**COMMENT UTILISE GIT ET GITHUB POUR LES DEBUTANTS.**

**Au nom de Dieu le tout miséricorde, le très miséricorde.**

**Auteur :** **Kiima Samuel (Africanboy Kiima).**

Email : [fadhilkiima@gmail.com](mailto:fadhilkiima@gmail.com).

Twitter : @AfricanboyKiima.

**Introduction.**

**C’est quoi git ?**

Git est un **système de control de version distribuée** (en anglais **DISTRIBUTED VERSION CONTROL SYSTEM**) qui permet un ou plusieurs personnes de travailler sur le même projet. Il est très utilisé par beaucoup des gens pour avoir une bonne suite de leurs projets au moment qu’ils développent leurs applications, chacun peut voir les changements faites par l’autre donc ayant l’histoire de leur projet quand il la développe. Dans notre 21eme siècle, il est obligatoire que les informaticiens apprennent comment l’utiliser. Git a été commencé par **Linus Torvalds** qui l’utilisaient pour maintenir, avec, ses autres collègues, le développement du noyau linux.

Git est un outil essentiel dans nos carriers tant qu’informaticiens, combinée avec les plateformes d’hébergement de projet (Code hosting plateformes comme GitHub), il est utilisé par les meilleures compagnies de la technologie comme :

**Companies Using Git (Compagnies utilisant git et GitHub)**

* Google.
* Facebook.
* Microsoft.
* Twitter.
* LinkedIn.
* Netflix.
* Perl.
* PostgreSQL

**NB : Connaitre git et GitHub est permis les prérequis dans beaucoup des compagnies.**

**C’est aussi tres important de savoir utiliser la ligne de commande beaucoup.**

**GitHub.**

Créé en 2007, GitHub est la plateforme d’hébergement de nos projets **(code bases/projects)** en anglais, qui nous permet aussi de collaborer avec d’autres personnes sur le même projet. GitHub a été acheté récemment (2018) par **Microsoft** est-il fait un bon boulot en créant d’autres fonctionnalité comme nous verrons dans cette tutoriel. GitHub n’est pas le seul hébergeur de projets, il existe d’autres comme GitLab, Bitbucket, AWS code commit et beaucoup d’autres mais le plus utilisé et connu est GitHub. Sur GitHub il y a beaucoup des projets dont vous pouvez contribuer, permis beaucoup c’est le SGBD « open source » PostgreSQL, Node.js et d’autres.

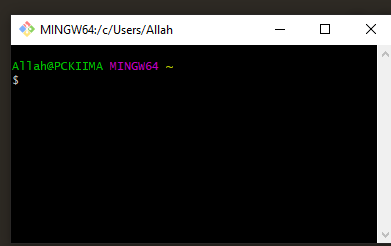
Pendant le lock down(confinement) de 2020 suite à la corona virus, beaucoup des ingénieurs informaticiens ne travaillaient plus dans leurs bureaux y compris ceux de Google, Microsoft etc. Le seul moyen qu’ils ont été capable de travailler ensemble était en utilisant cette plateforme qui veut dire que quelqu’un peut travailler pour une compagnie qui est aussi très loin de chez lui. Les anglophones appellent ça **« working remotely »** qui veut dire travaillant à distance

**NB : Pour votre information, tu ne dois pas être un expert pour contribuer aux projets open source sur GitHub, quelqu’un peut contribuer à la documentation ou à changer une balise etc…**

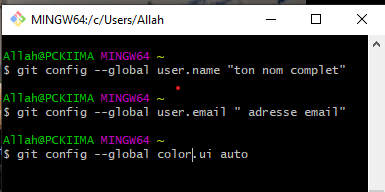
Pour utiliser git, pour les gens de Windows, il faut avoir un outil/logiciel installé appelée **gitbash** vous pouvez la télécharger de cette lien <https://gitforwindows.org>/ ,pour les gens de mac vous devez installer **Homebrew** de cette lien <https://brew.sh/>.

Pour les illustrations, nous allons utiliser **gitbash** mais si vous utilisée un autre ne vous inquiétez pas, on aura les mêmes résultats.

Voici une photo de **gitbash.**



Bon, on doit configurer notre logiciel en précisant le nom et l’adresse email ainsi que le couler de notre outil, juste dans le logiciel, tapez **git config --global user.name et git config --global user.email ainsi que git config –global color.ui auto**

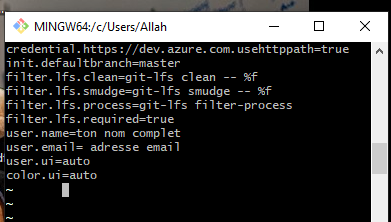
****

**git config --global user.name** nous permet de configurer notre nom.

**git config --global user.email** nous permet de configurer notre email

**git config --global color.ui auto** nous permet d’avoir une interface de commande d’une belle couler donc c’est pour l’output du couleur.

Pour vérifier nos configurations, on tape **git config -l** ce qui donne :



J’ai beaucoup parler, il est temps d’apprendre ce qu’on doit apprendre.

Normalement quand on travaille sur un projet, on travaille avec un dossier contenant

des fichiers de toutes sortes. On travaille avec notre ordinateur qui a des dossiers et permis ces dossiers on peut y trouver nos fichiers sur laquelle on travaille comme dans le diagramme sous dessous :

Index.css

yes.py

Index.js



On suppose que ces fichiers sont dans un dossier, le problème avec ça et que si notre ordinateur a un problème, on risque de perdre nos projets mais comme on a GitHub, on a la possibilité de garder nos projets sur le nouage dont même si on a perdu nos projets, on peut les récupérer facilement. Un dossier est une place qu’on garde nos fichiers dans l’ordinateur.

Quand nous travaillons avec git, on a le « staging area », prenez-le comme un arrêt, et le « commit history », prenez-le comme un point qui est en lieu sûr. Pour arriver à travailler avec les autres ou soit même, on doit faire des «commits», les commits sont comme un travail qui t’a été donné et tu fais maintenant le dépôt de cette travail, donc c’est un travail finale que vous pensez peut être maintenant déposer, par exemple quand tu travail sur le «front end» et tu vois une caractéristique comme changement de couleur de bouton quand on fait un « hover » travail, la tu peux maintenant faire un commit comme cette particularité marche, ça veut dire qu’on fait normalement des commit quand on est assuré qu’on a vraiment finit un travail donné. Mais, avant d’arriver à faire un commit, on doit passer par le « staging area », un diagramme peut nous faciliter la tâche :

COMMIT HISTORY (Prenez le comme un lieu sur)

Index.css

yes.py

Index.js

git commit



git add

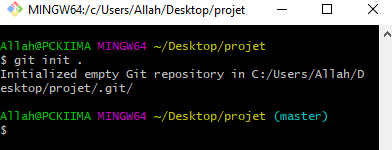
Comme on a dit, pour arriver à faire un commit, on passe par le staging area et pour mettre nos fichiers dans le staging area on utilise la commande **git add** suivi du nom du fichier, exemple **(git add yes.py),** cette commande mettra le fichier dans le staging area, au cas où on a plusieurs fichiers que nous voulons mettre dans le staging area au même moment, on tape (**git add .**). En plus, cette commande ajoute les fichiers de notre dossier récent vers le bas. Git bash se comporte comme la ligne de commande**(terminal),** ça veut dire qu’on peut créer des dossier et fichiers juste en utilisant la ligne de commande.

Nous allons créer nos fichiers et dossiers pour y les mettre, pour créer un dossier on utilise la commande **mkdir** qui veut dire créer un dossier en anglais suivi par le nom du dossier, je vais utiliser mon desktop**(bureau)** pour créer mon dossier. Je commence d’abord en changeant le dossier à Desktop en utilisant la commande **cd Desktop,** après je crée un dossier avec mkdir suivi du nom du dossier que je veux créer.

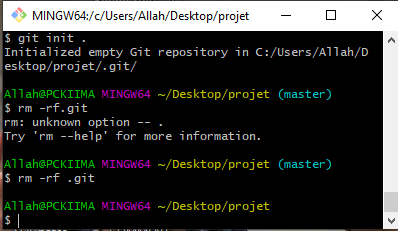


Après avoir créé le dossier, on change encore ça à notre dossier que nous avons créé, comme vu dans la photo, maintenant on se trouve dans notre dossier « projet » qui contiendra toutes nos sous dossiers ainsi que nos fichiers qu’on va utiliser.

Quand on travaille avec git, on appelle plus ça un dossier mais un « repository » normalement appeler **repo,** un repo est une collection de fichiers de diffèrent version d’un projet. On doit maintenant initialiser cette dossier à un repo, et pour initialiser un dossier à un repo on tape la commande **git init .**

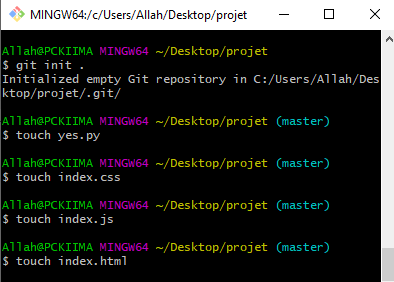


Cette commande crée un repo donc on peut maintenant commencer à utiliser les commandes git sur ce repo, mais en regardant notre git bash, on voit des phrases en anglais qui nous confirme qu’on vient d’initialiser un repo vide qui signifie simplement qu’on a réussi à initialiser notre repo, en regardant encore, on voit aussi entre parenthèses un mot en écriture bleu, on appelle ça **master branch,** les branches sont un concept qu’on verra plus tard mais qui est indispensable pour utiliser git. On peut au même moment retourner notre repo à un dossier en tapant **rm –rf .git**

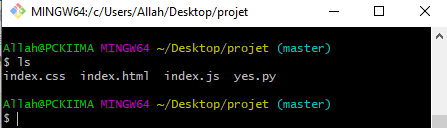


Assurer vous qu’il y a un espace entre -rf et .git, retapez la commande (**git init . ).**

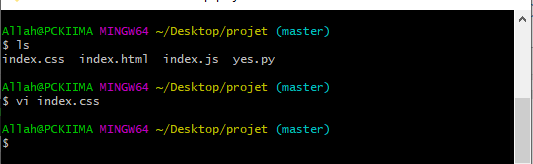
On va créer maintenant l’un des fichiers en utilisant la commande **touch** suivi du nom du fichier et son extension par exemple **touch yes.py,** cette commande nous permet de créer des fichiers avec notre ligne de commande git bash, on fait ainsi pour toutes nos autres fichiers. J’ajoute un fichier index.html



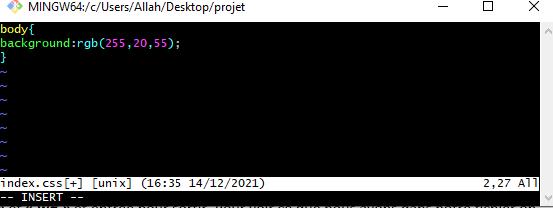
Pour voir la liste de toutes ces fichiers, on tape **ls** et la liste de toutes les fichiers vont apparaitre ce qui fait quatre fichiers de type différente.



La ligne de commande nous donne la capacité de modifier nos fichiers au cas où nous voulons le faire, git bash a un éditeur intégré appelée vim qui nous aide à accomplir cette tâche. On utilise la commande **vi** suivi du nom du fichier ainsi que son extension et en tapant entrée, on tape **i** qui veut dire insérer en anglais, là, on peut modifier nos fichiers ayant nos code source. Pour sortir de cette éditeur, on tape **ESC/Echappée** suivi de deux point **« : »** et **« wq »** et entrée pour sortir. Pour voir ce que nous avons dans notre fichier on tape **« cat »** suivi du nom de fichier comme dans l’image ci-dessous.



En cliquant entrée, on se trouve dans une autre fenêtre, là, on peut tapez « **i** » qui va nous permettre d’écrire dedans comme dans la photo suivante.

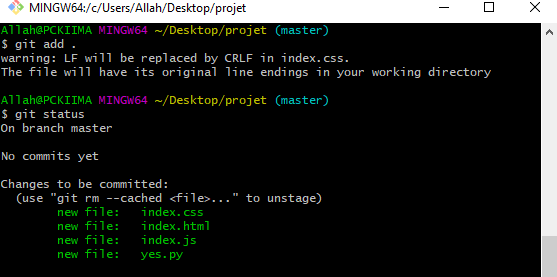


**NOTE : N’oublie pas de tapez « i » pour pouvoir écrire dans cette éditeur.**

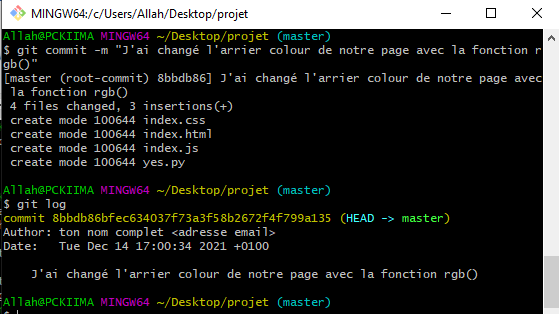
**Pour sortir de cette éditeur, tapez « échappée », deux point « : » et « wq » pour sortir et retourner à gitbash.**

**Quelques pushes.**

Pour faire de pushes on doit faire des commits. Pour faire de commits on doit passer par le staging area, on y va.

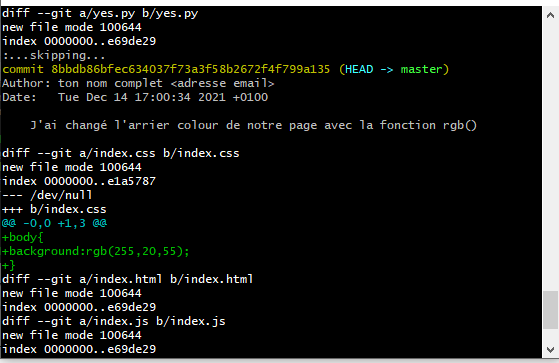


On vient d’ajouter nos fichiers à la staging area. Donc on peut faire des commits.



Comme vu, on a fait un commit en tapant la commande **« git commit –m »** suivi du message de mon commit, cette commit est un message qu’on écrit pour dire aux autres ce qui a été fait donc les changements fait à notre application ou votre contribution au projet en cours. Pour mon cas, j’ai utilisé qu’une fichier css, mais on peut faire des modifications sur beaucoup des fichiers. En plus, si vous voulez faire un changement de votre dernier message de commit on tape la commande **« git commit --amend –m »** suivi du message qui remplacera votre message pour la corriger.

En regardant notre photo, on trouve la commande **« git log »** et un mélange des chiffres et lettres s’est apparu, on appelle ça un **« hash »** en tapant **« git show »** suivi du **« hash »** on va voir les différences entre nos changements anciens et nos changements courant. Voici une photo.



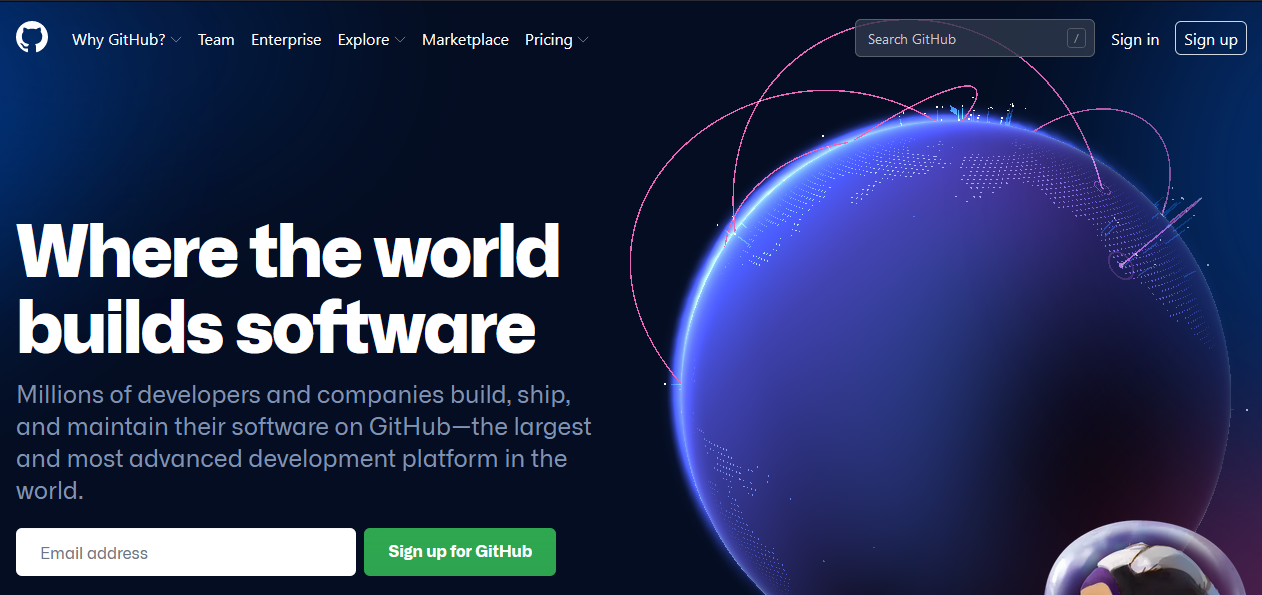
Les lignes en vert montrent ce que nous avons ajouté. Donc, git show suivi du « hash » nous aide à voir toutes les différences entre les fichiers en terme des additions et soustractions.

**GITHUB.**

**Faire des pushes**

Normalement après avoir faire nos commits, on doit faire des pushes, les pushes sont le moyen d’envoyer nos changements/contributions à GitHub. Avec notre commit, on va essayer de faire un push en tapant **« git push –u origin main ».**

Pour utiliser GitHub, il faut avoir un compte de cette plateforme. Voici le lien vers leur site <https://github.com> .

****

**“Where the world builds software.”** veut simplement dire ou le monde construisent des applications/logiciels.

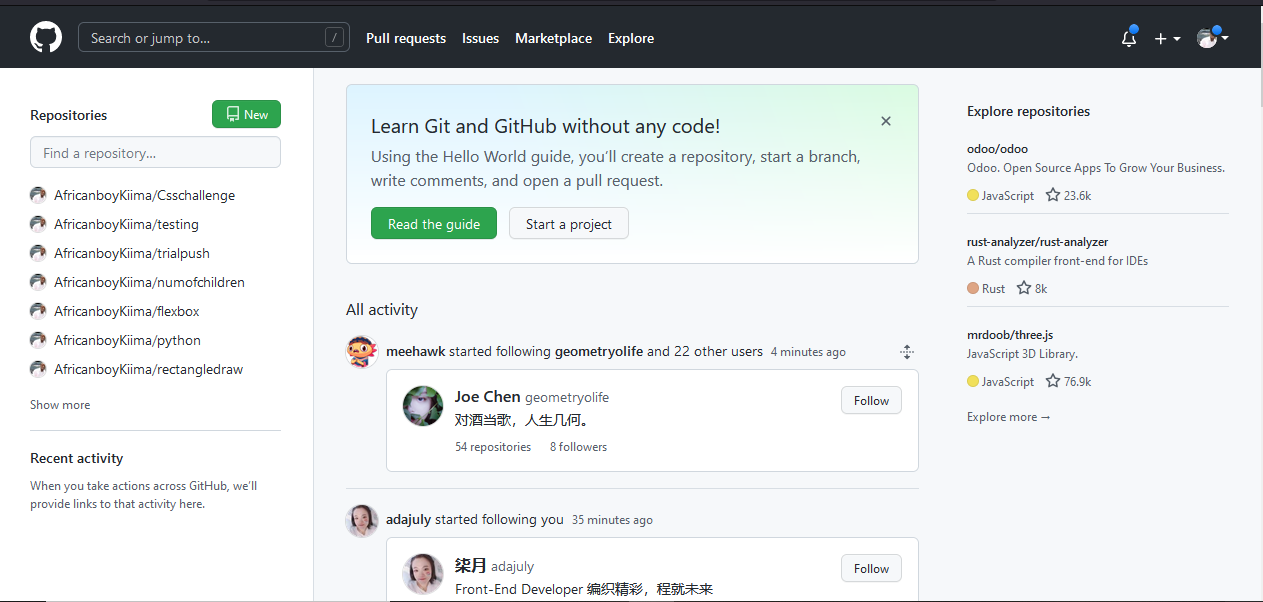
GitHub est le plateforme d’hébergement de nos projets, il est très utiliser par presque toutes les organisations mondiales et il est le plus populaire, mais il existe d’autres plateformes comme GitLab, Bitbucket, Amazon code commit et d’autres, on peut collaborer sur cette plateforme avec d’autres ingénieurs quel que soit l’endroit.

Pour faire des pushes, donc, d’envoyer vos changements(commits) sur GitHub, on doit créer un protocole SSH (Secure Shell). Par protocole, on signifie une règle de communication. Donc on doit créer une communication entre notre gitbash et GitHub. C’est avec GitHub qu’on doit générer des clés SSH, une clé privée et publique.

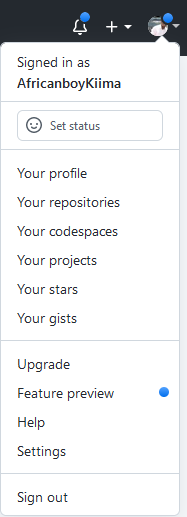
**SET UP DE CLE SSH.**

On doit maintenant générer des clés SSH pour pouvoir faire des pushes vers GitHub sinon on ne saura pas communiquer avec. N’oublie pas, on doit avoir un compte GitHub, moi j’en ai.

**NB : Fait ton recherche sur YOUTUBE comment on ouvre un compte sur GitHub.**

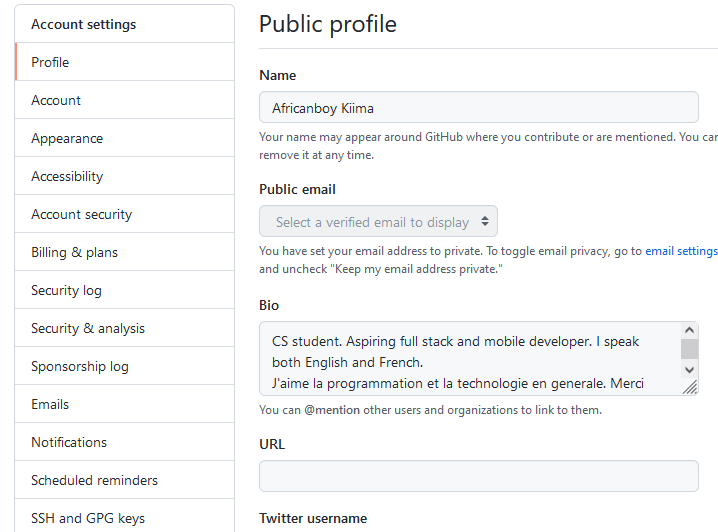
Voici comment une page de GitHub se ressemble normalement. 

Vous trouverez presque les memes choses sur la plus part de comptes GitHub. Au droit, vous trouvez une petite photo avec un bouton de liste, voici la photo :

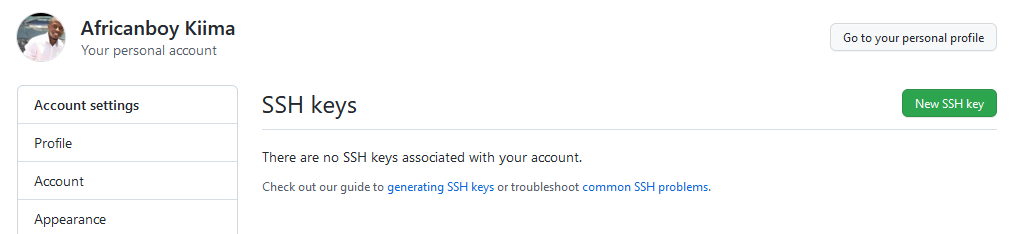


Tapez sur Settings.

Une page s’ouvre encore.



Clicker sur SSH and GPG keys.

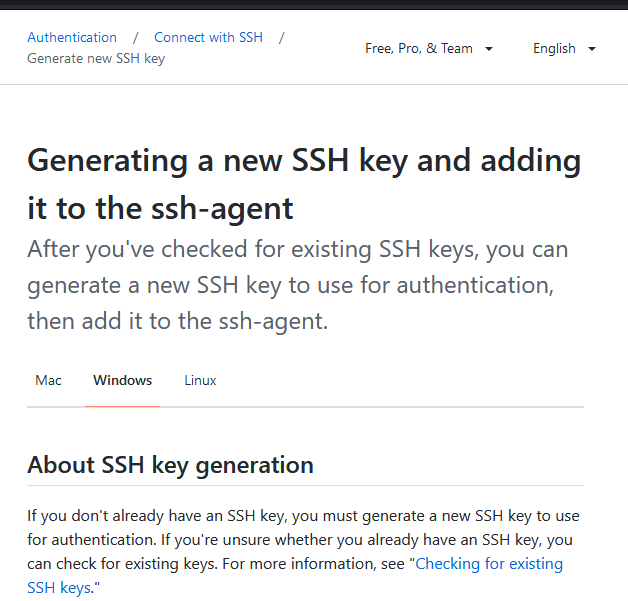


On nous dit qu’il y a aucun clé SSH associé avec notre compte donc on doit générer une clé SSH. Cliquer sur generating SSH Keys.



Une autre page s’ouvre, clicker maintenant sur Generating a new SSH key and adding it to the ssh-agent.

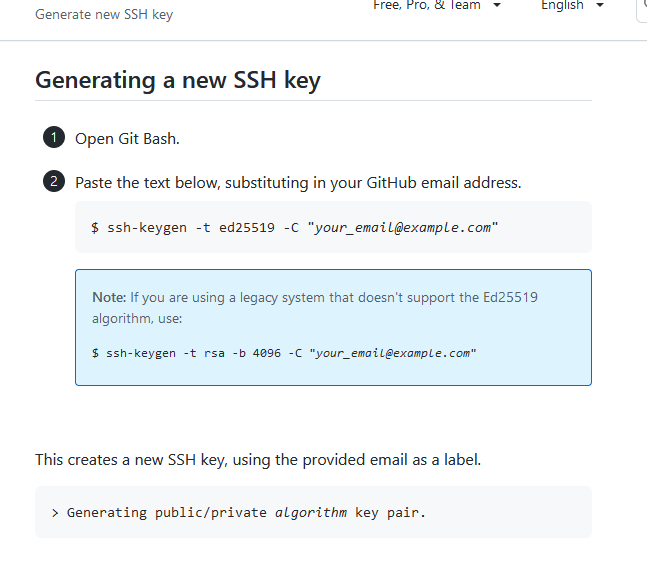
On se trouve encore sur une autre page sur lequel on vas choisir les commandes selon notre système d’exploitation (OS).

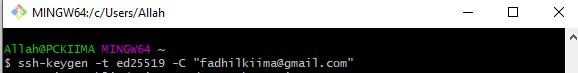


**NB : L’authéntication est la procédure de prouver si quelque chose est vrai.**

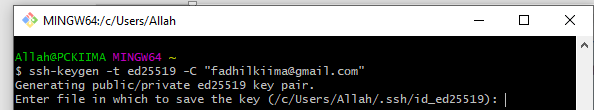
Ouvrez votre git bash.

Collez les commandes comme vu dans cette photo dans votre gitbash.



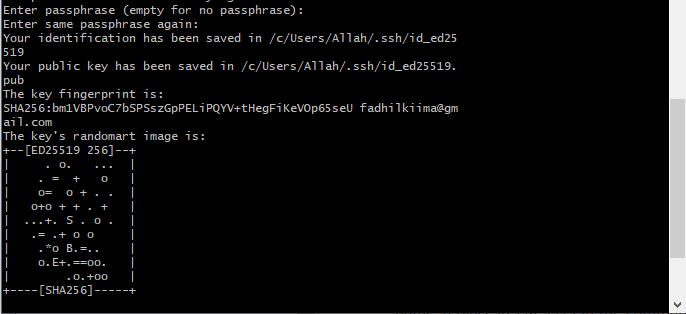


Il faut mettre votre adresse email moi je mets la mienne.



On nous dit d’entrée le fichier dans lequel on va sauvegarder notre clé, dans c’est cas accepté le fichier proposé par défaut. Appuyez sur Entrée/ENTER.

On nous dit d’entrée un mot-de-passe(**passphrase**), comme vu dans le fenêtre, c’est toujours important de mettre un mot de passe pour garder vos choses.



En faisant entrée votre passphrase, on vous redemande d’entrer le même passphrase/mot-de-passe encore **(Enter same passphrase again).** Fait comme on vous dit, en saisissant le mot de passe on vous informe sur l’endroit dont on a gardé votre identification et ainsi que votre clé publique.

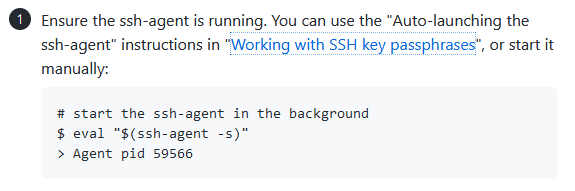
Quand on voit l’identification et la clé publique on trouve une différence entre le deux comme ci-dessous :



Le premier doit être garder dans le SSH agent **(c’est un programme qui contient la clé privée utilisé pour l’authéntication de la clé publique).**

L’authéntication veut dire la procédure de prouver si quelque chose est vraie.

Pour ajouter notre clé privée à la SSH agent, copier le code et coller l’en gitbash :

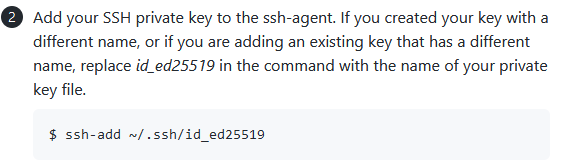


Je copie seulement la deuxième commande comme dans mon gitbash, cette commande fait que le SSH agent soit prêt pour recevoir la clé privée.



Il y a quelques mots avec des chiffres qui apparut, les numéros se génère automatiquement est sont différent donc là votre sera différent de la mienne.

On doit maintenant garder notre clé privée dans le SSH agent, pour faire cela on doit utiliser la commande ci-dessous :



**NB : Ces commandes sont données par GitHub donc ce n’est pas à nous de s’en occupé.**

**Nous pouvons vérifier notre clé privée ainsi que notre clé publique. Comment ? voyons comment :**

**Pour voir la clé prive on tape :**



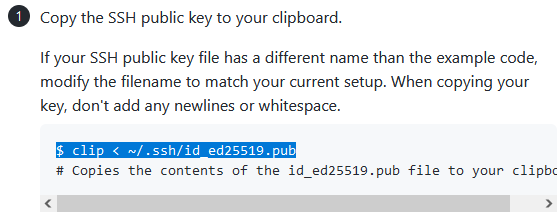
**Pour voir la clé publique on tape :**

****

Nous devons maintenant ajouter notre clé SSH à notre compte de GitHub.



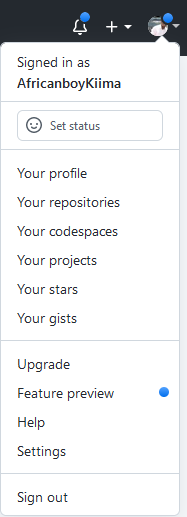
Cliquer sur Adding a new SSH key to your GitHub account.



Collez maintenant sur votre gitbash :

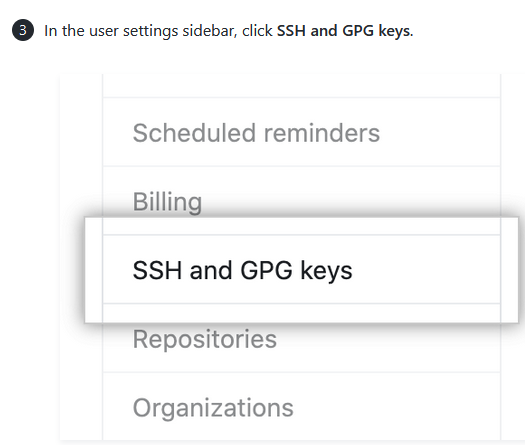


Sur n’importe quel page de votre compte GitHub, clicker/Appuyez sur votre photo de profile et clicker sur **settings.**

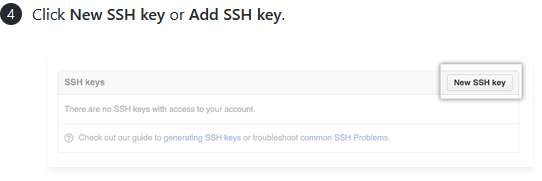


Settings

Dans le side-bar de Settings, appuyez sur **SSH and GPG keys :**

****

On se trouve dans une autre fenêtre, appuyez sur **New SSH key**, qui montre qu’on ajoute une nouvelle clé SSH.

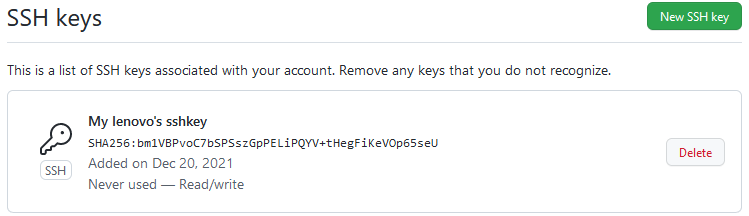
****

J’ajoute maintenant la clé publique.



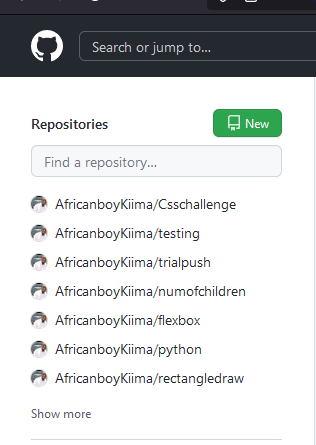
Après, dans le zone de texte de **Title,** je peux nommer mon clé selon mon choix par exemple selon le nom de ton machine, « Mon clé Lenovo », j’appuie sur **Add SSH key.**

Et voilà les résultats

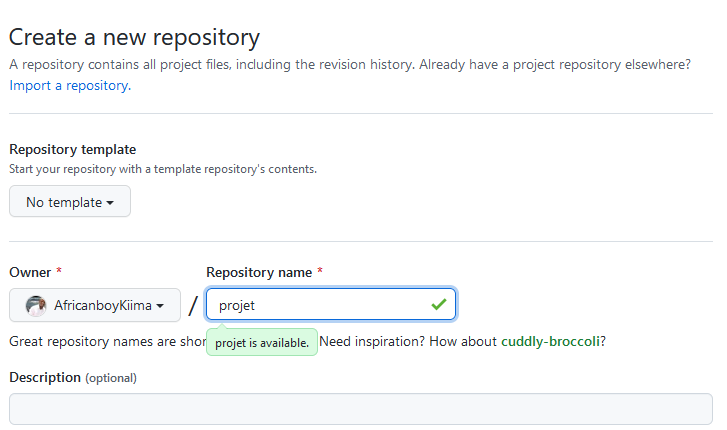


On a donc créé un protocole de communication entre gitbash et GitHub et on sera maintenant capable de faire des **« pushes »** ou d’envoyer nos changements vers GitHub. Essayons de faire un push vers GitHub.

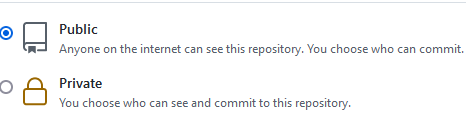
Pour faire un projet, on clique sur New dans notre compte GitHub comme ci-dessous :



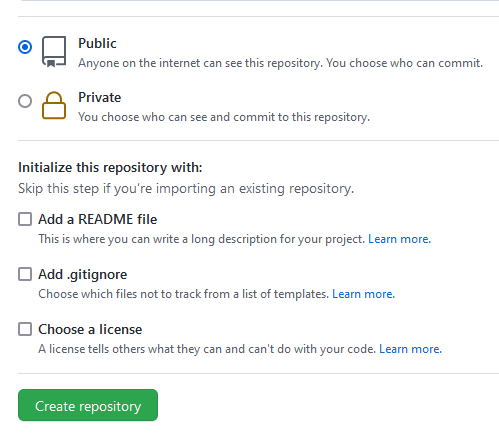
On se trouve sur une autre page et c’est ici qu’on va nommer notre projet, on nomme selon le nom qu’on avait nommé déjà quand on créait notre nouveau projet avec gitbash.



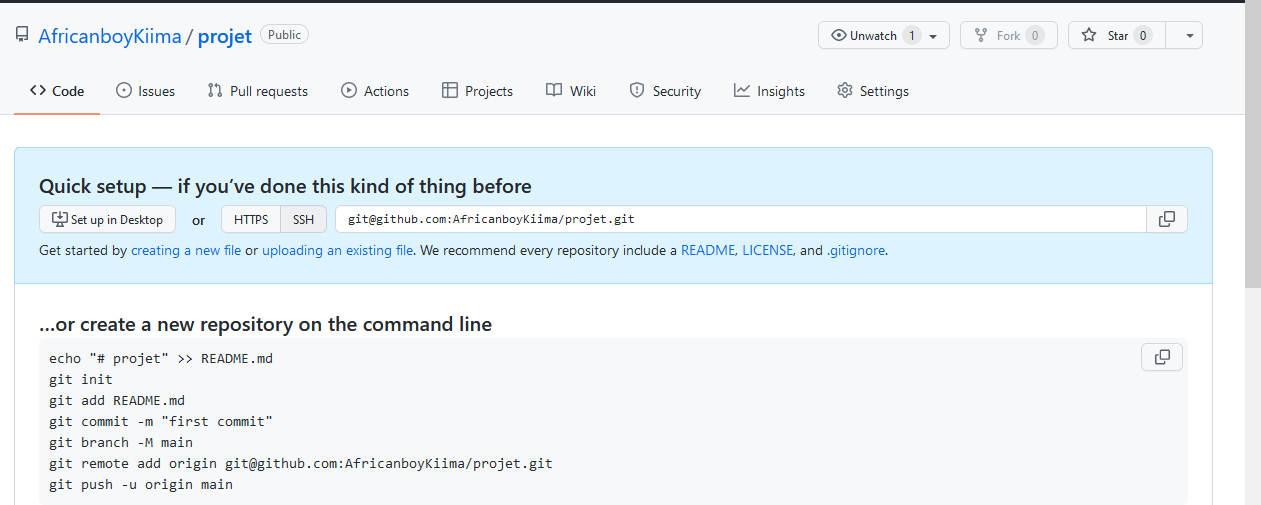
On peut ou ne pas ajouter une description de notre projet , ainsi, on peut aussi privatiser notre repo or le faire publique comme dans le choix donné :

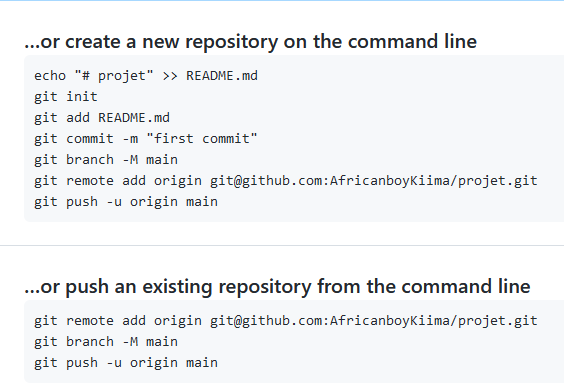


Ne touche à rien d’autre et appuyez sur **Create repository** comme dans le photo ci-dessous :



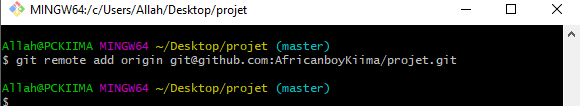
On se trouve sur une autre page avec beaucoup de commandes git comme ci-dessous :



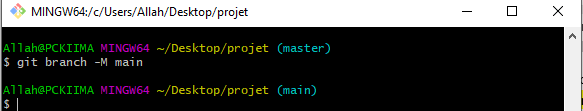


Ces commandes vont beaucoup nous aider pour que notre gitbash communique avec GitHub. Comme on avait déjà quelque commits, on va maintenant essayer d’envoyer nos changements ou notre code source vers GitHub car on a créé le protocole SSH.

On va commencer d’abord par décrire notre **« remote »,** remote est juste la manière que git dites « L’endroit ou plateformes que votre code source est garder/stocker » donc remote peut etre, GitHub, GitLab mais nous notre remote est GitHub etc…



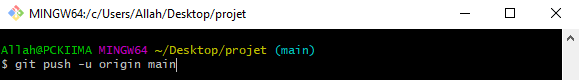
On change maintenant notre branche. Avec git, on a un concept qu’on appelle le branchement, cette branchement est un concept sur laquelle git se repose, c’est actuellement les branches qui fait git ce qu’il est. On verra plus tard ça. Microsoft, qui a acheté GitHub, a changé le nom de la branche principale de **« master »** à **« main »** donc on écrira :



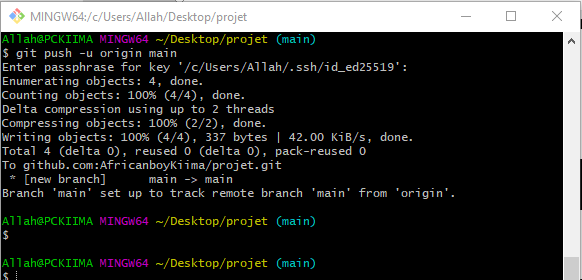
En regardant attentivement, on voit un changement de branches, ce qui était **« master »** est devenu **« main ».**

**Faire un push.**

Pour faire un push on écrit la commande :

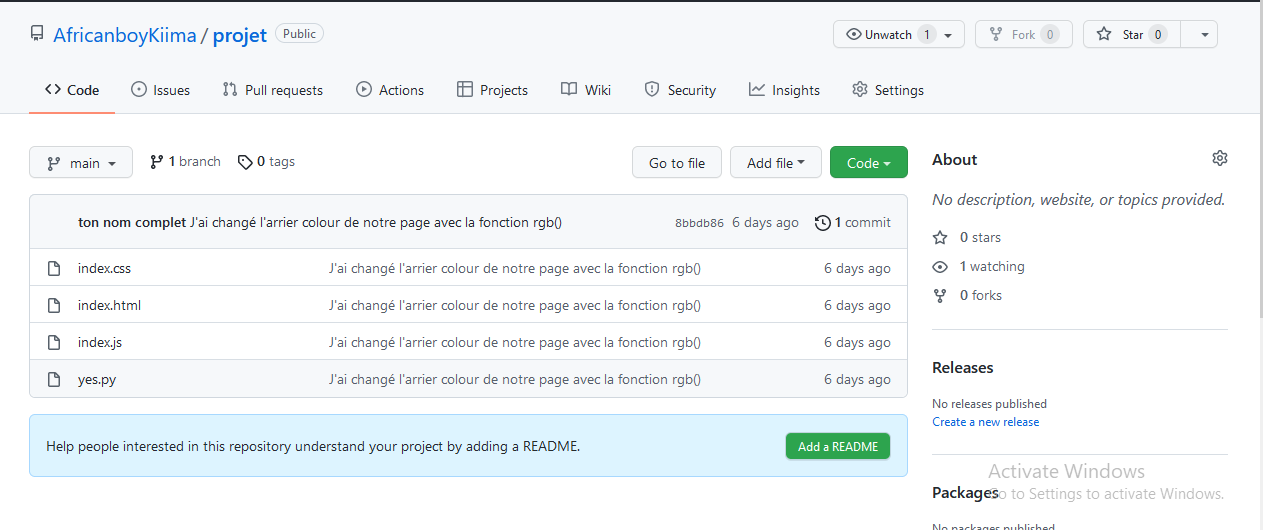


Voici les résultats :



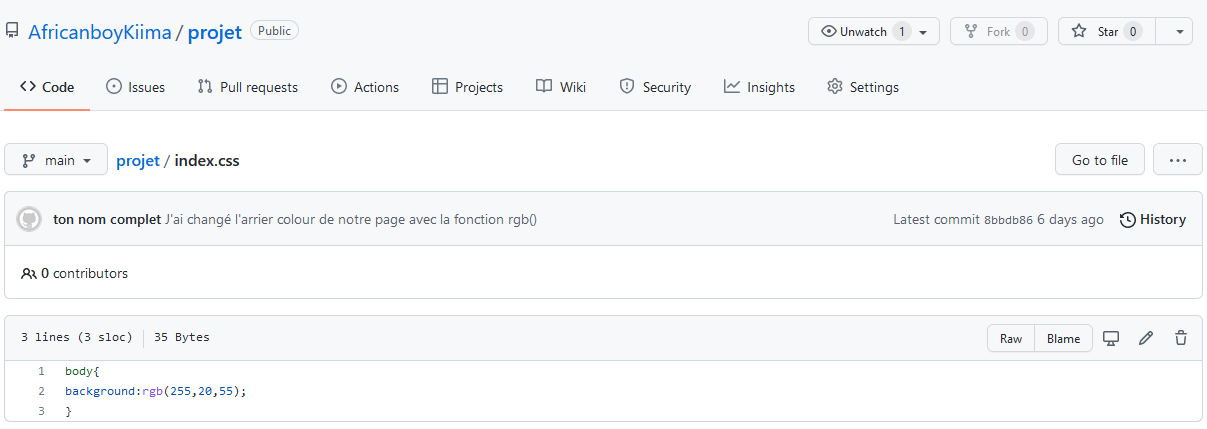
On nous demande d’insérer le passphrase et moi j’ai mis le mot de passe que j’avais mis comme mot de passe. En insérant mon mot-de-passe, il y a un succès de pushes et si nous parton à GitHub, nous trouverons dans notre repo le code source qu’on avait écrit quand on créait nos fichiers divers.

On click seulement sur le mot projet pour voir nos fichiers ainsi que nos commits.



Nous voyons notre message de commit que nous avons écrit quand-t-on envoyait à la commit history. Cette message de commit s’applique sur toutes les fichiers de notre repo sur GitHub. On voit aussi au droit le nombre de jours depuis qu’on a fait la dernier commit.

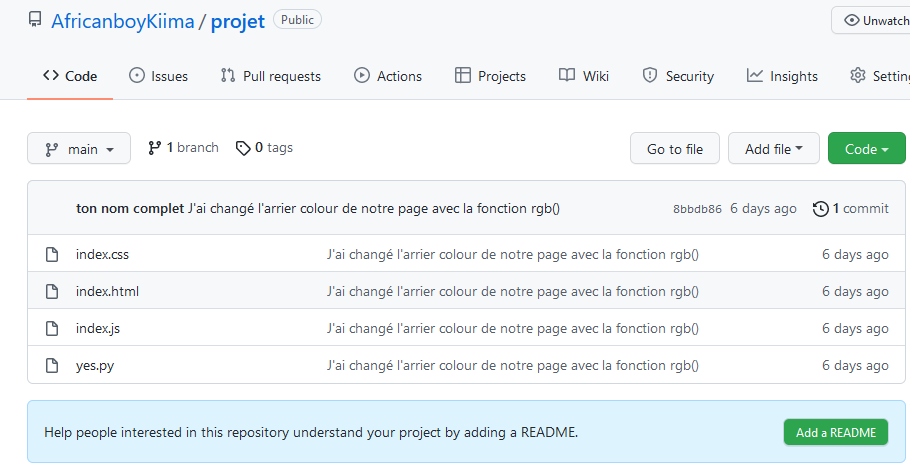
Si nous essayons d’entrer dans nos fichiers, nous verrons toutes le code source. Par exemple si on clique sur notre fichier index.css, on trouve le code source qu’on avait écrit :



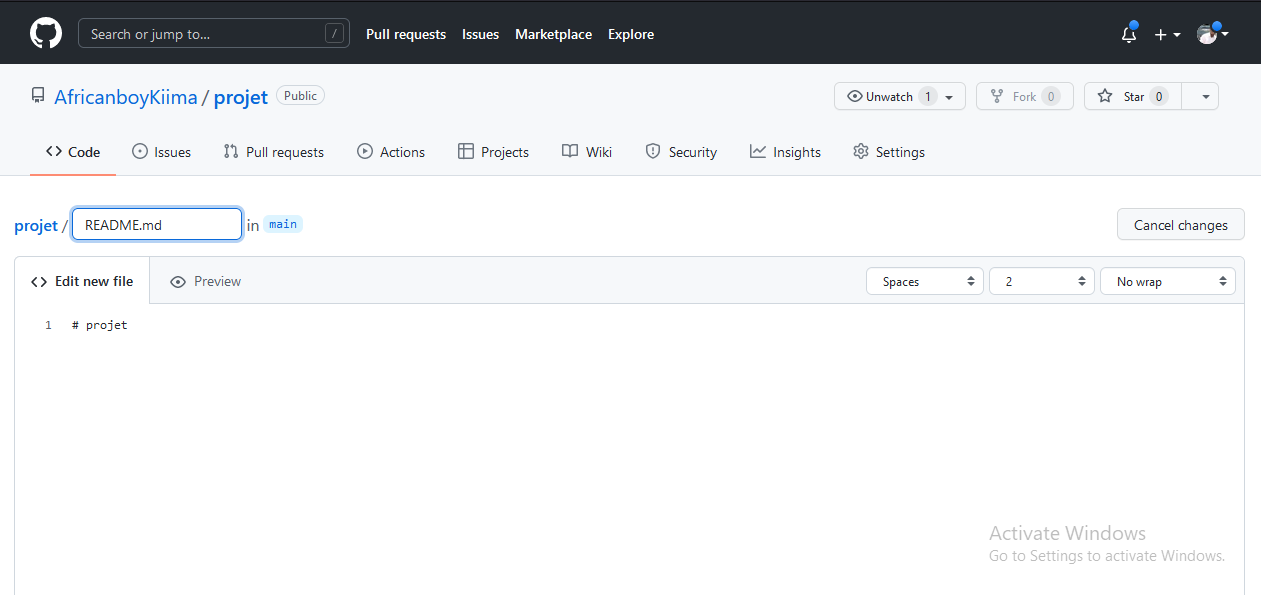
**Essayez vous-même les autres fichiers et amusez-vous.**

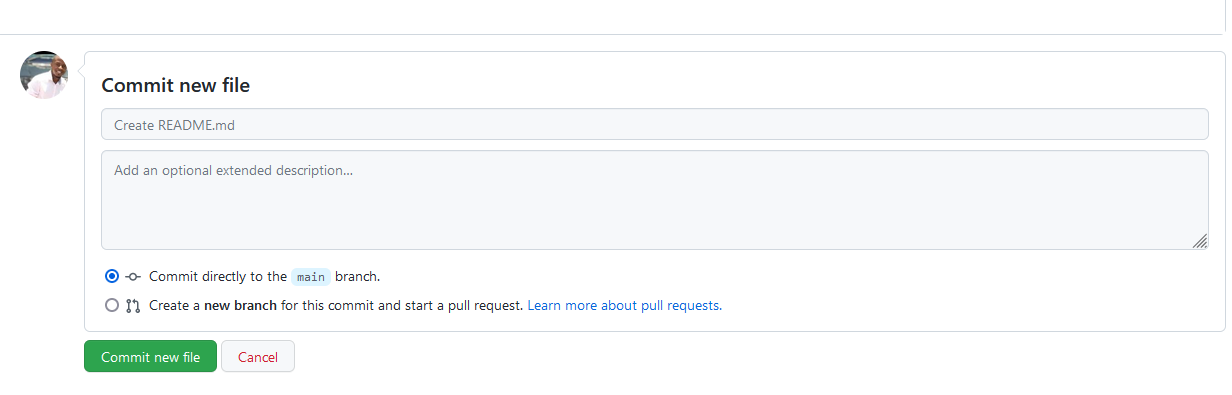
**README.**

C’est un fichier dans lequel on fait la documentation de notre projet, on peut mettre des choses de types différent comme des liens, check box, des petits vidéos, photos et diagramme expliquant notre projet aux autres. Nous allons ajouter des textes dans notre fichier **Readme.** Allons-y.

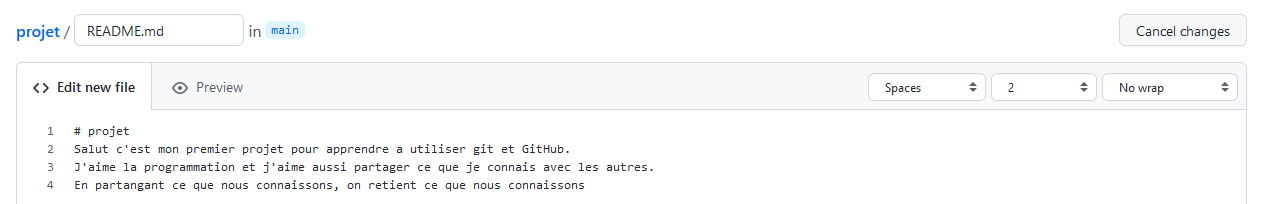


Cliquer sur Add a README, une autre page s’ouvre, c’est sur cette page que nous écrivons notre documentation et toutes sortes de choses que nous voulons mettre pour qu’on puisse expliquer or parler de notre projet sur GitHub.

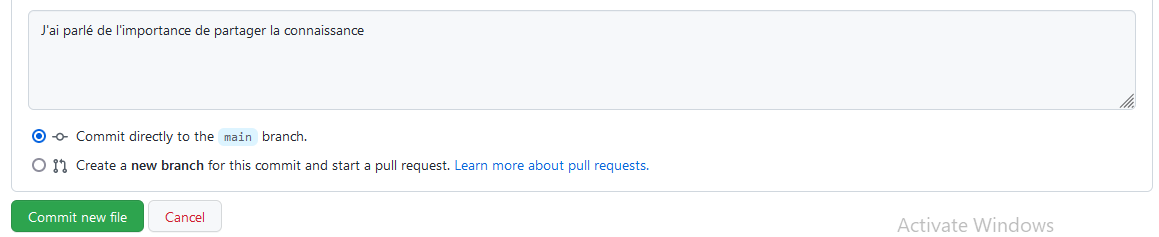




On peut maintenant écrire comme ci-dessous :

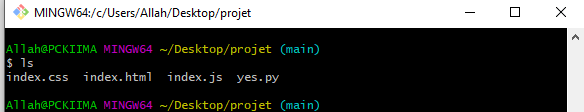


Il faut écrire aussi dans la zone de texte qui sera comme votre message de commit pour le README. Voici la zone de texte.

  
C’est dans cette zone de texte que nous écrivons notre commit comme j’ai dit de notre fichier README

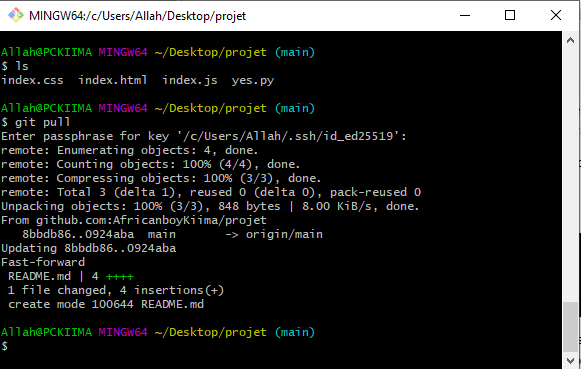
Faisons maintenant un commit de ce que nous avons écrit en cliquant sur **Commit new file** cette commit ira directement dans la branche principale donc le branche **« main »,** on voit que toutes les changements que nous avons fait sont ajout, le README a été ajouter sur notre liste de fichier ainsi que le message de commit qu’on avait écrit dans la zone de texte.

Allons un peu dans notre gitbash et vérifier si nous avons le fichier README en locale(gitbash).



En tapant ls, on voit qu’il n’y a pas notre fichier README, mais n’oublie pas que on doit l’avoir car comme développeur-collaborateur, on doit avoir toute le code source de la branche principale.

Imaginez maintenant que cette changement fait à la branche principale a été faite par un autre collaborateur et vous voulez avoir ces contributions, allons-y.



On nous a demandé de mettre le mot de passe et j’ai mis. Sur la onzième ligne après le git pull, on voit README.md | 4 ++++ qui veut dire le fichier ajouter est le README.md et il a quatre ligne de code qui ont été ajouter. On a réussi à faire un push et un pull.

**BRANCHEMENT (BRANCHING).**

Les branches sont un concept très important.

**DEFN**: Le branchement en git signifie la manière de travailler sur différentes versions d’un « repository / repo ». Il représente une ligne indépendante de développement.

Quand on travaillait avant je vous ai parlé de cette concept très importante, et nous avons vu la branche appeler **« main »** c’est lui qui est la branche définitive, donc on ne peut pas lui changer. On utilise les branches pour expérimenter et faire des éditions de la code source ou ajouter d’autre fonctionnalités avant d’envoyer ça à notre **code base(main).** Quand on crée une branche de la branche principale**(main)**, on est en train de créer une copie de la branche principale appeler « **main »** les anglophones appellent ça **snapshot donc une copier.**