

Gestion de versions

avec git

Walter Rudametkin

Adaptation
M.E. Kessaci, O. Caron, J. Dequidt

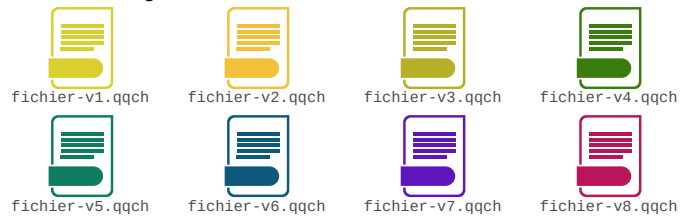
Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

Bureau F011
© Polytech Lille

1 / 38

Comment gérez-vous vos fichiers ?

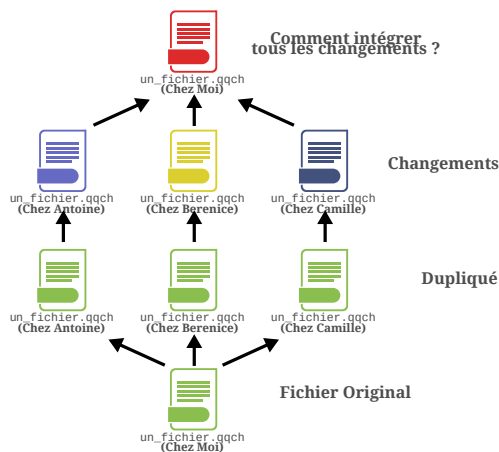
- Garder l'historique
- Partager



Versionnement manuel de fichiers

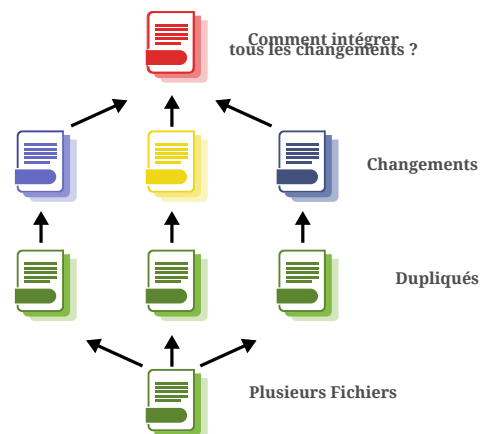
2 / 38

Comment collaborer sur un fichier ?



3 / 38

Comment collaborer sur plusieurs fichiers ?



4 / 38

D'autres solutions ?



5 / 38

Problématique : développement logiciel

- Un **projet** de développement logiciel est une activité longue et complexe.
- Concerne plusieurs **fichiers** (milliers !)
- De multiples **itérations** sont nécessaires.
- A certains moments, on peut identifier des **versions** et/ou **variantes** du logiciel.
- Les erreurs sont possibles, **revenir en arrière** est parfois nécessaire.
- Un projet peut se faire à plusieurs, les développeurs peuvent travailler sur les mêmes fichiers (**conflits**)

6 / 38

Définitions

Simple

- Un **gestionnaire de versions** est un logiciel qui **enregistre les évolutions d'un ensemble de fichiers** au cours du temps de manière à ce qu'on puisse rappeler une version antérieure à tout moment.

Définition Wikipedia¹

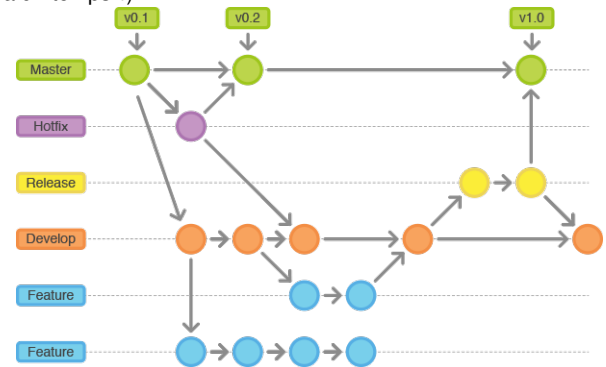
- La **gestion de versions** (en anglais *version control* ou *revision control*) consiste à maintenir l'**ensemble des versions d'un ou plusieurs fichiers** (généralement en texte). Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout **la gestion des codes source**.

¹https://fr.wikipedia.org/wiki/Gestion_de_versions

7/38

Gestion de versions

Le développement logiciel est un processus sinueux à notion de **branche** (chaque noeud représente un **ensemble de fichiers** à un temps t) :



8/38

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)

9/38

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 ...)

10/38

Why the git ?

C'est Ze Standard

- *git - the stupid content tracker*
- Linus Torvalds (2005)
- Outil professionnel, rapide, multi-plateforme, flexible, puissant, complètement distribué

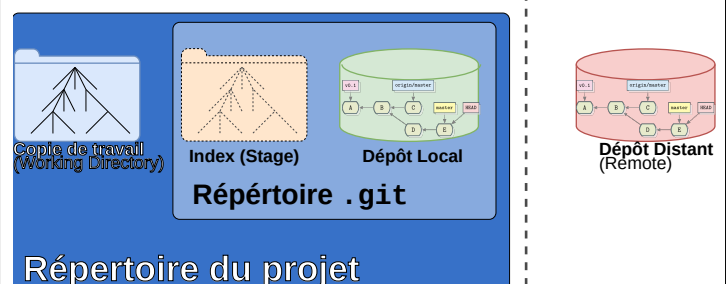
To Share or Not to Share ?

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

11/38

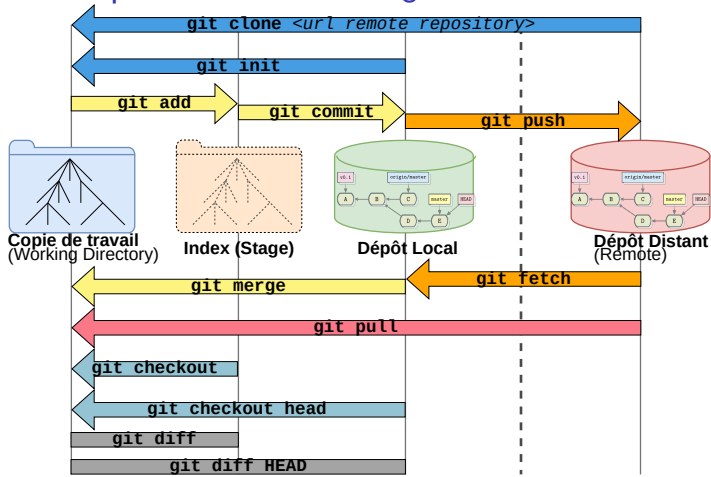
Concepts et commandes git

Réseau



12/38

Concepts et commandes git



12/38

Le Graphe Orienté Acyclique de commits

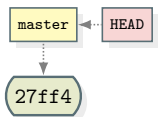
(a) Dépôt vide

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
```

13/38

Le Graphe Orienté Acyclique de commits



(a) Premier commit

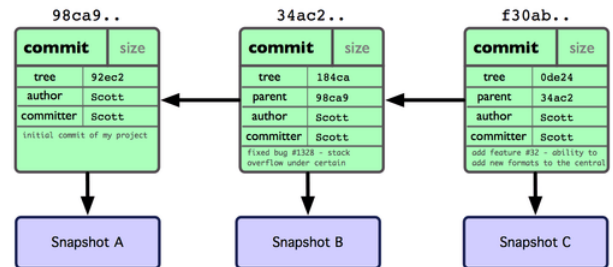
Dans un terminal ...

```
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 toute commande!

13/38

C'est quoi un commit ?

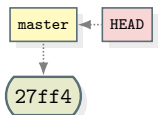


- Le Commit-ID est une *empreinte* calculé en utilisant la fonction de hachage SHA-1 sur
 - Tout le contenu du commit + Date + Nom et email du commiteur + Message de log + ID du commit parent + ...

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

14/38

Le Graphe : Commit 2



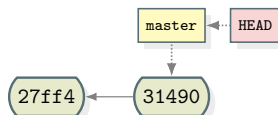
(a) État avant deuxième commit

Dans un terminal ...

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

15/38

Le Graphe : Commit 2



(a) Deuxième commit

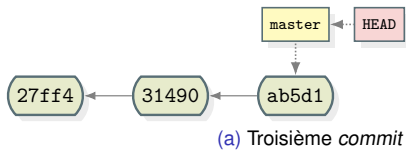
Dans un terminal ...

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

↪

15/38

Le Graphe : Commit 3



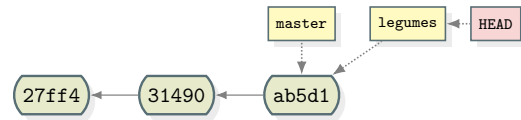
Dans un terminal ...

```

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

16/38

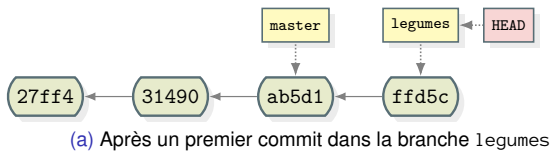
Le Graphe : Branche legumes



(a) Après branche
 ⇒ une nouvelle *étiquette* (legumes) apparaît, elle pointe vers le commit courant (ab5d1), et la commande checkout fait pointer HEAD sur legumes
 git branch legumes ; git checkout legumes

17/38

Le Graphe : Branche legumes

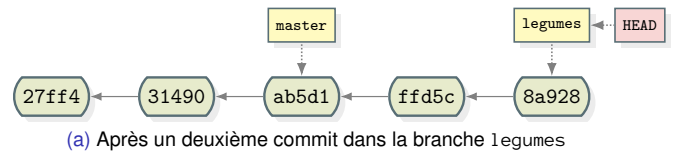


```

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

17/38

Le Graphe : Branche legumes

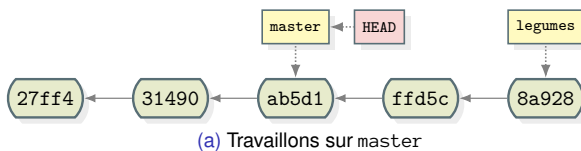


```

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
  
```

17/38

Le Graphe : Branche master



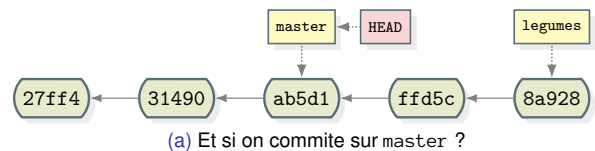
⇒ legumes.txt n'existe plus dans la Copie de Travail (Working Directory)

```

...
git checkout master
  
```

18/38

Le Graphe : Branche master

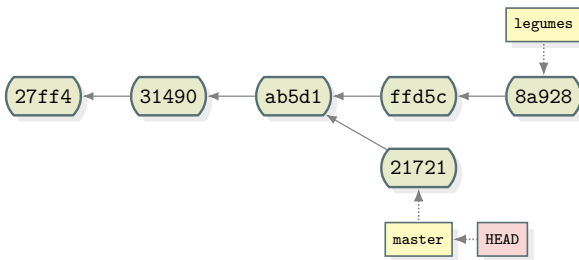


```

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

18/38

Le Graphe : Branche master

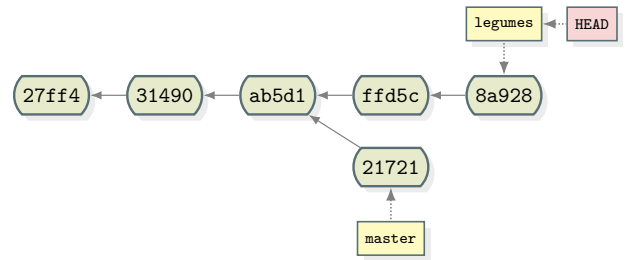


(a) Après un nouveau commit sur master

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

18/38

Le Graphe : Merge master ⇒ légumes

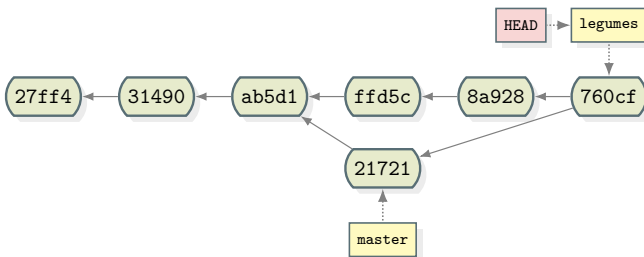


(a) Allons sur légumes, regardons les différences avec diff

```
git checkout legumes
git diff master
↪ git merge master
```

19/38

Le Graphe : Merge master ⇒ légumes



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

```
git checkout legumes
git diff master
git merge master
↪
```

19/38

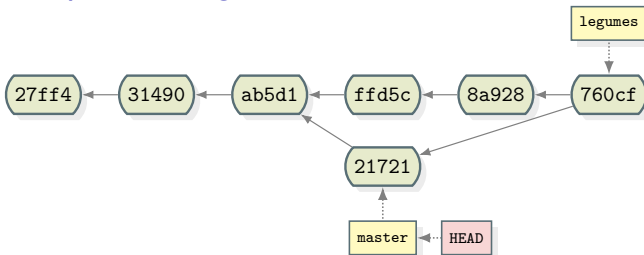
Merge : Vue dans la console

```
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]
* 1888839 [2017-12-01] Ajout courgette à légumes [rudametw]
* ffd5c3e [2017-12-01] Ajout de légumes [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

20/38

Le Graphe : Merge légumes ⇒ master

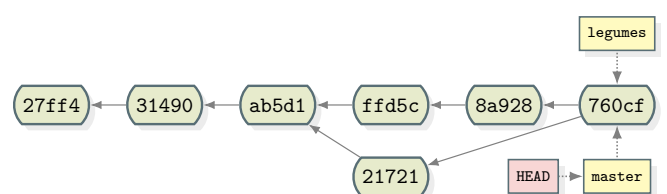


(a) Allons sur master

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

21/38

Le Graphe : Merge légumes ⇒ master

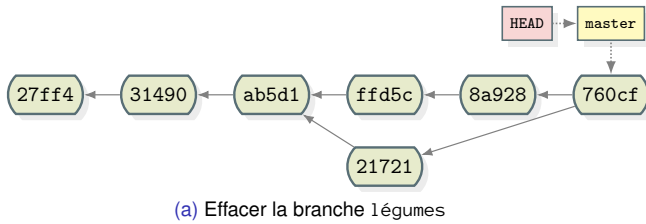


(a) Merger légumes dans master : pas de nouveau commit

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

21/38

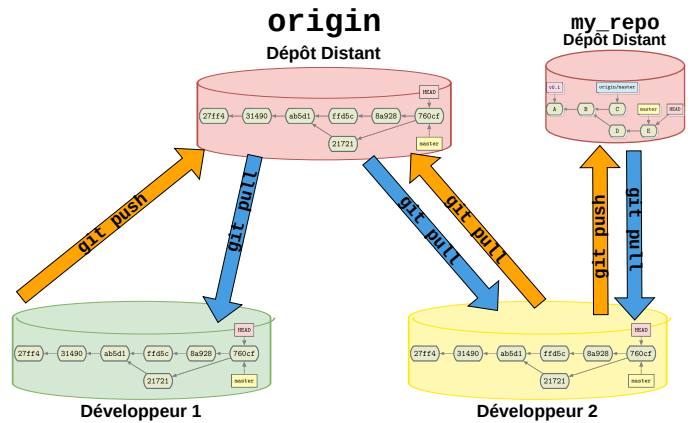
Le Graphe : Merge légumes⇒master



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

21/38

Partager : dépôts distants

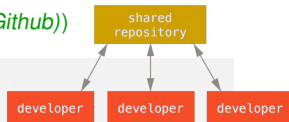


22/38

Dépôt Centralisée : initialisation

Créer le dépôt
(le dépôt distant doit exister (ici c'est chez Github))

```
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
```

23/38

Dépôt Centralisée : méthode de travail idéal

Chacun et chaque fonctionnalité sur sa branche.
Une fois la fonctionnalité fini, on merge dans master.

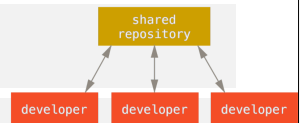
```
git pull //update & check work
git branch fonctionnalitéX
git checkout fonctionnalitéX
```

while (je travaille = vrai)

```
git diff
git add <fichiers>
git commit -m "message"
```

```
git pull --all
git merge master //gérer conflits et TESTER !
```

```
//intégrer votre travail
git checkout master
git merge fonctionnalitéX
git pull ; git push
```



24/38

Dépôt Centralisée : méthode de travail idéal

En pratique, vérifier l'état de votre dépôt
cooonstaaaaaament !!!

```
git status ; git pull ; git status //update & check work
git branch fonctionnalitéX
git checkout fonctionnalitéX
```

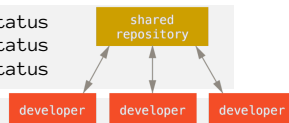
while (je travaille = vrai)

```
git diff ; git status
git add <fichiers> ; git status
git commit -m "message" ; git status
```

```
git pull --all ; git status
git merge master //gérer conflits et TESTER !
git status
```

//intégrer votre travail

```
git checkout master ; git status
git merge fonctionnalitéX ; git status
git pull ; git push ; git status
```

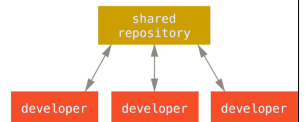


25/38

Dépôt Centralisée : méthode de travail simple

Sans branches. Commitez souvent.

```
git status
git pull //mise à jour du dépôt local
git add <fichiers>
git commit -m "message"
git pull --all
git status
git push //mise à jour du dépôt distant
git status
```



26/38

Résolution de conflits

Des conflits vont se produire ...

... comment faire pour les résoudre ?

27/38

Provoquer un conflit dans fruits.txt

Branche ananas

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

Branche kaki

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

Branche ananas

fruits.txt :

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

Branche kaki

fruits.txt :

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

28/38

Merger un conflit dans fruits.txt

Branche ananas

fruits.txt :

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

Branche kaki

fruits.txt :

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

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.
```

29/38

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.

30/38

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)

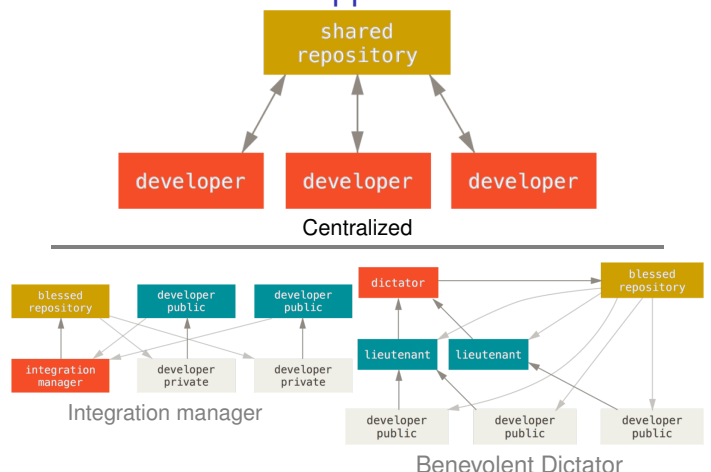
```
1 pomme
2 banane
3 ananas
4 kaki
5 poire
```

Résolution du conflit (sur terminal)

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

31/38

Git distribué : Développements distribués



32/38

Premiers pas : configuration de git

```
git config --global user.name "votre nom"
git config --global user.email nom.prenom@polytech-lille.net
git config --global core.editor 'kate -b'
git config --global push.default simple
git config --global color.decorate full
git config --global merge.conflictstyle diff3
```

- ▶ À faire une seule fois: informations stockées dans `~/.gitconfig`
- ▶ Choix de l'éditeur : kate, gedit, emacs, vim, ...
- ▶ Disposez d'un prompt adapté :
`source ~wrudamet/public/bashrc-students`
à ajouter dans votre `~/.bashrc`

33/38

Quelques astuces (1/2)

- ▶ Joli log avec graphe
`git log --graph --oneline --decorate --all`
- ▶ Annuler un merge en cas de conflit
`git merge --abort`
- ▶ Sauvegarder votre mot de passe (accès https, 1h)
`git config --global credential.helper cache --timeout=3600`
- ▶ Corriger origin ou faire du multi-dépôt

```
# Après un clone ...
git clone git@archives.plil.fr:jdequidt/ima3_projet_pa_2018.git
# ... on peut ajouter, renommer ou effacer les remotes
git remote rename sujet-dequidt
git remote add origin
↪ https://archives.plil.fr/rudametw/ima3_projet_pa_2018.git
git remote add depot-ssh git@github.com:rudametw/projet_ima3.git
git remote -v #listes toutes les remotes
```

34/38

Quelques astuces (2/2)

- ▶ Pour ne pas commiter des fichiers générés, créez le fichier `.gitignore` à la racine du projet

```
#Exemple de .gitignore
*~
*.o
a.out
build/
bin/
```

- ▶ Écrire la documentation en Markdown
 - ▶ Syntaxe simple, propre, comme Wikipédia
 - ▶ README.md automatiquement converti en HTML
 - ▶ Permet de créer tous types de document, très puissant si combiné avec pandoc
 - ▶ Inspirez vous de <https://gist.github.com/PurpleBooth/109311bb0361f32d87a2>

35/38

Conclusion

- ▶ Ce cours est une [introduction](#) de git
- ▶ Gestionnaire de versions, élément [incontournable](#) du développeur ou équipe de développeurs
- ▶ git : outil performant et [massivement utilisé](#)
- ▶ git : spécialisé pour le texte et la ligne de commande mais de nombreuses extensions et outils graphiques
 - ▶ gitk, smartgit, tortoise (windows), EGit pour environnement Eclipse, ...

36/38

Liens, aides et outils (1/2)

- ▶ Références 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>
 - ▶ Présentation "Les bases de GIT" <https://fr.slideshare.net/PierreSudron/diapo-git>
- ▶ Où stocker vos projets
 - ▶ <https://gitlab.univ-lille.fr/>
 - ▶ <https://archives.plil.fr/> ← Polytech
 - ▶ <https://gitlab.com/>
 - ▶ <https://github.com/>
 - ▶ <https://bitbucket.org/>
 - ▶ Votre serveur perso

37/38

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>

38/38