

# Bonnes pratiques & Git

David Beauchemin, BSc.

.Layer, Université Laval, CRDM, GRAAL

21 juin 2019





Figure – <https://xkcd.com/1597/>



# Agenda

---

- 1 Introduction
- 2 Ligne de commande
- 3 Git
- 4 Aller plus loin avec Git
- 5 Être expert en Git



- 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](https://dotlayer.org)

 : [MeetupMLQuebec](#)



- 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

---

- 1 Introduction
- 2 Ligne de commande
- 3 Git
- 4 Aller plus loin avec Git
- 5 Être expert en Git



Pourquoi utiliser l'invite de commande ?

- Permet plus de flexibilité.



Pourquoi utiliser l'invite de commande ?

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





Pourquoi utiliser l'invite de commande ?

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



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*.



- Tab est votre amie!



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



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



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



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





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



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



- `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 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 connaître son emplacement actuel.



- `mkdir` permet de créer un répertoire.



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



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





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



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



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](#)



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 davidbeauchemin@132.203.120.91
```





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}
```



■ `ssh {hostname}`



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



- `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}`



## 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

---

- 1 Introduction
- 2 Ligne de commande
- 3 Git
- 4 Aller plus loin avec Git
- 5 Être expert en Git



# 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.

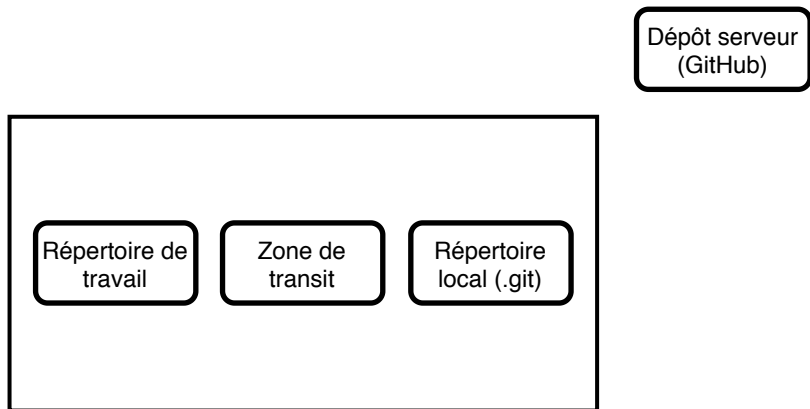


- Dépôt (repository)
- Commit
- Branche (branch)
- master
- Local = Votre ordinateur
- Remote = GitHub/GitLab



## Pro Git How to teach Git







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

- Initialize le dépôt,



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

- Initialize le dépôt,
- initialise les remotes,

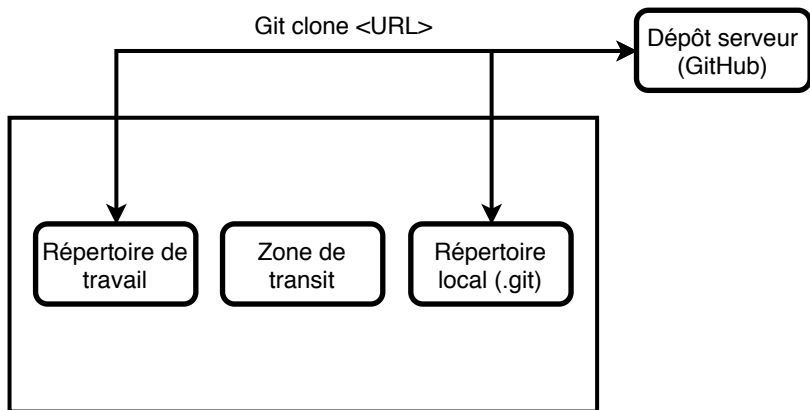


Le plus simple pour partir un dépôt git est de créer un dépôt sur GitHub et de faire un `git clone <url>`.

- Initialize le dépôt,
- initialise les remotes,
- s'initialise avec des fichiers de base automatiquement.



# Cloner un répertoire

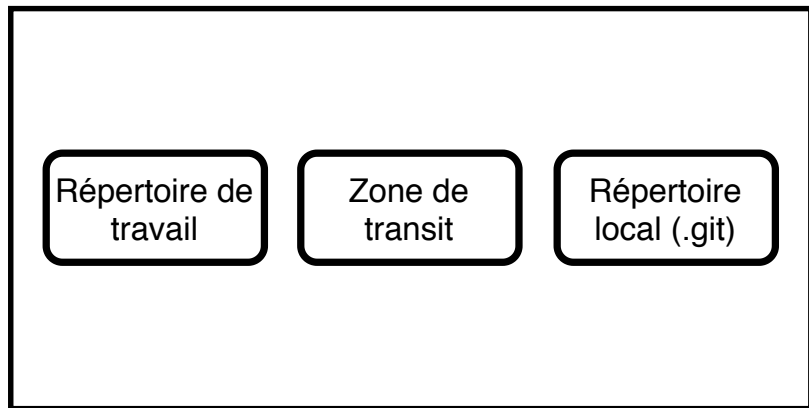


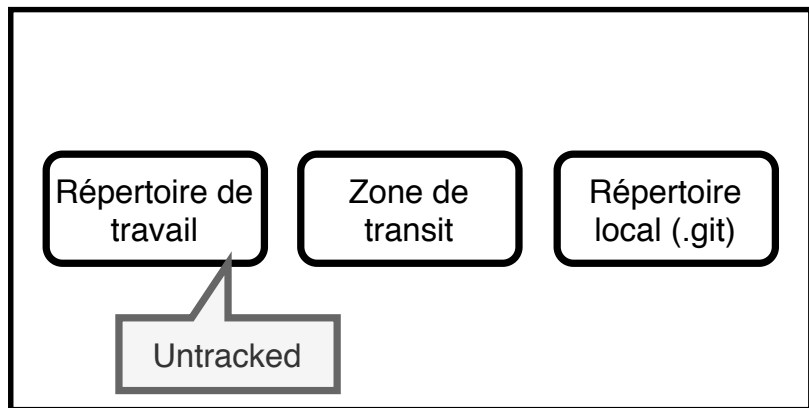
## Cloner un dépôt

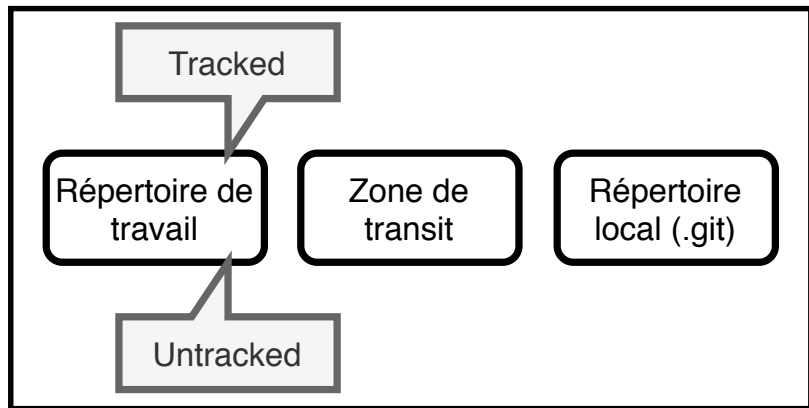
Aller cloner le dépôt de la formation

<https://github.com/davebulaval/bonnes-pratiques-git.git>







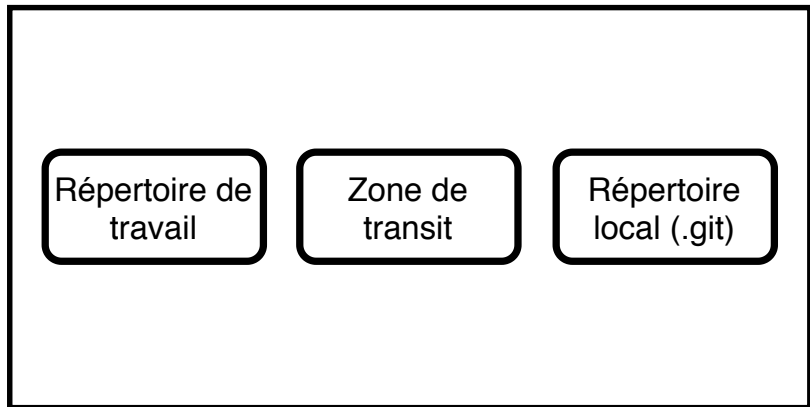


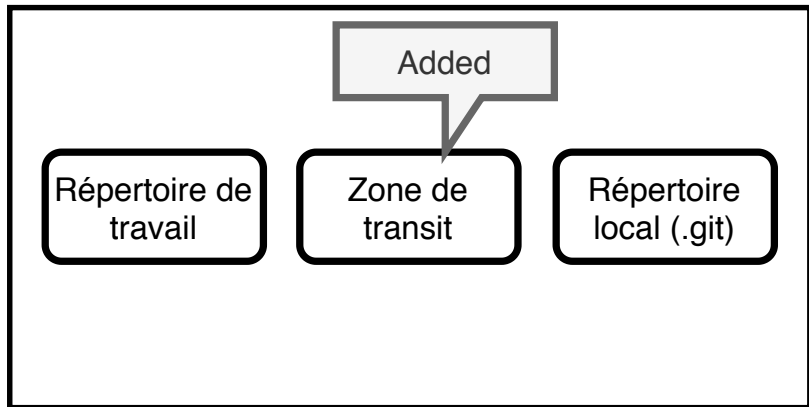


Avec Git, il est possible de facilement consulter l'état de notre environnement de travail.

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







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}`



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`



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 .`



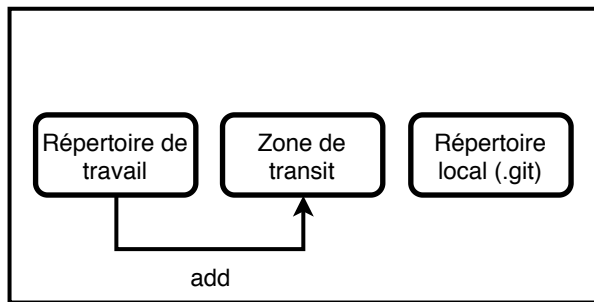
Pour préparer un commit, on doit indiquer à Git les fichiers que celui-ci doit contenir. Pour cela on passe par la zone de transit à l'aide de la commande `git add`.

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

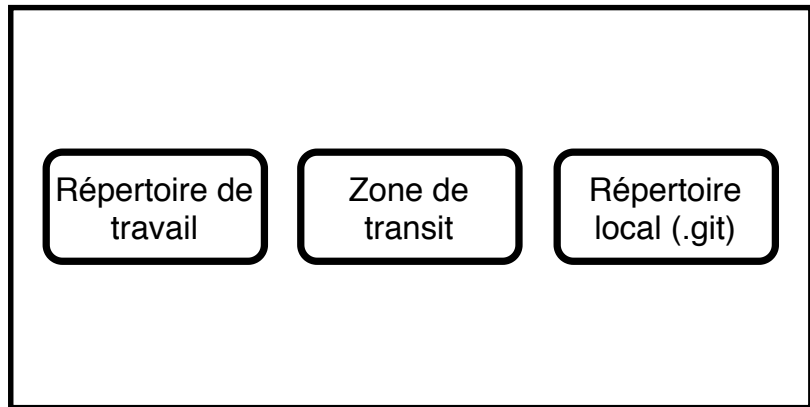


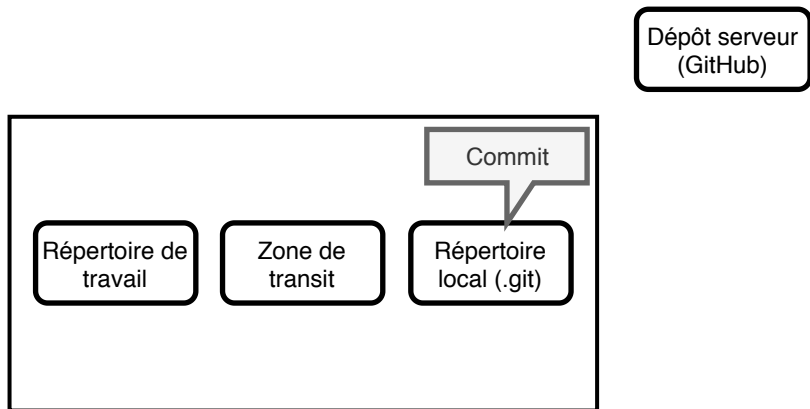
# Workflow git add

Dépôt serveur  
(GitHub)









Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment  $t$ . Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

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

- `git commit`



Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment  $t$ . 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"`



Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment  $t$ . 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"`



Lorsqu'on crée un commit, on prend un *snapshot* de l'état du code à un moment  $t$ . Un commit contient un titre et peut aussi contenir un contenu plus exhaustif.

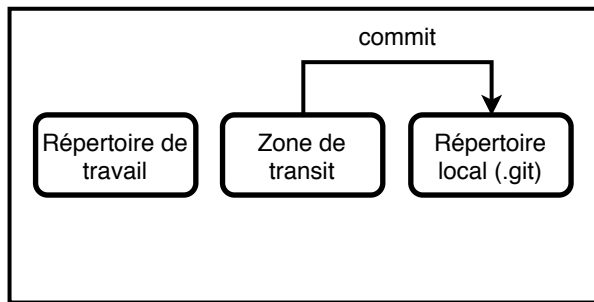
Pour créer cela, on utilise la commande `git commit`.

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



# Workflow git commit

Dépôt serveur  
(GitHub)



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





C'est là 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`



C'est là 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`



C'est là 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`



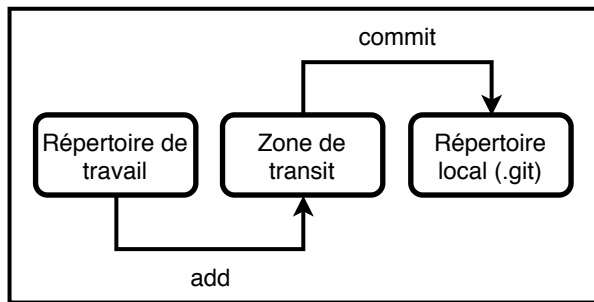
C'est là que `git diff` entre en jeu. Avec cette commande, on est en mesure, *ligne par ligne*, de voir les lignes de codes modifier. Ainsi, on est en mesure de savoir les deltas entre chaque version du code.

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



# Workflow git add; git commit

Dépôt serveur  
(GitHub)



## Workflow

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

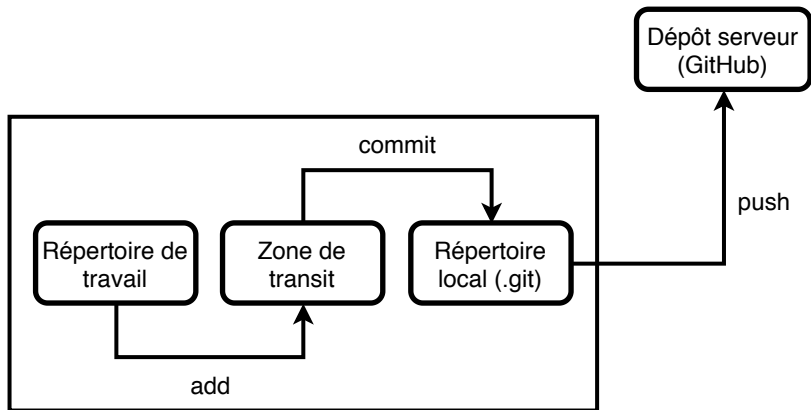
**Note :** Tenter d'utiliser les commandes `git status` et `git diff` tout au long du processus pour valider les actions prises.



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

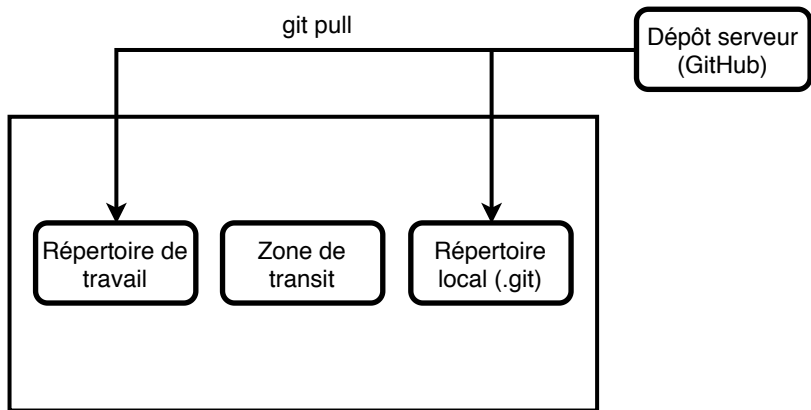




Pour **se** mettre à jour avec le serveur distant, il suffit de lui demander l'information avec la commande `git pull`.



# Workflow git pull



## *Push*

Pousser les modifications au serveur.



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`.



Si Git n'arrive pas à résoudre les conflits de fusion, il va vous indiquer dans le code source les endroits où 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`



## 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`.



Fusionner une branche

Fusionner votre branche avec *master*.



- Commit often



# Bonnes pratiques Git

---

- Commit often
- Commit clearly



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





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



- 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.



- 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'appropriier des branches (code\_dbeauchemin).



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



# Agenda

---

- 1 Introduction
- 2 Ligne de commande
- 3 Git
- 4 Aller plus loin avec Git**
- 5 Être expert en Git



`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.



## Ajout d'une remote

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





## 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.



## Mise à jour d'une remote

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



Comment *revert* les changements à un fichier ?

```
git checkout -- fichier
```



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

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



Comment supprimer un fichier suivi par Git?

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



Comment annuler un commit non push ?

```
git reset HEAD^
```



Comment modifier le message d'un commit ?

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



Comment ajouter un fichier dans le commit précédent ?

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





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 locale.



Comment commit des portions de modification d'un fichier ?

```
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

Effectuer des changements dans le code à de multiple endroit.

Faites plusieurs commits en regroupant différentes modifications.

Mettez à jour le répertoire origin.



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.



Comment faire le ménage des branches avec un merge ?

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



Autres commandes intéressantes :

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





## 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

---

- 1 Introduction
- 2 Ligne de commande
- 3 Git
- 4 Aller plus loin avec Git
- 5 Être expert en Git**



## 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](#)



## 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](#)



## 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](#)



## git rebase

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

[Tutoriel rebase par Atlassian](#)

[Tutoriel merge vs rebasing par Atlassian](#)





## git reset soft, mixt and hard

explication reset 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

