

Introduction à Git

Par Ulrich MONJI



Objectifs de la formation

A la fin de notre formation, nous serons capables de:

- 1. Connaître les spécificités de git**
- 2. Savoir mettre en place git et le configurer**
- 3. Savoir utiliser les bonnes pratiques d'utilisation et comprendre leur intérêt**
- 4. Etre à l'aise avec les outils git**

Plan

- Présentation du formateur
- Présentation de git
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

Plan

- Présentation du formateur
- Présentation de git
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

Ulrich MONJI – Consultant Cloud/DevOps

Atos-Worldline - Ingénieur Système

- Build et Run de plateforme Cloud
- Virtualisation - Stockage - Automatisation
- Comptes clients: Carrefour, Auchan, ARJEL, SAMU

Adneom - Consultant IT

Groupe SII - Consultant IT (Cloud/Devops)

- Consultant chez Orange France
- Migration d'une application monolithique en microservice

Formateur et blogueur chez eazytraining

Plan

- Présentation du formateur
- **Présentation de git**
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

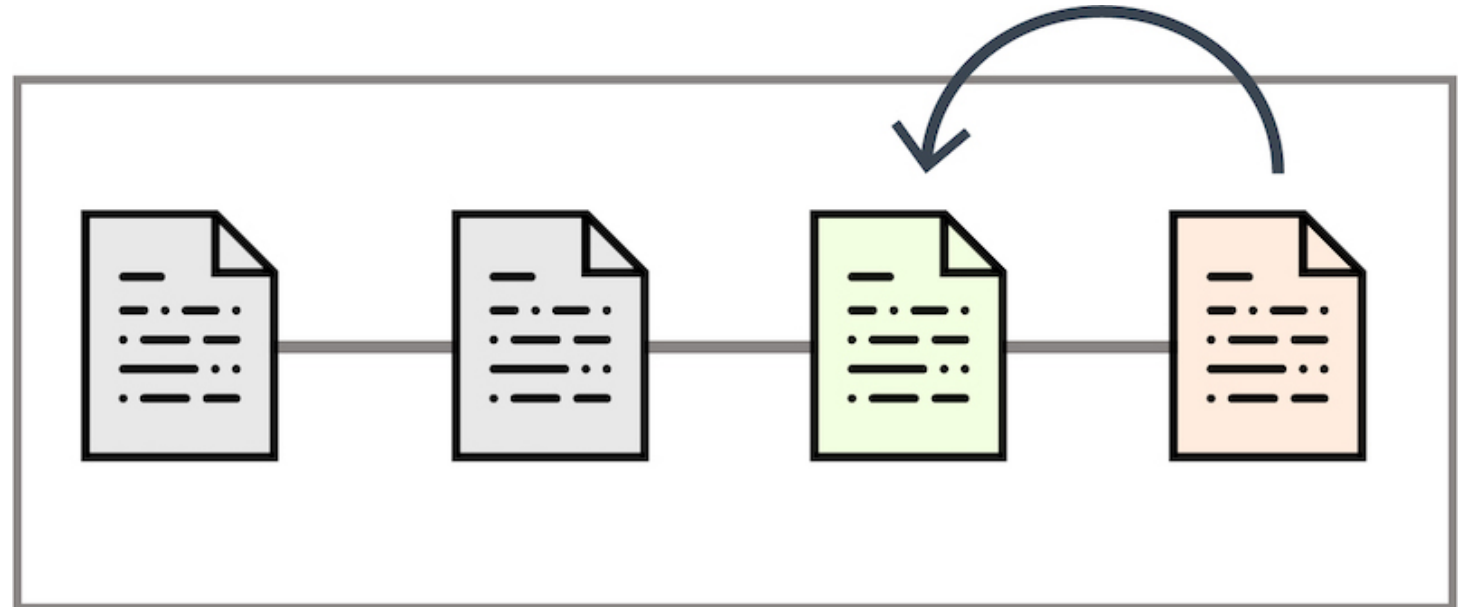
1 - Concepts de base du versionning

- Revenir en arrière
- Avoir un historique de son travail
- Travail à plusieurs
- Evolutivité de son code



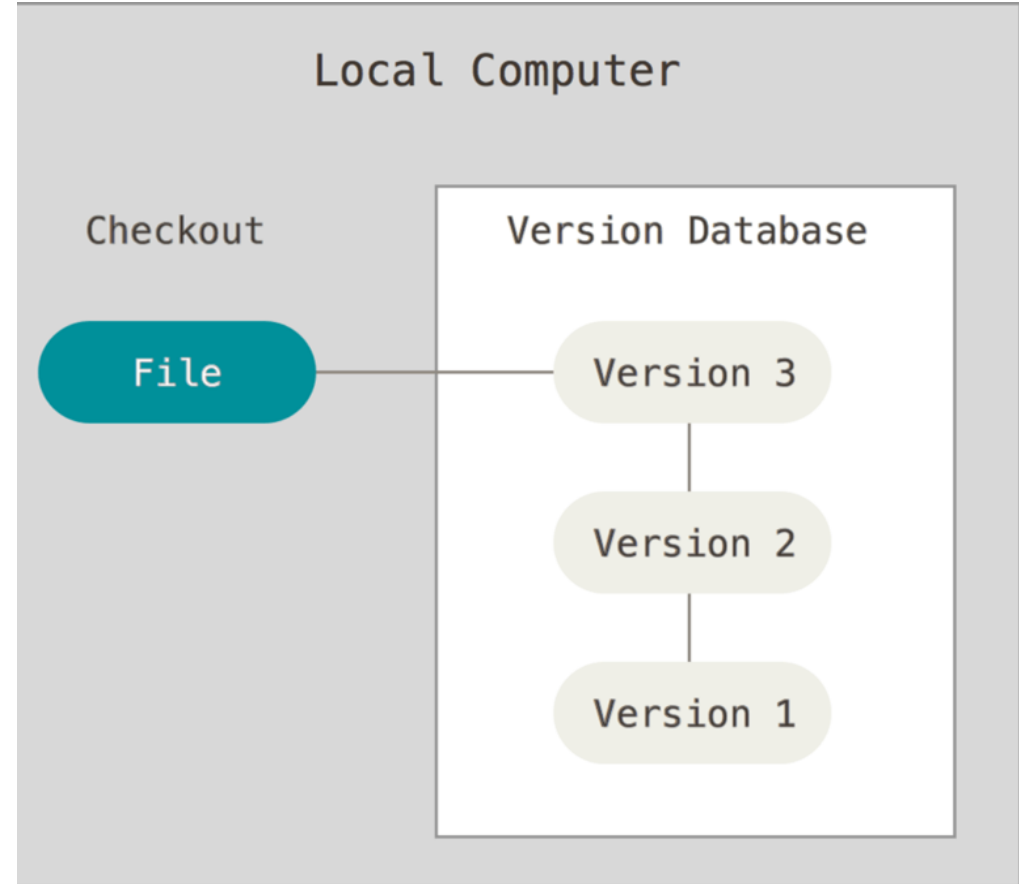
1 - Concepts de base du versionning

Version en production fonctionnelle puis publication de la nouvelle version qui **ajoute des bugs!!!!**



2 - Types de versionning

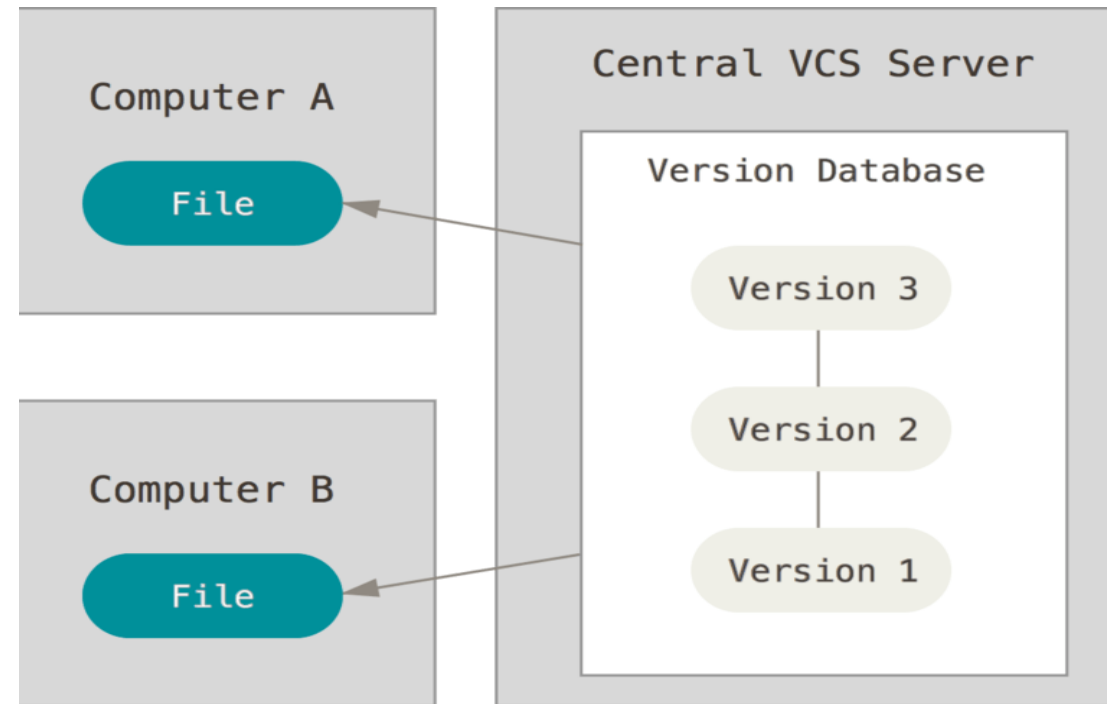
1 – Versionning local



2 - Types de versionning

- CVS, Subversion ou encore perforce
- Collaboration
- SPOF

2 – Versionning central



2 - Types de versionning

2 – Versionning central

Principe

“ Il existe une seule copie **centrale** de votre projet quelque part et les développeurs vont **commiter** leurs changements sur celle-ci »

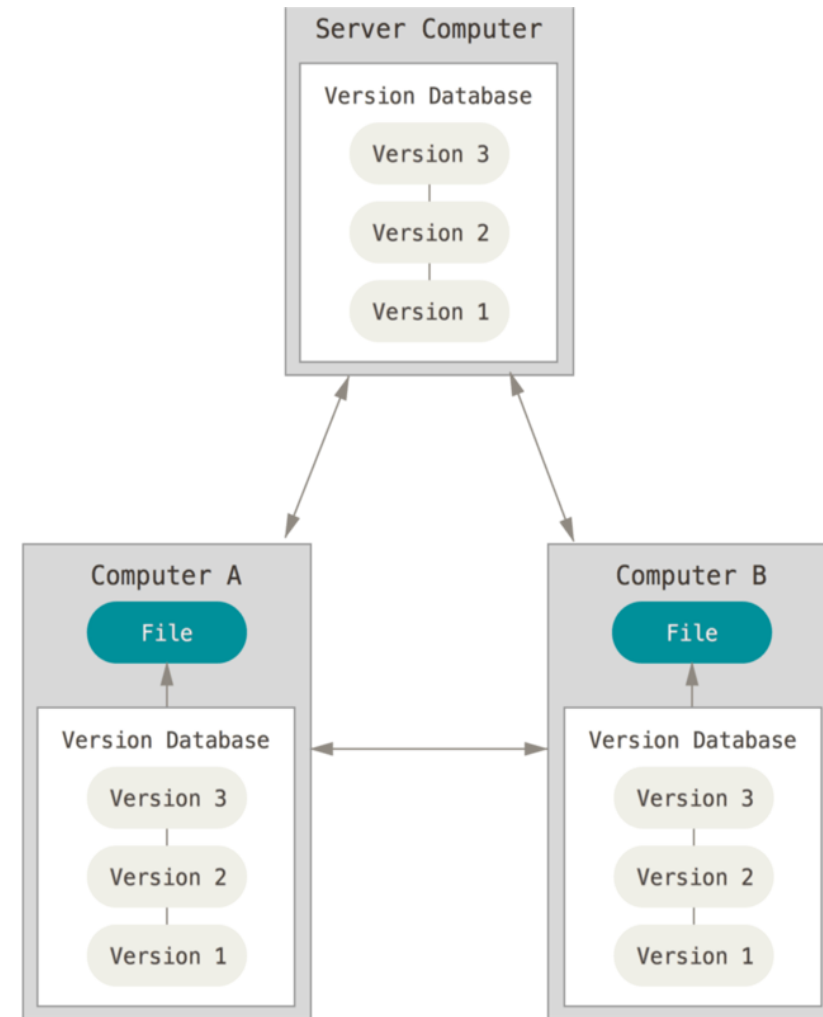
Avantages

- Les systèmes centralisés sont plus faciles à comprendre et utiliser
- Gestion des droits directement au niveau du dossier principal
- Meilleure gestion des fichiers binaires

2 - Types de versionning

- Git, Mercurial ou Bazaar
- Copie Identique sur chaque machine
- Collaboration optimisée
- Gestion des branches et fusion de celles-ci assez facile
- Pas besoin d'être connecté tout le temps
- Bonnes performances (systèmes distribués)

3 – Versionning décentralisé



3 - A propos de Git



- BitKeeper
- Développé en 2005
- Multiplateforme
- Rapide
- Simple
- Robuste
- Distribué

Plan

- Présentation du formateur
- Présentation de git
- **Installation et configuration de GIT**
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

2 - Installation de git sur windows

- Téléchargement de git
<http://git-scm.com/download/win>
- Téléchargement de github for windows
<http://windows.github.com>

2 - Installation de git sur unix

- Téléchargement des sources (image basée sur debian)

```
$ sudo apt install git-all (debian et derrivés)  
$ sudo yum install git (Redhat et dérivés)
```


2 - Configuration de git

- Configuration du fichier .gitconfig

- Windows

Global	C:\Users\username\.gitconfig	.gitconfig
--------	------------------------------	------------

- Linux

Global	~home/<username>/.gitconfig or ~root/.gitconfig	.gitconfig
--------	---	------------

Doc : <https://git-scm.com/book/fr/v2/Personnalisation-de-Git-Configuration-de-Git>

2 - Configuration de git

```
$ git config -- global user.name "John Doe"  
$ git config -- global user.email johndoe@example.com  
$ git config -- global core.editor "code"  
$ git config -- list  
$ git help config
```

QUIZ 1 : Vue d'ensemble de GIT



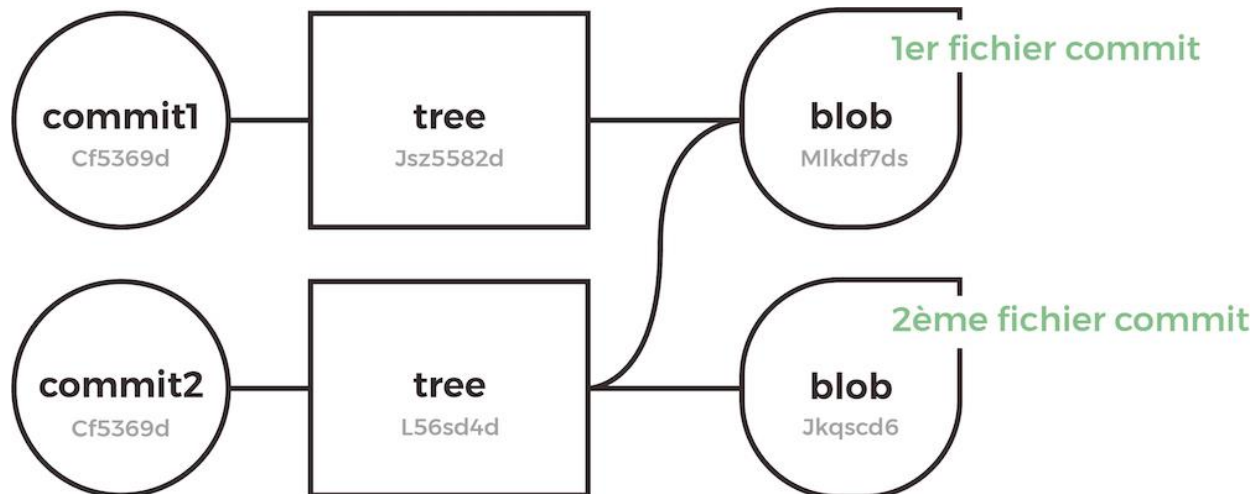
TP-1: Installation de Git

- Installation de Git sur une machine linux
- Installation de Git sur Windows
- Vérifier que git est bien installé
- Configuration de git
- Configuration de visual code ou vi comme éditeur par défaut de git

Plan

- Présentation du formateur
- Présentation de git
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

1 - Le modèle objet GIT



- **blob**

- stockage des données d'un fichier.

- **tree**

- A l'image d'un répertoire
- référence une liste d'autres « tree » et/ou d'autres « blobs »

- **commit**

- Pointeur vers un unique « tree »
- Stockage des méta-informations
 - le timestamp du commit
 - l'auteur du contenu
 - un pointeur vers le (ou les) dernier(s) commit(s), etc.

2 - Les environnements de travail

Working
Directory

- Zone de travail
- **Pas de prise en compte** par git
- Fichiers **juste modifiés**

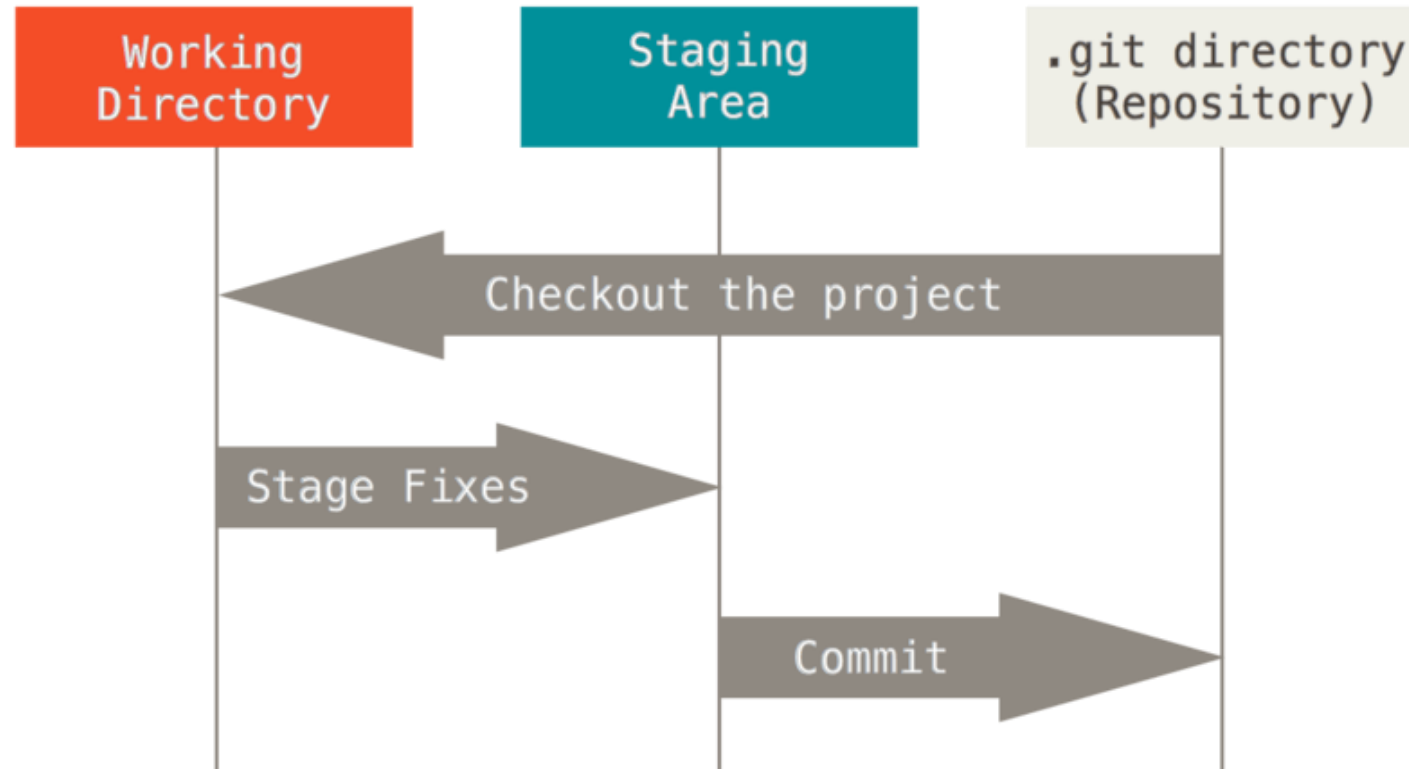
Staging
Area

- Zone « tampon »
- **Annonce** à Git les changements à venir
- Stockage des modifications **avant** le commit
- encore appelé **INDEX**

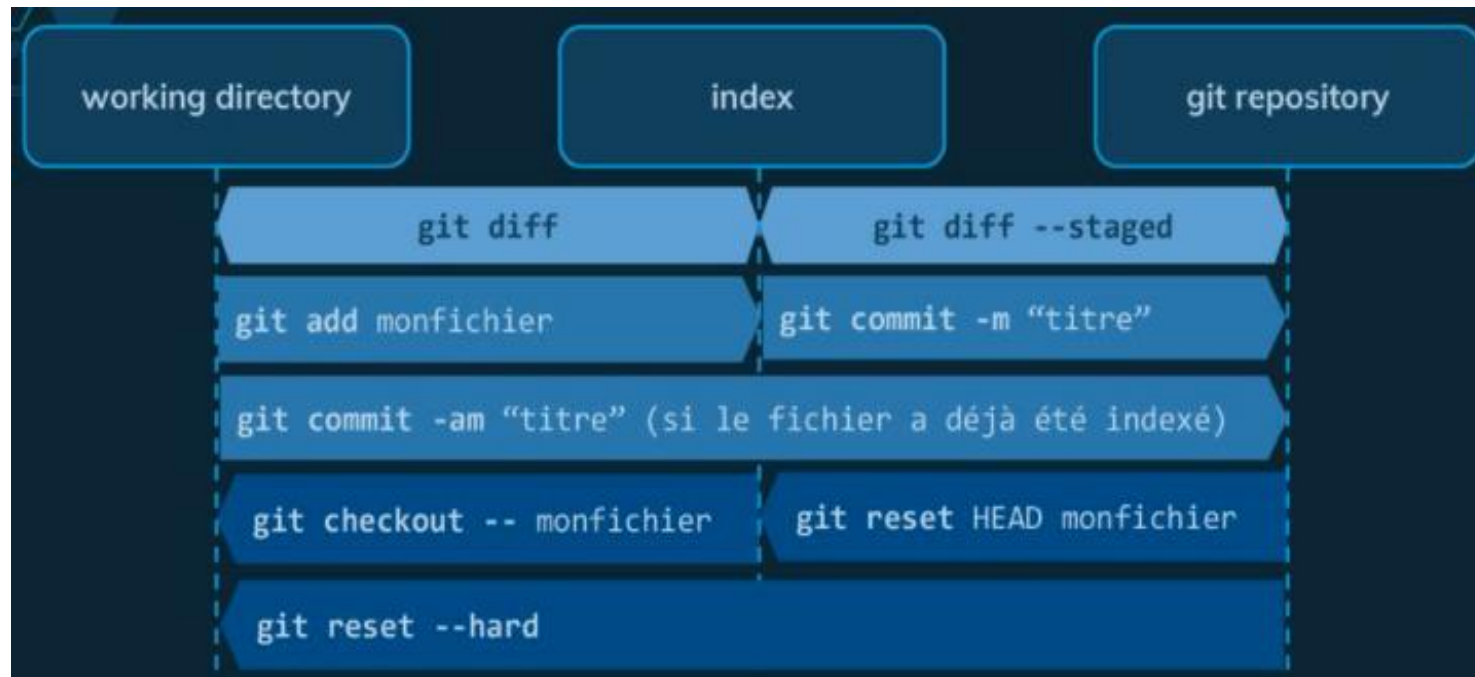
.git directory
(Repository)

- Code **commité** prêt à être envoyé sur le serveur distant

2 - Les environnements de travail



2 - Les environnements de travail



2 - Les environnements de travail

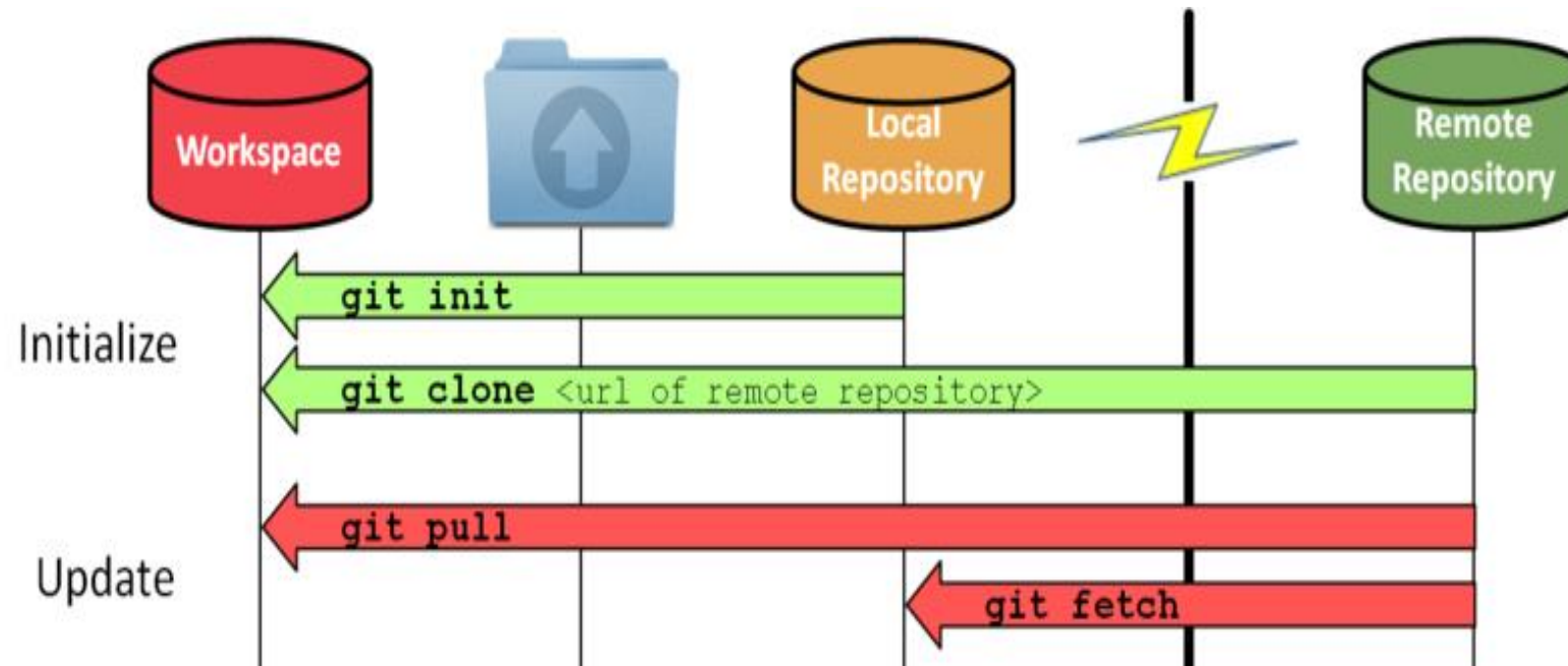
- Ecraser et remplacer un commit

```
git add nom-du-fichier  
git commit --amend
```

- Annuler les modifications apportées à un fichier

```
git checkout – nom-du-fichier  
ou  
git restore nom-du-fichier
```

3 - Création et initialisation d'un repo



3 - Création et initialisation d'un repo

A. Création d'un nouveau répertoire

Création du dossier caché .git
Début du **tracking**
Nouveau repository

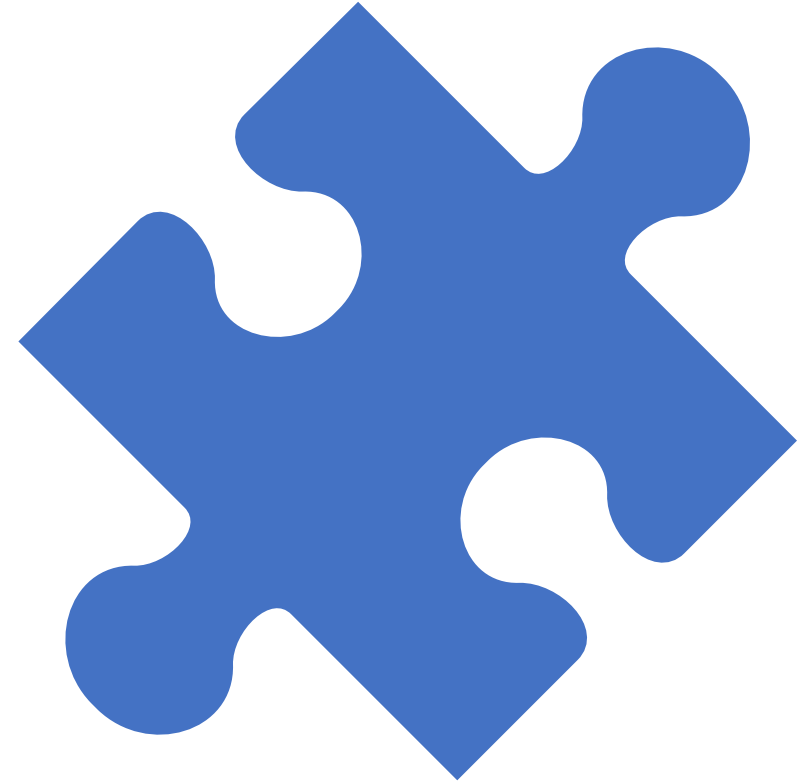
< *git init* >

B. Clonage d'un répertoire distant existant

gitlab
github

< *git clone repo_distant* >

QUIZ 2 : Fondamentaux de Git

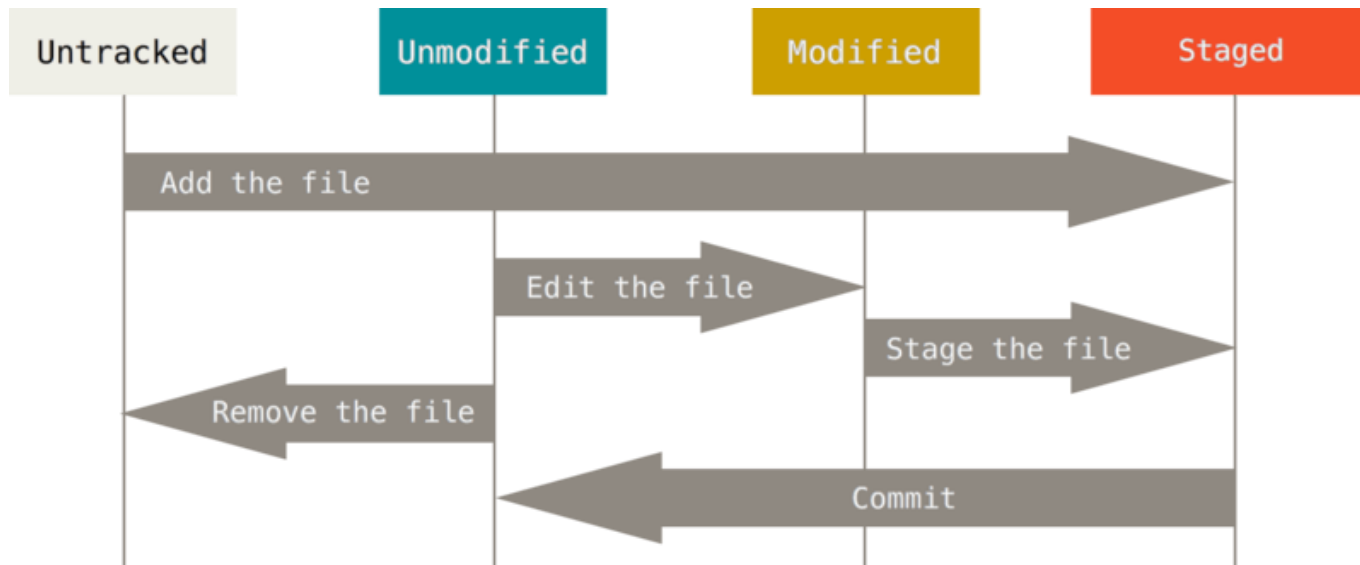


Plan

- Présentation du formateur
- Présentation de git
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

1 - Etat des fichiers

\$ git status



Etat	Modifs	Suivi par git	Stocké dans Index
Untracked	Oui / Non	Non	Non
Unmodified	Non	Oui	Non
Modified	Oui	Oui	Non
Staged	Oui	Oui	Oui

2 - Ignorance de fichiers .gitignore

```
❖ .gitignore
1  .alpackages/
2  .vscode/
3  *.app
4  |
```

Exemples de fichiers à ignorer :

- Logs
- Résultats de build
- Fichiers de configuration locale

3 - Historique des révisions dans git

```
commit 4ca28c40f5b63c24bad37becc0f1c97258fa05ec (HEAD -> logbranch)
Author: Wonder Woman <ww@dccomics.com>
Date: Sat Apr 13 12:21:15 2019 -0400

    add tree.c

commit 48b72fd19d4cd1286d321785fa3e5f42c336c67b
Author: Wonder Woman <ww@dccomics.com>
Date: Sat Apr 13 12:20:55 2019 -0400

    add bird.c

commit 8d046d6b3e9c182f603dcc146a8cb2c8236cce2e
Author: Wonder Woman <ww@dccomics.com>
Date: Sat Apr 13 12:20:41 2019 -0400

    add cat.c

commit 63e4ff822dcea231c319d72a6c81d27a320c9cc0
Author: Wonder Woman <ww@dccomics.com>
Date: Sat Apr 13 12:20:22 2019 -0400

    add dog.c
```

Visualiser l'historique des évènements sur le répertoire git.

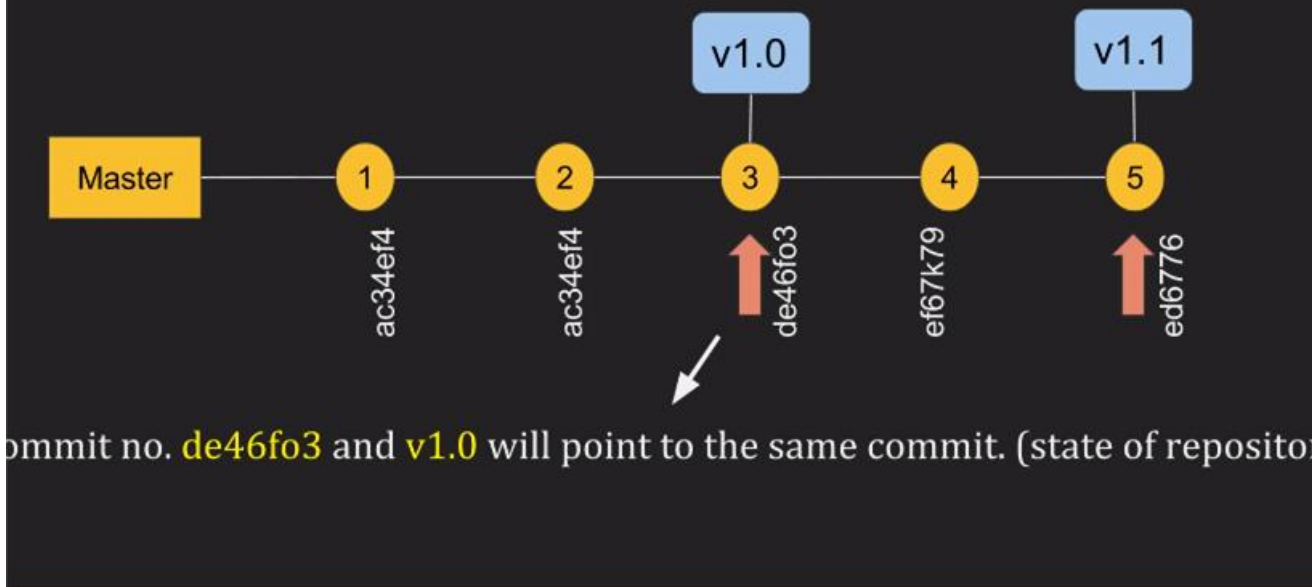
Quelques exemples:

- `git log`
- `git log -p -2`
- `git log --pretty=oneline`
- `git log --graph`
- `git log --since` ex: 2.weeks ou 2019-10-10
- `git log --author`

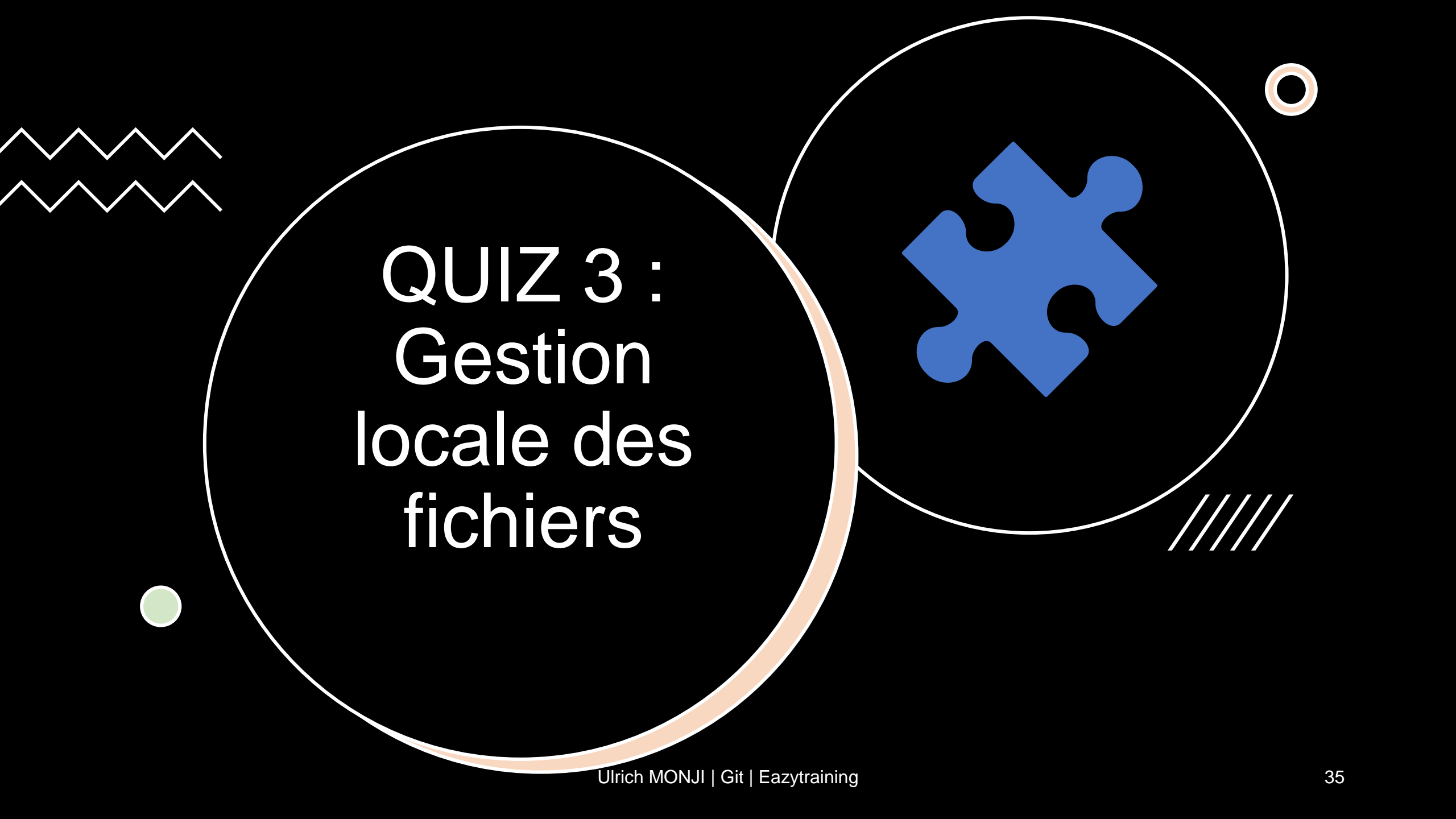
4 - Le tagging dans git

`git tag v1.0` (creates the tag)

`git push tags` (pushes tag to server)



- Release
- Version
- Immuable



QUIZ 3 : Gestion locale des fichiers

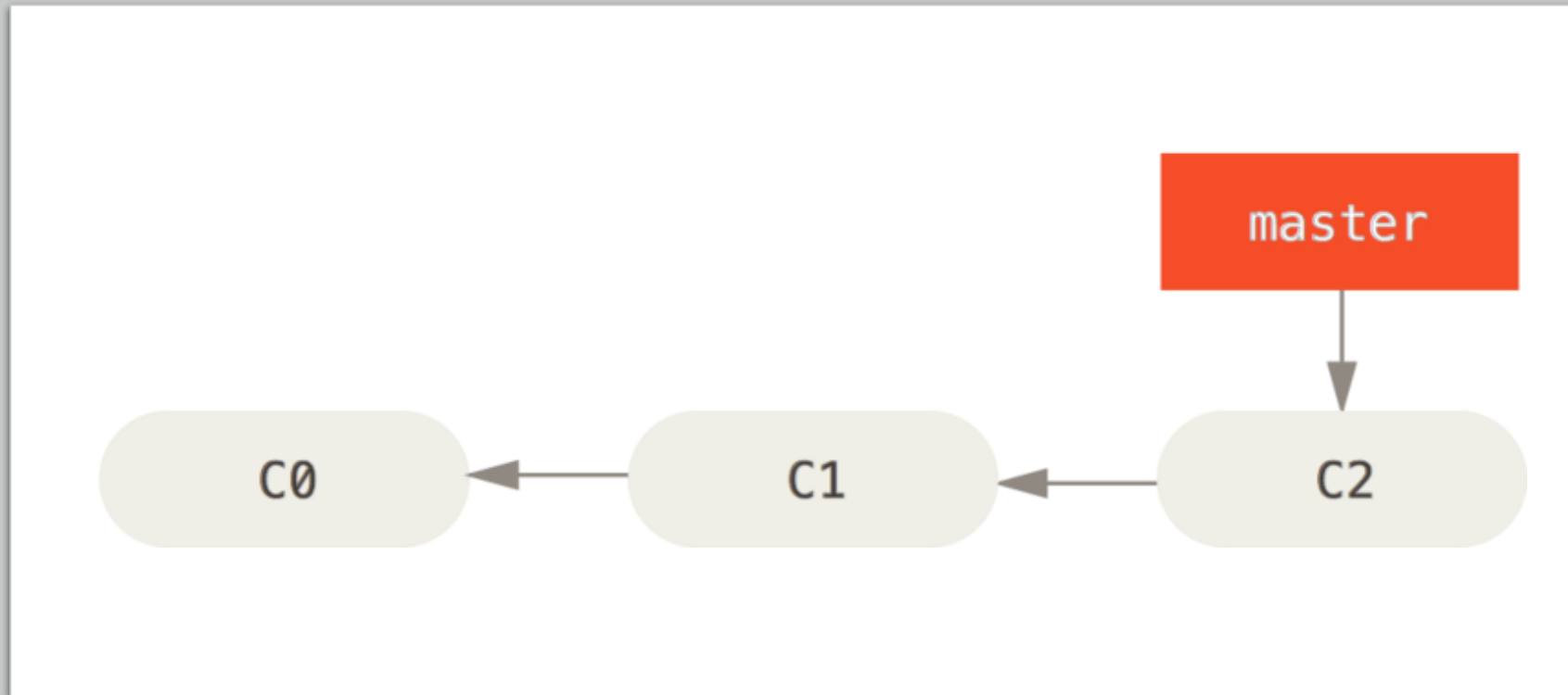


TP-2: Les bases de GIT

- Récupérez le dossier deploy-users dans les fichiers fournis avec les supports de cours
- Copiez le dossier dans votre environnement de travail et rendez-vous dans ce dossier
- Initialisez un repo et faites un commit après avoir mis du contenu dans le fichier README.MD
- Créez un fichier bug.txt et modifiez le fichier README.MD, enfin entérinez la modification en faisant un commit
- Vérifiez la différence entre les 2 derniers commits du fichier README.md
- Créez un fichier gitignore afin de ne pas tracker les fichiers de log (.log) et commit(ez) le changement
- Créez ensuite un fichier website.log et vérifiez bien que le fichier n'est pas traqué
- Listez vos deux derniers commits
- Créez la release 1.0.0 à l'aide des tags
- Créez des deux alias pour que "git st = git status" et "git br = git branch"

Plan

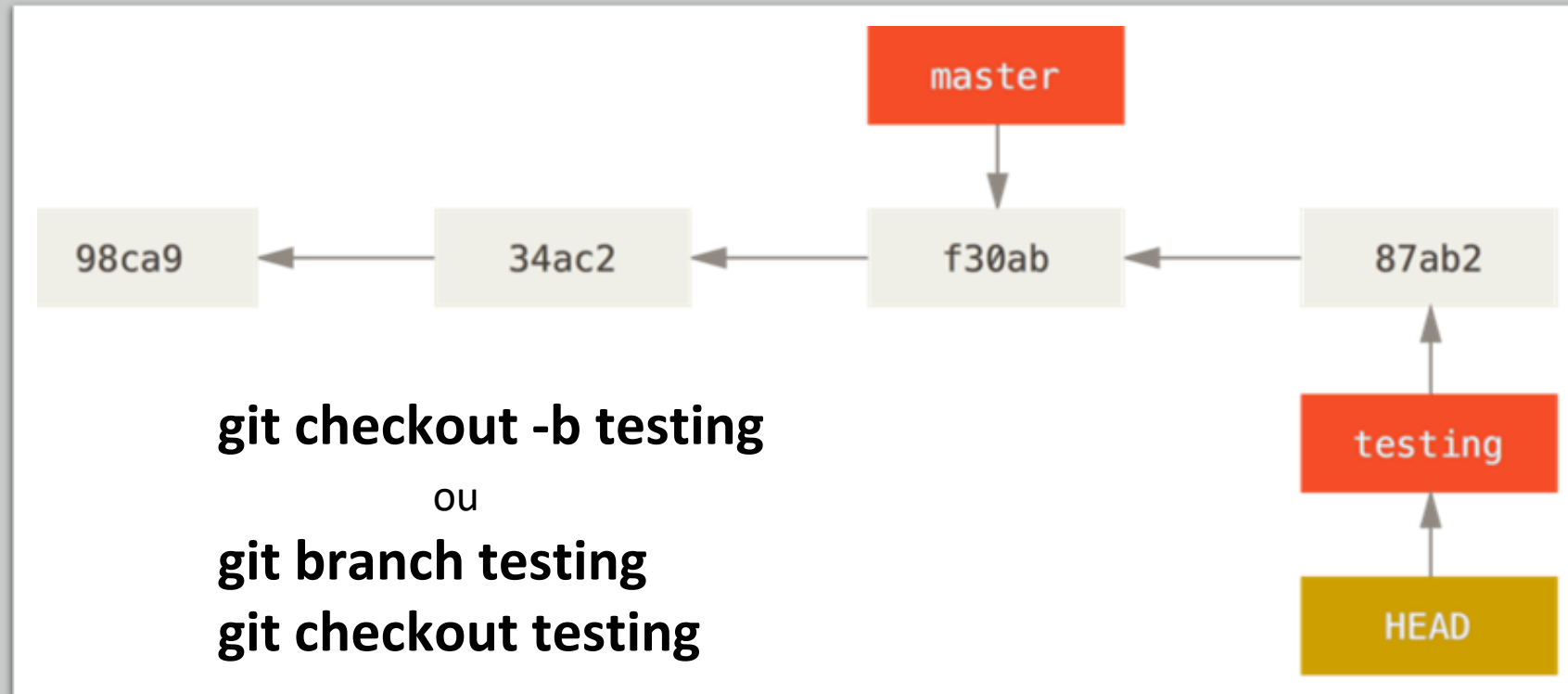
- Présentation du formateur
- Présentation de git
- Installation et configuration de GIT
- Utilisation de git et les fondamentaux
- Gestion locale des fichiers
- Gestion des branches
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github



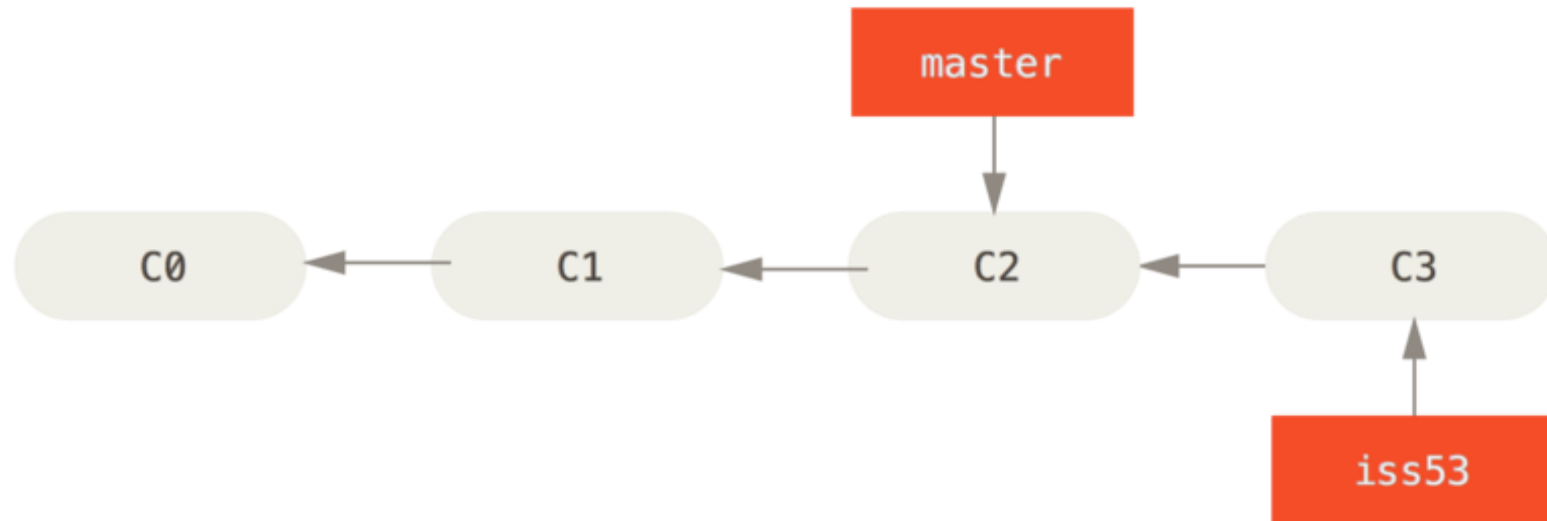
Les branches git (1/5): Définition

- Pointeur commit
- Principale: Master
- Courante: HEAD
- Orientation
- Isolation

git branch « nom_branche »
git checkout « nom_branche »
git checkout -b « nom_branche »



Les branches GIT

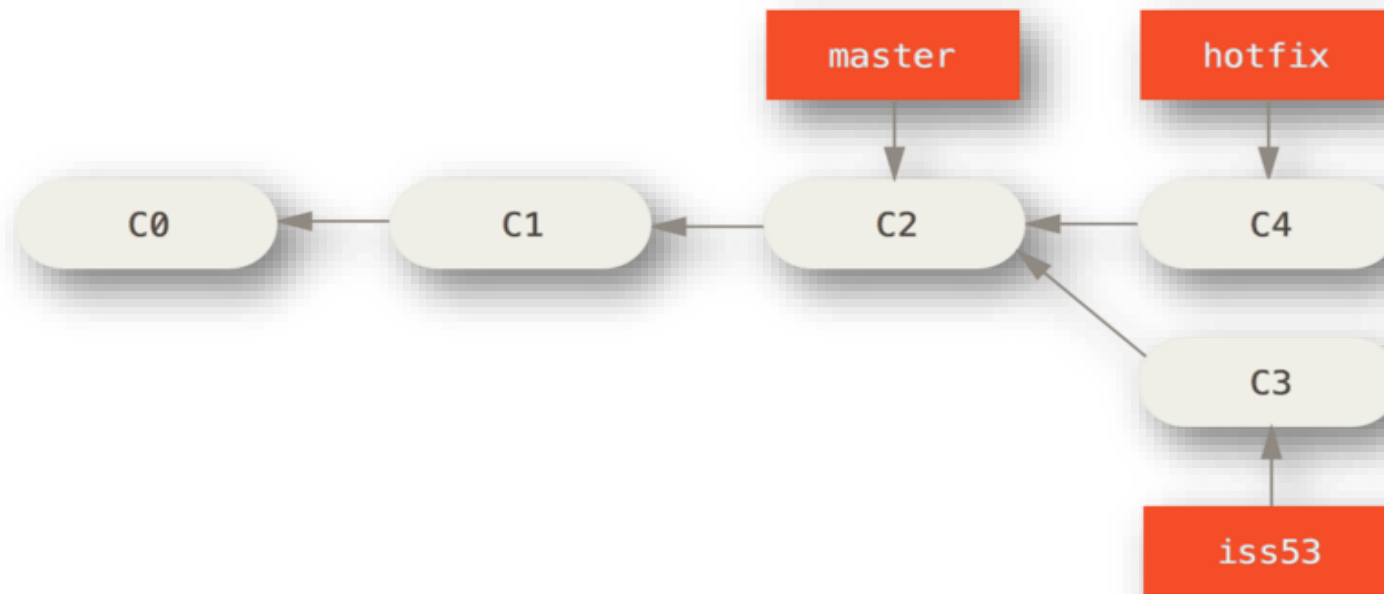


\$ vim index.html

\$ git commit -a -m "ajout d'un pied de page [problème 53]"

Les branches GIT - Définition

« Tout va bien dans le meilleur des mondes, Je teste une nouvelle fonctionnalité 😊 »



Les branches GIT

Définition

```
$ git checkout -b correctif
```

Switched to a new branch 'correctif'

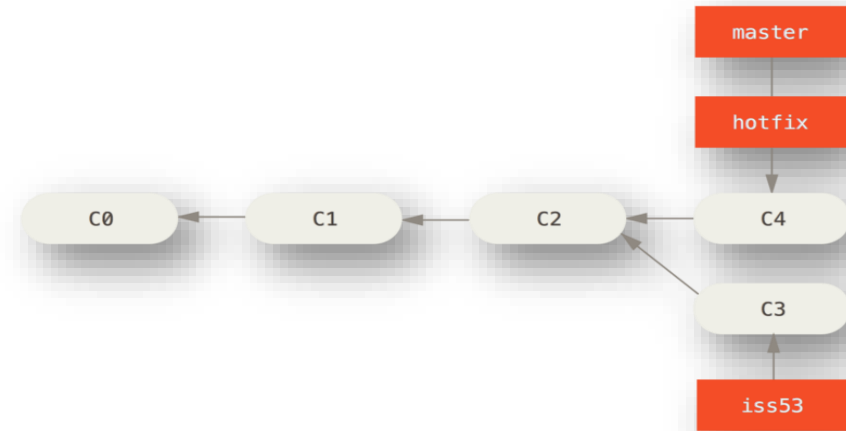
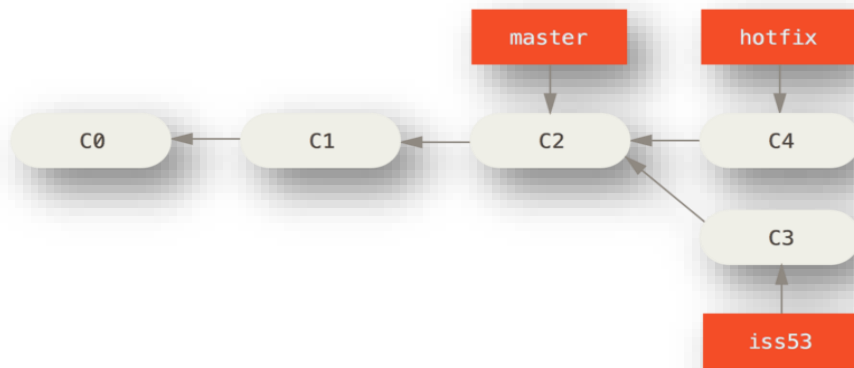
```
$ vim index.html
```

```
$ git commit -a -m "correction de l'adresse email incorrecte"
```

[correctif 1fb7853] "correction de l'adresse email incorrecte"

1 file changed, 2 insertions(+)

1 – Les branches - merge



FAST-FORWARD

\$ git checkout master

\$ git merge correctif

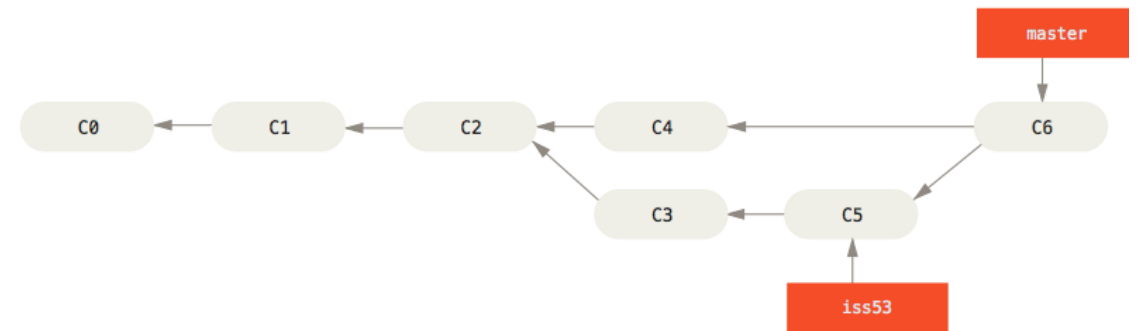
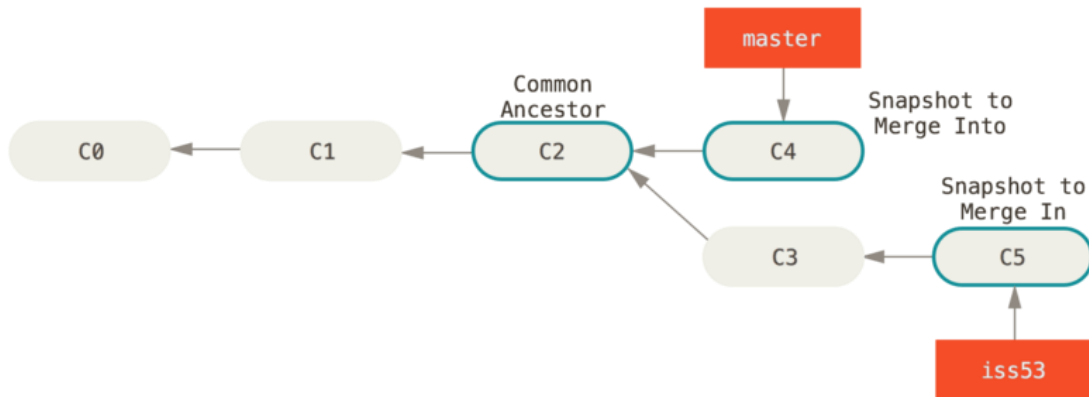
Updating f42c576..3a0874c

Fast-forward

index.html | 2 ++

1 file changed, 2 insertions(+)

1 – Les branches - merge



3-way merge + merge commit

\$ git checkout master

Switched to branch 'master'

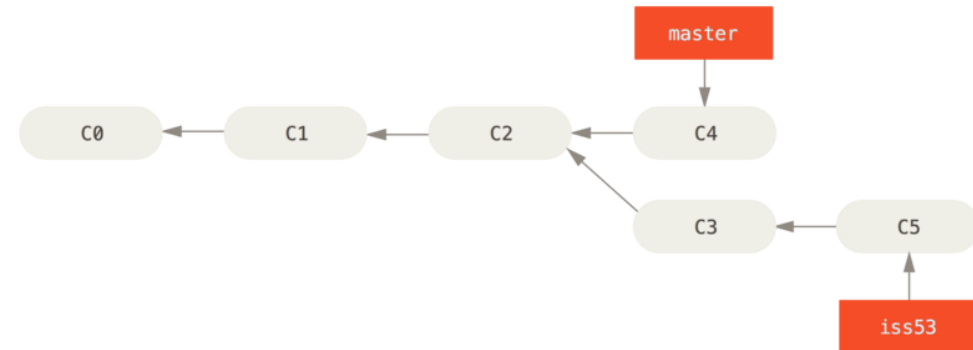
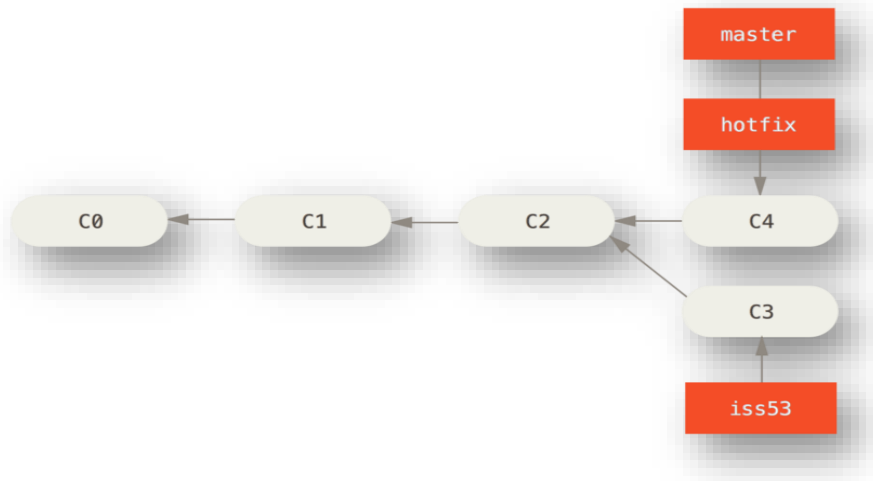
\$ git merge iss53

Merge made by the 'recursive' strategy.

README | 1 +

1 file changed, 1 insertion(+)

2 – Les branches - delete

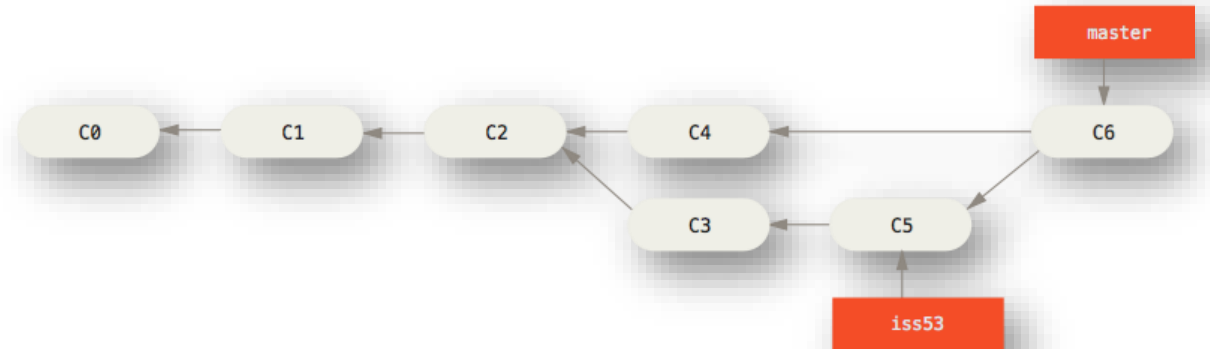
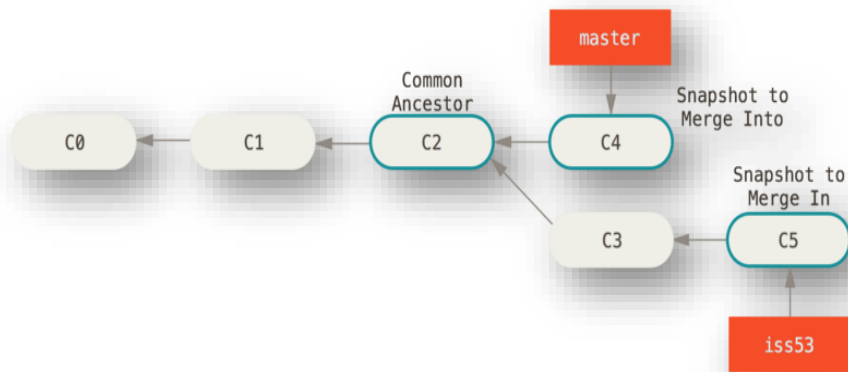


DELETE

\$ git branch -d correctif

Deleted branch correctif (3a0874c).

3 – Les branches - les conflits



```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

3 – Les branches - les conflits

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

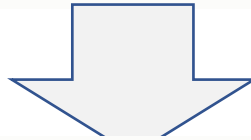
Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:    index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

3 – Les branches - les conflits

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```



```
<div id="footer">
please contact us at email.support@github.com
</div>
```

\$ git add index.html

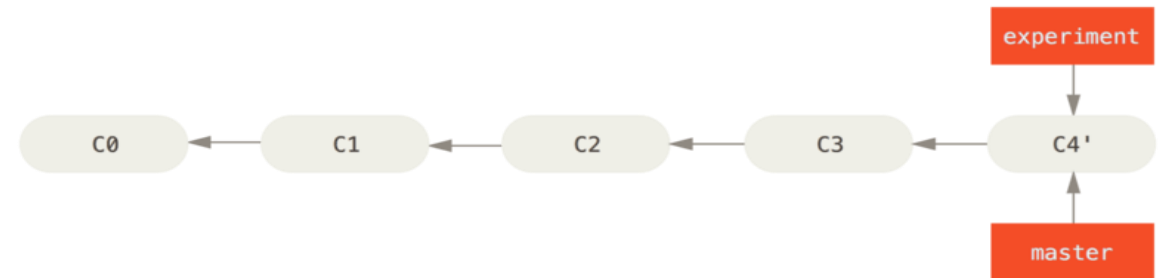
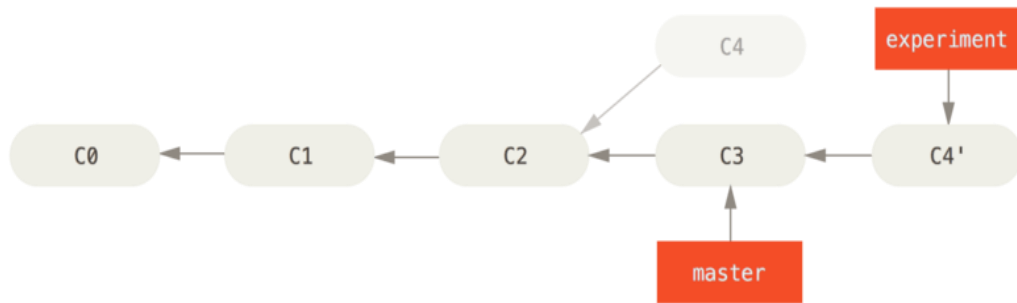
\$ git commit -m « fix merge conflict »

TP-3: Les branches GIT

PARTIE 1

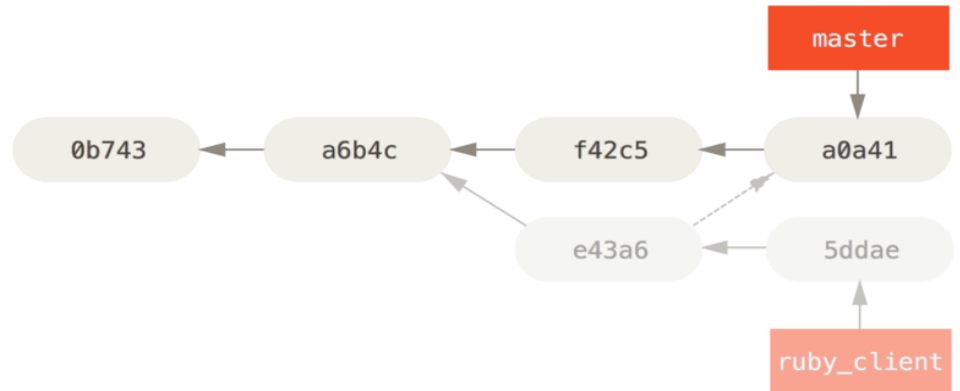
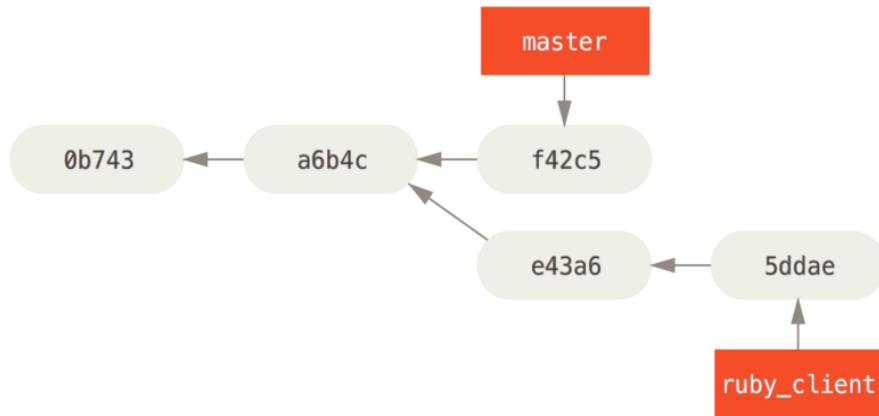
- Rendez-vous dans votre dossier deploy-user
- Créez une branche `update_default_value` dans laquelle vous allez modifier `index.html` pour mettre toto comme `firstname` et titi comme `lastname`, `commit(ez)` le changement
- Mergez la branche `update_default_value` (supprimez la après le merge) avec votre branche `master`
- Créez une branche `password_linux` dans laquelle vous allez modifier le message de demande de mot de passe (dans `linux_users.sh`), `commit(ez)` le changement
- Déplacez-vous sur la branche `master` et modifiez le message de demande de mot de passe (dans `linux_users.sh`), `commit(ez)` le changement
- Tentez de merger la branche `master` avec `password_linux`, que remarquez-vous ? Il y a un conflit
- Réglez le conflit et `commit(ez)` la modification finale

4 – Les branches – rebase (conserver l'historique des commits)



```
$ git checkout experience  
$ git rebase master  
First, rewinding head to replay your work on top of it...  
Applying: added staged command
```

5 – Les branches – cherry-pick



```
$ git cherry-pick e43a6fd3e94888d76779ad79fb568ed180e5fcdcf
Finished one cherry-pick.
[master]: created a0a41a9: "More friendly message when locking the index fails."
3 files changed, 17 insertions(+), 3 deletions(-)
```

TP-3: Les branches GIT

PARTIE 2: Rebase

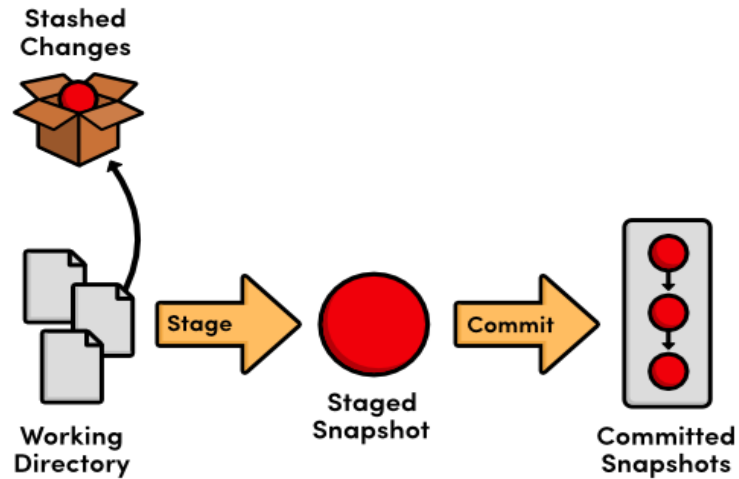
- Rendez-vous dans votre dossier deploy-user
- Créez une branche feature1 dans laquelle vous allez modifier « index.html » pour ajouter un champ input qui prendra un numéro de téléphone, <input type=« text » name=« telephone » value=« 0000000 »> n'oubliez pas de commiter les modifications
- Créez une branche feature2 dans laquelle vous allez modifier le fichier « linux_users.sh » pour ajouter un commentaire quelconque au début du fichier et un autre à la fin. (phrase commençant par #) et commitez.
- Faites un rebase de "feature1" sur la branche "feature2" pour rappatrier les modifications faites dans feature1.
- Faites afficher l'historique de feature2 que remarquez-vous ? Il y a le commit de feature1
- Mergez feature2 sur master
- Effacez feature 1

TP-3: Les branches GIT

PARTIE 3: Cherry-pick

- Rendez-vous dans votre dossier deploy-user
- Créez un branche feature1 dans laquelle vous allez modifier « index.html » modifier la valeur initiale du numero de téléphone, <input type=« text » name=« telephone » value=« 99999999 »> n'oubliez pas de commiter les modifications
- Modifiez « linux_users.sh » et mettez plutôt « Enter very strong password please » et commitez les modifications
- picorez la fonctionnalité qui vous permet de changer le message dans « linux_users.sh » et appliquez la à la master
- Affichez le contenu du fichier « linux_users.sh » et l'historique. que constatez-vous?

6 – Les branches – stash



```
git stash save
# or
git stash
# or with a message
git stash save "this is a message to display on the list"
```

```
git stash apply
# or apply a specific one from out stack
git stash apply stash@{3}
```

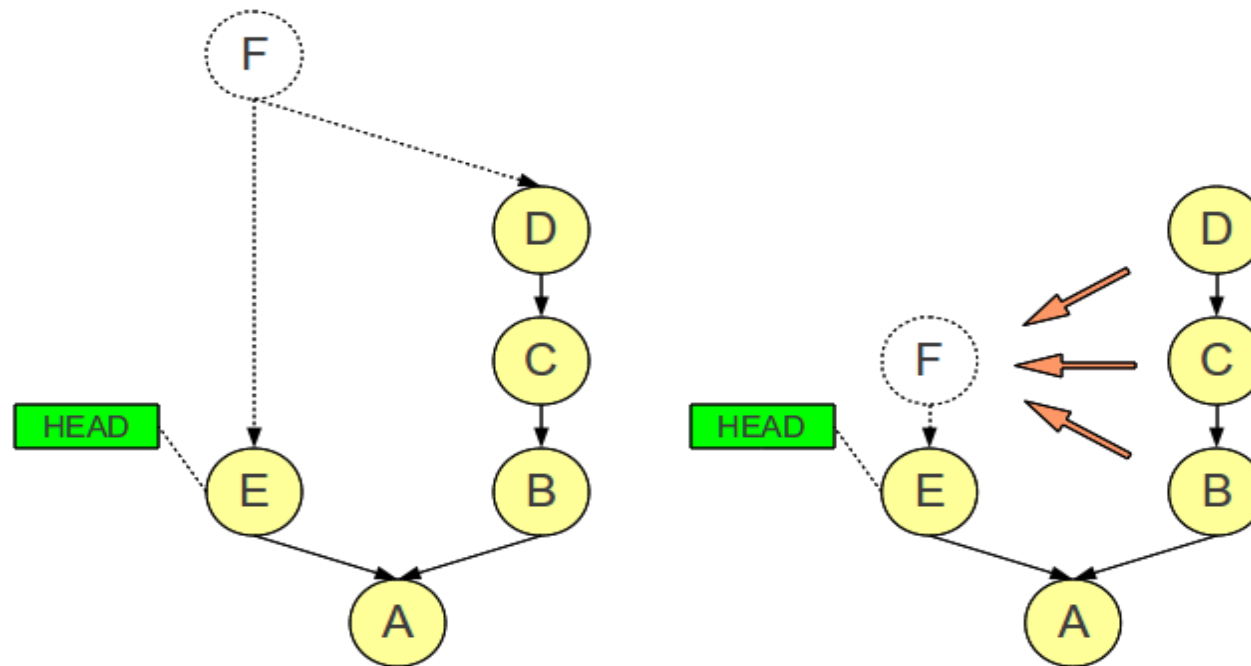
```
# Modify edit_this_file.rb file
git add .

git stash save "Saving changes from edit this file"

git stash list
git status

git stash pop
git stash list
git status
```

7 – Les branches - squash



git rebase -i commit_de_depart

TP-3: Les branches GIT

PARTIE 4: stash

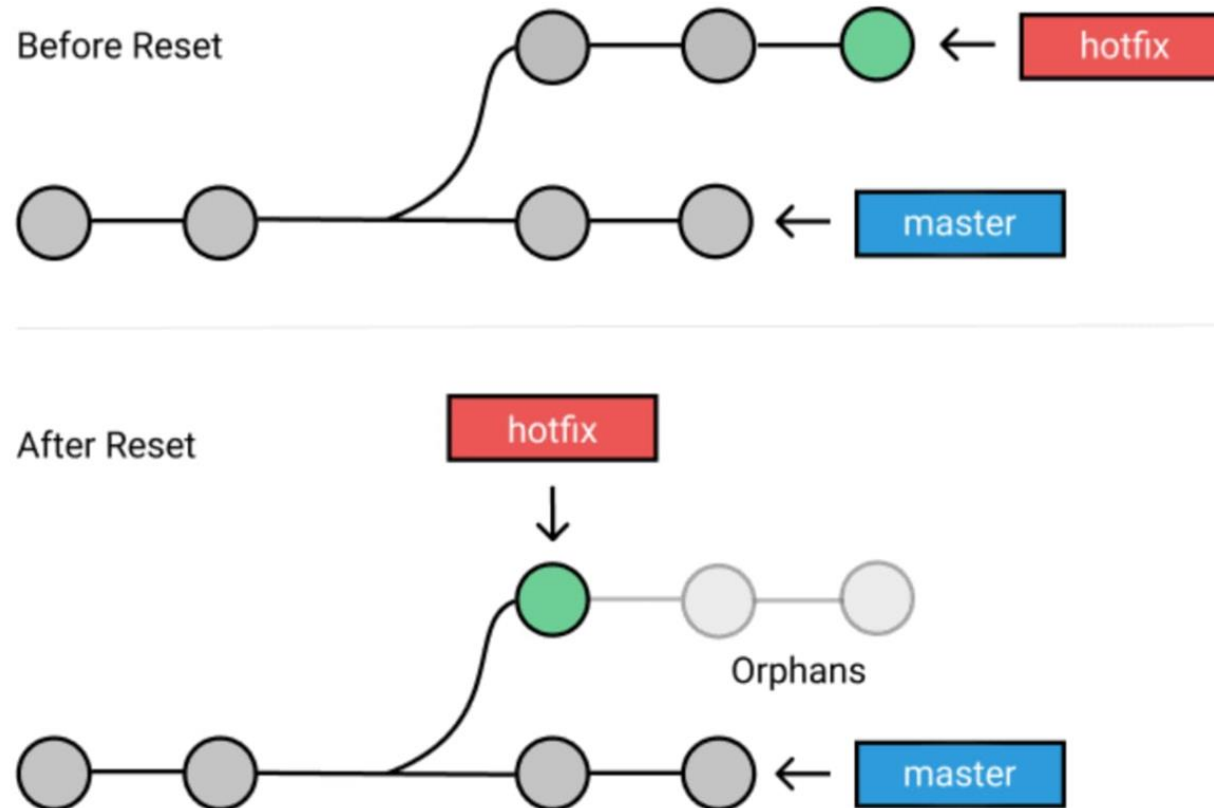
- Rendez-vous dans votre dossier deploy-user
- Créez un branche feature1 dans laquelle vous allez modifier « index.html » modifier la valeur initiale du numero de téléphone, `<input type=« text » name=« telephone » value=« 8888888 »>` ne committez pas car vous êtes loins d'avoir fini quand,
- Un bug en production fait paniquer tout le monde et on nous demande de tout laisser pour passer dessus, "mettez votre travail de côté » avec un stash
- Modifiez le fichier linux et ajoutez « please » dans la phrase du mot de passe et committez vous avez resolu le bug de production
- Récupérez votre stash pour pouvoir reprendre où vous êtes arrêté
- Après l'avoir appliqué, n'oubliez pas de supprimer le stash, de commiter les modifications sur index.html.

TP-3: Les branches GIT

PARTIE 5: Squash

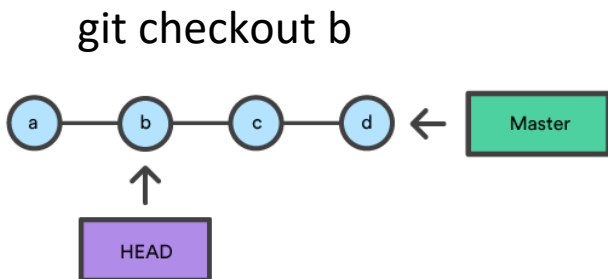
- Rendez-vous dans votre dossier deploy-user
- la branche feature1 a 2 commits que l'on souhaite réécrire en un seul
- utilisez la commande git rebase -i HEAD~2
- remplacer « pick » par « squash » devant le dernier commit
- sauvegardez la modification
- mettre à jour le message de combinaison et fermer vi
- selectionnez la branche master
- mergez feature1 sur master

8 – Les branches - reset

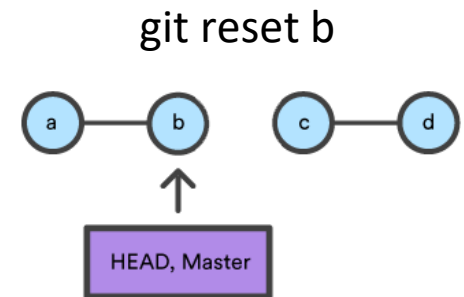


8 – Les branches - reset

Options (Actions du reset sur l'index et le working directory)



- `git reset -- soft`
 - reset HEAD sur un autre commit
 - Aucune action sur Index et working directory
- `git reset -- mixed`
 - Reset HEAD sur un autre commit
 - Reset l'index au niveau du commit
 - Aucune action sur working directory
- `git reset -- hard`
 - Reset tout au niveau du commit



TP-3: Les branches GIT

PARTIE 6: Reset

- Modifier le README et committez
- Autre modification et mise en index
- Git reset –hard HEAD^
- Modification du README et commit
- Modification du README et mise en index
- Modification du README
- Git reset –mixed HEAD^
- Modification du fichier « linux » et commit
- Modification du fichier « linux » et mise en index
- Modification du fichier « linux »
- Git reset –soft HEAD^

9 – Les branches – suivi à distance

« Les références distantes sont des références (pointeurs) vers les éléments de votre dépôt distant tels que les branches, les tags, etc... »

git ls-remote (remote)

git remote show (remote)

Forme:

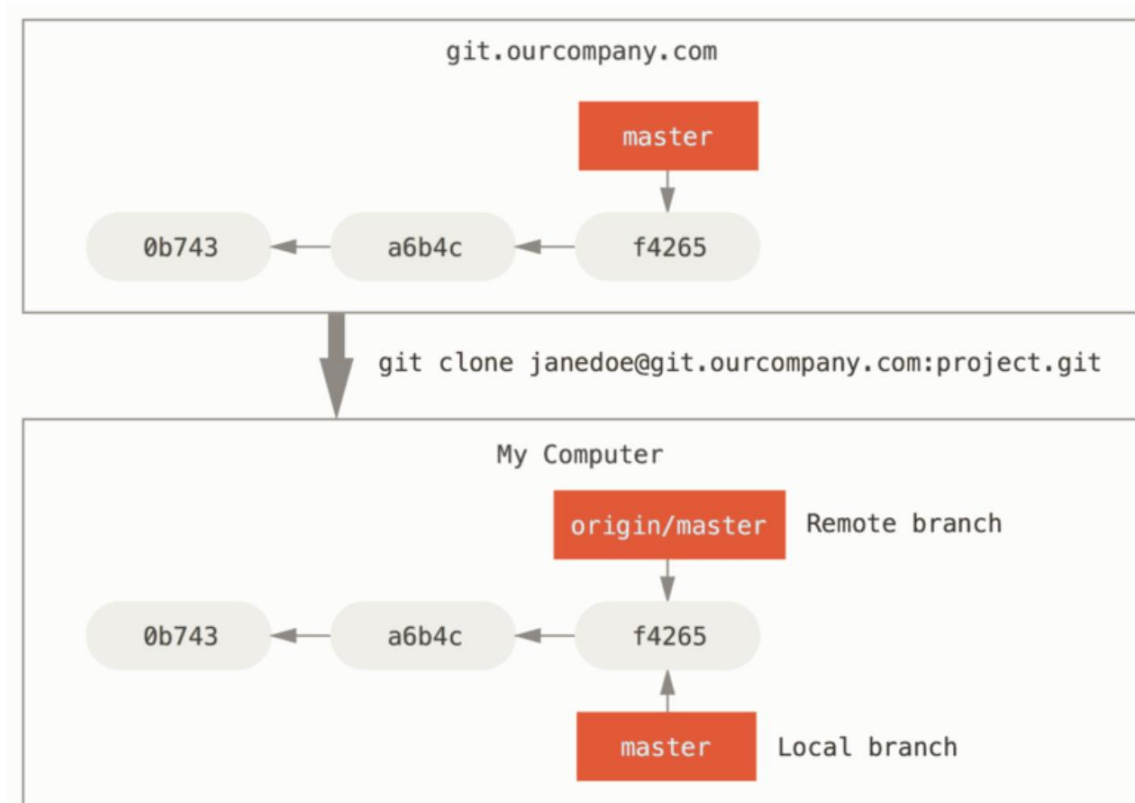
`(distant)/(branche).`

Exemple:

`origin/master`

9 – Les branches – suivi à distance

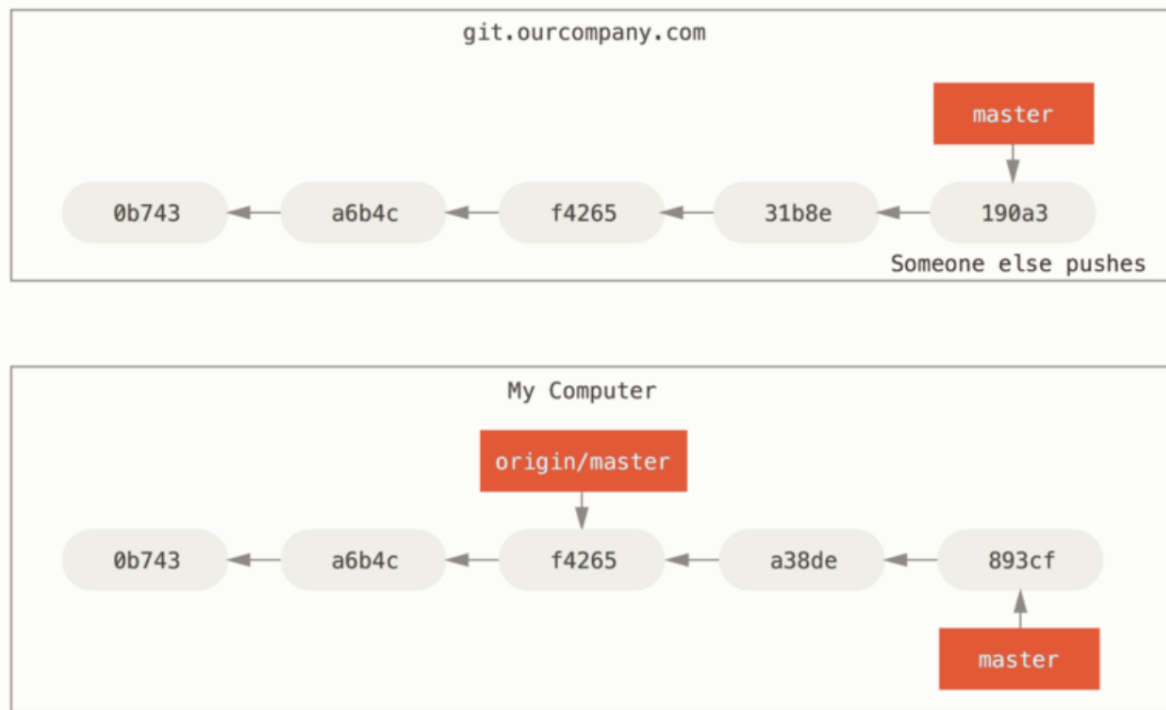
Répertoire distant et local après un clonage :



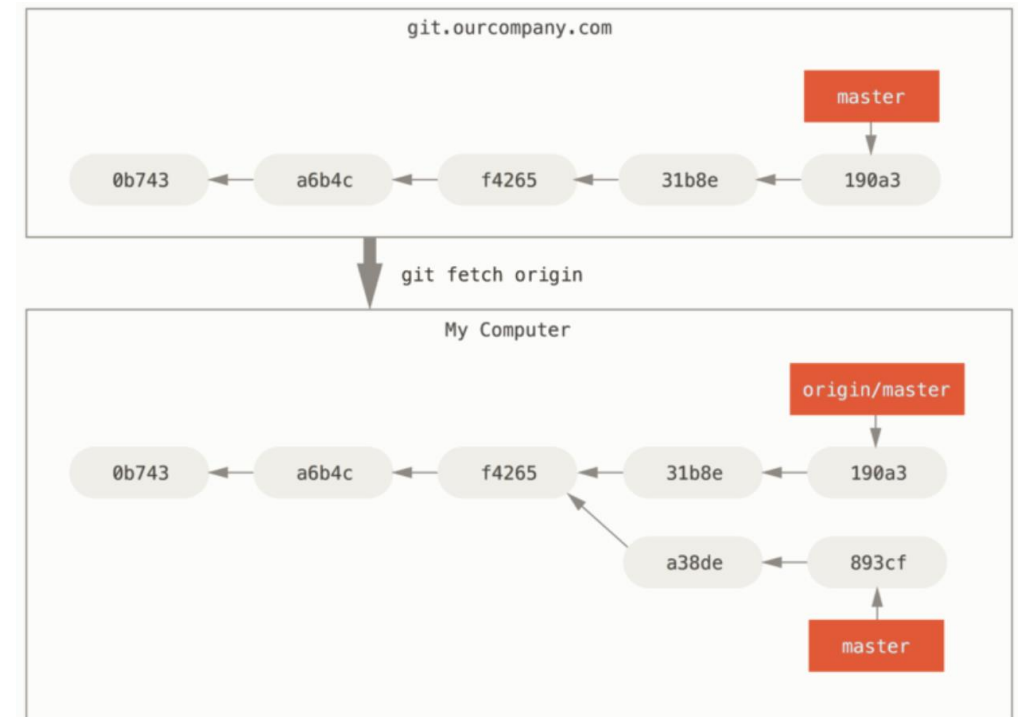
9 – Les branches – suivi à distance

Evolution des branches locale et distante:

Les branches divergent



Récupération du contenu des branches distantes



9 – Les branches – suivi à distance

Evolution des branches locale et distante:

`git push (serveur distant) (branche)`

Pousser mes modifications pour les partager

```
$ git push origin correctionserveur
Counting objects: 24, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (24/24), 1.91 KiB | 0 bytes/s, done.
Total 24 (delta 2), reused 0 (delta 0)
To https://github.com/schacon/simplegit
* [new branch]      correctionserveur -> correctionserveur
```

Récupérer des modifications pour les utiliser

`git merge origin/correctionserveur`

Créer une nouvelle branche à partir d'une branche distante

```
$ git checkout -b correctionserveur origin/correctionserveur
Branch correctionserveur set up to track remote branch correctionserveur from origin.
Switched to a new branch 'correctionserveur'
```

9 – Les branches – suivi à distance

Suivre les branches :

« L'extraction d'une branche locale à partir d'une branche distante crée automatiquement ce qu'on appelle une "branche de suivi" (*tracking branch*) et la branche qu'elle suit est appelée "branche amont" (*upstream branch*) »

```
$ git checkout --track origin/correctionserveur
Branch correctionserveur set up to track remote branch correctionserveur from origin.
Switched to a new branch 'correctionserveur'
```

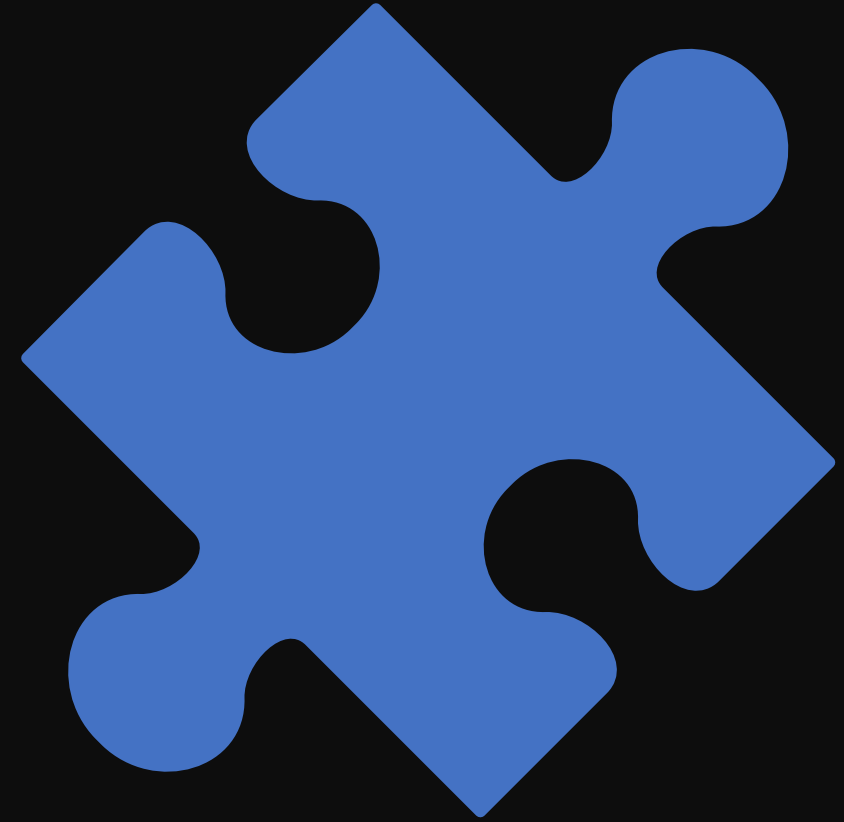
Tirer une branche:

```
git pull = git fetch + git merge
```

Suppression des branches:

```
$ git push origin --delete correctionserveur
To https://github.com/schacon/simplegit
- [deleted]          correctionserveur
```

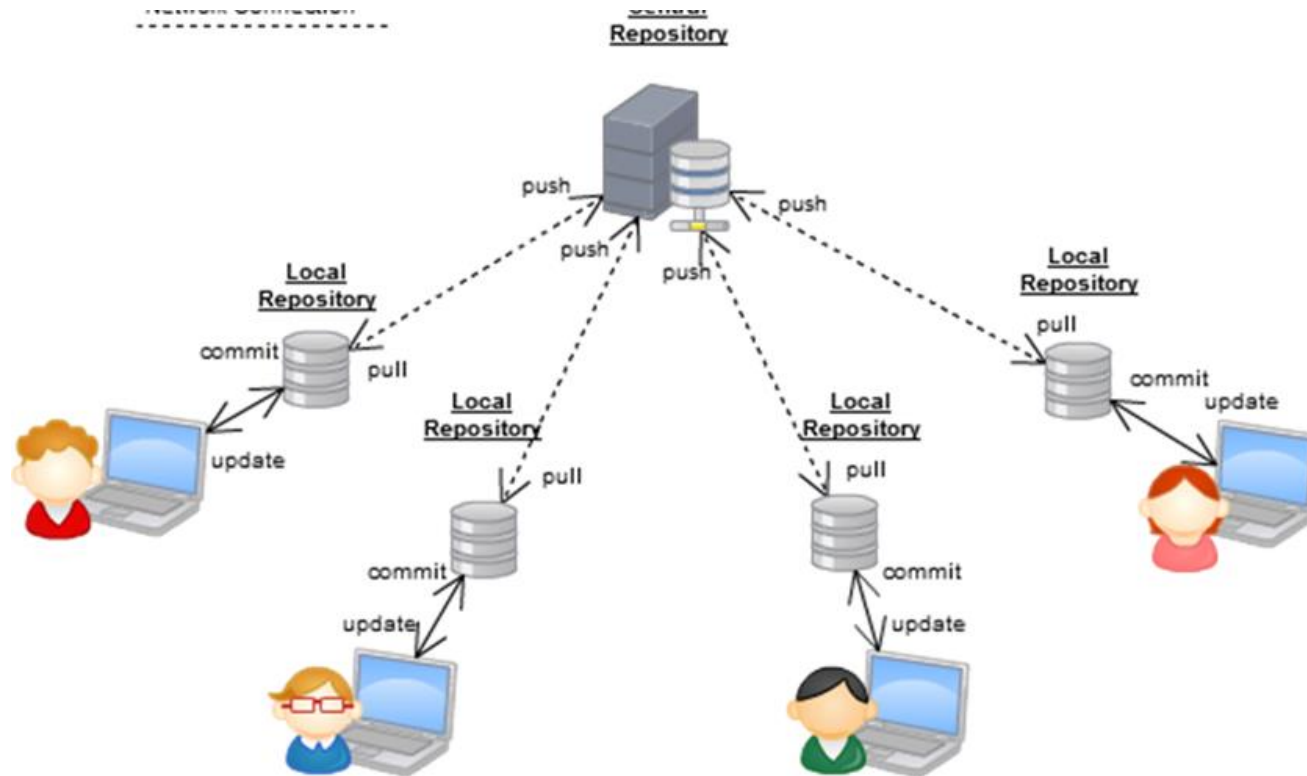

QUIZ 4 : Gestion des branches



Plan

- Présentation du formateur
- Introduction au versioning
- Les bases de GIT
- Les Branches GIT
- **Serveur Git**
- Travail collaboratif
- Quelques outils git
- Github

Serveur Git: Explication

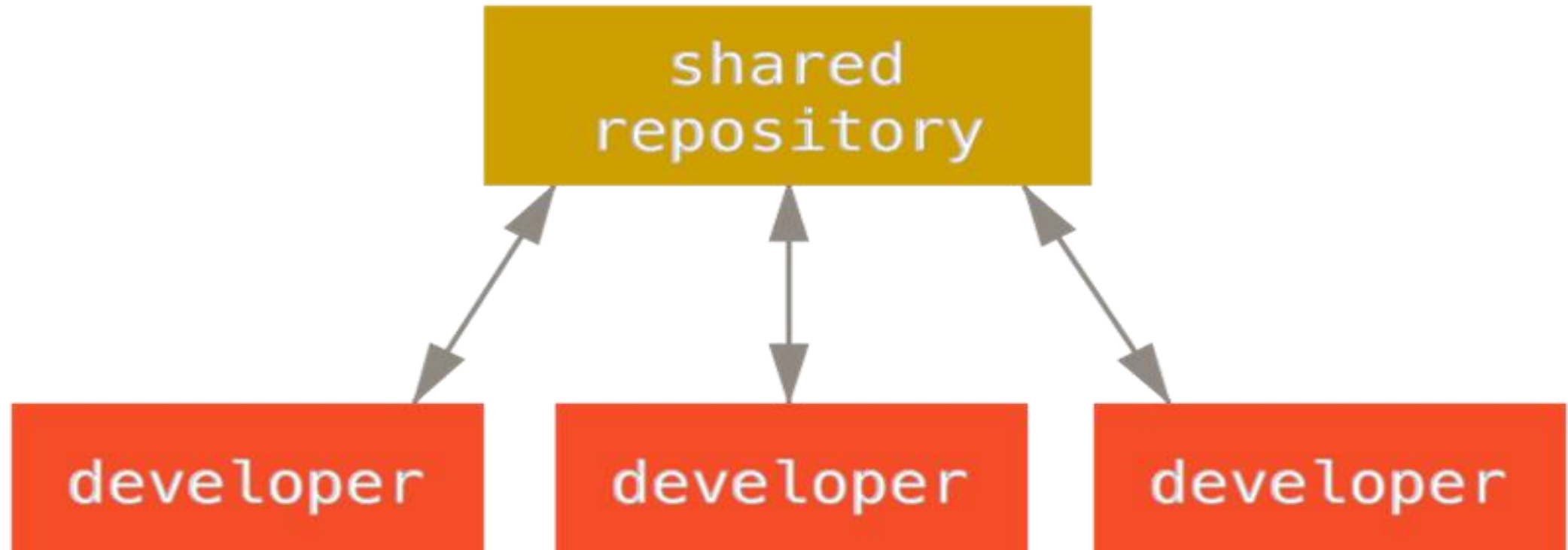


- Centralisation
- Collaboration
- On-premise (gitlab-ce, bitbucket ...) ou Cloud (github, gitlab ...)
- Backup
- Gestion d'accès

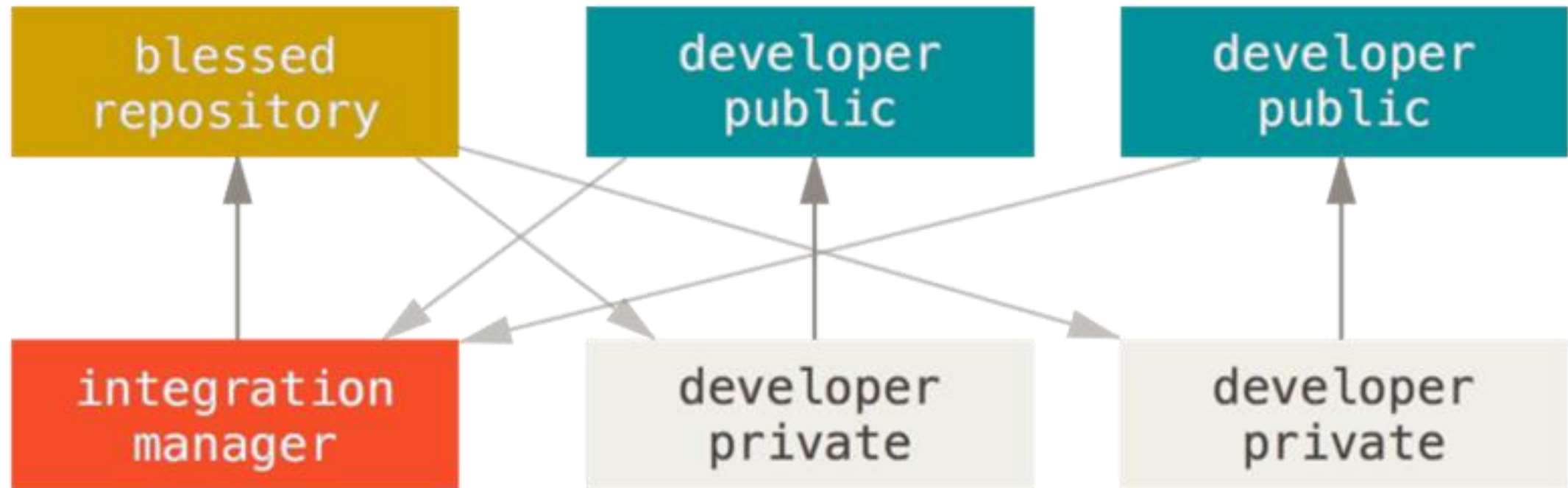
Plan

- Présentation du formateur
- Introduction au versioning
- Les bases de GIT
- Les Branches GIT
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

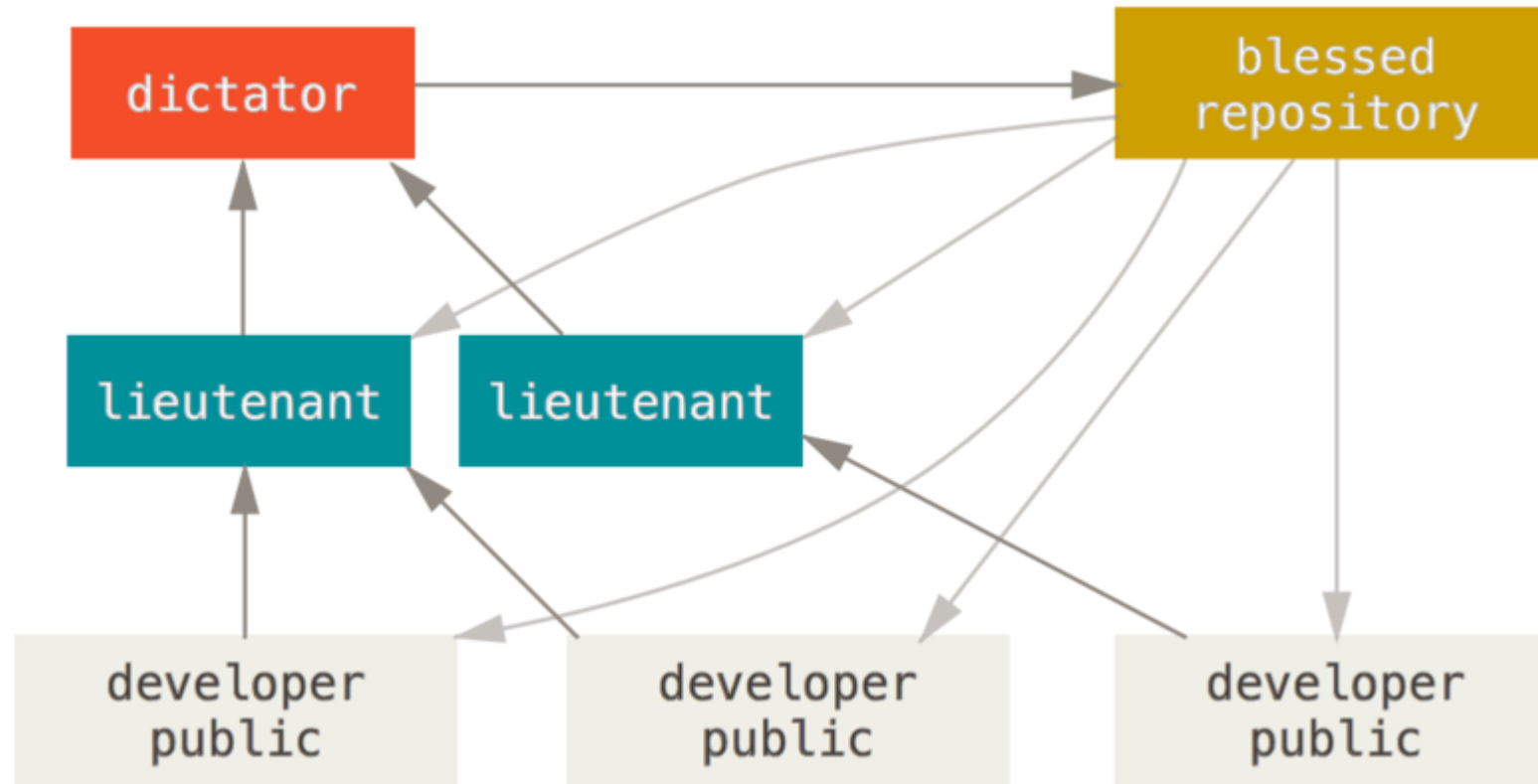
1 - Travail collaboratif: Workflow centralisé



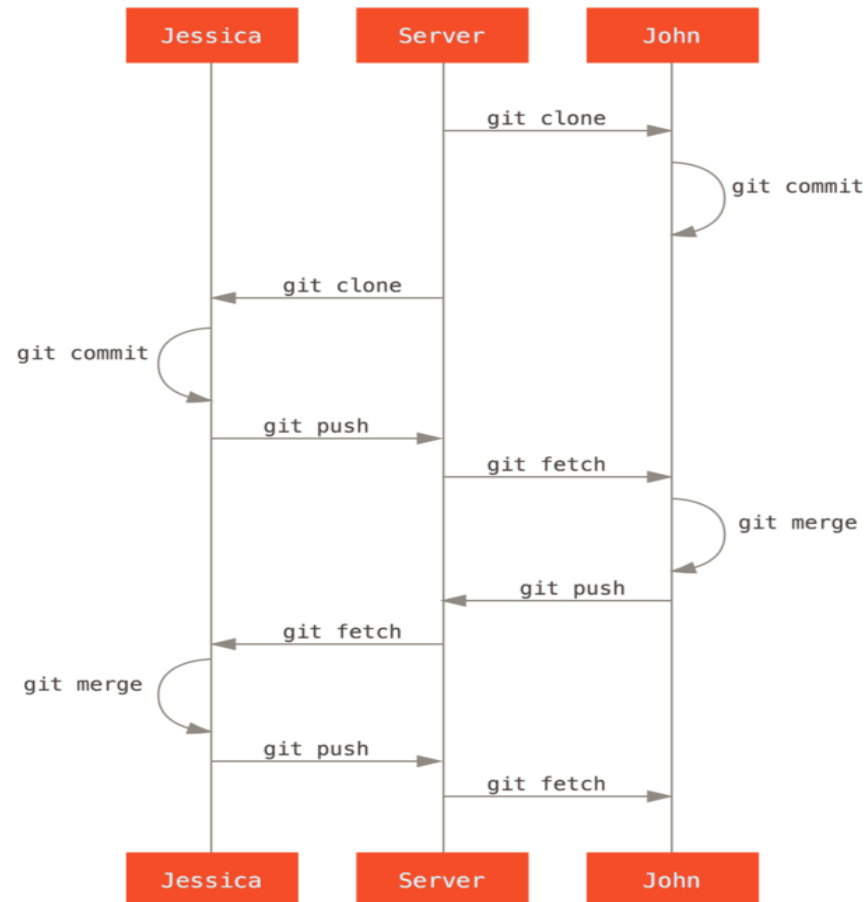
2 - Travail collaboratif: Integration manager

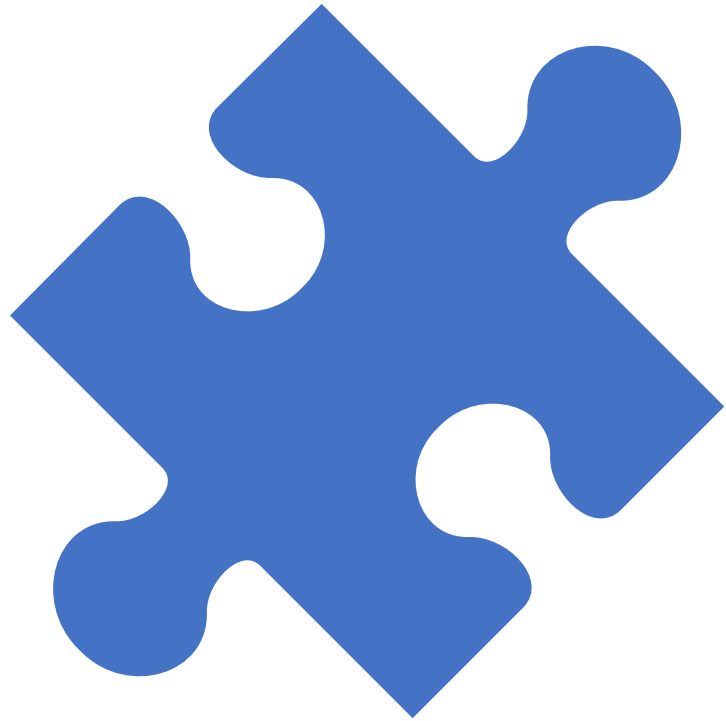


3 - Travail collaboratif: Workflow dictateur et lieutenants (Unix)



4 - Scénario de collaboration





QUIZ 5 : Remote Management

TP-5: Travail collaboratif

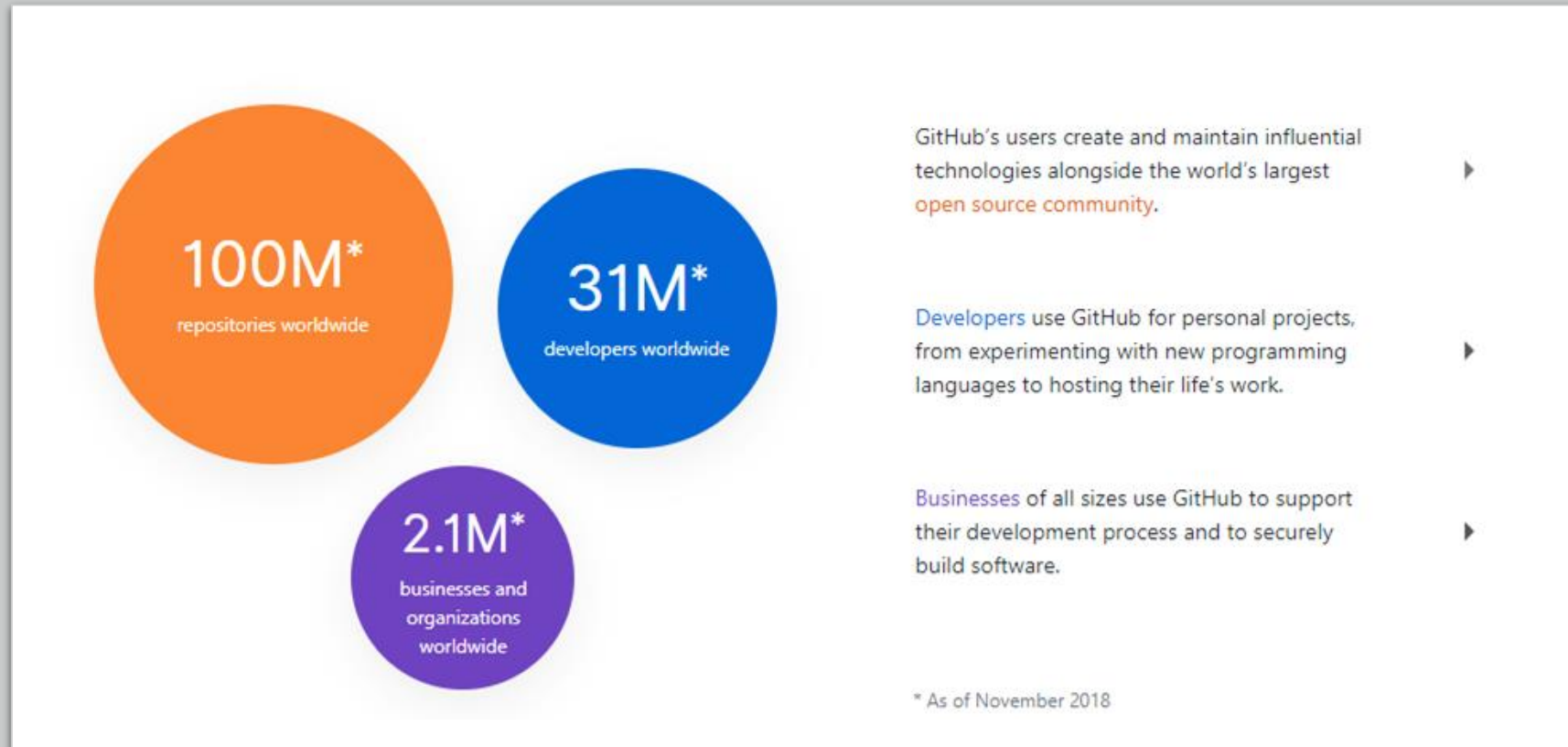
- Un repertoire git distant sera mis à disposition.
- Par paires de contributeurs, il sera question de simuler le scénario de collaboration abordé plus tôt

Plan

- Présentation du formateur
- Introduction au versioning
- Les bases de GIT
- Les Branches GIT
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github

Plan

- Présentation du formateur
- Introduction au versioning
- Les bases de GIT
- Les Branches GIT
- Serveur Git
- Travail collaboratif
- Quelques outils git
- Github



Github (1/3): Quelques chiffres

Individuals

Free

\$0

Per month

The basics of GitHub for every developer

- Unlimited public repositories
- Unlimited private repositories **NEW**
- 3 collaborators for private repositories
- Issues and bug tracking
- Project management

Included alongside other real-world development tools in the [GitHub Student Developer Pack](#)

Choose Free

Pro


\$7

Per month

Pro tools for developers with advanced requirements

- Unlimited public repositories
- Unlimited private repositories
- Unlimited collaborators
- Issues and bug tracking
- Project management
- [Advanced tools and insights](#)

Choose Pro



Teams

Team

\$9

Per user / month

Advanced collaboration and management tools for teams

- Unlimited public repositories
- Unlimited private repositories
- Unlimited collaborators
- Team access controls
- User management and billing
- Issues and bug tracking
- Project management
- [Advanced tools and insights](#)

Starts at **\$25 / month** and includes your first 5 users

Free for [academic faculty](#) for teaching or non-profit research

Enterprise

\$21

Per user / month

Security, compliance, and deployment controls for organizations

- Everything included in Team
- Self-hosted or cloud-hosted
- SAML single sign-on
- Access provisioning
- Simplified account administration
- Unified search and contributions
- Priority support
- 99.95% uptime SLA for Enterprise Cloud
- Invoice billing
- Advanced auditing

Questions?
→ [Contact Sales](#)

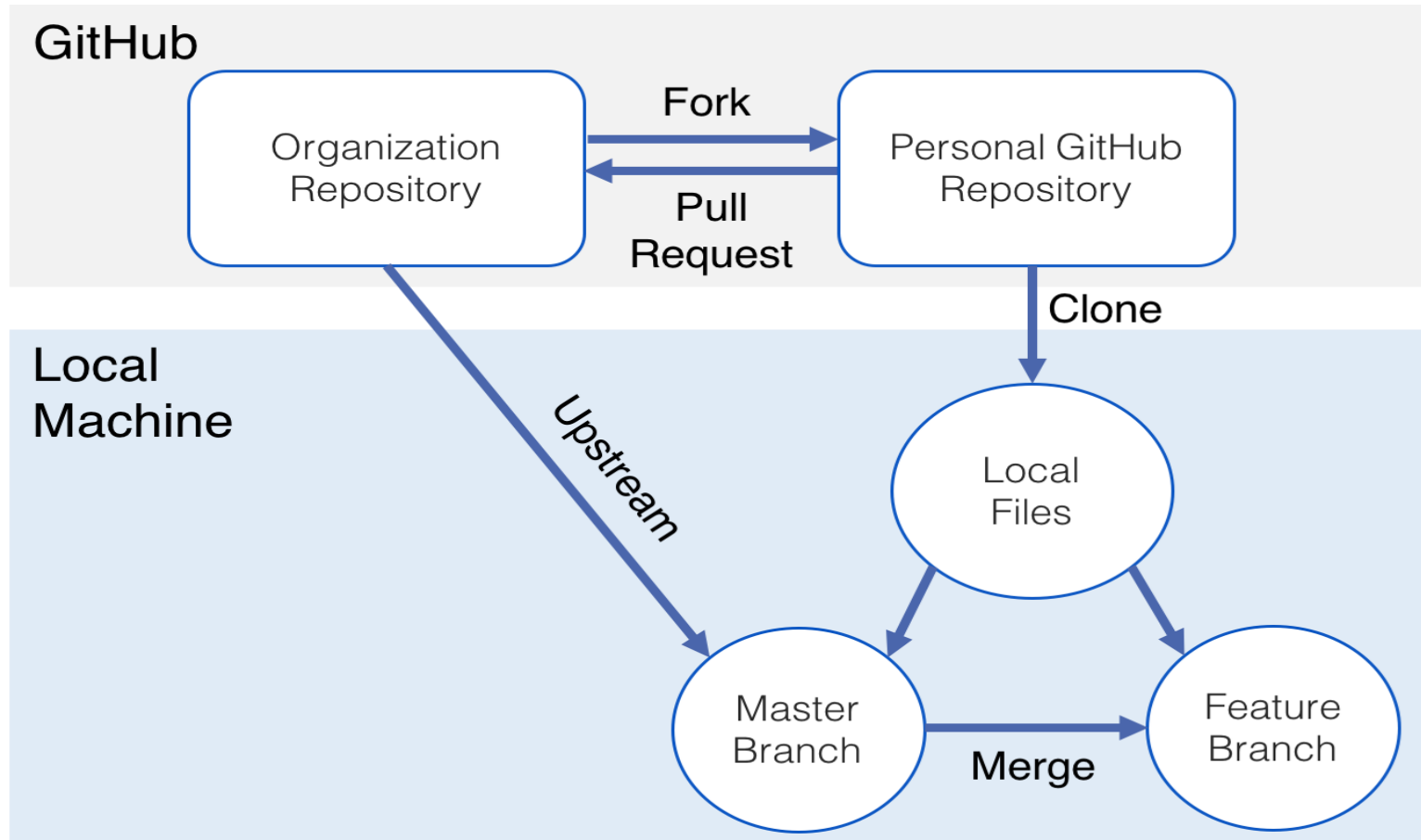
Free for educational institutions participating in the [GitHub Education](#) program

Github (2/3): Plans

Ulrich MONJI | Git | Eazytraining

78

Github (3/3): Contribution à un projet



TP-6: Github

- Créez un compte github
- Suivez le workflow de contribution pour faire une PR au repo <https://github.com/eazytrainingfr/webapp.git>
- Comme modification que vous apporterez, rajouter juste votre prenom dans le fichier README.md

Merci pour votre attention