

Gestion de versions

avec git

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr

<https://rudametw.github.io/teaching/>

Bureau F011
Polytech Lille

Moi... *(et ma décharge de responsabilité)*

- ▶ Je suis étranger (hors UE)
- ▶ J'ai un accent
- ▶ Je me **trompe beaucoup** en français
 - ▶ et en info, et en math, et ...
 - ▶ n'hésitez pas à me corriger ou à me demander de répéter
- ▶ Je commence à enseigner
 - ▶ ce cours est tout nouveau
 - ▶ j'accepte des critiques (constructives mais pas que) et surtout des recommandations
 - ▶ n'hésitez pas à poser des questions
- ▶ Je ne suis pas un expert

Comment gérez-vous vos fichiers ?

- ▶ Garder l'historique
- ▶ Partager

Comment gérez-vous vos fichiers ?

- ▶ Garder l'historique
- ▶ Partager



fichier-v1.qqch



fichier-v2.qqch



fichier-v3.qqch



fichier-v4.qqch



fichier-v5.qqch



fichier-v6.qqch



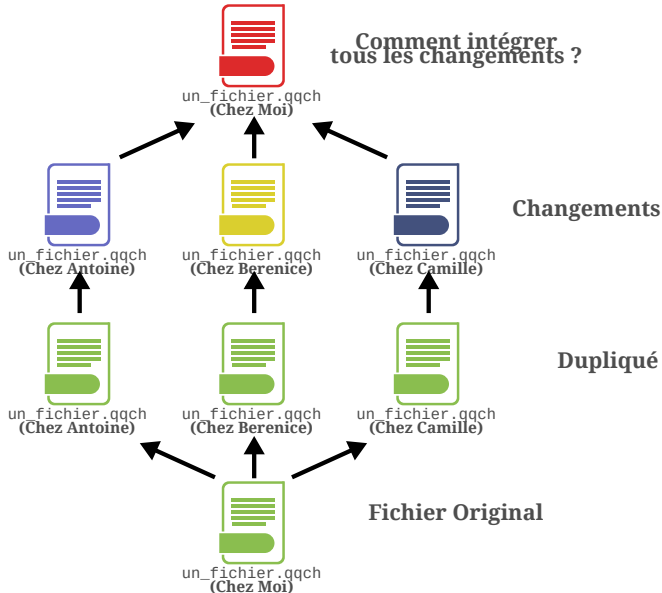
fichier-v7.qqch



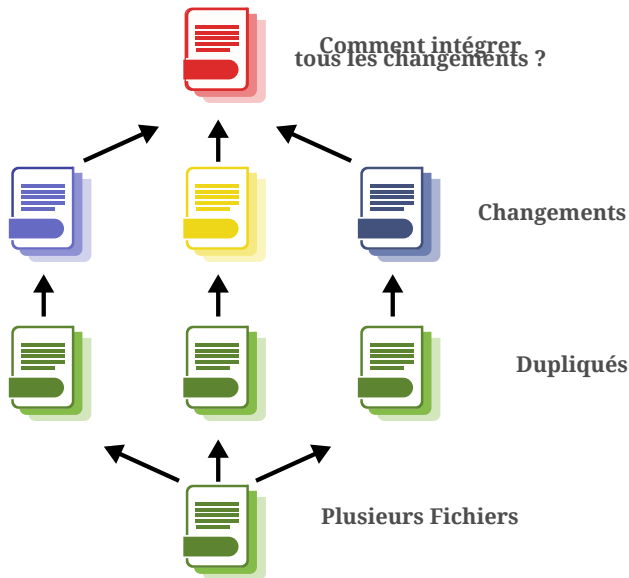
fichier-v8.qqch

Versionnement manuel de fichiers

Comment collaborer sur un fichier ?



Comment collaborer sur plusieurs fichiers ?



D'autres solutions ?



Google docs

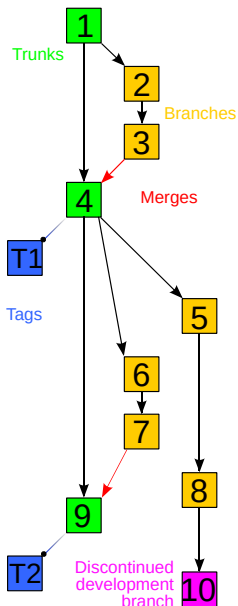


Gestion de versions

La **gestion de versions** (en anglais *version control* ou *revision control*) consiste à maintenir **l'ensemble des versions d'un ou plusieurs fichiers** (généralement en texte). Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout **la gestion des codes source**.

https://fr.wikipedia.org/wiki/Gestion_de_versions

Gestion de versions



Par Revision_controlled_project_visualization.svg: *Subversion_project_visualization.svg: Traced by User:Stannered, original by en:User:Sami Kero-laderivative work: Moxfyre (talk)derivative work: Echion2 (talk) Revision_controlled_project_visualization.svg, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=9562807>

Avantages de la gestion de versions

- ▶ Sauvegarde / Restauration
- ▶ Synchronisation du travail (partage, collaboration)
- ▶ Suivi de changements (très détaillé)
- ▶ Suivi de responsabilités / propriétaires / coupables
- ▶ *Sandboxing* (espace confiné, environnement de test, isolation)
- ▶ *Branching and merging*
- ▶ Passage à l'échelle (10, 100, 1.000, 10.000 développeurs)

Que mettre dans un Logiciel de Gestion de Versions ?

- ▶ Tous les sources du projet
 - ▶ code source (.c .cpp .java .py ...)
 - ▶ scripts de build (Makefile pom.xml ...)
 - ▶ Documentation (.txt .tex Readme ...)
 - ▶ Ressources (images ...)
 - ▶ Scripts divers (déploiement, .sql, .sh ...)

Que mettre dans un Logiciel de Gestion de Versions ?

- ▶ Tous les sources du projet
 - ▶ code source (.c .cpp .java .py ...)
 - ▶ scripts de build (Makefile pom.xml ...)
 - ▶ Documentation (.txt .tex Readme ...)
 - ▶ Ressources (images ...)
 - ▶ Scripts divers (déploiement, .sql, .sh ...)

À NE PAS METTRE

- ▶ Les fichiers générés
 - ▶ Résultat de compilation (.class .o .exe .jar ...)
 - ▶ Autres fichiers générés (.ps .dvi .pdf javadoc ...)

Why the git ?

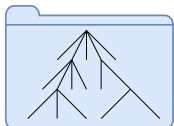
C'est Ze Standard

- ▶ *git - the stupid content tracker*
- ▶ Outil professionnel
- ▶ Rapide, multi-plateforme, flexible, puissant

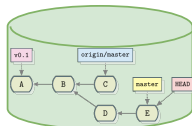
To Share or Not to Share ?

- ▶ Enrichissez vos CV
 - ▶ <https://github.com/>
- ▶ Choisir sa licence
 - ▶ Code — GPL, Apache, BSD, MIT, Propriétaire
<https://choosealicense.com/>
 - ▶ Documents/Rapports — Creative commons
<https://creativecommons.org/>

Concepts et commandes git

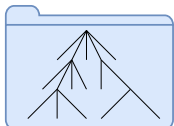


Copie de travail
(Working Directory)



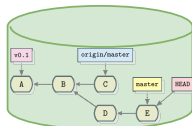
Dépôt

Concepts et commandes git

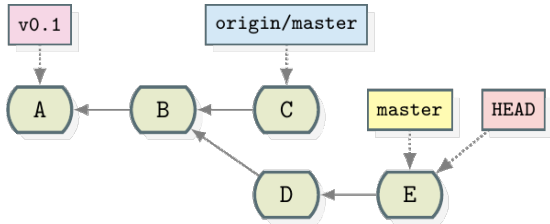


Copie de travail
(Working Directory)

```
├── bin
│   └── program.exe
├── build
├── doc
├── INSTALL
├── lib
│   ├── smixer-hda.so
│   └── libgui.a
├── LICENSE
├── Makefile
├── README
├── src
│   ├── program.c
│   ├── utils.c
│   └── utils.h
```

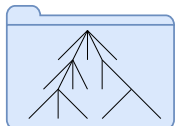


Dépôt

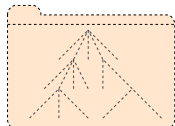


Concepts et commandes git

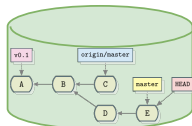
Réseau



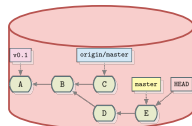
Copie de travail
(Working Directory)



Index (Stage)



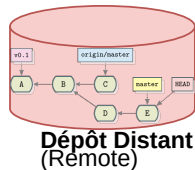
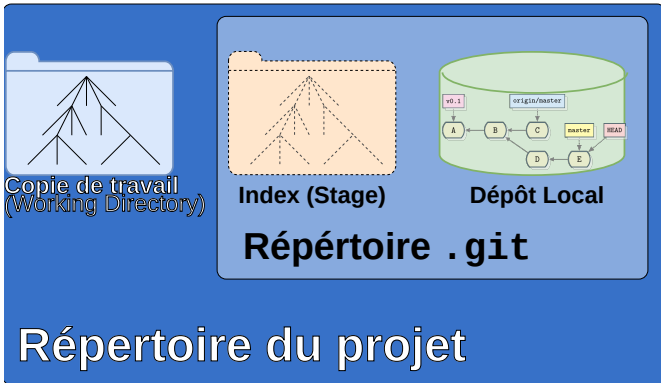
Dépôt Local



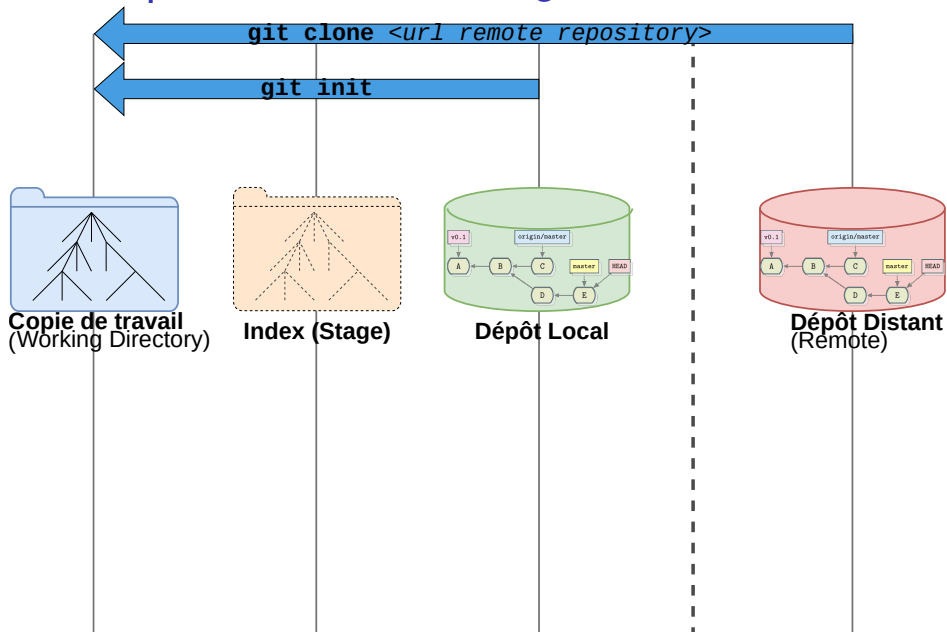
Dépôt Distant
(Remote)

Concepts et commandes git

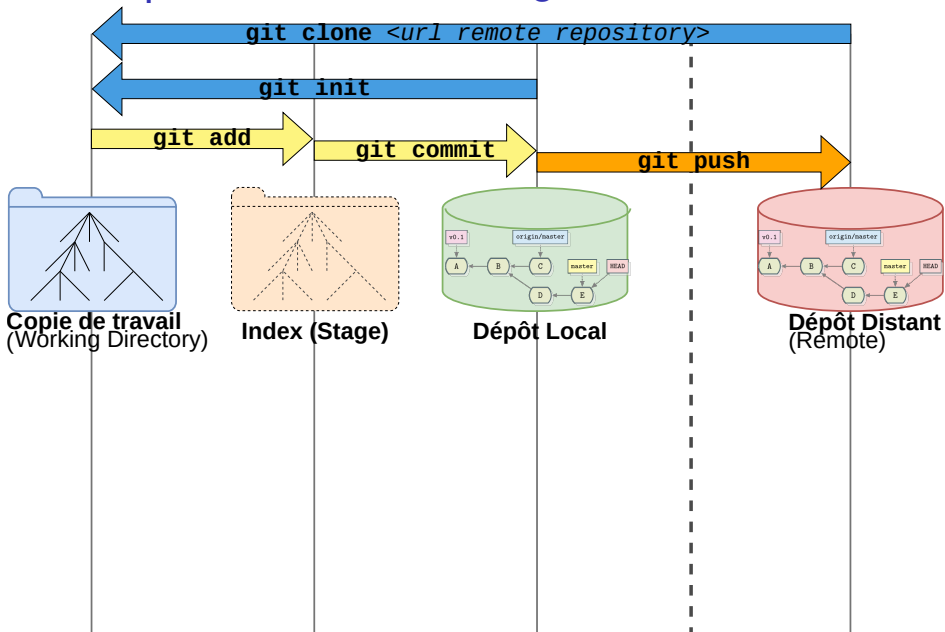
Réseau



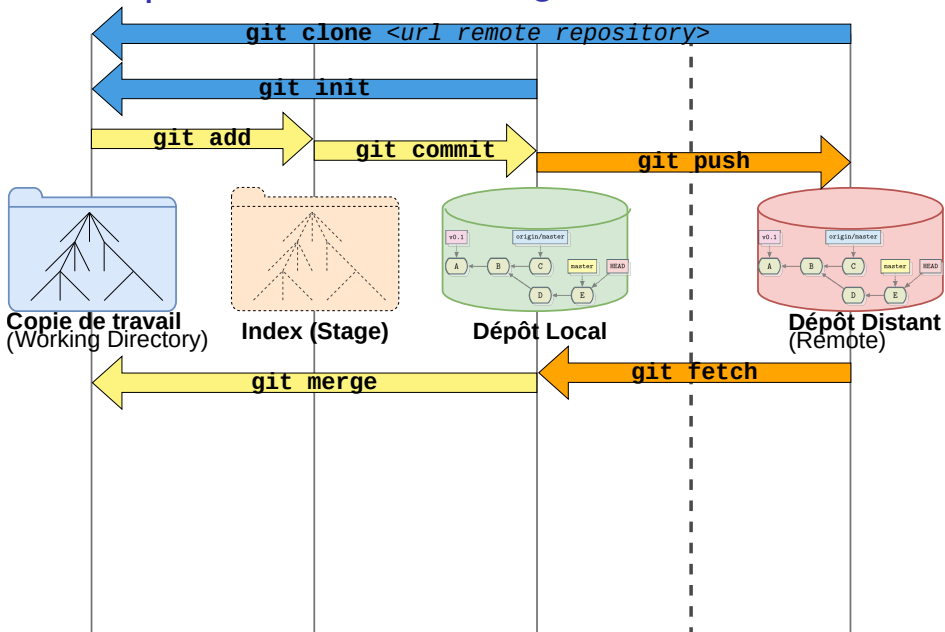
Concepts et commandes git



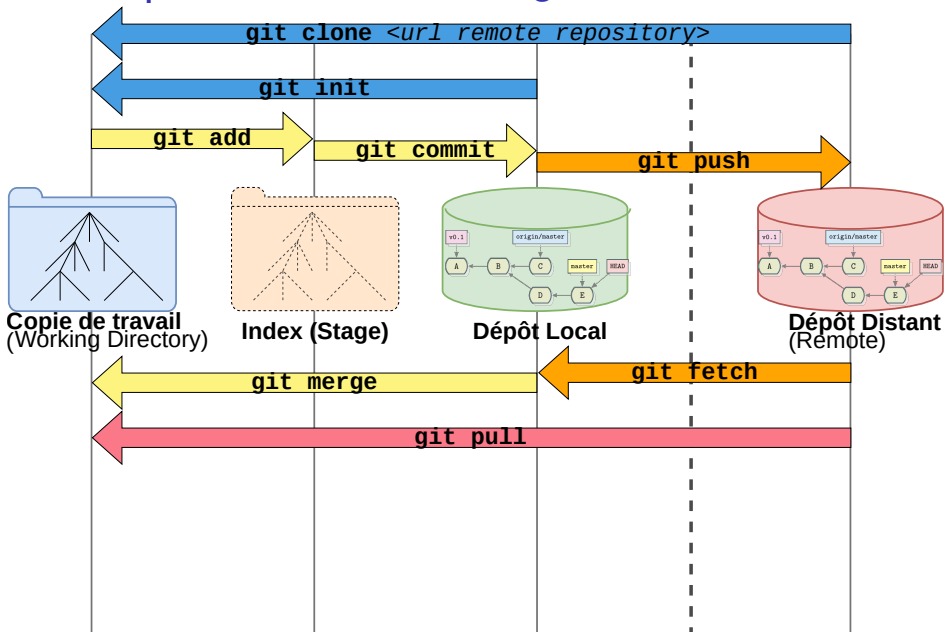
Concepts et commandes git



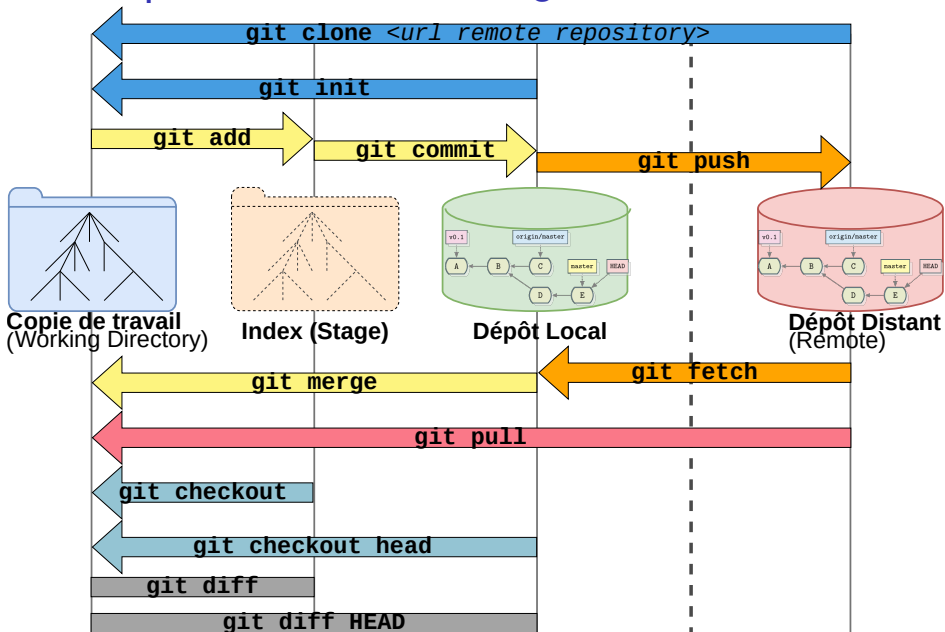
Concepts et commandes git



Concepts et commandes git



Concepts et commandes git



Le Graphe Orienté Acyclique de commits

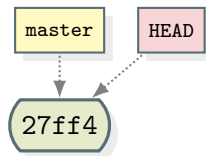
(a) Dépôt vide

Dans un terminal ...

```
mkdir mon_depot ; cd mon_depot
git init .
echo "pomme" >> fruits.txt
git add fruits.txt
git commit -m "Pomme ajouté à la liste de fruits"
⇒ ID = 27ff4
```

Faire `git status` et `git log` après chaque commande!!!

Le Graphe Orienté Acyclique de commits



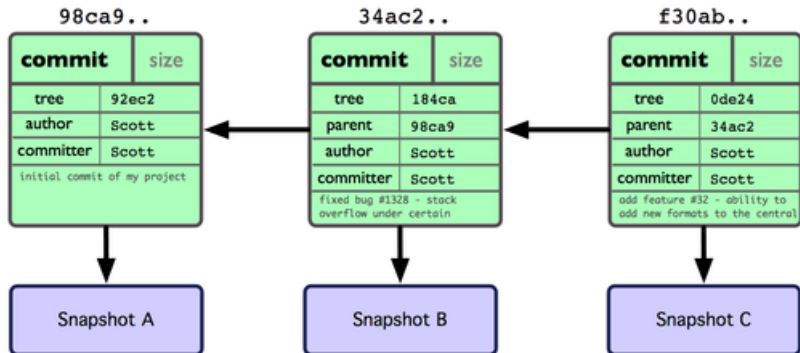
(a) Premier *commit*

Dans un terminal ...

```
mkdir mon_depot ; cd mon_depot
git init .
echo "pomme" >> fruits.txt
git add fruits.txt
git commit -m "Pomme ajouté à la liste de fruits"
⇒ ID = 27ff4
```

Faire `git status` et `git log` après chaque commande!!!

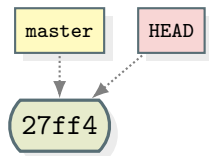
C'est quoi un commit ?



- ▶ Le Commit-ID est une *empreinte* calculé en utilisant la fonction de hachage SHA-1 sur
 - ▶ **Tout** le contenu du commit + Date + Nom et email du commiteur + Message de log + ID du commit parent + ...

Propriété : **Unicité** quasi-universelle de l'ID

Le Graphe : Commit 2



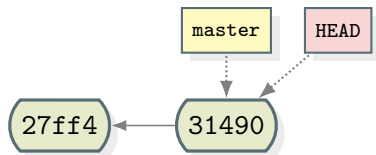
(a) État avant deuxième commit



Dans un terminal ...

```
echo banane >> fruits.txt
git add fruits.txt
git commit -m "Ajouté banane à fruits.txt"
⇒ ID = 31490
```

Le Graphe : Commit 2

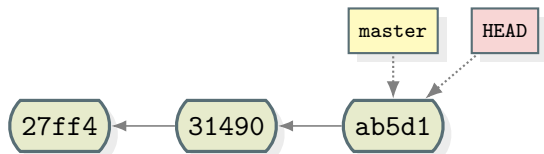


(a) Deuxième *commit*

Dans un terminal ...

```
echo banane >> fruits.txt
git add fruits.txt
git commit -m "Ajouté banane à fruits.txt"
⇒ ID = 31490
```

Le Graphe : Commit 3

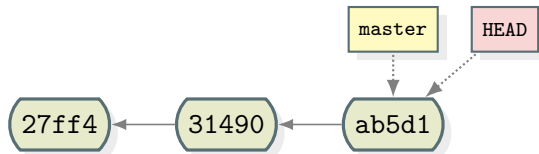


(a) Troisième *commit*

Dans un terminal ...

```
1      echo orange >> fruits.txt
2      git add fruits.txt
3      git commit -m "Ajouté orange à fruits.txt"
4      ⇒ ID = ab5d1
```

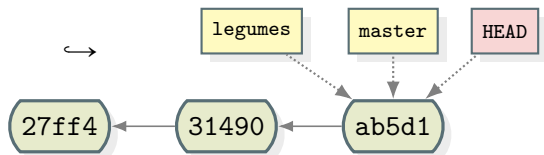
Le Graphe : Branche legumes



(a) Avant branche

```
git branch legumes ; git checkout legumes
```

Le Graphe : Branche legumes

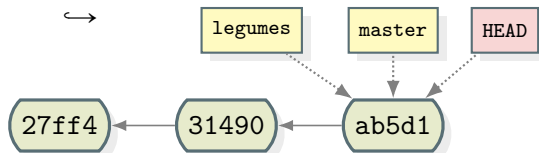


(a) Après branche

⇒ une nouvelle *étiquette* (legumes) apparaît, elle pointe vers le même commit que HEAD

```
git branch legumes ; git checkout legumes
```

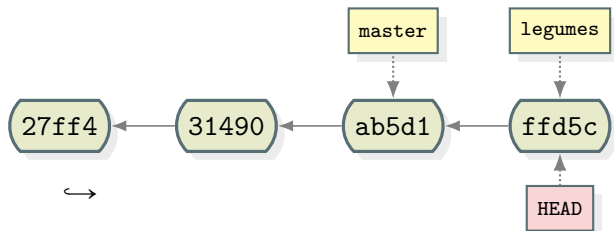
Le Graphe : Branche legumes



(a) Après branche

```
git branch legumes ; git checkout legumes
echo aubergine >> legumes.txt ; git add legumes.txt
git commit -m "Ajout aubergine à legumes"
⇒ ID = ffd5c
```

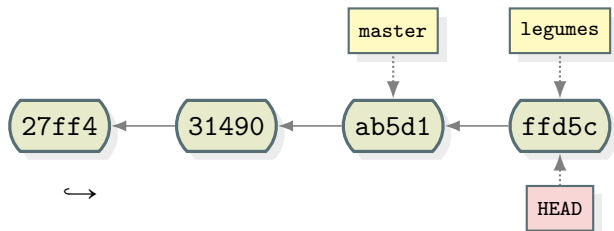
Le Graphe : Branche legumes



(a) Après un premier commit dans la branche legumes

```
git branch legumes ; git checkout legumes
echo aubergine >> legumes.txt ; git add legumes.txt
git commit -m "Ajout aubergine à legumes"
⇒ ID = ffd5c
```


Le Graphe : Branche légumes

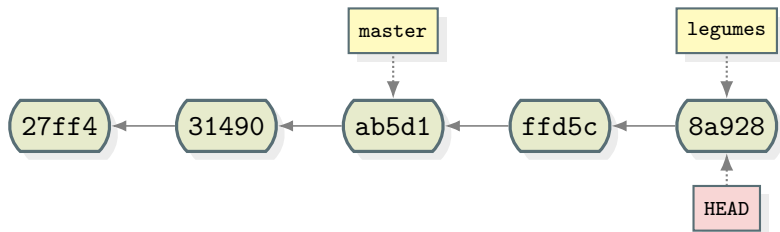


(a) Après un premier commit dans la branche légumes

```
git branch legumes ; git checkout legumes
echo aubergine >> legumes.txt ; git add legumes.txt
git commit -m "Ajout aubergine à legumes"
⇒ ID = ffd5c

echo courgette >> legumes.txt ; git add legumes.txt
git commit -m "Ajout courgette à legumes"
⇒ ID = 8a928
```

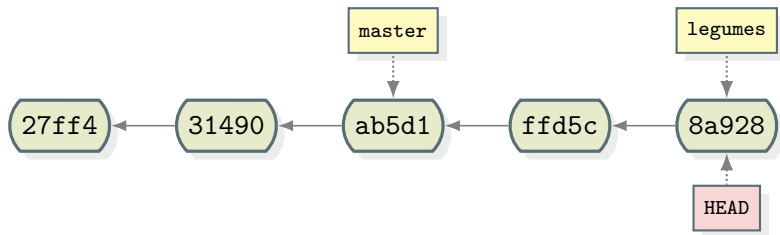
Le Graphe : Branche legumes



(a) Après un deuxième commit dans la branche legumes
↳

```
git branch legumes ; git checkout legumes
echo aubergine >> legumes.txt ; git add legumes.txt
git commit -m "Ajout aubergine à legumes"
    ⇒ ID = ffd5c
echo courgette >> legumes.txt ; git add legumes.txt
git commit -m "Ajout courgette à legumes"
    ⇒ ID = 8a928
```

Le Graphe : Branche master

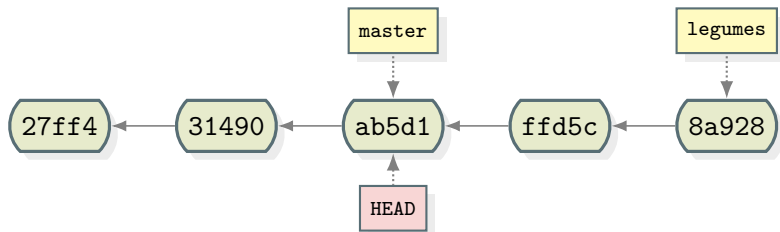


(a) Travaillons sur master



git checkout master

Le Graphe : Branche master

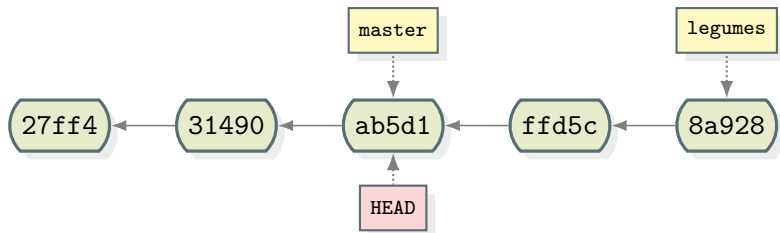


(a) Travaillons sur master

⇒ legumes.txt n'existe plus dans la Copie de Travail
(*Working Directory*)

```
git checkout master
```

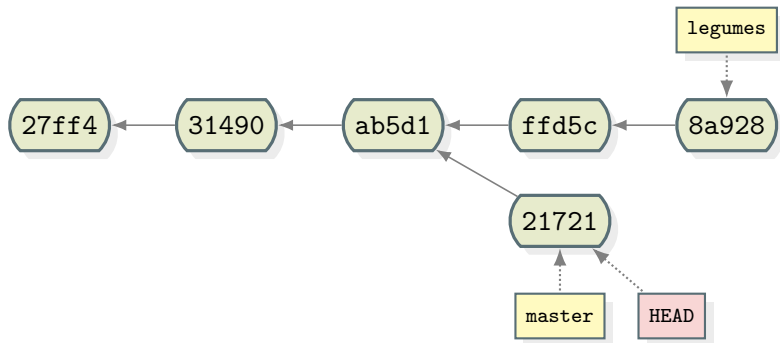
Le Graphe : Branche master



(a) Travaillons sur master

```
git checkout master  
↪  
echo poire >> fruits.txt ; git add fruits.txt  
git commit -m "Ajouté poire à fruits.txt"  
⇒ ID = 21721
```

Le Graphe : Branche master



(a) Après nouveau *commit* sur master

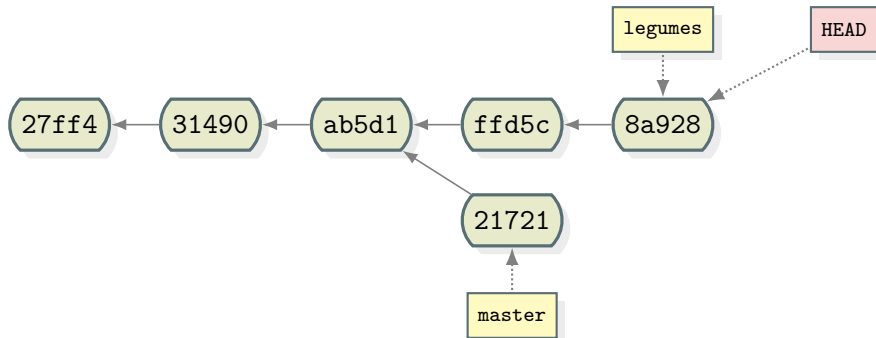
```
git checkout master
```

```
echo poire >> fruits.txt ; git add fruits.txt
```

```
git commit -m "Ajouté poire à fruits.txt"
```

```
↪ ⇒ ID = 21721
```

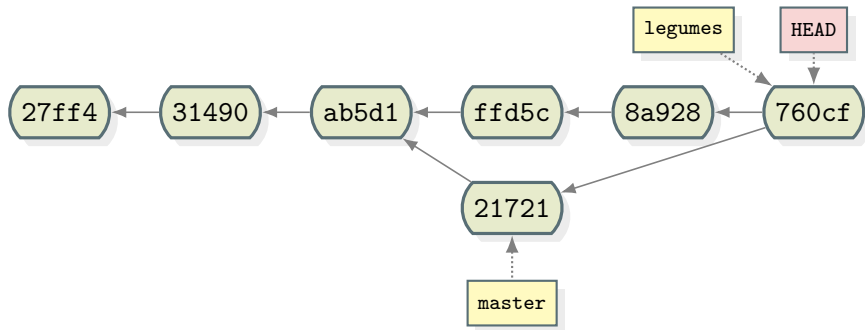
Le Graphe : Merge master \Rightarrow légumes



(a) Allons sur légumes, regardons les différences

```
git checkout légumes  
git diff master
```

Le Graphe : Merge master \Rightarrow légumes



(a) Merger master dans légumes : produit un nouveau commit

```
git checkout légumes
```

```
git diff master
```

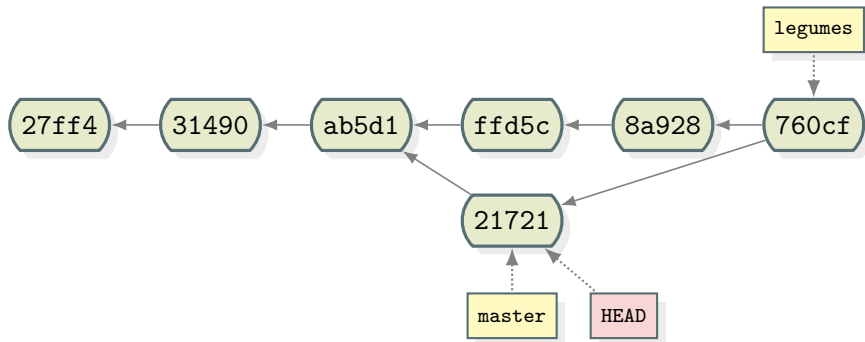
```
git merge master
```


Merge : Vue dans la console

```
wrudamet@beaner[legumes L|✓] ~/COURS/Git/mon_depot $ git l
* 760cf0e [2017-12-01] (HEAD -> refs/heads/legumes) Merge branch 'master' into legumes [rudametw]
|
| * 8a928c9 [2017-12-01] (refs/heads/master) Ajouté poire à fruits.txt [rudametw]
| * 1888830 [2017-12-01] Ajout courgette à legumes [rudametw]
| * ffd5c3e [2017-12-01] Ajout de legumes [rudametw]
|/
* ab5d1c0 [2017-12-01] Ajouté orange à fruits.txt [rudametw]
* 3149017 [2017-12-01] Ajouté banane à fruits.txt [rudametw]
* 27ff4c1 [2017-11-30] Pomme ajouté à la liste de fruits [rudametw]
```

`git log --all --graph --oneline --date=short`

Le Graphe : Merge legumes⇒master

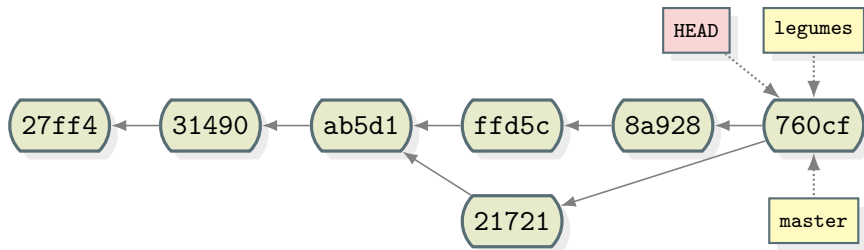


(a) Allons sur master



`git checkout master`

Le Graphe : Merge légumes⇒master



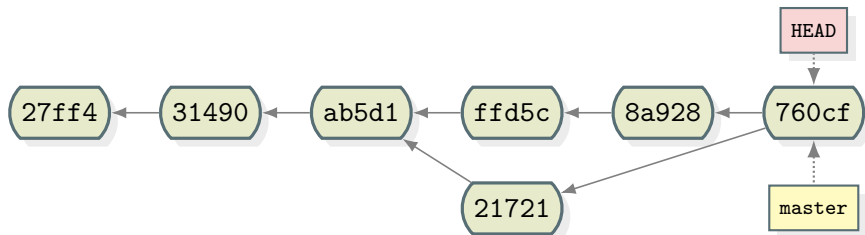
(a) Merger légumes dans master : pas de nouveau commit

```
git checkout master
```

```
git diff légumes
```

```
git merge légumes
```

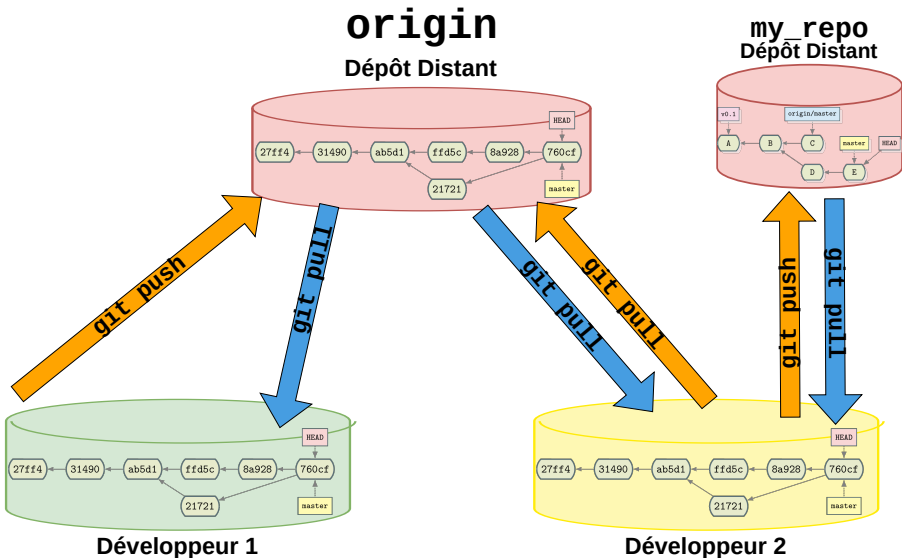
Le Graphe : Merge légumes⇒master



(a) Effacer la branche légumes

```
git checkout master  
git diff légumes  
git merge légumes  
git branch -d légumes
```

Partager : dépôts distants

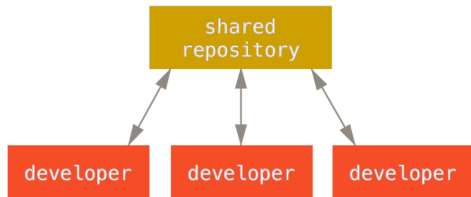


Dépôt Centralisée : initialisation

Premier commit

(dépôt central doit être créé et vide)

```
1 git init .  
2 git add .  
3 git commit -m "first commit"  
4  
5 git remote add origin  
  → git@github.com:rudametw/Learning-Git-Test-Repo.git  
6 git push -u origin master
```

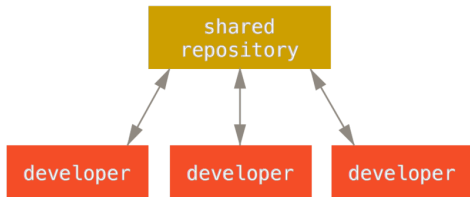


Dépôt Centralisée : initialisation

Premier commit

(dépôt central doit être créé et vide)

```
1 git init .
2 git add .
3 git commit -m "first commit"
4
5 git remote add origin
  → git@github.com:rudametw/Learning-Git-Test-Repo.git
6 git push -u origin master
```



Chaque développeur clone une seule fois

```
1 git clone https://github.com/rudametw/Learning-Git-Test-Repo.git
2 cd Learning-Git-Test-Repo/
3 git remote -v //permet de vérifier les addresses
```

Dépôt Centralisée : travail

Chacun travaille sur une branche `fonctX`. Une fois la fonctionnalité fini, on merge `foncX` dans `master`.

```
git pull ; git status //update & check work
```

```
git branch fonctionnalitéX
```

```
git checkout fonctionnalitéX
```

```
while (je travaille = vrai) {
```

```
    git status ; git diff ;
```

```
    git add XXX
```

```
    git commit XXX}
```

```
}
```

```
git pull -all
```

```
git merge master
```

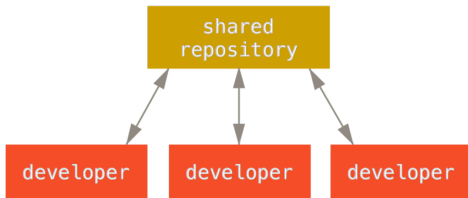
```
//gérer conflits s'il y en a
```

```
//tester que tout marche
```

```
git checkout master
```

```
git merge fonctionnalitéX
```

```
git pull ; git push
```



Résolution de conflits

Des conflits vont se produire ...

... comment faire pour les résoudre ?

Provoquer un conflit dans fruits.txt

Branche ananas

```
git checkout master      1
git branch ananas        2
git checkout ananas      3
awk 'NR==3\{print        4
↪ "ananas"\}1' fruits.txt >
↪ fruits.txt
git add fruits.txt        5
git commit -m "+ananas"   6
```

Branche kaki

```
git checkout master
git branch kaki
git checkout kaki
awk 'NR==3\{print kaki\}1'
↪ fruits.txt | grep -v
↪ orange > fruits.txt
git add fruits.txt
git commit -m "+kaki -orange"
```

Provoquer un conflit dans fruits.txt

Branche ananas

```
git checkout master      1
git branch ananas        2
git checkout ananas      3
awk 'NR==3\{print        4
  ↪ "ananas"\}1' fruits.txt >
  ↪ fruits.txt
git add fruits.txt       5
git commit -m "+ananas"  6
```

Branche kaki

```
git checkout master
git branch kaki
git checkout kaki
awk 'NR==3\{print kaki\}1'
  ↪ fruits.txt | grep -v
  ↪ orange > fruits.txt
git add fruits.txt
git commit -m "+kaki -orange"
```

Les merges

```
1 git checkout master
2 git merge ananas
```

```
3 git merge kaki
```

Sorties console

```
Updating 760cf0e..1711864
Fast-forward
fruits.txt | 1 +
1 file changed, 1 insertion(+)
```

```
Auto-merging fruits.txt
CONFLICT (content): Merge conflict in fruits.txt
Automatic merge failed; fix conflicts and then
  ↪ commit the result.
```

diff entre ananas et kaki avant de merger

```
wrudamet@beaner[merge_fruits L|✓] ~/COURS/Git/mon_depot $ git diff 1711864 34dabb6
diff --git a/fruits.txt b/fruits.txt
index e3922ba..5dbddd0 100644
--- a/fruits.txt
+++ b/fruits.txt
@@ -1,5 +1,4 @@
  pomme
  banane
- ananas
- orange
+ kaki
  poire
```

Différences entre les *commits* réalisés sur les branches kaki et ananas qui avaient pour objectif de produire un conflit. En rouge, les lignes qui existent sur la branche ananas et pas kaki. En vert les lignes qui existent sur la branche kaki et pas ananas.

Résoudre un conflit dans fruits.txt

immédiatement après la commande `git merge kaki`

Conflit dans fruits.txt

git ajoute des guides pour s'y
retrouver

```
1 pomme
2 banane
3 <<<<<< HEAD
4 ananas
5 orange
6 ||| merged common ancestors
7 orange
8 =====
9 kaki
10 >>>>>>
11 poire
```

Résoudre un conflit dans fruits.txt

immédiatement après la commande `git merge kaki`

Conflit dans fruits.txt

git ajoute des guides pour s'y
retrouver

```
1 pomme
2 banane
3 <<<<<< HEAD
4 ananas
5 orange
6 ||| merged common ancestors
7 orange
8 =====
9 kaki
10 >>>>>>
11 poire
```

Solution (édité à la main)

```
1 pomme
2 banane
3 ananas
4 kaki
5 poire
```

Résoudre un conflit dans fruits.txt

immédiatement après la commande `git merge kaki`

Conflit dans fruits.txt

git ajoute des guides pour s'y retrouver

```
1 pomme
2 banane
3 <<<<<< HEAD
4 ananas
5 orange
6 ||| merged common ancestors
7 orange
8 =====
9 kaki
10 >>>>>>
11 poire
```

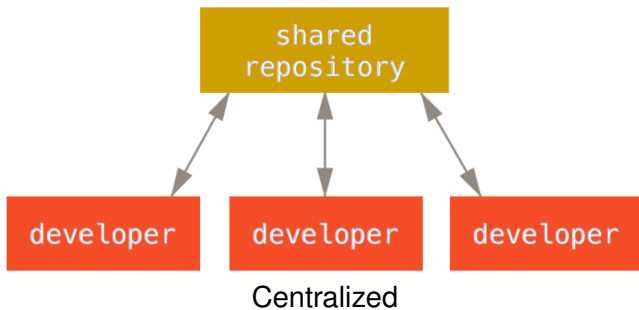
Solution (édité à la main)

```
1 pomme
2 banane
3 ananas
4 kaki
5 poire
```

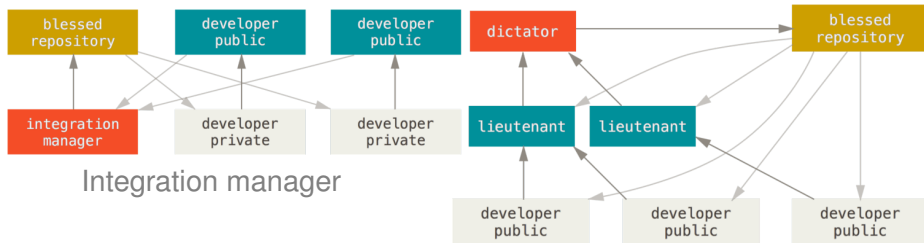
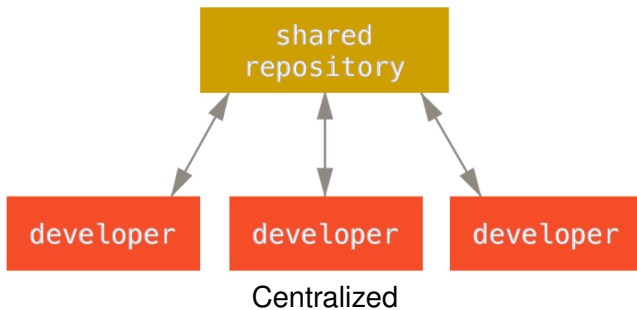
Résolution du conflit

```
1 git add fruits.txt
2 git status
3 git commit -m "Merge branch
   ↳ 'kaki' into master"
4 git pull
5 git push
```

Git distribué : Développements distribués



Git distribué : Développements distribués



Liens, aides et outils (1/2)

► References bibliographiques

- Livre Pro-Git De Scott Chacon and Ben Straub

<https://git-scm.com/book/fr/v2>

- Git Magic (Stanford)

<https://crypto.stanford.edu/~blynn/gitmagic/intl/fr/book.pdf>

- Presentation GIT Les bases de GIT <https://fr.slideshare.net/PierreSudron/diapo-git>

► Où stocker vos projets

- <https://archives.plil.fr/>

- <https://github.com/>

- <https://bitbucket.org/>

- Votre serveur perso

Liens, aides et outils (2/2)

► Tutoriels

- <http://www.cristal.univ-lille.fr/TPGIT/>
- <https://learngitbranching.js.org/>
- <https://try.github.io/>
- <https://www.miximum.fr/blog/enfin-comprendre-git/>

► Vidéos

- <https://www.youtube.com/watch?v=0qmSzXDrJBk>
- https://www.youtube.com/watch?v=uR6G2v_WsRA
- <https://www.youtube.com/watch?v=3a2x1iJFJWc>
- <https://www.youtube.com/watch?v=1ffbJ4sVUb4>
- <https://www.youtube.com/watch?v=duqBHik7nRo>