

# Ligne de commande et gestion de versions avec Git



# Ligne de commande et gestion de versions avec Git

**Vincent Goulet**

Professeur titulaire  
École d'actuariat, Université Laval

Version 2019.10-2



Vincent Goulet, 2019

© 2019 par Vincent Goulet. « Ligne de commande et gestion de versions avec Git » est mis à disposition sous licence **Attribution-Partage dans les mêmes conditions 4.0 International** de Creative Commons. En vertu de cette licence, vous êtes autorisé à :

- **partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats ;
- **adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.

L'Offrant ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence.

Selon les conditions suivantes :



**Attribution** — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.



**Partage dans les mêmes conditions** — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous devez diffuser l'œuvre modifiée dans les mêmes conditions, c'est-à-dire avec la même licence avec laquelle l'œuvre originale a été diffusée.

#### Code source

 [Voir sur GitLab](#)

#### Illustrations de Git

Des illustrations de la deuxième partie proviennent de « Pro Git » de Scott Chacon et Ben Straub sous licence **CC BY-SA 3.0 non transposé**. Les illustrations de branches Git sont inspirées des tutoriels de Atlassian sous licence **CC BY 2.5 Australie**.

#### Couverture

Brebis manech tête rousse sur le Suhalmendi à Ascaïn, dans le Pays basque français. Brebis la plus productive des Pyrénées, le lait de la manech tête rousse est notamment utilisé dans la confection de la tomme d'appellation d'origine contrôlée ossau-iraty.

Crédit photo : © Basotxerri **CC BY-SA 4.0 International**, via [Wikimedia Commons](#).

## Ligne de commande

Introduction

Commandes essentielles

Transfert de données et redirection

## Gestion de versions avec Git

Introduction

Démarrer avec Git

Bases de Git

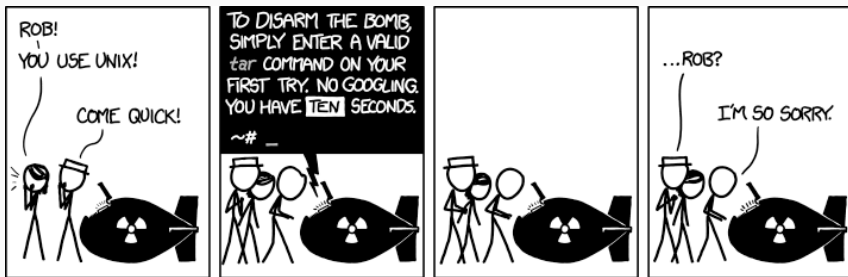
Branches

Aller plus loin

- Utiliser la ligne de commande Bash pour effectuer des opérations simples dans le système de fichiers
- Utiliser le système de contrôle de versions Git pour les projets informatiques, en particulier pour les travaux réalisés en équipe

Première partie

**Ligne de commande**



Tiré de [XKCD.com](https://xkcd.com)

# Introduction

---



# Un outil puissant toujours pertinent

La ligne de commande du système d'exploitation demeure une interface importante pour les programmeurs.

- Parfois **plus simple** qu'une interface graphique
- Souvent **plus rapide** qu'une interface graphique
- Parfois la **seule option** (notamment pour les utilitaires Unix comme **grep**, **sed**, **awk**)

**(Interface en) Ligne de commande** Mode d'interaction avec un programme informatique dans lequel l'utilisateur dicte les commandes et reçoit les réponses de l'ordinateur en mode texte (*command line interface*, CLI)

**Interpréteur de commandes** Programme qui gère l'interface en ligne de commande (*terminal*, *shell*)

**Invite de commande** Symbole affiché par l'interpréteur de commande pour indiquer qu'il est prêt à recevoir une commande (*command prompt*)

# Ligne de commande Windows

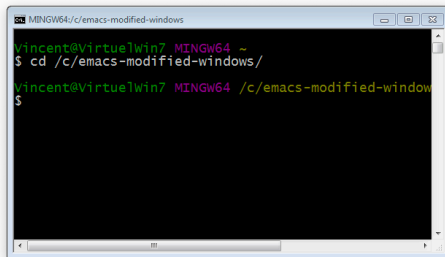
## cmd.exe

- Accessoires | Invite de commandes
- pas très puissant
- invite par défaut : **C:\>**



## Git Bash ou MSYS

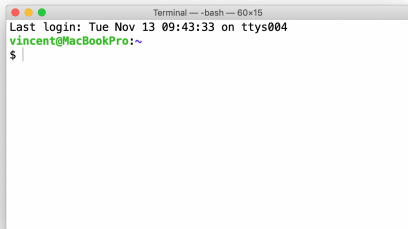
- interpréteur Bash très puissant
- standard Unix
- invite par défaut : **\$**



# Ligne de commande macOS

## Terminal

- Applications | Utilitaires | Terminal
- interpréteur Bash ou Z Shell (à partir de macOS 10.15 Catalina)
- invite par défaut : \$

A screenshot of the macOS Terminal application. The title bar reads "Terminal — bash — 60x15". The terminal content shows the last login information: "Last login: Tue Nov 13 09:43:33 on ttys004". Below this, the prompt "vincent@MacBookPro:~" is displayed in green, followed by a green dollar sign "\$" and a cursor. The terminal window has a light gray border and standard macOS window controls (red, yellow, green buttons) in the top-left corner.

```
Terminal — bash — 60x15
Last login: Tue Nov 13 09:43:33 on ttys004
vincent@MacBookPro:~
$
```



Vous pouvez modifier l'apparence de la ligne de commande dans les préférences de l'application Terminal

1. Démarrer la ligne de commande de votre système d'exploitation
2. Identifier le répertoire courant dans l'invite de commande
3. Repérer ce répertoire dans le système de fichier avec l'Explorateur Windows ou le Finder (garder la fenêtre ouverte)

# Commandes essentielles

---

# Rudiments de la ligne de commande

Nous allons uniquement étudier les commandes qui permettent de :

- se déplacer dans le système de fichiers
- afficher la liste des fichiers d'un répertoire
- afficher le contenu d'un fichier
- créer et supprimer un fichier

Commandes Bash (Unix) seulement.

## Afficher le répertoire courant

`pwd` (*print working directory*)

- affiche le chemin d'accès absolu du répertoire courant

```
~$ pwd  
/Users/vincent
```



# Changer de répertoire

**cd** (*change directory*)

- change le répertoire courant pour celui donné en argument
- argument peut être un chemin d'accès **absolu** ou **relatif**
- nom fictif « **.** » identifie le répertoire courant
- nom fictif « **..** » identifie le parent du répertoire courant
- sans argument, ramène au répertoire personnel « **~** »

```
~$ cd Desktop/  
/Users/vincent/Desktop  
~/Desktop$ cd ~/Documents/cours/  
~/Documents/cours$ cd  
~$
```

# Lister les fichiers

`ls` (*list*)

- affiche les fichiers du répertoire en argument
- sans argument, affiche les fichiers du répertoire courant

```
~$ ls  
Desktop  
Documents  
Downloads  
<...>
```

# Afficher le contenu d'un fichier

`cat` (*catenate*)

- affiche le contenu du fichier donné en argument
- utile uniquement pour les fichiers en texte brut

```
$ cat hello.txt  
Hello World!
```

# Créer et supprimer un fichier

## touch

- crée le fichier donné en argument
- fichier vide

```
$ touch foo  
$ cat foo  
$
```

## rm (*remove*)

- supprime le fichier donné en argument
- fichier disparu à jamais

```
$ rm foo
```

1. Afficher le répertoire courant. Comparer avec celui mentionné dans l'invite de commande.
2. Afficher la liste des fichiers du répertoire courant. Comparer avec la liste de l'Explorateur Windows ou du Finder.
3. Choisir un sous-répertoire du répertoire courant que vous savez contenir lui-même un sous-répertoire (par exemple : `Documents/Cours`).
4. Afficher, **sans d'abord s'y déplacer**, la liste des fichiers du premier sous-répertoire (`Documents`), puis celle du second (`Cours`).
5. Faire du sous-sous-répertoire ci-dessus (`Cours`) le répertoire courant.
6. Afficher une liste détaillée des fichiers du nouveau répertoire courant avec la commande `ls -l`.
7. Revenir au répertoire personnel avec une seule commande.
8. Afficher la liste de tous les fichiers du répertoire personnel, y compris les fichiers cachés, avec la commande `ls -a`.

## **Transfert de données et redirection**

---

Les programmes Unix en ligne de commande :

- reçoivent leurs données d'une **entrée standard** (*standard input, stdin*)
- écrivent leurs résultats vers la **sortie standard** (*standard output, stdout*)
- émettent leurs erreurs vers l'**erreur standard** (*standard error, stderr*)

Les programmes Unix en ligne de commande :

- reçoivent leurs données d'une **entrée standard** (*standard input, stdin*)
- écrivent leurs résultats vers la **sortie standard** (*standard output, stdout*)
- émettent leurs erreurs vers l'**erreur standard** (*standard error, stderr*)



Dans un terminal, l'entrée standard est le clavier et la sortie et l'erreur standards, l'écran



# Transfert de données

`stdout` | `stdin` (« tuyau », *pipe*)

- transfère la sortie d'un programme directement à l'entrée d'un autre programme
- premier programme passe son résultat en entrée au second programme
- similaire à la composition de fonctions  $g \circ f$  en mathématiques

```
~$ pwd
/Users/vincent
~$ pwd | cut -d / -f 3
vincent
```

# Redirection

*stdout* > *fichier*

- redirige la sortie standard d'un programme vers un fichier
- si le fichier existe déjà, son contenu est écrasé
- variante double >> pour ajouter le contenu à la fin du fichier

```
$ echo 'Hello World!' > hello.txt
$ cat hello.txt
Hello World!
$ echo 'Salut la compagnie!' >> hello.txt
$ cat hello.txt
Hello World!
Salut la compagnie!
```

## Redirection (suite)

*stdin < fichier*

- déverse le contenu d'un fichier dans l'entrée standard d'un programme
- variante double existe aussi

```
$ tr ! ? < hello.txt  
Hello World?  
Salut la compagnie?
```

Créer un fichier nommé **foobar** depuis la ligne de commande avec la commande :

```
$ echo 'aaa.bbb.ccc' > foobar
```

Déterminer ensuite le résultat des commandes suivantes.

1. `cat foobar`
2. `cat foobar | cut -d . -f 1`
3. `cat foobar | tr a z`
4. `tr a z < foobar`
5. `rm foobar`

Deuxième partie

## **Gestion de versions avec Git**

THIS IS GIT. IT TRACKS COLLABORATIVE WORK  
ON PROJECTS THROUGH A BEAUTIFUL  
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL  
COMMANDS AND TYPE THEM TO SYNC UP.  
IF YOU GET ERRORS, SAVE YOUR WORK  
ELSEWHERE, DELETE THE PROJECT,  
AND DOWNLOAD A FRESH COPY.



# Introduction

---

# Vous avez vécu cette situation

Quelle est la plus récente version de ce fichier?

foo\_20 septembre-final\_FINAL.doc vs  
foo\_18 septembre-corrigé-final.doc

Qui a la dernière version de ce fichier?

foo\_20 septembre-final\_VG\_FINAL.doc vs  
foo\_18 septembre\_VG-corrigé-final\_CD.doc

Comment éviter que Alexandre écrase le travail de Marianne?



# Vous avez vécu cette situation

Quelle est la plus récente version de ce fichier?

foo\_20 septembre-final\_FINAL.doc vs  
foo\_18 septembre-corrigé-final.doc

Qui a la dernière version de ce fichier?

foo\_20 septembre-final\_VG\_FINAL.doc vs  
foo\_18 septembre\_VG-corrigé-final\_CD.doc

Comment éviter que Alexandre écrase le travail de Marianne?

Pourquoi travaillons-nous comme ça?

**Les informaticiens ont résolu ces problèmes depuis longtemps avec les systèmes de gestion de versions**

# Fonctionnalités d'un système de gestion de versions



- Gérer l'ensemble des versions — ou **révisions** — d'un ou plusieurs fichiers, normalement en format texte brut
- Enregistrer la nature, la date et l'auteur de chaque révision
- Retourner à des révisions précédentes
- Comparer des révisions
- Fusionner des révisions
- Conserver une copie de sauvegarde dans un serveur central



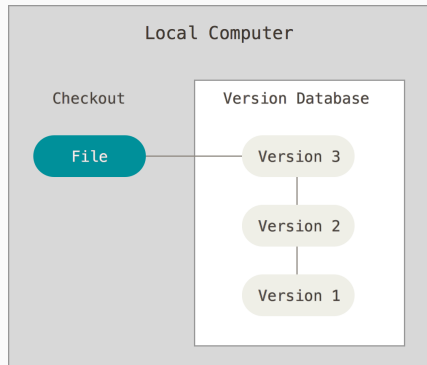
Un système de gestion de versions vaut le coup même pour vos projets individuels, ne serait-ce que pour la copie de sauvegarde et le suivi des versions

# Systèmes de première génération

Les systèmes de contrôle de version de première génération voient le jour dans les années 1970 et 1980 :



- **Source Code Control System**  (SCCS, 1972);
- **Revision Control System**  (RCS, 1982).

Ces systèmes ne permettent pas de travailler à plusieurs à la fois sur un même fichier.

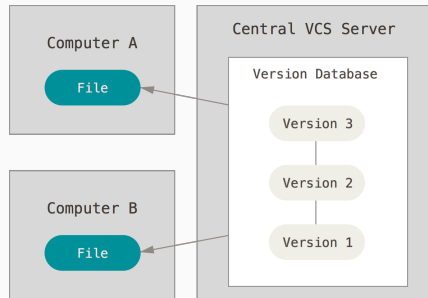


# Systèmes centralisés de seconde génération




Les systèmes centralisés se développent au cours des années 1990 et 2000 grâce à l'accès de plus en plus généralisé à la réseautique, notamment à Internet :

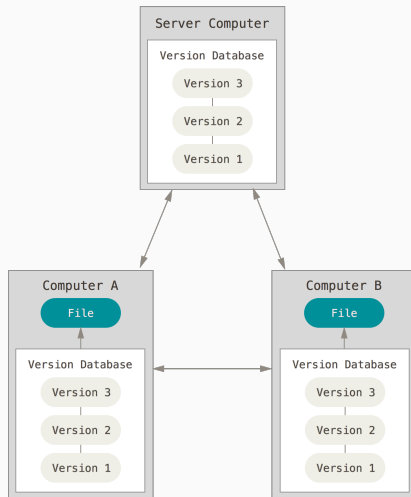
- **Concurrent Versions System**  (CVS, 1990);
- **Apache Subversion**  (SVN, 2000).

Subversion devient rapidement le standard *de facto*. Encore utilisé pour le code source de plusieurs projets.




Les systèmes distribués ne nécessitent pas un accès au serveur central pour la moindre opération de publication d'une modification à un fichier :

- **Git**  (2005)
- **Mercurial**  (hg, 2005)
- **Bazaar**  (bzd, 2005)



Git se prononce avec un « g » dur, « **guitte** ».

- Développé à l'origine par Linus Torvalds pour administrer le code source du noyau du système d'exploitation Linux
- Popularité foudroyante grâce, notamment, à **GitHub**  et son interface très conviviale
- Aujourd'hui le système de gestion de versions le plus utilisé dans le monde

Principaux autres sites d'hébergement gratuits : **GitLab** , **BitBucket** .

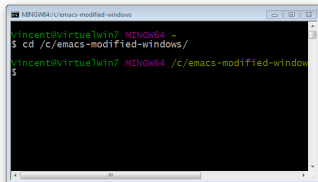


# Démarrer avec Git

---

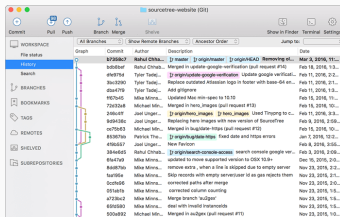
# Interface

## Débutant

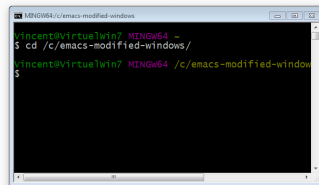


```
MINGW64/c/emacs-modified-windows
$ cd /c/emacs-modified-windows/
$ cd /c/emacs-modified-windows/
$
```

## Intermédiaire



## Expert



```
MINGW64/c/emacs-modified-windows
$ cd /c/emacs-modified-windows/
$ cd /c/emacs-modified-windows
$
```

## Deux types de fichiers

Pour Git il y a deux types de fichiers sur votre disque :

**Sous suivi** (*tracked*) le fichier fait partie de la liste des fichiers suivis par Git

**Non suivi** (*untracked*) Git ne s'occupe pas de ce fichier

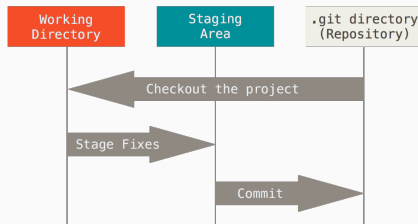
# Concept fondamental

Un fichier **sous suivi** est dans l'un ou l'autre de trois états :

**modifié** (*modified*) le fichier est modifié, mais n'a pas encore été publié dans le dépôt

**indexé** (*staged*) le fichier modifié a été marqué pour faire partie de la prochaine publication

**publié** (*committed*) le fichier est publié dans le dépôt local



La plupart des commandes de Git affichent des suggestions des prochaines étapes à suivre ou des correctifs à apporter si quelque chose n'a pas fonctionné.



Lisez les messages de Git et suivez les conseils!

## Votre identité

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

## Votre éditeur de texte

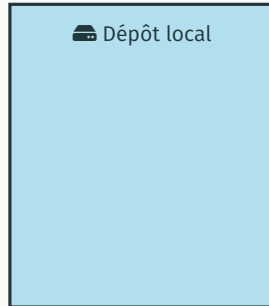
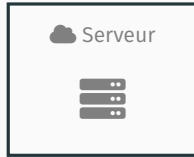
Windows

```
$ git config --global core.editor notepad
```

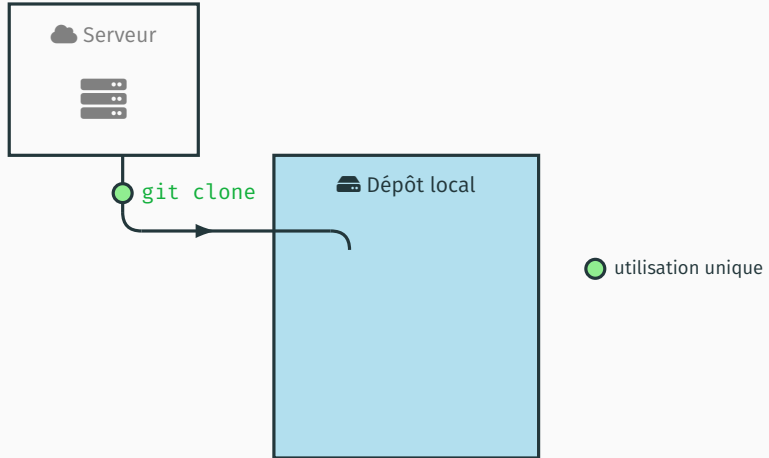
macOS

```
$ git config --global core.editor "open -W -n"
```

# Cycle de travail

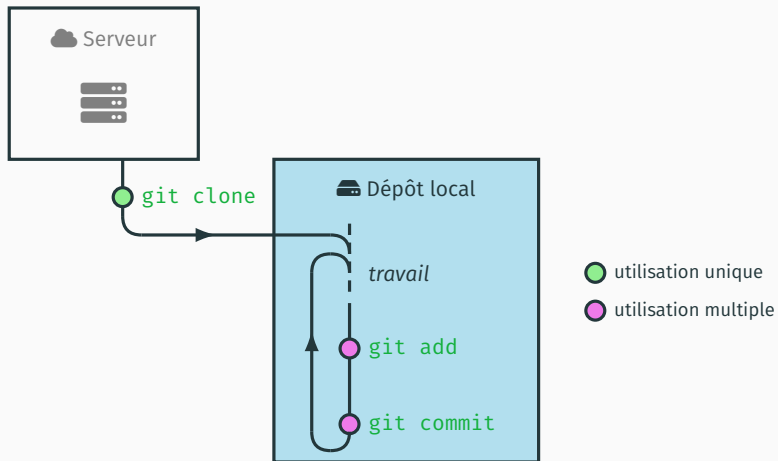


# Cycle de travail

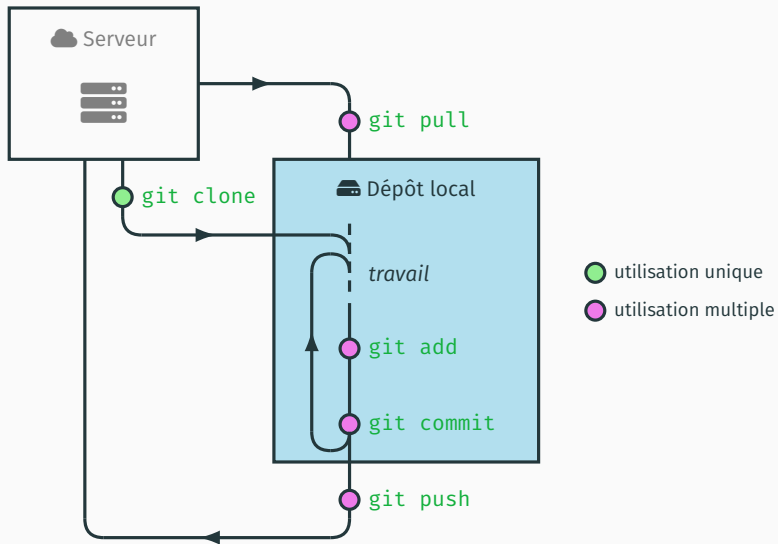




# Cycle de travail



# Cycle de travail





Suivez Marianne et Alexandre au fil de leur collaboration sur un projet avec Git

 Séance de travail collaboratif avec Git

# Bases de Git

---

# Démarrer un nouveau dépôt

Il y a plusieurs manières de démarrer un nouveau dépôt avec Git.

Voici de loin la plus simple.




Créez d'abord le dépôt dans le site web du serveur central  
(GitLab, BitBucket, GitHub, ...)

# Obtenir le dépôt

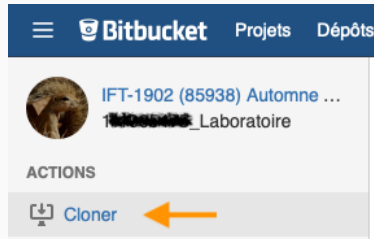
`git clone <url>`

- télécharge une copie du dépôt sur votre disque
- intégralité du dépôt, y compris l'historique
- répertoire contient uniquement des fichiers sous suivi



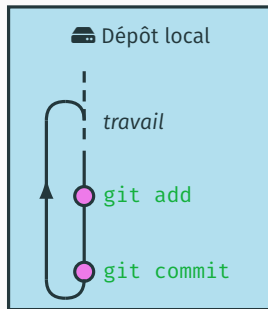
Cloner votre dépôt consacré à ce laboratoire dans le serveur **BitBucket de la Faculté des sciences et de génie** .

1. Accéder à la page de votre dépôt
2. Dans la barre latérale, sélectionner « Cloner » et copier l'url du dépôt
3. Coller l'url à la ligne de commande
4. Changer le répertoire pour celui du projet



# Travailler sur le projet

- Créer un nouveau fichier
- Modifier un fichier sous suivi
- Effacer un fichier sous suivi





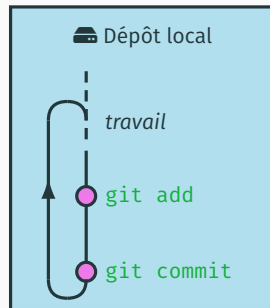
# Vérifier l'état des fichiers

## `git status`

- affiche l'état du dépôt
- indications des prochaines étapes
- liste des fichiers non suivis

## `git diff`

- affiche les changements dans les fichiers



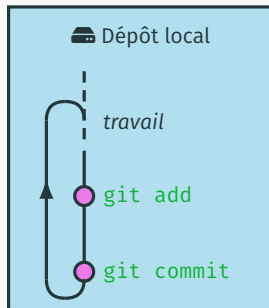
# Publier les modifications (localement)

`git add {fichiers}`

- indexe les *{fichiers}* en argument
- indique à Git les fichiers à publier
- statut des fichiers passe de *non suivi* ou *modifié* à *indexé*

`git commit`

- publie les fichiers indexés dans le dépôt **local**
- message de validation **obligatoire** et **important**
- prendre le temps d'écrire des messages **utiles**  
(**voici comment** [🔗](#))



Ajouter et publier un nouveau fichier dans le projet.

1. Créer un fichier quelconque (en texte brut) dans le répertoire du projet
2. Exécuter la commande `git status`
3. Exécuter la commande `git add`
4. Exécuter de nouveau la commande `git status`
5. Exécuter la commande `git commit` et écrire un message de validation dans votre éditeur
6. Exécuter la commande `git log -v`

# Partager son travail aux collaborateurs

## git push

- pousse les modifications locales vers le serveur
- copie de sauvegarde
- impossible d'écraser le travail **des autres**



# Récupérer le travail des collaborateurs

## `git pull`

- récupère (*fetch*) du serveur les ajouts au dépôt + fusionne (*merge*) les ajouts au dépôt local
- impossible d'écraser **votre** travail



Publier votre travail dans BitBucket.

1. Exécuter la commande `git push`
2. Vérifier dans l'interface web de BitBucket que le fichier se trouve bien dans le serveur

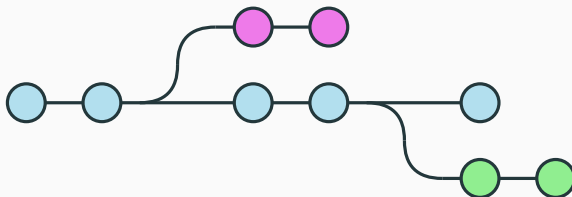
# Branches

---

# Au cœur de la philosophie d'utilisation de Git

Les branches permettent de disposer de plusieurs versions d'un projet.

- Développement d'une nouvelle fonctionnalité
- Test d'une correction
- Développement en parallèle avec des collaborateurs
- Travail à jeter après usage
- etc.



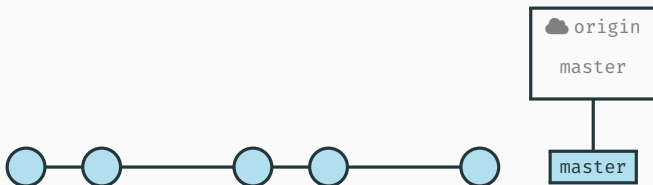


## Avant d'aller plus loin

La branche par défaut d'un projet est toujours **master**.

Lors du clonage d'un dépôt :

- le dépôt distant (*remote*) est identifié comme **origin**
- la branche **master** locale est reliée la branche **master** de **origin**



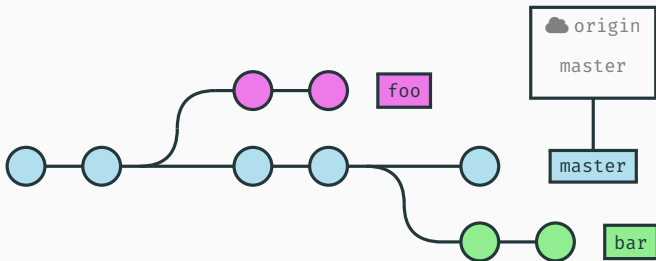
# Créer des branches et basculer de l'une à l'autre

`git checkout -b <foo>`

- crée la branche locale <foo> à partir de la **branche courante** + bascule vers cette branche

`git checkout <foo>`

- bascule vers la branche <foo>
- fichiers modifiés doivent être publiés ou remisés



# Administrer les branches

`git branch`

- affiche la liste des branches locales

`git branch <foo>`

- crée la branche `<foo>` sans quitter la branche actuelle

`git branch -d <foo>`

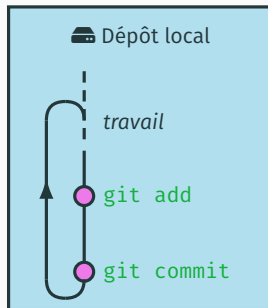
- supprime la branche `<foo>`

`git branch -m <foo> <bar>`

- change le nom de la branche `<foo>` pour `<bar>`

# Cycle de travail dans une branche

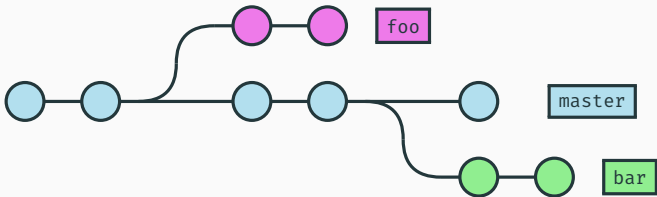
Le cycle de travail est exactement le même dans une branche `<foo>` que dans la branche `master`.



## ⚙ Exercice

Reproduire la structure de branches des illustrations dans votre dépôt local.

1. Créer une branche `foo` et basculer vers celle-ci
2. Modifier le fichier du dépôt et le publier (localement)
3. Revenir à la branche `master` et vérifier que le fichier ne contient pas la modification de la branche `foo`
4. Créer une branche `bar` à partir de `master` et basculer vers la nouvelle branche
5. Modifier le fichier du dépôt et le publier (localement)
6. Basculer directement vers la branche `foo`
7. Revenir à la branche `master` et afficher la liste de toutes les branches



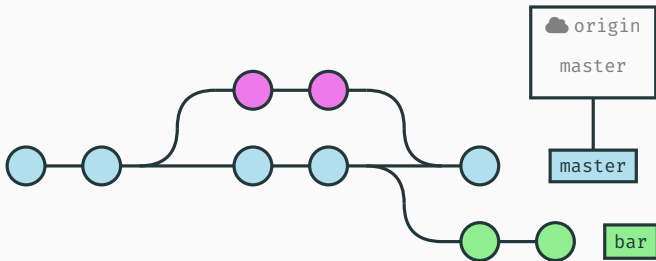
# Fusionner une branche

`git diff <foo> <bar>`

- affiche les différences entre les branches `<foo>` et `<bar>`

`git merge <foo>`

- fusionne la branche `<foo>` à la branche courante



Fusionner les changements de la branche `foo` effectués à l'exercice précédent.

1. S'assurer de se trouver dans la branche `master`
2. Examiner les différences entre les deux branches avec la commande `git diff`
3. Fusionner la branche `foo` à la branche `master`
4. Supprimer la branche `foo`

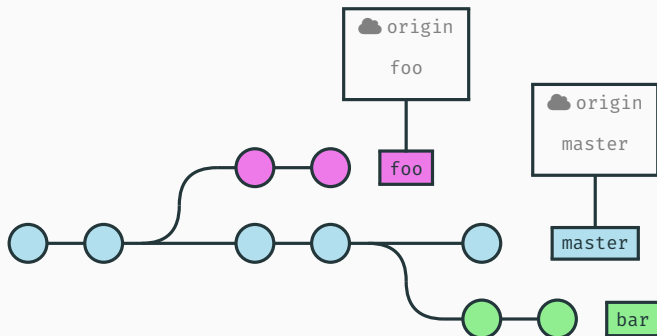
# Sauvegarder et partager une branche

`git push -u <origin> <foo>`

- relie la branche courante à la branche `<foo>` sur le serveur `<origin>`

`git push`

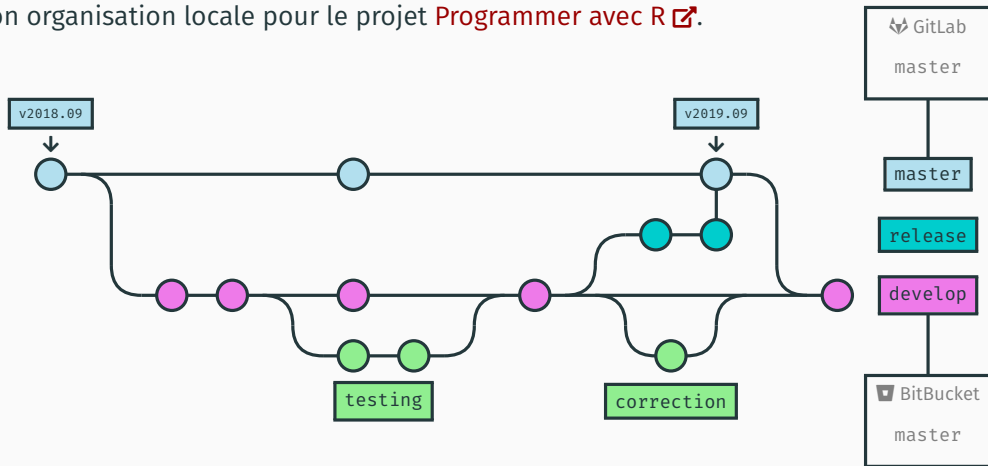
- pousse les modifications locales vers le serveur





## Exemple (avancé)

Mon organisation locale pour le projet [Programmer avec R](#).



**Aller plus loin**

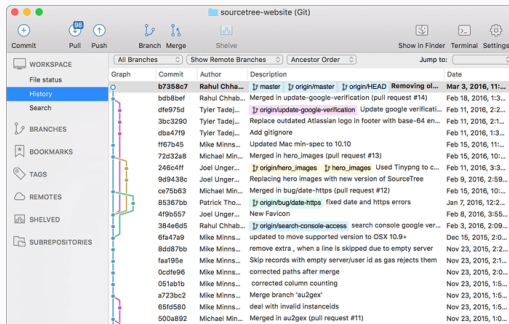
---

# Interfaces graphiques

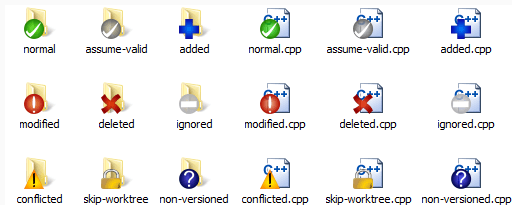
Vous voudrez utiliser une interface graphique pour travailler avec Git.

Il en existe **une multitude** .

## SourceTree



## TortoiseGit (Windows)



## Indiquer à Git d'ignorer certains fichiers

Inclure dans le dépôt un fichier `.gitignore` contenant des motifs de noms de fichiers à ignorer.

```
$ cat .gitignore  
*.aux  
*.log  
*.blg  
*.bbl  
*.out  
...
```

## Autres commandes utiles

### `git rm`

- efface le fichier sous suivi donné en argument

### `git mv`

- renomme un fichier sous suivi

### `git tag`

- étiquette des jalons dans l'historique de publication

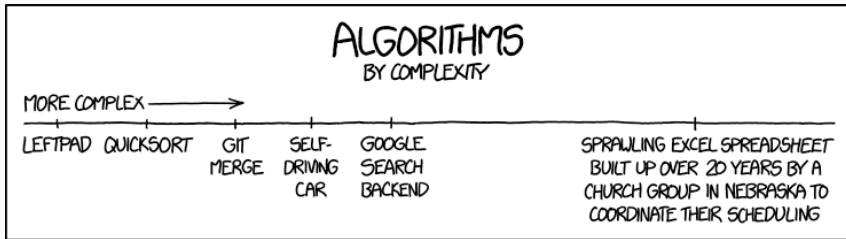
### `git log`

- affiche le journal (*log*) des publications

### `git stash`

- remise (ou met de côté) le travail non publié afin de pouvoir changer de branche

- [Pro Git](#), de Scott Chacon et Ben Straub. Excellente documentation officielle de Git. Disponible en plusieurs langues, dont le français.
- [Tutoriels de Atlassian](#), la compagnie derrière BitBucket. Ils sont particulièrement bien faits, souvent mieux encore que ceux de GitHub. En anglais.
- [How to teach Git](#) de Rachel M. Carmena. Une présentation alternative des concepts essentiels couverts dans ce laboratoire. Plusieurs liens utiles. En anglais.
- [StackOverflow](#). Gigantesque base de connaissances. Nous y allons tous un jour.



Tiré de [XKCD.com](https://xkcd.com)

Ce document a été produit par le système de mise en page  $\text{X}_{\text{Y}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  avec la classe **beamer** et le thème Metropolis. Les titres et le texte sont composés en Fira Sans, les mathématiques en Arev Math et le code informatique en Fira Mono. Les icônes proviennent de la police Font Awesome. Les illustrations ont été entièrement réalisées avec  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ .



