

# Gestion de versions

avec git

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr  
<https://rudametw.github.io/teaching/>

Bureau F011  
Polytech Lille

1/30

## 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

2/30

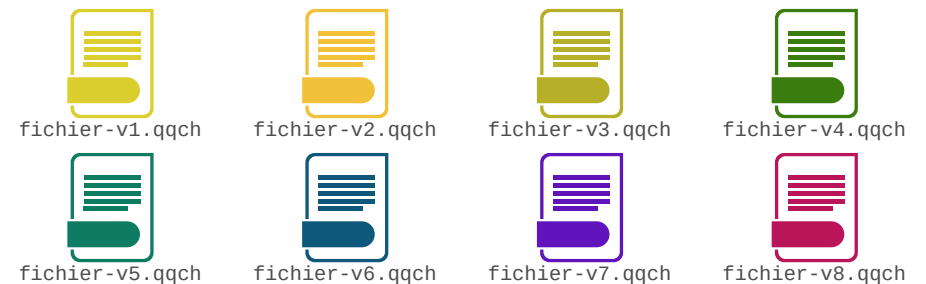
## Comment gérez-vous vos fichiers ?

- ▶ Garder l'historique
- ▶ Partager

3/30

## Comment gérez-vous vos fichiers ?

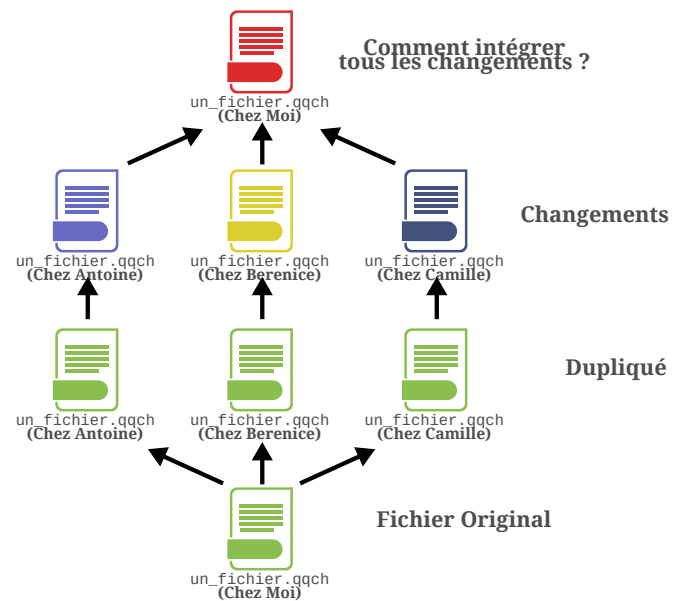
- ▶ Garder l'historique
- ▶ Partager



**Versionnement manuel de fichiers**

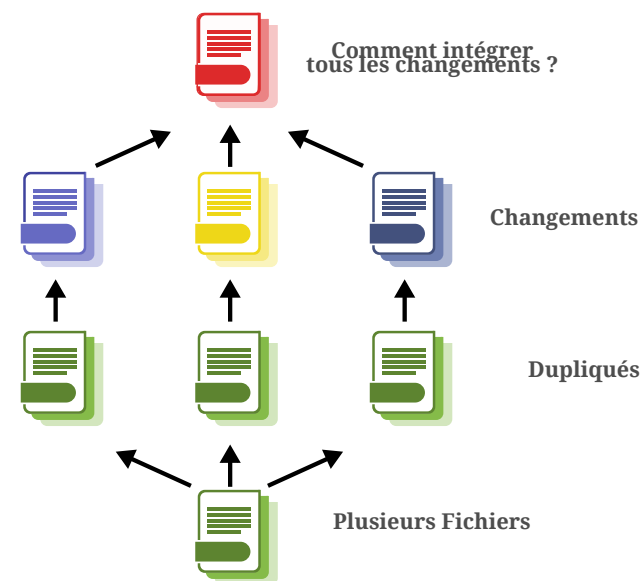
3/30

## Comment collaborer sur un fichier ?



4 / 30

## Comment collaborer sur plusieurs fichiers ?



5 / 30

## D'autres solutions ?



6 / 30

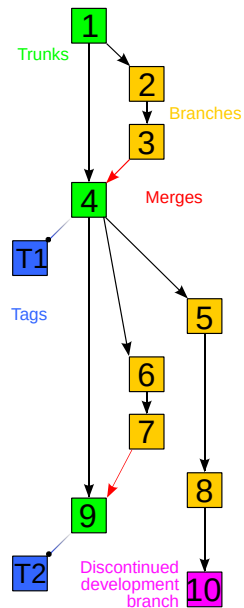
## Gestion de versions

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](https://fr.wikipedia.org/wiki/Gestion_de_versions)

7 / 30

## 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>

8/30

## 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)

9/30

## 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 ...)

10/30

## 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 ...)

10/30

# Why the git ?

## C'est Ze Standard

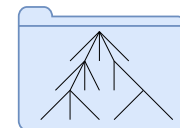
- ▶ *git - the stupid content tracker*
- ▶ Outil professionnel
- ▶ Rapide, multi-plateforme, flexible, puissant

## To Share or Not to Share ?

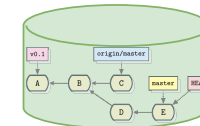
- ▶ Enrichissez vos CV
  - ▶ <https://github.com/>
- ▶ Choisir sa licence
  - ▶ Code — GPL, Apache, BSD, MIT, Propriétaire  
<https://choosealicense.com/>
  - ▶ Documents/Rapports — Creative commons  
<https://creativecommons.org/>

11 / 30

# Concepts et commandes git



**Copie de travail**  
(Working Directory)



**Dépôt**

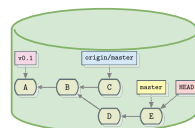
12 / 30

# Concepts et commandes git

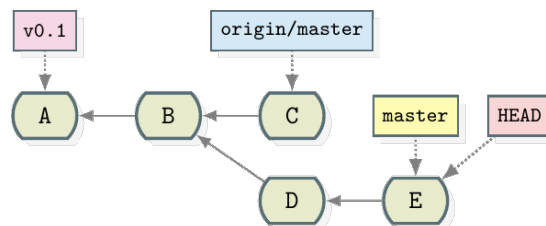


**Copie de travail**  
(Working Directory)

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

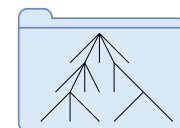


**Dépôt**

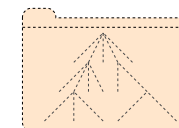


12 / 30

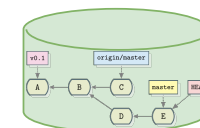
# Concepts et commandes git



**Copie de travail**  
(Working Directory)

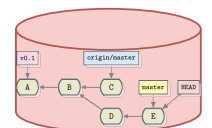


**Index (Stage)**



**Dépôt Local**

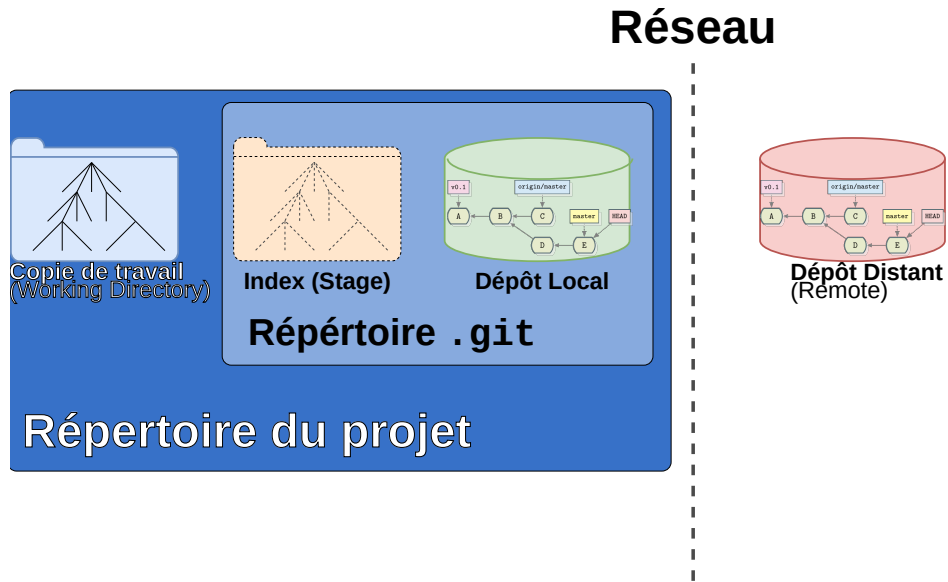
## Réseau



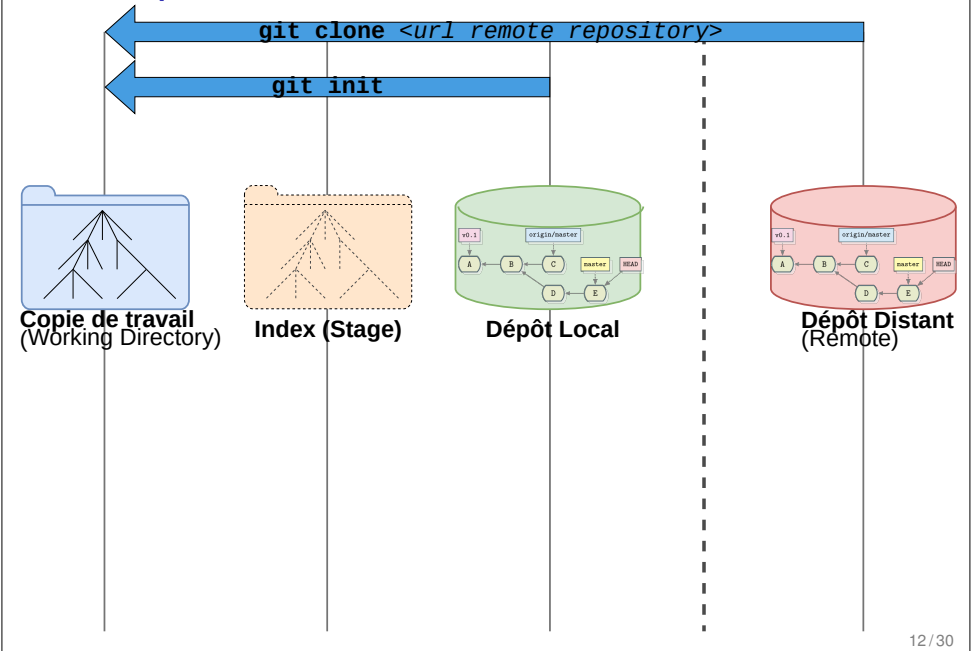
**Dépôt Distant**  
(Remote)

12 / 30

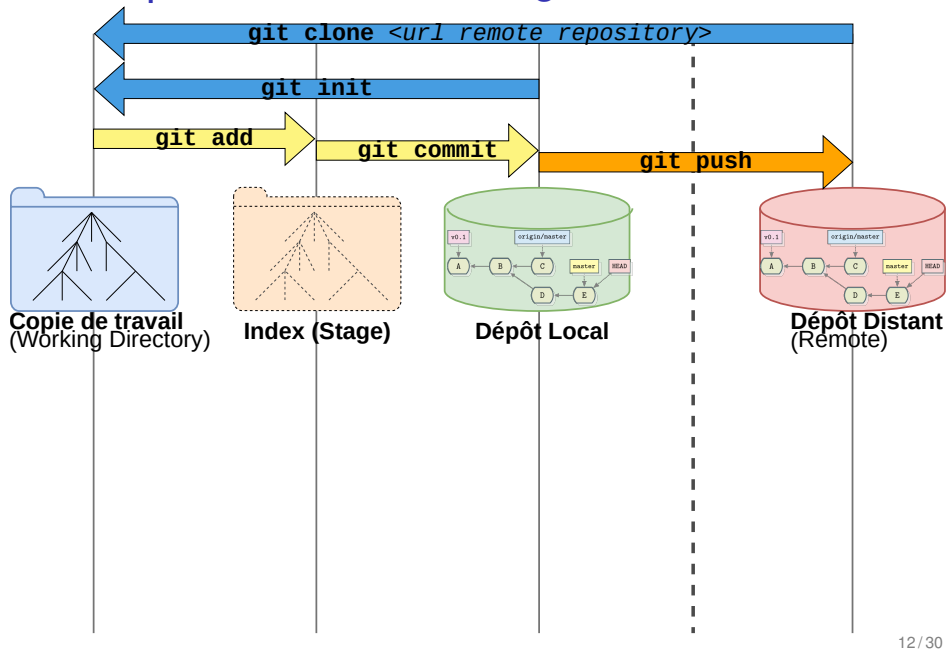
## Concepts et commandes git



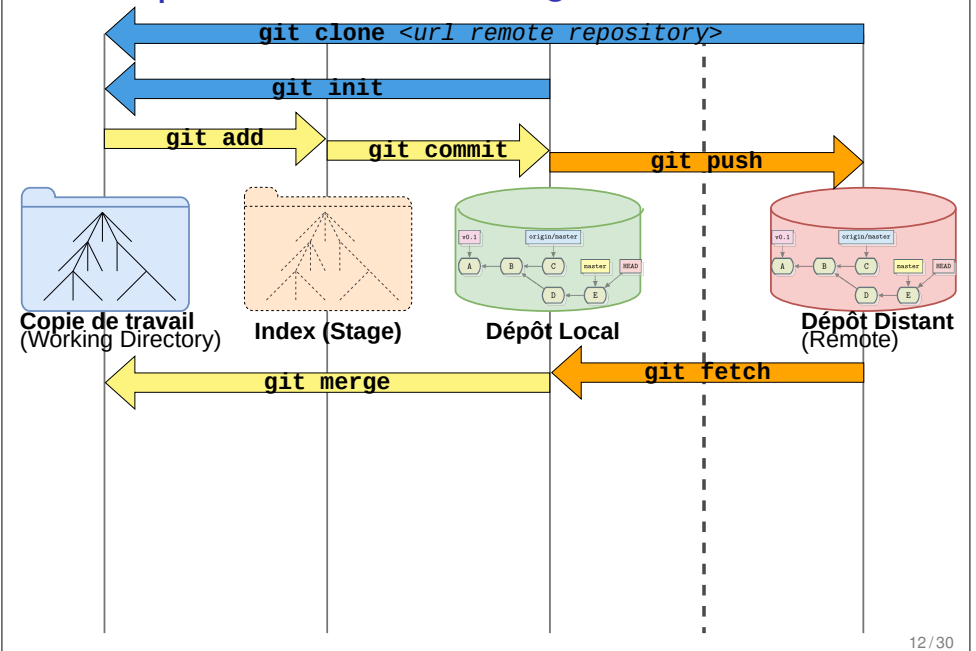
## Concepts et commandes git



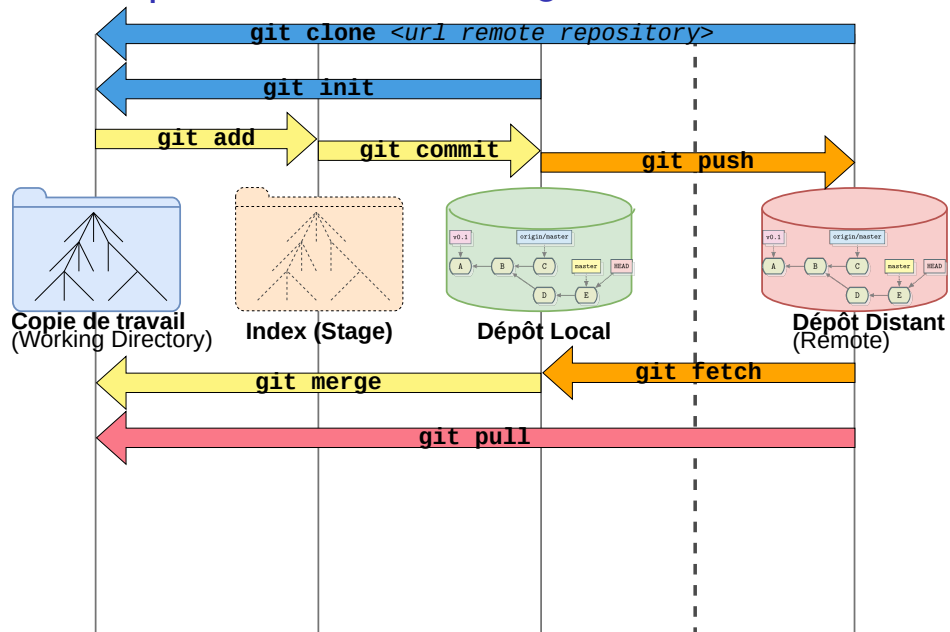
## Concepts et commandes git



## Concepts et commandes git

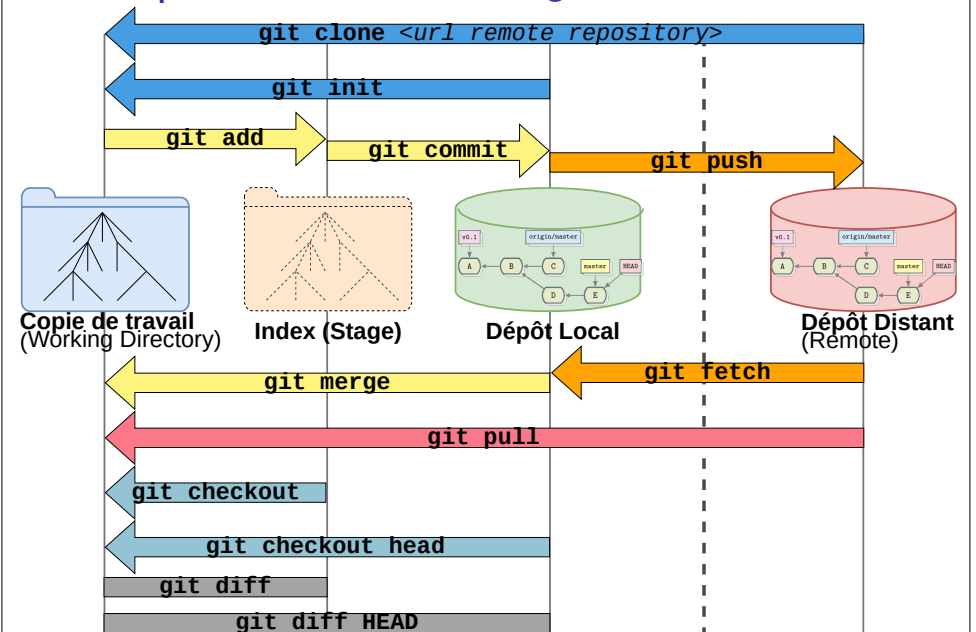


## Concepts et commandes git



12/30

## Concepts et commandes git



12/30

## Le Graphe Orienté Acyclique de commits

(a) Dépôt vide

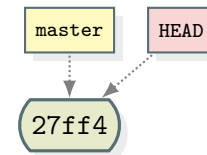
### Dans un terminal ...

```
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!!!

13/30

## Le Graphe Orienté Acyclique de commits



(a) Premier commit

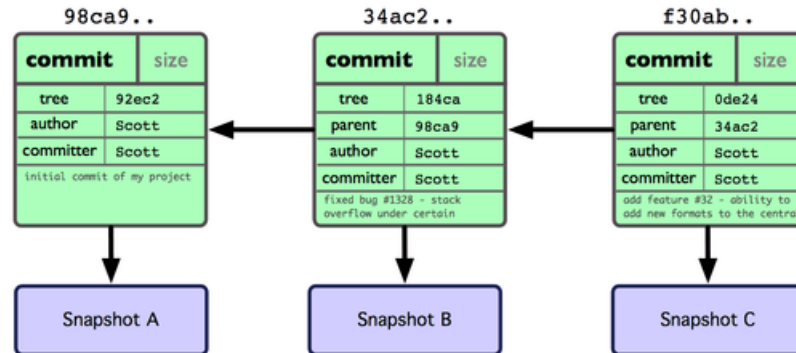
### Dans un terminal ...

```
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!!!

13/30

## C'est quoi un commit ?

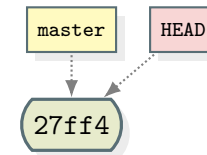


- Le Commit-ID est une *empreinte* calculé en utilisant la fonction de hachage SHA-1 sur
  - Tout** le contenu du commit + Date + Nom et email du commiteur + Message de log + ID du commit parent + ...

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

14/30

## Le Graphe : Commit 2



(a) État avant deuxième commit

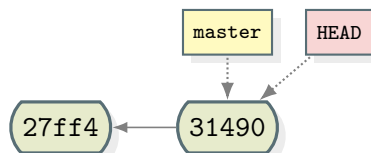
### Dans un terminal ...

```

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

15/30

## Le Graphe : Commit 2



(a) Deuxième commit

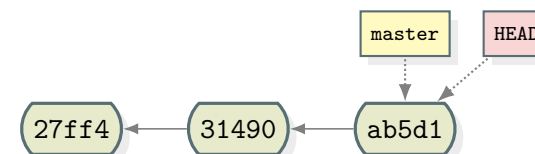
### Dans un terminal ...

```

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

15/30

## Le Graphe : Commit 3



(a) Troisième commit

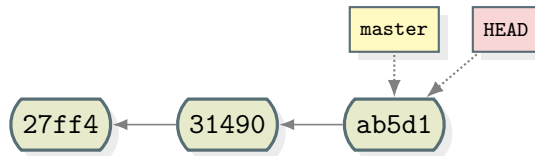
### Dans un terminal ...

```

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

16/30

## Le Graphe : Branche legumes

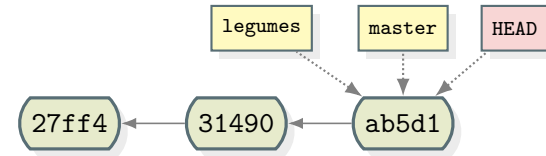


(a) Avant branche

↪ `git branch legumes ; git checkout legumes`

17/30

## Le Graphe : Branche legumes



(a) Après branche

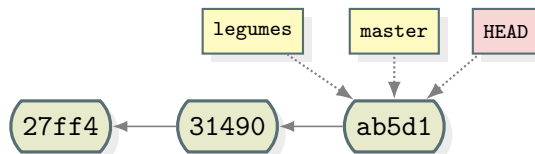
⇒ une nouvelle *étiquette* (legumes) apparaît, elle pointe vers le même commit que HEAD

`git branch legumes ; git checkout legumes`

↪

17/30

## Le Graphe : Branche legumes

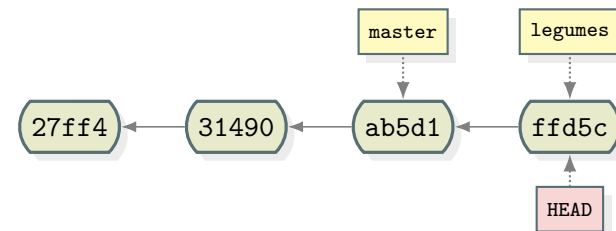


(a) Après branche

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

17/30

## Le Graphe : Branche legumes



(a) Après un premier commit dans la branche legumes

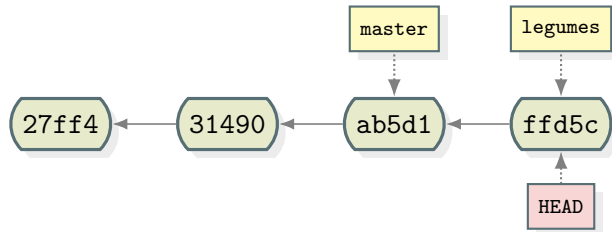
`git branch legumes ; git checkout legumes`  
`echo aubergine >> legumes.txt ; git add legumes.txt`  
`git commit -m "Ajout aubergine à legumes"`  
 ⇒ ID = ffd5c

↪

17/30



## Le Graphe : Branche legumes



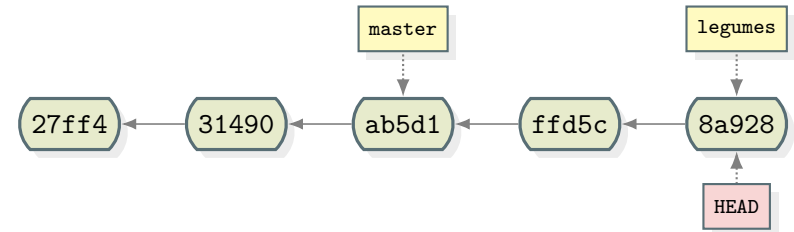
(a) Après un premier commit dans la branche legumes

```

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

17/30

## Le Graphe : Branche legumes



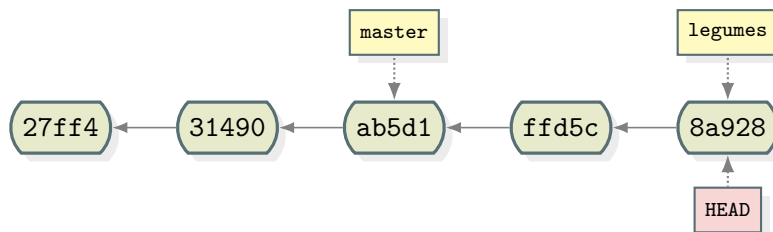
(a) Après un deuxième commit dans la branche legumes

```

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
  
```

17/30

## Le Graphe : Branche master

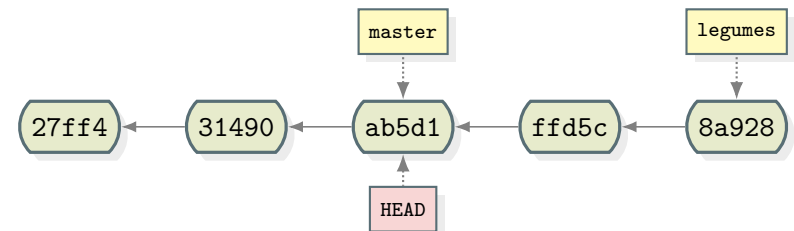


(a) Travaillons sur master

↪ git checkout master

18/30

## Le Graphe : Branche master



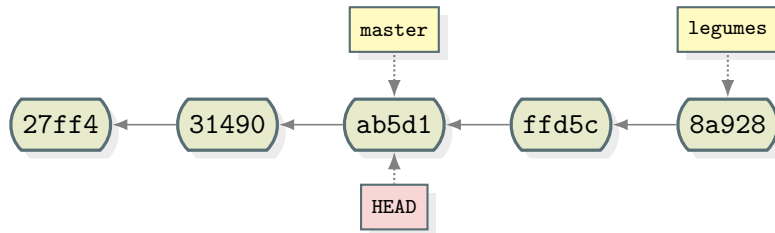
(a) Travaillons sur master

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

↪ git checkout master

18/30

## Le Graphe : Branche master



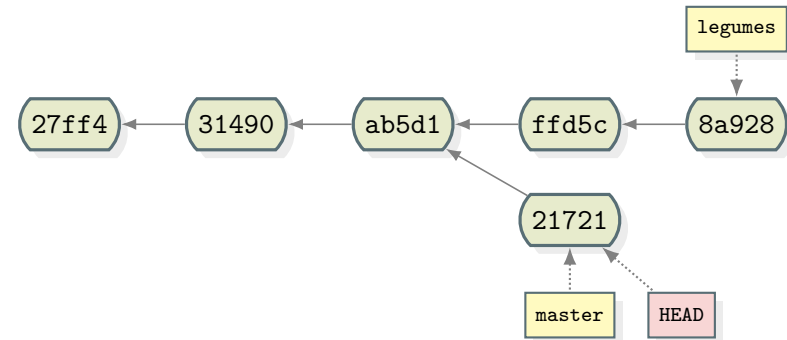
(a) Travaillons sur master

```
git checkout master
```

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

18/30

## Le Graphe : Branche master



(a) Après nouveau commit sur master

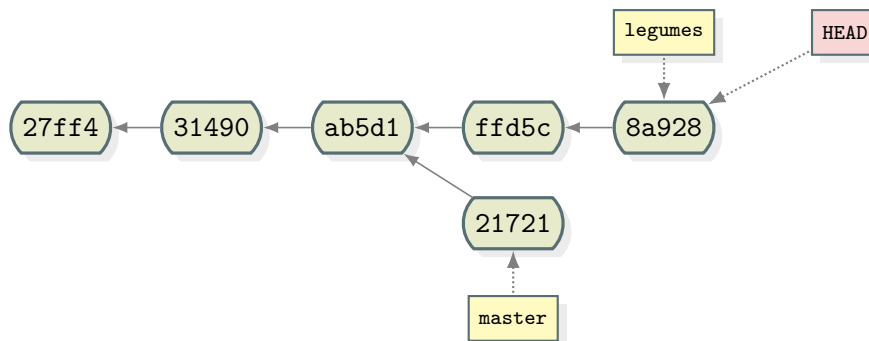
```
git checkout master
```

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

↪

18/30

## Le Graphe : Merge master ⇒ légumes



(a) Allons sur légumes, regardons les différences

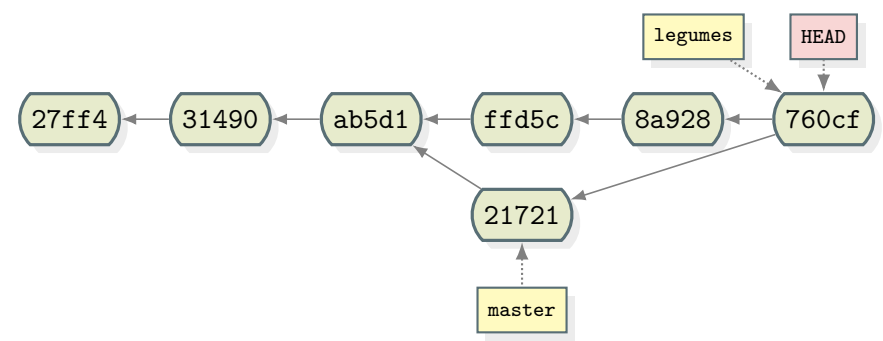
```
git checkout legumes
```

```
git diff master
```

↪

19/30

## Le Graphe : Merge master ⇒ légumes



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

```
git checkout legumes
```

```
git diff master
```

```
git merge master
```

↪

19/30

## Merge : Vue dans la console

```

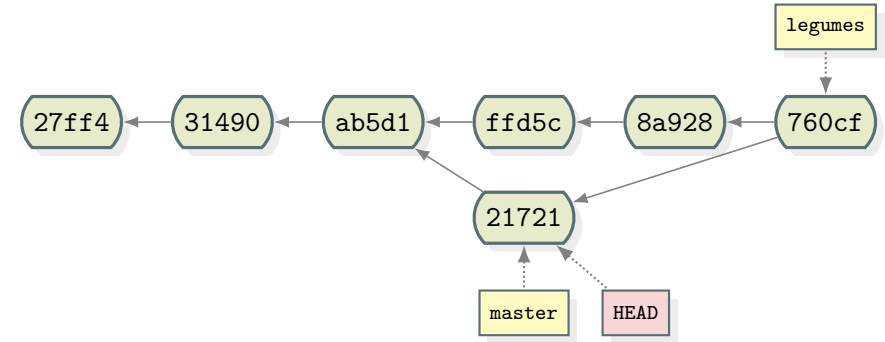
rudamet@beaner[legumes L|✓] ~/COURS/Git/mon_depot $ git l
* 760cf0e [2017-12-01] (HEAD -> refs/heads/legumes) Merge branch 'master' into legumes [rudametw]
* 8a928c9 [2017-12-01] (refs/heads/master) Ajouté poire à fruits.txt [rudametw]
* 1888830 [2017-12-01] Ajout courgette à legumes [rudametw]
* ffd5c3e [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

20/30

## Le Graphe : Merge legumes⇒master



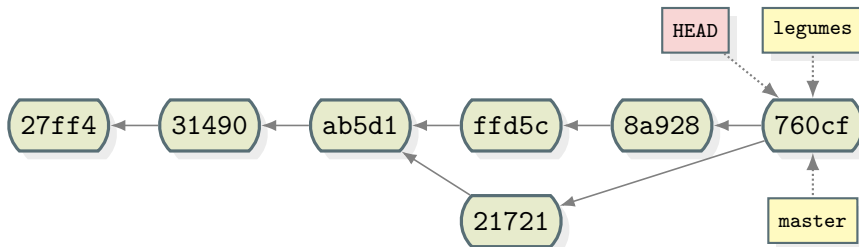
(a) Allons sur master

git checkout master

↩

21/30

## Le Graphe : Merge legumes⇒master



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

```

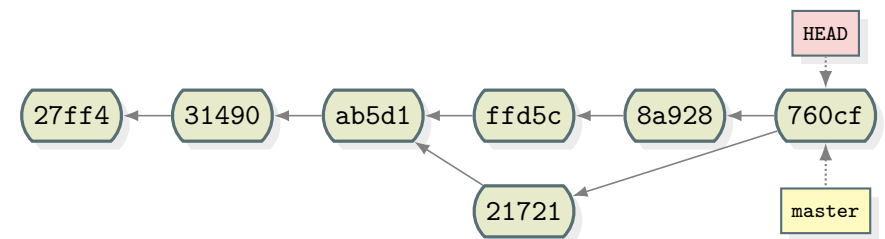
git checkout master
git diff legumes
git merge legumes

```

↩

21/30

## Le Graphe : Merge legumes⇒master



(a) Effacer la branche légumes

```

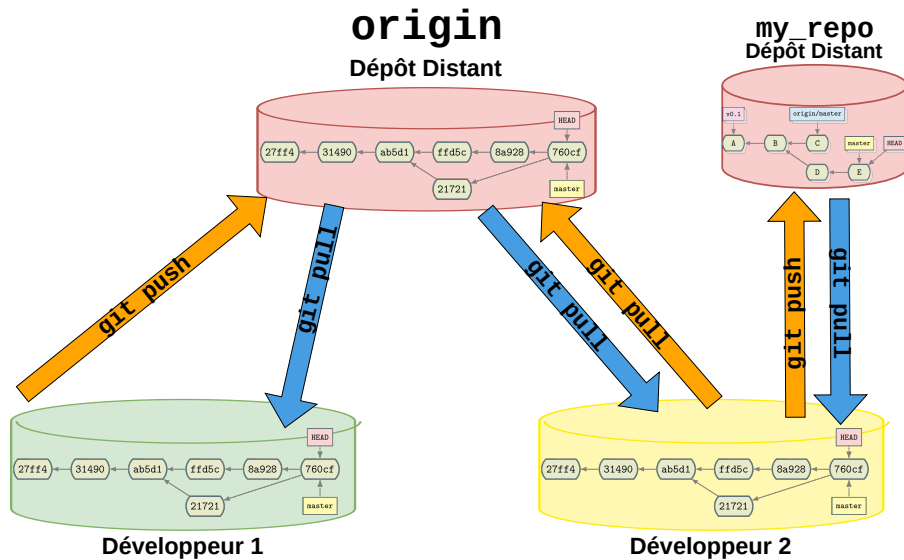
git checkout master
git diff legumes
git merge legumes
git branch -d legumes

```

↩

21/30

## Partager : dépôts distants



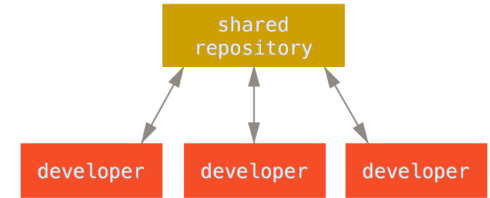
22/30

## 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`



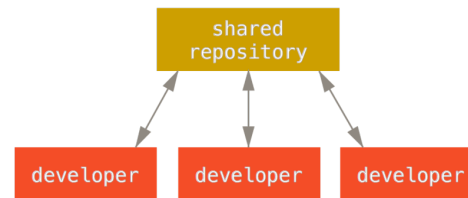
23/30

## 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

23/30

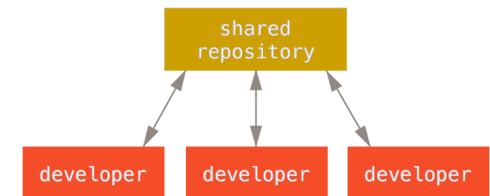
## Dépôt Centralisée : travail

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
```



24/30

## Résolution de conflits

Des conflits vont se produire ...

... comment faire pour les résoudre ?

25/30

## 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"
```

26/30

## 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
```

### Sorties console

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

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

26/30

## 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.

27/30

## 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
```

28 / 30

## 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
```

28 / 30

## 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)

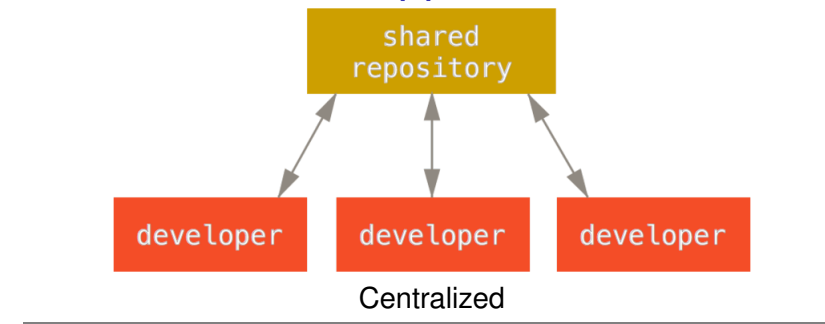
```
1 pomme
2 banane
3 ananas
4 kaki
5 poire
```

### 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
```

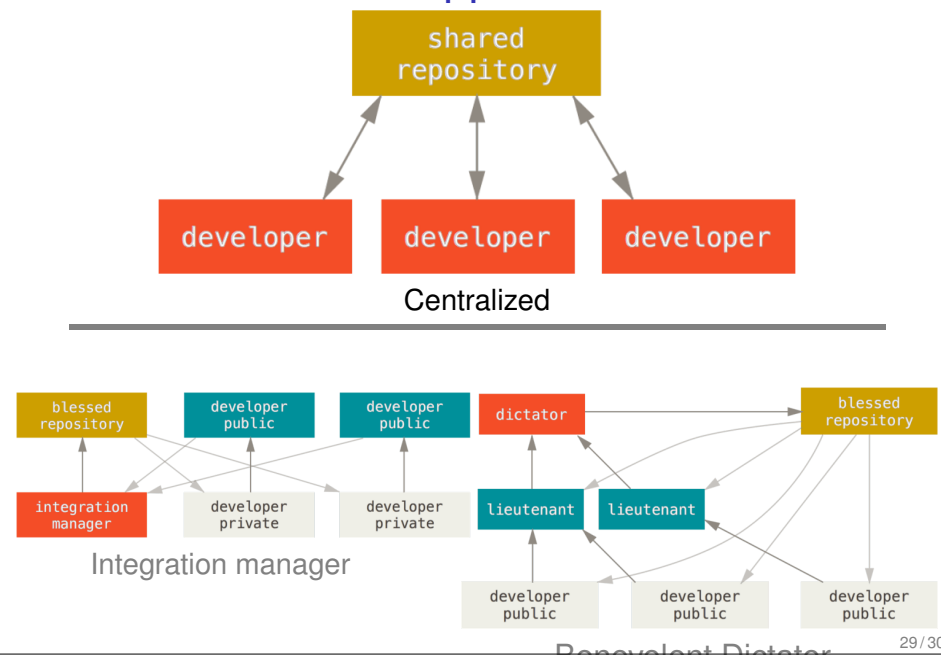
28 / 30

## Git distribué : Développements distribués



29 / 30

## Git distribué : Développements distribués



29/30

## Liens, aides et outils

- ▶ Où stocker vos projets
  - ▶ <https://archives.plil.fr/>
  - ▶ <https://github.com/>
  - ▶ <https://bitbucket.org/>
  - ▶ Votre serveur perso
- ▶ Tutoriels
  - ▶ <http://www.cristal.univ-lille.fr/TPGIT/>
  - ▶ <https://crypto.stanford.edu/~blynn/gitmagic/intl/fr/book.pdf>
  - ▶ <https://learngitbranching.js.org/>
  - ▶ <https://try.github.io/>
  - ▶ <https://git-scm.com/book/fr/v2>
- ▶ Vidéos
  - ▶ <https://www.youtube.com/watch?v=OqmSzXDrJBk>
  - ▶ [https://www.youtube.com/watch?v=uR6G2v\\_WsRA](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>

30/30