Bonnes pratiques & Git

David Beauchemin, BSc.

.Layer, Université Laval, CRDM, GRAAL

21 juin 2019

UNIVERSITÉ LAVAL

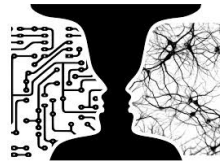


Figure – https://xkcd.com/1597/

Agenda

À propos

- Membres de .Layer, une communauté ayant comme mission de promouvoir la collaboration et le partage de connaissances dans le domaine de la science des données.
- Organisation de conférence et d'atelier.

dotlayer.org

: MeetupMLQuebec

Objectifs

- Développer des bonnes pratiques de travail.
- Apprendre à utiliser l'invite de commande.
- Apprendre à utiliser les commandes de bases de Git.
- Apprendre à utiliser les commandes plus avancées de Git.

Agenda

Ligne de commande

Pourquoi utiliser l'invite de commande ?

- Permet plus de flexibilité.

Ligne de commande

Pourquoi utiliser l'invite de commande ?

- Permet plus de flexibilité.
- Permet de comprendre ce que l'on fait.

Ligne de commande

Pourquoi utiliser l'invite de commande ?

- Permet plus de flexibilité.
- Permet de comprendre ce que l'on fait.
- Permet d'en pratiquer l'utilisation.

Ligne de commande

Pourquoi utiliser l'invite de commande ?

- Permet plus de flexibilité.
- Permet de comprendre ce que l'on fait.
- Permet d'en pratiquer l'utilisation.
- Permet d'avoir l'air plus *élite*.

Astuces

- Tab est votre amie!

Astuces

- Tab est votre amie!
- `ctrl+r` pour faire une recherche dans les commandes antérieures.

Astuces

- Tab est votre amie !
- `ctrl+r` pour faire une recherche dans les commandes antérieures.
- ↑↓ pour naviguer dans les dernières commandes.

Astuces

- Tab est votre amie !
- `ctrl+r` pour faire une recherche dans les commandes antérieures.
- ↑↓ pour naviguer dans les dernières commandes.
- `man` pour afficher le manuel d'une fonction.

ls/la/ll

- `ls` Permet d'afficher tous les fichiers et répertoires visibles.

ls/la/ll

- `ls` Permet d'afficher tous les fichiers et répertoires visibles.
- `la` Agit comme `ls` mais affiche en plus les fichiers et répertoires cachés.

ls/la/ll

- `ls` Permet d'afficher tous les fichiers et répertoires visibles.
- `la` Agit comme `ls` mais affiche en plus les fichiers et répertoires cachés.
- `ll` permet d'afficher la taille des fichiers et répertoires visibles.

cd & pwd

- `cd dir_name` permet de se déplacer dans un répertoire.

cd & pwd

- `cd dir_name` permet de se déplacer dans un répertoire.
- `cd ..` permet de remonter d'un répertoire.

cd & pwd

- `cd dir_name` permet de se déplacer dans un répertoire.
- `cd ..` permet de remonter d'un répertoire.
- `cd` permet de remonter à la racine.

cd & pwd

- `cd dir_name` permet de se déplacer dans un répertoire.
- `cd ..` permet de remonter d'un répertoire.
- `cd` permet de remonter à la racine.
- `pwd` permet de connaitre son emplacement actuel.

mkdir & rm

- `mkdir` permet de créer un répertoire.

mkdir & rm

- `mkdir` permet de créer un répertoire.
- `touch` permet de créer un fichier vide.

mkdir & rm

- `mkdir` permet de créer un répertoire.
- `touch` permet de créer un fichier vide.
- `mkdir -p` permet de créer un répertoire et ses parents.

mkdir & rm

- `mkdir` permet de créer un répertoire.
- `touch` permet de créer un fichier vide.
- `mkdir -p` permet de créer un répertoire et ses parents.
- `rm` permet de supprimer un fichier.

mkdir & rm

- `mkdir` permet de créer un répertoire.
- `touch` permet de créer un fichier vide.
- `mkdir -p` permet de créer un répertoire et ses parents.
- `rm` permet de supprimer un fichier.
- `rm -fr` permet de supprimer un répertoire.

mv & cp

- `mv` permet de renommer/déplacer un fichier ou répertoire.

mv & cp

- `mv` permet de renommer/déplacer un fichier ou répertoire.
- `cp` permet de copier-coller un fichier ou un répertoire.

Exercice

Pour mettre en pratique les commandes, nous allons naviguer avec l'invite de commande pour créer un répertoire GitHub qui va contenir l'ensemble des répertoires git.

- On débute en affichant les répertoires locaux.
- On se déplace dans l'endroit qu'on souhaite avoir le répertoire (selon votre discrétion).
- On crée le répertoire.

Comment quitter Vim ?

```
1 Esc – :q
```

Comment configurer un autre éditeur de texte ?

Stack Overflow

SSH

Les serveurs de calcul fonctionnent habituellement en invite de commande. Pour communiquer avec eux on peut, en autre, utiliser SSH.

Secure SHell (SSH)

Il s'agit d'un protocole de communication sécurisé pouvant être établi entre des ordinateurs. Ce protocole permet d'établir une session à distance sécurisée.

SSH

```
ssh davidbeauchemin@132.203.120.91
```

SSH - config

Configuration d'identifiant de connexion SSH pour un serveur de calcul distant.

Éditer le fichier ~/.ssh/config

```
1 Host {credential_name}
2     Hostname {ip_address}
3     User {username}
```

Astuces SSH

- `ssh {hostname}`

Astuces SSH

- `ssh {hostname}`
- `ssh-copy-ip {hostname}`

Astuces SSH

- `ssh {hostname}`
- `ssh-copy-ip {hostname}`
- `ssh -L {port_local}:localhost:{port_remote} {host}`

Transferer des fichiers

- `scp {remote}:{files_path} {local_path}`

Transferer des fichiers

- `scp {remote}:{files_path} {local_path}`
- `scp -r {local_directory}:{remote} {path}`

Transferer des fichiers

- `scp {remote}:{files_path} {local_path}`
- `scp -r {local_directory}:{remote} {path}`
- `scp -r {remote}:'{path}{pattern}' {local_path}`

SCP - REGEX

Example

```
1 scp -r deepnlp:'/mnt/cnn_dm/model_1/*.png' Result
```

Autres outils intéressants

- tmux
- cURL
- tail -f
- ZSH
- Oh-My-ZSH
- crontab/systemd

Agenda

Pourquoi faire du versionnage

- Permet de suivre les changements pour un ensemble de fichiers.

Pourquoi faire du versionnage

- Permet de suivre les changements pour un ensemble de fichiers.
- Permet de revenir à des versions antérieures du code.

Pourquoi faire du versionnage

- Permet de suivre les changements pour un ensemble de fichiers.
- Permet de revenir à des versions antérieures du code.
- Permet de comparer les changements.

Nomenclature

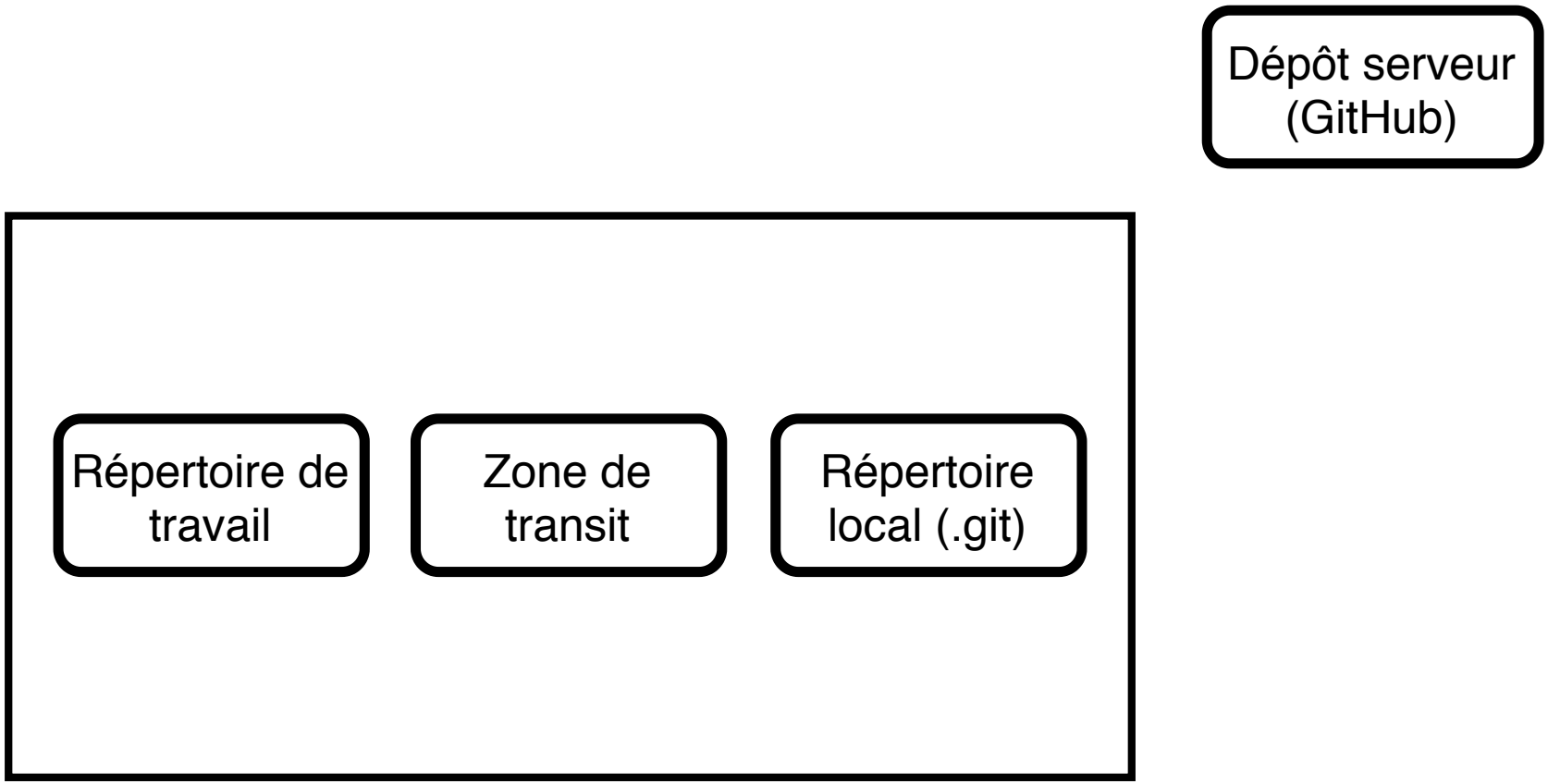
- Dépôt (repository)
- Commit
- Branche (branch)
- master
- Local = Votre ordinateur
- Remote = GitHub/GitLab

Inspiration

Pro Git

How to teach Git

Environnement



Le plus simple pour partir un dépôt git est de créer un dépôt sur GitHub et de faire un `git clone <url>`.
Cette commande initialise

- le dépôt,

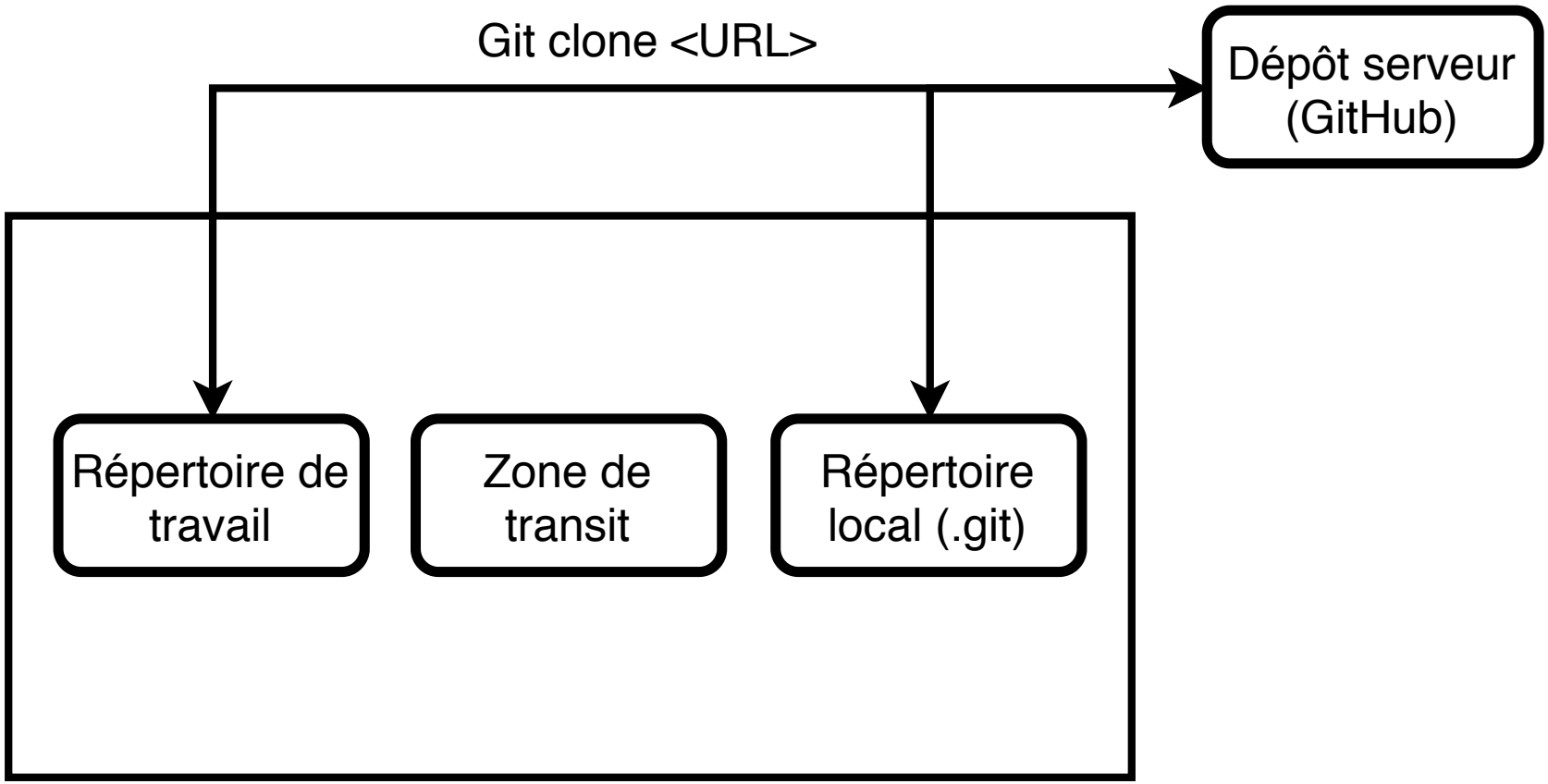
Le plus simple pour partir un dépôt git est de créer un dépôt sur GitHub et de faire un `git clone <url>`.
Cette commande initialise

- le dépôt,
- les remotes,

Le plus simple pour partir un dépôt git est de créer un dépôt sur GitHub et de faire un `git clone <url>`.
Cette commande initialise

- le dépôt,
- les remotes,
- des fichiers de base dans le dépôt (*.gitignore*, *README.md*).

Cloner un répertoire

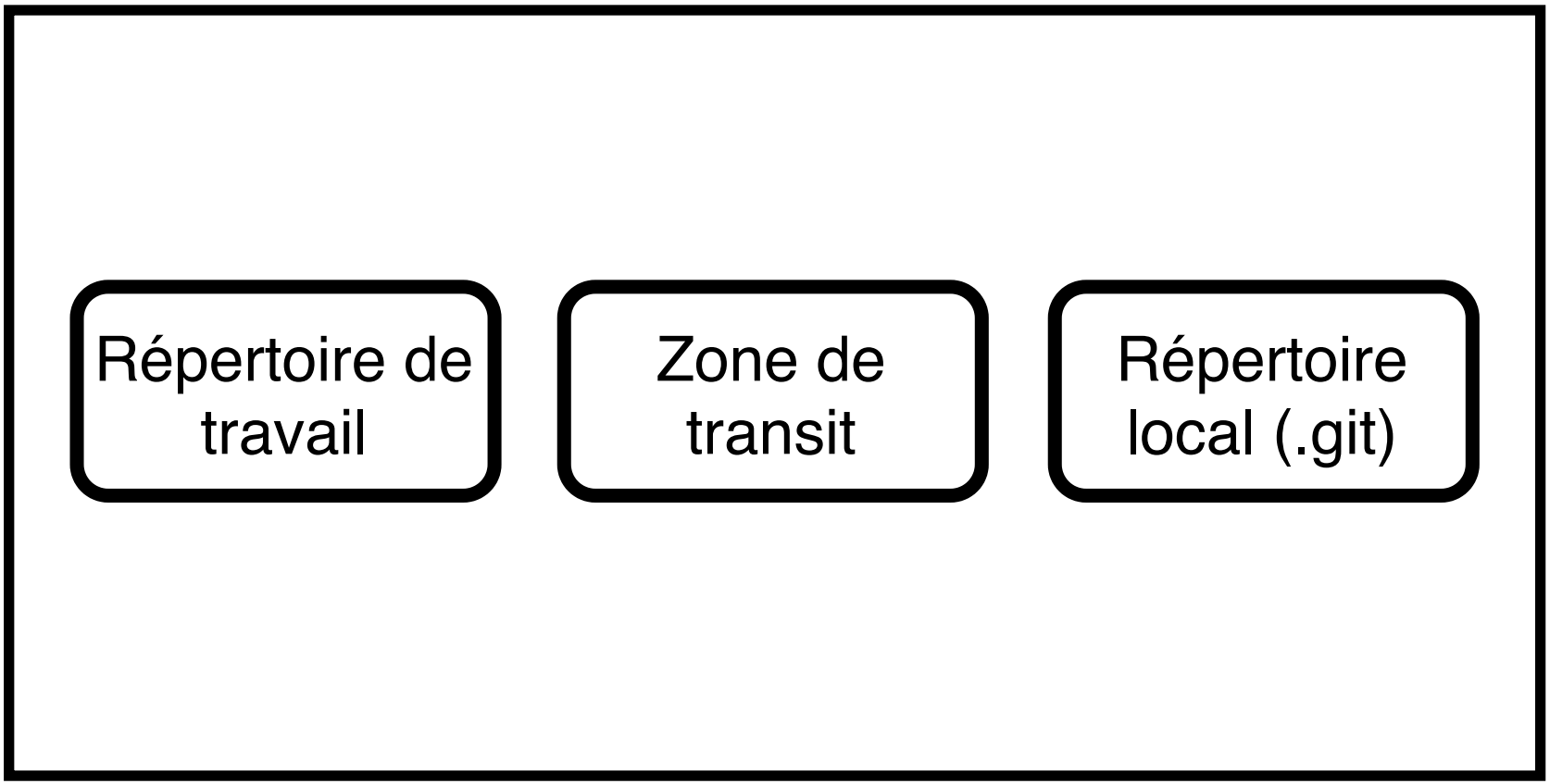


Exercice

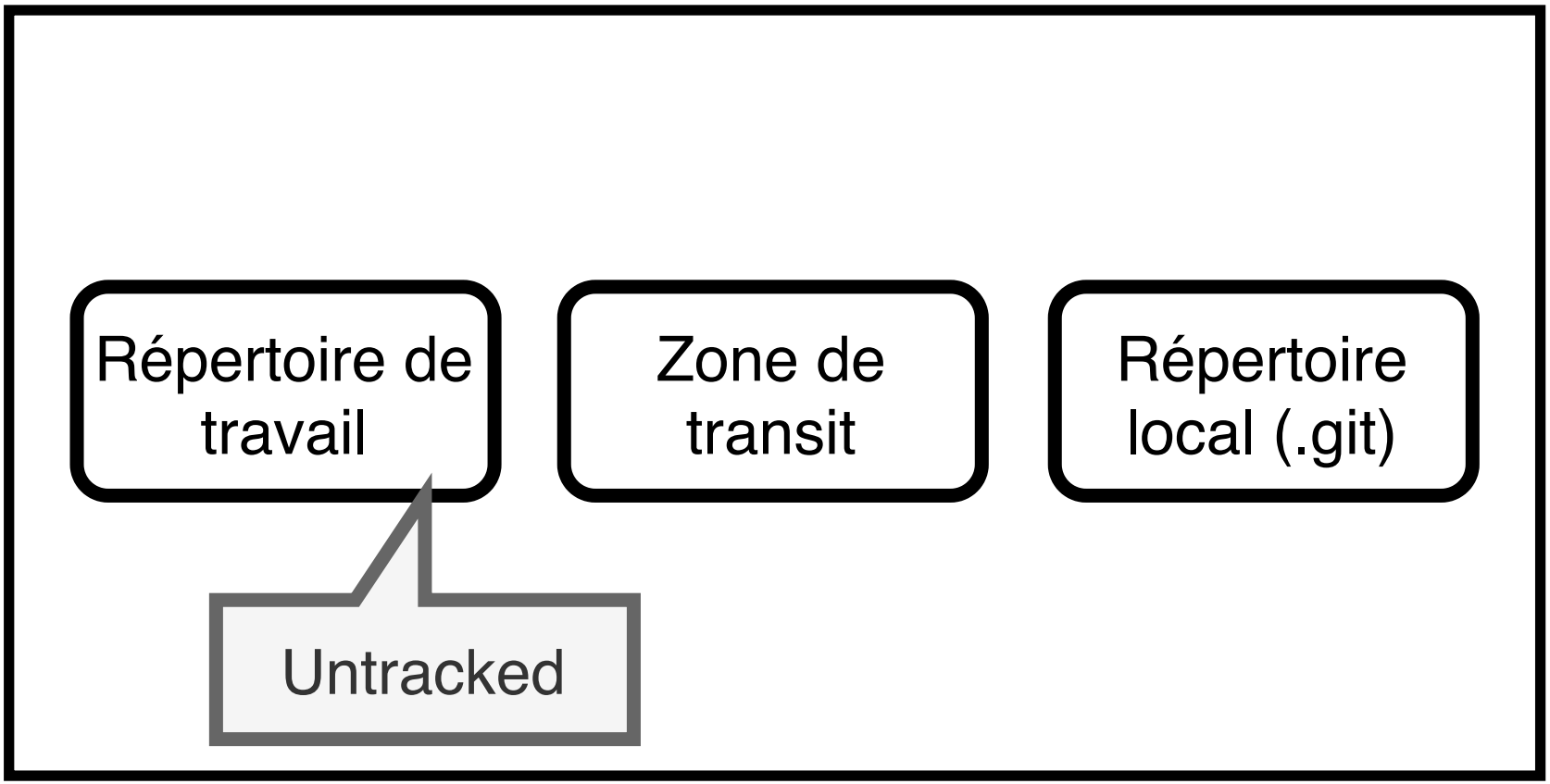
Cloner un dépôt

Allez cloner le dépôt de la formation
`https://github.com/davebulaval/bonnes-pratiques-git.git`

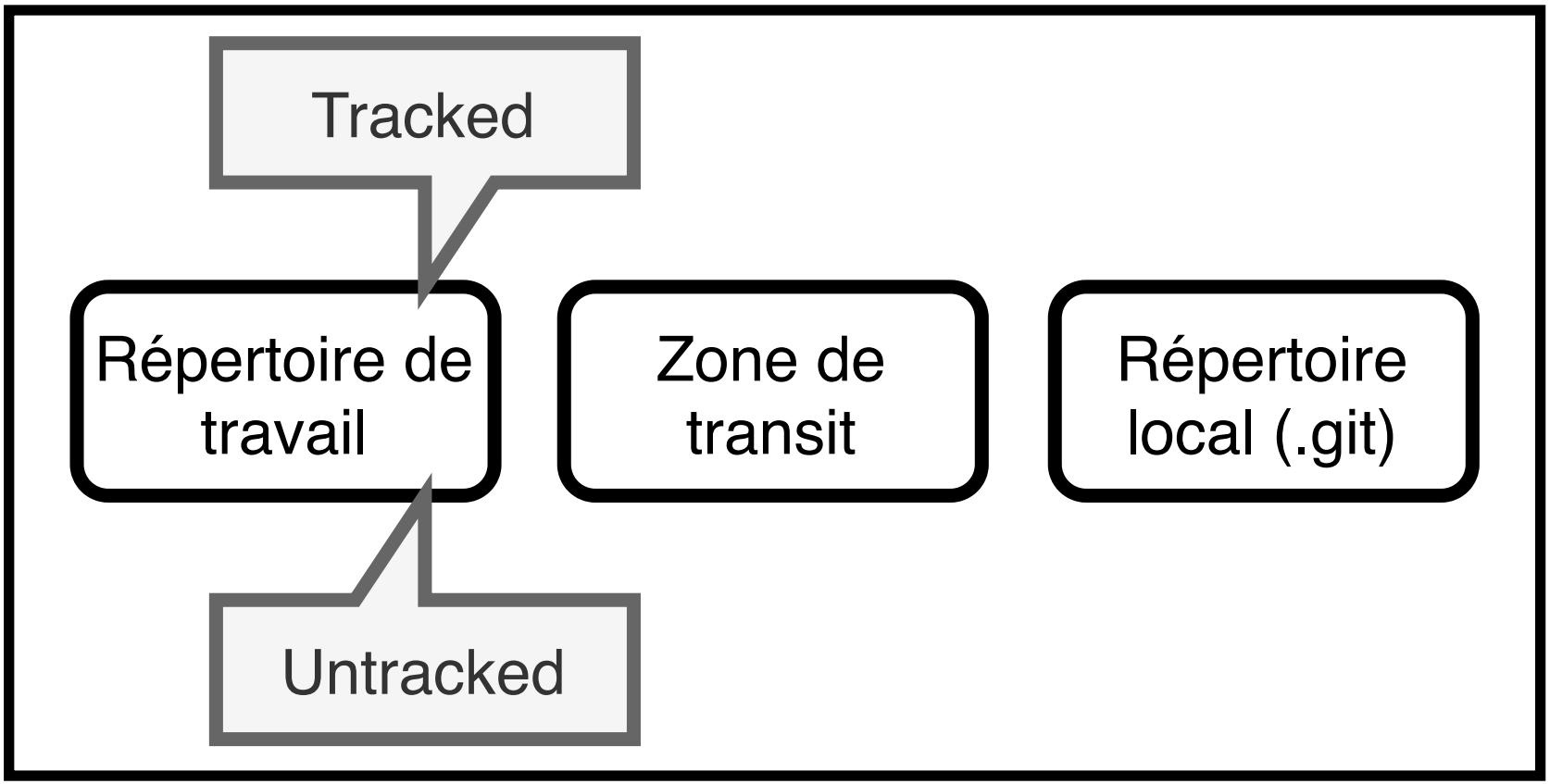
Modification locale



Modification locale



Modification locale

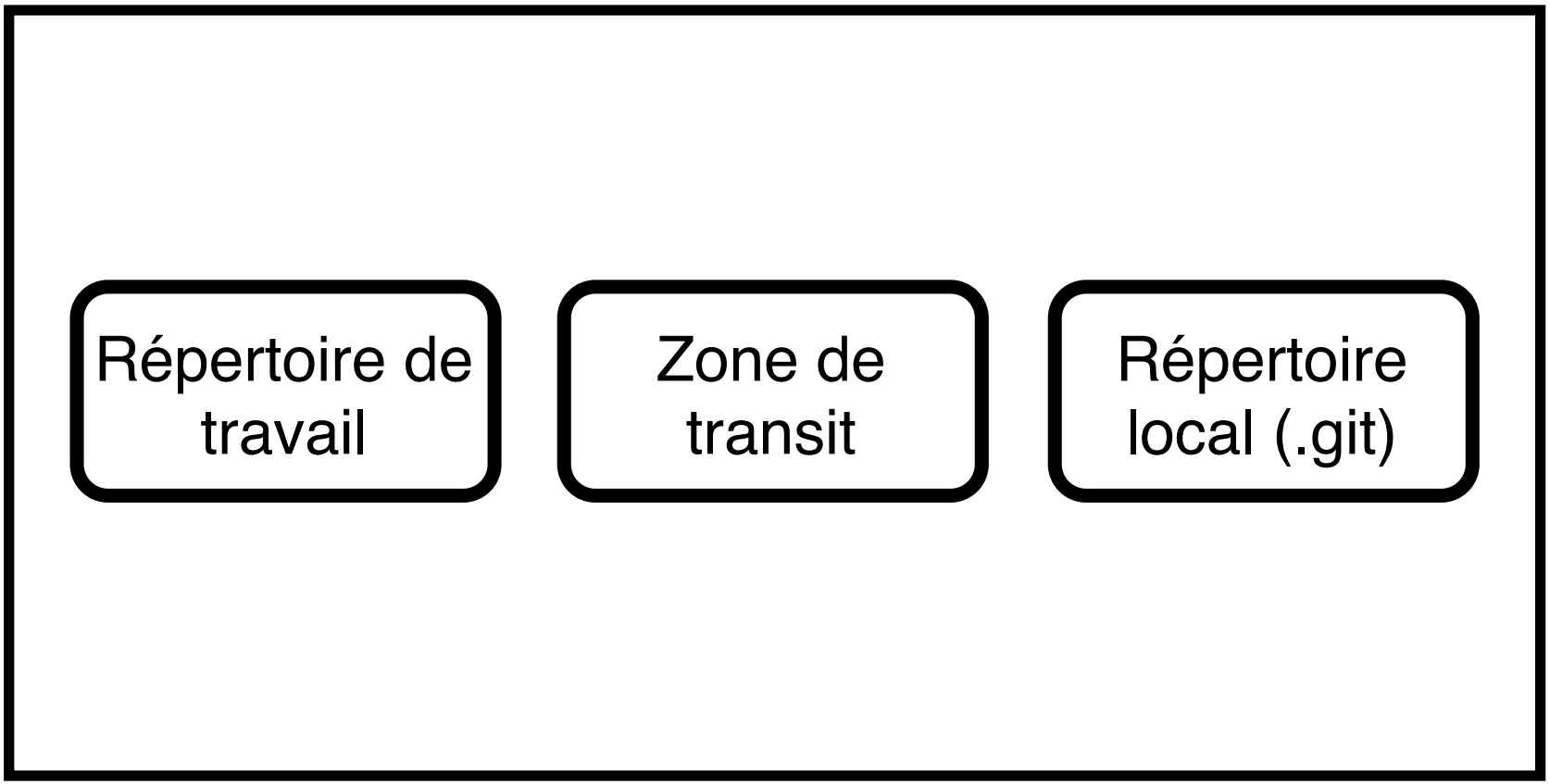


Modification locale

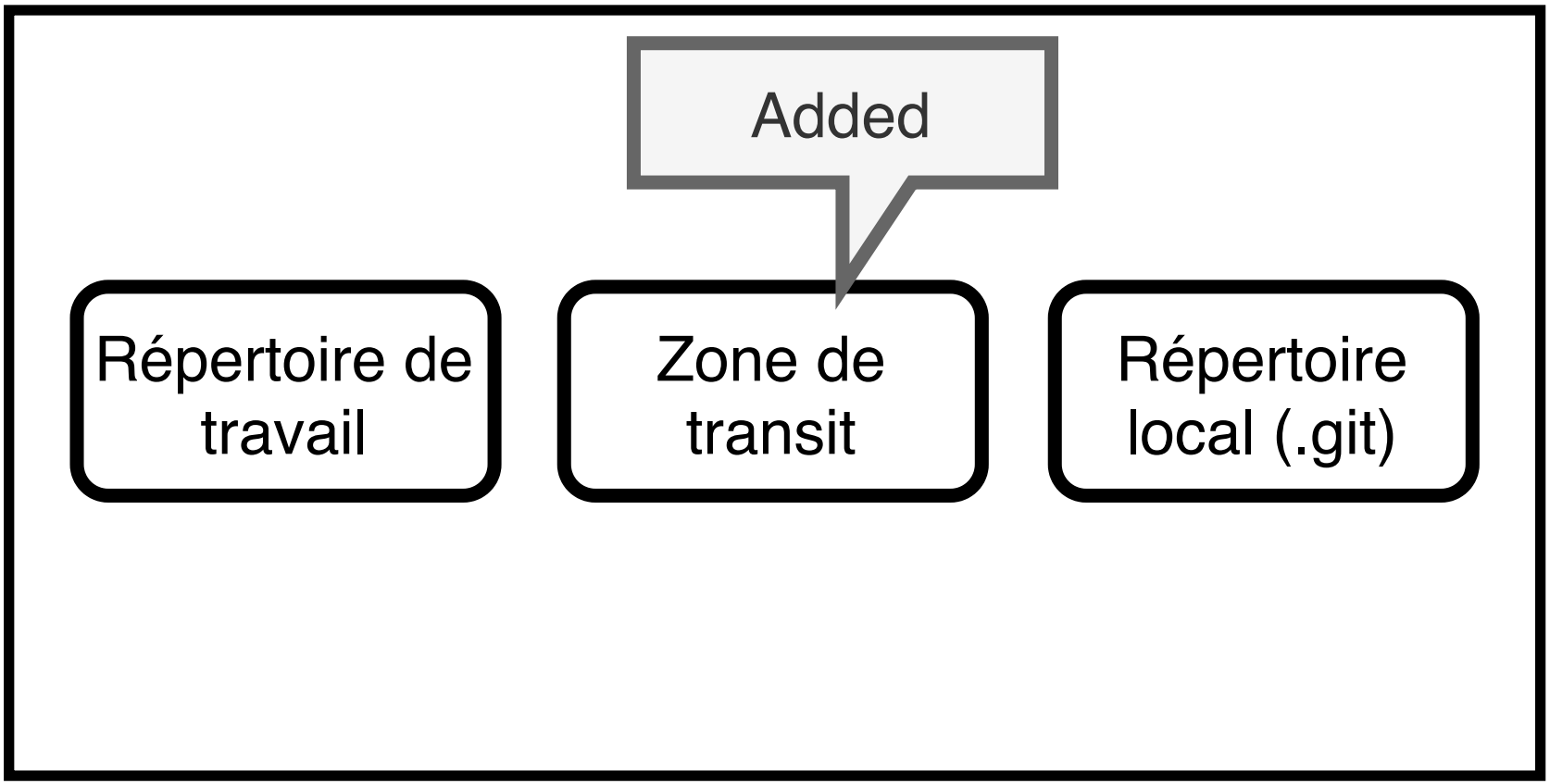
Avec Git, il est possible de facilement consulter l'état de notre environnement de travail.

- Avec `git status` il est facile de connaitre l'état des fichiers suivis et non suivis.

Zone de transit



Zone de transit



add this

Pour préparer un commit, on doit indiquer à Git les fichiers que celui-ci doit contenir. Pour cela on passe par la zone de transit à l'aide de la commande `git add`.

- `git add {fichiers}`

add this

Pour préparer un commit, on doit indiquer à Git les fichiers que celui-ci doit contenir. Pour cela on passe par la zone de transit à l'aide de la commande `git add`.

- `git add {fichiers}`
- `git add -u`

add this

Pour préparer un commit, on doit indiquer à Git les fichiers que celui-ci doit contenir. Pour cela on passe par la zone de transit à l'aide de la commande `git add`.

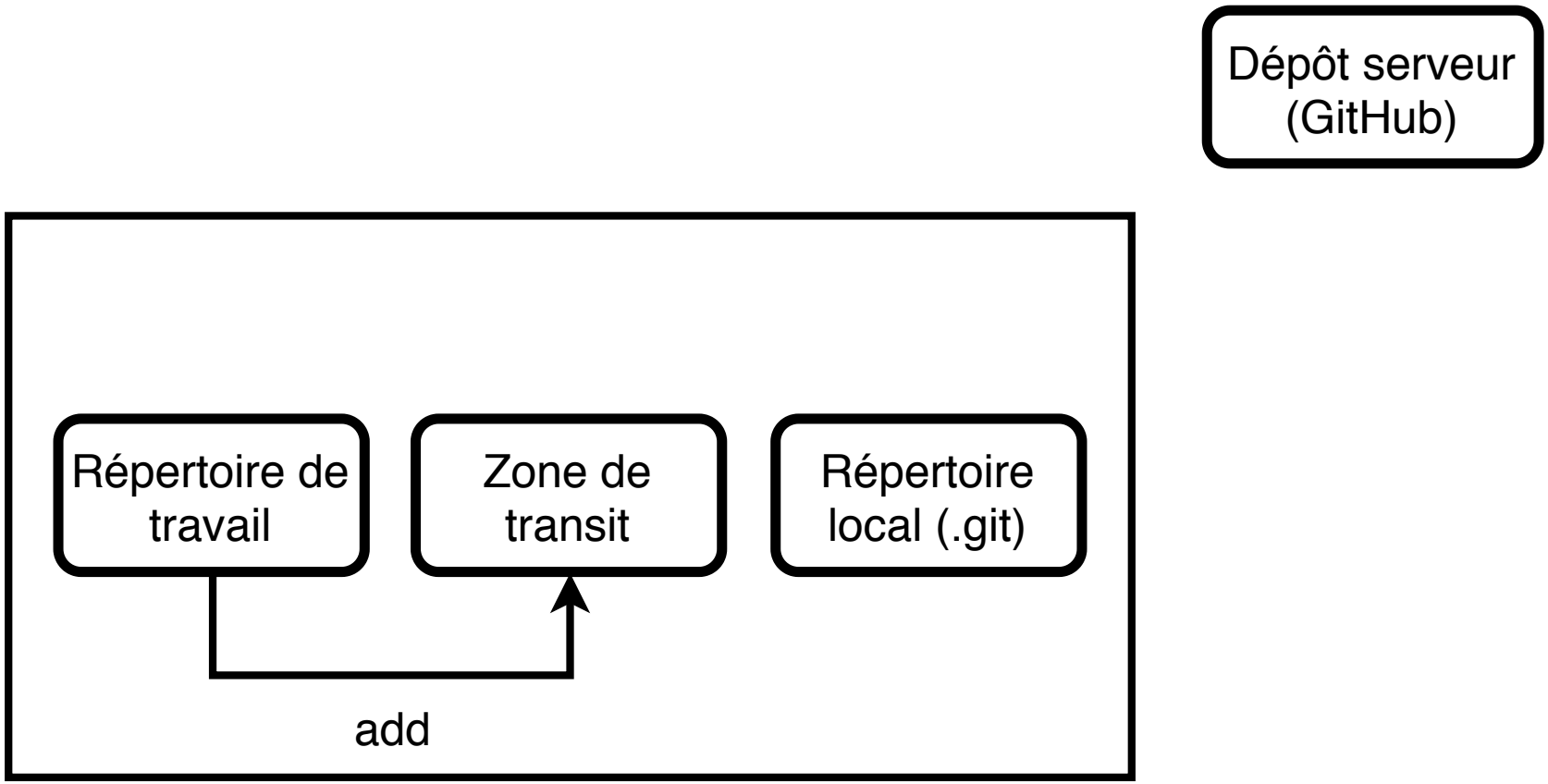
- `git add {fichiers}`
- `git add -u`
- `git add .`

add this

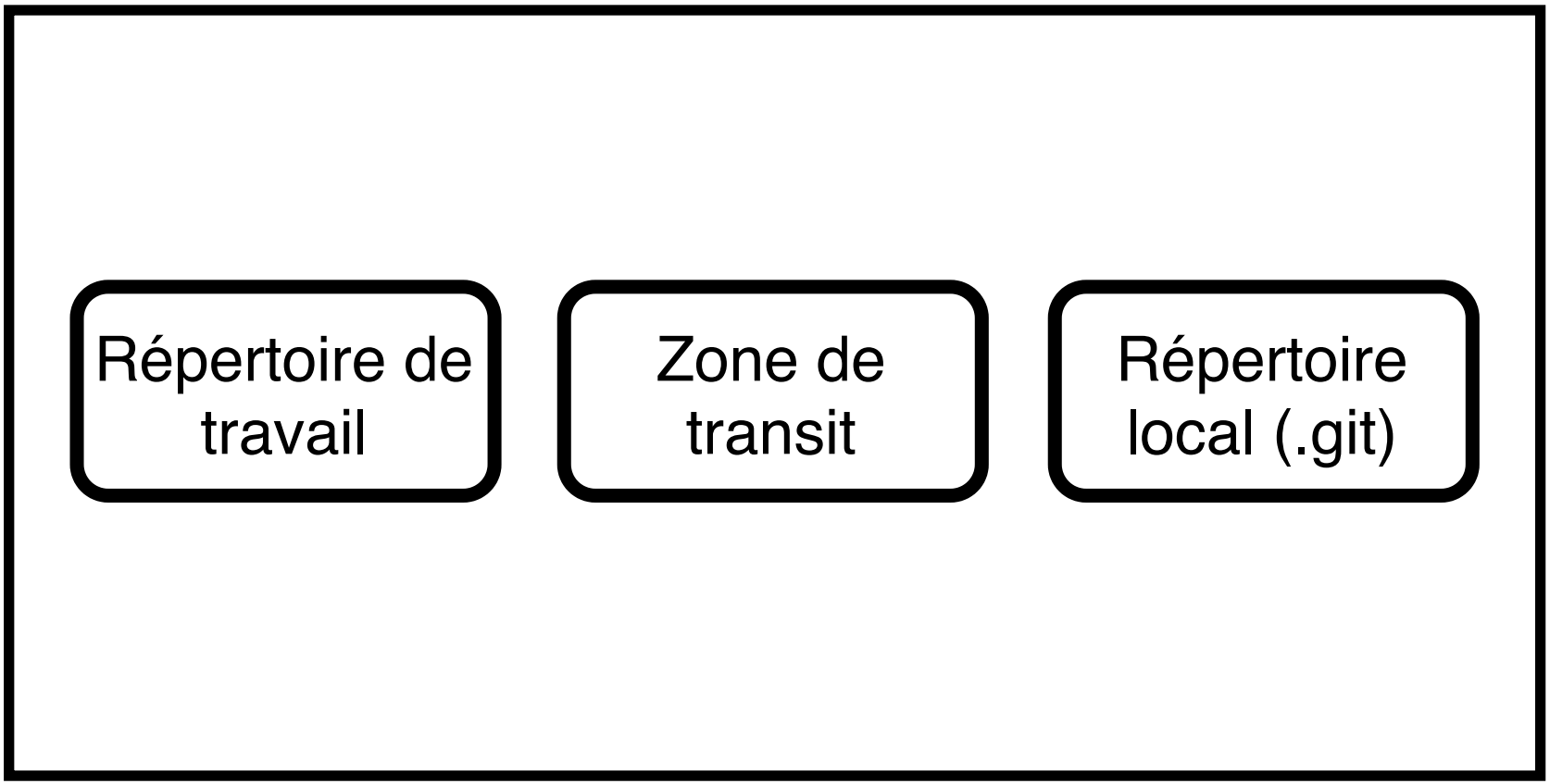
Pour préparer un commit, on doit indiquer à Git les fichiers que celui-ci doit contenir. Pour cela on passe par la zone de transit à l'aide de la commande `git add`.

- `git add {fichiers}`
- `git add -u`
- `git add .`
- `git add -A`

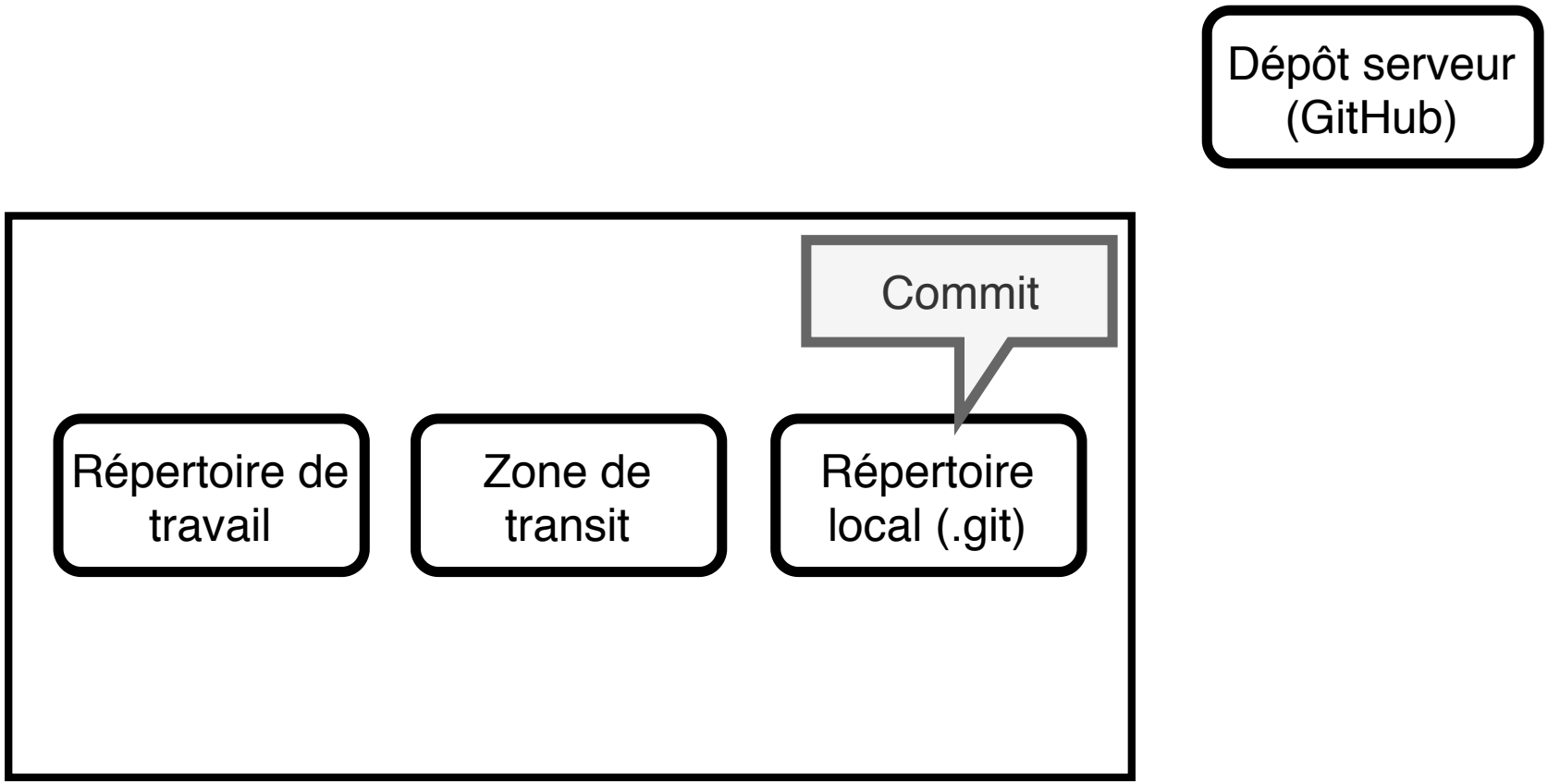
Workflow git add



Répertoire local



Répertoire local



commit that

Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment précis. Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

Pour créer cela, on utilise la commande `git commit`.

- `git commit`

commit that

Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment précis. Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

Pour créer cela, on utilise la commande `git commit`.

- `git commit`
- `git commit -m "message"`

commit that

Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment précis. Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

Pour créer cela, on utilise la commande `git commit`.

- `git commit`
- `git commit -m "message"`
- `git commit -a -m "message"`

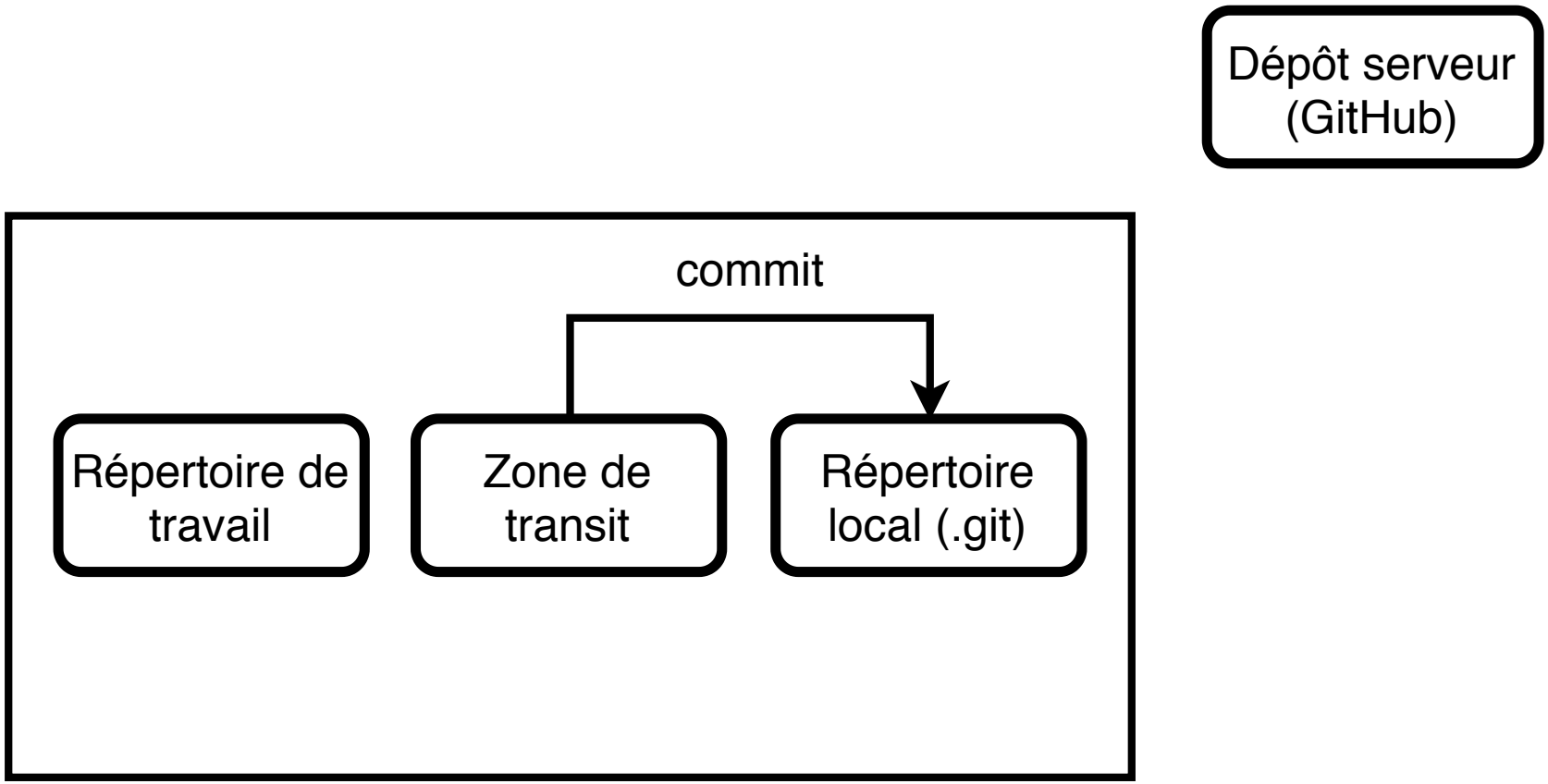
commit that

Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment précis. Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

Pour créer cela, on utilise la commande `git commit`.

- `git commit`
- `git commit -m "message"`
- `git commit -a -m "message"`
- `git commit -m "message" --dry-run`

Workflow git commit



diff then

Jusqu'à présent, toutes ces étapes compliquées ne semblent pas apporter grand-chose. On ajoute des fichiers dans une zone de transit pour ensuite *figer* les fichiers dans cette zone. On pourrait très bien avoir un nommage des versions.

- rapport_1.pdf
- rapport_final.pdf
- rapport_final-2.pdf
- code_final-2_final_vrai.py

diff then

C'est la que `git diff` entre en jeu. Avec cette commande, on est en mesure, *ligne par ligne*, de voir les lignes de codes modifier. Ainsi, on est en mesure de savoir les deltas entre chaque version du code.

- `git diff`

diff then

C'est la que `git diff` entre en jeu. Avec cette commande, on est en mesure, *ligne par ligne*, de voir les lignes de codes modifier. Ainsi, on est en mesure de savoir les deltas entre chaque version du code.

- `git diff`
- `git diff nom_fichier`

diff then

C'est la que `git diff` entre en jeu. Avec cette commande, on est en mesure, *ligne par ligne*, de voir les lignes de codes modifier. Ainsi, on est en mesure de savoir les deltas entre chaque version du code.

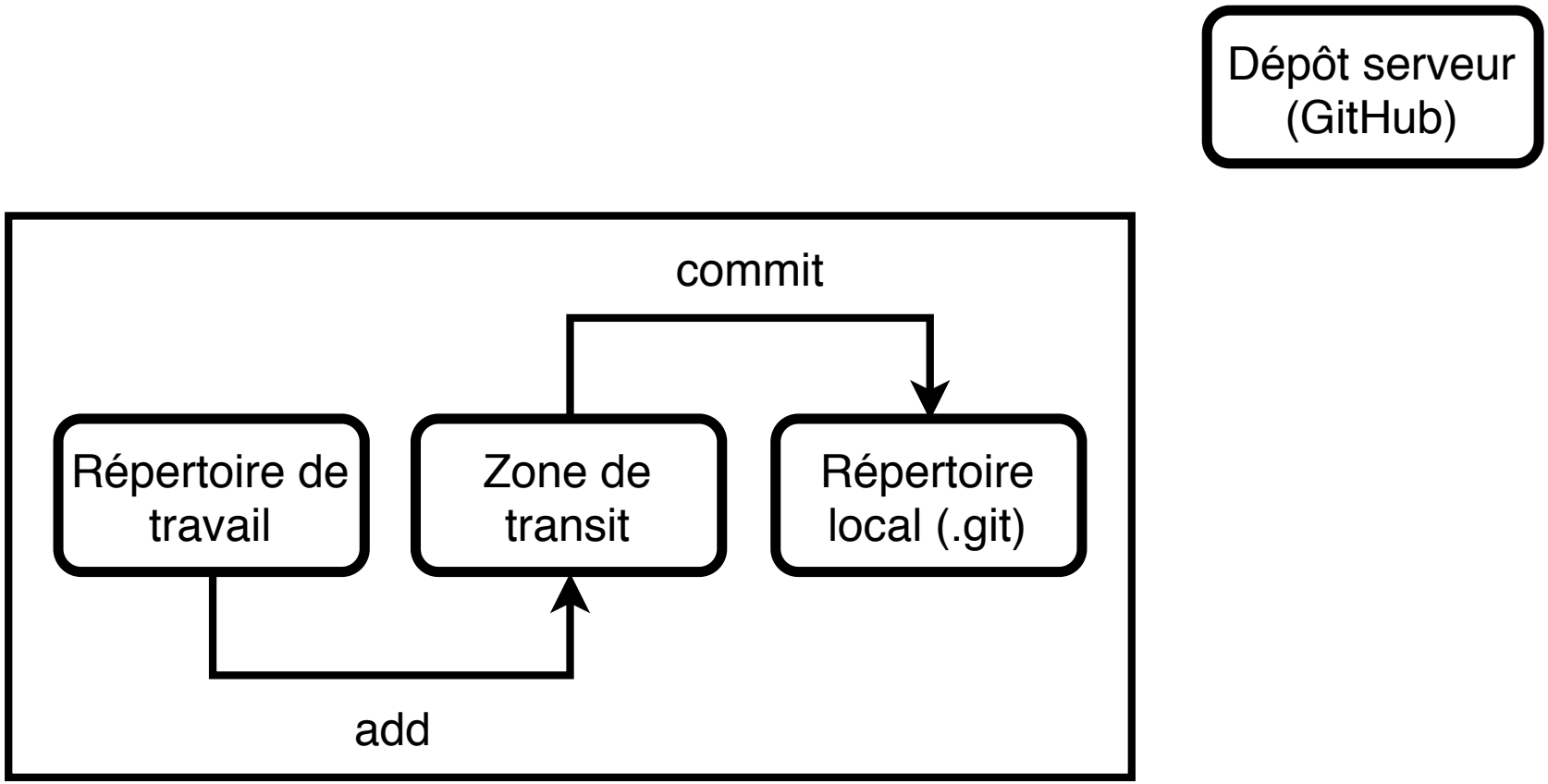
- `git diff`
- `git diff nom_fichier`
- `git diff --staged`

diff then

C'est la que `git diff` entre en jeu. Avec cette commande, on est en mesure, *ligne par ligne*, de voir les lignes de codes modifier. Ainsi, on est en mesure de savoir les deltas entre chaque version du code.

- `git diff`
- `git diff nom_fichier`
- `git diff --staged`
- `git diff commit hash_number`

Workflow git add ; git commit



Exercice

Workflow

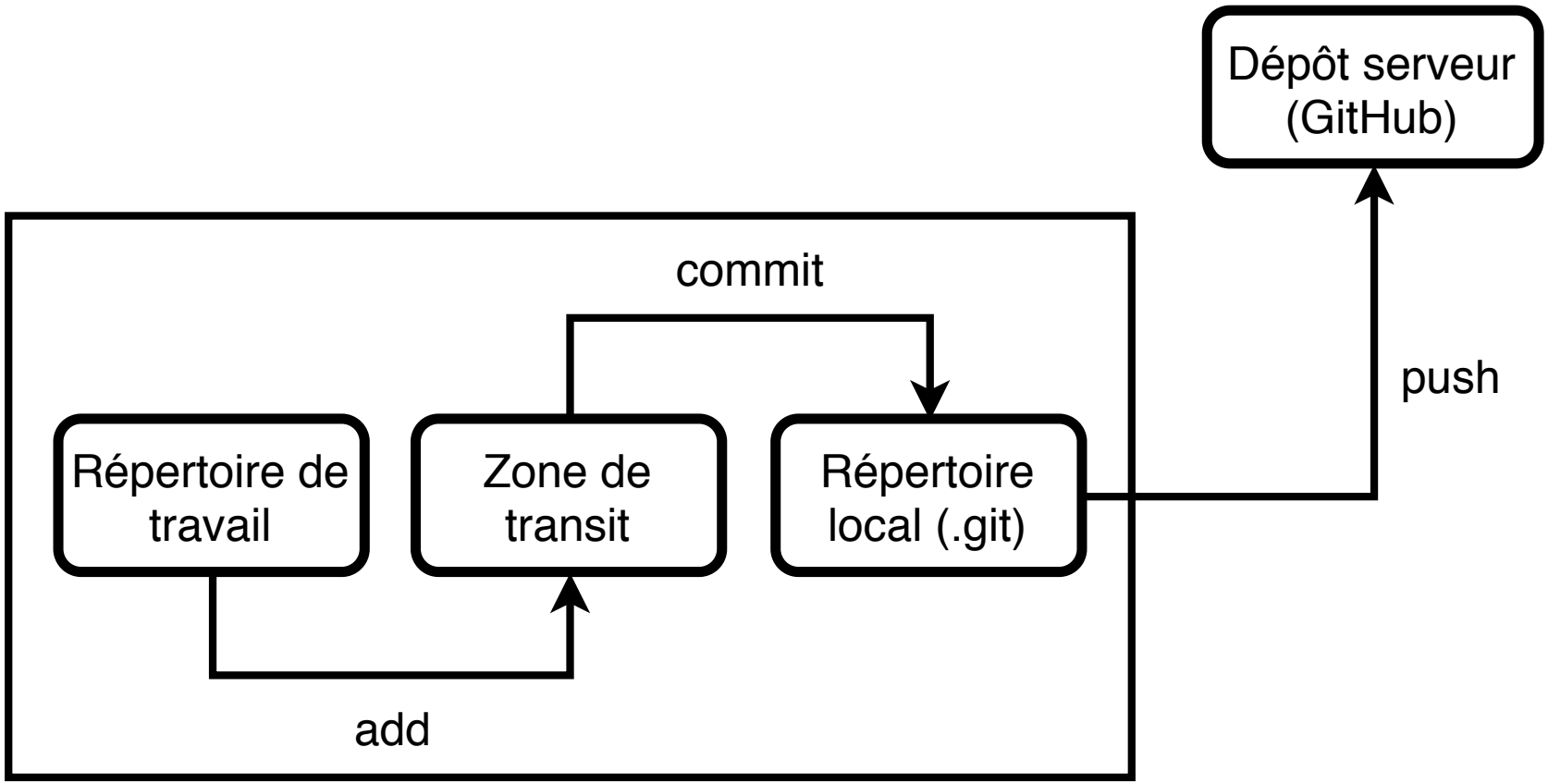
Ajouter et modifier un fichier dans le répertoire cloner.
Commiter uniquement le fichier modifier.
Commiter le nouveau fichier.

Note : Tenter d'utiliser les commandes `git status` et `git diff` tout au long du processus pour valider les actions prises.

Remote

Pour mettre à jour le serveur distant, il suffit de faire lui *pousser* l'information avec la commande `git push`.

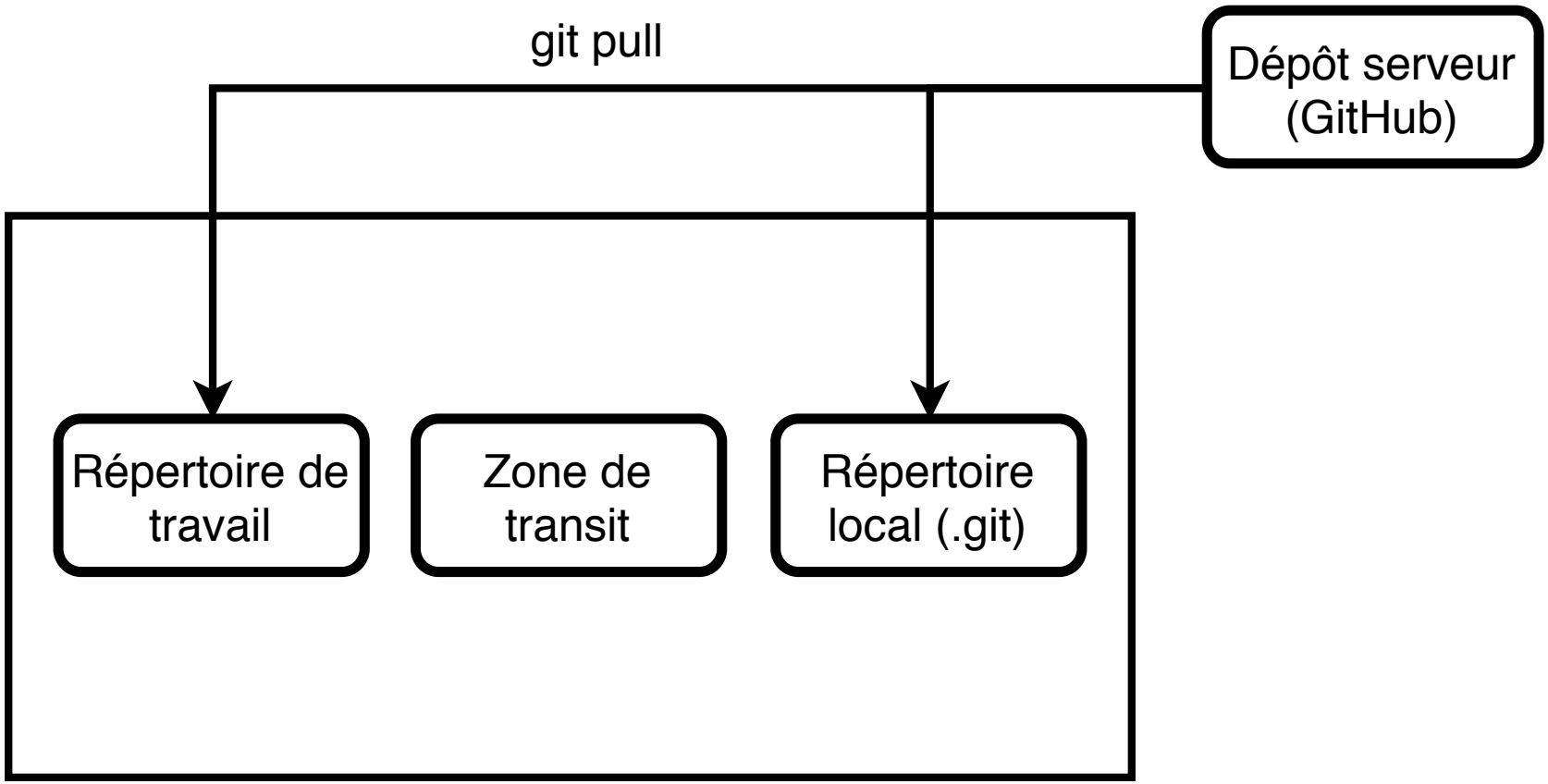
Workflow git add ; git commit ; git push



Remote

Pour **se** mettre à jour avec le serveur distant, il suffit de lui *demander* l'information avec la commande `git pull`.

Workflow git pull



Exercice

Push

Pousser les modifications au serveur.

Remote

Vous ne pouvez pas `git push` si le serveur distant détecte qu'il vous manque des commits. Il faudrait alors faire un `git pull` et potentiellement régler un `merge conflict`.

Merge

Si Git n'arrive pas à résoudre les conflits de fusion, il va vous indiquer dans le code source les endroits ou vous devez résoudre l'ambiguïté.

```
1 If you have questions, please
2 <<<<<<< HEAD
3 open an issue
4 =======
5 ask your question in IRC.
6 >>>>>>> branch-a
```

A branch for this and a branch for that!

Un autre point qui distingue Git d'une utilisation simple de nommage est la possibilité de faire cohabiter plusieurs versions du code en même temps. Pour cela on utilise les branches.

- `git branch nom_branche`

A branch for this and a branch for that!

Un autre point qui distingue Git d'une utilisation simple de nommage est la possibilité de faire cohabiter plusieurs versions du code en même temps. Pour cela on utilise les branches.

- `git branch nom_branche`
- `git branch -D nom_branche`

A branch for this and a branch for that!

Un autre point qui distingue Git d'une utilisation simple de nommage est la possibilité de faire cohabiter plusieurs versions du code en même temps. Pour cela on utilise les branches.

- `git branch nom_branche`
- `git branch -D nom_branche`
- `git checkout nom_branche`

A branch for this and a branch for that!

Un autre point qui distingue Git d'une utilisation simple de nommage est la possibilité de faire cohabiter plusieurs versions du code en même temps. Pour cela on utilise les branches.

- `git branch nom_branche`
- `git branch -D nom_branche`
- `git checkout nom_branche`
- `git checkout -b nom_branche`

A branch for this and a branch for that!

Un autre point qui distingue Git d'une utilisation simple de nommage est la possibilité de faire cohabiter plusieurs versions du code en même temps. Pour cela on utilise les branches.

- `git branch nom_branche`
- `git branch -D nom_branche`
- `git checkout nom_branche`
- `git checkout -b nom_branche`
- `git push --set-u origin nom_branche`

Exercice

Créer une branche

Créer une branche.
Ajoutez un fichier et mettez à jour le serveur.

Fusionner des branches

L'intérêt d'avoir des branches est de pouvoir ensuite les fusionner. Pour cela, on utilise la commande `git merge nom_branche`.

Exercice

Fusionner une branche

Fusionner votre branche avec *master*.

Bonnes pratiques Git

- Commit often

Bonnes pratiques Git

- Commit often
- Commit clearly

Bonnes pratiques Git

- Commit often
- Commit clearly
- Écrire de **vrais** messages dans vos commits. (git-commit-rules)

Bonnes pratiques Git

- Commit often
- Commit clearly
- Écrire de **vrais** messages dans vos commits. (git-commit-rules)
- Utiliser les branches pour diviser vos problèmes.

Bonnes pratiques Git

- Commit often
- Commit clearly
- Écrire de **vrais** messages dans vos commits. (git-commit-rules)
- Utiliser les branches pour diviser vos problèmes.
- Utiliser les *pull request* pour faire du *code review*.

Bonnes pratiques Git

- Commit often
- Commit clearly
- Écrire de **vrais** messages dans vos commits. (git-commit-rules)
- Utiliser les branches pour diviser vos problèmes.
- Utiliser les *pull request* pour faire du *code review*.
- Utiliser SSH pas HTTPS.

Bonnes pratiques Git - branches

Ne pas s'approprier des branches (`code_dbeauchemin`).

Ressources pour se pratiquer

- Manuel Git
- Atlassian Git tutorial
- Pratiquer les branches Git
- Visualizing Git
- Pour mieux comprendre les concepts

Agenda

La gestion des *remotes*

`git clone` configure automatiquement les remotes du serveur distant. De base celle-ci s'appelle *origin* mais elles pourraient s'appeler n'importe quel nom. Par exemple, `bitbucket` ou `source`.

Cela signifie qu'on peut avoir plusieurs sources distinctes.

La gestion des *remotes*

Ajout d'une remote

```
git remote -v
git remote add bitbucket URL
```

Exercice

Ajouter une remote

Ajouter la remote suivante au dépôt `bonnes-pratiques-git`
`https://davebulaval@bitbucket.org/davebulaval/bonnes-pratiques-git.git`
Mettez à jour avec ce dépôt.

La gestion des *remotes*

Mise à jour d'une remote

```
git remote -v
git remote set-url origin URL
```

La gestion des fichiers

Comment *revert* les changements à un fichier ?

```
git checkout -- fichier
```

La gestion des fichiers

Comment *revert* les changements à un fichier en zone de transit (add) ?

```
git reset HEAD fichier
git checkout -- fichier
```

La gestion des fichiers

Comment supprimer un fichier suivi par Git ?

```
git rm [-r]
```

La gestion des commits

Comment annuler un commit non push ?

```
git reset HEAD^
```

La gestion des commits

Comment modifier le message d'un commit ?

```
git commit --amend -m "nouveau message"
```

La gestion des commits

Comment ajouter un fichier dans le commit précédent ?

```
git add fichier_oublier
git commit --amend --no-edit
```

La gestion des commits

Comment ajouter un fichier dans le commit précédent et éditer le message du commit ?

```
git commit --amend fichier
```

Attention, l'opération de changer le message et son contenu modifie le SHA du commit. Il faut faire cette opération uniquement si les modifications sont locales.

La gestion des commits

Comment *commit* des portions de modification d'un fichier ?

```
git add --patch
```

git add –patch

Cette commande permet de segmenter les modifications dans les fichiers, pour en regrouper les modifications portant sur un même sujet. Les principales options disponibles durant l'application de la *patch* sont :

- `y` Yes, ajouter ce morceau.
- `n` No, ne pas ajouter ce morceau.
- `d` Don't, ne pas ajouter ce morceau et tous les morceaux suivants.
- `s` Split, découper ce morceau en plus petit morceau.

Exercice

git add –patch

Déplacez-vous sur la branche patch
Effectuez des changements dans le code à de multiple endroit.
Faites plusieurs commits en regroupant différentes modifications.
Mettez à jour le répertoire origin.

La gestion des branches

Comment fusionner deux branches en réduisant le nombre de *commit* ?

```
git merge branch_to_merge --squash
```

Exercice

git *squash*

Fusionner la branche patch dans master en mode squash.

La gestion des branches

Comment faire le ménage des branches avec un merge ?

```
git remote prune origin [--dry-run]
```

La gestion du code

Autres commandes intéressantes :

- `git stash`
- `git stash apply`
- `git blame`
- `git clean -df`

Exemple

Gestion du code

Faire un `git blame` sur le fichier `fichier_blame.txt`
Stasher le fichier `fichier_stash.txt`
Cleaner le dépôt des fichiers et répertoires non suivis.
Réappliquer le fichier *stasher*.

Consulter le log

```
git log --stat
```

Consulter le log des points de référence.

```
git reflog
```

Consulter le log amélioré

```
git lg
```

Pour avoir l'alias `git lg`.

Agenda

La gestion pro des commandes

git alias

Permet de configurer de nouvelles commandes ou de simplifier des commandes déjà existantes de git. Par exemple, l'ajout de la commande `git lg`.

Tutoriel Git alias par Atlassian

git hooks

Les git hooks sont des scripts qui seront exécutés automatiquement à chaque fois qu'un évènement survient dans le dépôt git.

Tutoriel Git hooks par Atlassian

La gestion *pro* des fichiers volumineux

git LFS

GitHub limite la taille des fichiers à 100 mo. Lorsqu'il y a des fichiers volumineux, les push/pull/clone sont très lentes. Pour corriger ce problème on utilise Git LFS git permet de réduire considérablement les commandes.

Tutoriel Git LFS par Atlassian

La gestion *pro* des commits

git cherry-pick

Cette commande permet d'appliquer les changements d'un commit sur une autre branche. Pratique pour réorganiser une branche master ou pour défaire une modification.

Tutoriel cherry pick par Atlassian

La gestion *pro* des commits

git rebase

Similaire à cherry-pick mais plutôt que de prendre les commits on réapplique les modifications à l'histoire. Cela peut entrainer des suppressions et réécrit l'histoire des commits.

Tutoriel rebase par Atlassian
Tutoriel merge vs rebasing par Atlassian

La gestion *pro* des commits

git reset soft, mixt and hard

Permet de remettre la tête courante (HEAD) à un état désigné. Il y a plusieurs applications :

- Soft, déplace seulement les fichiers commit dans staged.
- mix, les fichiers staged et commit sont unstaged et ne touche pas aux fichiers unstaged.
- hard ; restore tout du commit. Donc, supprime tous les changements.

Git reset démystifié
Tutoriel reset par Atlassian