



DÉCOUVRIR LE VERSIONNING AVEC GIT

# GITHUB



Git

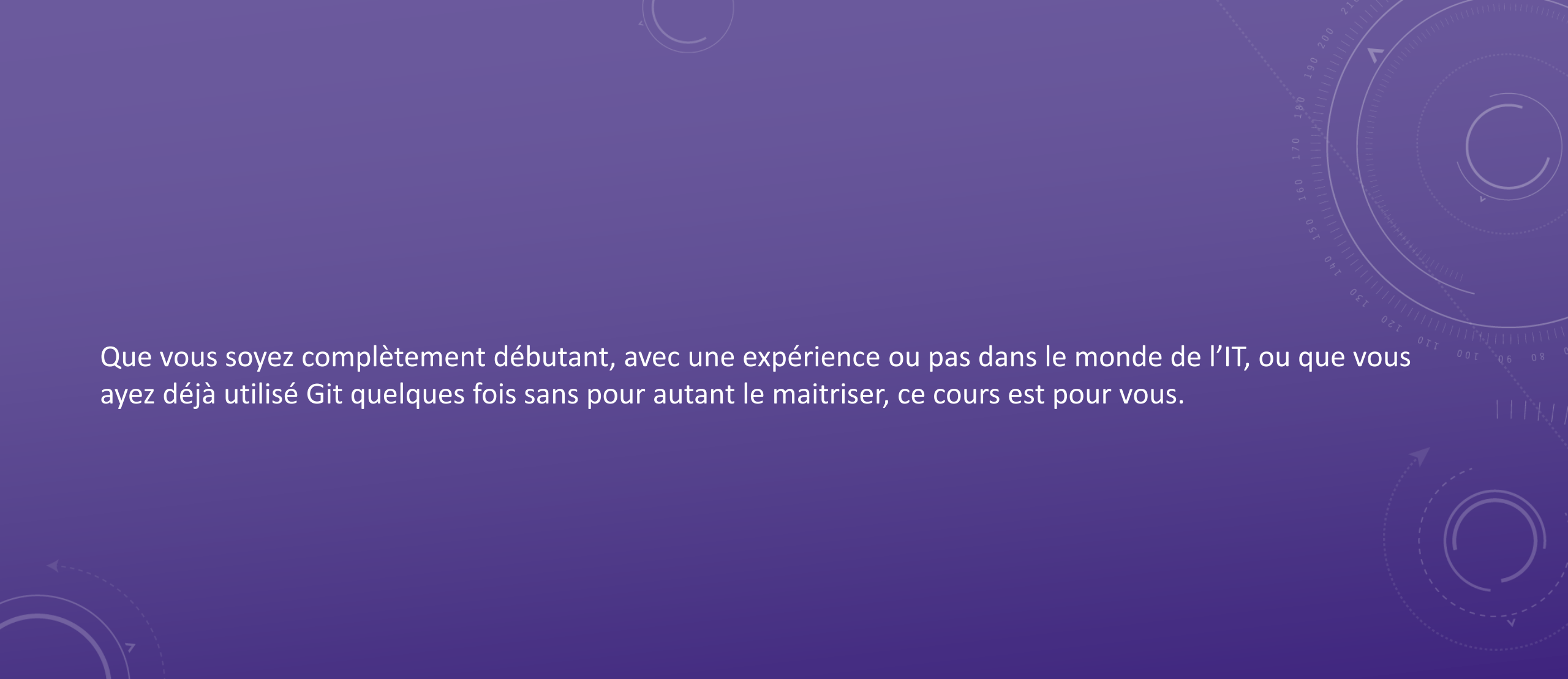


GITHUB



GIT  
Actions





Que vous soyez complètement débutant, avec une expérience ou pas dans le monde de l'IT, ou que vous ayez déjà utilisé Git quelques fois sans pour autant le maîtriser, ce cours est pour vous.

AUDIENCE



# MOTIVATION DERRIÈRE GIT

- Vous avez déjà fait l'une des choses suivantes en travaillant individuellement :
  - Vous aviez un code qui fonctionnait, vous avez fait un tas de changements, ce qui a cassé le code, et maintenant vous voulez juste récupérer l'ancienne version de travail qui marchait...
  - Suppression accidentelle d'un fichier critique, disparition de centaines de lignes de code...
  - Vous avez en quelque sorte sali la structure/contenu de votre base de code, et vous voulez juste "défaire" l'action folle que vous venez de faire.
  - Crash du disque dur !!!! Tout est parti, la veille de l'échéance.

# MOTIVATION DERRIÈRE GIT

- Avez-vous déjà fait l'une des choses suivantes en travaillant en équipe ?
  - A qui appartient l'ordinateur qui stocke la copie "officielle" du projet ?
  - Serons-nous en mesure de lire/écrire les modifications de chacun ?
  - Des fichiers de codes modifié envoyés par mails dans tous les sens.
  - Nous sommes deux à vouloir modifier le même code.
  - Un membre vient d'écraser un fichier sur lequel j'ai travaillé pendant 6 heures !
  - Nous avons corrompu un fichier important.
  - Comment puis-je savoir sur quel code chaque coéquipier travaille ?

## SOLUTION :

Système de contrôle de version :

Logiciel qui permet de suivre et de gérer les modifications apportées à un ensemble de fichiers et de ressources.

# PRÉSENTATION DE L'OUTIL

- Service propriétaire Microsoft de gestion de versions collaborative lancé en 2008.
- Nombre d'inscrits  $\approx$  50 millions
- Service web d'hébergement et de gestion de développement de logiciels.
- GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres.
- Une sécurité accrue : Les packages peuvent être publiés en privé, au sein de l'équipe, ou publiquement à la communauté open-source.



# LES PLUS DE L'OUTIL

- Une sécurité accrue : Les packages peuvent être publiés en privé, au sein de l'équipe, ou publiquement à la communauté open-source.
- Sécurité accrue du code : GitHub utilise des outils pour identifier et analyser les vulnérabilités du code que d'autres outils ont tendance à manquer.
- Gestion efficace des équipes : GitHub aide l'équipe à rester sur la même longueur d'onde et organisés. Les outils de modération comme le verrouillage des demandes d'émission et d'extraction aident l'équipe à se concentrer sur le code.
- Amélioration de la qualité du code : Les demandes d'extraction aident les organisations à examiner, développer et proposer un nouveau code. Les membres de l'équipe peuvent discuter de toute mise en œuvre et proposition avant de modifier le code source.







Certains font la comparaison entre Git et les poupées russes dans le principe

# LES CONCURRENTS DE GITHUB

Le marché offre de nombreuses alternatives et concurrents à GitHub. Parmi les meilleurs choix, on peut citer :

1. Bitbucket
2. Microsoft Team Foundation Server
3. Phabricator
4. GitLab
5. Assembla
6. Beanstalk
7. SourceForge
8. Helix Core



## Langages de programmation supportés par Github

Language	2020 Ranking	2019 Ranking	2018 Ranking
JavaScript	1	1	1
Python	2	2	3
Java	3	3	2
TypeScript	4	7	4
C#	5	5	6
PhP	6	4	4
C++	7	6	5
C	8	9	8
Shell	9	8	9
Ruby	10	10	10

Source: [GitHub](#)

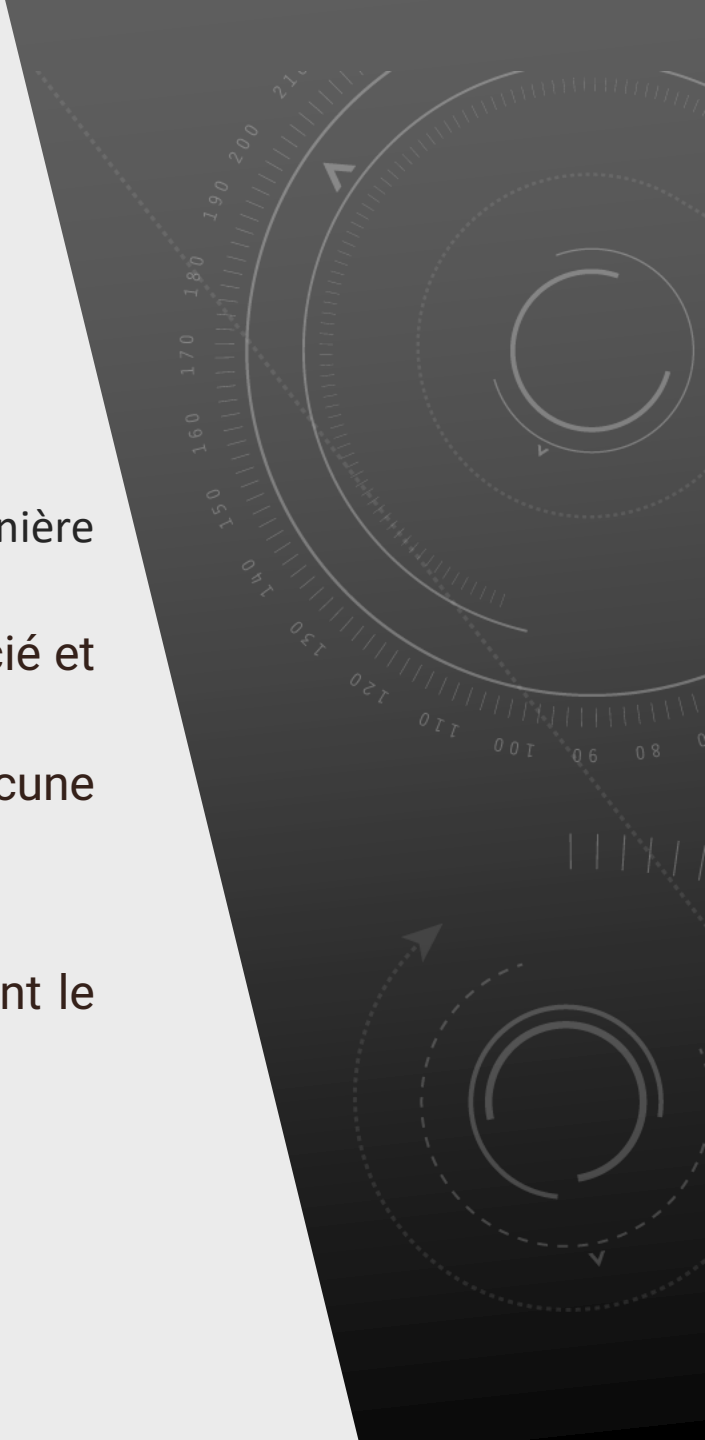
**Dice**

# QUELQUES FONCTIONNALITÉS

- Hébergement de projets.
- Partage de projets.
- Octroi d'accès en écriture aux projets.
- Gestion de versions
- Possibilité de suivre des personnes (des projets).
- Possibilité de commenter les projets.
- **L'intégration continue**

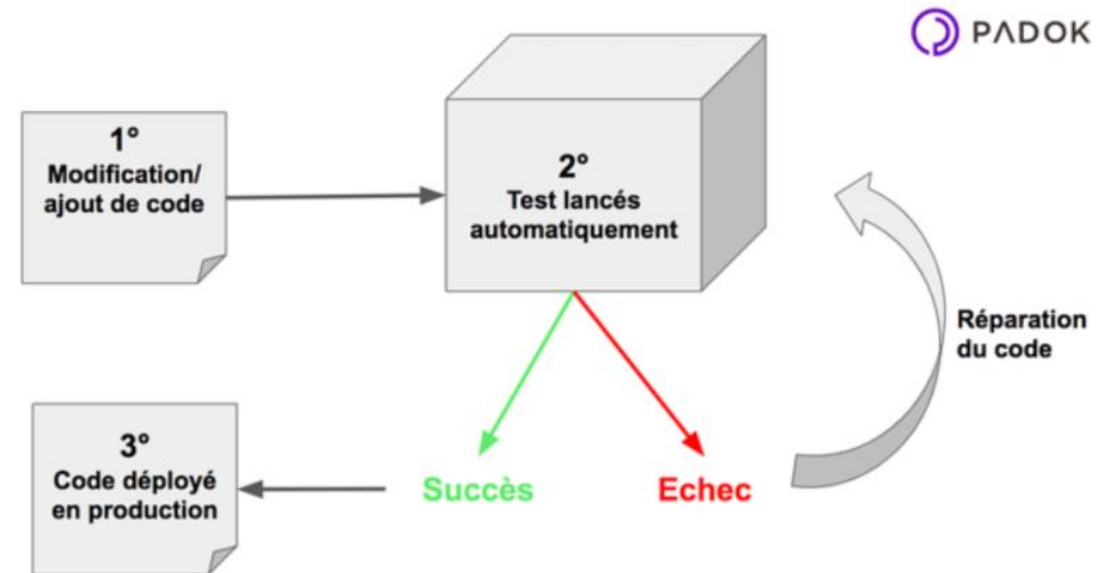
# INTÉGRATION CONTINUE

- L'intégration continue est un ensemble de pratiques consistant à tester de manière automatisée chaque révision de code avant de le déployer en production.
- Lorsque le développeur code une fonctionnalité, il conçoit également le test associé et ajoute le tout à son dépôt de code.
- Le serveur d'intégration va ensuite faire tourner tous les tests pour vérifier qu'aucune régression n'a été introduite dans le code source suite à cet ajout.
- Si un problème est identifié, le déploiement n'a pas lieu et les Dev sont notifiés.
- Si aucune erreur n'est remontée, le serveur d'intégration peut déployer directement le code en production.
- Ainsi, avec l'intégration continue la phase de tests automatisés est complètement intégrée au flux de déploiement.



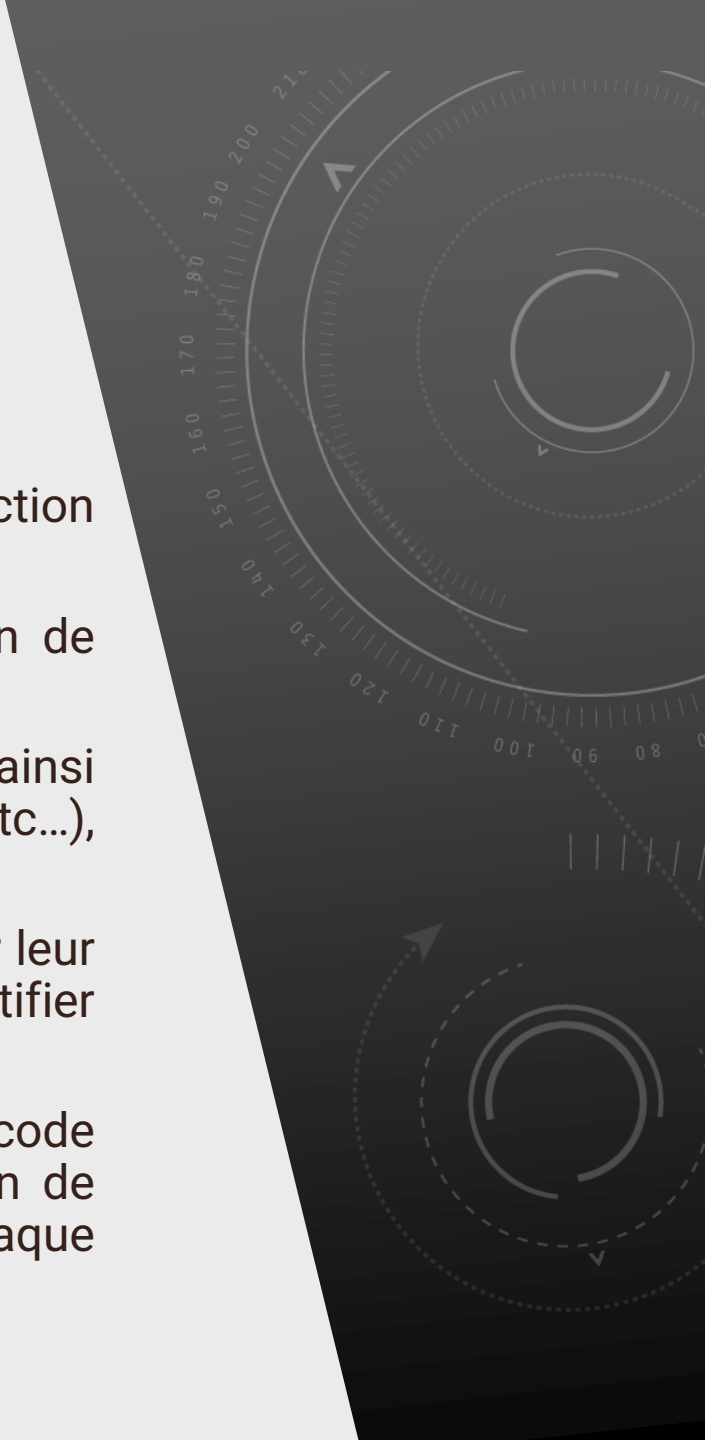


# INTÉGRATION CONTINUE



# INTÉGRATION CONTINUE

- Bénéfice : Garantir en production un code de qualité, et donc une meilleure satisfaction des utilisateurs finaux.
- L'automatisation des tests de tout le code source à chaque ajout/modification de fonctionnalités permet d'éviter l'introduction de régressions en production.
- Les développeurs peuvent configurer les notifications du serveur d'intégration et ainsi être prévenus sur le service de leur choix en cas d'anomalies (webhook, email, etc...), gagnant ainsi un temps précieux.
- L'intégration continue permet également aux Dev d'avoir un retour plus rapide sur leur développement. Il n'y a donc plus besoin d'attendre plusieurs semaines pour identifier les erreurs et les corriger.
- l'intégration continue favorise le travail en équipe. En intégrant les révisions de code quotidiennement, le risque d'erreur est réduit au minimum. Les conflits au sein de l'équipe se font plus rares et les Dev n'ont plus peur de "casser le code" à chaque déploiement.



# CHAMP D'ACTION GITHUB

- Github est une solution qui permet de répondre à des heuristiques comme :
  - Comment mon code sera enregistré ?
  - Comment je mets mon code à disposition des autres collaborateurs ?
  - Comment je mets à jour mon code sans perdre les anciennes versions ?
  - Comment ne pas perdre mon code même si je perds mon ordinateur ?
  - Quel collaborateur a effectué tel changement ?
  - Puis je revenir à la version précédente ?



# JARGON GIT

- Directory : Folder (Dossier)
- Terminal ou Command Line : Interface de commandes
- CLI : Command Line Interface
- cd : Change directory (change de dossier)
- Code Editor : Programme d'édition de texte conçu spécifiquement pour l'édition du code source des programmes informatiques.
- Repository : Projet, ou le dossier où est stocké le projet.
- GitHub : Un site pour héberger vos repositories en ligne.



# REPO GITHUB

- Repository (aka “repo”): endroit où est conservée une copie de tous les fichiers.
  - Vous ne modifiez pas les fichiers directement dans le repo ;
  - Vous éditez une copie de travail locale ou "arbre de travail".
  - Puis vous déposez vos fichiers édités dans le repo
- En général, chaque utilisateur a sa propre copie du repo.
- Les fichiers de votre répertoire de travail doivent être ajoutés au repo afin d'être tracés (traçabilité).





# QUE CONTIENT UN REPO GITHUB?

- Tout ce qu'il faut pour créer votre projet :
  - Code source (Exemples : .java, .c, .h, .cpp )
  - Fichiers de compilation (Makefile, build.xml)
  - Autres ressources nécessaires à la construction de votre projet : versions requirements, icônes, texte, etc.
- Des choses qui ne sont généralement PAS mis en repo (elles peuvent être facilement recréées et ne prennent que de la place) :
  - Fichiers objets (.o)
  - Les exécutables (.exe)



# RÈGLES (STANDARDS) DE GITHUB

- Sur Git vous devez suivre des règles, ces règles peuvent être définies par votre organisation / groupe / laboratoire ...
- Mais il y a des règles sur lesquels tous les utilisateurs se sont mis d'accord et qui sont aujourd'hui entrain de devenir des standards si l'on veut que notre Git soit ISO.



# RÈGLES (STANDARDS) DE GITHUB

- Règle n° 1 : Créer un dépôt Git pour chaque nouveau projet.
- Règle n°2 : créer une nouvelle branche pour chaque nouvelle caractéristique.
- Règle n°3 : utiliser les pull requests pour fusionner le code avec le code maître.



# GIT x GITHUB

## Quelle Est La Différence Entre Git Et GitHub?

- Réponse courte : GitHub est un site web qui vous permet de mettre en ligne vos repos Git.



# GIT

- **Git** est un outil de gestion de version ou VCS (version control system) qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.
- Il fait parti de la famille des VCS dit décentralisés car dans son fonctionnement chaque développeur va avoir en local une copie complète de l'historique de son code source (repository).





# GIT

- Git est actuellement le gestionnaire de version le plus utilisé à travers le monde avec plus de douze millions d'utilisateurs.
- Git est également un incontournable des équipes de développement de la majorité des entreprises privées et des équipes Opensource.

👉 Donc oui, tout développeur se doit de connaître et maîtriser les bases de Git.



# GIT

- Une des grandes forces de Git, c'est qu'il est multi-plateforme (Windows, Linux, Mac) et possède deux modes de fonctionnements:
  - Terminal: Git peut être utilisé en ligne de commande dans un terminal. Par exemple la commande "git version" permet d'afficher le numéro de version de l'outil.
  - Interface graphique: Git peut également être utilisé via des interfaces graphique plus conviviales que le terminal.
- Alternatives Git : CVS, SVN, Perforce, Mercurial, Bazaar.



# GITHUB

- **Github** est un service en ligne qui permet entre autre d'héberger des dépôts Git.
- Il est totalement gratuit pour des projets ouverts au public mais il propose également des formules payantes pour les projets que l'on souhaite rendre privés.
- Github propose également de nombreux autres services très intéressants comme par exemple:
  - Partager du code source avec d'autres développeurs.
  - Signaler et gérer les problèmes ou bugs de votre code source via les issues.
  - Partager des portions de code via les Gists
  - Proposer des évolutions pour un projet opensource.



# GITHUB

- Github comptait plus de 10 millions de projets en 2013.
- Son succès a attiré de très nombreuses entreprises comme par exemple Google ou encore Microsoft.
- Cette dernière à d'ailleurs racheté la plateforme début 2018 pour la modique somme de 7,5 milliards de dollars, ce qui laisse présager encore une longue vie pour Git et Github.



# GITHUB

## Alternatives :

- Il existe d'autres plateformes hébergement et de partage de code source mais Github reste de loin le numéro 1 pour le moment.
- Dans les alternatives, on peut citer par exemple Bitbucket qui contrairement à GitHub, permet d'avoir des dépôts privés gratuitement mais limité au niveau de la taille de équipe qui peut y accéder.





# DIFFÉRENTS REPOS GIT



GITLAB



GITHUB



BITBUCKET

Quick Action Buttons



Supports adding images



Supports adding other type of attachments



Support for multiple markup languages



Possibility to view the history on code level



# INSTALLATION GIT



# GIT (LOCAL) WALK AROUND, INSTALLATION :

- Linux (Debian)

```
$ sudo apt-get install git
```

- Linux (Fedora)

```
$ sudo yum install git
```

- Mac

<http://git-scm.com/download/mac>

- Windows

<https://git-scm.com/download/windows>





Git is a [free and open source](#) distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is [easy to learn](#) and has a [tiny footprint with lightning fast performance](#). It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like [cheap local branching](#), convenient [staging areas](#), and [multiple workflows](#).



## About

The advantages of Git compared to other source control systems.



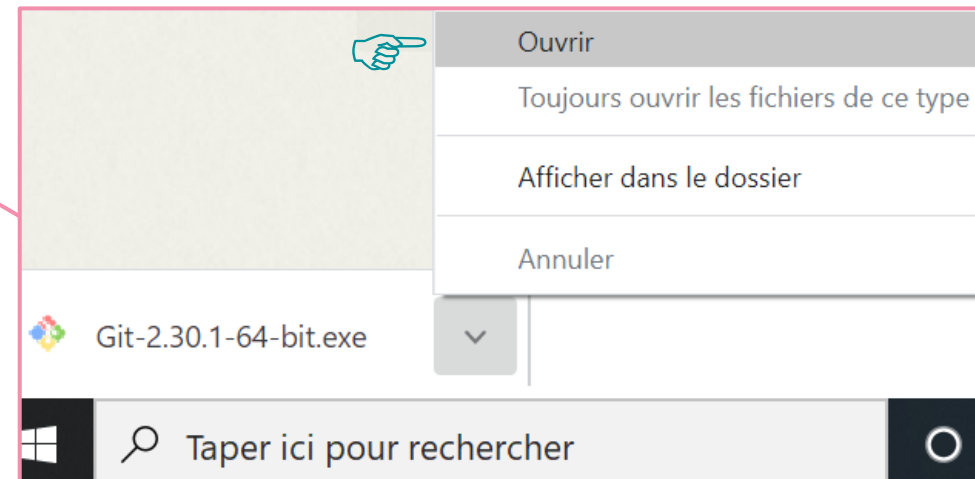
## Documentation

Command reference pages, Pro Git book content, videos and other material.

Latest source Release

**2.30.1**

[Release Notes](#) (2021-02-08)





### Information

Please read the following important information before continuing.



When you are ready to continue with Setup, click Next.

## GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.  
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

### Preamble

The licenses for most software are designed to take away your  
freedom to share and change it. By contrast, the GNU General Public  
License is intended to guarantee your freedom to share and change

<https://gitforwindows.org/>

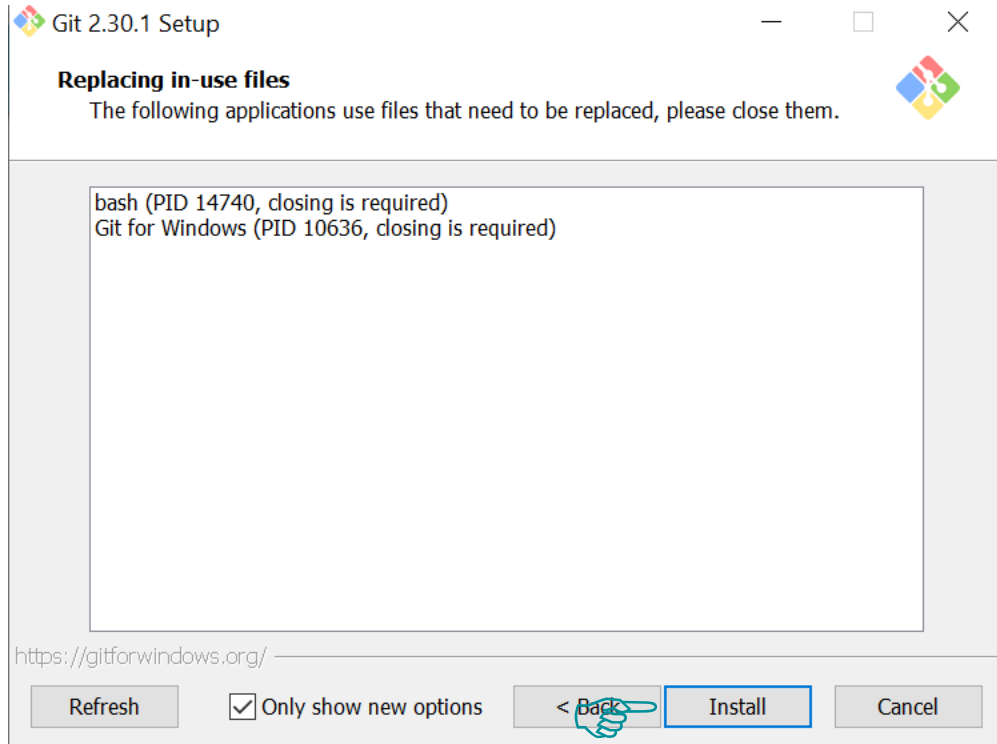
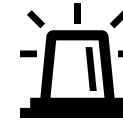
☒ Only show new options



Next >

Cancel





 Git 2.30.1 Setup

### Replacing in-use files

The following applications use files that need to be replaced, please close them.



bash (PID 14740, closing is required)  
Git for Windows (PID 10636, closing is required)

<https://gitforwindows.org/>

Refresh

☒ Only show new options

< Back

Install

Cancel

Vous continuez en appuyant sur suivant à chaque fois pour garder les paramètres par défaut.

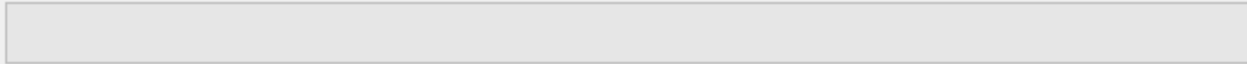
Il y a des applications qui doivent être fermées au moment de l'installation de Git, ceci n'est pas systématique ni par défaut, si comme moi vous avez des app marquées avec PID XXXXX, vous allez devoir les fermer avant de continuer.

 Git 2.30.1 Setup



## Installing

Please wait while Setup installs Git on your computer.



<https://gitforwindows.org/>

☒ Only show new options

Cancel

## Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.



☐ Launch Git Bash

☐ View Release Notes

☒ Only show new options

Finish

MINGW64:/c/Users/ahamid

ahamid@

MINGW64 ~

\$ |

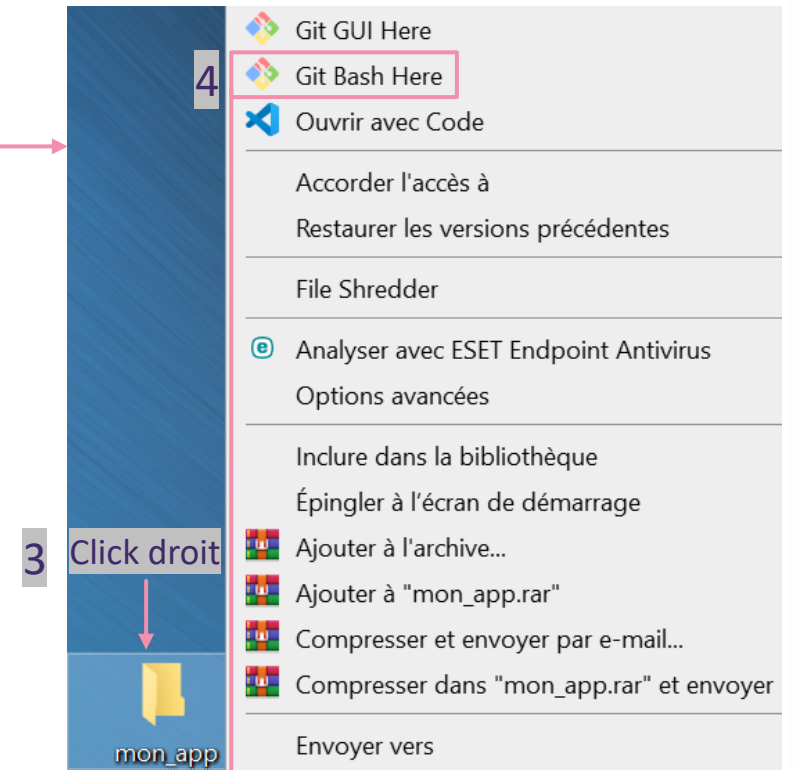
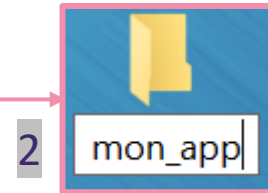
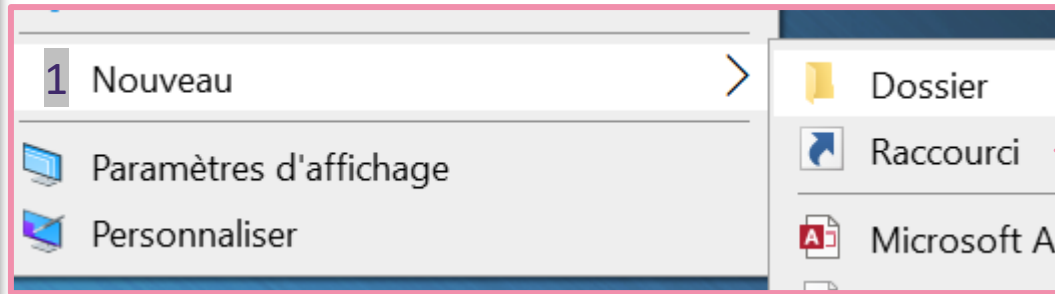
MINGW64:/c/Users/ahamid

ahamid@

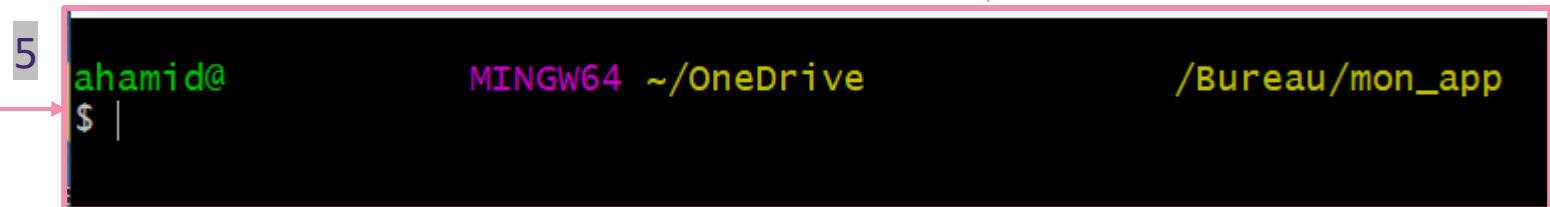
MINGW64 ~

\$ git --version

git version 2.30.1.windows.1



Pour agrandir, garder la touche ctrl appuyée et tourner la molette de la souris.





Installer notepad++ :

<https://notepad-plus-plus.org/downloads/>



current Version 7.9.5

🔖 Home

🔖 **Download**

## Downloads

- 🔖 Notepad++ 7.9.5 release

---

🔖 Notepad++ 7.9.4 release

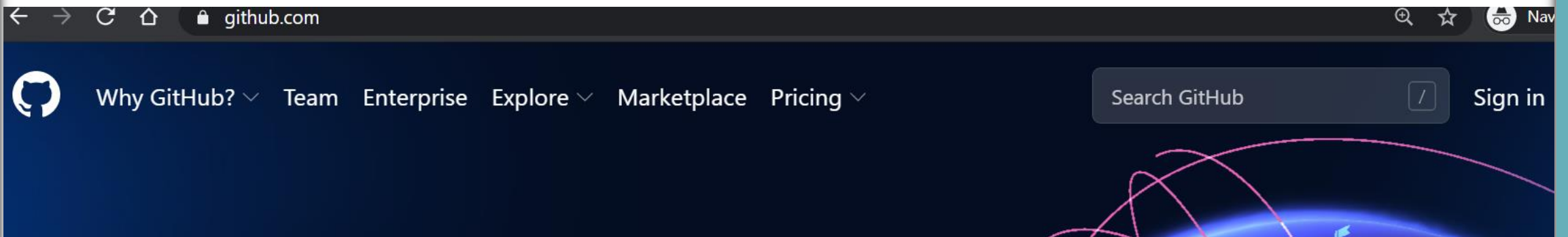
---

🔖 Notepad++ 7.9.3 release

---

Visiter GitHub voir des repos : php, python, html, java ...

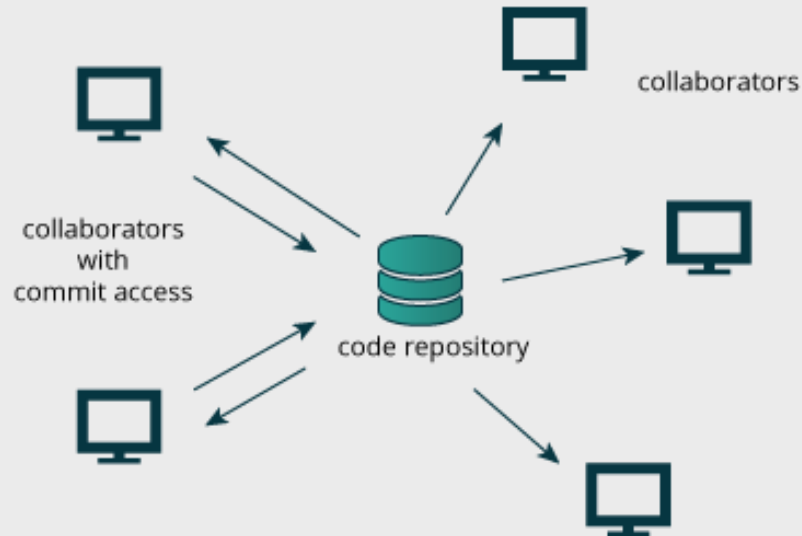
<https://github.com/>



# ARCHITECTURE SIMPLE GITHUB

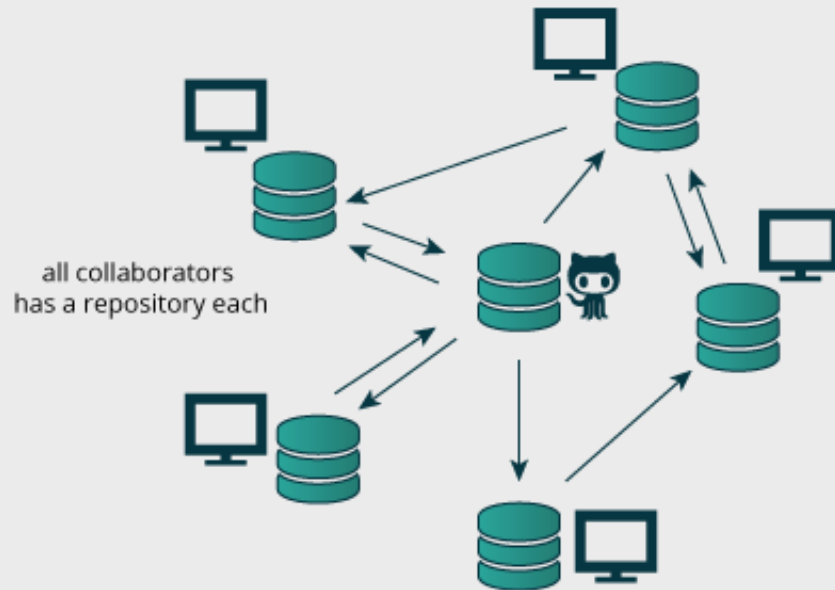


# ARCHITECTURE SIMPLE GITHUB



- Supposons que vous ayez développé 3 lignes de code et que votre ami en ai développé 2,
- Au lieu que le merge (**concaténation**) des deux bouts de code se fasse manuellement (mail, message...), vos 3 lignes de code sont sur le serveur central (**code repo**), un collaborateur peut récupérer les 3 lignes en utilisant la fonction **pull**, ensuite quand il aura modifié, il pourra (s'il est autorisé) faire un envoi de la MAJ, il fait donc un **push**.
- La nouvelle version sera donc centralisée sur le serveur central et mutualisé pour tous les collaborateurs. Cette opération est complètement transparente, vous recevrez une notification de MAJ, donc vous ferez un **pull** pour la récupérer.

# ARCHITECTURE DE SYSTÈME DISTRIBUÉ



- Dans un système distribué, vous obtenez votre propre dépôt lorsque vous clonez le projet.
- Cela signifie que vous pouvez travailler et ajouter du code au dépôt même lorsque vous êtes hors ligne, puisque le dépôt vit sur votre ordinateur.
- Mais cela signifie également que vous devez faire preuve d'un peu de discipline pour rester en phase avec le reste du développement, puisque votre dépôt est séparé des autres.

# WORKFLOW GÉNÉRAL GITHUB

- **fork** un projet sur github.
- **clone** le fork du github à votre machine.
- **create a topic branch** pour votre avancement dans votre clone local.
- **commit** les modifications apportées à votre dépôt local.
- **push** les modifications apportées à votre fork github.
- Envoyer un **pull request** de retrait au projet initial.





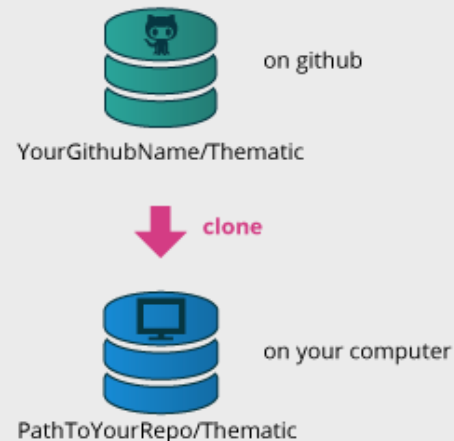
# WORKFLOW GÉNÉRAL GITHUB

## FORK ET CLONE

- Tout d'abord, nous devons 'fork'. Dans github, il vous suffit de cliquer sur ce bouton 'fork' et vous obtenez votre propre dépôt personnel de thèmes copié sur votre compte d'utilisateur github.



- Lorsque vous voulez vraiment travailler sur votre dépôt, vous devez le cloner sur votre ordinateur, et vous possédez maintenant deux dépôts : un distant sur github et un local sur votre ordinateur.



# WORKFLOW GÉNÉRAL GITHUB

## CREATE A TOPIC BRANCH

- Les dépôts Git sont organisés avec des tags et des branches.
- Une branche est un moyen de garder les lignes de développement séparées.
- La branche par défaut dans git est généralement nommée master.
- Les branches créées à partir de master sont communément appelées branches thématiques.
- Un exemple de cas d'utilisation des branches est lorsque vous travaillez sur un site web et que vous avez une idée mais que le fait de travailler dessus pourrait casser quelque chose. Créez une branche pour votre idée, engagez-y votre travail et faites vos tests, et lorsque tout fonctionne comme prévu, vous fusionnez la branche en master.



# WORKFLOW GÉNÉRAL GITHUB

## CREATE A TOPIC BRANCH

- Un autre cas d'utilisation pourrait être que vous avez deux idées que vous voulez comparer, comme par exemple deux schémas de couleurs pour votre site web. Faites une branche pour chaque idée et vous pourrez facilement passer de l'une à l'autre pour les comparer.
- Lorsque vous créez une branche, elle n'existera que sur votre repo local, à moins ou jusqu'à ce que vous décidiez de la partager avec d'autres. Et d'autres dépôts peuvent avoir plusieurs branches que vous choisirez de copier et de suivre ou que vous décidez de laisser tranquilles. Vous pouvez avoir autant de branches que vous le souhaitez.



# WORKFLOW GÉNÉRAL GITHUB

## CREATE A TOPIC BRANCH

- Une chose courante pour le développement de logiciels est de créer une branche de sujet pour chaque ticket de bug sur lequel on travaille. Lorsqu'un bug a été résolu, cette branche est fusionnée dans master. De cette façon, on peut travailler sur plusieurs bugs en parallèle sans interférer les uns avec les autres.
- Les grands projets peuvent avoir des flux de travail spécifiques que les développeurs sont censés suivre.



# WORKFLOW GÉNÉRAL GITHUB

## COMMIT ET CHECKOUT DES FICHIERS

- Lorsque vous avez effectué des modifications, vous les transférez dans votre dépôt. Les validations sont des morceaux logiques de modifications qui sont sauvegardés dans l'ordre, formant un historique que vous pouvez parcourir plus tard.
- Les modifications apportées à plusieurs fichiers peuvent aller dans le même commit, c'est comme si vous enregistriez l'état de tous les fichiers suivis dans votre répertoire en une seule fois. Chaque commit reçoit un timestamp, un nom d'auteur, un message de commit et un hachage généré automatiquement.
- Le hachage est la façon dont git garde la trace de ses commits et les référence. Un hachage typique ressemble à 27b6b79fca466af4648f9f8042e1f159b392293d. Github vous permet de parcourir l'historique des commit.



# WORKFLOW GÉNÉRAL GITHUB

## COMMIT ET CHECKOUT DES FICHIERS

- Si vous voulez voir comment les fichiers de votre projet se étaient à un moment donné, vous pouvez les consulter. Vous pouvez utiliser les hash-tags d'un commit comme référence, mais le plus courant est d'utiliser les noms ou les tags des branches.
- C'est ainsi que vous pouvez passer d'une branche à l'autre, en faisant un checkout d'une branche et un checkout d'une 'autre.





# WORKFLOW GÉNÉRAL GITHUB

## PULL = FETCH ET MERGE

- Ok, maintenant les fichiers sur votre ordinateur sont dans un dépôt qui leur est propre, localement sur votre ordinateur. Alors comment vous synchronisez-vous avec les autres dépôts ?
- Cela se fait en **fetch and merge**. Vous récupérez les informations en amont et vous fusionnez ces modifications dans votre propre dépôt. Fetch se connectera uniquement au dépôt distant et téléchargera les dernières modifications dans votre historique, Merge mettra en fait ces modifications dans une de vos branches et, les fusionnera avec vos propres modifications. L'historique n'est pas spécifique à une seule branche, il garde la trace de toutes les branches en même temps.



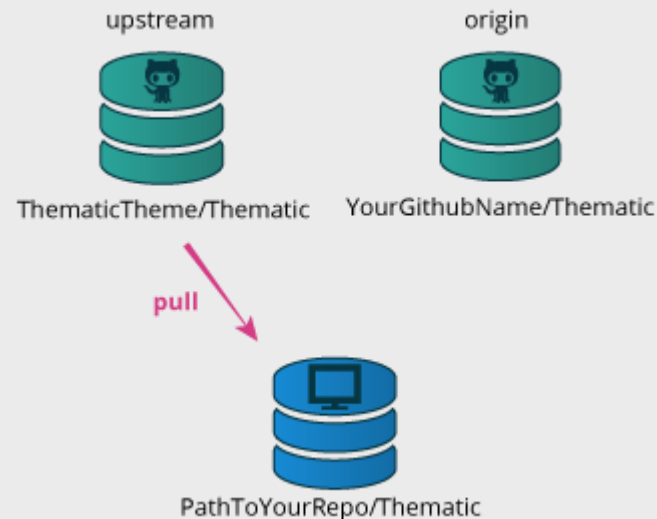
# WORKFLOW GÉNÉRAL GITHUB MERGE

- Merge : signifie simplement que Git va essayer de combiner les changements de deux endroits, en plaçant les commits à la suite les uns des autres.
- Si deux commit tentent de modifier le même morceau de code, vous obtenez ce que l'on appelle un conflit de fusion et vous devrez examiner manuellement les fichiers et déterminer la version que vous souhaitez conserver. Mais la plupart du temps, il n'y a pas de conflit et la fusion se fait sans problème.



# WORKFLOW GÉNÉRAL GITHUB PULL

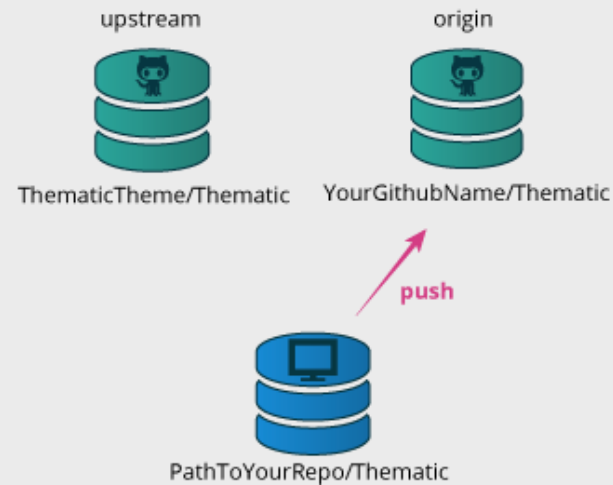
- Il peut être fastidieux de toujours aller chercher les changements en premier et de les fusionner ensuite, surtout lorsque vous êtes certain de vouloir mettre les engagements dans votre branche. Heureusement, il existe une commande qui fera les deux choses à la fois, à savoir pull.
- Vous pouvez effectuer une extraction à partir de n'importe quel dépôt, vous n'avez pas besoin d'autorisations spéciales pour effectuer une extraction. En général, vous devez spécifier quel dépôt et quelle branche vous souhaitez extraire, et l'extraction se fera sur la branche qui est actuellement extraite dans votre répertoire. Vous pouvez mettre en place un suivi automatique des branches, mais c'est un peu hors de portée pour cette recherche.



# WORKFLOW GÉNÉRAL GITHUB

## PUSH, I.E PULL REQUEST

- Lorsque vous faites un commit dans votre dév vous souhaitez partager votre code avec d'autres, vous pouvez le push vers des dépôts auxquels vous avez accès. Votre propre pôt local, les commits n'existent que sur votre machine. Si github fork appelé origin en est bien sûr un. Poussez vers l'origine et vos commits seront maintenant sur github !



- Si vous voulez contribuer à des endroits où vous n'avez pas d'accès en mode "push", comme le thème "repo", vous pouvez leur envoyer une demande de "pull" sur github. Vous ne pouvez pas leur imposer de modifications, mais ils peuvent vous en demander.



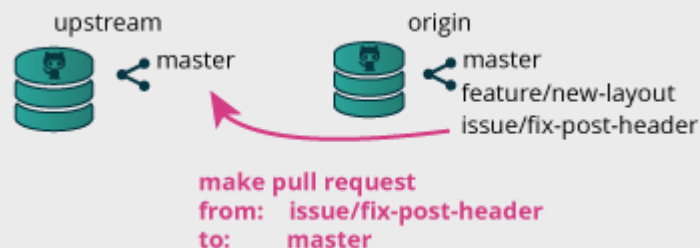
# WORKFLOW GÉNÉRAL GITHUB

## PUSH, I.E PULL REQUEST

- Si vous voulez contribuer à des endroits où vous n'avez pas d'accès en mode "push", vous pouvez leur envoyer une demande de "pull" sur github.



- Là encore, l'approche par branche thématique présente un avantage. Github vous permet de spécifier dans laquelle de vos branches vous voulez que le pull cible (la tête), ainsi que la branche dans laquelle vous voulez que le pull fusionne (la base). Mais il ne peut y avoir qu'une seule demande active par branche. Si vous voulez contribuer à plusieurs choses, vous devez avoir une branche séparée par contribution.



# WORKFLOW GÉNÉRAL GITHUB

## PUSH, I.E PULL REQUEST

- Une fois que votre demande pull est acceptée et fusionnée par le responsable du projet, vous pouvez alors supprimer la branche thématique, à la fois sur votre ordinateur et sur votre repo github ("origine").
- Vos commit feront désormais partie du projet principal et la prochaine fois que vous ferez un pull, ils seront dans votre branche master.





# WORKFLOW GÉNÉRAL GITHUB

## PUSH, I.E PULL REQUEST

- Une fois que votre demande pull est acceptée et fusionnée par le responsable du projet, vous pouvez alors supprimer la branche thématique, à la fois sur votre ordinateur et sur votre repo github ("origine").
- Vos commit feront désormais partie du projet principal et la prochaine fois que vous ferez un pull, ils seront dans votre branche master.



# WORKFLOW GÉNÉRAL GITHUB

## GITHUB README

- Quand vous réalisez un projet, vous voulez que n'importe quelle personne autorisée à travailler dessus puisse reproduire les mêmes résultats sans devoir à vous appeler, ni d'attendre que vous reveniez de votre congé.
- Comme quand vous allez chez IKEA, on vous mets dans le carton des notices de montage pour vos meubles.
- Le ReadMe doit contenir:
  - **La description** : Qu'est ce que c'est tout ça déjà ?
  - **Les spécificités** : Pourquoi ça a été fait comme ça ?  
Quelle version devrais-je installer pour que ça marche ?
  - **Exemples** : Comment faire marcher ça ?
  - **Références** : D'où viennent ces idées (Articles scientifiques, articles internes...)  
Qui contacter en cas de problème ?

<https://www.makeareadme.com/>



# RÉCAPITULATIF

- Le contrôle de version consiste à gérer plusieurs versions de programmes, sites web, etc.
- Vous donne une "machine à remonter le temps" pour revenir aux versions précédentes.
- Vous offre un support important pour les différentes versions (application web, machine learning, etc.) d'un même projet.
- Simplifie grandement le travail simultané, en fusionnant les changements (en équipe).
- Partager votre expérience avec un recruteur.
- Permet d'être plus efficace, de donner un meilleur flux de travail.

The background features a gradient from red at the top to blue at the bottom, overlaid with faint, white geometric patterns including concentric circles, arcs, and degree markings (e.g., 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260). A large, solid blue triangle with a thin red border is positioned on the left side, pointing towards the right.

- DEMO -



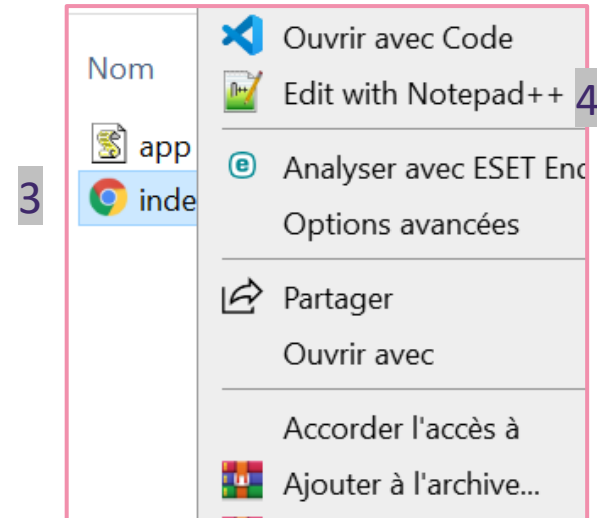
# COMMANDES GITHUB

command	description
<code>git clone <i>url</i> [<i>dir</i>]</code>	copy a git repository so you can add to it
<code>git add <i>files</i></code>	adds file contents to the staging area
<code>git commit</code>	records a snapshot of the staging area
<code>git status</code>	view the status of your files in the working directory and staging area
<code>git diff</code>	shows diff of what is staged and what is modified but unstaged
<code>git help [<i>command</i>]</code>	get help info about a particular command
<code>git pull</code>	fetch from a remote repo and try to merge into the current branch
<code>git push</code>	push your new branches and data to a remote repository
others: <code>init</code> , <code>reset</code> , <code>branch</code> , <code>checkout</code> , <code>merge</code> , <code>log</code> , <code>tag</code>	

```
1 ahamid@ MINGW64 ~/OneDrive - /Bureau/mon_app
$ touch index.html

2 ahamid@ MINGW64 ~/OneDrive - /Bureau/mon_app
$ touch app.js
```

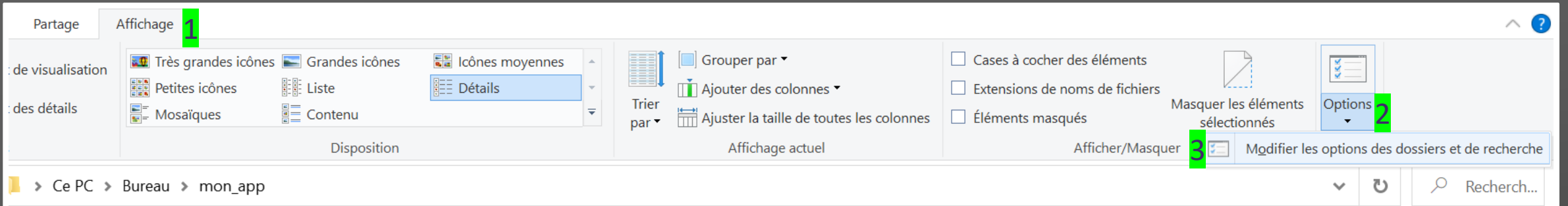
```
5 <html>
  <head>
    <title> Mon application demo </title>
  </head>
  <body>
    Ceci est mon application en cours de dev
  </body>
</html>
```



Ensuite nous allons vouloir initialiser notre dossier comme repo git de la manière suivante :

```
6 ahamid@ MINGW64 ~/OneDrive - /Bureau/mon_app
$ git init
Initialized empty Git repository in C:/Users/ahamid/OneDrive - /Bureau/mon_app/.git/
```





	Nom	Modifié le	Type	Taille
	app	06/03/2021 18:38	Fichier de JavaScript	0 Ko
	index	06/03/2021 19:04	Chrome HTML Docu...	1 Ko

#### Options des dossiers

Général Affichage Rechercher

##### Affichage des dossiers

Vous pouvez appliquer cet affichage (Détails ou Icônes, par exemple) à tous les dossiers du même type.

Appliquer aux dossiers

Réinitialiser les dossiers

##### Paramètres avancés :

###### Fichiers et dossiers

- ☒ Afficher l'icône des fichiers sur les miniatures
- ☒ Afficher la barre d'état
- ☒ Afficher la légende des dossiers et des éléments du Bureau
- ☐ Afficher le chemin d'accès complet dans la barre de titre
- ☐ Afficher les dossiers et les fichiers NTFS chiffrés ou compressés
- ☒ Afficher les gestionnaires d'aperçu dans le volet de visualisation
- ☒ Afficher les informations concernant la taille des fichiers dans la barre de titre
- ☒ Afficher les lettres de lecteur
- ☒ Afficher les notifications du fournisseur de synchronisation
- ☒ Fichiers et dossiers cachés
  - ☒ Afficher les fichiers, dossiers et lecteurs cachés

Paramètres par défaut

OK

Annuler

Appliquer

7

Nom

.git  
app  
index

On remarque qu'un git folder a été créé par le init  
Nous pouvons à présent utiliser les commandes Git.  
Avant de rien toucher il va vous falloir donner votre  
nom et adresse mail à Git.

8

```
ahamid@MINGW64 ~/OneDrive - /Bureau/mon_app (master)
```

```
$ git config --global user.name 'professeur git'
```

9

```
ahamid@MINGW64 ~/OneDrive - /Bureau/mon_app (master)
```

```
$ git config --global user.email 'professeurgit@me.com'
```

Maintenant il va falloir rajouter le fichier index.html à notre repo Git

```
1 $ git add index.html
```

Pour regarder ce qui se passe dans l'environnement de staging git :

```
2 $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app.js
```

Pour supprimer le 'index.html' on utilise 'rm' :

```
3 $ git rm --cached index.html
```

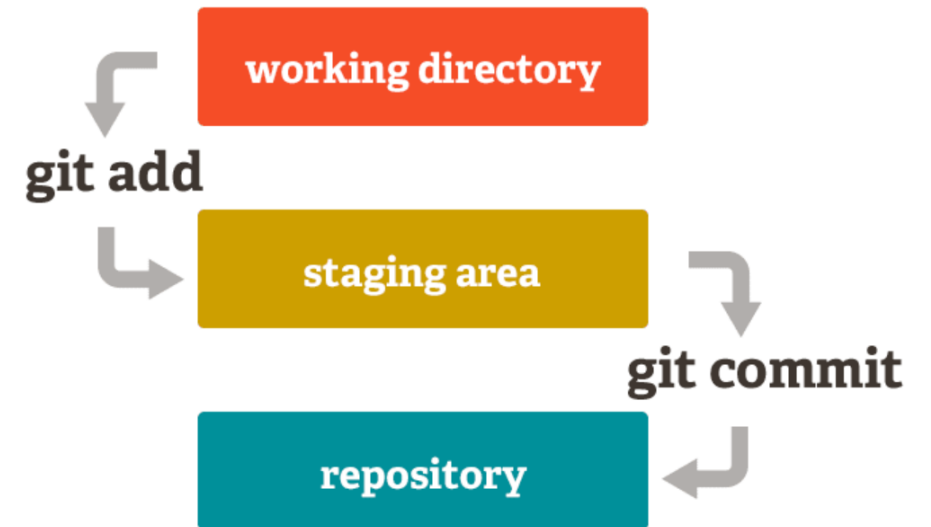
Que remarquez-vous si vous regardez le statut Git ?

Pour rajouter par exemple tous les fichiers html en une fois :

```
4 $ git add *.html
```

Comment rajouter tous les fichiers qu'on a dans le dossier ?

Ce que le résultat nous dit que le fichier 'index' a bien été rajouté à notre staging area et que app.js est 'Untracked'.



Exercice :

1. Ajouter le fichier index.html à git, ensuite repartez sur votre fichier index.html
2. Ouvrez le fichier avec votre éditeur de texte (notepad++)
3. Changer quelque chose dans le corps du message, par exemple : rajoutez un (!)
4. Enregistrer le fichier
5. Que voyez vous si vous vérifiez le statut Git
6. Que vous dis le message du status
7. `git add .`
8. `git status`

Question :

Comment faire un commit ?

1 `$ git commit`

2 Pour taper sur la fenêtre qui s'ouvre il faut appuyer sur la touche (i) pour passer au mode INSERT

3 Une fois dans l'insert mode, tapez ceci : Initial commit.  
Ceci fera référence à ce commit en particulier fais aujourd'hui.

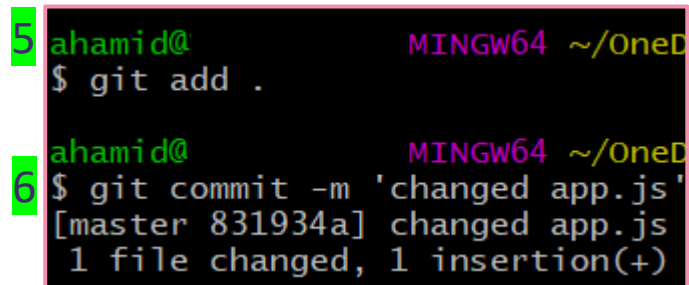
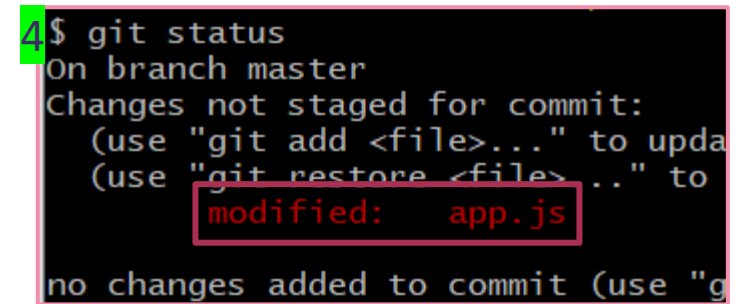
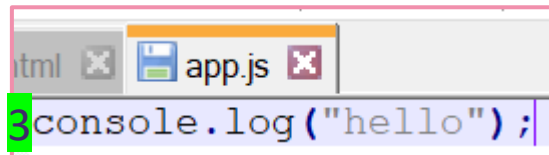
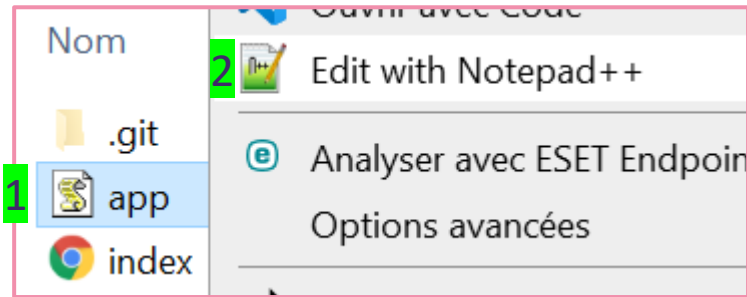
4 Pour sortir du mode INSERT, il faut appuyer sur la touche échap (esc).

5 Pour enregistrer et quitter, tapez (:wq)

Que représente le résultat affiché après l'enregistrement ?

Quelle information est rapportée si regarde le status ?

Sans lancer aucune commande, que se passera-t-il si on change notre index.html et qu'on fasse status ?



Ceci est une manière de faire le commit en rajoutant un message titre du commit sans passer par VIM.

On va créer un git ignore qui permet qu'un fichier spécifique ne soit jamais rajouté au commit même avec 'add.' :

7\$ touch .gitignore

8\$ touch indésirable.txt

Dans le fichier 'indésirable.txt', écrivez une chaîne de caractère, par exemple : erreur dans les logs.

Dans '.gitignore', écrivez le nom exact du 'indésirable.txt', sans espaces en plus.

9\$ git add .

10 Que ce passe t-il ? Regarder le status.

Il est également possible de rajouter des dossiers :

1. Sur votre dossier app, créer un nouveau dossier appelé '**dir1**' dans lequel vous allez créer un fichier '**file.js**' contenant **console.log(123);**
2. Sur votre dossier app, créer un nouveau dossier appelé '**dir2**' dans lequel vous allez créer un fichier '**file.js**' contenant **console.log(123);**
3. Sur votre .gitignore, rajoutez **/dir2** dans une nouvelle ligne.
4. Maintenant ajoutez tous le contenu du work directory à git
5. Que nous dis le status ?
6. `$ git commit -m 'another change'`

### Maintenant nous allons travailler sur la création de nouvelle branche :

```
$ git branch login
```

Si on regarde le status on remarque qu'on est toujours sur la branche master.

Pour aller à la branche qu'on vient de créer : `$ git checkout login`

1. Maintenant il faut faire un touch login.html, ensuite écrire 'login' dans le fichier html.

```
$ git add .
```

```
$ git commit -m 'login form'
```

```
$ git checkout master
```

Retourner sur le dossier mon\_app, rafraichir, que remarquez vous ?

Pourquoi une telle différence ?



Cela est arrivé par ce que les données login sont dans la branche login

Si l'on veut faire un merge (joindre) les données des deux :

1. On se positionne sur le master : `$ git checkout master`

2. Taper : `$ git commit`

Nous nous retrouvons avec le même éditeur de texte, taper (i), ensuite écrivez (**rajout de login**), tapez **échap**, tapez (:wq), appuyer sur **entr** pour enregistrer et quitter.

3. A présent si vous revenez au dossier, vous devriez être capable de voir le fichier login.html

**Félicitations vous connaissez maintenant toutes les étapes et fonctionnalités de base de GIT.**

**Des questions ?**