

Prise en main de [git](#) en ligne de commande

Git est un logiciel de gestion de versions décentralisé c-a-d un **DVCS** (**D**ecentralized **V**ersion **C**ontrol **S**oftware).

Un logiciel de gestion de versions est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.» (Extrait Wikipédia)

Le site de référence de git est : <https://git-scm.com>

Pour tout savoir sur git, un guide officiel (le Pro Book Git) est disponible en ligne en version française [ici](#).

Dans ce tutoriel, nous verrons comment :

- [Installer git](#)
- [Configurer git](#)
- [Créer un nouveau dépôt par initialisation](#)
- [Effectuer un premier commit](#)
- [Continuer à committer](#)
- [Créer une branche](#)
- [Se déplacer de branche en branche](#)
- [Fusionner deux branches \(fast-forward\)](#)
- [Travailler sur deux branches simultanément puis les fusionner](#)
- [Tagger un commit](#)
- [Gérer un conflit](#)
- [Visualiser un workflow à l'aide d'un éditeur graphique](#)
- [Supprimer une branche](#)
- [Cloner un dépôt existant distant](#)
- [Travailler en local sur un dépôt cloné](#)
- [Synchroniser les dépôts](#)

Il y aura aussi :

- [Des exercices interactifs en ligne pour les plus rapides...](#)
- [Annexe](#)

Ce tutoriel devrait vous permettre de prendre en main [git](#).

Pour cela, vous allez refaire pas à pas les exemples du cours d'Introduction à git disponible [ici](#).

N'hésitez pas à naviguer entre ce tutoriel et le cours

Installer git

Pour installer [git](#) sous votre environnement de travail, rendez-vous sur : **<https://git-scm.com/download>**

Pour vérifier que git est bien installé, ouvrir un terminal (console) et par taper : `git --help`

Remarque : vous pouvez éventuellement jeter un petit coup d'oeil [ici](#) pour savoir **How to properly set up Git on your computer!**

Configurer git

- **Sous Git, il n'y a pas de commit anonyme**, dans la console vous devez commencer par :
 - Configurer votre **nom** : `git config --global user.name "Prénom Nom"`
 - Configurer votre **email** : `git config --global user.email email@domaine.extension`
 - Pour **vérifier votre configuration**, vous pouvez taper :
`git config --global user.name`
`git config --global user.email`

Pour en savoir plus :

<https://help.github.com/articles/setting-your-username-in-git>

<https://help.github.com/articles/setting-your-commit-email-address-in-git>

<https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration>

<https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Param%C3%A9trage-%C3%A0-la-premi%C3%A8re-utilisation-de-Git>

- **Pour aider à la lisibilité des messages**, vous pouvez aussi activer la couleur.
`git config --global color.diff auto`
`git config --global color.status auto`
`git config --global color.branch auto`
- Le fichier contenant toutes ces configurations est le fichier : **.gitconfig**
Recherchez ce fichier dans votre répertoire personnel et ouvrez-le. Ce fichier contient pour l'instant les rubriques suivantes : **[user]**, **[color]** et peut-être **[filter "lfs"]**.

Ajoutez la rubrique **[alias]** configurée de la manière suivante :

```
[alias]
ci = commit
co = checkout
st = status
```

br = branch

Ces alias vous permettront de raccourcir certaines commandes de git très fréquemment utilisées.

Par exemple, au lieu d'écrire **git commit**, vous pourrez écrire **git ci**.

Vous pouvez bien sûr personnaliser ces alias à votre convenance et en créer de nouveaux pour d'autres commandes git.

Créer un nouveau dépôt par initialisation

- Dans votre explorateur de fichiers, commencez par **créer un répertoire demo**.
- **Si vous êtes sous Windows**, pour manipuler git, mieux vaut utiliser la console **git Bash**. Ouvrez la avant de continuer et travaillez dorénavant dans cette console.
- Depuis la console, placez-vous dans le répertoire demo que vous venez de créer et **initialisez un dépôt git sur ce répertoire** grâce à la commande : **git init**
 - Normalement la mention (**master**) s'affiche dans votre prompt à côté du chemin pour vous indiquer que ce répertoire est désormais sous contrôle du gestionnaire de version et que vous êtes actuellement sur la *branche principale master*.
 - Consultez le contenu du répertoire :
 - > Soit depuis l'explorateur de fichiers
 - > Soit directement depuis la console en utilisant la commande **ls -a** pour pouvoir visualiser les fichiers cachés.

Remarque : Dans le **git Bash**, vous devez utiliser les commandes Unix :

- Lien vers les principales commandes Unix : https://fr.wikipedia.org/wiki/Commandes_Unix
- Lien vers une liste très fournie de commandes unix : <http://cb.vu/unixtoolbox.shtml>

Attention, si vous êtes encore dans une simple console sous Windows, vous ne verrez pas le label (**master**), passez sous **git Bash** !!!
- Vérifiez qu'un répertoire **.git** a bien été créé !

Effectuer un premier commit

- Dans le répertoire **demo**, ajoutez un fichier **test.txt** dans lequel vous écrirez par exemple : **Bonjour tout le monde !**
- Pour que **git** puisse versionner ce fichier, vous devez :

○

- a. dans un premier temps, commencer par **ajouter ce fichier dans l'Index (add)** via le commande : `git add test.txt`

Remarque : Pour vérifier que ce fichier a bien été ajouté dans l'Index, vous pouvez effectuer la commande : `git status`.

Cette commande permet d'afficher l'état courant du dépôt. Le fichier `test.txt` devrait donc apparaître **en vert** comme un fichier prêt à être commité (*to be committed*).

○

- b. dans un second temps, **enregistrer réellement les modifications** contenues dans l'index via la commande `commit` suivi d'un **message explicite** indiquant l'objet du commit : `git commit -m "premier commit"`

Remarque : Pour vérifier que ce fichier a bien été commité, exécutez à nouveau : `git status` qui indique cette fois-ci qu'il n'y a plus rien à commiter (*nothing to commit*).

- Pour obtenir un **historique des commits**, tapez : `git log`
La liste des commits, réduit à un seul pour le moment, devrait s'afficher.
- Pour obtenir le **détail du dernier commit**, tapez : `git show`
Le détail du commit précédent devrait s'afficher.

Continuer à committer

- Ajoutez dans votre fichier `test.txt` la ligne suivante : **J'utilise git !**
N'oubliez pas d'enregistrer ce fichier avant de continuer 😊
- Vous pouvez comparer la dernière version committée et la version en cours de votre fichier.
Pour visualiser la **différence** entre ces deux versions, tapez : `git diff`
- Pour que **git** puisse versionner ce fichier c-a-d ajouter cette modification à l'historique des commits, il faut réaliser les deux étapes présentées précédemment, à savoir l'ajout du fichier dans l'index (**add**) avant de pouvoir réellement commiter.
Ces deux étapes peuvent être regroupées en une seule ligne de commande en utilisant l'option `-am`, l'option `-a` permettant d'ajouter dans l'index tous les fichiers trackés (c-a-d les fichiers qui ont déjà été ajoutés une fois dans l'index). Pour enregistrer ce nouveau commit en une seule instruction, tapez la commande suivante : `git commit -am "ajout du message sur git"`
- Demandez l'**historique des commits** avec `git log` pour visualiser les deux commits effectués jusqu'à présent.

- Pour obtenir le **détail du dernier commit**, tapez : `git show`

Créer une branche

Jusqu'ici vous avez travaillé sur une seule branche **master** qui est la branche principale, celle qui en général contient le « vrai » code source de votre projet.

- Pour voir toutes les branches, tapez la commande `git branch`
Pour l'instant, vous n'avez que la branche **master** 😊
- Nous allons créer une branche **anglais** pour internationaliser le texte du fichier `test.txt` !
Pour créer une branche, il faut utiliser la commande `branch`
Tapez : `git branch anglais`
- Visualisez toutes les branches avec la commande `git branch`
Le symbole étoile * indique la branche sur laquelle vous vous trouvez.
On est donc toujours sur **master** 😊
- Une fois la branche créée, il faut s'y positionner dessus si on souhaite enregistrer les commits sur cette branche.
Pour se déplacer sur une branche, il faut utiliser la commande `checkout`
Tapez : `git checkout anglais`
- Que s'est-il passé dans le prompt de la console ?
La mention **anglais** vous indique sur vous êtes désormais sur la nouvelle branche **anglais**.
Tapez `git branch` et vérifiez que l'étoile * désigne bien la branche **anglais**.
- Dans le répertoire **demo**, créez un nouveau fichier `test_EN.txt` que vous remplirez avec le texte suivant :

```
Hello world !  
I'm using git !
```
- Ajoutez ce nouveau fichier dans l'index via la commande : `git add test_EN.txt`
- Commitez ces modifications via l'instruction : `git commit -am "ajout du message en anglais"`
- Demandez l'**historique des commits** avec `git log`

Se déplacer de branche en branche

- Dans votre explorateur de fichiers, consultez le contenu de votre répertoire **demo** (ou tapez l'instruction `ls` dans la console).
Vous devez actuellement visualiser dans ce répertoire des fichiers `test.txt` et `test_EN.txt`.
- Pour vous repositionner sur la branche **master**, tapez dans la console : `git checkout master`
Si la commande s'est correctement exécutée, la mention **master** apparaît désormais dans le prompt de la console.

- Dans votre explorateur de fichiers, consultez le contenu de votre répertoire **demo** (ou tapez l'instruction `ls` dans la console). Vous devez visualiser dans ce répertoire uniquement le fichier **test.txt** puisque vous êtes actuellement sur la branche **master** et que le fichier **test_EN.txt** n'a été ajouté que dans la branche **anglais**.

Fusionner deux branches (fast-forward)

- Positionnez-vous sur la branche **master**
- Pour fusionner deux branches, il faut utiliser l'instruction : `merge`
Dans la console, tapez : `git merge anglais`
- Demandez l'**historique des commits** avec `git log`
Vous venez de faire une **fusion de type fast-forward**.
C'est la fusion la plus simple : elle ne crée pas un nouveau commit lors de la fusion, mais déplace seulement le sommet de la branche **master** puisqu'ici seules des modifications ont eu lieu sur une seule branche (la branche **anglais**) et le dernier commit de la branche **master** est resté inchangé.
- Dans votre explorateur de fichiers, consultez le contenu de votre répertoire **demo** (ou tapez l'instruction `ls` dans la console). Vous devez actuellement visualiser dans ce répertoire les deux fichiers **test.txt** et **test_EN.txt**, ce qui confirme qu'ils ont bien été tous deux fusionnés dans la branche **master**.

Travailler sur deux branches simultanément puis les fusionner

- Tapez `git branch`
Vous êtes actuellement sur la branche **master**.
Notez au passage, que la branche **anglais** continue à exister dans votre historique...

Nous allons maintenant faire évoluer l'état de ces branches en faisant apparaître de nouveaux commits sur chacune des branches, puis nous fusionnerons le tout...

- Assurez-vous d'être sur la branche **master**
Ouvrez le fichier **test.txt** et ajoutez dans ce fichier la ligne suivante :
git est un logiciel de gestion de versions décentralisé.
- Enregistrez ce fichier, puis **committer** avec le message suivant : **"ajout définition git en français"**
- Demandez l'**historique des commits** avec `git log`

- Positionnez-vous maintenant sur la branche **anglais**
Ouvrez le fichier **test_EN.txt** et ajoutez dans ce fichier la ligne suivante :
git is a distributed version control system.
- Enregistrez ce fichier, puis **committer** avec le message suivant : "**ajout définition git en anglais**"
- Demandez l'**historique des commits** avec **git log**
- Positionnez-vous maintenant sur la branche **master**
- Fusionner les deux branches en tapant : **git merge anglais**
Attention, comme il y a eu des modifications (nouveaux commits) sur les deux branches à fusionner, un commit de fusion est nécessaire pour centraliser le tout.
La fusion va donc donner naissance à un nouveau commit qui doit avoir son propre message ...
- L'éditeur s'ouvre donc dans la console pour vous permettre d'ajouter le message de commit.
A la place de **Merge branch 'anglais'**, écrire le nouveau message de commit :
Fusion définition git en anglais
Attention, vous êtes dans un éditeur de type **vim** : pour enregistrer vos modifications vous devez faire **Escape** puis **:wq** puis **Entrée**.
Remarque : Un lien vers la liste des commandes **vim** : <https://doc.ubuntu-fr.org/vim>
A retenir plus particulièrement :
:w pour enregistrer le fichier
:q pour quitter vi
:wq pour enregistrer le fichier et quitter vi
:q! pour quitter vi en annulant les changements
- Demandez l'**historique des commits** avec **git log**
La fusion a donc créé un nouveau commit faisant apparaître le message que vous venez de saisir.

Tagger un commit

- Vérifiez que vous êtes toujours bien positionner sur la branche **master**, si ce n'est pas le cas positionnez-vous dessus.
- Pour tagger le commit courant, il faut utiliser l'instruction **tag**
Tapez l'instruction suivante : **git tag v1.0.0**
- Demandez l'**historique des commits** avec **git log**
Vous devriez visualiser le tag sur le dernier commit.

A quoi sert un tag ?

Un tag permet de repérer plus facilement un commit.

Un commit est habituellement identifié via son hash.

Pour se déplacer sur un commit, il faut utiliser l'instruction **checkout** suivi du **hash**.

Si un tag a été placé sur un commit, il est possible de se repositionner sur ce commit à l'aide de l'instruction **checkout** suivi du **nom du tag**.

Gérer un conflit

Un conflit peut apparaître lors d'une fusion entre deux branches.

Un conflit pourra apparaître si les deux branches ont évoluées chacune de leur côté avec des modifications qui ne sont pas compatible.

Nous allons illustrer cela à l'aide de l'exemple suivant.

1. Créer une nouvelle branche

- Nous allons créer une nouvelle branche **español**.
Il est possible de créer une nouvelle branche et s'y positionner directement dessus en faisant appel à l'instruction **checkout** suivie de l'option **-b** (au lieu d'un appel à **branch** puis d'un appel à **checkout**)
Tapez l'instruction suivante : **git checkout -b español**
- Visualisez toutes vos branches avec la commande **git branch** et vérifiez que l'étoile * indique bien la branche **español**.
- Sur la branche **español**, dans le répertoire **demo**, créez un nouveau fichier **test_ES.txt** que vous complétez avec le texte suivant :

```
Hola Mundo !  
git fue creado por Linus Torvalds en 2005.  
Estoy usando git !  
git es un software de control de versiones.
```

Enregistrez ce fichier.

- Toujours sur la branche **español**, ouvrez le fichier **test.txt** et corrigez une petite erreur au passage 😊 c-a-d remplacez la première ligne **Bonjour tout le monde !** par **Bonjour le monde !** (c-a-d supprimer le mot **tout**)
- Faites un **git status**
Cette commande vous indique entre autres que le nouveau fichier **test_ES.txt** n'est pas encore tracké.
Ajoutez ce nouveau fichier **test_ES.txt** dans l'index via la commande : **git add test_ES.txt**
- Committez toutes les modifications précédentes via l'instruction : **git commit -am "ajout git en español"**

2. Continuer à travailler sur la branche master

- Positionnez-vous maintenant sur la branche **master**

- Ouvrez la fichier `test.txt`
Ajoutez comme 2ème ligne de ce fichier (après Bonjour tout le monde !), la ligne suivante :
`git a été créé par Linus Torvalds en 2005.`
Enregistrez ce fichier.
- Ouvrez la fichier `test_EN.txt`
Ajoutez comme 2ème ligne de ce fichier (après Hello world !), la ligne suivante :
`git was created by Linus Torvalds in 2005.`
Enregistrez ce fichier.
- **Committez** ces deux modifications en indiquant le message suivant : "**ajout création de git**"
- Demandez l'**historique des commits** avec `git log`

3. Fusionner les deux branches et gérer le conflit...

- En étant sur la branche **master**, demandez la fusion de la branche **espagnol** en tapant l'instruction suivante : `git merge espagnol -m "fusion message en espagnol"`

Un conflit (**CONFLICT**) est détecté.

git détecte un conflit lorsque deux personnes modifient la même zone de code en même temps.

Comme il ne peut pas décider seul quel code est le bon et doit être gardé, il signale un conflit.

git met alors la fusion en pause ce qui est indiqué dans le prompt par

: (master|MERCING)

Pour voir les fichiers en conflit, vous pouvez taper : `git status`

Le `git status` indique deux solutions pour résoudre ce problème de fusion :

--> solution n°1 : résoudre le conflit puis redemander un `git commit`

--> solution n°2 : abandonner la fusion avec un `git merge --abort`

Nous allons mettre en place la solution n°1, c-a-d résoudre le conflit pour mener la fusion à son terme.

3.a Résoudre manuellement le conflit de manière à obtenir le fichier souhaité

- Pour résoudre le conflit, git a modifié un fichier pour montrer les différences entre la branche en cours et celle que vous essayez de merger. C'est pour cela que lors du `git status`, le fichier `test.txt` apparaît en **rouge** : c'est le fichier qui a été modifié par git (**modified**).
- Ouvrir ce fichier via l'instruction : `vim test.txt`
Recherchez la ligne contenant les symboles <<<<<<<<<
Ces symboles indiquent le début du conflit et délimitent le contenu de la

branche de base (**HEAD**) en conflit.

Les symboles ===== permettent de séparer les deux contenus en conflit.

Les symboles >>>>>>> indiquent la fin du contenu en conflit sur la branche à fusionner (ici la branche **espagnol**)

Il ne reste donc plus qu'à *nettoyer* ce fichier pour qu'il affiche uniquement le contenu à committer c-a-d supprimer les marqueurs de conflits (<, =, >), les noms de branches et procéder à une fusion manuelle de la *bonne* version à commiter.

- Nettoyez le fichier **test.txt** pour qu'il ne contienne plus que :

```
Bonjour le monde !
git a été créé par Linus Torvalds en 2005.
J'utilise git !
git est un logiciel de gestion de versions décentralisé.
```

- Puis enregistrez vos modifications et quittez l'éditeur **vim** à l'aide de **Escape** puis **:wq** puis **Entrée**.

3.b Avertir git que le conflit a été résolu

Une fois le conflit résolu, il faut procéder à un nouveau commit pour terminer la fusion.

Il faut donc commencer par avertir git que le conflit a été résolu en ajoutant le fichier modifié dans l'index.

Pour cela, tapez l'instruction : **git add test.txt**

Et vérifier à l'aide d'un **git status** que git a bien pris en compte que le conflit a été résolu.

3.c Terminer la fusion par un commit

Il ne reste plus qu'à terminer la fusion en (re)demandant le commit de fusion.

Pour cela, tapez : **git commit -m "fusion message en espagnol"**

Si le commit s'est bien passé, vous devriez avoir un prompt qui affiche à nouveau seulement (**master**)

Remarque : Vous pouvez également vérifier la prise en compte de ce commit avec un petit **git log** ou **git show**

Visualiser un workflow à l'aide d'un éditeur graphique

Un outil graphique peut simplifier la visualisation et la gestion du workflow.

Il existe plusieurs outils graphiques. Vous en trouverez des listes [ici](#) ou [là](#).

Dans ce tutoriel, nous allons par exemple installer **gitKraken**, mais vous pouvez très bien choisir un autre outil graphique dans les listes précédentes si ça vous fait plaisir 😊

Rendez-vous sur le site : <https://www.gitkraken.com>

Téléchargez-le et installez-le.

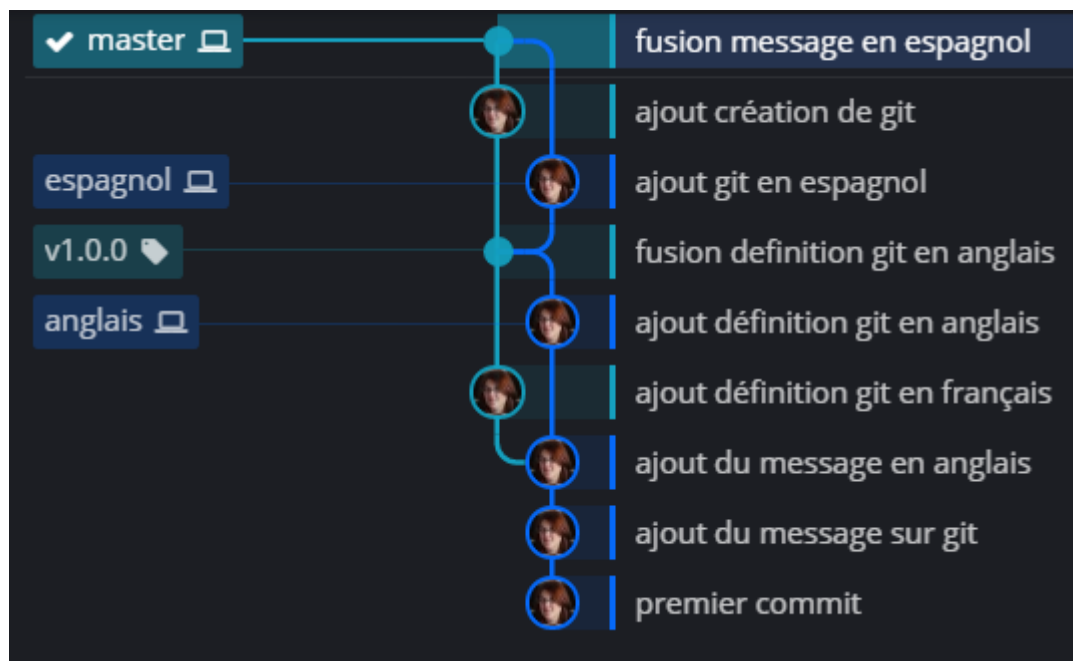
Cliquez sur l'icône répertoire en haut à gauche (au dessous du menu **File**) pour ouvrir un répertoire.

Choisir **Open**, puis **open a repository**.

Sélectionnez votre répertoire **demo**.

Votre workflow s'affiche dans la fenêtre.

Pour *bien* faire apparaître les branches : placez-vous sur un trait vertical (une double flèche s'affiche) et tirez de manière à obtenir :



Cliquez sur un commit. Son détail s'affiche à droite.

Lorsque ce détail est affiché, vous pouvez cliquer sur un fichier pour voir son contenu (**File View**) ou son diff par rapport au commit précédent (**Diff View**)

Remarque : GitKraken (et tout ce genre d'outil graphique) vous permet de manipuler facilement votre workflow. Nous ne nous attarderons pas plus que cela dans le cadre de ce tutoriel, mais si vous voulez en savoir plus sur les possibilités de GitKraken, n'hésitez pas à visualiser (un peu plus tard) les vidéos suivantes :

- [GitKraken Tutorial: For Beginners \(4 minutes\)](#)
- et [comprendre git : GitKraken \(12 minutes\)](#)

Supprimer une branche

Lorsque vous fusionnez une branche, cette branche existe toujours dans votre workflow.

Par exemple, si vous tapez dans la console **git branch**, vous visualisez que les branches **anglais**, **espagnol** et **master** sont présentes et donc toujours accessible.

Pour supprimer une branche, il faut utiliser l'option **-d** de la commande **branch**

Par exemple, pour supprimer la branche **anglais**, tapez dans la console : **git branch -d anglais**

Faites maintenant un **git branch** et vous constaterez que seules les branches **master** et **espagnol** sont désormais connues.

Remarques :

- Si vous visualisez votre workflow à l'aide de GitKraken, vous constaterez bien sûr également que la branche **anglais** a disparu du workflow 😊
- Il peut être utile de supprimer des branches pour nettoyer le **workflow** (l'historique).
Des commandes telles que **rebase** permettent par exemple de réorganiser le workflow.
Nous ne travaillerons pas sur ces commandes dans le cadre de ce tutoriel, si vous êtes intéressés des références à propos de merge et rebase vous sont données dans le cours 😊

Et pour la suite ?

Créer un nouveau dépôt git peut se faire de deux manières :

- Soit il s'agit d'un nouveau dépôt, l'instruction **git init** est utilisée pour mettre en place le contrôleur de version : c'est ce que vous devez faire sur vos nouveaux projets.
- Soit il existe déjà un dépôt git du projet sur lequel vous travaillez et vous souhaitez récupérer ce dépôt (son historique) pour travailler en local, dans ce cas-là il faudra **cloner le dépôt existant**, c'est que nous allons voir maintenant avec **git clone**.

Cloner un dépôt existant distant

Cloner un dépôt existant consiste à récupérer tout l'historique et tous les codes source d'un projet en utilisant l'instruction **git clone**

- Depuis la console, placez-vous dans votre répertoire de travail à l'endroit où vous souhaitez cloner votre nouveau dépôt (sortez par exemple de **demo** avec l'instruction : **cd ..**).
Le prompt ne vous indique plus **master** puisque vous n'êtes plus (ou pas encore 😊) dans un dépôt git ...

- Vous allez récupérer en local le dépôt existant distant : <https://github.com/iblasquez/geekgit>
Pour cela, tapez l'instruction : `git clone https://github.com/iblasquez/geekgit.git`
- Rendez-vous ensuite dans le répertoire `geekgit` qui vient d'être créé : `cd geekgit`
Le prompt vous indique que vous vous trouvez sur **master** !
- Consultez l'historique des commits en tapant `git log`
Ceci vous confirme bien que lors d'un `git clone`, git fournit à chaque développeur sa propre copie du dépôt avec son propre historique.
Remarque : si vous le souhaitez, vous pouvez aussi visualiser l'historique de ce dépôt avec GitKraken 😊

Travailler en local sur un dépôt cloné

Faites-vous plaisir, effectuez des modifications en local dans ce dépôt en prenant soin de commiter régulièrement et de manipuler toutes les commandes git vues précédemment ...

Synchroniser les dépôts

Nous ne traiterons pas cette partie dans le cadre de ce tutoriel : elle sera mise en pratique lors de la séance suivante.

Prenez quand même le temps de consulter les transparents du cours relatifs à la partie **Synchroniser les dépôts** et notez bien que :

- grâce à l'instruction `git push origin master`, vous pourriez **publier vos modifications** (dépôt local) vers le dépôt distant (remote) c-a-d le dépôt que vous venez de cloner.
- grâce aux instructions `git pull origin`, vous pourriez **récupérer les commits du dépôt distant** c-a-d mettre à jour votre historique local. Cette instruction revient à réaliser deux commandes :
 - la première qui consiste à télécharger les commits du dépôt distant (`git fetch`)
 - et la seconde qui consiste à fusionner ces commits avec votre historique local (`git merge`).

Des exercices interactifs en ligne pour les plus rapides...

- Commencez par la manipulation autour des branches :
[Learn Git Branching](#) : apprendre le branching sous git

- Continuez par la simulation d'un workflow et la manipulation/visualisation de commandes comme revert ou cherry-pick :
[Visualizing git](#)
- Terminez par [Git-It](#) : application multi-plateforme ([à installer](#)) proposant des défis utilisant *vraiment* git et GitHub sans passer par un émulateur...

Remarques :

- Les précédentes ressources sont accessibles sur la page [Resources to learn git](#)
- [Git-Gud](#) est un autre simulateur web très simple d'utilisation pour manipuler/visualiser les commandes de bases de git.

Annexe

- **Pro Git Book** (accessible en ligne)
 - Version française [ici](#)
 - Version anglaise [là](#)
- **Editeur vim**
 - Attention, par défaut la console **git Bash** va vous ouvrir l'éditeur **vim**
Un lien vers la liste des commandes **vim** : <https://doc.ubuntu-fr.org/vim>.
A retenir plus particulièrement :
 :w pour enregistrer le fichier
 :q pour quitter vi
 :wq pour enregistrer le fichier et quitter vi
 :q! pour quitter vi en annulant les changements
 - Sachez que si cet éditeur ne vous convient pas, il est possible d'en configurer un autre via la commande : **git config --global cores.editor nomEditeur**
Pour en savoir plus rendez-vous [ici](#)
- **Autres commandes git qui pourraient vous être utile...**
 - **git commit --amend** pour pouvoir modifier le dernier message de commit
 - **git stash**
 -pour éviter d'avoir à faire un commit au milieu d'un travail en cours : les fichiers sont sauvegardés et mis de côté (le *working directory* apparaîtra alors comme propre si vous faites un **git status**)
 -**git stash apply** pour vous permettre de restaurer les fichiers *stashés* :

les fichiers seront restaurés et retrouveront l'état dans lequel ils étaient avant git stash

- **Autres liens à consulter**

- [Essential git commands every developer should know](#)
- Théorie des graphes appliqués à git : [la vidéo](#) et [les transparents](#)