



git

- Logiciel de Gestion de versions
 - Garder une trace des modifications
 - Se déplacer dans le temps
 - Travailler à plusieurs sur un même projet
 - Tester des solutions



Concurrents

2

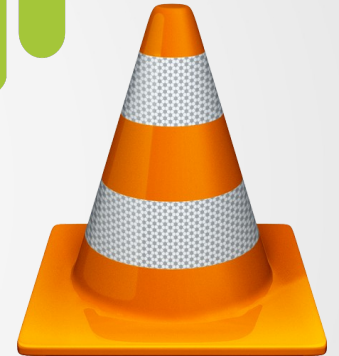
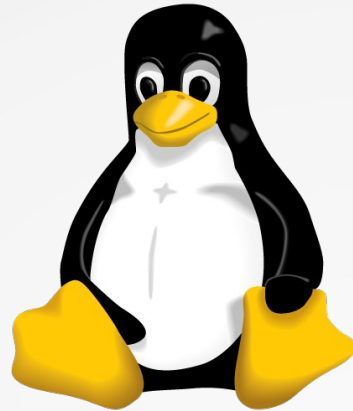
- CVS (pourri, mort et oublié)
- Clearcase (Payant, vieux, mais utilisé)
- Mercurial (Payant)
- SVN (Gratuit, opensource)



Utilisations

3

- Linux Kernel
- Android
- GNOME
- PhpMyAdmin
- Qt, GTK+
- VLC
- Wine
- ...



Prérequis

4

- Bases en shell Linux
- Installation de Git
 - Ubuntu : `sudo apt-get install git`
 - CentOS : `sudo yum install git`
 - Arch : `sudo pacman -S git`
 - Windaube : <http://git-scm.com/download/win>
- Configuration de Git :
 - `git config --global user.email "foo.bar@isen.fr"`
 - `git config --global user.name "Foo BAR"`
- Éventuellement installation de ZSH et OhMyZSH!

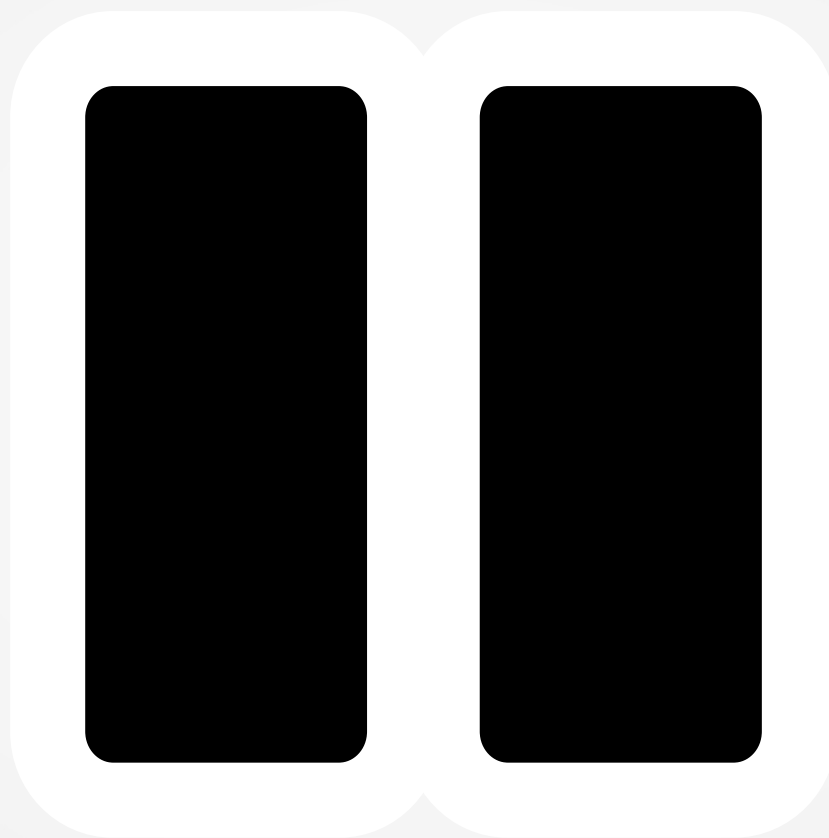


Plan du cours

5

- Utilisation basique de Git
- Utilisation basique de Git, avec plusieurs utilisateurs
- Git avec un serveur central hébergé
- Codez un "Jeu de la Vie" en 5min !
- Gestion des conflits
- Branches, branches et branches partout
- Le Stash
- Les git hooks : comment mettre à jour son site web via Git

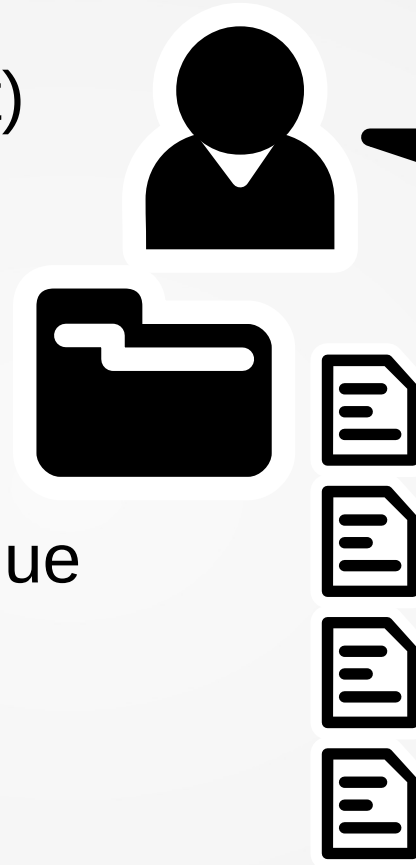




Utilisation basique de Git

7

- Créer un repository (projet)
- Ajouter des fichiers
- Modifier des fichiers
- Logger les modifications
- Se déplacer dans l'historique
- Le .gitignore



Je suis tout seul ;(
mais je peux faire
ce que je veux !



Création du projet

8

- Création du dossier du projet :

```
mkdir TestGit
```

```
cd TestGit
```

- Initialisation du repository :

```
git init
```

```
Initialized empty Git repository in [...]/TestGit/.git/
```

```
ls -la
```

Un dossier `.git` est créé. Il contient toutes les données nécessaires au fonctionnement de Git sur ce projet



Voir les modifications (et ajouts)

9

- Copiez/collez quelques fichiers bidons dans le repository
- Lister les modifications faites sur le repository :

```
git status
```

```
# On branch master
# Initial commit
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   Async.cbp
#   Async.cpp
#   Async.hpp
#   README
nothing added to commit but untracked files present (use "git add" to
track)
```

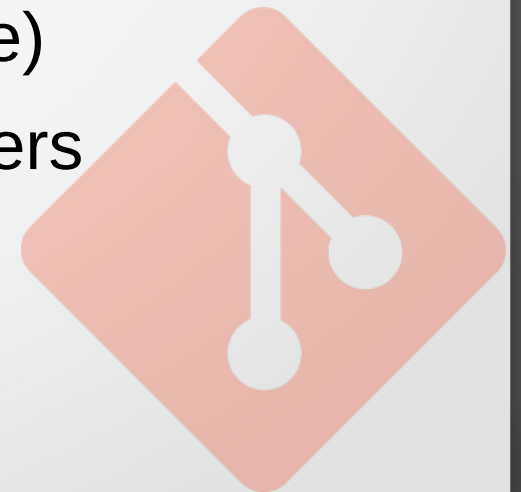
- Conseil : Ajoutez un fichier README, contenant un court résumé du projet



Faisons un Commit !

10

- On va maintenant faire un Commit
 - C'est un point d'avancement du projet
 - Il contient la liste de fichiers modifiés
 - Il contient un commentaire
- Pour faire un commit :
 - On sélectionne une liste de fichiers (Stage)
 - On écrit le commit qui contiendra les fichiers



Sélection des fichiers (Stage)

11

- Ajout de tous les fichiers récemment ajoutés

```
git add --all
```

Note : --all sélectionne tous les fichiers modifiés. On peut aussi écrire le nom de chaque fichier

- On vérifie que les fichiers ont été sélectionnés :

```
git status
```

```
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   Async.cbp
#   new file:   Async.cpp
#   new file:   Async.hpp
#   new file:   README
```

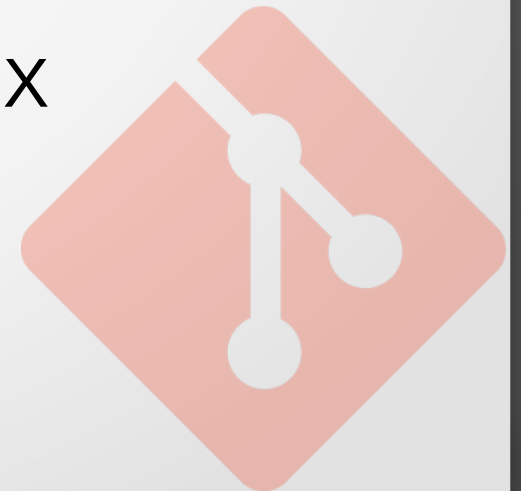


Le commit !

12

```
git commit
```

- Ouvre l'éditeur de texte en console par défaut
- Écrivez un commentaire correspondant aux modifications effectuées
 - Ici, ça serait « First Commit »
- Validez en enregistrant les modifications
 - Ubuntu (nano) : CTRL+O, Entrée, CTRL+X
 - Autres (vi) : ECHAP, :wq, Entrée



On regarde ce qui c'est passé

13

```
git status
```

```
# On branch master  
nothing to commit, working directory clean
```

- Les modifications ont été commitées, elle n'apparaissent plus.

- Affichage des commits :

```
git log
```

```
commit c2f2ee2aa1c77ccb11379165b1d342ae44799ef5  
Author: Crom (Thibaut CHARLES) <thibaut.charles@isen.fr>  
Date: Wed Dec 18 21:53:34 2013 +0100  
First Commit !
```

Identifiant du commit
(commit hash)

Auteur

Date et heure

Commentaire

Allez on recommence !

14

- Cette fois on ne va pas ajouter des fichiers
On va les modifier !

Je vous laisse faire les modifications...

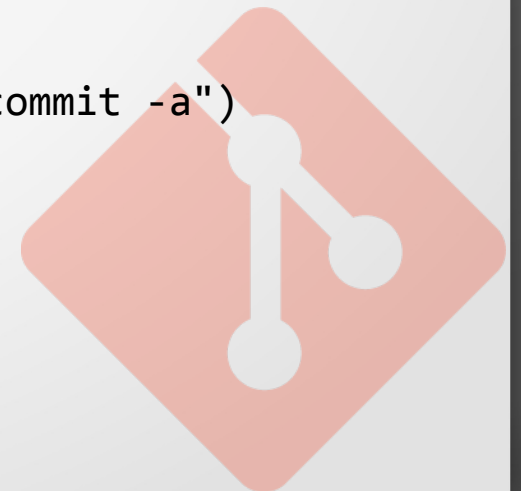


On regarde quels fichiers ont été modifiés

15

- J'ai modifié le fichier Async.hpp pour y ajouter un peu de documentation
- `git status`

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#   modified:   Async.hpp
#
no changes added to commit (use "git add" and/or "git commit -a")
```



Stagging des modifications

16

- Stage des modifications

```
git add Async.hpp
```

Note : Pour ajouter plusieurs fichiers, `git add fichier1 fichier2 ...`

`--all` permet d'ajouter tous les fichiers modifiés, on le fait rarement

- `git status`

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   Async.hpp
```



Commit

17

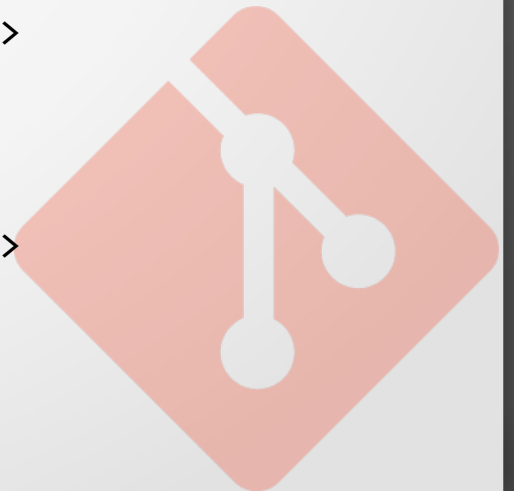
```
git commit
```

```
git status
```

```
# On branch master  
nothing to commit, working directory clean
```

```
git log
```

```
commit 377ad3d01268835a1649c9f7294401ba03888a40  
Author: Crom (Thibaut CHARLES) <thibaut.charles@isen.fr>  
Date:   Wed Dec 18 22:38:35 2013 +0100  
    Ajout de documentation sur GetFirstSlot  
  
commit c2f2ee2aa1c77ccb11379165b1d342ae44799ef5  
Author: Crom (Thibaut CHARLES) <thibaut.charles@isen.fr>  
Date:   Wed Dec 18 21:53:34 2013 +0100  
    First Commit !
```



- Voir ce qui a été modifié dans un commit

```
git show 377ad3d01268835a1649
```

```
diff --git a/Async.hpp b/Async.hpp
index 4cb10df..f15dfa2 100755
--- a/Async.hpp
+++ b/Async.hpp
@@ -101,6 +101,10 @@
     //=====
+    /**
+     * @brief returns the index of the first empty slot available
+     * for another callback
+     * @return index in m_Functions
+     */
     int GetFirstSlot();
```

Nom du fichier avant modification

Nom du fichier après modification

Lignes affichées/ajoutées

Note : les 4 premiers caractères du commit suffisent dans la plupart des cas



Se déplacer dans l'historique

19

- Pour se déplacer, `git status` doit renvoyer « working directory clean »
- Sinon il faut commiter les modifications ou les supprimer avec :
`git reset --hard HEAD`

Note : HEAD désigne le dernier commit visible dans les logs

- Déplacement à un commit donné
`git checkout c2f2ee2aa1c77ccb11`
- Vous pouvez regarder le code, mais pas faire de commit
(On verra plus tard comment faire, cf Branches)



Revenir au dernier commit

20

- Si des modifications ont été faites :
Nettoyage du repository :

```
git reset --hard HEAD
```

- Déplacement au dernier commit

```
git checkout master
```

Note : master est le nom de la branche sur laquelle on travaille, ça va nous ramener au dernier commit effectué sur le repository



Le .gitignore

21

- Il s'agit d'un fichier qui définit quels fichiers doivent être ignorés par Git
- Exemple : on a pas besoin de commiter les fichiers .o, .exe, ...
- On écrit normalement ce fichier au début du projet

```
touch .gitignore
```

```
gedit .gitignore
```



Exemple de .gitignore :

22

```
bin/  
obj/  
*.exe  
*.o  
project.layout
```



- On va créé des fichiers :

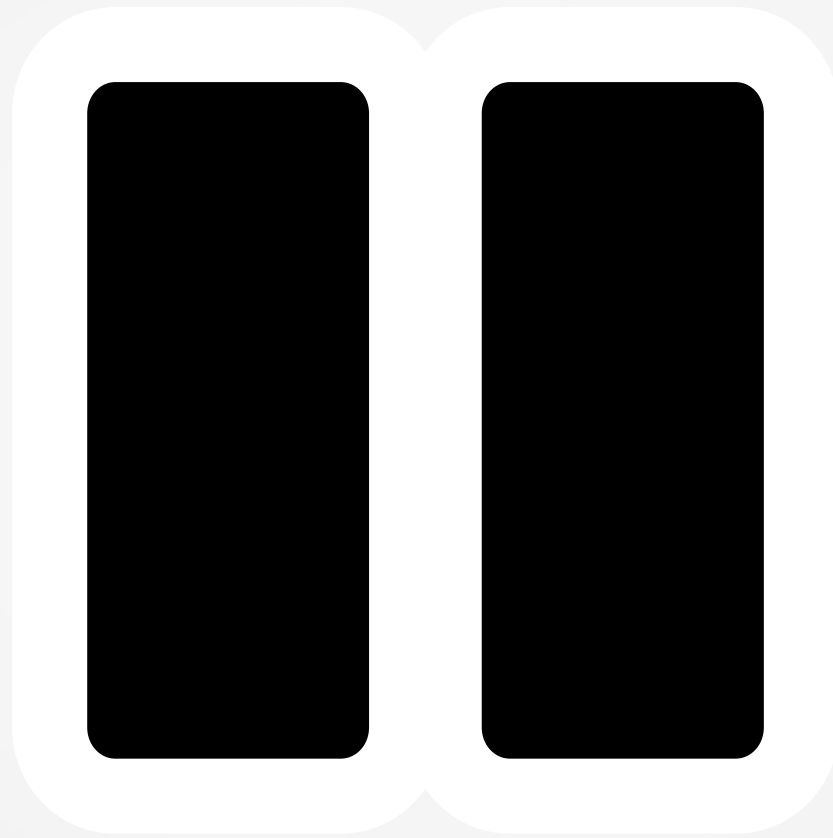
```
touch testgitignore.o  
mkdir bin/  
touch bin/testgitignore.c
```

- On regarde si ils apparaissent :

```
git status
```

Aucun des fichiers créés ne sont visibles

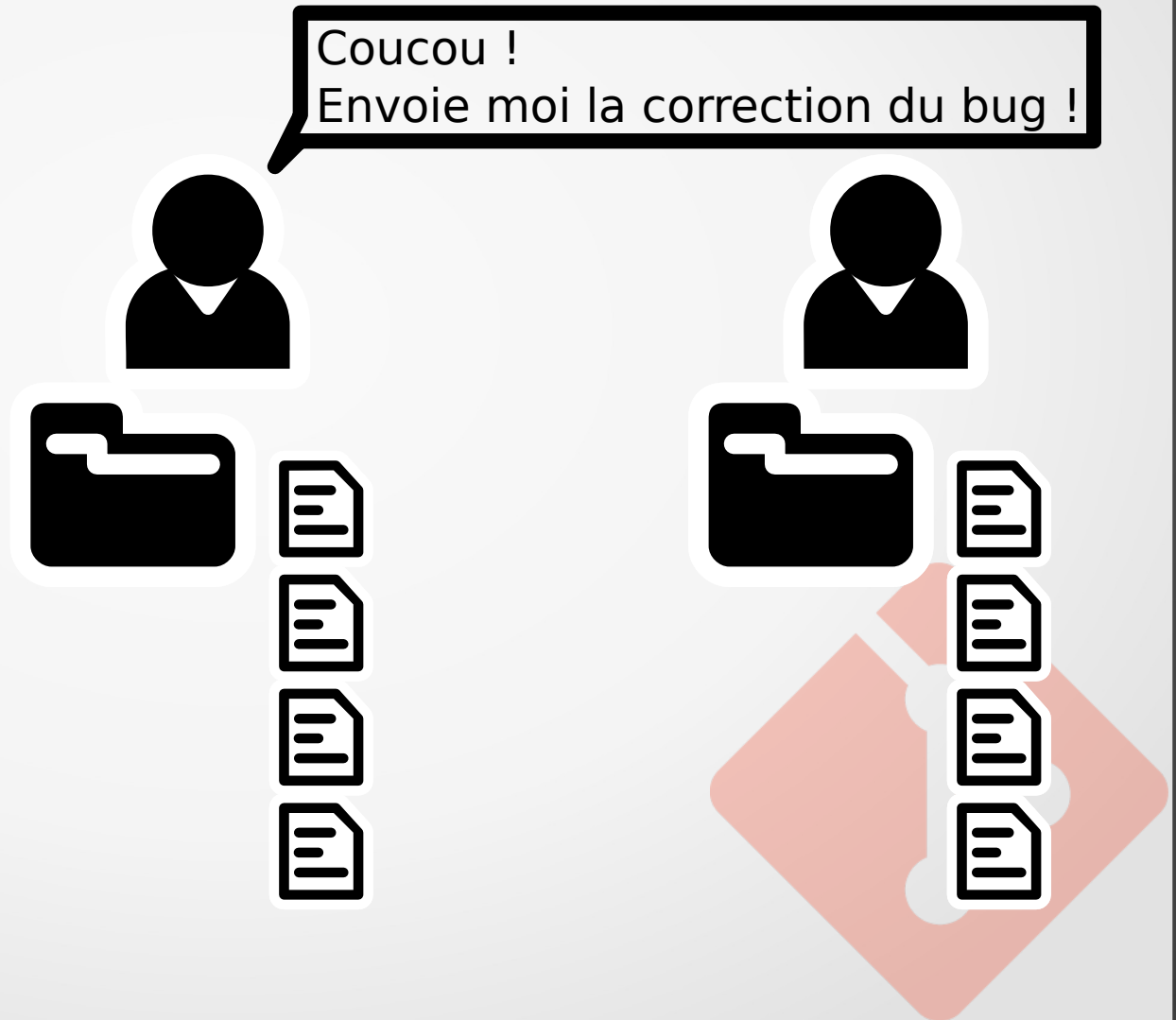




Avec plusieurs utilisateurs

25

- Structure Client-Serveur
- Clonage
- Remotes
- Push/Pull
- Merging

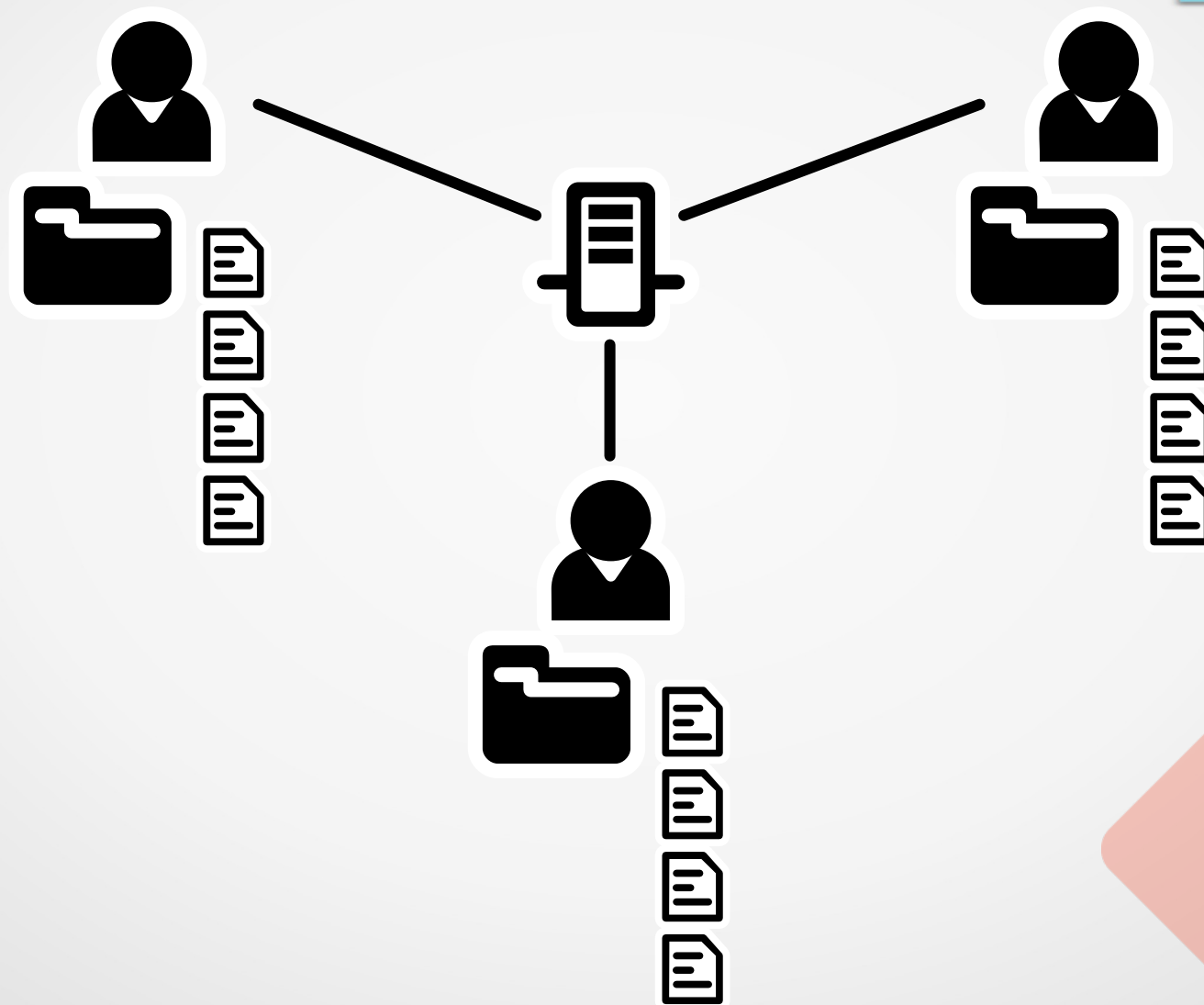


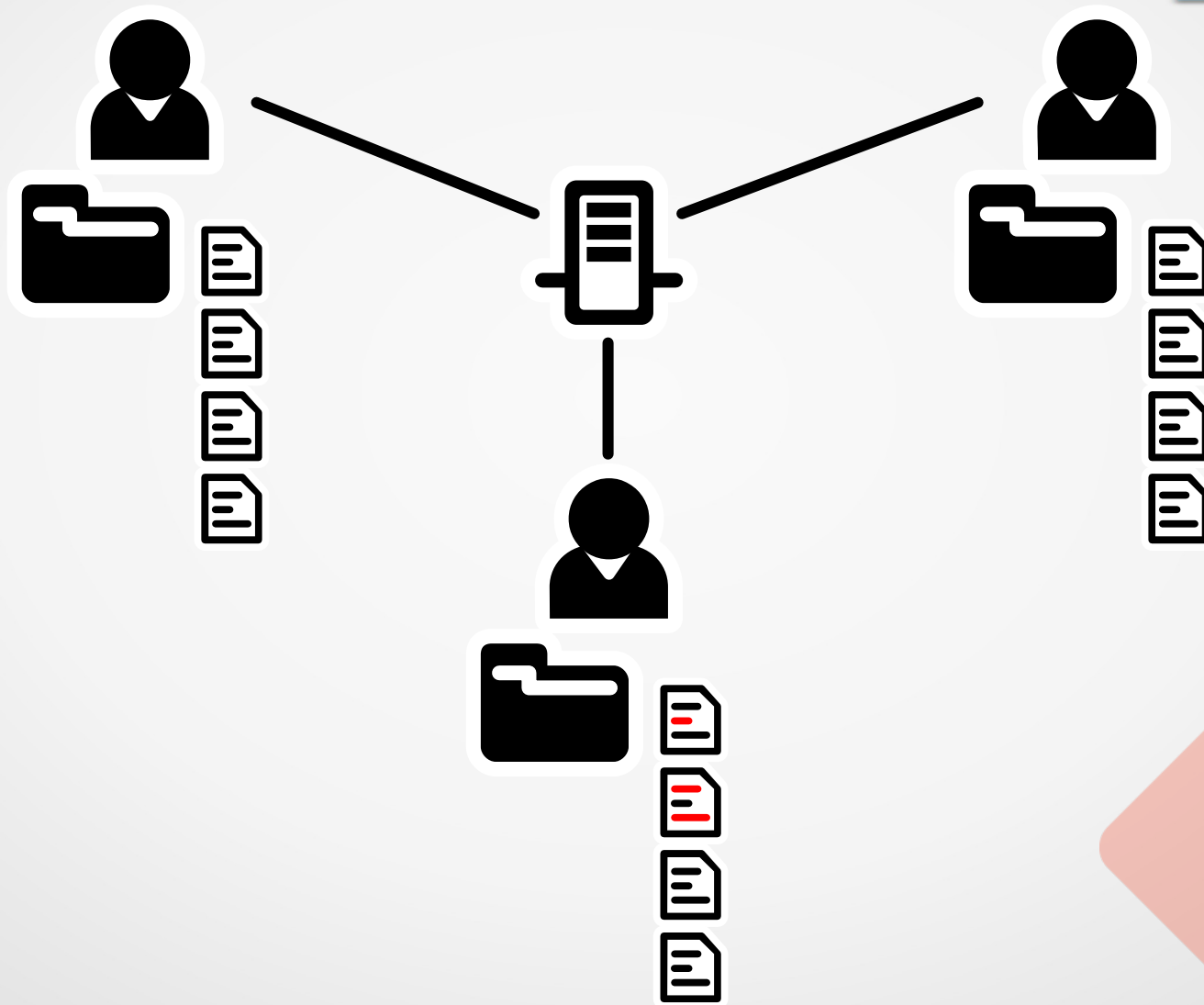
- Git est fait pour être utilisé en local, ainsi la structure Client-Serveur peut être réalisée avec des dossiers :
 - Un dossier « Serveur » ou bare repository créé avec `git init --bare`
 - Un ou plusieurs dossiers « Développeur »
- Il est tout de même pratique de faire héberger sur internet ses repositories git (Accès facile au repo, visibilité, ...)
On verra plus tard comment faire

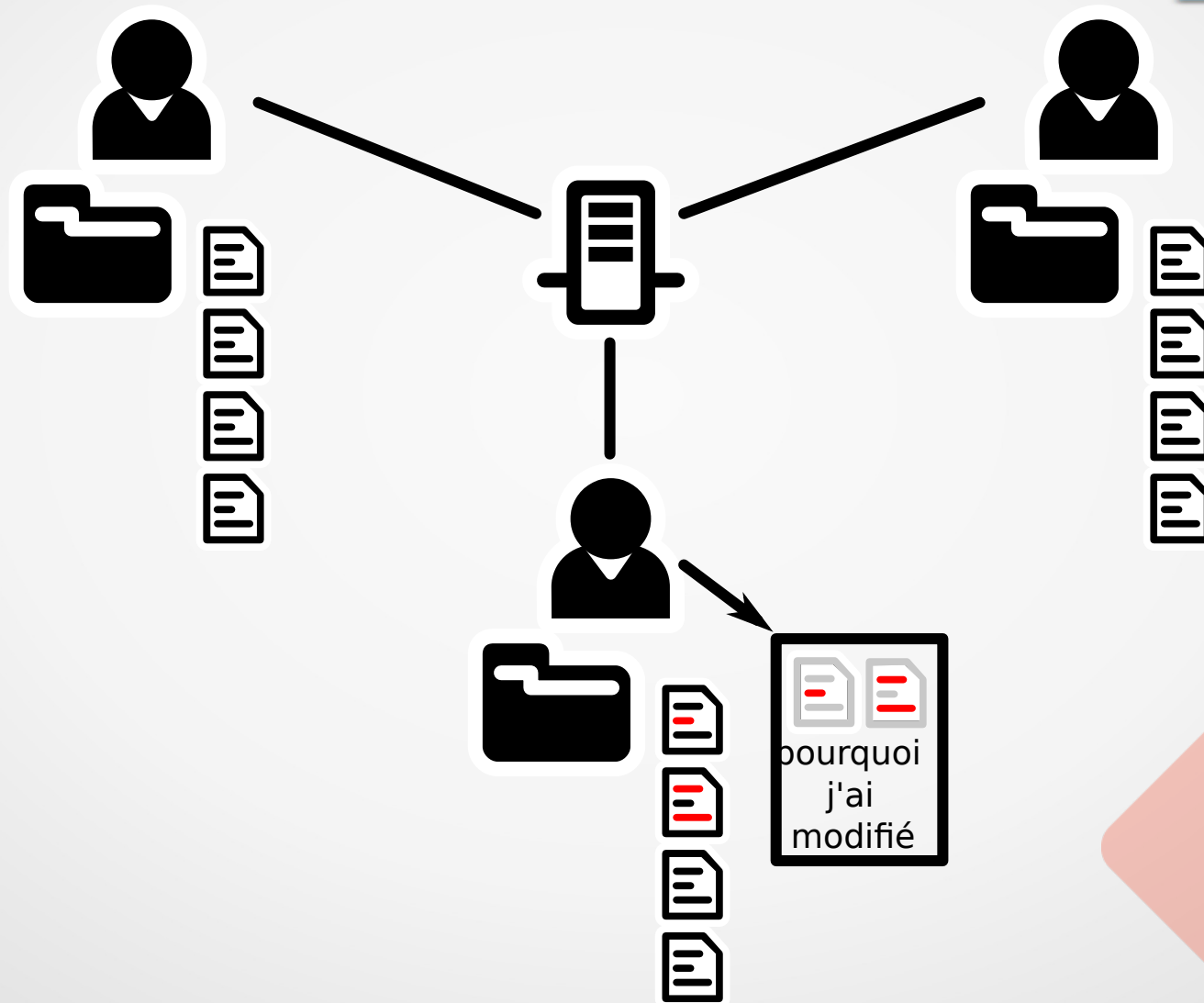


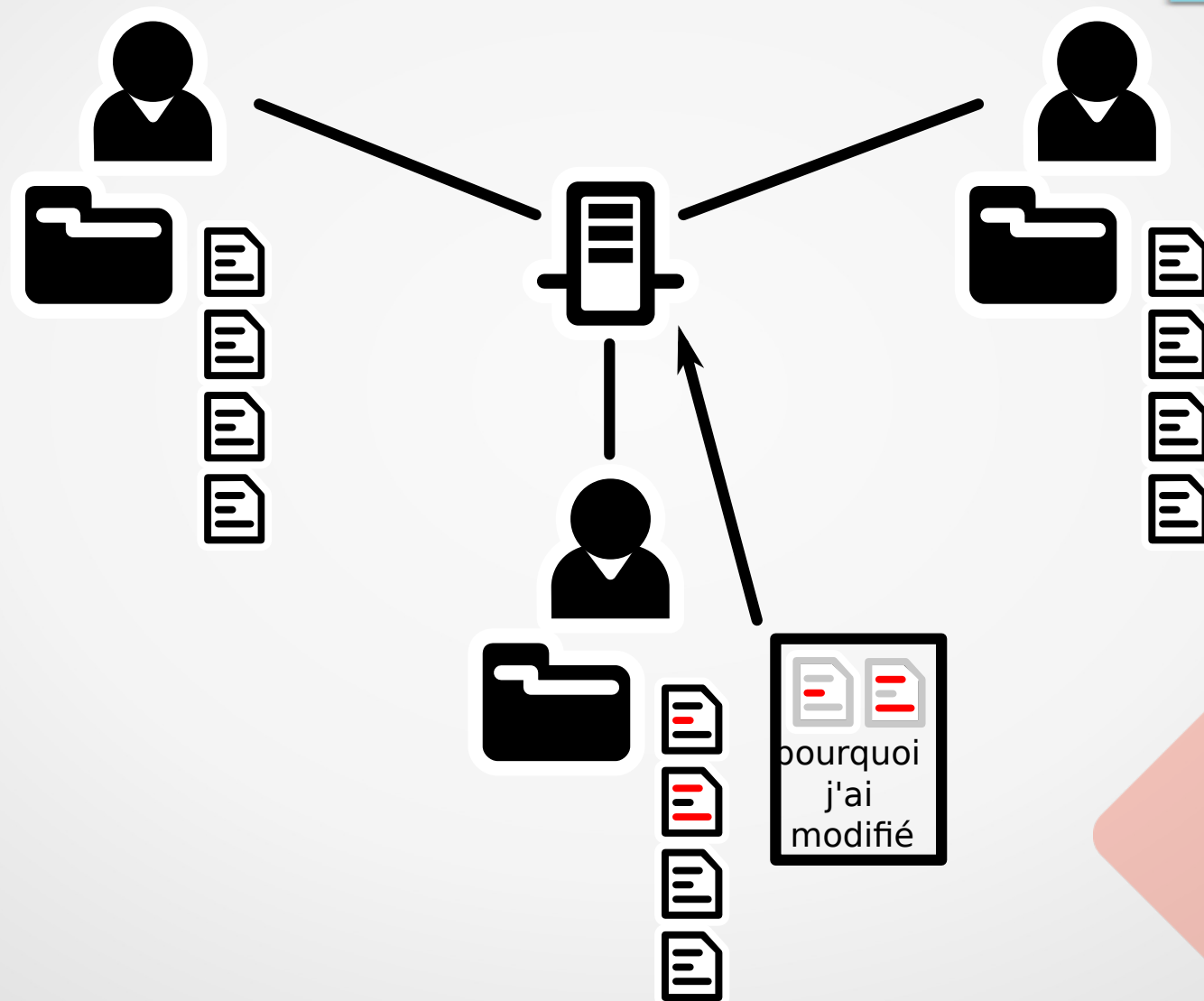
Fonctionnement Client-Serveur

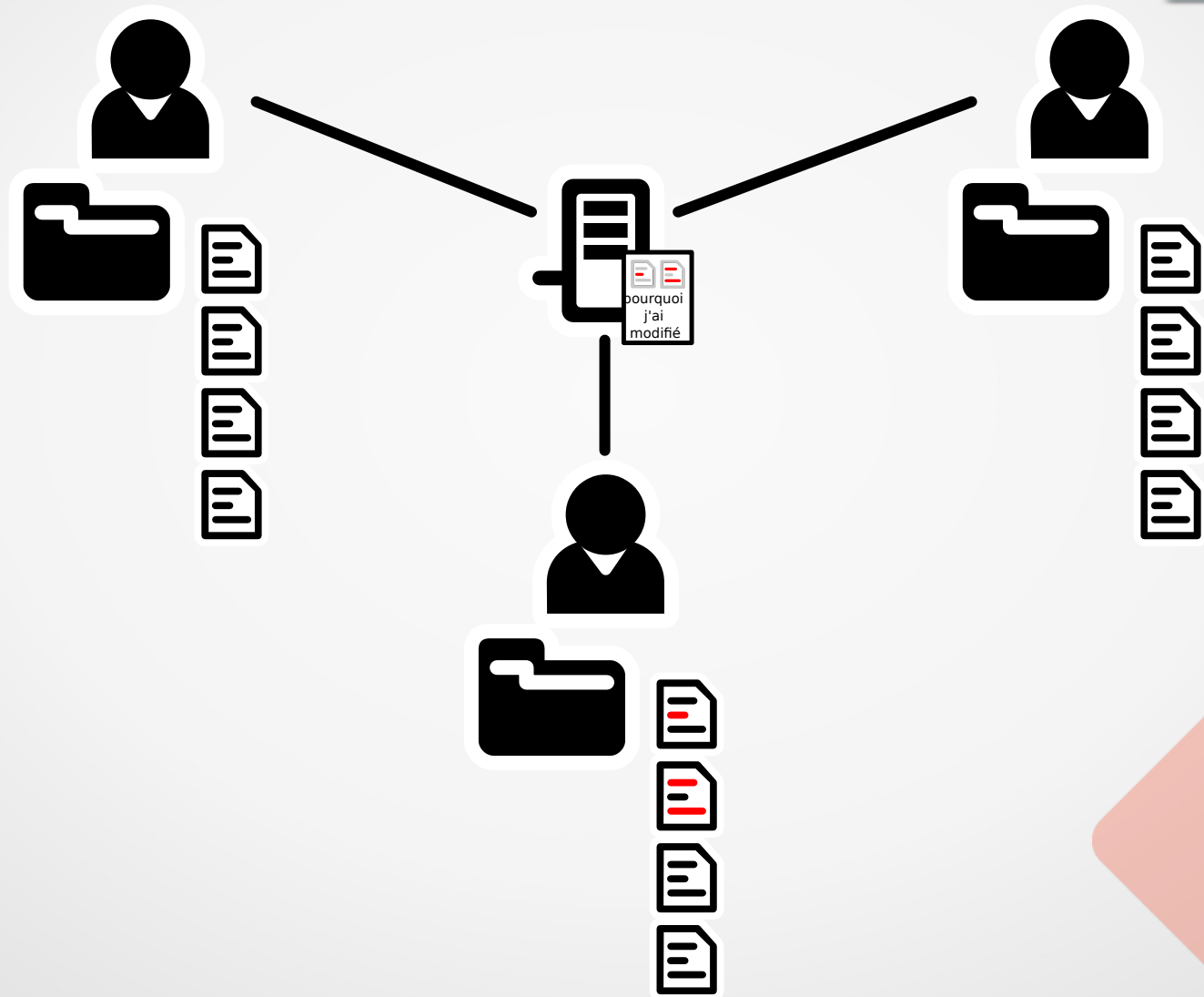
27

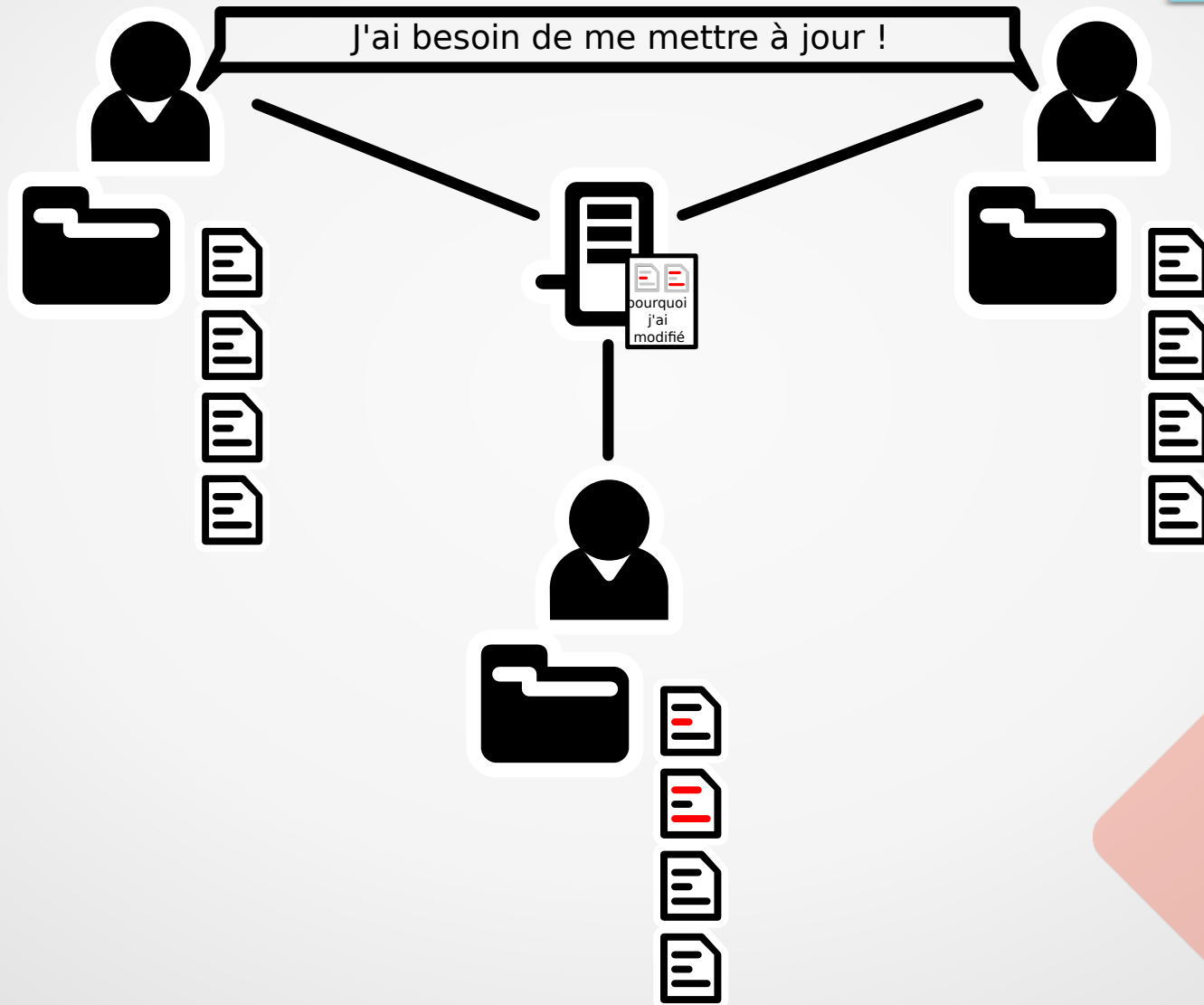


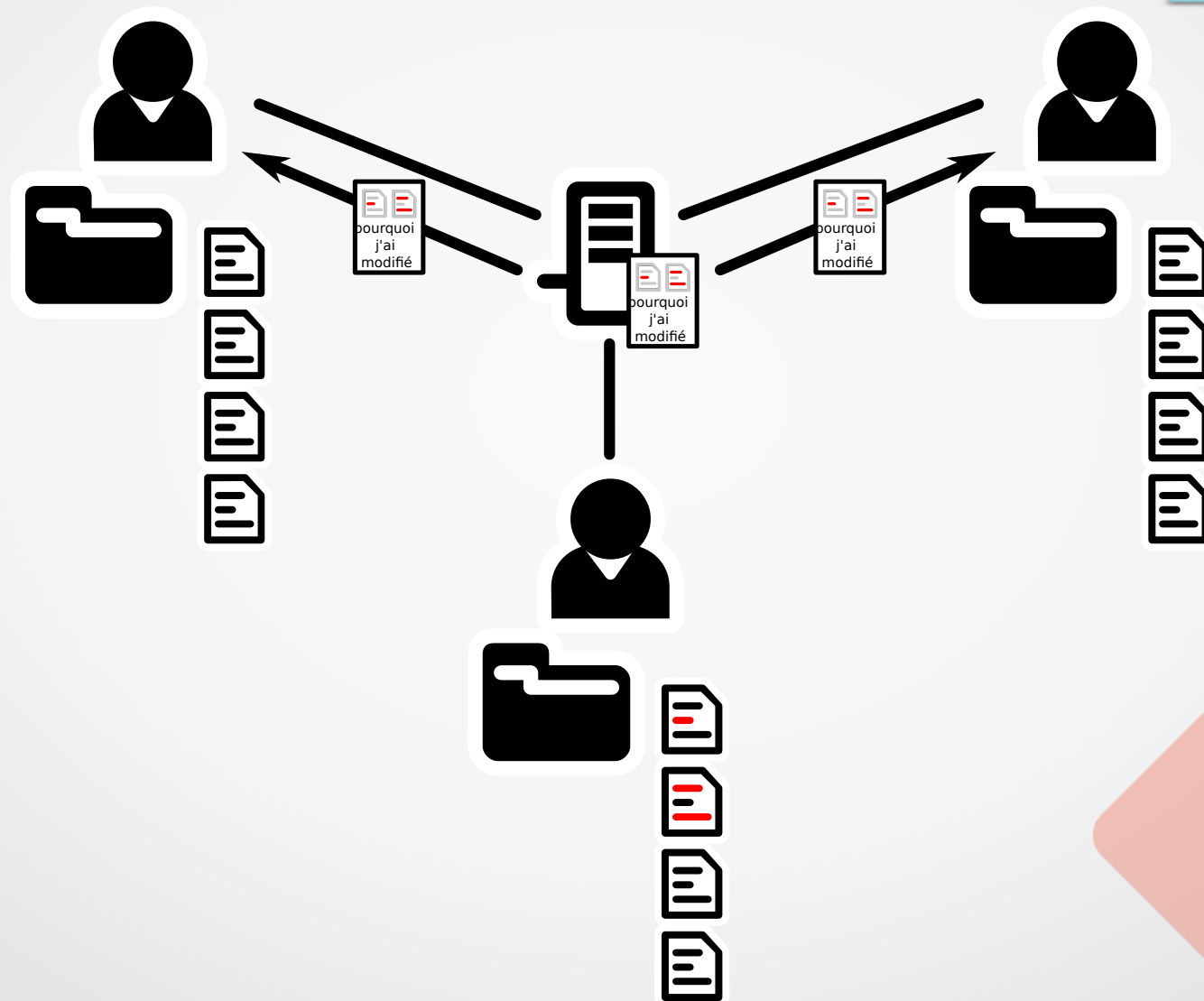






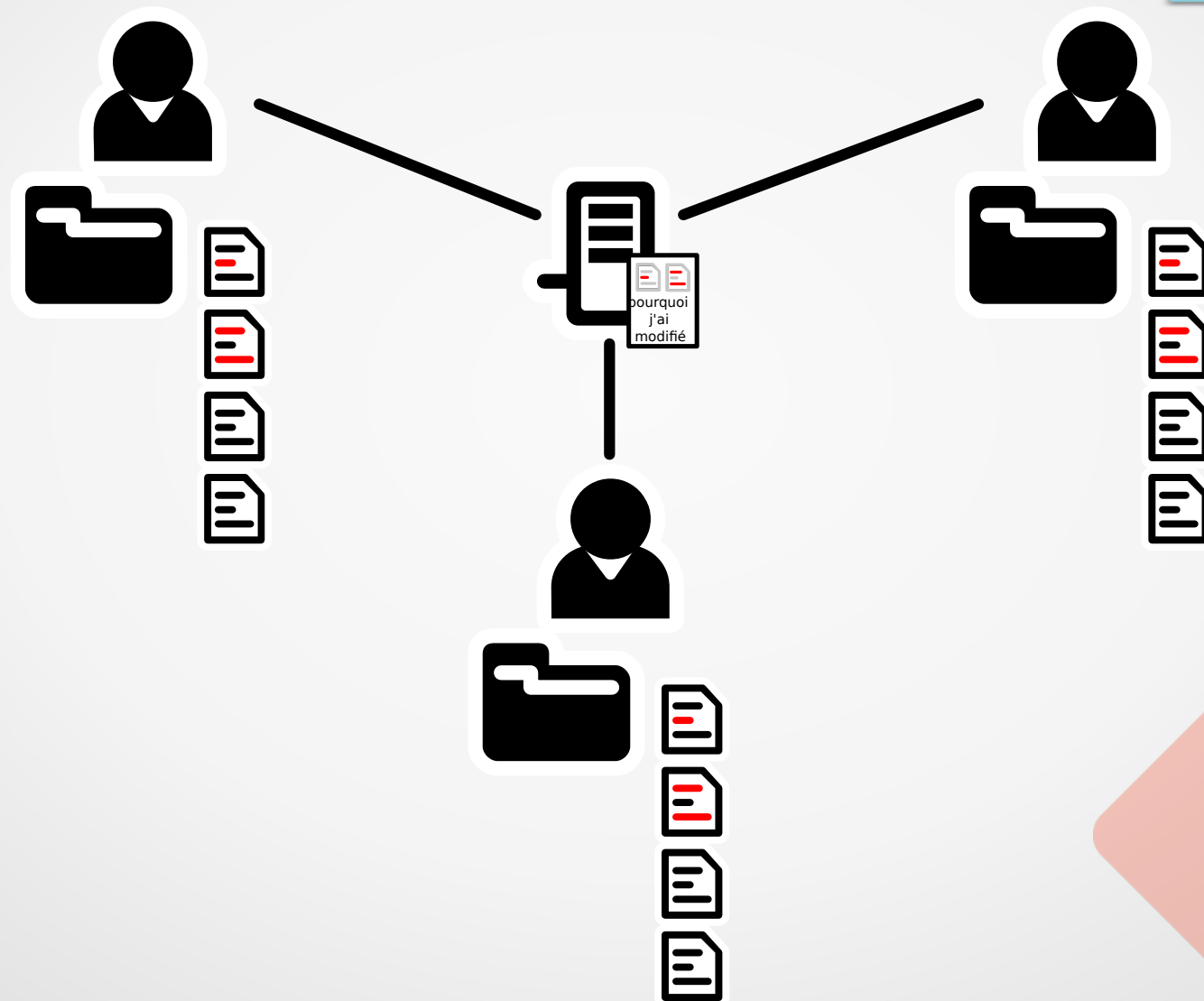






Git Merge

34



En pratique !

35

- 1ère étape, récupérer l'arch. Client-Serveur
<http://thibautcharles.net/coursgit/ClientServer.zip>
- Extraire à un endroit sympa
- Plusieurs dossiers :
 - Serveur : c'est le dossier ne contient que des informations git, c'est le repository central
 - Crom : Copie du projet d'un développeur
 - LinusTorvalds : idem

Note : Les repositories ont été configurés pour que les commits faits avec soient annotés du nom du dev et non le votre



- On va cloner le repository du serveur :

```
git clone <chemin vers Serveur> MyRepo
```

- Note : git prends en charge les protocoles SSH, HTTP(S) et FTP. On peut cloner un repository depuis un serveur sur internet en mettant son adresse.
Exemple : `git clone https://github.com/gitlabhq/gitlabhq`

- Le dossier MyRepo est créé, et contient les fichiers du projet !



On regarde ce qui a été créé

37

```
cd MyRepo  
git log
```

```
commit ed524be1e238690652ec85429c4c59a21cbbe368  
Author: Crom (Thibaut CHARLES) <crom29@hotmail.fr>  
Date: Thu Dec 19 00:28:42 2013 +0100  
    Fix: camelcase reported to cpp file  
    @linus : don't break my code ! ^^
```

```
commit 60582f9287aafc18457b8faf6a509182ac8dad2b  
Author: Linus Torvalds <Linus.Torvalds@nothing.foo>  
Date: Thu Dec 19 00:18:28 2013 +0100  
    Come on ! you should use camelCase !
```

```
[...]
```



Faisons quelques modifications...

38

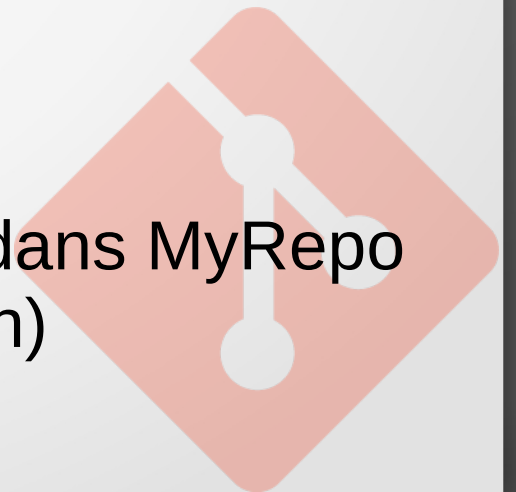
- Modification des fichiers...
- `git add <fichiermodifié>`
- `git commit`
- `git log`
-



Voyons ce que ça a fait sur les autres repo

39

- `cd [...] /Server`
`git log`
Pas de trace du commit
- `cd [...] /Crom`
`git log`
Pas de trace du commit
- `cd [...] /LinusTorvalds`
`git log`
Pas de trace du commit
- Il n'y a aucune trace du commit ailleurs que dans MyRepo
Le commit n'a pas encore été publié (push)



Publication des commits & remotes

40

- Pour publier un commit, il faut le push vers un serveur via une « remote »
- Les remotes sont des adresses internet/locales qui pointent vers d'autres repositories
- Quand on clone un repo, la remote "origin" est créée, et utilise l'adresse utilisée lors du clonage
- On peut rajouter des remotes avec :
`git remote add <nom> <adresse>`

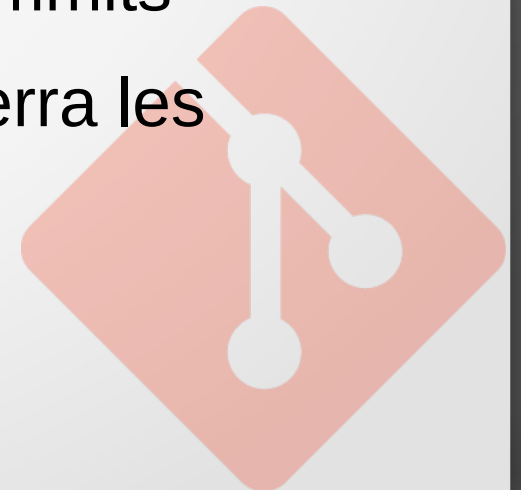


- Envoyons le commit vers le serveur :

```
git push origin master
```

```
Counting objects: 5, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 394 bytes | 0 bytes/s, done.  
Total 3 (delta 1), reused 0 (delta 0)  
To [...]ClientServer/Serveur  
   ed524be..f1932ad  master -> master
```

- origin : nom de l'endroit où envoyer le/les commits
- master : nom de la branche à envoyer (on verra les branches plus tard)



Voyons ce qui c'est passé

42

- `cd [...] /Server`
`git log`
On voit le nouveau commit
- `cd [...] /Crom`
`git log`
Pas de trace du commit
- `cd [...] /LinusTorvalds`
`git log`
Pas de trace du commit
- Les développeurs n'ont pas été mis à jour, c'est à eux de le faire



Prenons la place d'un autre développeur

43

- `cd [...]/Crom`
`git log`

Pas de trace du commit

- Mettons son repository à jour en tirant le nouveau commit :
`git pull origin master`

```
remote: Counting objects: 5, done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 3 (delta 1), reused 0 (delta 0)  
Unpacking objects: 100% (3/3), done.  
From [...]/ClientServer/Serveur  
* branch                master      -> FETCH_HEAD  
Updating ed524be..f1932ad  
Fast-forward  
 Async.cbp | 9 -----  
 1 file changed, 9 deletions(-)
```

Téléchargement du commit compressé

Merge des changements



Qu'est ce qui c'est passé

44

- Merge automatique :
 - Par défaut, quand on fait un Pull, Git fusionne les changements avec les fichiers présents.
- `git log`
 - Présence du dernier commit
- Les fichiers ont été modifiés pour inclure le dernier commit effectué

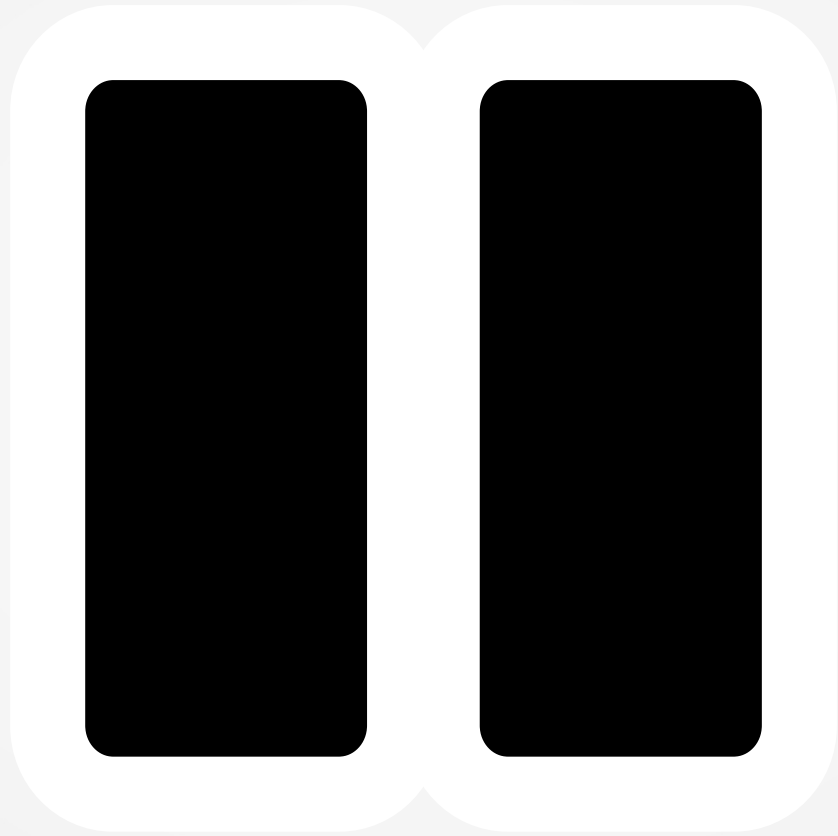


A vous de jouer !

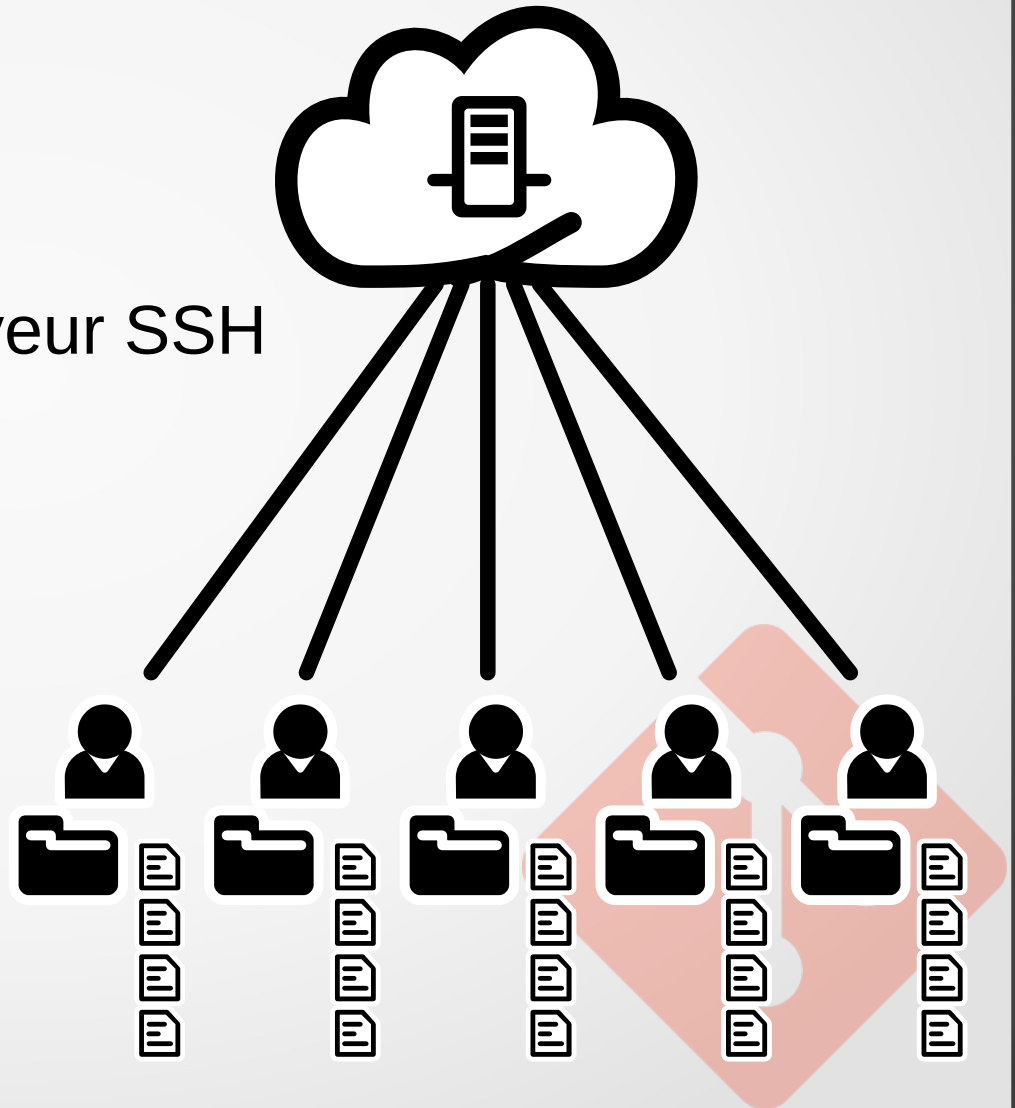
45

- Mettez à jour le repository LinusTorvalds





- Avantages
- Solutions d'hébergement
- Protocoles pris en charges
- Hébergement perso via serveur SSH
- Règles d'or



- Très simple à mettre en place
- Donne de la visibilité à votre projet
- Facilite l'accès aux sources

- Suivant l'hébergement du repository :
 - Report de bugs
 - Tickets
 - Tests automatiques



- GitHub
 - Git
 - Le plus connu, le plus utilisé
- Bitbucket
 - Git, SVN, Mercurial
- Gitlab
 - Git
 - A installer sur son site web
- Sur votre système Linux
 - Serveur SSH (le plus simple)



- La gestion des remotes est exactement la même que précédemment :

Au lieu de `origin=[...]/Serveur`

On aura `origin=https://github.com/[...]/MonRepo.git (https)`
ou `origin=git@github.com:[...]/MonRepo.git (ssh)`

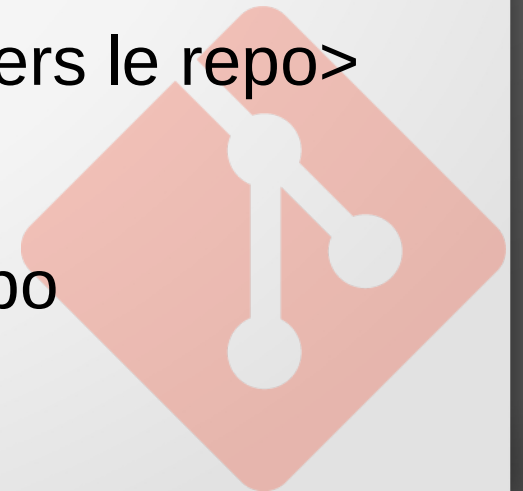
- Les `git clone` se feront avec l'adresse du repo sur le serveur
- Les `git push` et `git pull` se feront toujours avec `origin`



Hébergement personnel via SSH

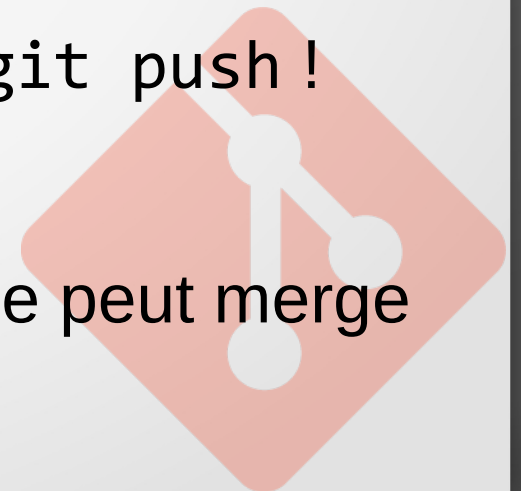
51

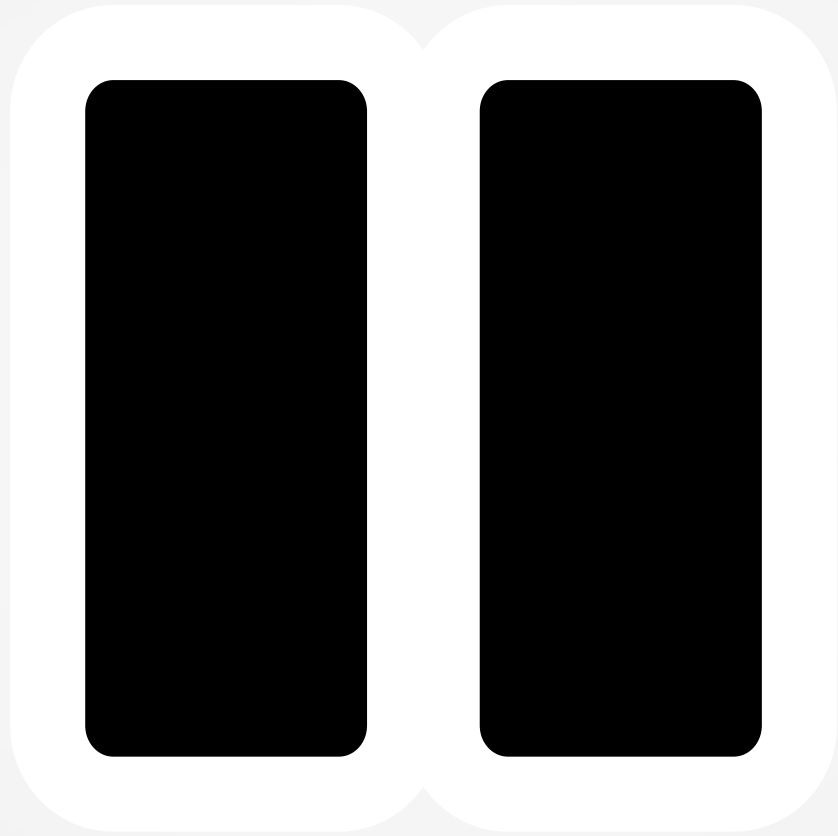
- C'est la solution la plus simple si on ne veut pas passer par un site web.
- Il faut un serveur ssh (openssh-server), installé par défaut sur la plupart des linux
- Les commandes se feront avec l'adresse :
`<utilisateur>@<ip du serveur>:<chemin vers le repo>`
- Exemple :
`crom@192.168.0.10:/home/crom/MonRepo`



- Vu qu'on travaille à plusieurs sur un projet, il est possible que deux développeurs fassent un commit en même temps.
- Cela pose problème puisque un serveur n'est pas capable de merge les commits lui même
C'est aux développeurs de le faire
- Il faut **toujours** faire un `git pull` avant un `git push` !

Nous verrons plus tard pourquoi le serveur ne peut merge tout seul (cf Gestion des conflits)

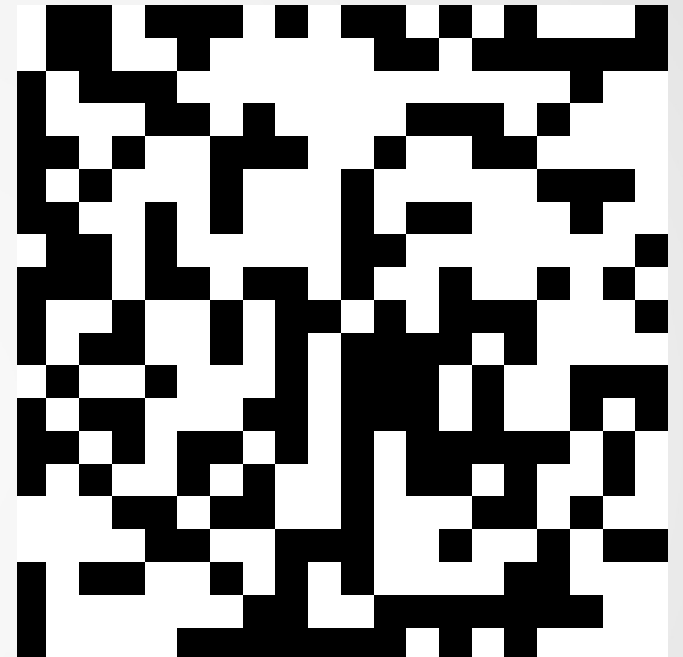




TP : Jeu de la vie

54

- But : compléter un programme le plus rapidement possible
- Constater que git, c'est bien !



TP : Jeu de la Vie

55

- Déjà on va récupérer le projet :

```
git clone git@thibautcharles.net:tcharl/coursgit-jeudelavie1.git
```

- User : guest

- Password : coursgit

- Par groupe de 5, chacun va :

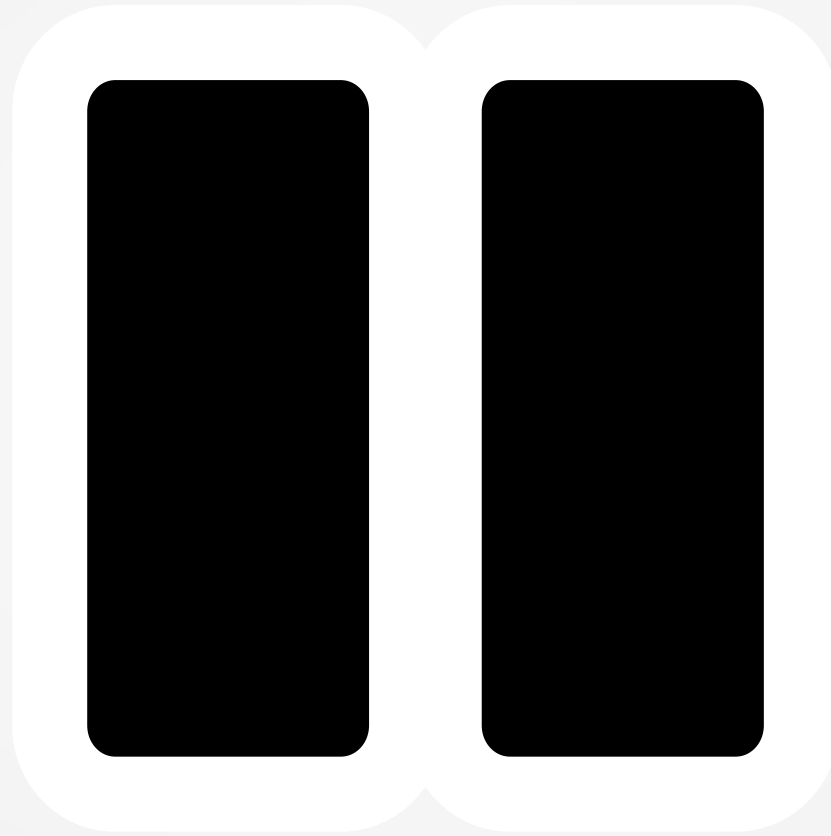
- compléter une des fonctions (ne pas modifier le code ailleurs)
- commiter ses modifications
- Pull puis Push ses commit(s)



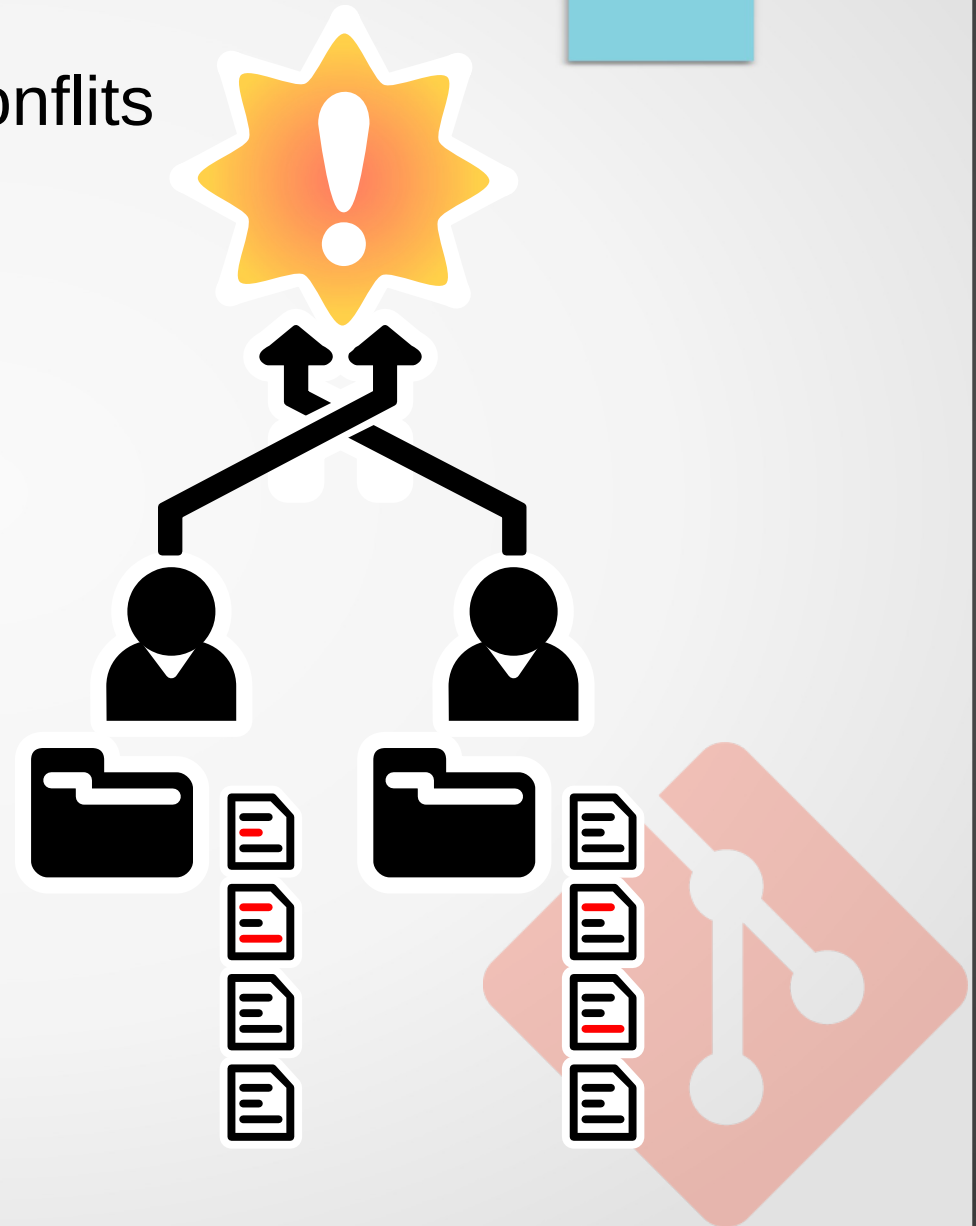
C'est parti !

56





- Comment apparaissent les conflits
- Gérer un conflit avec kdiff3



Apparition des conflits

59

- Crom modifie les lignes 5 à 16 de Async.hpp
Il commit les modifications et pull+push vers le serveur
- Pendant ce temps, Linus modifie les lignes 14 à 18 du même fichier et commit ses modifications
- Quand Linus fera un pull, il va y avoir un merge conflict sur les lignes 14, 15 et 16 de Async.hpp
- Git ne saura pas comment fusionner les deux fichiers
Il faudra lui dire comment faire



Créer un conflit

60

- On reprend ClientServer.zip

```
cd Crom
```

Suppression de la ligne X d'un fichier

```
git add <fichier>
```

```
git commit
```

```
git pull origin master
```

```
git push origin master
```

```
cd ../LinusTorvalds
```

Modification de la ligne X du même fichier

```
git add <fichier>
```

```
git commit
```

```
git pull origin master
```

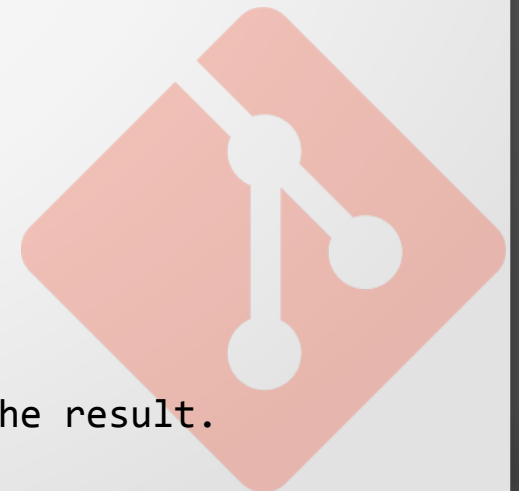
```
From ../Serveur
```

```
* branch                master      -> FETCH_HEAD
```

```
Auto-merging Async.cpp
```

```
CONFLICT (content): Merge conflict in Async.cpp
```

```
Automatic merge failed; fix conflicts and then commit the result.
```



- Kdiff est une interface graphique pour comparer les fichiers, et résoudre les conflits
 - Il en existe d'autres en console comme vimdiff

- Installation :

Ubuntu : `sudo apt-get install kdiff3`

- On va configurer git pour utiliser Kdiff :

- Linux :

Ajouter au fichier `~/.gitconfig` :

```
[merge]
  tool = kdiff3
```

- Windows :

`[utilisateur]/.gitconfig` :

```
[merge]
  tool = kdiff3
[mergetool "kdiff3"]
  path = C:/[...]/KDiff3/kdiff3.exe
```

Résoudre le conflit

62

- On lance la résolution du merge conflict :

```
git mergetool
```

```
Merging:  
Async.cpp
```

```
Normal merge conflict for 'Async.cpp':  
  {local}: modified file  
  {remote}: modified file  
Hit return to start merge resolution tool (kdiff3):
```

- Local : Version qu'on viens de modifier chez Linus
- Remote : Version sur le serveur



Async.cpp (Base) <-> Async.cpp (Local) <-> Async.cpp (Remote) - KDiff3

Fichier Édition Dossier Déplacement Différences Fusion Fenêtre Configuration Aide

A (Base): Async.cpp (Base) B: Async.cpp (Local) C: Async.cpp (Remote)

Première ligne 4 Encodage : UTF-8 Style de fin de lign Première ligne 4 Encodage : UTF-8 Style de fin de lign Première ligne 4 Encodage : UTF-8 Style de fin de lign

```
//=====
bool operator>=(const struct timeval& t
{
    if(t1.tv_sec>t2.tv_sec)
        return true;
    else if(t1.tv_sec<t2.tv_sec)
        return false;
    else
    {
        if(t1.tv_usec>=t2.tv_us
            return true;
        else
            return false;
    }
}
//=====
struct timeval operator+(const struct t
```

Avant que le fichier soit modifié

Version que vous avez modifié

Version qui est sur le serveur

Sortie: Async.cpp Encodage pour l'enregistrement: Codec depuis C: UTF-8 Style de fin de ligne: Unix (A, B, C)

?] <Conflit de fusion>
C] <Pas de ligne source>

```
return true;
else
{
    if(t1.tv_usec>=t2.tv_use
        return true;
    else
        return false;
}
//=====
struct timeval operator+(const struct timeval& t1, const struct timeval& t2)
{
    struct timeval ret;
```

Fichier Async.cpp (Local): Ligne 11

**Version finale,
conflit corrigé**

Résoudre le conflit

64

- On peut soit :
 - choisir la version A (base), B (local) ou C (remote)
 - Réécrire soi même la portion de code en conflit
- Une fois terminé, on sauvegarde et on quitte KDiff



Voyons ce qui c'est passé

65

git status

```
# On branch master
# All conflicts fixed but you are still merging.
#   (use "git commit" to conclude merge)
#
# Changes to be committed:
#   modified:   Async.cpp
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   Async.cpp.orig
```

- Async.cpp.orig : C'est le fichier tel qu'il était avant le merge. On peut le supprimer.



Il est temps de Commit !

66

- Maintenant qu'on a résolu le merge conflict, il faut commiter.

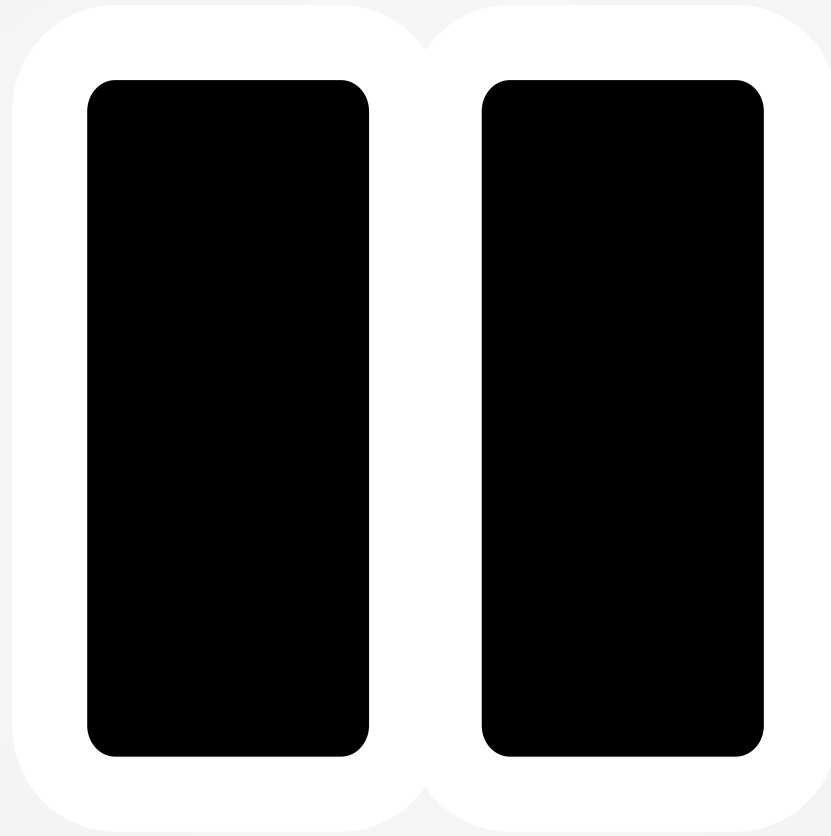
```
git commit
```

Note : les fichiers mergés lors du conflit ont déjà été ajoutés au commit
Il y a également un message par défaut

- On peut ensuite publier le commit de merge :

```
git push origin master
```





Branches, branches et branches partout

68



- Qu'est ce que c'est ?
- A quoi ça sert ?
- La branche master
- Création d'une branche
- Sauter de branche en branche
- Merge d'une branche
- Récupération d'une branche distante
- Un bon workflow avec des branches



Qu'est ce que c'est ?

69

- C'est un fil de développement du projet
Exemples :
 - Remplacement de la surcouche d'affichage
 - Essai d'implémentation de la fonctionnalité X



A quoi ça sert ?

70

- Essayer quelque chose qui n'est pas sûr d'aboutir
- Travailler indépendamment, sans être pollué par les commits des autres développeurs
- Ne pas altérer le développement normal du projet
- Proposer une solution sans faire partie de l'équipe du projet (fork & pull requests)



La Branche master

71

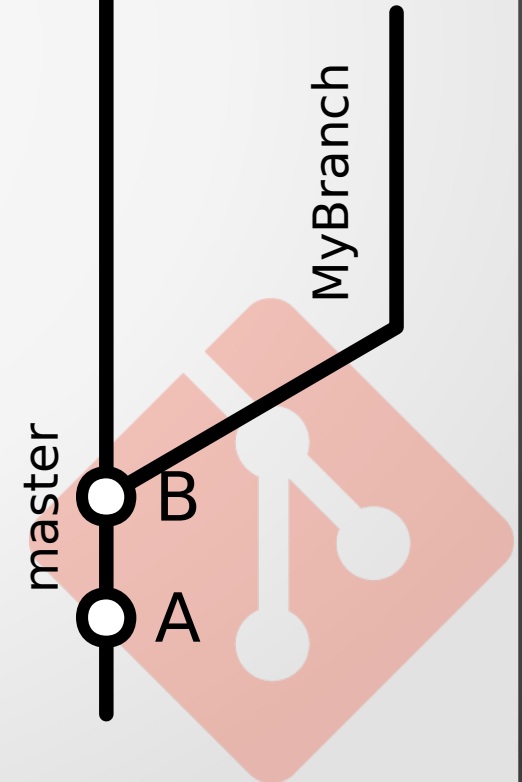
- On a déjà touché à une branche : master
- C'est la branche par défaut créée lors de la création du repository
- Sur les gros projets, master est souvent utilisée pour fournir les versions stables



Création d'une branche

72

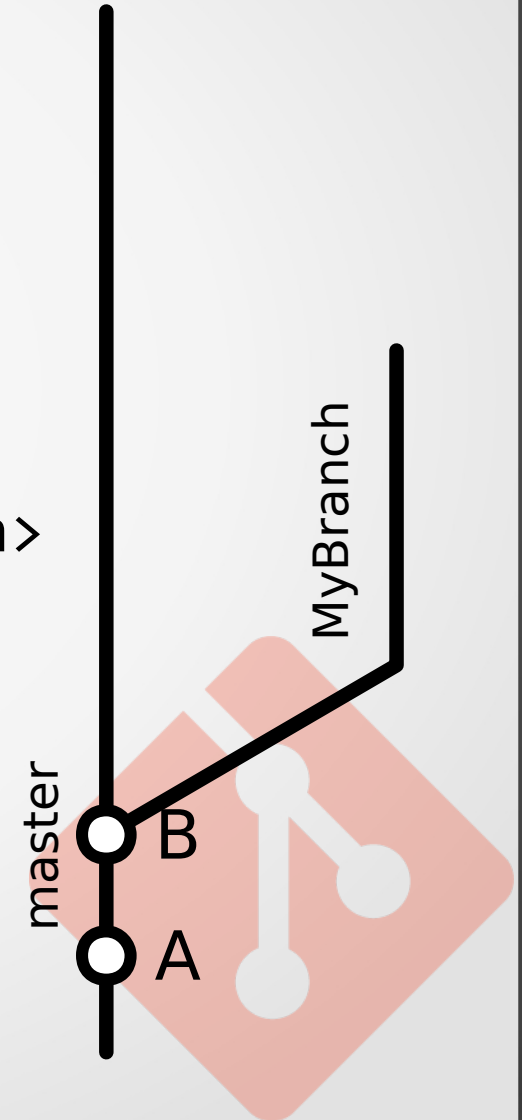
- Rien de plus simple :
`git branch MyBranch`
- Si on liste les branches, on voit la nouvelle :
`git branch`
 MyBranch
* master
- L'étoile indique sur quelle branche on est



Sauter de branche en branche

73

- Pour aller sur une branche :
`git checkout <destination>`
- Pour aller sur une branche distante
(disponible uniquement sur le serveur)
`git checkout -b <branch> origin/<branch>`



Commits dans la branche

74

- On va faire des commits dans les branches

```
git checkout MyBranch  
Modification de fichiers  
git add [...]  
git commit
```

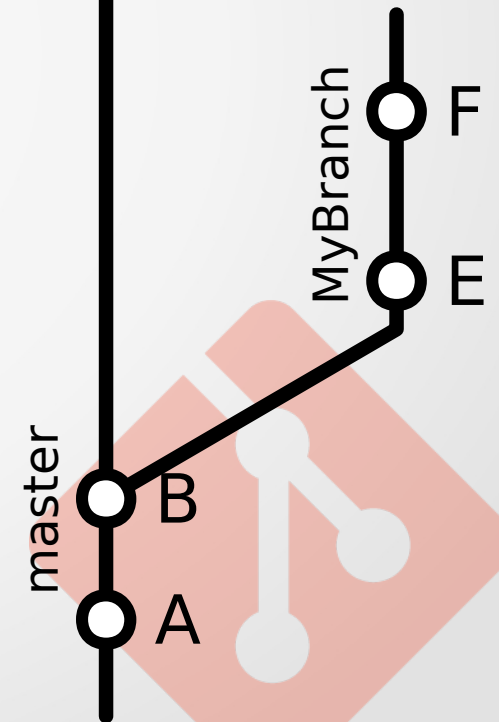
- Regardons ce qui c'est passé :

```
git log
```

Présence des commits A,B,E,F

```
git checkout master  
git log
```

Présence des commits A,B



Travail en parallèle

75

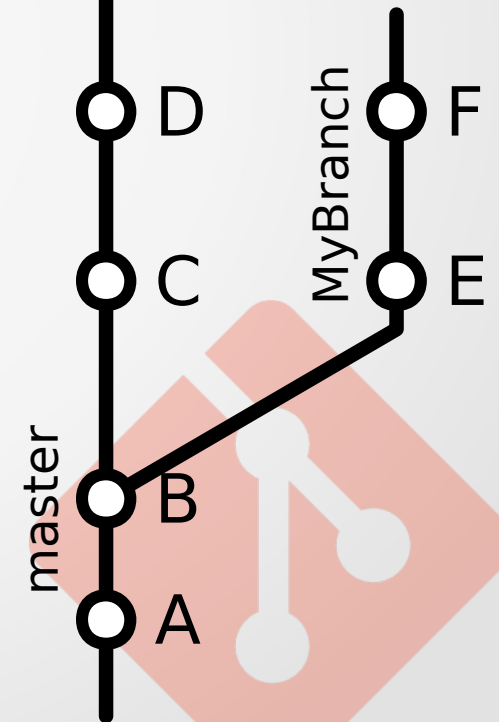
- On va faire quelques commits dans master :

```
git checkout master  
Modification de fichiers  
git add [...]  
git commit
```

- On regarde ce qui c'est passé :

```
git log
```

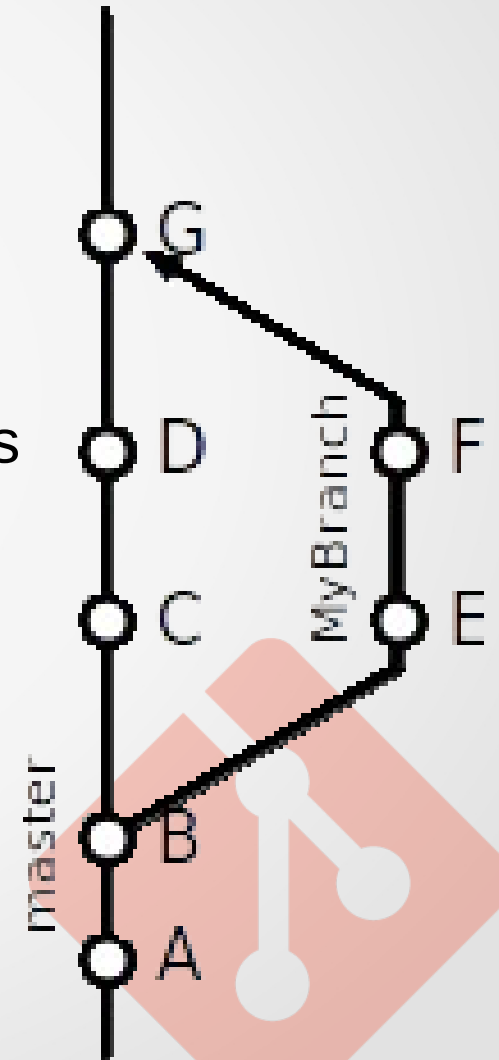
Présence des commits A,B,C,D



Merge d'une branche

76

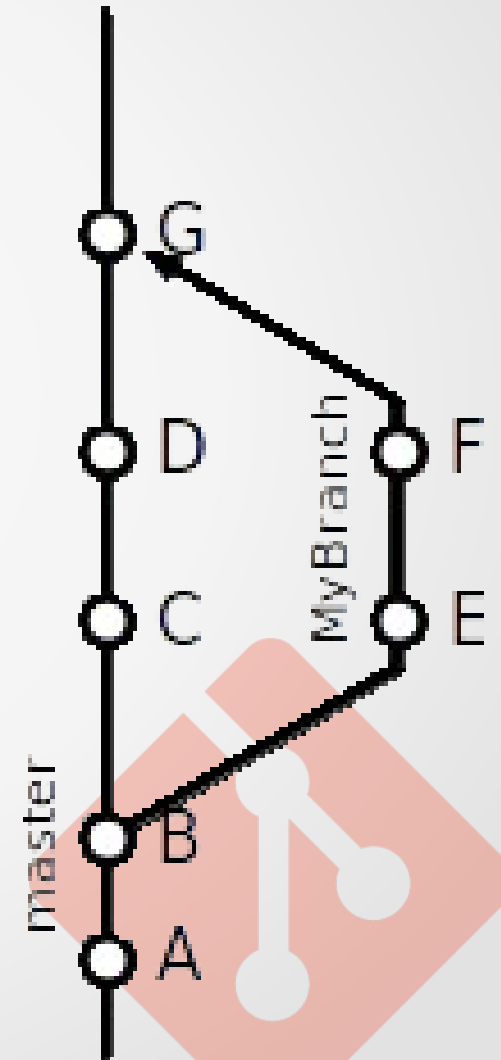
- C'est appliquer les commits d'une branche sur une autre
- Il peut y avoir des merge conflicts
- Le merge créé un commit dans la branche de destination
- Le merge commit contient l'ensemble des commits faits dans la branche mergée
- On merge une branche quand :
 - On a fini de travailler dessus et le travail a abouti
 - On veut ajouter les fonctionnalités de la branche dans une autre



Allons y !

77

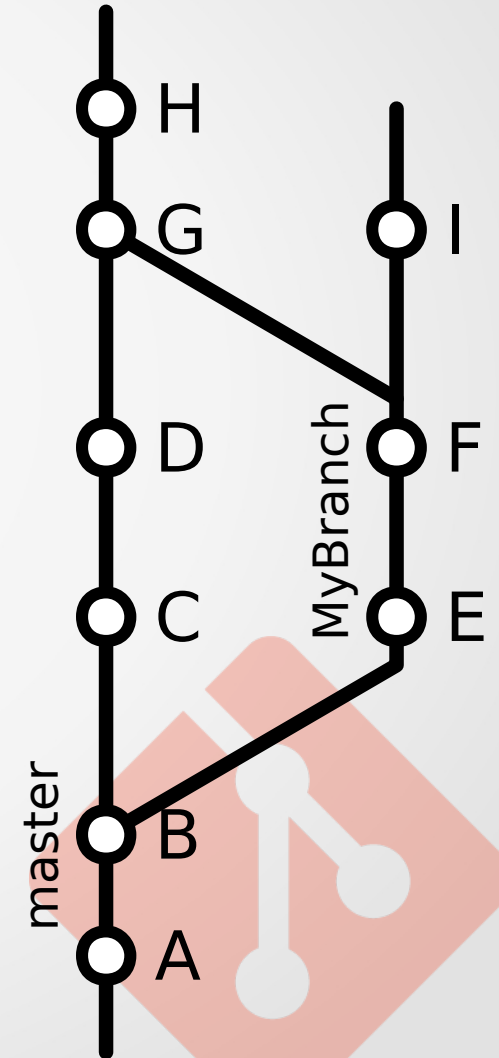
- Merge :
 - On va sur la branche de destination :
`git checkout master`
 - On merge la branche :
`git merge MyBranch`
 - Et on commente le commit
- `git log`
 - Présence de tous les commits



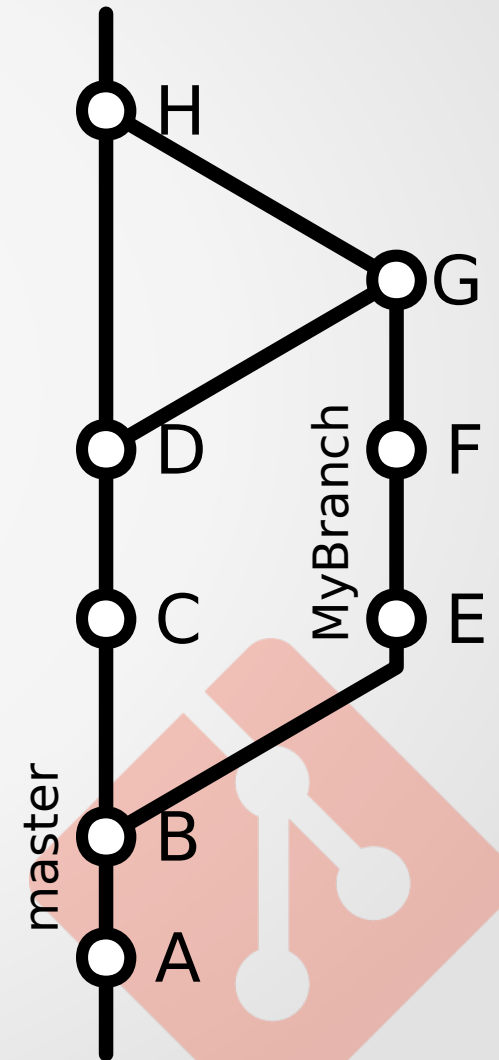
Quelques notes

78

- On peut continuer à travailler sur une branche après l'avoir mergée.
- Les commits faits sur la branche ne seront pas visibles dans une autre.

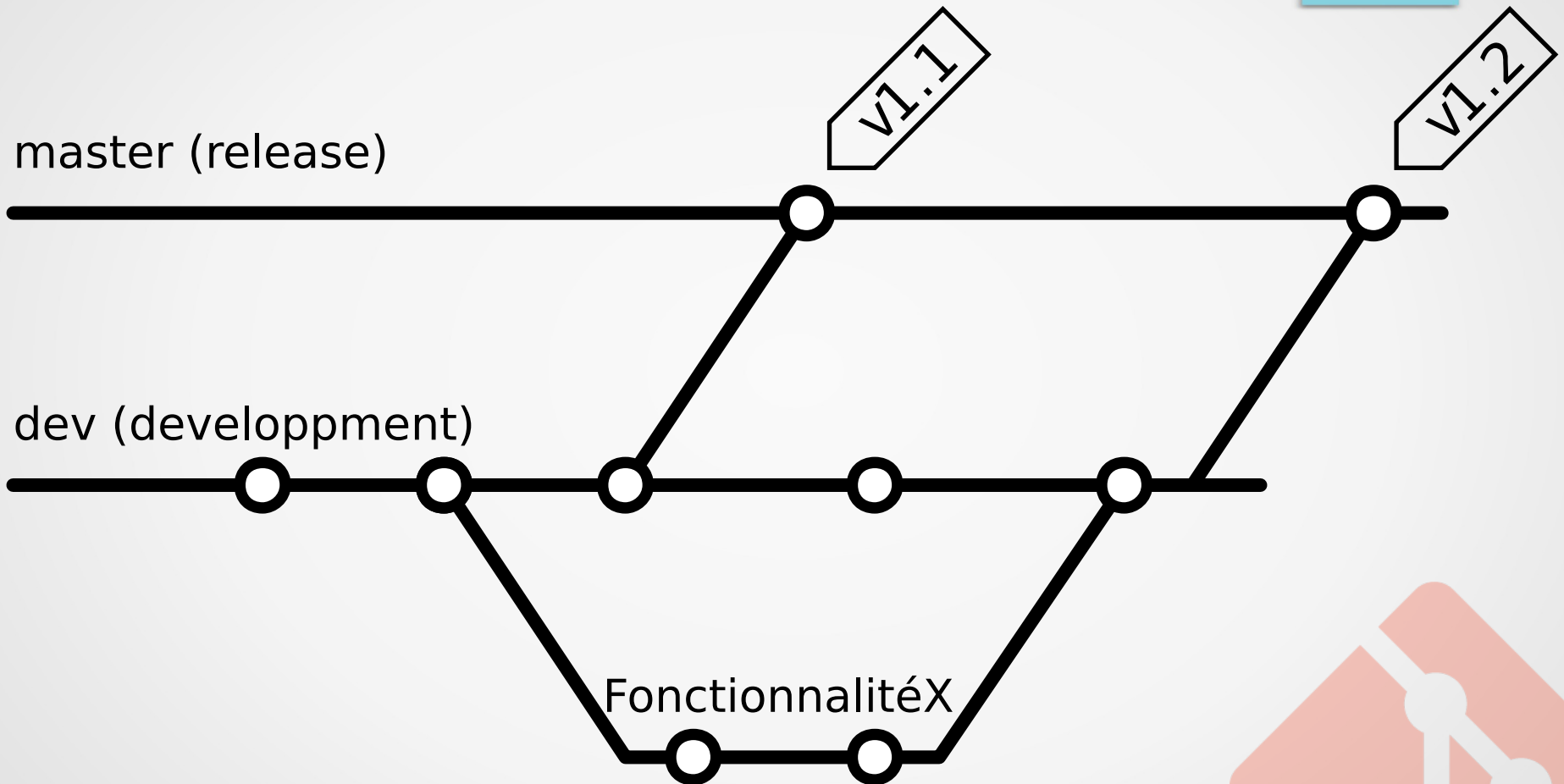


- Pour faciliter les merge et ne pas bloquer le développement dans master, on peut procéder ainsi :
 - Merge de master dans MyBranch
 - Les conflits peuvent être résolus tranquillement
 - On peut tester le résultat
 - Merge de MyBranch dans master
 - Aucun conflits



Un bon workflow avec des branches

80



- Il existe plein d'interfaces graphiques :
 - Linux :
 - gitg, giggle, qgit, ...
 - Windows :
 - Git Extensions, SourceTree, ...
 - Mac :
 - Tower, SourceTree, ...
 - Multiplateforme :
 - GitEye, SmartGit, ...



gitg - voilier-autonome (master)

Fichier Édition Affichage Aide

Historique Commit

Branche : Toutes les branches

Sujet	Auteur	Date
<input type="radio"/> unstaged Modifications dé-présélectionnées		mer. 05 févr. 2014 11:10:57
<input checked="" type="radio"/> master origin/master Added Hardware class & ANSI color support	Crom (Thibaut CHARLES)	mar. 04 févr. 2014 23:16:36
<input checked="" type="radio"/> vibe origin/vibe Merge branch 'master' into vibe	Crom (Thibaut CHARLES)	mar. 04 févr. 2014 13:16:02
<input type="radio"/> Updated makefile for cross compiling	Crom (Thibaut CHARLES)	mar. 04 févr. 2014 10:51:25
<input type="radio"/> Added API with sample helm handling	Thomas Abot	lun. 20 janv. 2014 15:28:53
<input type="radio"/> replaced Vibe's default main by custom main	Thomas Abot	lun. 20 janv. 2014 11:50:03
<input type="radio"/> Redesigned dynamic ip address finding	Thomas Abot	lun. 20 janv. 2014 11:46:01
<input type="radio"/> Merge branch 'master' into vibe	Thomas Abot	lun. 20 janv. 2014 10:25:33
<input type="radio"/> Ignoring deps to improve build time	Crom (Thibaut CHARLES)	mer. 15 janv. 2014 16:53:18
<input type="radio"/> Renamed SuivreCap to Autopilot	Crom (Thibaut CHARLES)	mer. 15 janv. 2014 16:50:33
<input type="radio"/> Added sample Thread & remove every crappy vibe shit	Crom (Thibaut CHARLES)	mer. 15 janv. 2014 14:07:56
<input type="radio"/> Removed vibe.d for better build times	Crom (Thibaut CHARLES)	mer. 15 janv. 2014 13:51:08
<input type="radio"/> Added README	Crom (Thibaut CHARLES)	mer. 15 janv. 2014 12:42:44
<input type="radio"/> Added little function comment	Thomas Abot	mer. 15 janv. 2014 12:02:08

Détails Modifications Fichiers

SHA : d439f375f06a268e5d3bc6840038ca5d1c77bf16

Auteur : [Crom \(Thibaut CHARLES\) <crom29@hotmail.fr>](#) (mar. 04 févr. 2014 13:16:02 CET)

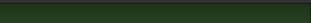
Committeur : [Crom \(Thibaut CHARLES\) <crom29@hotmail.fr>](#) (mar. 04 févr. 2014 13:16:02 CET)

Sujet : Merge branch 'master' into vibe

Parent : [c3f6e635f3a0d36925e4beb6f7e77eabb0073d5c](#) : Added REST API with sample helm handling
[7bf88d21364e51dac162038a6b12cc914e7c8317](#) : Updated makefile for cross compiling

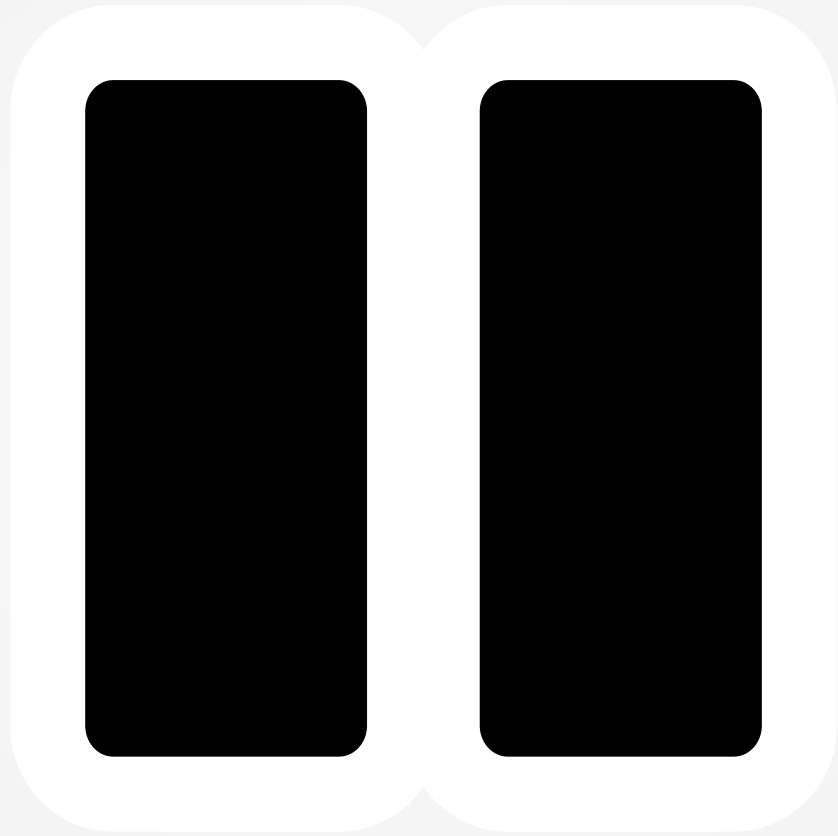
Merge branch 'master' into vibe

Conflicts:
Makefile

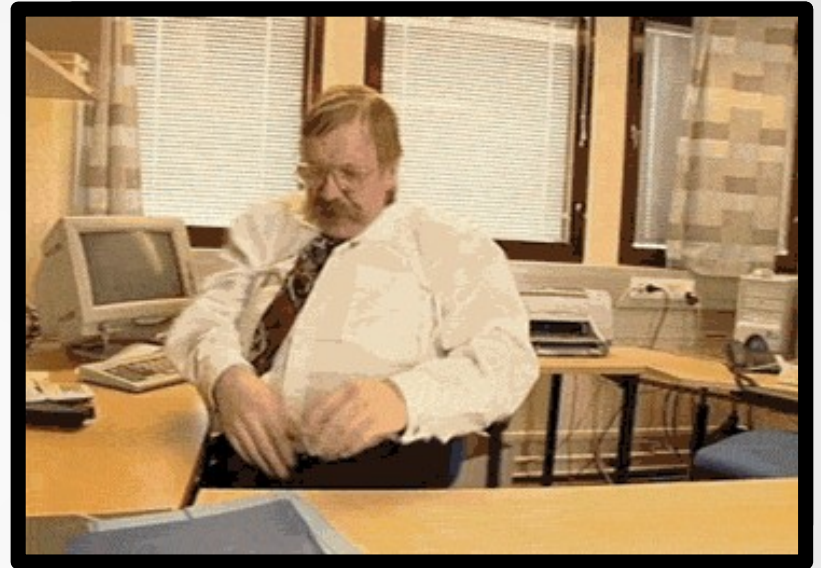
[Makefile](#)  17

18 révisions chargées en 0,03 s





- Pour tout mettre de côté sans se prendre la tête !
- Pourquoi s'en servir ?
- Comment s'en servir



- Permet de mettre de côté les modifications en cours, sans faire de commit
- On peut récupérer ces modifications à tout moment
- Permet de nettoyer rapidement le repository, en conservant les modifications quelque part



Pourquoi s'en servir ?

86

- On travaille sur quelque chose complètement inabouti, et on veut passer à autre chose
- Quelqu'un demande de corriger un bug d'urgence, et on abandonne son travail en cours
- On fait n'importe quoi et on veut recommencer, tout en gardant une trace de ce qu'on a fait



Comment s'en servir

87

- Mettre toutes les modifications en cours dans le stash :

```
git stash
```

- Lister toutes les stash disponibles :

```
git stash list
```

- Récupérer le contenu de la stash :

```
git stash apply stash@\{<numéro de la stash>\}
```

ou

```
git stash apply
```

Pour appliquer le dernier contenu ajouté



Essayons !

88

- Dans un repository :
 - Modification d'un fichier
 - `git status`
 - On voit la modification
 - `git stash`
 - `git status`
 - Le repository est propre
 - `git stash list`
 - On voit la stash sauvegardée
- Pour récupérer le contenu de la stash :
 - `git stash apply`
 - `git status`
 - On voit la modification



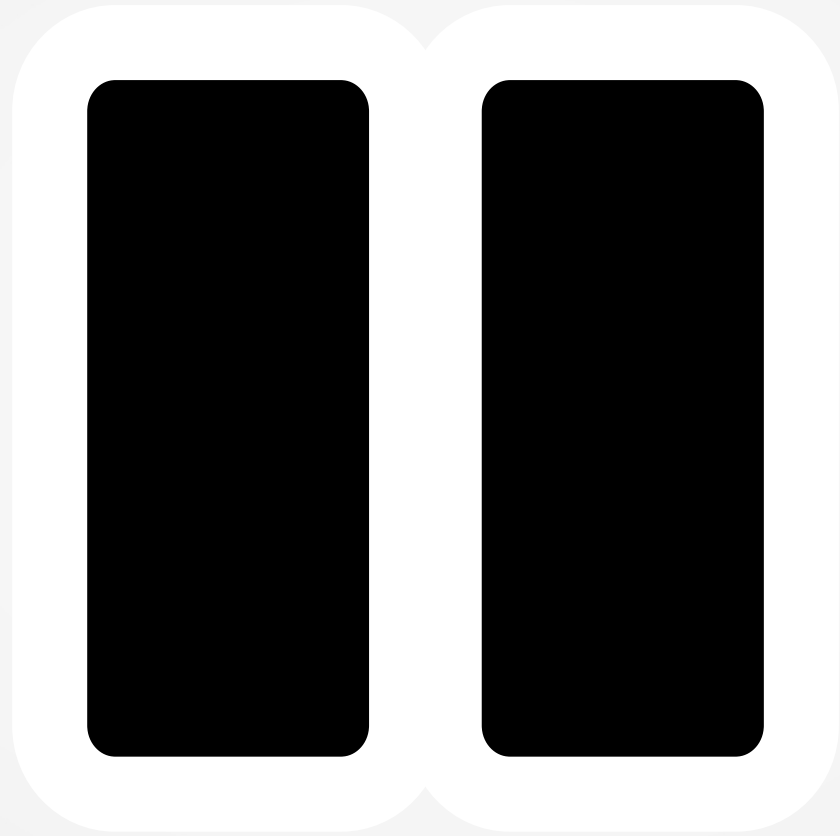
Suppression d'une stash

89

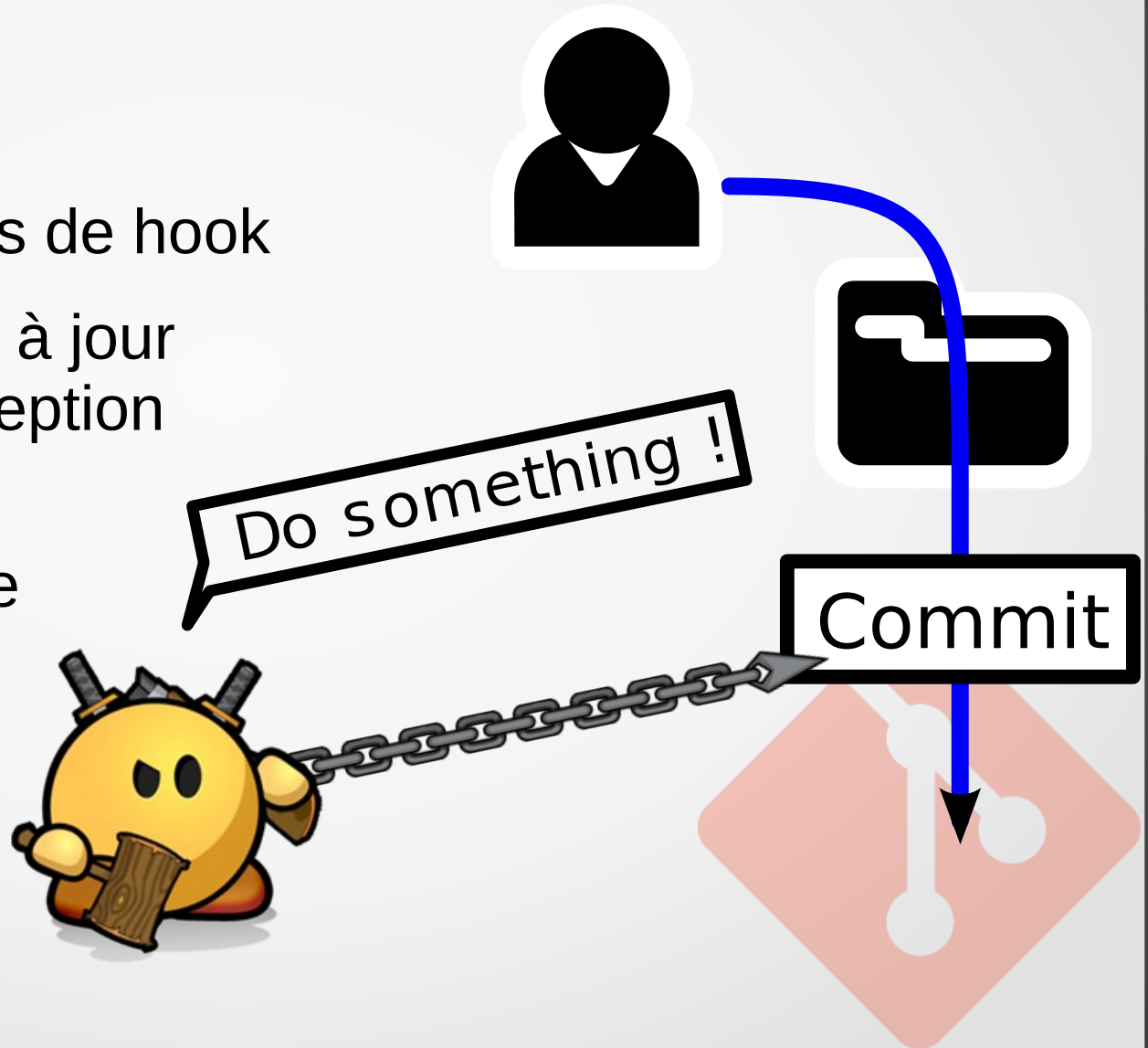
- Suppression d'une stash :

```
git stash drop stash@\{<numéro de la stash>\}
```





- Présentation
- Les différents hooks
- Détails sur les scripts de hook
- Configurer une mise à jour automatique par réception de données
- Mettre à jour son site web avec git



- Un hook permet d'exécuter des programmes/scripts lors d'une action sur le repository git
- Il faut placer les scripts dans `.git/hooks/`
- Exemples :
 - Envoyer des mails lors d'un commit sur master
 - Lancer une compilation lors d'un push vers le repository
 - Lancer des tests automatiques quand le code est modifié
 - Mettre à jour un repo. quand le serveur reçoit des commits
 - ...



- Pour les repository personnels
 - pre-commit : Juste après qu'on ait entré 'git commit'
 - post-commit : Une fois qu'une commit est effectué
 - post-checkout : Après avoir lancé un git checkout
 - post-merge : Après avoir mergé une branche (ou pull)
- Pour les repository 'serveur' (bare)
 - pre-receive : Juste avant que le serveur reçoive des commits
 - post-receive : Quand le serveur a reçu des commits



Détails d'un script de hook

94

- Le script peut être un script bash, perl, python, ...
- L'interpréteur est donné en entête de fichier :

```
#!<interpréteur>  
[...]
```

- Pour effectuer des commandes git, dans un autre repository, il faut configurer les variables :
 - GIT_WORK_TREE : Chemin vers le repository git
 - GIT_DIR : chemin vers le dossier .git



Mise à jour auto via réception de données

95

- Exemple de post-receive (Serveur/hooks/post-receive) :

```
#!/bin/bash

cd ../Crom #On se déplace vers le repo à mettre à jour
export GIT_WORK_TREE=`pwd` #On indique le repo
export GIT_DIR=`pwd`/.git #on indique le dossier .git/
git pull origin master #On le met à jour
```

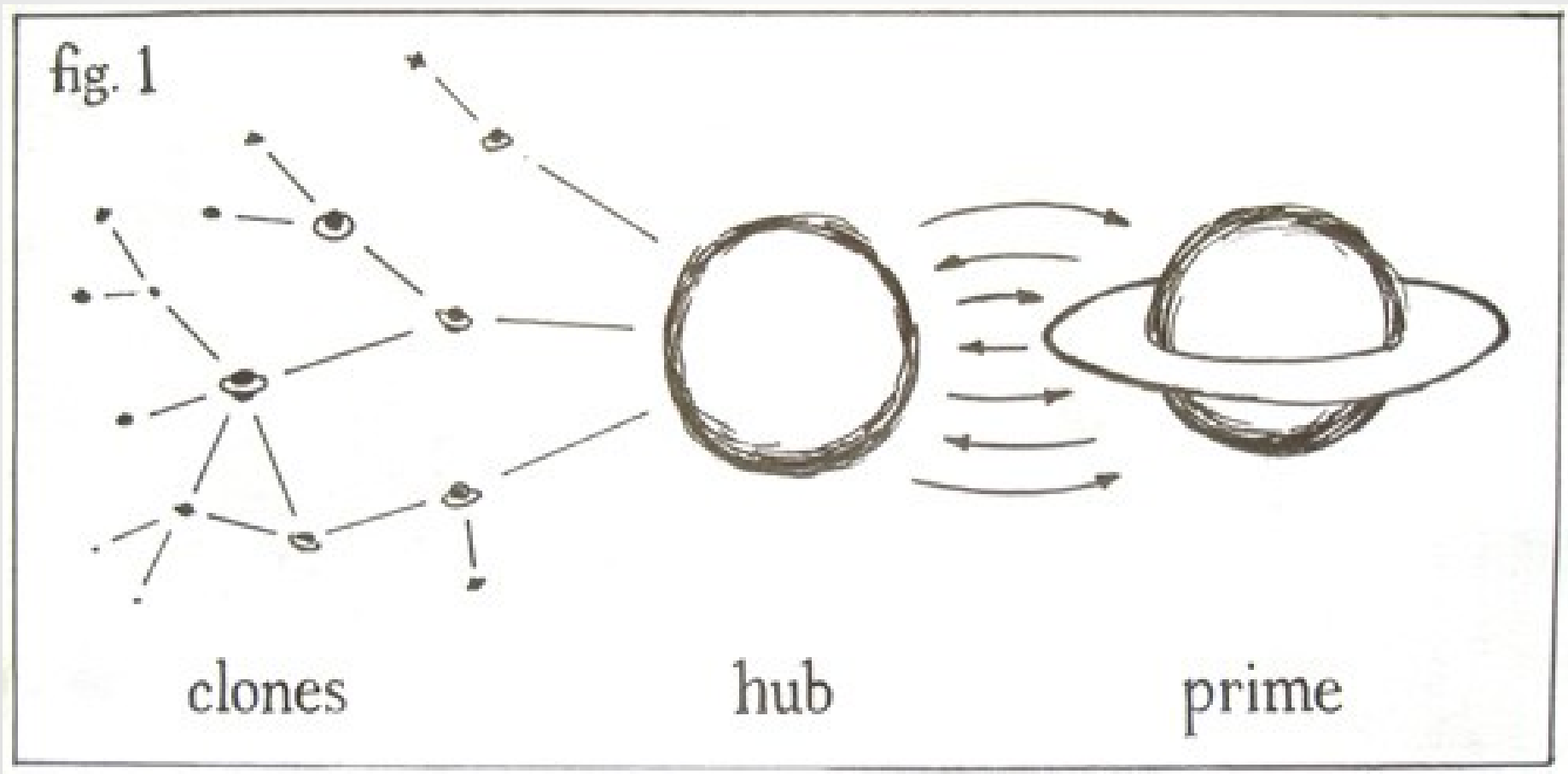
Maintenant, si on push un commit depuis le repository LinusTorvalds ou MyRepo, le serveur mettra automatiquement le repository de Crom à jour.



A Web Focused Git Workflow

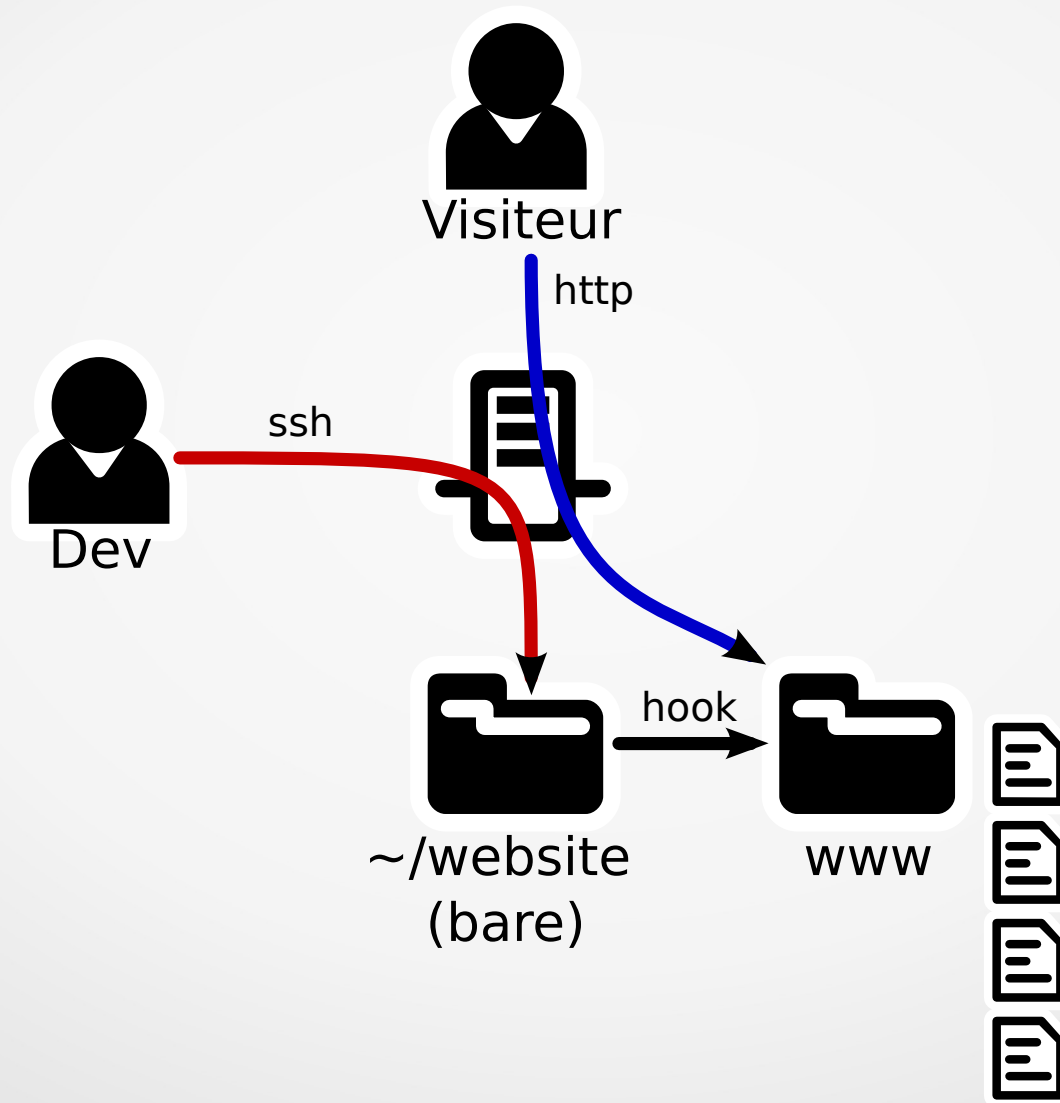
96

- <http://joemaller.com/990/a-web-focused-git-workflow/>



Mettre à jour son site web via git

97



Création des repositories

98

- On va créer un bare repository facilement accessible sur le serveur :

```
cd ~  
mkdir website  
git init --bare
```

- Ensuite le repository du site web, utilisé par le webserver :

```
cd /var/www  
git clone $HOME/website ./website
```



- Mise en place le git hook :

```
cd ~/website/hooks  
touch post-receive  
chmod +x post-receive  
nano post-receive
```

```
#!/bin/bash  
  
cd /var/www/website  
export GIT_WORK_TREE=`pwd`  
export GIT_DIR=`pwd`/.git  
git pull origin master
```

Note : **Attention** aux guillemets inversés (AltGr + 7)



Accès SSH

10
0

- La configuration du pc de développement :

```
git remote add server ssh://<user>@<ipserveur>:~/webserver
```

- L'accès SSH via clés :

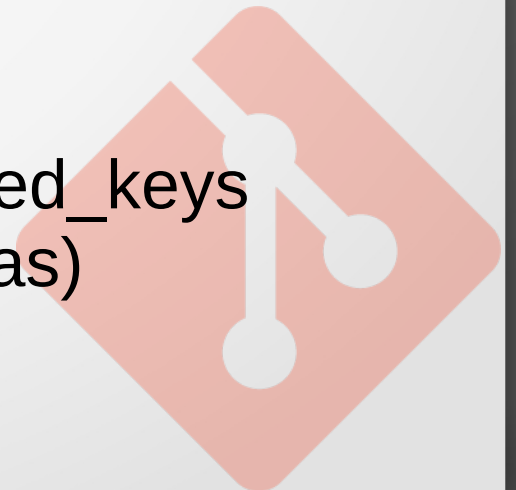
- Sur le PC de dev, si c'est pas déjà fait :

```
ssh-keygen
```

Copier le contenu de `~/.ssh/id_rsa.pub`

- Sur le serveur :

Coller le contenu dans `~/.ssh/.authorized_keys`
(Créer le fichier si celui ci n'existe pas)



Mettre à jour son site web via git

10
1

- On va créer un bare repository facilement accessible sur le serveur :

```
cd ~  
mkdir website  
git init --bare
```

- Ensuite le repository du site web :

```
cd /var/www  
git clone $HOME/website ./website
```



Mettre à jour son site web via git

10
2

- Pour mettre à jour le site web, il suffira simplement de faire (sur le pc de développement) :
`git push server master`
- Et c'est tout !
- Et en plus c'est rapide ! (fichiers modifiés compressés)



10
3

