



# git

DÉCOUVRIR LE VERSIONNING AVEC GIT

# GITHUB



# PRÉSENTATION DU FORMATEUR

- Profession : Data Scientist – Statisticien
- Domaines d'expertise : Finance, E-commerce, Cyber Sécurité
- Formations : IT / Big Data / Data Science

The background of the slide features a subtle, abstract design. It consists of several concentric circles in light gray and white, some with dashed outlines. Small black arrows point clockwise around these circles. The numbers 0, 80, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, and 210 are scattered across the right side of the slide, likely representing a scale or data points.

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.

AUDIANCE

# 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 certaines 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 modifiés 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 ≈ 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](#)

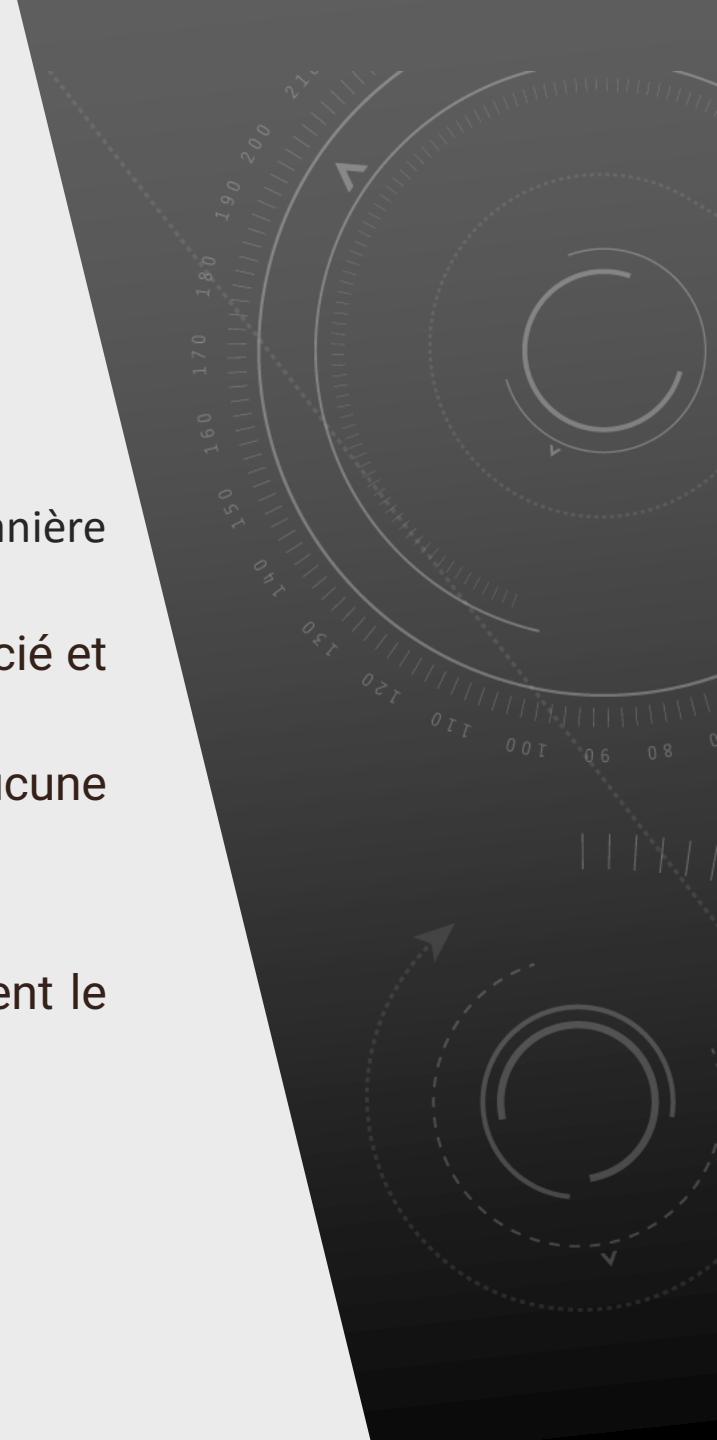


# QUELQUES FONCTIONNALITÉS

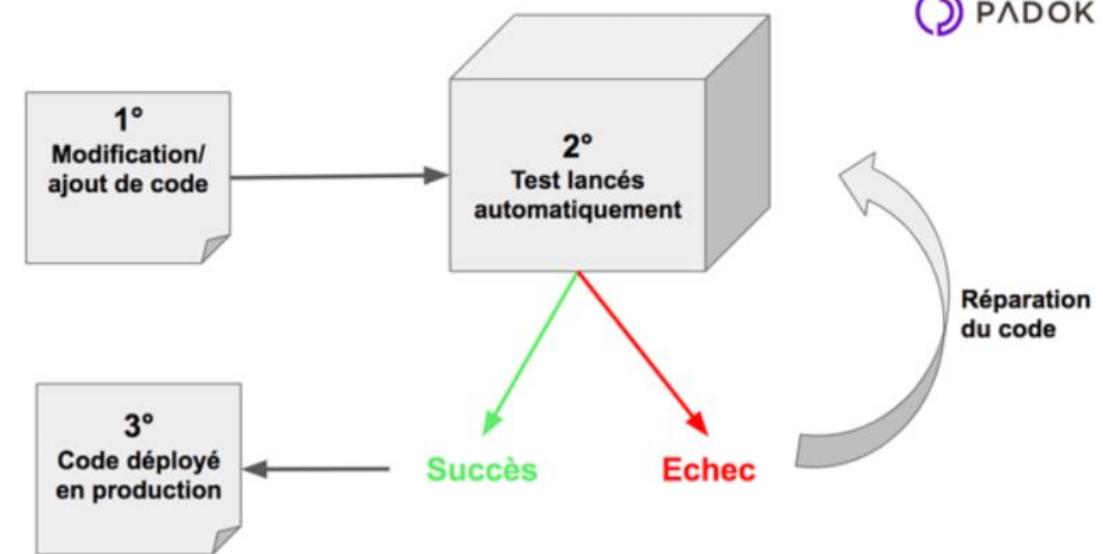
- Hébergement de projets.
- Partage de projets.
- Octroi d'accès en écriture aux projets.
- Gestion de versions
- Possibilité de suivre des personnes (des projets).
- Possibilité de commenter les projets.
- **L'intégration continue**

# 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.

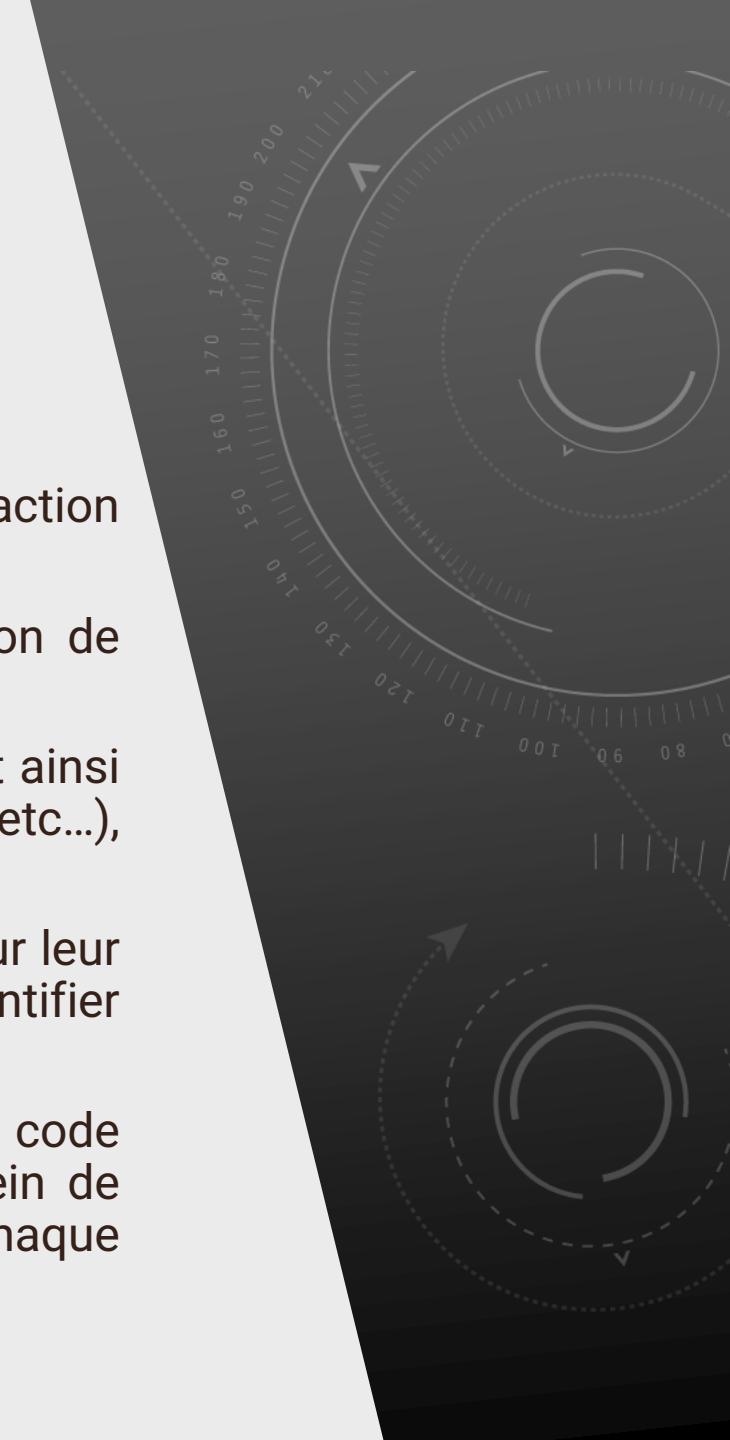


# INTÉGRATION CONTINUE



# 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 graphiques 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

	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>



Search entire site...

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](#).



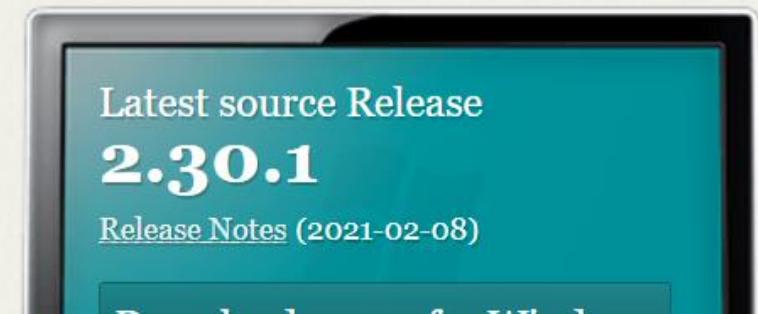
## About

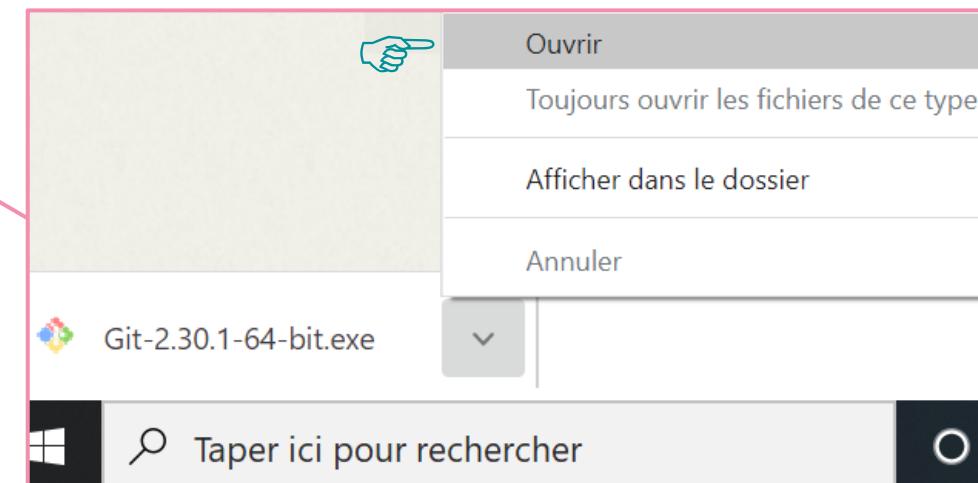
The advantages of Git compared to other source control systems.



## Documentation

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







## 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

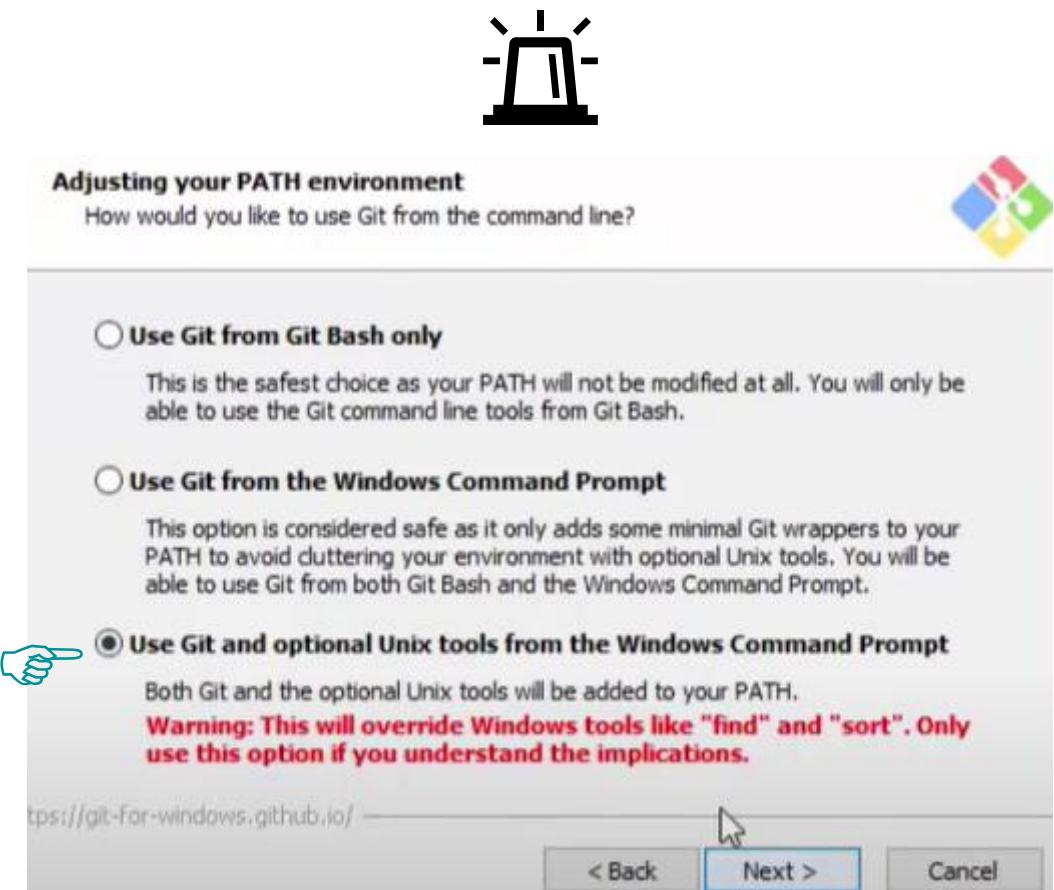
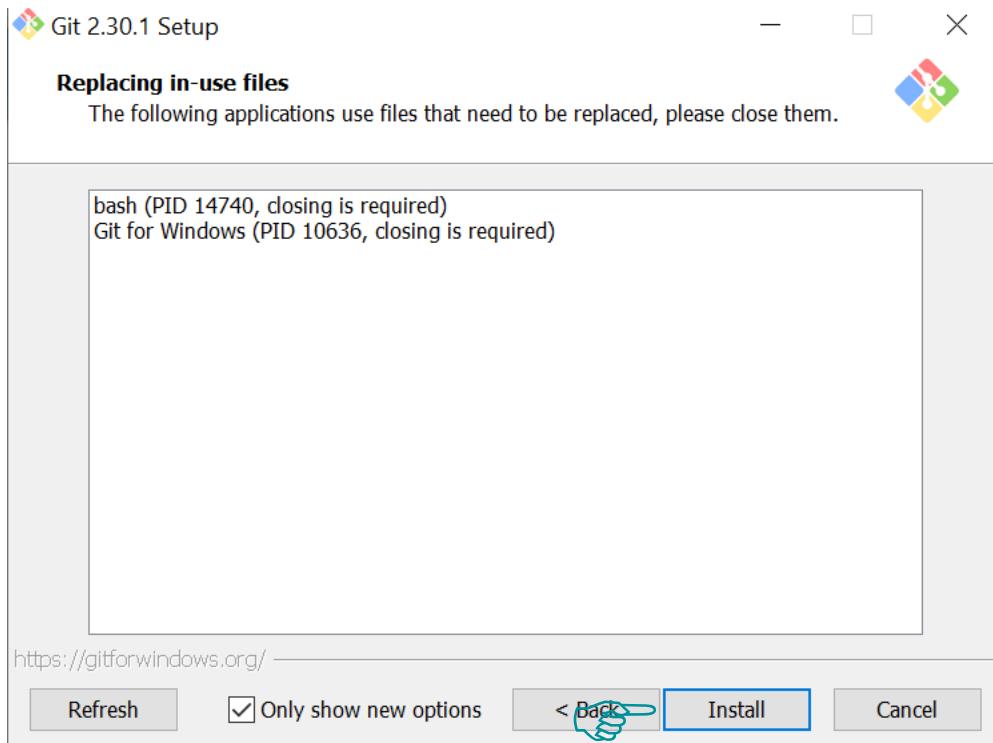
<https://gitforwindows.org/>

Only show new options



Next >

Cancel





## 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)



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.

<https://gitforwindows.org/>

Refresh

Only show new options

< Back

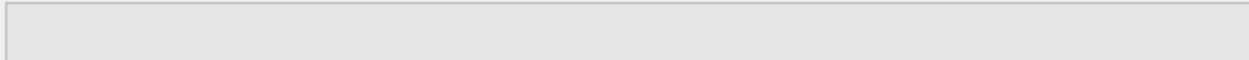
Install

Cancel



## 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

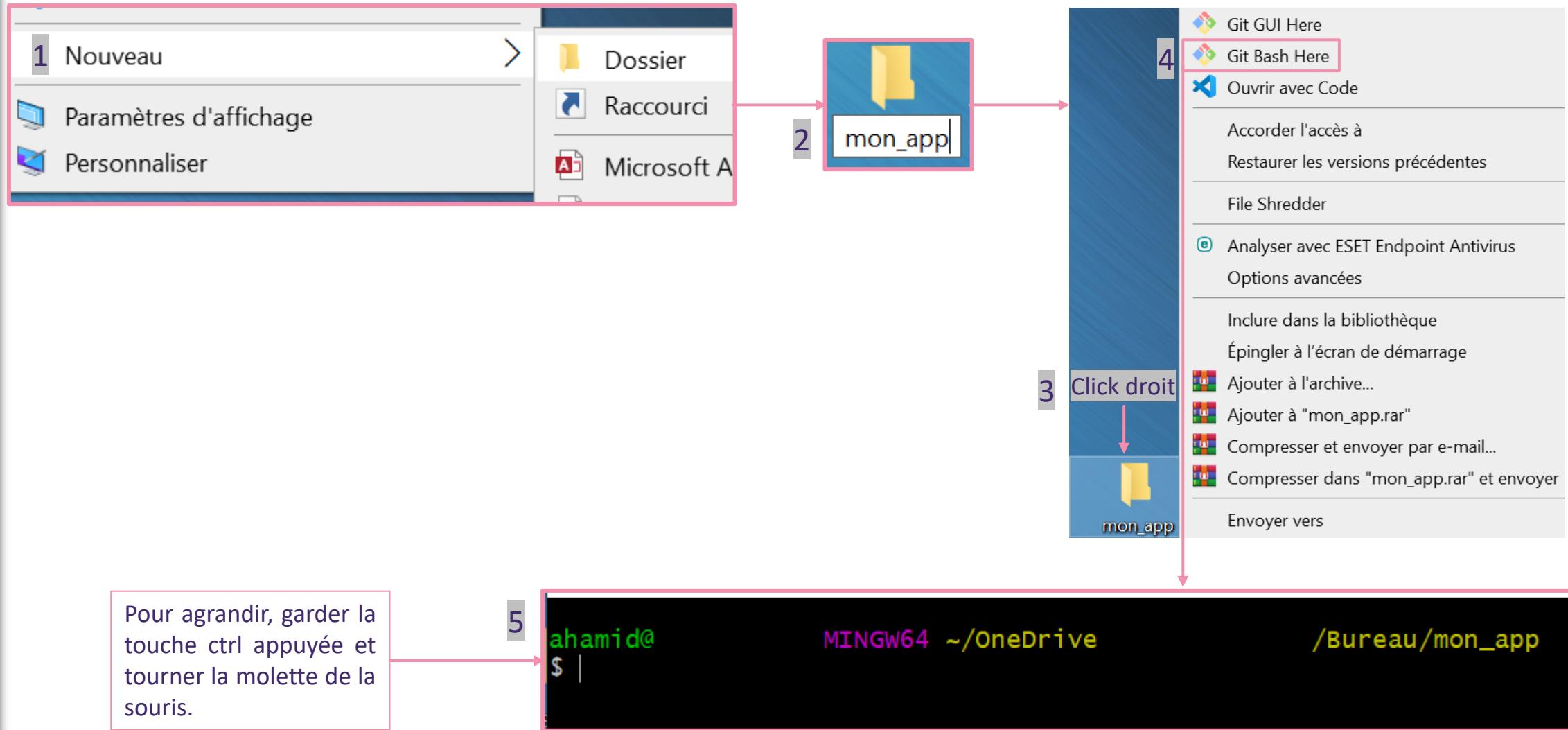
View Release Notes

Only show new options

Finish

MINGW64:/c/Users/ahamid  
ahamid@ ahmad ~ MINGW64 ~  
\$ | ↵

MINGW64:/c/Users/ahamid  
ahamid@ ahmad ~ MINGW64 ~  
\$ git --version  
git version 2.30.1.windows.1



Installer notepad++ :

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



## Downloads

- [Notepad++ 7.9.5 release](#)

---

[Notepad++ 7.9.4 release](#)

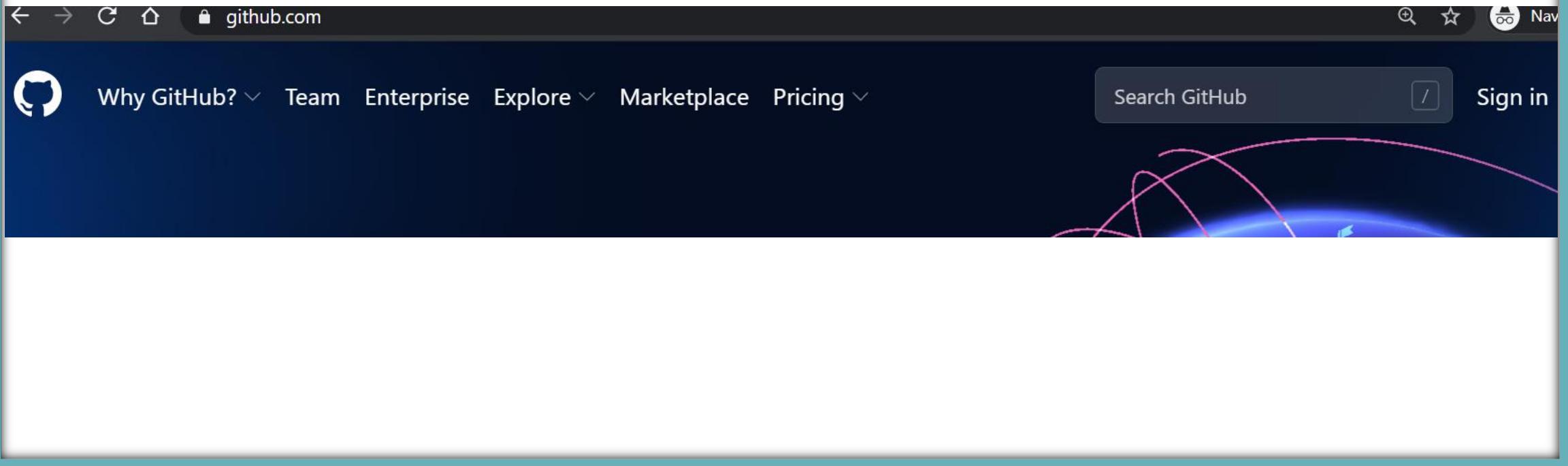
---

[Notepad++ 7.9.3 release](#)

---

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

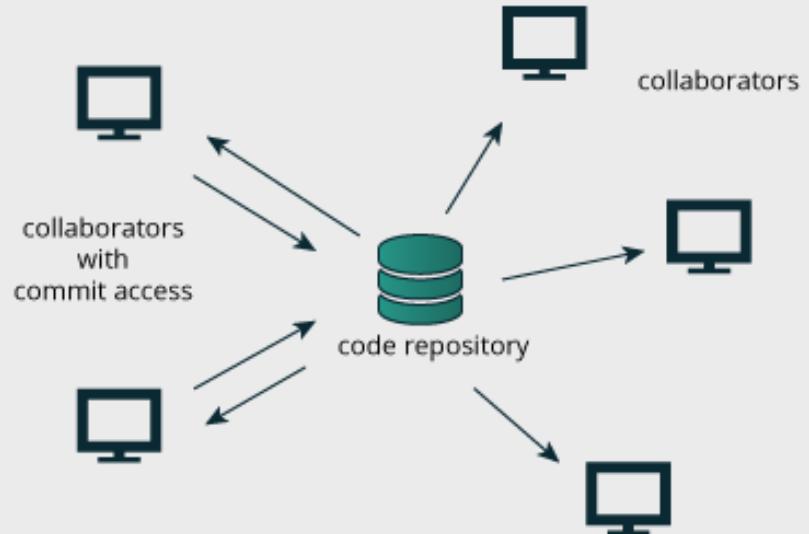
<https://github.com/>



# ARCHITECTURE SIMPLE GITHUB

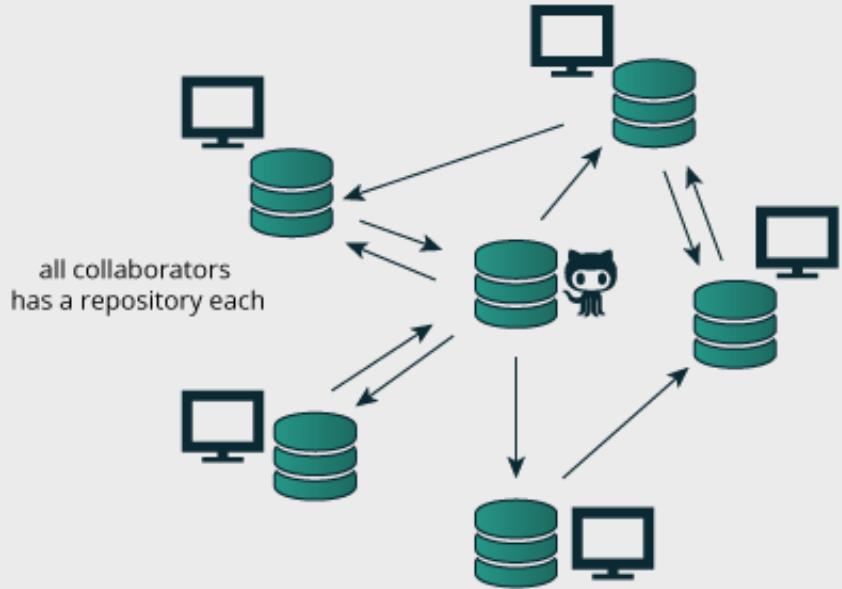


# ARCHITECTURE SIMPLE GITHUB



- 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.

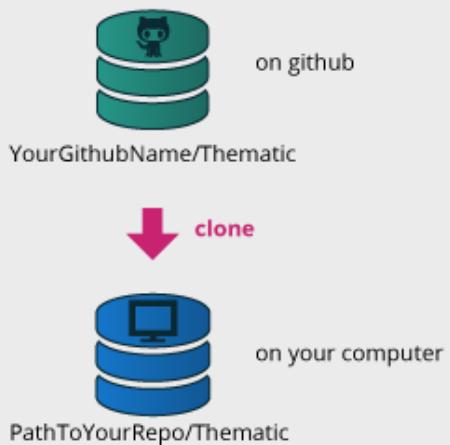


# WORKFLOW GÉNÉRAL GITHUB FORK ET CLONE

- 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 choisissez 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.



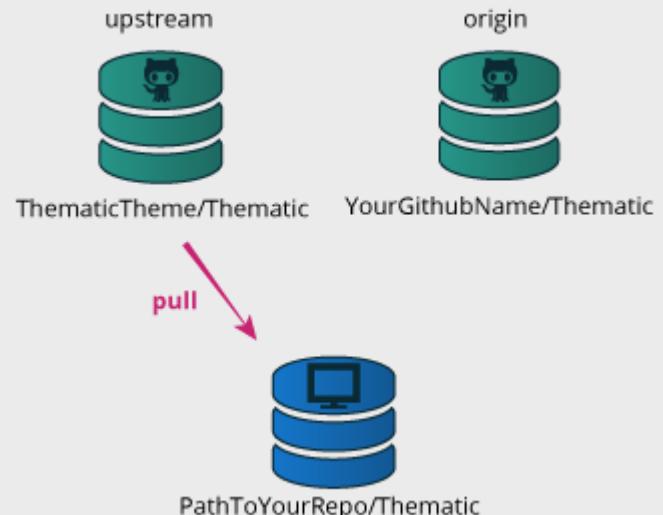
# WORKFLOW GÉNÉRAL GITHUB MERGE

- 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.



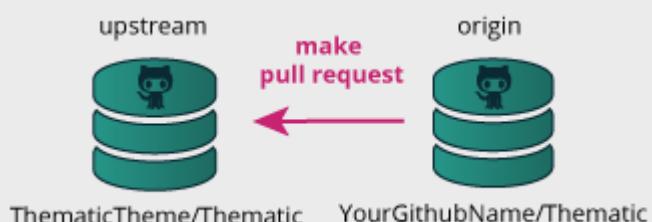
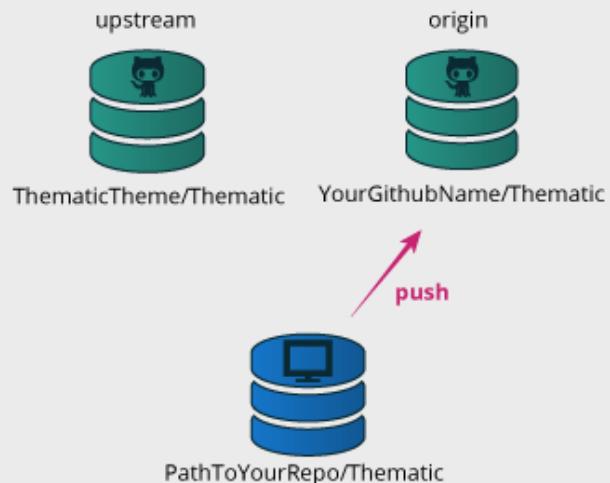
# WORKFLOW GÉNÉRAL GITHUB PULL

- 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.



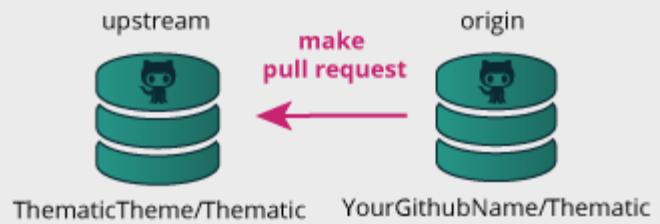
# WORKFLOW GÉNÉRAL GITHUB PUSH, I.E PULL REQUEST

- Lorsque vous faites un commit dans votre dévous 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.

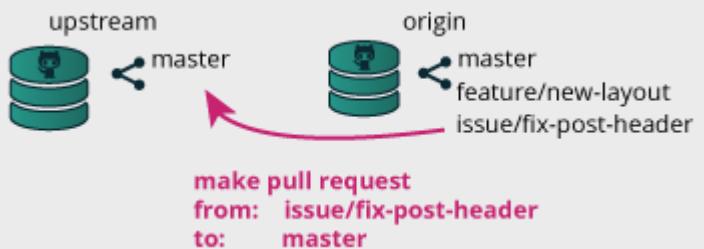


# WORKFLOW GÉNÉRAL GITHUB PUSH, I.E PULL REQUEST

- 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.

- DEMO -

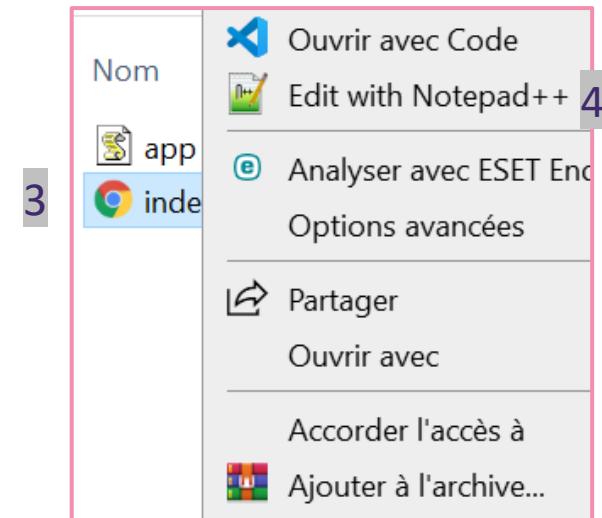


# COMMANDES GITHUB

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: init, reset, branch, checkout, merge, log, tag

```
1 ahamed@ MINGW64 ~/OneDrive - /Bureau/mon_app  
$ touch index.html  
  
2 ahamed@ 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 ahamed@ MINGW64 ~/OneDrive - /Bureau/mon_app  
$ git init  
Initialized empty Git repository in C:/Users/ahamed/OneDrive - /Bureau/mon_ap...  
p/.git/
```

Partage Affichage 1

: de visualisation : des détails

Disposition

Trier par ▾ Grouper par ▾ Ajouter des colonnes ▾

Ajuster la taille de toutes les colonnes

Cases à cocher des éléments  
Extensions de noms de fichiers  
Éléments masqués

Masquer les éléments sélectionnés

Options 2

Afficher/Masquer 3

Modifier les options des dossiers et de recherche

Rechercher... Rechercher

Ce PC > Bureau > mon\_app

Nom Modifié le Type Taille

app 06/03/2021 18:38 Fichier de JavaScript 0 Ko

index 06/03/2021 19:04 Chrome HTML Doc... 1 Ko

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.

ahamid@ MINGW64 ~/OneDrive - /Bureau/mon\_app (master)

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

ahamid@ MINGW64 ~/OneDrive - /Bureau/mon\_app (master)

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

Options des dossiers

Général 4 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 les dossiers

Afficher les lettres de lecteur

Afficher les notifications du fournisseur de synchronisation

Fichiers et dossiers cachés

Afficher les fichiers, dossiers et lecteurs cachés

5

6 OK Annuler Appliquer

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 supprime le 'index.html' on utilise 'rm' :

3 \$ git rm --cached index.html

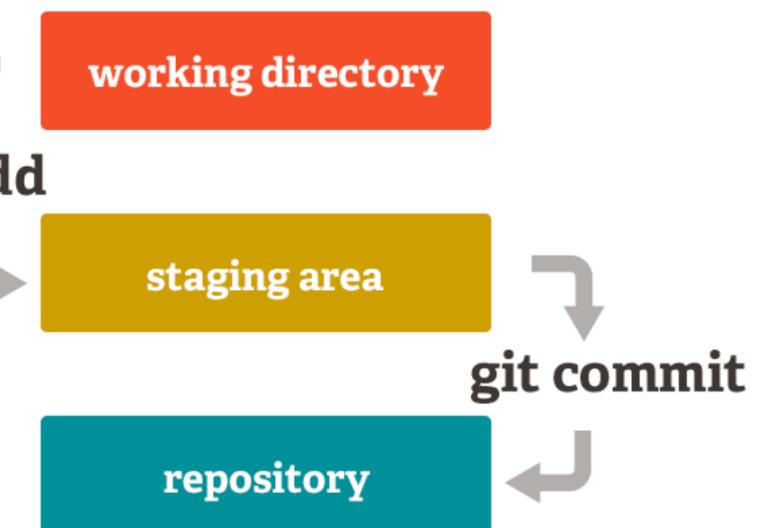
Que remarquez-vous si vous regardez le statut Git ?

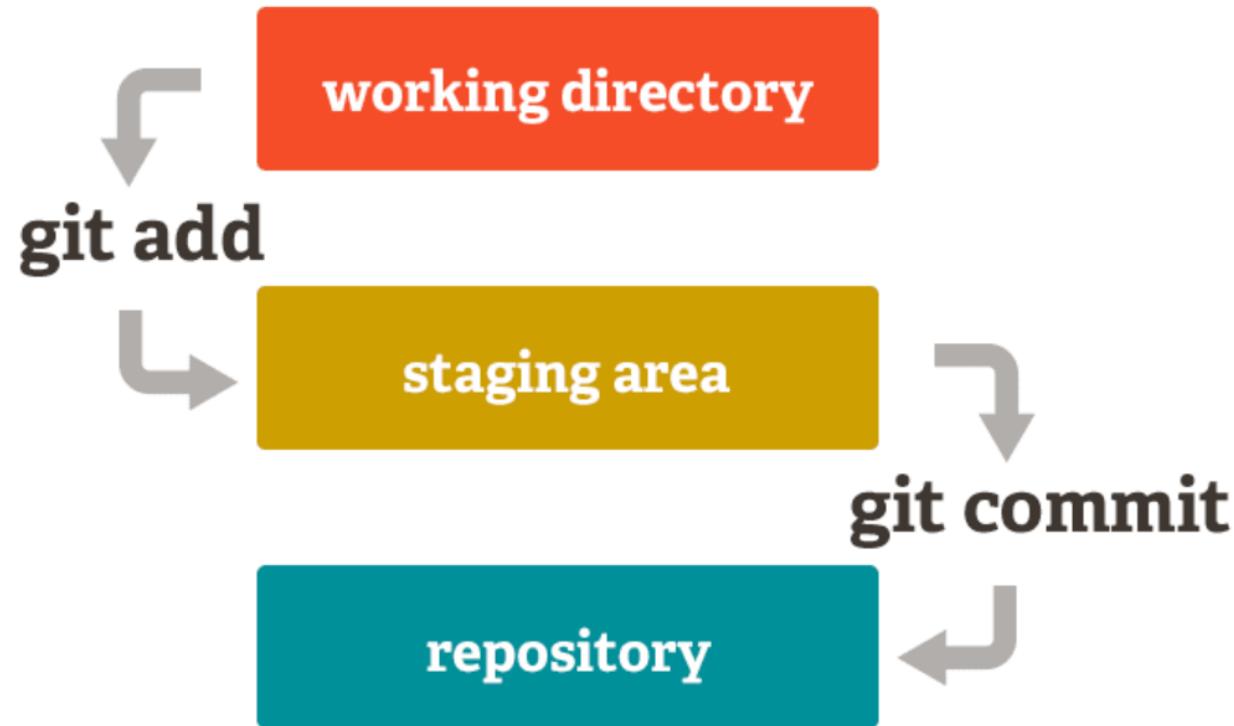
Pour rajouter par exemple tous les fichiers html en une fois :

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++ / bloc note / visual studio / sublime ...)
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.

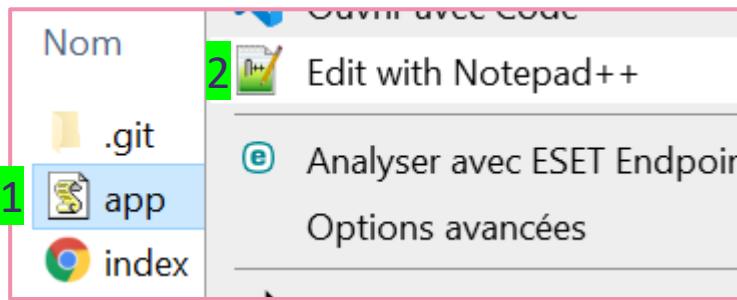
**4** Pour sortir du mode INSERT, il faut appuyer sur la touche échap (esc).

**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 ?



A screenshot of a code editor showing an 'app.js' file with the following content:

```
console.log("hello");
```

```
4$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update
   what will be committed)
    (use "git restore <file>..." to
     discard changes in working directory)
          modified:   app.js

no changes added to commit (use "g
```

```
5 ahamed@          MINGW64 ~/OneD
$ git add .

6 ahamed@          MINGW64 ~/OneD
$ git commit -m 'changed app.js'
[master 831934a] changed app.js
 1 file changed, 1 insertion(+)
```

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 indesirable.txt
```

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

Dans '.gitignore', écrivez le nom exact du 'indesirable.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**, rafraîchir, que remarquez vous ?

Pourquoi une telle différence ?

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

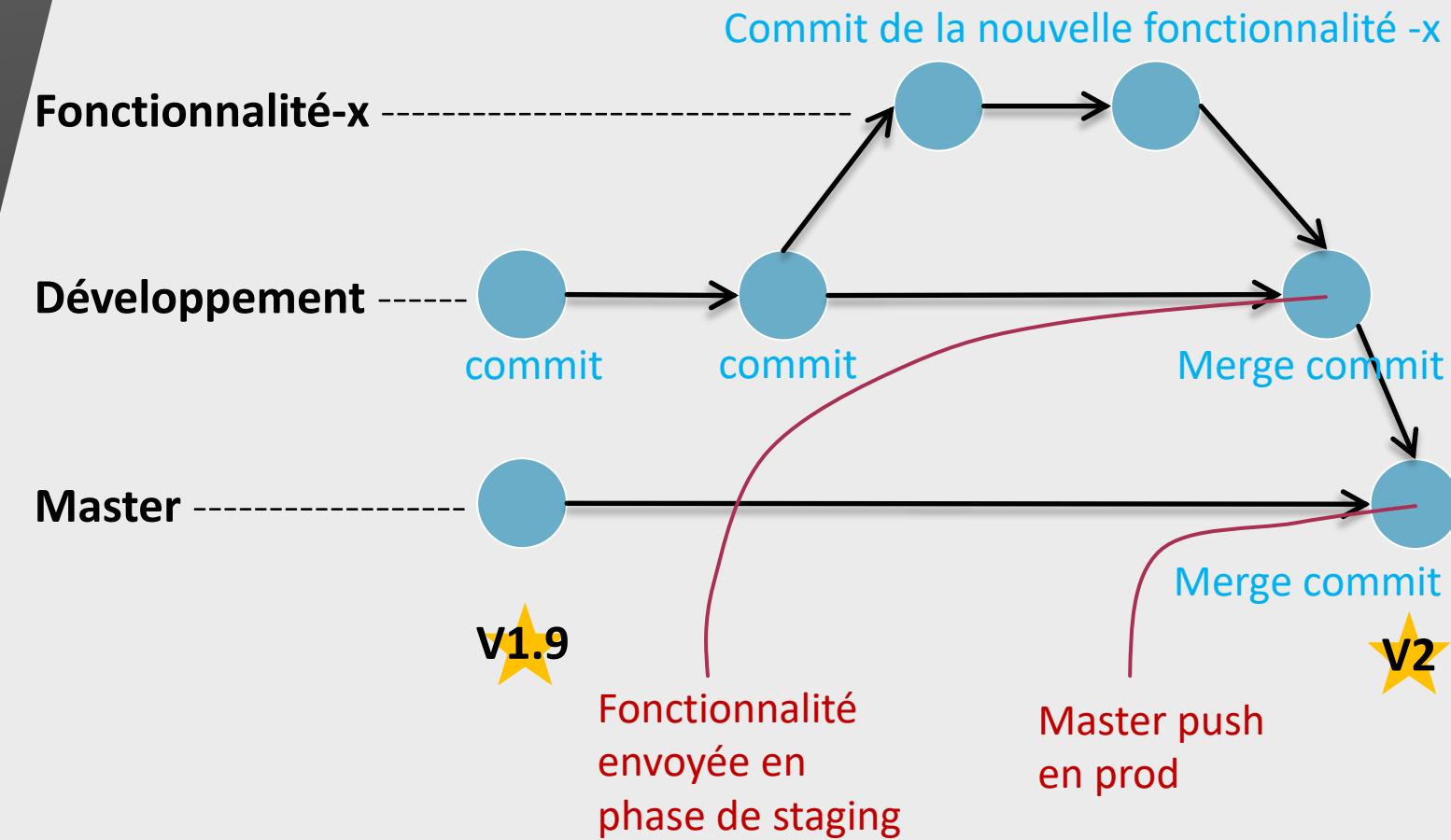
Si l'on veut faire un merge (joindre) les données des deux :

1. On se positionne sur le master : \$ git checkout master
  2. Taper : \$ git merge login
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 ?**

# EXEMPLE DE DÉROULEMENT D'UN PROJET

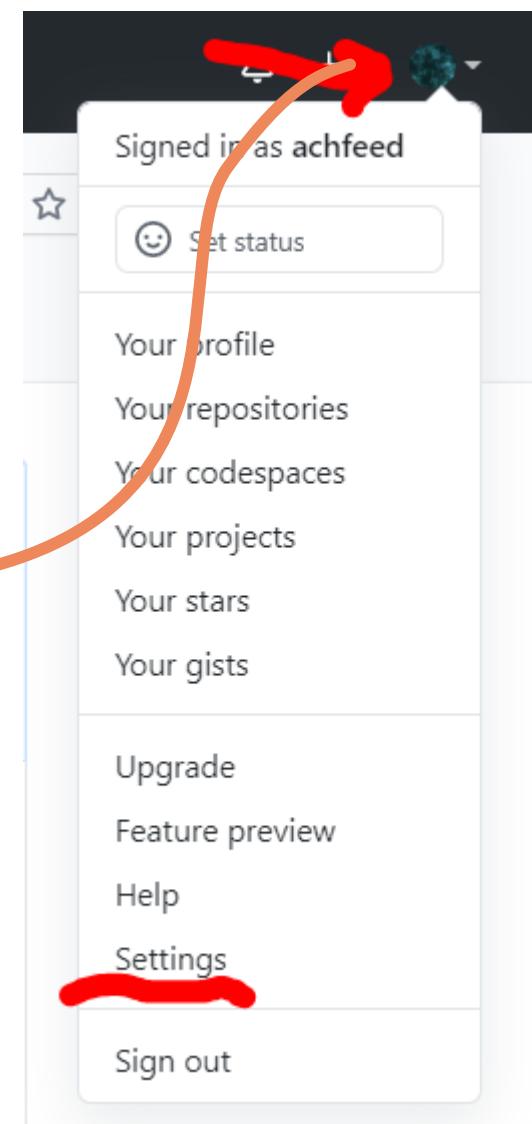


1. Créer un compte git en ligne : <https://github.com/>
2. Créer un nouveau repository : nom ...
3. \$ git config --global user.email 'achraf.....@outlook.com'
4. \$ git remote
5. \$ git remote add origin <git@github.com:achfeed/repr.git> (à copier à partir de votre repo).
6. ~~\$ git branch -M main~~
7. \$ git push -u origin main ////////////// \$ git push -u origin master
8. Il faut configurer 'ssh'.
9. Suivre :

10. Aller dans setting sur la partie : SSH and GPG keys

11. Remarquer la mention :

There are no SSH keys associated with your account.



## SSH keys

New SSH key

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

1

## 2 Generating a new SSH key and adding it to the ssh-agent

After you've checked for existing SSH keys, you can generate a new SSH key to use for authentication, then add it to the ssh-agent.

If you don't already have an SSH key, you must [generate a new SSH key](#). If you're unsure whether you already have an SSH key, check for [existing keys](#).

If you don't want to reenter your passphrase every time you use your SSH key, you can [add your key to the SSH agent](#), which manages your SSH keys and remembers your passphrase.

## Generating a new SSH key

- 1 Open Git Bash.
- 2 Paste the text below, substituting in your GitHub email address.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

**Note:** If you are using a legacy system that doesn't support the Ed25519 algorithm, use:

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private ed25519 key pair.
```

1. ssh-keygen -t ed25519 -C "[your-email@example.com](mailto:your-email@example.com)"
2. Pour les étapes d'après, laisser le chemin par défaut et appuyer sur entrer :  
"Enter file in which to save the key (/c/Users/ahamid/.ssh/id\_ed25519): "

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/ahamid/.ssh/id_ed25519): |
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/ahamid/.ssh/id_ed25519
Your public key has been saved in /c/Users/ahamid/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:IOiK7n2W0OPdnP2bvXhu5ju7G3sbKKue5nH1U/7E+/s achraf.hamid@outlook.com

The key's randomart image is:
+--[ED25519 256]--+
| . . . .
| . . . .
| . . . .
| . . . S o o . .
| .. . . + .. oo .
| o + . o ...=+
| . . = o .+ o .B@|
| .o .o . ==.. .* /E|
+---[SHA256]---
```

## Adding your SSH key to the ssh-agent

Before adding a new SSH key to the ssh-agent to manage your keys, you should have [checked for existing SSH keys](#) and [generated a new SSH key](#).

If you have [GitHub Desktop](#) installed, you can use it to clone repositories and not deal with SSH keys.

- 1 Ensure the ssh-agent is running. You can use the "Auto-launching the ssh-agent" instructions in "[Working with SSH key passphrases](#)", or start it manually:

```
# start the ssh-agent in the background
$ eval `ssh-agent -s`
> Agent pid 59566
```

1. \$ eval `ssh-agent -s`

```
ahamid@MIB-PORT36 MINGW32 ~/OneDrive - Mailinblack/Documents/mon_app (master)
$ eval `ssh-agent -s`
Agent pid 1668
```

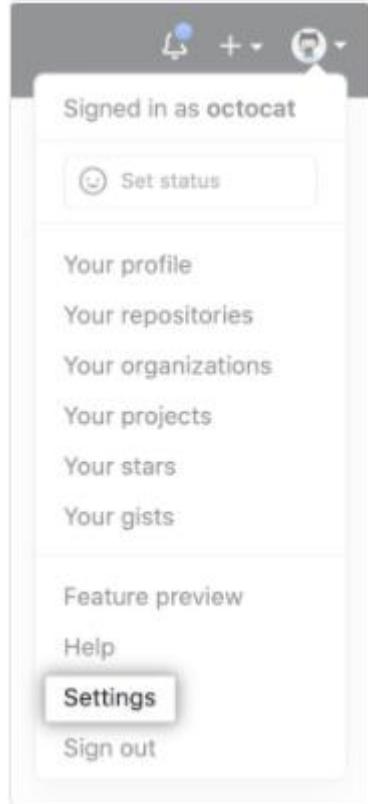
2. \$ ssh-add ~/.ssh/id\_ed25519

3. Add the SSH key to your GitHub account.

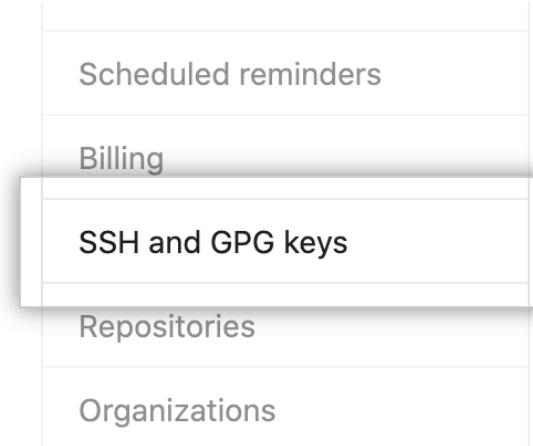
## Add the SSH key to your GitHub account.

1. clip < ~/.ssh/id\_ed25519.pub

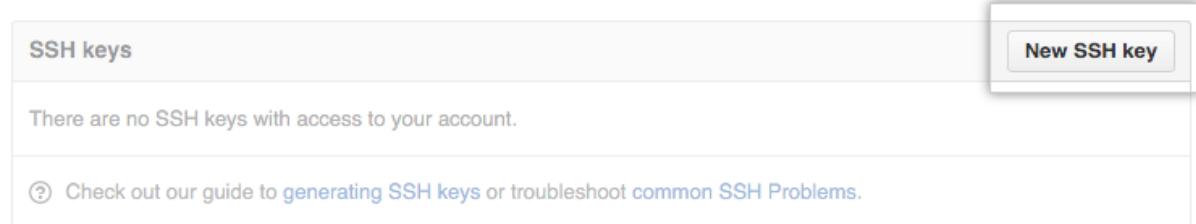
2.



3.



4. Click New SSH key or Add SSH key.



5. Dans le champ "Titre", ajoutez une description pour la nouvelle clé. Par exemple, si vous utilisez un ordinateur personnel, vous pouvez appeler cette clé "Ordinateur personnel".
6. Collez votre clé dans le champ "Clé" copiée à l'aide de la commande 'clip' :

SSH keys / Add new

---

Title

Key

```
ssh-ed25519 AAAAC3NzaC1I2D11NTE5AAAAIMPMdzQuBIT6ocTYispQbjIB4oQ1p1nMmY7
achraf      @outlook.com
```

|

---

Add SSH key

7.

Add SSH key

8. Rentrer le mot de passe si c'est demandé :

Confirm password to continue

Password

[Forgot password?](#)

Confirm password

9. La clé a bien été ajoutée : SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



SSH

Ordinateur personnel

SHA256:IOiK [REDACTED]

Added on 5 May 2021

Never used — Read/write

Delete

## Push :

```
git push -u origin master
```

```
$ git push -u origin master
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (18/18), 1.50 KiB | 770.00 KiB/s, done.
Total 18 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To github.com:achfeed/repr.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

1. Maintenant il faut aller sur Github.com
2. Regarder le contenu du repository :

 achfeed committing	52fa0fd 1 hour ago	 6 commits
 dir1	added file.js	3 hours ago
 .gitignore	added file.js	3 hours ago
 app.html	app.html	2 hours ago
 app.js	changed app.js	4 hours ago
 appli.html	appli.html	2 hours ago
 f.js	commiting	1 hour ago
 index.html	Initial commit	5 hours ago

Help people interested in this repository understand your project by adding a README.

Add a README



**GitHub**



# GIT PUSH



# GitHub



1. Créer un nouveau fichier : `vi main.go`
2. Dedans marquer une fonction vide : `func main() {}`
3. `git status`
4. `git add .`
5. `git status`
6. `git commit -m 'added main go empty function'`
7. `git log` : Que remarquez vous ?
8. Pour quitter les logs appuyer sur ‘q’
9. Que va-t-il se passer si je pars sur GitHub et que je rafraichis la page ?
10. Pour voir l'historique des commits :

The screenshot shows a GitHub repository interface. At the top, there are buttons for switching branches (master), viewing branches (1 branch), viewing tags (0 tags), navigating to a file, adding a file, and viewing code. Below this, a commit history is displayed. The first commit is shown with a green arrow pointing to it, indicating it is the current commit being discussed. The commit details are: author is achfeed committing, commit message is 'added file.js', date is 3 hours ago, and there are 6 commits in total.

File	Commit Message	Date	Commits
dir1	added file.js	3 hours ago	6 commits

## L'historique des commits :

master

- o Commits on May 5, 2021
  - commiting  
achfeed committed 1 hour ago
  - appli-html  
professeur git committed 2 hours ago
  - app.html  
professeur git committed 2 hours ago
  - added file.js  
professeur git committed 3 hours ago
  - changed app.js  
professeur git committed 4 hours ago
  - Initial commit  
professeur git committed 5 hours ago

Newer   Older

1. Pour le nouveau fichier de la machine local au remote GitHub, il faut utiliser git push :

```
$ git push
```

```
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:achfeed/repr.git
    52fa0fd..0be4f61  master -> master
```

2. On peut voir dans les commis :

The screenshot shows a GitHub commit history for the 'master' branch. The first commit, by user 'achfeed' on May 5, 2021, at 6 minutes ago, added a main go empty function. The second commit, also by 'achfeed' on May 5, 2021, at 2 hours ago, was a commit message. Both commits have their SHA-1 hash (0be4f61 and 52fa0fd) and a copy icon.

- o- Commits on May 5, 2021
  - added main go empty function  
achfeed committed 6 minutes ago
  - committing  
achfeed committed 2 hours ago

master

Commits on May 5, 2021

- added main go empty function  
achfeed committed 6 minutes ago
- commiting  
achfeed committed 2 hours ago



En cliquant sur le nom du commit on retrouve les informations ci-dessous :

added main go empty function

master

achfeed committed 7 minutes ago

1 parent 52fa0fd commit 0be4f61409311a486e1eb135e4a735f5b88128d7

Showing 1 changed file with 1 addition and 0 deletions.

Unified Split

1	main.go	...
...	...	@@ -0,0 +1 @@
1		+ func main() {}

0 comments on commit 0be4f61

Lock conversation



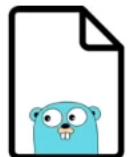
# GIT PULL





# GitHub





# GitHub

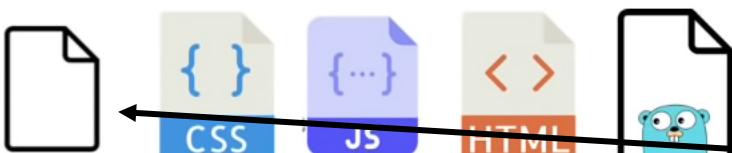


# GIT PULL

GitHub



git pull



## 1. Sur votre repo, appuyer sur Add a ReadMe

The screenshot shows a GitHub repository interface. At the top, there's a list of files and their commit history:

File	Commit Message	Time Ago
f.js	commiting	2 hours ago
index.html	Initial commit	5 hours ago
main.go	added main go empty function	16 minutes ago

Below the commit history, there's a blue call-to-action box containing the text: "Help people interested in this repository understand your project by adding:" followed by a green arrow pointing right, and a green "Add a README" button.

On the right side of the screen, there are sections for "Packages" (which says "No packages published" and "Publish your first package") and "Languages" (which shows a progress bar for HTML at 71.0%).

2. Vous pouvez copier ou créer votre ReadMe sur : <https://www.makeareadme.com/>
3. Pour l'ajouter, appuyer en bas sur le bouton 'Commit new file'.
4. Une fois ajouté, la next step est de Pull le ReadMe vers le local.
  
5. Sur Github : regarder l'historique des commits dans le repo
6. Comparer ça avec l'historique des commits en local avec 'git log', que remarquez vous ?

# GIT PULL

1. Pour le pull faire :
  - i. Git pull

```
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 792 bytes | 29.00 KiB/s, done.
From github.com:achfeed/repr
  0be4f61..c5f4202 master      -> origin/master
Updating 0be4f61..c5f4202
Fast-forward
 README.md | 11 ++++++++-
 1 file changed, 11 insertions(+)
 create mode 100644 README.md
```



2. Si on fait 'ls' on va voir que le fichier a été importé en local.

# CHANGEMENTS DE FICHIERS SUR GITHUB

1. Aller dans le repository et appuyer sur le fichier main.go
2. Pour éditer le fichier, appuyer sur le crayon comme indiqué ce dessous :

The screenshot shows a GitHub file view for 'main.go'. At the top, there's a dropdown for 'master', a 'repr' button, and a 'Jump to' dropdown. On the right are 'Go to file' and '...' buttons. Below that is a commit card for 'achfeed' adding an empty 'main' function. It shows 1 contributor and a commit made 31 minutes ago. To the right of the commit details are 'History' and 'Raw' buttons. At the bottom, the code editor shows a single line: 'func main() {}'. A green arrow points to the edit icon (pencil) in the toolbar.

master ▾ repr / main.go / <> Jump to ▾

achfeed added main go empty function

Latest commit 0be4f61 31 minutes ago History

1 contributor

1 lines (1 sloc) | 15 Bytes

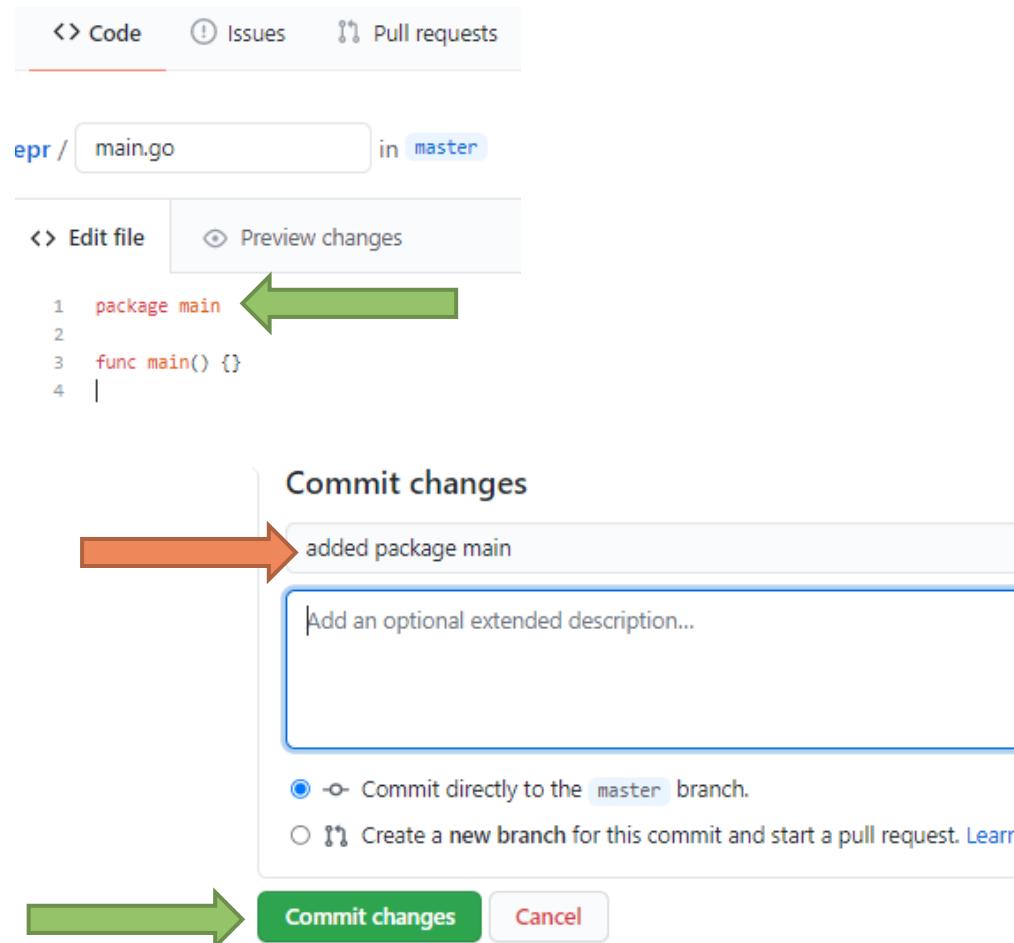
func main() {}

Raw Blame

Green arrow pointing to the edit icon (pencil) in the toolbar.

# CHANGEMENTS DE FICHIERS SUR GITHUB

1. Ajouter la ligne suivante :



2. Ensuite faire le commit :

# CHANGEMENTS DE FICHIERS SUR GITHUB

1. Sur le Git Bash, faire la commande suivante pour voir le contenu du fichier : cat main.go
2. Pour charger les nouvelles modifications, il faut faire : git pull
3. Si on refait : cat main.go

```
$ cat main.go
package main

func main() {}
```



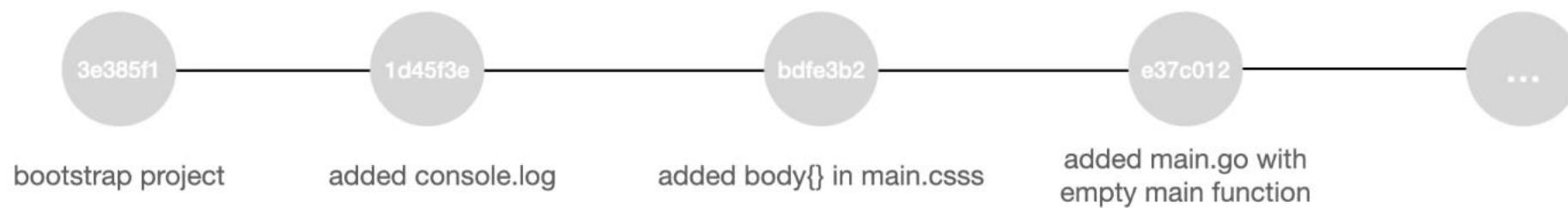


# BRANCHES



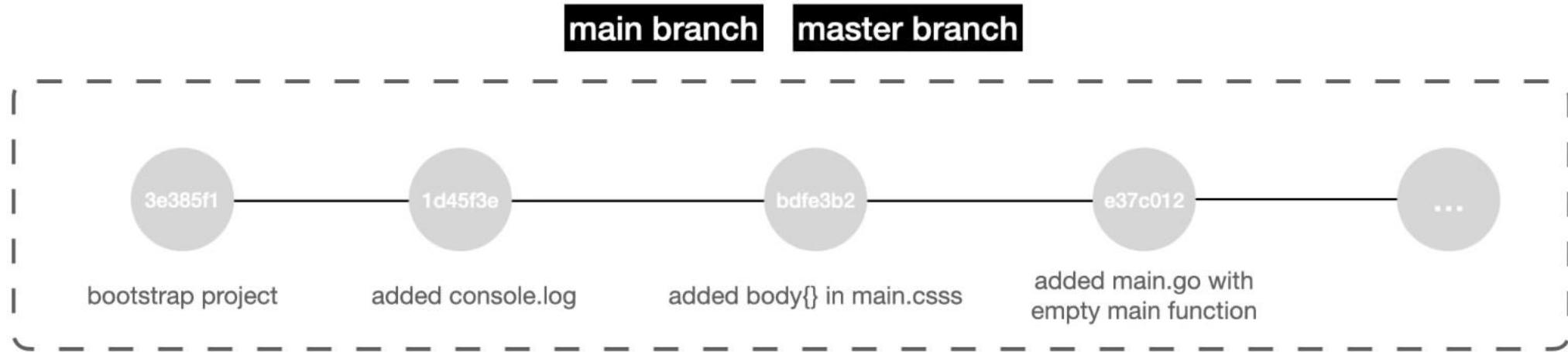


# BRANCHES



# BRANCHES

Tous les changements fais au fil du projet sont sur le main ou master qui sont deux appellations pour dire la même chose par ce que GitHub vient de faire une mise à jour pour appeler Master désormais Main mais c'est pas encore appliqué partout





# BRANCHES

default branch

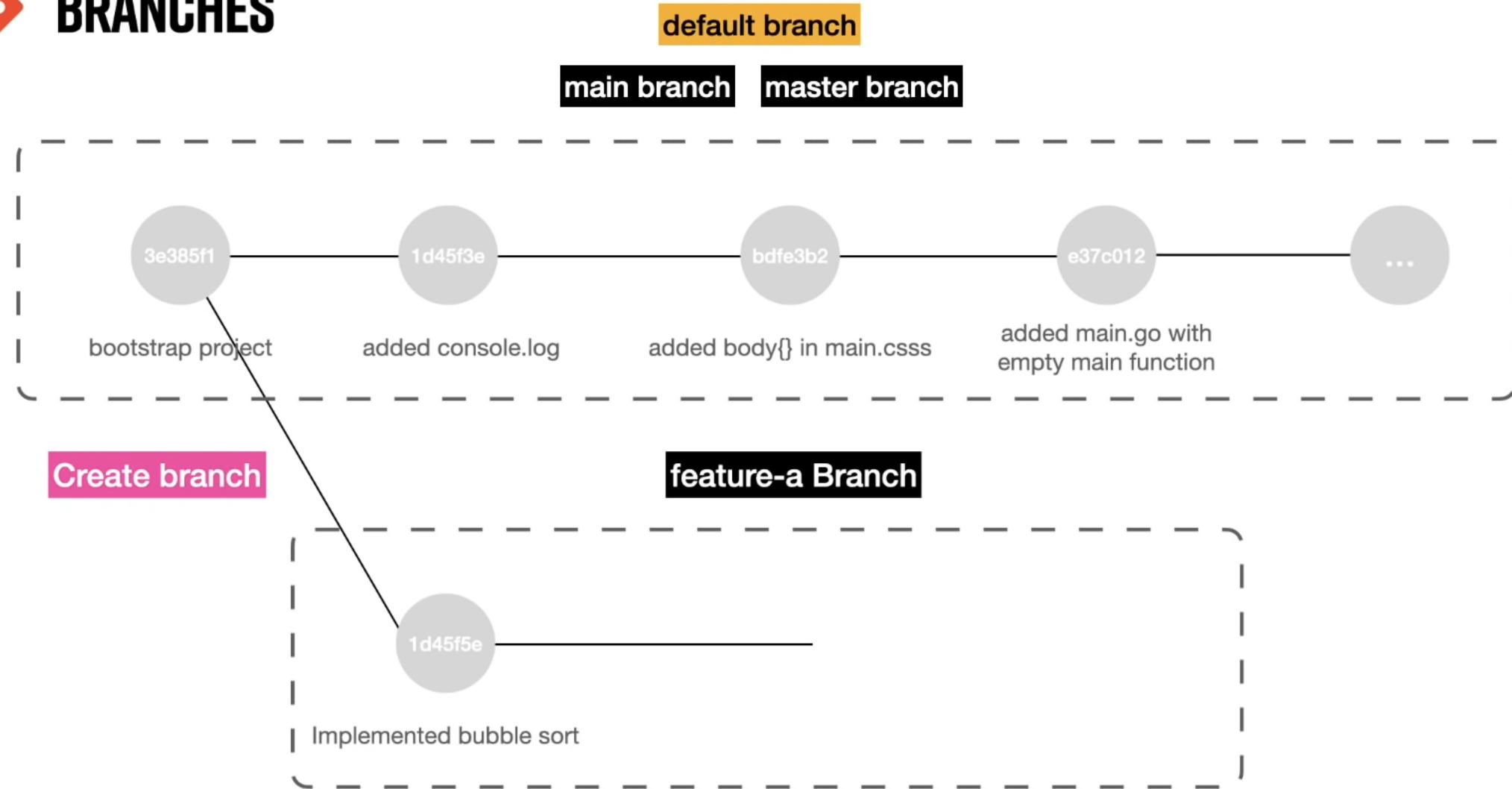
main branch

master branch





# BRANCHES



Pour développer une nouvelle fonctionnalité, on crée une nouvelle branche, ici on implémente un fonction 'bubble sort'



# BRANCHES

default branch

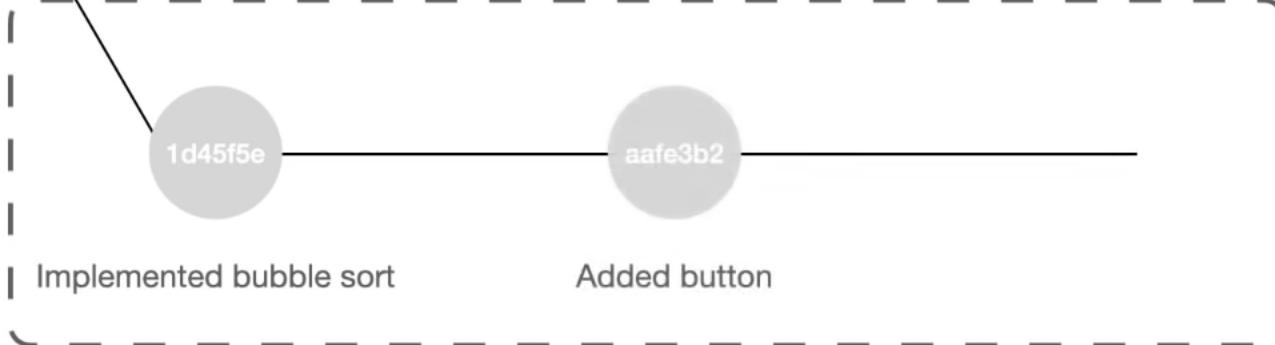
main branch

master branch



Create branch

feature-a Branch



Ensuite un bouton



# BRANCHES

default branch

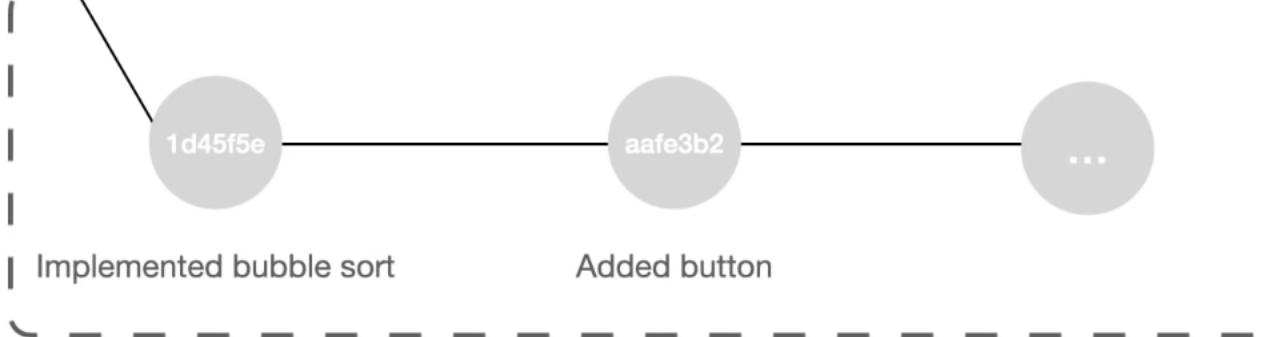
main branch

master branch



Create branch

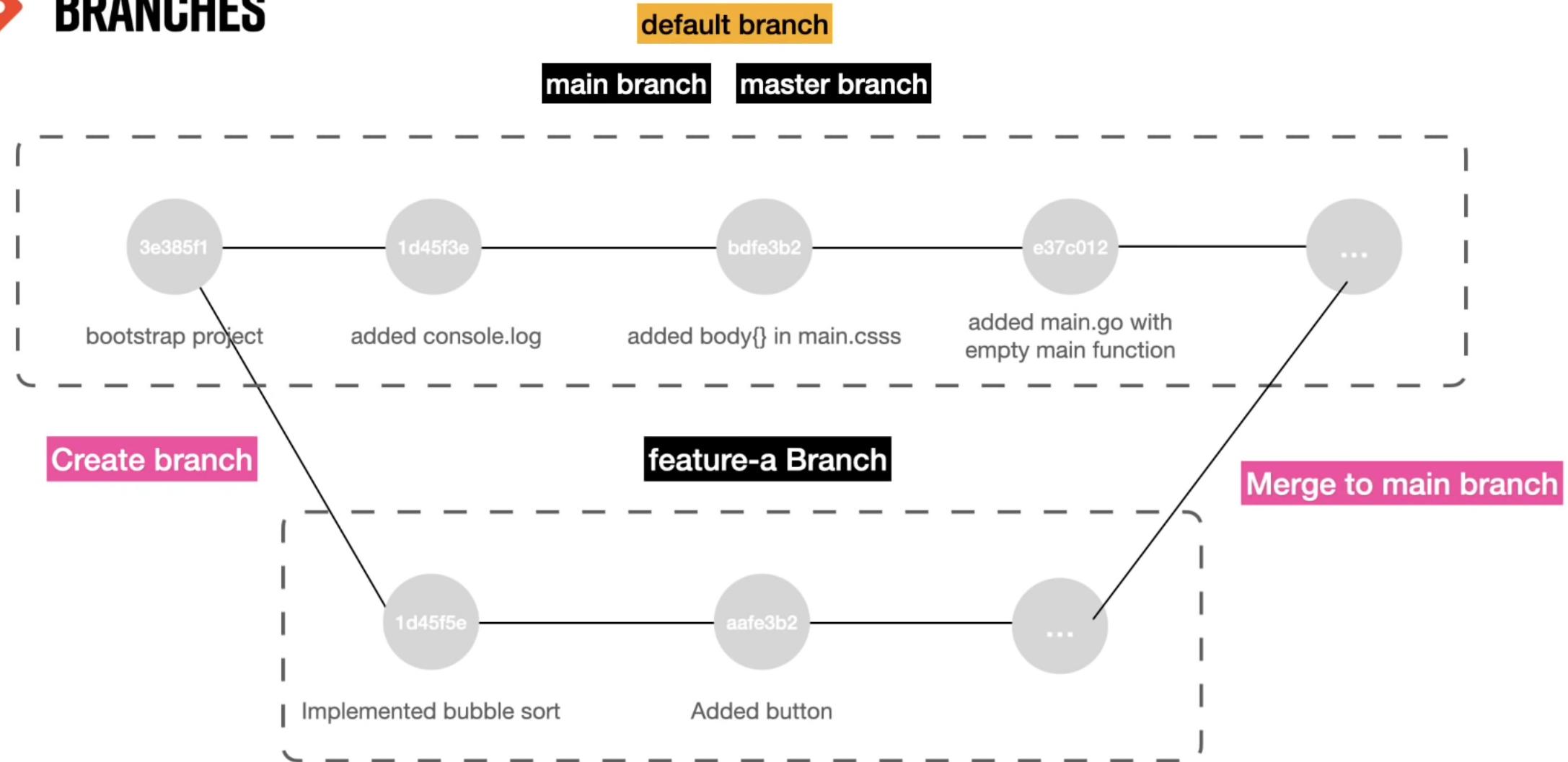
feature-a Branch



Ensuite une succession d'implémentations ...



# BRANCHES



A la fin il faut merge avec le master, ce nœud contiendra alors toutes les nouveautés à partir de la nouvelle branche.

# BRANCH

Pour savoir sur quelle branche on se situe :  
\$ git branch

```
$ git branch  
  login  
* master
```



Pour savoir si on est aussi connecté au remote :  
\$ git branch -r

```
$ git branch -r  
  origin/master
```

Pour afficher toutes les branches disponibles :  
\$ git branch -a

```
$ git branch -a  
  login  
* master  
  remotes/origin/master
```

# BRANCH

Pour ajouter une nouvelle branche

```
$ git branch feature-a
```

```
$ git branch -a
```

```
$ git branch -a
  feature-a
  login
* master
  remotes/origin/master
```

Pour changer de branche :

```
$ git checkout feature-a
```

Dans la nouvelle branche, créer un fichier :

```
$ vi utils.js
```

Dans le fichier, taper : // to do later

## BRANCH

```
$ git status  
$ git add .  
$ git commit -m 'utils.js avec todo'  
$ git log
```

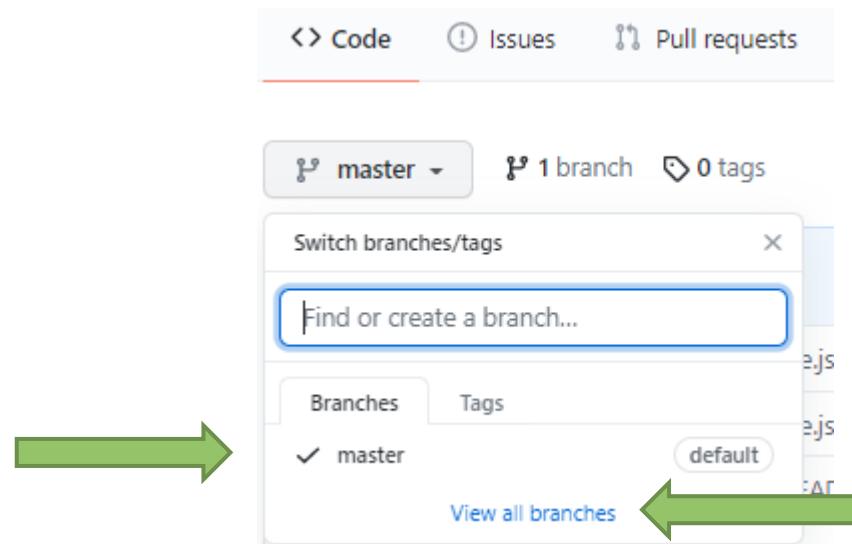
Si on fait un checkout vers master, que remarquez vous en voyant les logs ?

```
$ git checkout master  
$ git log
```

# BRANCH

Si on va sur le repo sur [github.com](https://github.com)

On peut voir sur les branches que la branche nouvellement créée n'est pas là



Si on appuie sur : View all branches on peut vérifier toutes les branches.

# BRANCH

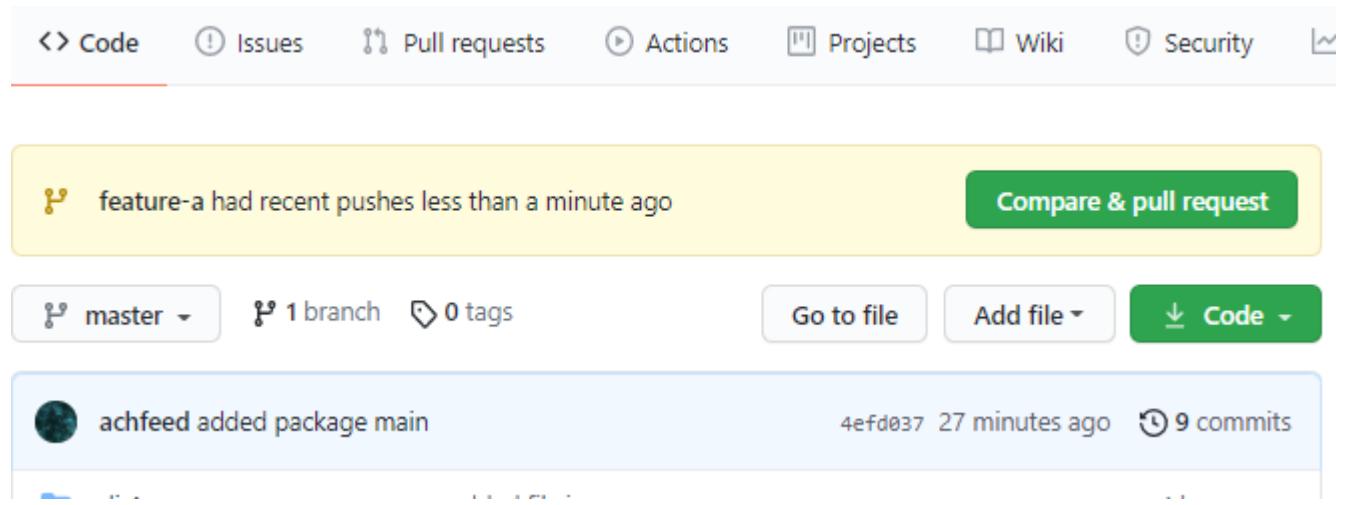
Pour rajouter cette nouvelles branche, il faut aller sur la branche:  
\$ git checkout feature-a

Ensuite faire un push:

\$ git push origin feature-a

Cette méthode nous permet de faire un push de cette branche uniquement, si on a d'autres branches qu'on ne veut pas ajouter par exemple.

Le message suivant s'affiche :



# BRANCH

Comparer les commits et les fichiers des deux branches.

## Supprimer une branche :

On crée d'abord la branche:

```
$ git checkout master  
$ git branch to-delete  
$ git branch -d to-delete
```

```
$ git branch -d to-delete  
Deleted branch to-delete (was 4efd037).
```



# PULL REQUESTS



# PULL REQUESTS

Si vous êtes en entreprise, vous n'allez jamais un push ou un pull directement vers le master.

Ce que vous allez faire c'est :

1. Créer une nouvelle branche ;
2. Faire vos changements ;
3. Faire un push pour ne pas perdre vos changements ;
4. Ensuite vous faites un pull request, de telle sorte à ce que quelqu'un fasse un review de vos modifications et s'assurer que tout est cohérent



# MERGING PR'S



# MERGING PULL REQUESTS

La meilleure façon de merge est de le faire à travers un pull request.

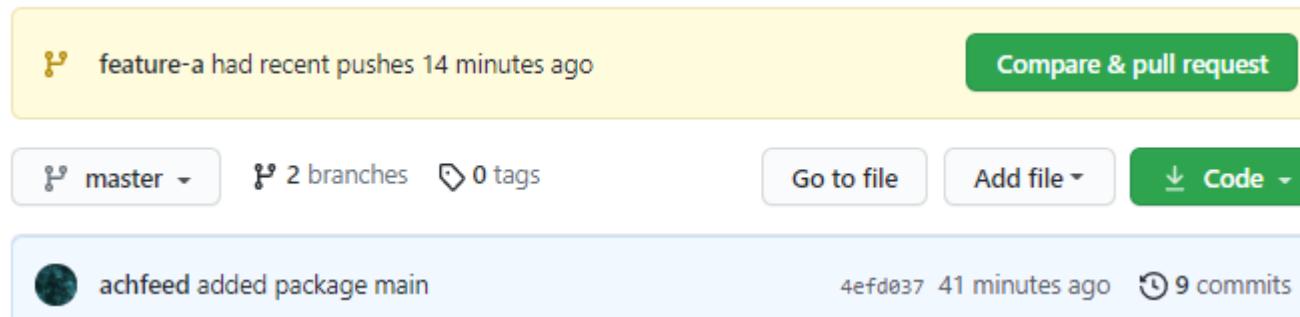
Nous avons déjà deux branches : Master et feature-a

Il est possible de faire : \$ git merge feature-a

Ceci va prendre les changements à partir de feature-a vers le Main ou le Master.

Pour faire ça proprement on va le faire avec un pull request.

Remarquer ce message sur votre Github :



Ici on appuie sur pull request.

# MERGING PULL REQUESTS

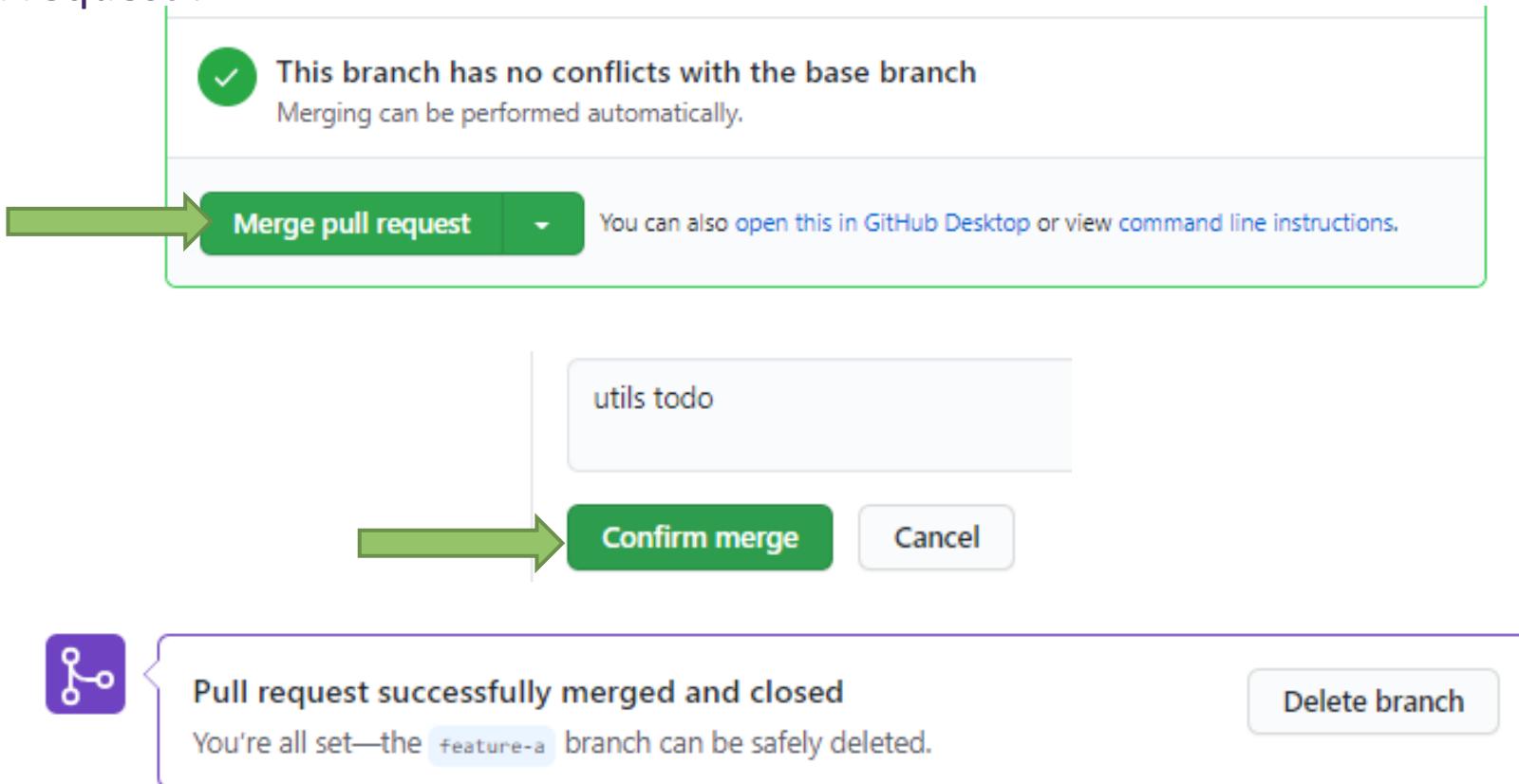
## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

The screenshot shows a pull request creation interface. At the top, there are dropdown menus for 'base: master' and 'compare: feature-a'. To the right of these is a green checkmark icon followed by the text 'Able to merge. These branches can be automatically merged.' Below this, there's a user profile picture and a text input field containing 'utils todo'. Underneath the input field is a toolbar with 'Write' (selected), 'Preview', and various rich text editing icons (H, B, I, etc.). A large text area below is labeled 'Leave a comment'. At the bottom, there's a note to 'Attach files by dragging & dropping, selecting or pasting them.' and a green 'Create pull request' button with a dropdown arrow. An orange arrow points to the 'Create pull request' button.

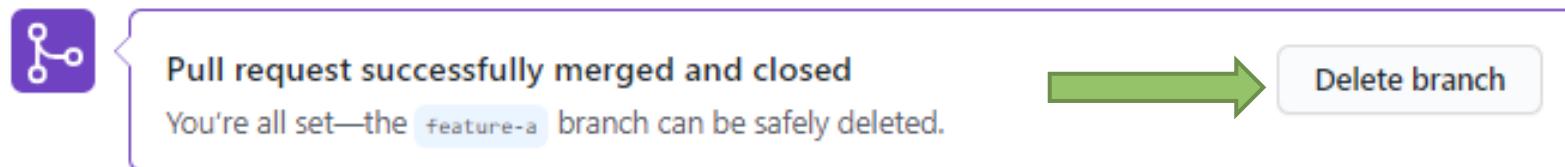
# MERGING PULL REQUESTS

En entreprise, ce serait à quelqu'un de faire le review, ici on saute cette étape et on fait directement le merge pull request :



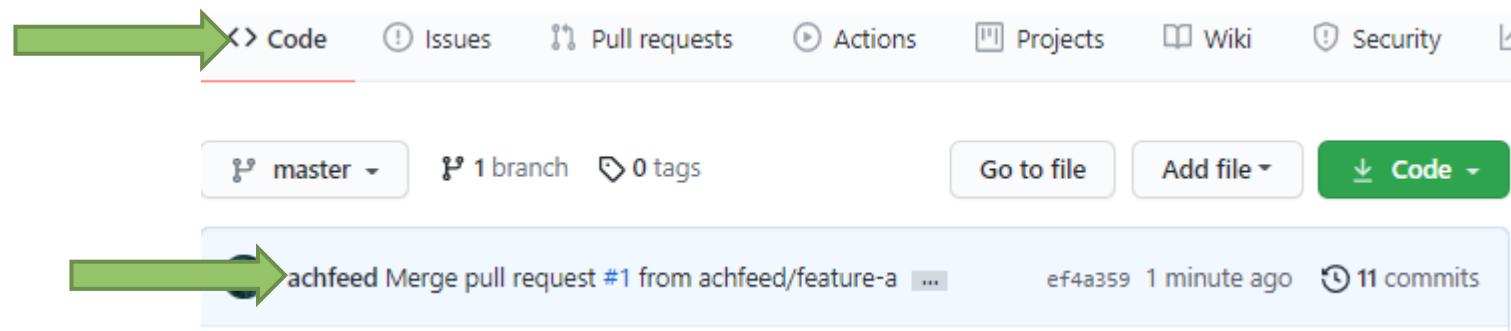
# MERGING PULL REQUESTS

On peut maintenant supprimer la nouvelle branche :



On pourra toujours la restaurer si besoin.

Si on revient sur l'onglet code du repo :



On peut rentrer regarder les détails dans les commits, on peut appuyer sur chaque commit pour voir plus en détails

# MERGING PULL REQUESTS

Sur le Git Bash, on reverifie les logs:

```
$ git log --oneline
```

On remarque que les nouveaux changements ne sont pas réalisés.

```
$ git pull  
$ git log --oneline
```

```
→ ef4a359 (HEAD -> master, origin/master) Merge pull request #1 from achfeed/feature-a  
09f63dc (origin/feature-a, feature-a) utils todo  
4efd037 added package main  
c5f4202 Create README.md  
0be4f61 added main go empty function  
52fa0fd committing  
e22e448 appli-html  
88299c2 app.html  
cc52ac2 added file.js  
49e8c48 changed app.js  
6c6f3d1 Initial commit
```

# MERGING PULL REQUESTS

On peut aussi supprimer la branche feature-a de notre machine en local :

```
$ git branch -d feature-a
```

Pour supprimer une branche en ligne et en local :

```
$ git push origin --delete feature-a
```

Si vous avez supprimer la branche en ligne mais pas en local :

```
$ git fetch --prune
```

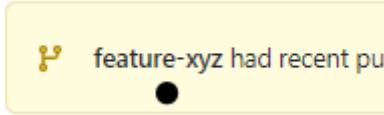
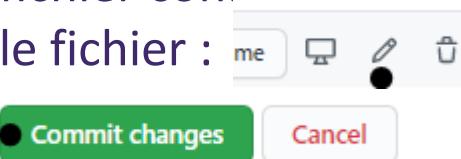


# DEALING WITH CONFLICTS



# CONFLICTS

Ici on va voir comment se comporter avec les conflits qui peuvent arriver :

1. Créer une nouvelle branche : \$ git branch feature-xyz
2. Y accéder : \$ git checkout feature-xyz
3. Créer un fichier html : \$ touch conflict.html
4. Aller sur le lien : <https://www.w3schools.com/html/>
5. Copier l'exemple dans votre 'conflict.html' 
6. Add : \$ git add .
7. Commit : \$ git commit -m "add html"
8. Push : git push --set-upstream origin feature-xyz
9. Aller sur GitHub.com
10. Ouvrir la nouvelle branche feature-xyz : 
11. Ouvrir le fichier conflict.html
12. Modifier le fichier :  Rajouter quelque lignes entre <p> et </p>
13. Commit

## Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
```



# CONFLICTS

On vient maintenant de faire une simulation du cas où quelqu'un d'autre a changé un fichier sur lequel nous sommes entrain de travailler en local. Si on regarde le fichier en local, les lignes qu'on a rajouté en ligne n'y sont pas, vérifiez cela.

En local sur votre fichier, rajouter dans le body du html : <p>This is another a paragraph.</p>  
Maintenant qu'on a terminé nos modifications, on essaie de faire un push :

1. \$ git add .
2. \$ git commit -m "added a paragraph"
3. \$ git push

```
! [rejected]          feature-xyz -> feature-xyz (fetch first)
error: failed to push some refs to 'github.com:achfeed/repr.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes.
```



# MERGING CONFLICTS



# MERGE DES CONFLICTS

Avant de faire le push il faut commencer par un pull :

```
$ git pull
```

```
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 699 bytes | 77.00 KiB/s, done.  
From github.com:achfeed/repr  
  63f75e5..cf5da5e  feature-xyz -> origin/feature-xyz  
Auto-merging conflict.html  
CONFLICT (content): Merge conflict in conflict.html  
Automatic merge failed; fix conflicts and then commit the result.
```



# MERGE DES CONFLICTS

En ouvrant maintenant le fichier avec notepad++ ou Vscode, on remarque que les deux commits sont là.

Il faut résoudre cette erreur.

Tout ce qui avant le séparateur ‘=====’ fait référence aux changements que nous avons fais en local.

Tout ce qui est après le séparateur fait référence aux changements faits par d'autre personnes.

Les modifications des autres sont suivies par le hash de leur commit.

>>>>> cf5da5ee42d74afcc52a2f2b7e09fb649c9e6898

Créer une nouvelle version en nettoyant le code :

```
--<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<p>This is another a paragraph.</p>
<p>This is a paragrfd
      dfdffaph.</p>
</body>
</html>
```

# MERGE DES CONFLICTS

Nouvelle réponse de Git avec : \$ git status



```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified: conflict.html
```

```
$ git add .
$ git commit -m 'conflict resolved'
$ git push
```

Pas de conflits cette fois tout est réglé !

On peut vérifier ça sur le fichier conflict.html directement sur GitHub.com

# EXERCICES

## Exercice 1

1. Créer une branche qui s'appelle : apprentissage
2. Aller dans cette branche
3. Créer un nouveau fichier html dans cette branche en utilisant touch
4. Créer un deuxième fichier nommé (info.txt) que vous allez rajouter dans votre gitignore
5. Envoyer cette branche en ligne.
6. Faire un merge avec main ou master en ligne

## Exercice 2

1. Créer une branche qui s'appelle : apprentissagelocal
2. Aller dans cette branche
3. Créer un nouveau fichier html dans cette branche en utilisant touch
4. Faire un merge de cette branche avec le main ou master en local.

## Exercice 3

1. Sur une branche de votre choix, créez un conflit et gérez le en local :



# BONNES PRATIQUES GITHUB

- Ne poussez pas directement vers le maître.
- Ne pas faire de commit de code en tant qu'auteur non reconnu.
- Définir les propriétaires des codes pour un examen plus rapide des codes
- Ne divulguez pas de mots de passes dans le contrôle des sources.
- Créer un fichier git ignore
- Archiver les dépôts périmés/dépassés.
- Supprimer les membres inactifs de GitHub.
- Activer les alertes de sécurité.

<https://www.datree.io/resources/github-best-practices>

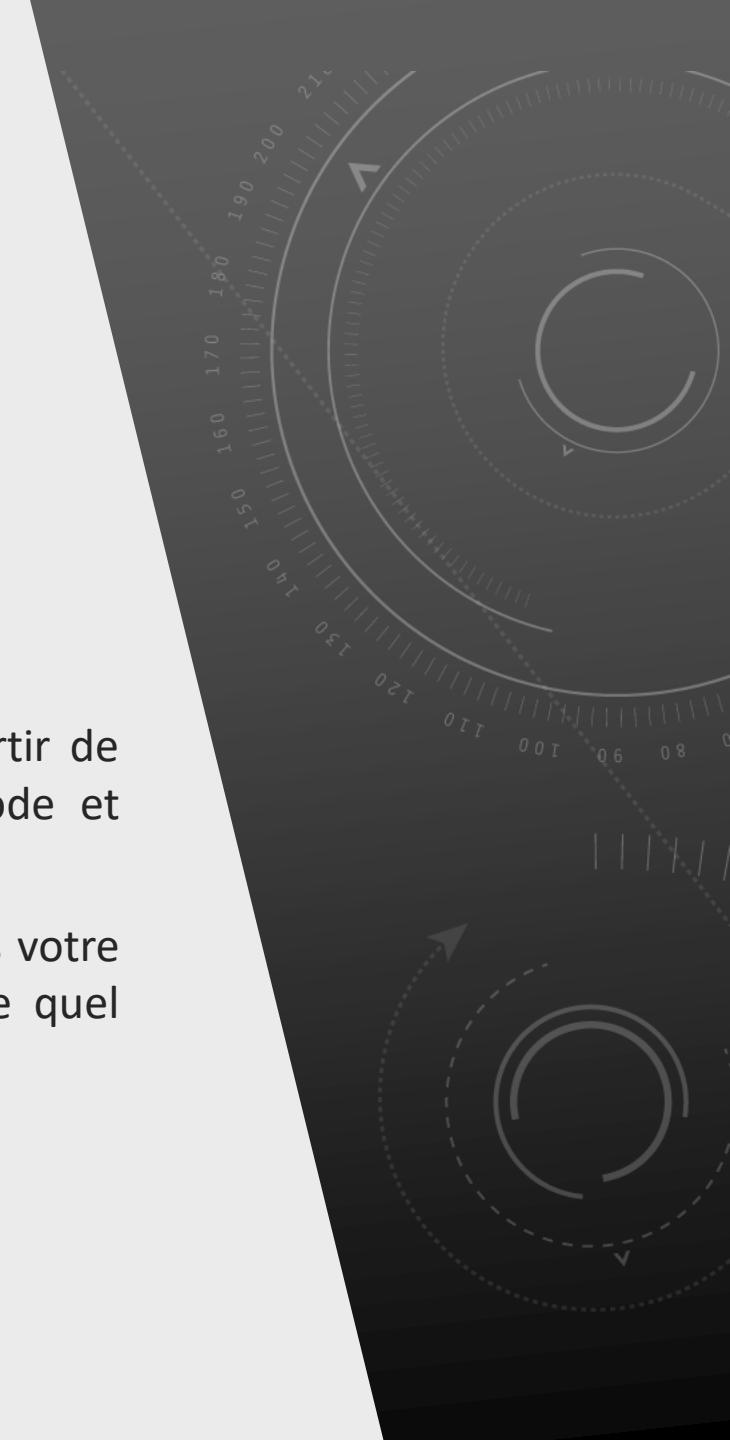


GITHUB ACTIONS

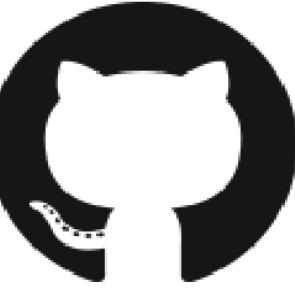
# GITHUB ACTIONS

GitHub Actions vous aide à automatiser vos workflows de développement logiciel à partir de GitHub. Vous pouvez déployer des workflows à l'emplacement où vous stockez le code et collaborez sur des demandes et des problèmes de tirage.

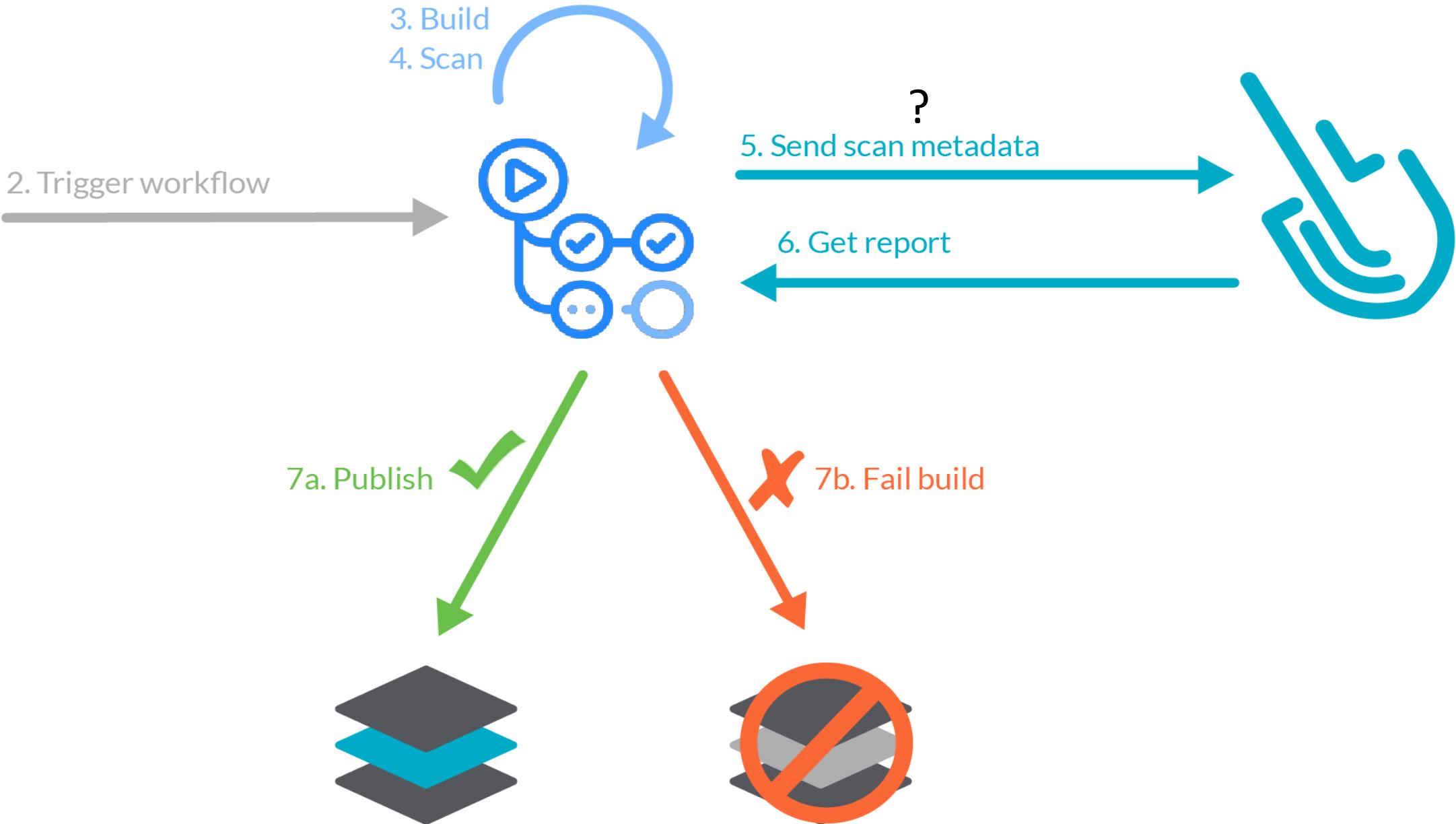
Dans GitHub Actions, un **workflow** est un processus automatisé que vous configurez dans votre dépôt GitHub. Vous pouvez générer, tester, empaqueter, publier ou déployer n'importe quel projet sur GitHub avec un workflow.



1. Commit



# GITHUB ACTIONS WORKFLOW EXAMPLE



# GITHUB ACTIONS

Pour automatiser un ensemble de tâches , vous devez créer des flux de travail dans votre repo GitHub. Des événements tels que les validations, l'ouverture ou la fermeture de pull requests ou les mises à jour du wiki du projet déclenchent le démarrage d'un workflow. Pour une liste complète des événements disponibles.

Les flux de workflows sont composés de travaux, qui s'exécutent simultanément par défaut. Chaque travail doit représenter une partie distincte de votre flux de travail.

Par exemple, vous pouvez avoir une tâche pour exécuter vos tests, une autre pour la publication de votre logiciel et une troisième pour le déploiement dans votre environnement de production.

Vous pouvez configurer les travaux pour qu'ils dépendent du succès des autres travaux dans le même flux de travail. Par exemple, l'échec des tests peut empêcher le déploiement en production.



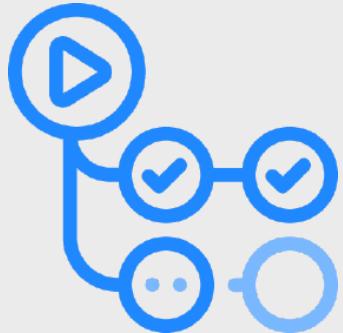
# GITHUB ACTIONS

Les jobs contiennent une liste d'étapes que GitHub exécute dans l'ordre. Une étape peut être un ensemble de commandes shell ou une action, qui est une étape pré-construite et réutilisable.

Certaines actions sont fournies par l'équipe GitHub, tandis que la communauté open-source en maintient bien d'autres.



# GITHUB ACTIONS CLOUD COMPATIBLE



GitHub Actions pour Azure est développé par Microsoft et conçu pour être utilisé avec Azure. Vous pouvez voir l'intégralité de GitHub Actions pour Azure dans GitHub Marketplace.

Azure Pipelines et GitHub Actions vous aident tous deux à automatiser les workflows de développement logiciel. Apprenez-en davantage sur les différences entre les services et sur la migration d'Azure Pipelines vers GitHub Actions.



# GITHUB ACTIONS CLOUD FONCTIONNALITÉS AZURE



- Déployer sur une application web statique
- Déployer sur Azure Functions pour les conteneurs
- Connexion Docker
- Kubernetes - Définir le contexte
- Déclencher une exécution Azure Pipelines
- Connexion Machine learning
- Entraînement Machine learning
- Machine learning - déployer un modèle
- Déployer sur l'action Azure MySQL
- Analyse de conformité



# DEVEZ-VOUS UTILISER LES ACTIONS GITHUB?

La réponse est, comme toujours: *cela dépend*.



# GITHUB ACTIONS

Si vous utilisez déjà GitHub pour héberger le code source de votre projet, il est facile de démarrer avec les actions GitHub. Le fait qu'il s'intègre pleinement à l'ensemble de l'écosystème GitHub signifie que votre équipe peut doubler l'utilisation de la plate-forme dans le cadre de votre processus de développement logiciel.

GitHub Actions est gratuit pour tous, les projets et les référentiels privés atteignent 2000 minutes (33h20min) par mois avant d'être facturés quoi que ce soit.

Pour les petits projets, cela signifie pouvoir profiter pleinement de l'automatisation dès le début sans frais supplémentaires. Vous pouvez même utiliser le système gratuitement pour toujours si vous utilisez des self-hosted runners.

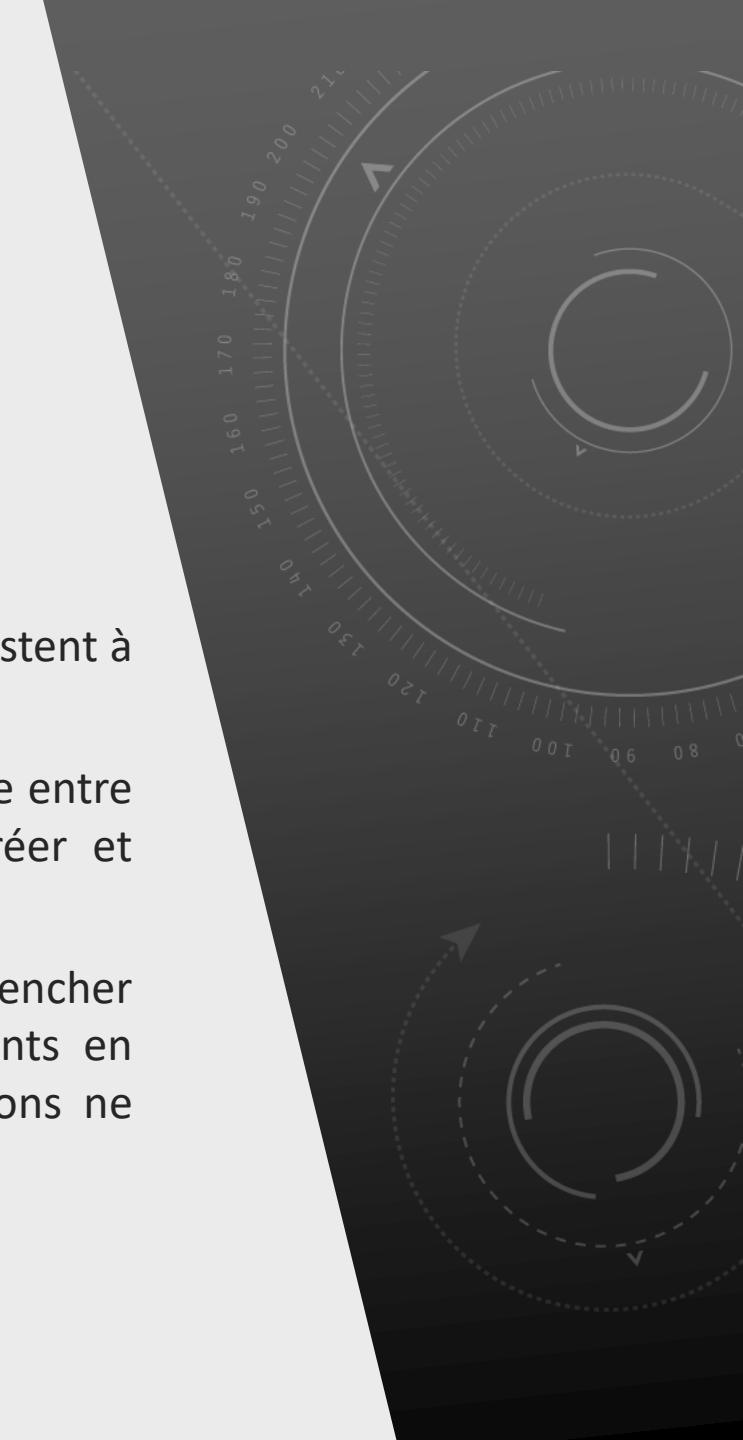


# GITHUB ACTIONS

Alors que la communauté Market place a un potentiel immense, de nombreuses actions restent à construire par des personnes comme vous et moi.

Dans les grandes organisations, le développement d'actions personnalisées et leur partage entre les équipes peuvent réduire considérablement la quantité de travail requise pour créer et maintenir des flux de travail entre les projets.

Cependant, une fonctionnalité qui me manque personnellement est la possibilité de déclencher manuellement des tâches spécifiques. Certaines équipes préfèrent que les déploiements en production nécessitent une intervention humaine, ne serait-ce qu'un clic. GitHub Actions ne fournit pas encore d'option intégrée pour cela.



# GITHUB ACTIONS

Les développeurs peuvent adapter leur flux de travail afin que seuls les commits balisés aboutissent à des déploiements de production, par exemple.

Alors que GitHub encourage les équipes à adopter la pratique courante du déploiement continu, certains ne souhaitent peut-être pas encore franchir ce pas.

Dans l'ensemble, mon avis est que les actions GitHub valent la peine d'être essayées. Que ce soit le système d'automatisation le mieux adapté à votre équipe dépend de vos besoins spécifiques.





# GITLAB

GitLab est un système de contrôle de version populaire employé principalement en développement logiciel. Programmé avec « Ruby on Rails » par Dmitri Saporoschez, le logiciel basé sur Web sorti en 2011 est aujourd’hui devenu incontournable dans les cercles de développeurs.

L'avantage majeur de GitLab est qu'il facilite grandement le développement logiciel agile et collaboratif. Plusieurs développeurs peuvent travailler en même temps sur un projet et par exemple éditer des fonctions en parallèle. La journalisation continue de toutes les opérations assure que les modifications apportées au code ne sont jamais perdues ni écrasées par mégarde. De plus, les modifications déjà effectuées peuvent être annulées sans peine.



# GITLAB

GitLab est basé sur le logiciel de gestion de versions très répandu Git. Librement accessible car open source, Git fait partie des systèmes de gestion de versions les plus utilisés.

GitLab représente l'une des alternatives à GitHub les plus connues (suite au rachat de GitHub par Microsoft en 2018, beaucoup d'utilisateurs sont passés à GitLab).



# CARACTÉRISTIQUES DE GITLAB

Citons parmi les principales caractéristiques de GitLab :

- Interface utilisateur conviviale
- Branches privées et publiques
- Gestion de plusieurs repositories
- Revue de code
- Suivi intégré des bugs et issues
- Intégration/livraison continue (CI/CD) gratuite incluse
- Wikis de projet
- Création facile de snippets (partage de petits bouts de code)



# DIFFÉRENCE ENTRE GITHUB ET GITLAB

**De base :** GitHub et GitLab sont tous deux des services d'hébergement de référentiels Git basés sur le Web, qui suivent l'évolution des projets de développement logiciel et de ses fichiers au fil du temps, ce qui permet aux développeurs de collaborer sur des projets Web sous un même toit.

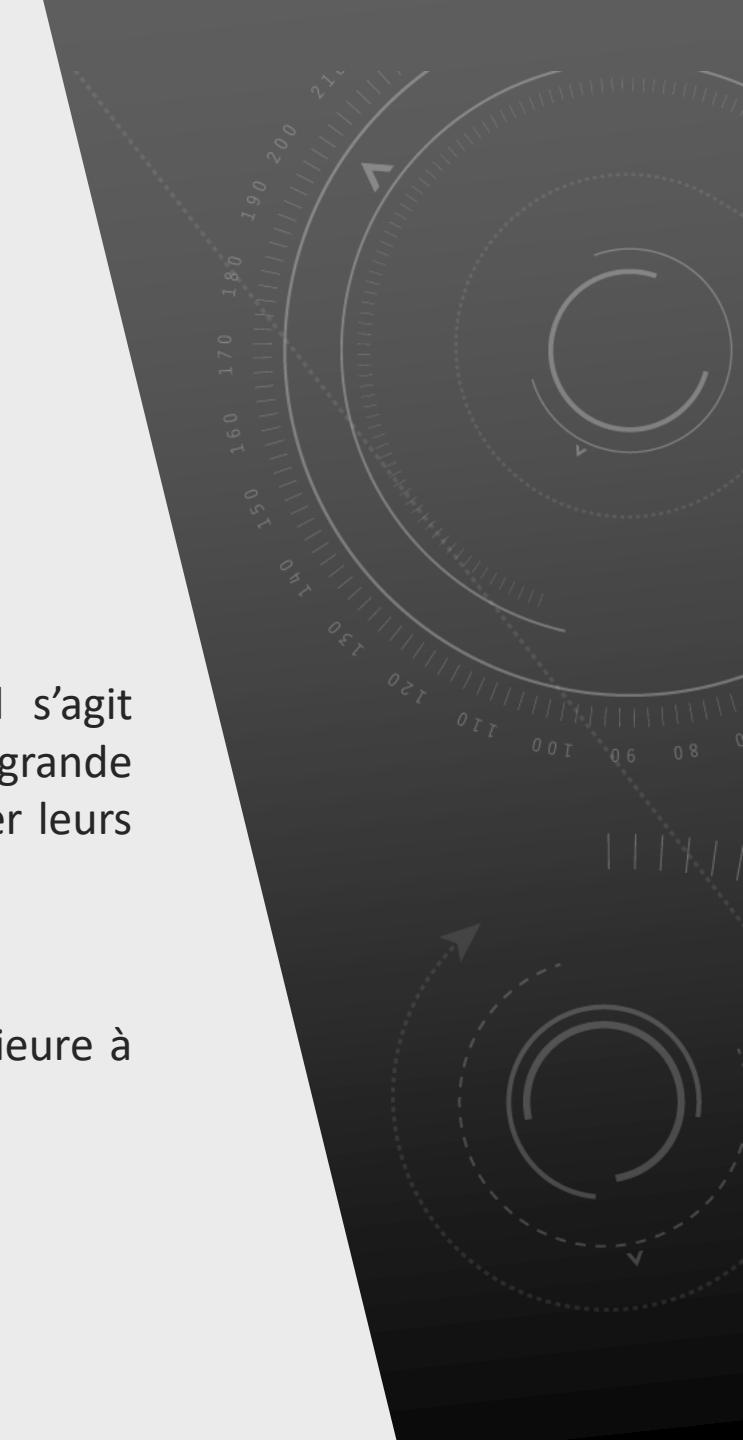
Tout comme GitHub, GitLab est un gestionnaire de référentiels pour la collaboration collective, mais avec une interface utilisateur plus intuitive et sa protection de branche, ses autorisations et ses fonctionnalités d'authentification, font la différence.



# DIFFÉRENCE ENTRE GITHUB ET GITLAB

**Popularité** : GitHub est probablement le premier nom qui frappe l'esprit quand il s'agit d'hébergement de référentiels avec contrôle de version, qui rassemble la plus grande communauté de développeurs du monde pour collaborer sur des projets Web et partager leurs idées en matière de processus de développement de logiciels.

En tant que principal service d'hébergement de référentiels, sa popularité est bien antérieure à celle de GitLab, une plate-forme beaucoup plus récente lancée en 2011.



# DIFFÉRENCE ENTRE GITHUB ET GITLAB

**Open source :** L'une des principales différences entre les deux est que GitHub n'est pas un logiciel à source ouverte, mais qu'il propose des plans payants pour des référentiels privés couramment utilisés pour héberger des projets Web à code source ouvert.

Le service hébergé est en fait gratuit pour les projets open-source, mais le logiciel sur lequel il est basé n'est pas open source. GitLab, en revanche, est une source libre et ouverte pour Community Edition, tandis que l'édition Enterprise est en source fermée.



# DIFFÉRENCE ENTRE GITHUB ET GITLAB

**Niveau d'authentification :** Il fait référence à une autorisation basée sur les niveaux d'accès.

Dans GitHub, les propriétaires ou les équipes d'organisation peuvent ajouter des référentiels Git ainsi que modifier leurs accès en lecture, en écriture et administrateur à ces référentiels.

Vous pouvez également inviter des utilisateurs à collaborer sur votre référentiel personnel en tant que collaborateurs.

Dans GitLab, les utilisateurs ont différents niveaux d'accès dans un groupe ou un projet particulier en fonction de leurs rôles respectifs.

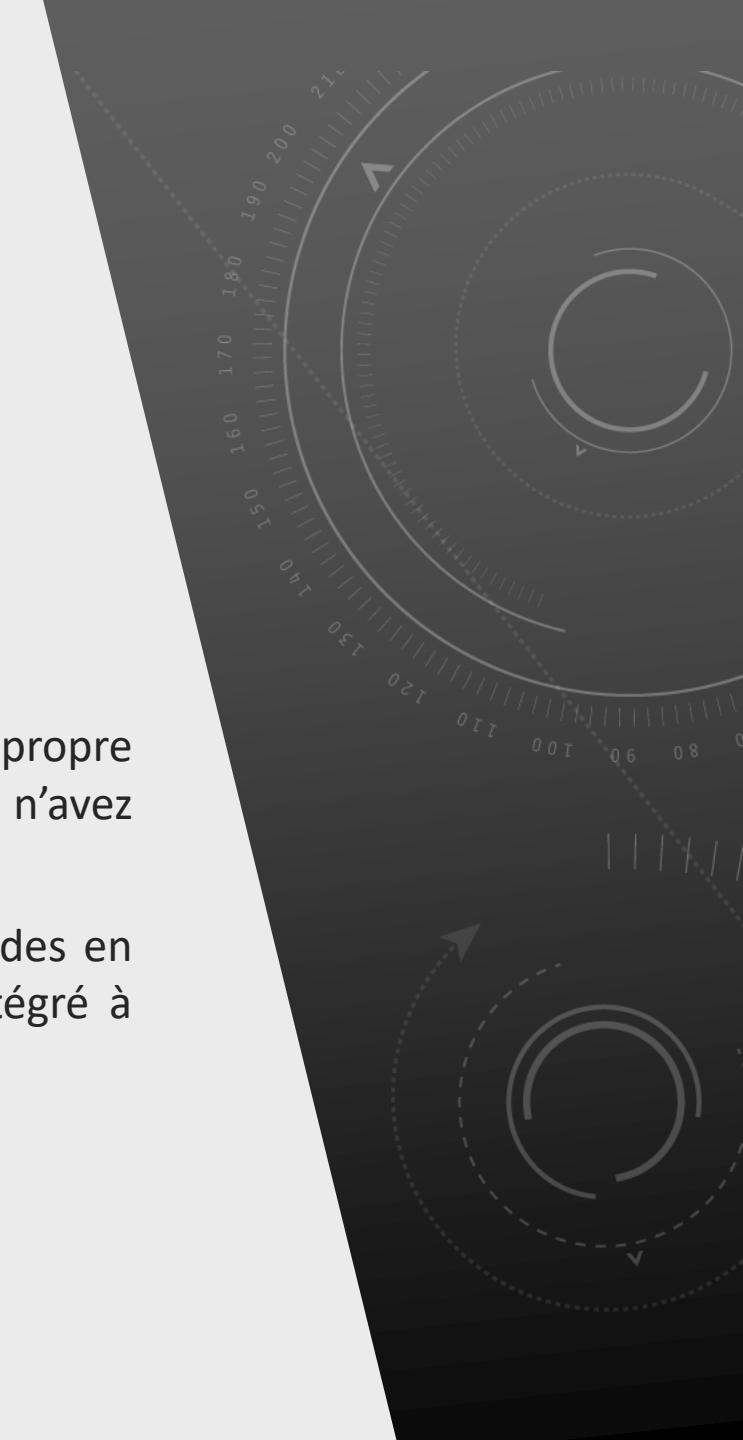
Les administrateurs de GitLab reçoivent essentiellement toutes les autorisations.



# DIFFÉRENCE ENTRE GITHUB ET GITLAB

**CI / CD intégré** : L'une des principales différences entre les deux est que GitLab offre son propre système d'intégration / livraison continue (CI / CD) pré-construit, ce qui signifie que vous n'avez pas besoin de l'installer séparément.

Cela aidera les équipes à réduire les erreurs de code et à obtenir des résultats plus rapides en respectant les normes de qualité de votre équipe. Au contraire, il ne vient pas pré-intégré à GitHub; en fait, il existe plusieurs outils pour cela.



# CLOUD ET NAS OÙ STOCKER LES DONNÉES DE MON ENTREPRISE ?

---

# CLOUD x NAS

NAS ou Cloud, deux solutions différentes pour le stockage des données en ligne de votre entreprise.

Le NAS (Network Attached Storage) est un serveur de fichiers indépendant connecté au réseau de l'entreprise. Il prend la forme d'un boîtier contenant le disques durs, la carte mère et le système logiciel. Il sert au stockage et à la sauvegarde des données, qui restent alors en interne, dans les locaux de l'entreprise.

Le Cloud, quant à lui, permet à l'entreprise d'externaliser le stockage de ses données, via un prestataire. C'est alors lui qui est en charge de la maintenance des serveurs de stockage. Si l'entreprise opte pour une solution Cloud en mode SaaS, le prestataire gère également l'application. L'entreprise n'a plus qu'à souscrire à un abonnement pour accéder au service, et ne gère plus ni l'installation ni la maintenance de la solution de stockage de données.

# CLOUD x NAS : LE COÛT

Opter pour un NAS signifie des coûts initiaux élevés pour l'achat du matériel. A ceux-ci s'ajoutent les frais liés à l'installation, au paramétrage, et à la maintenance sur le long terme. En effet, un NAS suppose l'intervention d'un informaticien pour l'administration et la mise à jour du système.

Les solutions Cloud en mode SaaS, quant à elles, vous permettent d'optimiser le coût du stockage de vos données, puisque vous pouvez bénéficier de prestations sur-mesure pour ne payer que ce que vous consommez : vous n'achetez pas par exemple un NAS de 2 To alors que vous avez 500 Go de données à stocker. Vous pouvez ajuster l'espace de stockage lorsque vous le souhaitez, vous jouissez ainsi d'une plus grande flexibilité. De plus, vous n'avez qu'à souscrire à un abonnement pour profiter d'une solution clé en main, prête à l'emploi, qui ne demande aucune installation, et dont les mises à jours sont déployées automatiquement par votre prestataire. Oubliez toutefois les solutions grands publics, bien que gratuites elles n'offrent pas assez de garanties de sécurité pour une utilisation professionnelle.

# CLOUD x NAS : L'ACCESSIBILITÉ AUX DONNÉES

Si le NAS convient très bien au stockage de données, qu'en est-il de l'accès distant ?

Les solutions SaaS de Cloud proposent en général des applications mobiles gratuites pour accéder à vos fichiers sur smartphones et tablettes, et si certains fabricants de NAS en disposent aussi, il est néanmoins beaucoup plus compliqué d'accéder à ses documents en mobilité.

De la même manière, si le fabricant de NAS ne propose pas de synchronisation, vous serez entièrement dépendant d'une bonne connexion Internet pour accéder à vos documents lors de déplacements.

# AVANTAGES DU CLOUD POUR ENREGISTRER LES DONNÉES COURANTES

Outre leur PC lorsqu'ils sont au bureau, certains collaborateurs utilisent un notebook, une tablette ou un smartphone pour accéder aux données de chiffre d'affaires d'un tableau Excel lorsqu'ils sont en déplacement.

Ils apprécient alors le grand avantage du stockage sur le cloud pour leur travail quotidien: les documents sont accessibles partout, à condition d'être connecté à Internet.

Et ce, sans avoir à établir une connexion externe potentiellement dangereuse au réseau de l'entreprise, comme le nécessiterait un NAS.

# AVANTAGES DU CLOUD POUR ENREGISTRER LES DONNÉES COURANTES

La flexibilité est l'un des points forts du stockage sur le cloud pour le quotidien professionnel:

- Pour modifier l'espace mémoire et le nombre d'utilisateurs, il suffit d'opter pour un abonnement de niveau supérieur ou inférieur.
- Tous les membres de l'équipe ont accès aux documents Office pour les modifier. Cela évite d'avoir plusieurs versions d'un même document en circulation – oubliez les noms de fichier de type «annee\_final\_v2-misajour.xlsx»!
- Les boîtes e-mail sont moins chargées car, au lieu d'échanger des pièces jointes, les collaborateurs envoient des e-mails contenant simplement les liens d'accès aux documents.

# AVANTAGES DU CLOUD POUR ENREGISTRER LES DONNÉES COURANTES

Les PME bénéficient de la protection de leurs données, que le fournisseur de services cloud assure en vertu des dernières normes de sécurité. Le stockage en ligne présente des avantages en termes de protection et de sécurité des données:

- Lorsqu'un document a été supprimé par erreur, il est possible de le restaurer dans un intervalle de temps défini.
- En cas d'infection par un ransomware, l'entreprise peut restaurer la dernière version de ses documents avant blocage.

Mettre en œuvre les mêmes standards de sécurité avec un NAS supposerait d'investir du temps dans la maintenance du dispositif de stockage en réseau et de développer le savoir-faire approprié. Ces aspects sont pour ainsi dire inclus dans l'offre de services cloud.

# AVANTAGES DU NAS POUR ENREGISTRER LES DONNÉES COURANTES

La rapidité est un point fort du NAS par rapport au cloud. En règle générale, l'accès aux fichiers volumineux est plus rapide sur le réseau local (du moins via le câblage Ethernet) qu'à travers la connexion Internet. Toutefois, l'argument n'est guère significatif en ce qui concerne les documents Office usuels, et les bandes passantes toujours plus généreuses relativisent cet avantage.

Les entreprises qui disposent d'un serveur local trouveront probablement un intérêt à utiliser le NAS pour stocker les données de leurs applications métier de type ERP et CRM. Mais il convient d'étudier si une solution purement basée sur le cloud ne constituerait pas une alternative plus fiable en termes de sécurité et de disponibilité. Si cela représente une charge trop importante ou si les applications de l'entreprise ne sont pas compatibles avec le cloud, alors le recours au NAS est justifié pour stocker les données du serveur.

# AVANTAGES DU NAS POUR ENREGISTRER LES DONNÉES COURANTES

La rapidité est un point fort du NAS par rapport au cloud. En règle générale, l'accès aux fichiers volumineux est plus rapide sur le réseau local (du moins via le câblage Ethernet) qu'à travers la connexion Internet. Toutefois, l'argument n'est guère significatif en ce qui concerne les documents Office usuels, et les bandes passantes toujours plus généreuses relativisent cet avantage.

Les entreprises qui disposent d'un serveur local trouveront probablement un intérêt à utiliser le NAS pour stocker les données de leurs applications métier de type ERP et CRM. Mais il convient d'étudier si une solution purement basée sur le cloud ne constituerait pas une alternative plus fiable en termes de sécurité et de disponibilité. Si cela représente une charge trop importante ou si les applications de l'entreprise ne sont pas compatibles avec le cloud, alors le recours au NAS est justifié pour stocker les données du serveur.

# LES INDICATEURS POINTENT VERS LE CLOUD

Sans surprise, les faits se confirment depuis plusieurs années: les avantages du cloud sont majoritaires tant pour le stockage de documents de travail que pour la sauvegarde de données/le backup.

Flexibilité, sécurité et accès collaboratif sont les principaux points forts du cloud, auxquels s'ajoutent la transparence des coûts.

# TECHNIQUE DE SAUVEGARDE DE DONNÉES : LE CRYPTAGE

---

# LE CRYPTAGE DES DONNÉES

Par principe, une sauvegarde externalisée (aussi appelée télésauvegarde) transmet des données informatiques d'un poste local vers un serveur distant.

C'est lors de cette étape du transfert que le cryptage est indispensable. Les données doivent être chiffrées avant l'envoi pour que personne ne puisse les utiliser.

Si un pirate s'introduit dans votre système d'informations et capte les données en cours de transfert, il se trouvera dans l'incapacité de les interpréter.

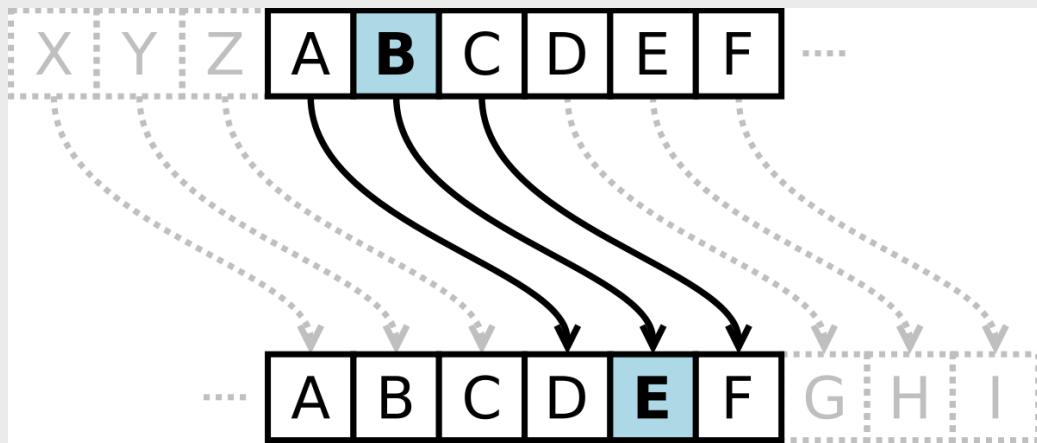
# QU'EST-CE QUE L'ON APPELLE CRYPTAGE ?

Le cryptage consiste à rendre un message uniquement lisible et compréhensible aux personnes possédant la clé de chiffrement.

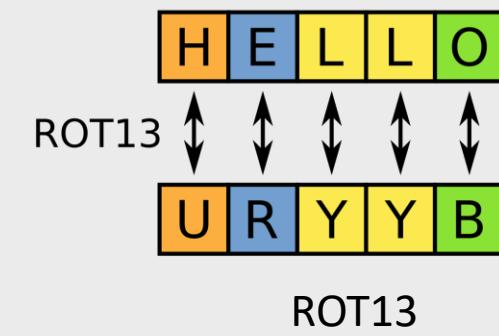
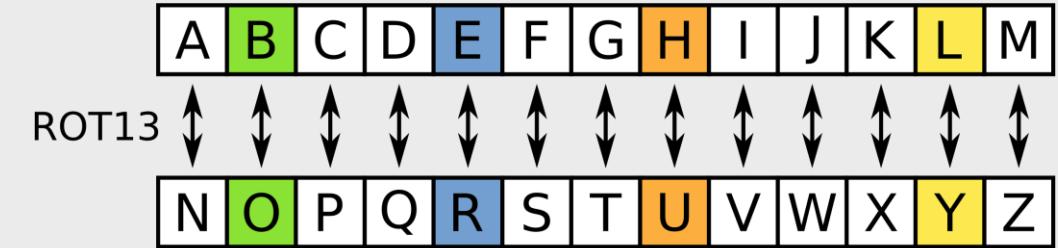
Si l'on voulait simplifier au maximum cette définition, nous pourrions dire que cette fameuse clé de chiffrement fonctionne comme un mot de passe.

Les données cryptées ne sont pas cachées ou enfermées dans une sorte d'archive protégée par mot de passe. Tout au contraire, les informations chiffrées sont parfaitement lisibles en apparence. À cette nuance près qu'elles sont rendues incompréhensibles par cette fameuse clé servant à la fois au chiffrement et au déchiffrement.

# EXEMPLES SIMPLES

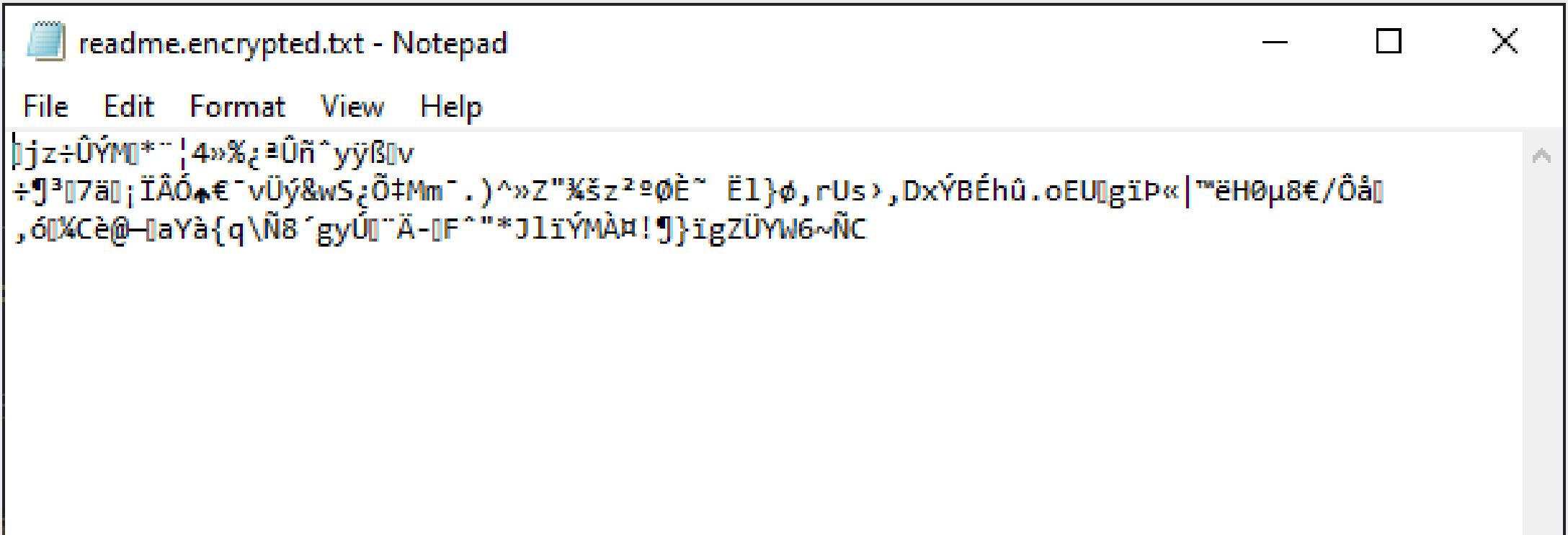


Chiffrement par décalage



ROT13

# EXEMPLES MESSAGE CRYPTÉ



# CRYPTAGE

De cette manière, une information chiffrée devient illisible lors d'une transmission.

Cette technique est valable pour tout type d'information, qu'il s'agisse d'une lettre manuscrite ou d'un fichier informatique.

De tout temps, le cryptage servit à des fins stratégiques et militaires. L'on pourrait d'ailleurs rapprocher le cryptage et l'invention de l'informatique en évoquant Allan Turing. Ce scientifique britannique est l'inventeur du premier computer en 1936 et décrypteur du code Enigma utilisé par les nazis lors de la Seconde Guerre mondiale.

# CRYPTAGE

Pour finir sur le principe du cryptage, il est important de mentionner qu'il existe plusieurs types de protocoles de chiffrement.

Ces protocoles rendent plus ou moins facile le décryptage des informations par un pirate. C'est pourquoi les systèmes de chiffrage deviennent de plus en plus complexe.

L'on parle d'ailleurs aujourd'hui de recherches sur un système de cryptographie quantique dont la finalité serait d'établir un protocole de chiffrement totalement inviolable.

# PROTECTION DES DONNÉES AVEC LE SYSTÈME EFS (ENCRYPTING FILE SYSTEM)

L'une des solutions pour réduire les risques de vol de données et améliorer leur sécurité consiste à crypter les fichiers sensibles avec le système EFS (Encrypting File System).

Le système EFS est une technologie Microsoft permettant de crypter les données et de contrôler qui peut les décrypter ou les récupérer. Lorsque des fichiers sont cryptés, les données utilisateur ne peuvent pas être lues - même par un pirate disposant d'un accès physique au système de stockage de l'ordinateur.

Pour utiliser EFS, il est nécessaire de disposer de certificats, qui sont des documents numériques permettant à leur titulaire de crypter et décrypter les données. Il est également nécessaire de disposer d'autorisations pour modifier les fichiers.



MERCI!

MAIL