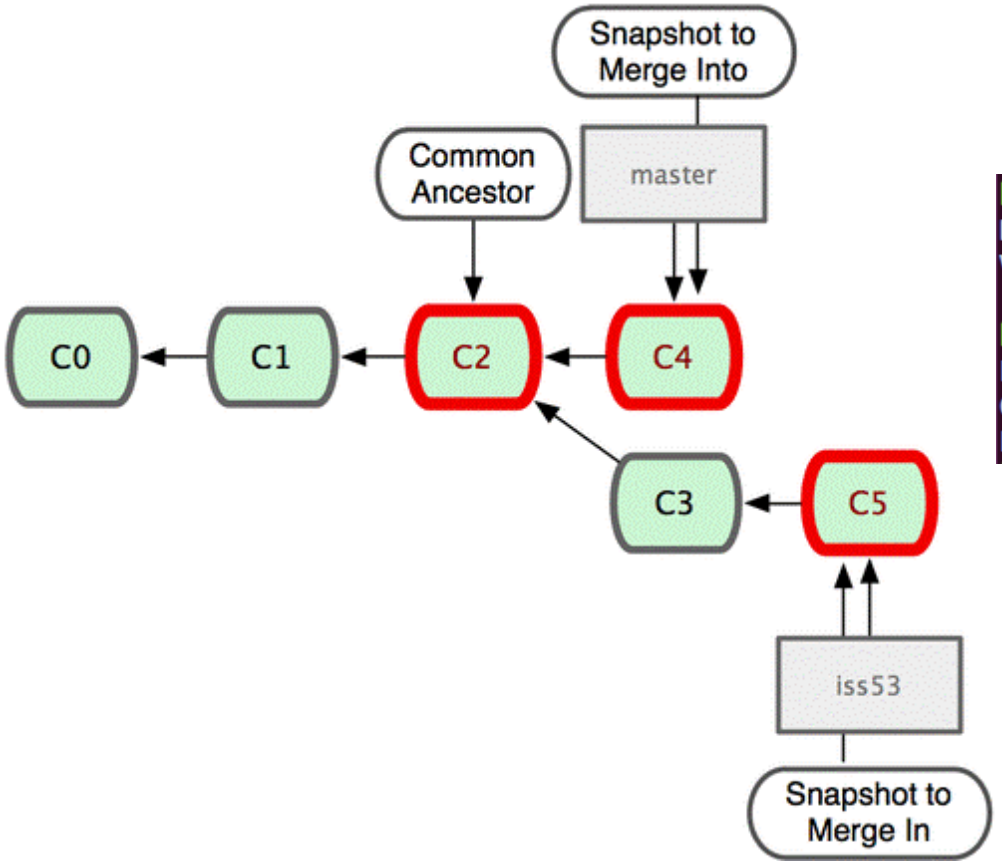
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ git branch -d hotfix
Branche hotfix supprimée (précédemment 75a1ca1).
harbined@harbined-VirtualBox:~/mon_projet$ git checkout iss53
Basculement sur la branche 'iss53'
harbined@harbined-VirtualBox:~/mon_projet$ vim index.html
harbined@harbined-VirtualBox:~/mon_projet$ git commit -a -m "Nouvelle modif"
[iss53 fb6f0c2] Nouvelle modif
 1 file changed, 1 insertion(+), 1 deletion(-)
```

*Suppression de la branche hotfix*

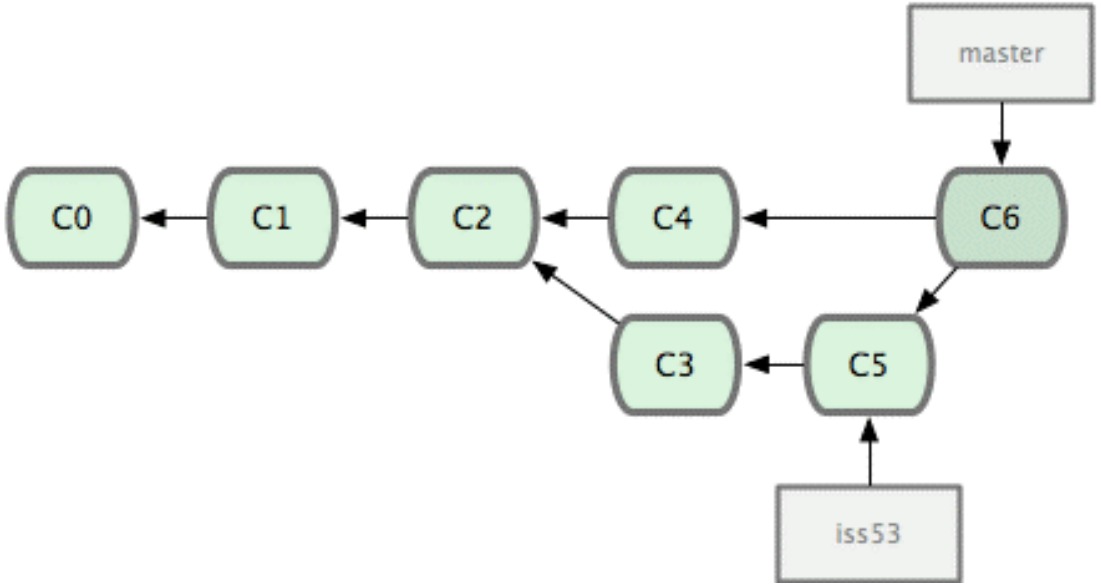
# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ git checkout master
Basculement sur la branche 'master'
Votre branche est en avance sur 'origin/master' de 1 commit.
  (utilisez "git push" pour publier vos commits locaux)
harbined@harbined-VirtualBox:~/mon_projet$ git merge iss53
Fusion automatique de index.html
CONFLIT (contenu) : Conflit de fusion dans index.html
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

*Tentative de merge entre les branches master et iss53*

# Les branches : exemple




```
harbined@harbined-VirtualBox:~/mon_projet$ vim index.html
harbined@harbined-VirtualBox:~/mon_projet$ git add index.html
harbined@harbined-VirtualBox:~/mon_projet$ git commit -m "modif de index"
[master b529895] modif de index
harbined@harbined-VirtualBox:~/mon_projet$ git merge iss53
Already up-to-date.
```

*Merge entre les branches master et iss53*

# Travailler avec un dépôt distant

**Quelques commandes utiles :**

- **$ git clone *<dépôt_distant>***
  Télécharger le *<dépôt_distant>* dans l'espace de travail et se positionner sur la HEAD de sa branche master
- **$ git fetch *<référence>***
  Mettre à jour le dépôt local avec le dépôt distant (désigné par *<référence>*)
- **$ git pull *<dépôt_distant> <référence>***
  Récupérer les informations associées à la référence du *<dépôt_distant>* et les fusionner dans la branche courante
- **$ git push *<dépôt_distant> <branche>***
  Pousse la *<branche>* spécifiée vers le *<dépôt distant>*
- **$ git checkout --track *<branche> <branche_distante>***
  Créer une *<branche>* locale qui suit la *<branche_distante>*

# Travailler avec un dépôt distant : fetch vs. pull

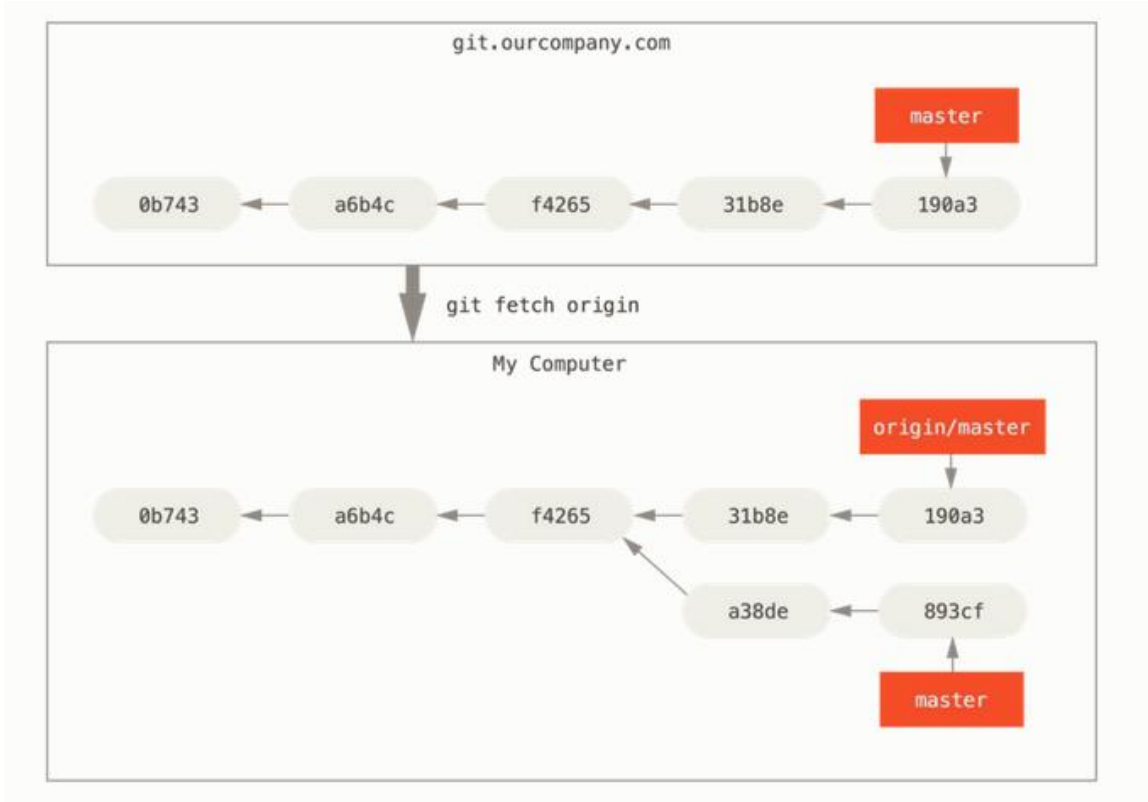



*Illustration de la commande git fetch*

**git pull ≈ git fetch + git merge**

*« Il est généralement préférable de simplement utiliser les commandes fetch et merge explicitement plutôt que de laisser faire la magie de git pull qui peut s'avérer source de confusion. » - https://git-scm.com*

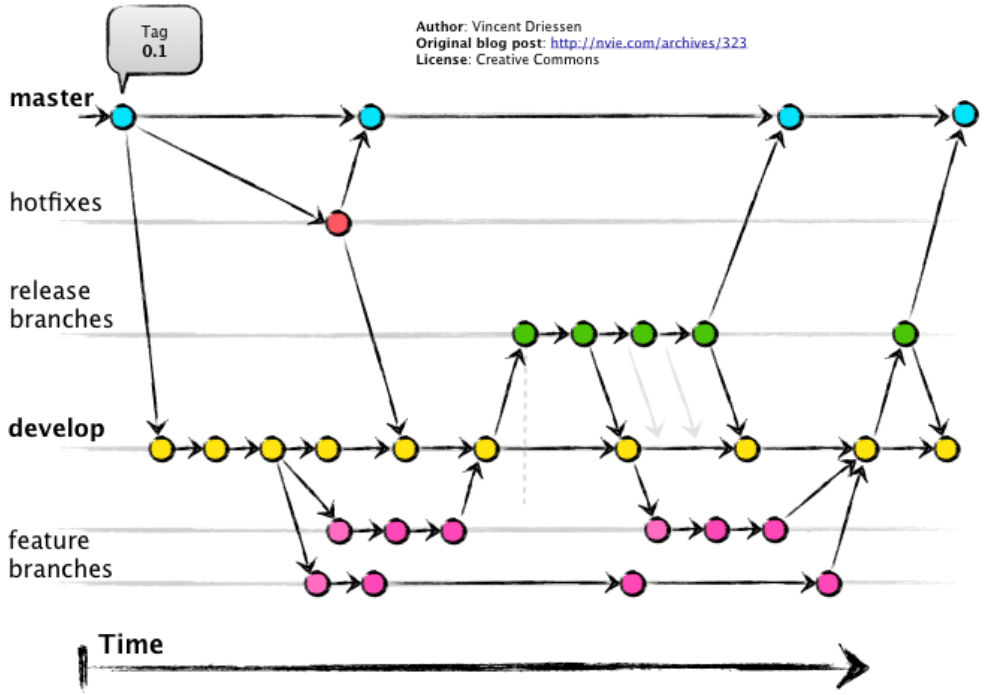
# Travailler avec un dépôt distant : suivre une branche

```
harbined@harbined-VirtualBox:~$ git clone git@gricad-gitlab.univ-grenoble-alpes.fr:harbined/encore_un_projet_bis.git
Clonage dans 'encore_un_projet_bis'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Réception d'objets: 100% (6/6), fait.
Vérification de la connectivité... fait.
harbined@harbined-VirtualBox:~$ cd encore_un_projet_bis/
harbined@harbined-VirtualBox:~/encore_un_projet_bis$ git branch -vv
* master 0cff7f2 [origin/master] Add readme.md
harbined@harbined-VirtualBox:~/encore_un_projet_bis$ git checkout --track origin/
origin/branch1   origin/HEAD      origin/master
harbined@harbined-VirtualBox:~/encore_un_projet_bis$ git checkout --track origin/branch1
La branche branch1 est paramétrée pour suivre la branche distante branch1 depuis origin.
Basculement sur la nouvelle branche 'branch1'
harbined@harbined-VirtualBox:~/encore_un_projet_bis$ git branch -vv
* branch1 638b869 [origin/branch1] Update README.md
  master  0cff7f2 [origin/master] Add readme.md
```

Cloner un projet ne permet pas d'obtenir les différentes branches localement

# Comment utiliser GIT/GITLab au mieux ?

**<u>Gestion :</u> GIT Workflow / GITHub Flow (Gestion de versions ou pas)**

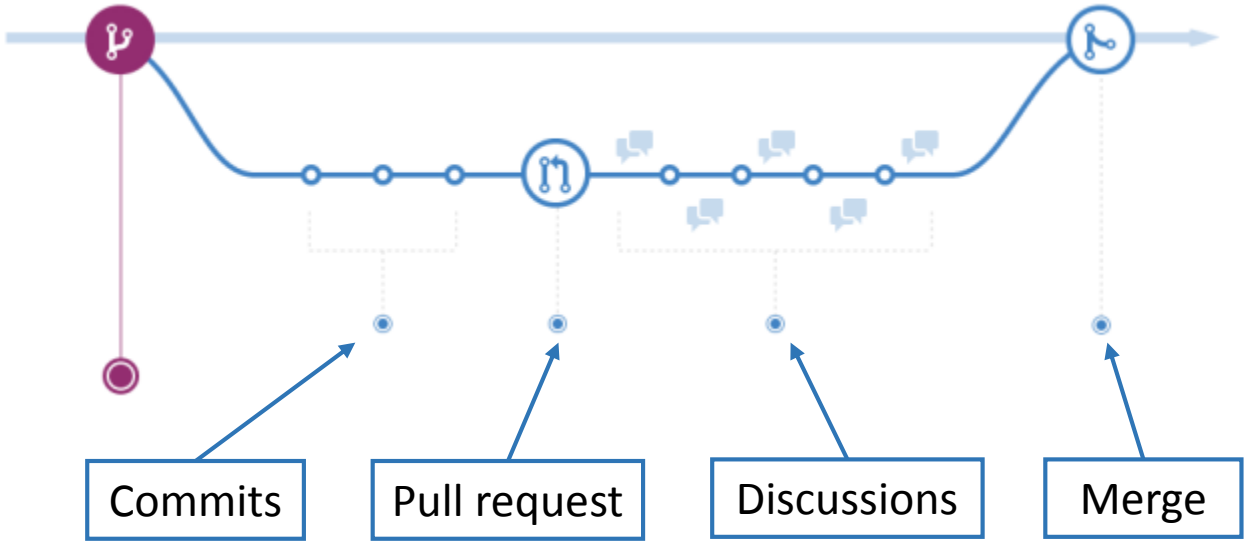



*<u>GIT Workflow</u>*

http://www.nicoespeon.com/fr/2013/08/quel-git-workflow-pour-mon-projet/

# Comment utiliser GIT/GITLab au mieux ?

**Gestion : GIT Workflow / GITHub Flow (Gestion de versions ou pas)**




*GITHub Flow*

http://www.nicoespeon.com/fr/2013/08/quel-git-workflow-pour-mon-projet/

# Comment utiliser GIT/GITLab au mieux ?

**Intégration continue (CI)**

« Ensemble de pratiques consistant à vérifier à chaque modification du code source que le résultat des modifications ne produit pas de régression dans l'application développée. » - Wikipédia

Intérêts :

- Détecter les problèmes au plus tôt et en continu
- Automatiser l'exécution des suites de tests
- Tester immédiatement les modifications

Avec GITLab :

- Besoin d'installer un runner
- Tests écrits dans un fichier : « .gitlab-ci.yml »
- Plus d'informations sur https://docs.gitlab.com/ee/ci/quick_start/

# Comment utiliser GIT/GITLab au mieux ?

**Issues**

L'onglet « Issues » permet de lister différentes idées, améliorations, tâches ou bugs concernant le projet sur GITLab.

Intérêts :

- Meilleure organisation (utilisation de labels, milestones)
- Possibilité d'assigner la tâche à une ou plusieurs personnes
- Réception de notifications (par mail ou dans l'interface GITLab)
- Possibilité de discuter à propos de la tâche
- Automatisation de la gestion des issues
- Gestion des priorités

# Quelques liens

Les commandes :
http://ndpsoftware.com/git-cheatsheet.html

Commencer à utiliser Git en ligne de commande :
https://gricad-gitlab.univ-grenoble-alpes.fr/help/gitlab-basics/start-using-git.md

Créer et ajouter une clé SSH :
https://gricad-gitlab.univ-grenoble-alpes.fr/help/gitlab-basics/create-your-ssh-keys.md

Livre complet :
https://git-scm.com/book/fr/v2