**GIT**

Table des matières

[Chapitre 1 INTRODUCTION 2](#_Toc94210273)

[Chapitre 2 CREATION D’UN REPERTOIRE GIT DANS DOSSIER GIT ET GESTION DE SES FICHIERS : 2](#_Toc94210274)

[Chapitre 3 : LES BRANCHES : 5](#_Toc94210275)

# 

# INTRODUCTION

Git est un outil de référence (contrairement aux autres outils : SVN, CVS qui sont en voie de disparition sur le marché du High-tech) dont le noyau est développé par le patron de Linux (Linus Torvalds) dans les années 2000 afin de pouvoir faciliter le développement la gestion des projets informatiques (en équipe ou en solo) aux développeurs et aux entreprises qui les font travailler.

Il est organisé et discipliné, son organisation se manifeste en la présence de branche(s) locale(s) qu’on peut enregistrer sur un serveur (Git Hub) grâce à l’application client git Bash qui est une console. Il privilégie toujours l’intégrité des fichiers stockés dans le serveur, il répercute donc les erreurs humaines sur leurs fichiers stockés en locale.

Il est basé sur un système de gestion de l’état d’avancement à chaque moment (Il permet de stocker l’historique d’un projet)

Le fait qu’un fichier peut être utilisé par plusieurs développeurs en même temps, de travailler en même temps et en temps réel le rend un outil complètement distribué, et grâce à ses milliers de branches parallèles sur un seul projet.

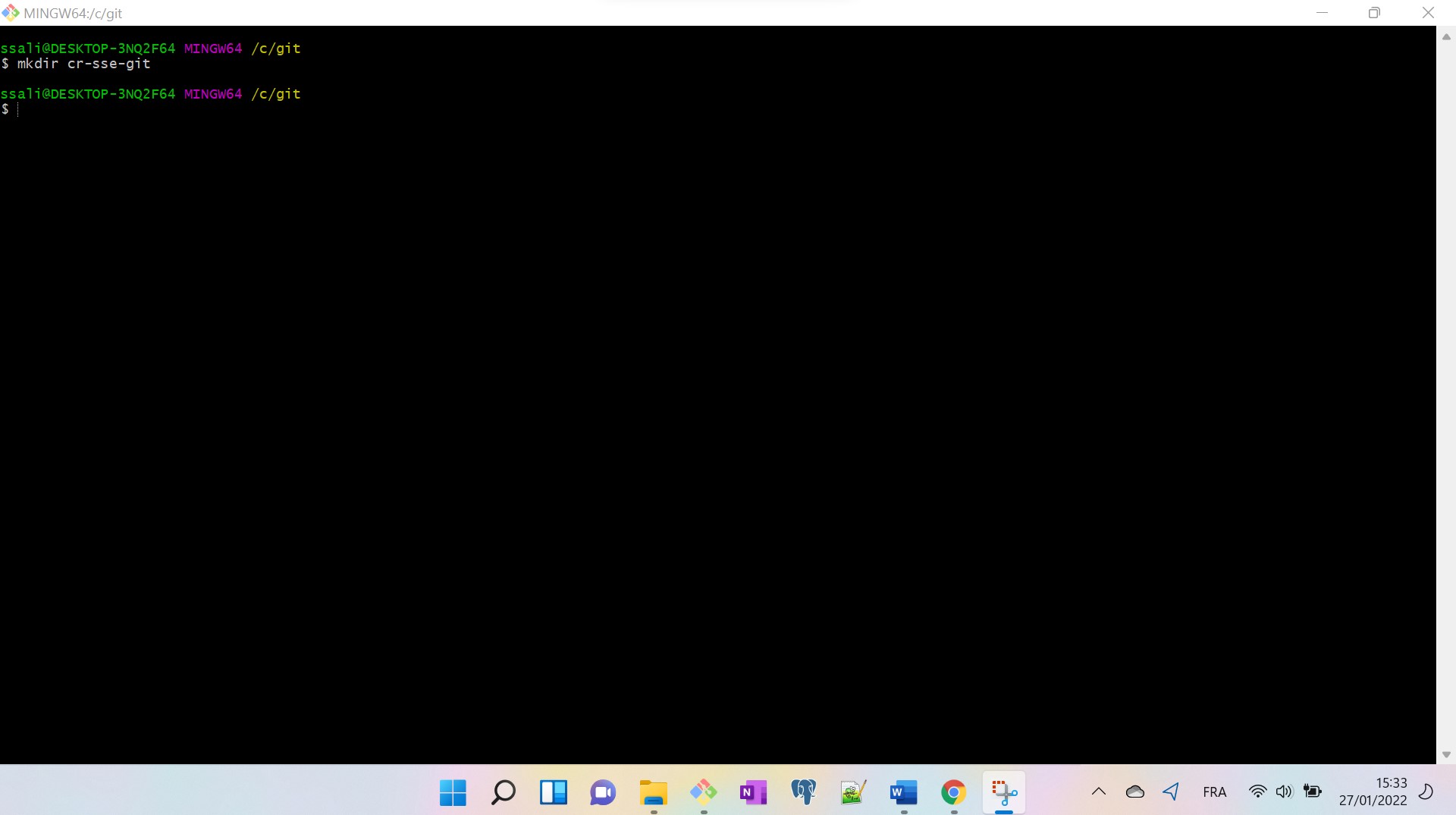
Il existe 4 états de fichiers dans Git :

1. Untracked Files : c’est les nouveaux fichiers qui ne sont pas suivis par git (la commande git add suivie de la commande git commit permet de les rendre des fichiers gérés par git)
2. Unmodified Files : c’est les fichiers qui existent déjà mais qui ne sont pas modifiés (sont ivisible sur la console).
3. Modified Files : c’est les fichiers qui existent déjà et qui ont subi des modifications. (la commande git add suivie de la commande git commit permet de les rendre des fichiers gérés par git)
4. Les staged Files : c’est les fichiers qui ont été ajoutés par la commande git add mais pas comittés sur une branche.

En outre la commandes git push permettent de faire enregistrer les fichiers ainsi leurs modifications sur le serveur après être comittés et les récupérer sur n’importe quel poste (client) avec de simples commande (git pull).

# CREATION D’UN REPERTOIRE GIT DANS DOSSIER GIT ET GESTION DE SES FICHIERS :

1. Grâce à la commande Unix mkdir (make directory) on créera un répertoire (cr-sse-git) dans le dossier git comme le montre la figure suivante :

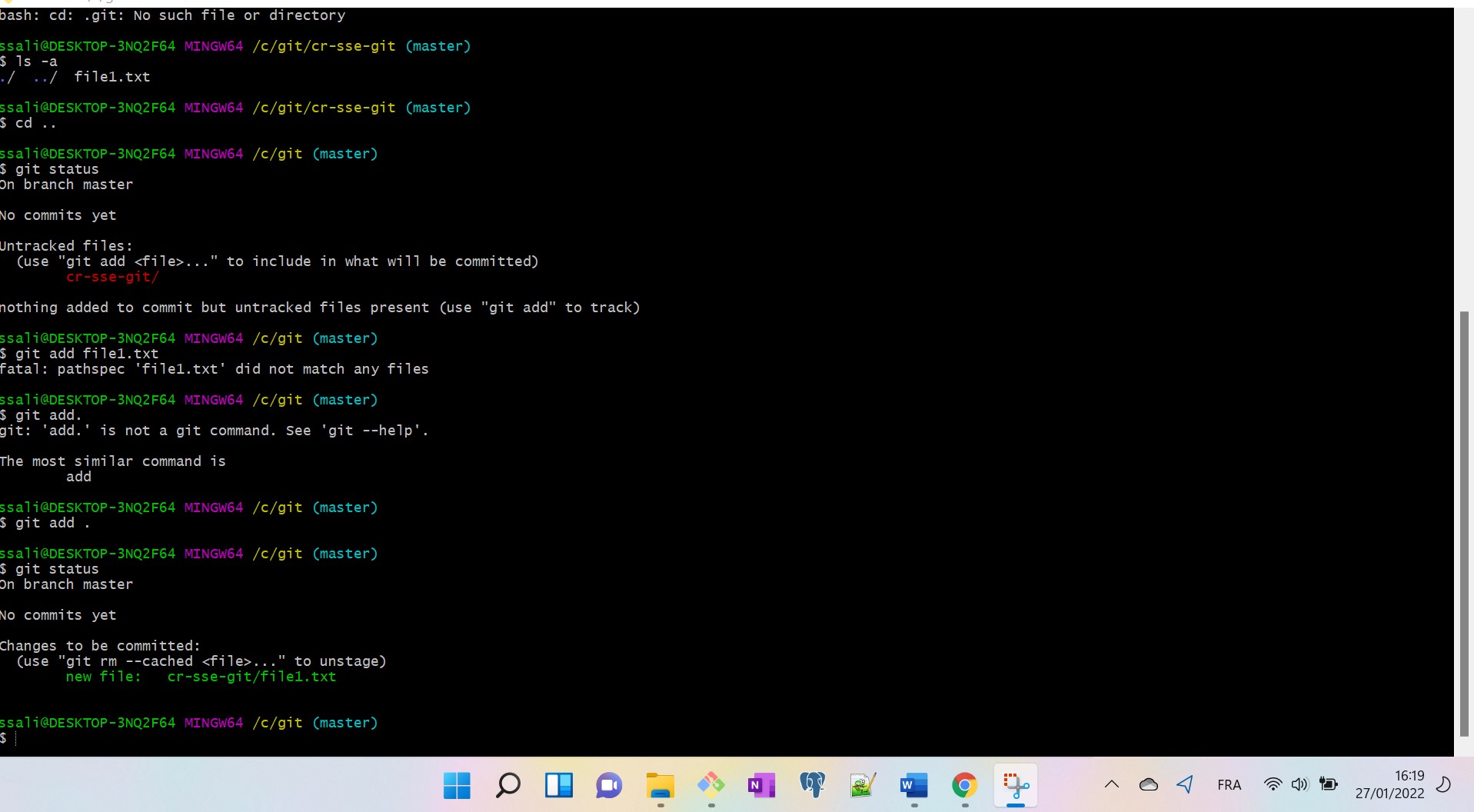


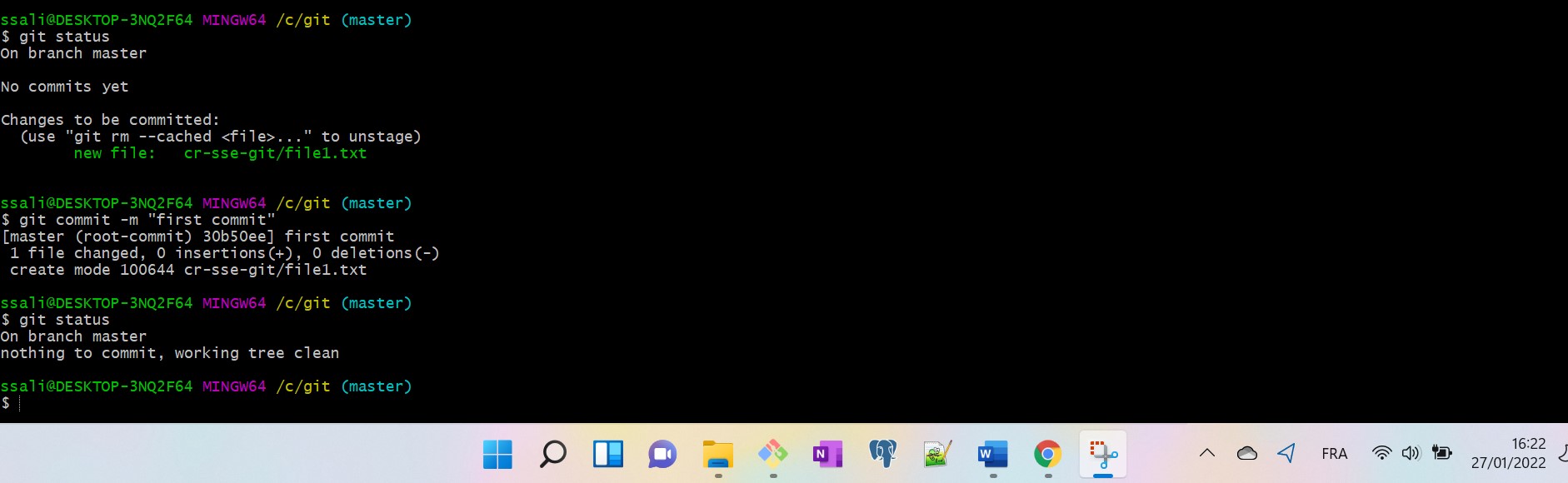
1. Grâce à la commande Linux cd cr-sse-git on va rentrer dans le répertoire (cr-sse-git) créé afin de créer des fichiers .txt avec la commande de Linux ‘touch nom\_fichier ‘.

* La commande git init permet de créer un sous répertoire git dans le but de créer des fichiers cachés qui vont initialiser le. Git et initialiser la brache master.
* La commande git status permet de faire un état des lieux dans le répertoire dans lequel je me retrouve.
* Quand on a un un fichier en rouge (status c’est untracked ou modified) c’est le cas de file1.txt qu’on vient de créer, les fichiers verts c’est les fichiers en attente de commit.

Quand il n’y a rien à ajouter pour être commité la commande status nous le renseigne.

* La commande git add . ou git add (nom\_du\_fichier), permet de rejouter tout les fichiers ou le seul fichier qu’on veut stagé.
* La commande ls permet de nous renseigner le ou les fichiers créés dans un répertoire git.

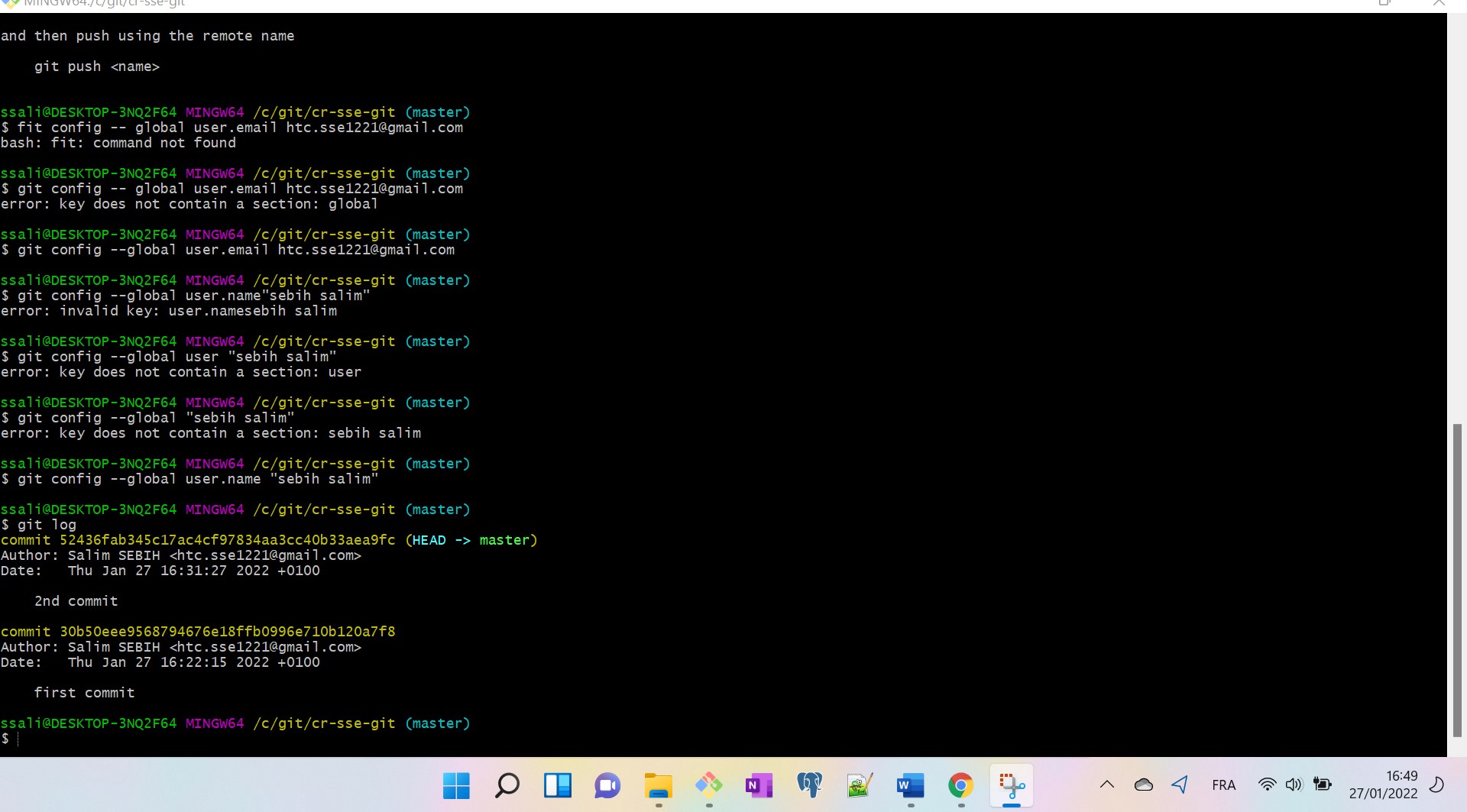


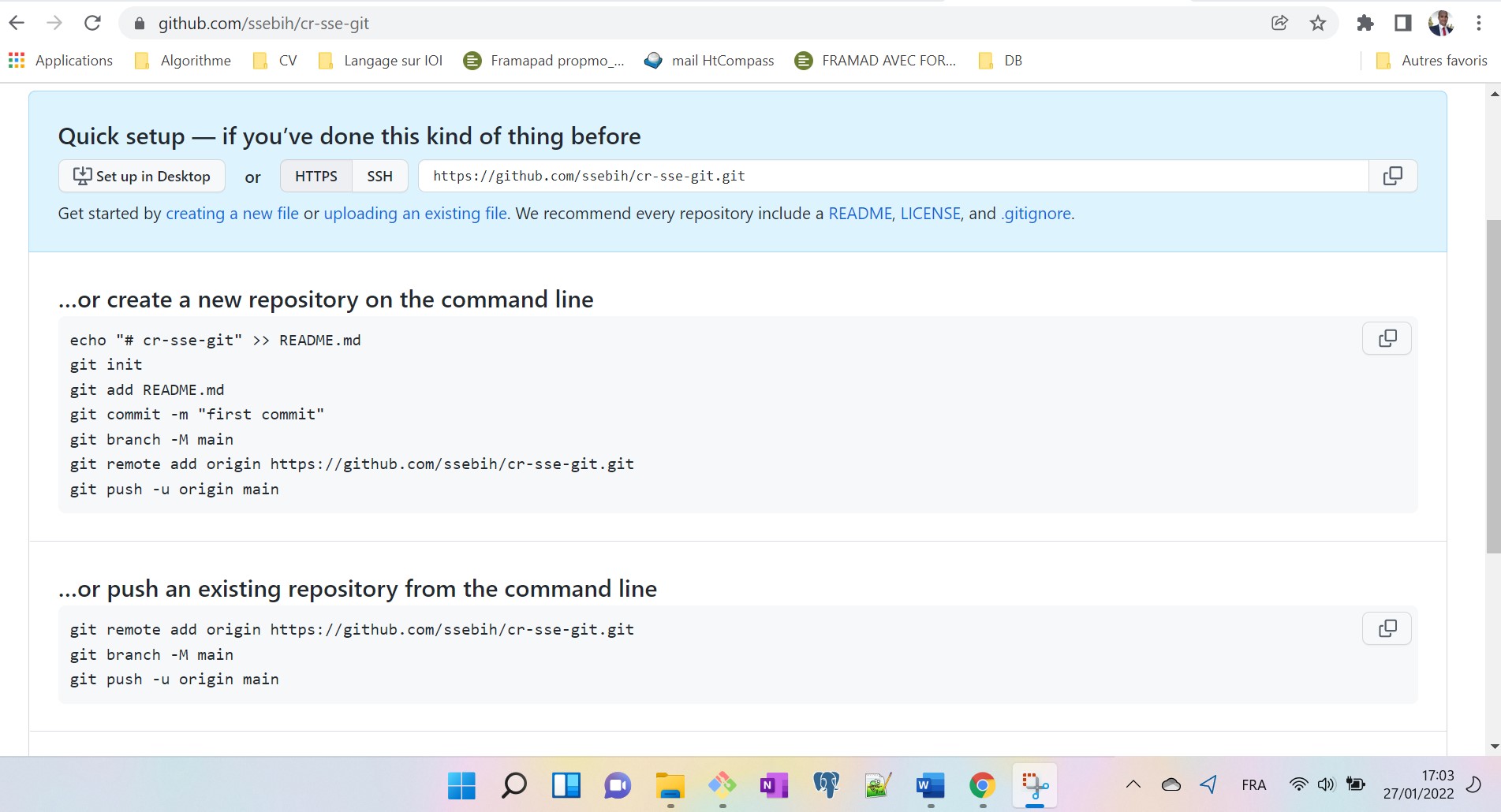


* L commande git log permet de montrer tout les commit qu’on a faits.
* Après avoir créé les fichier 1 2 3 et 4 .txt on essaie maintenant de les enregistrer au serveur.

Oups on ne peut pas et git nous guide afin de pouvoir réussir ceci, il nous demande de renseigner d’abord sur quel compte au serveur on doit enregistrer cela.

On utilise donc notre compte créé en ligne précédemment sur la plateforme Git Hub afin de pouvoir réussir notre push. Il faut donc utiliser la commande git config –global user.email et git config –global user.name. En suite il faut aller sur le site github.com pour créer le repository (git)



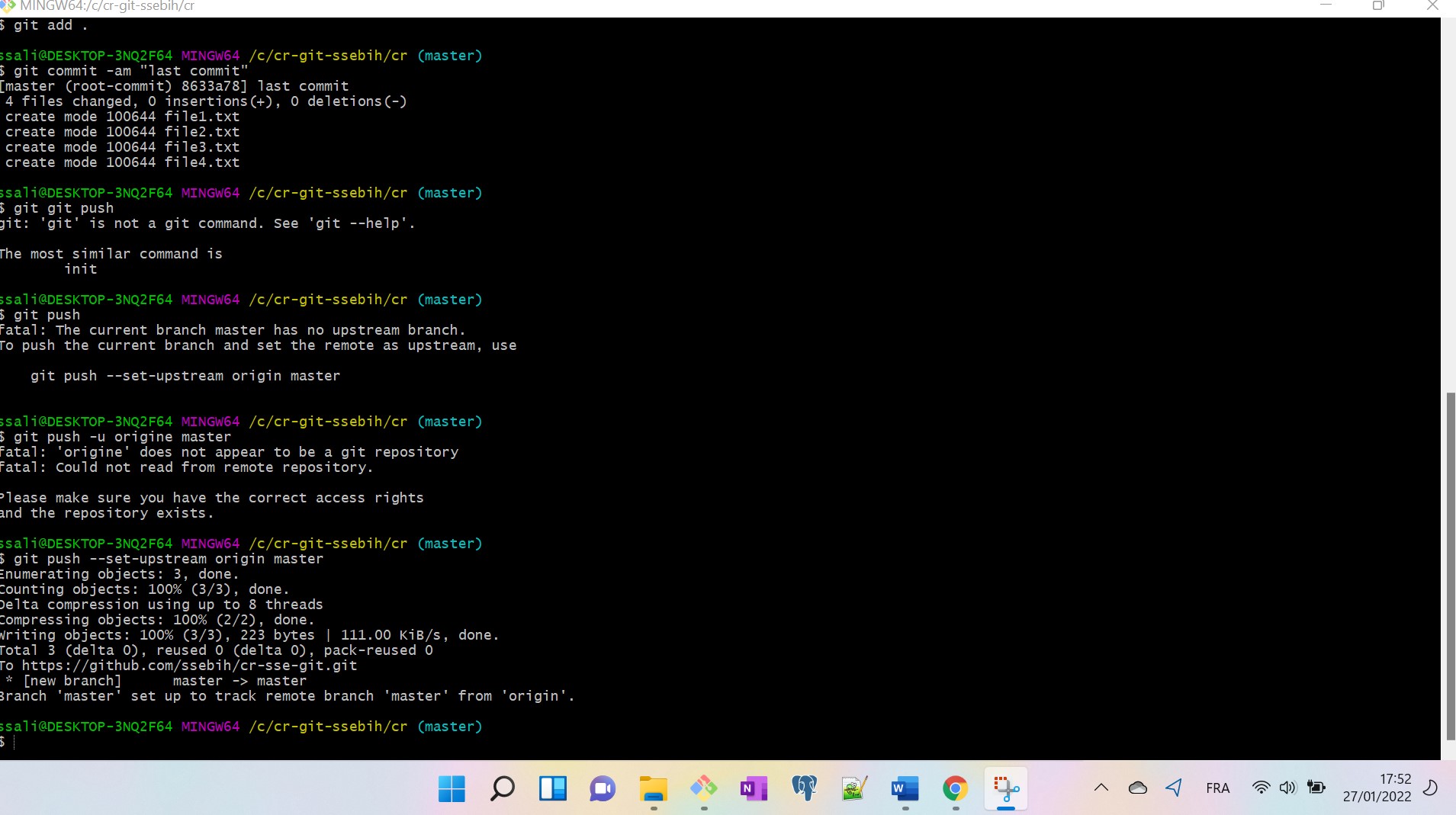


# 

Par la suite il faut copier le lien du repesotory et le coller sur la console git bash afin qu’il puisse faire un git push et ceci grâce à la commande git remote :

git remote add origin git remote add origin https://github.com/ssebih/cr-sse-git.git

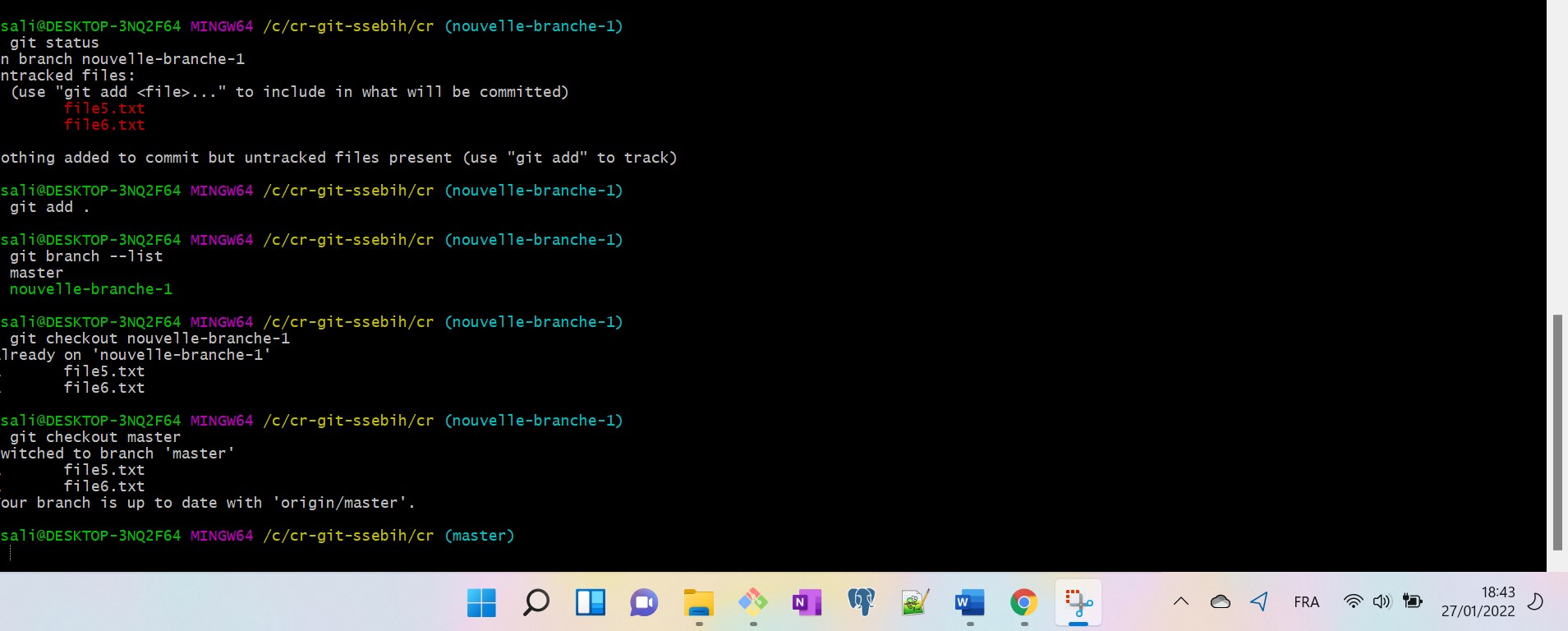
La commande rm est une commande linux qui permet d’effacer des fichiers.



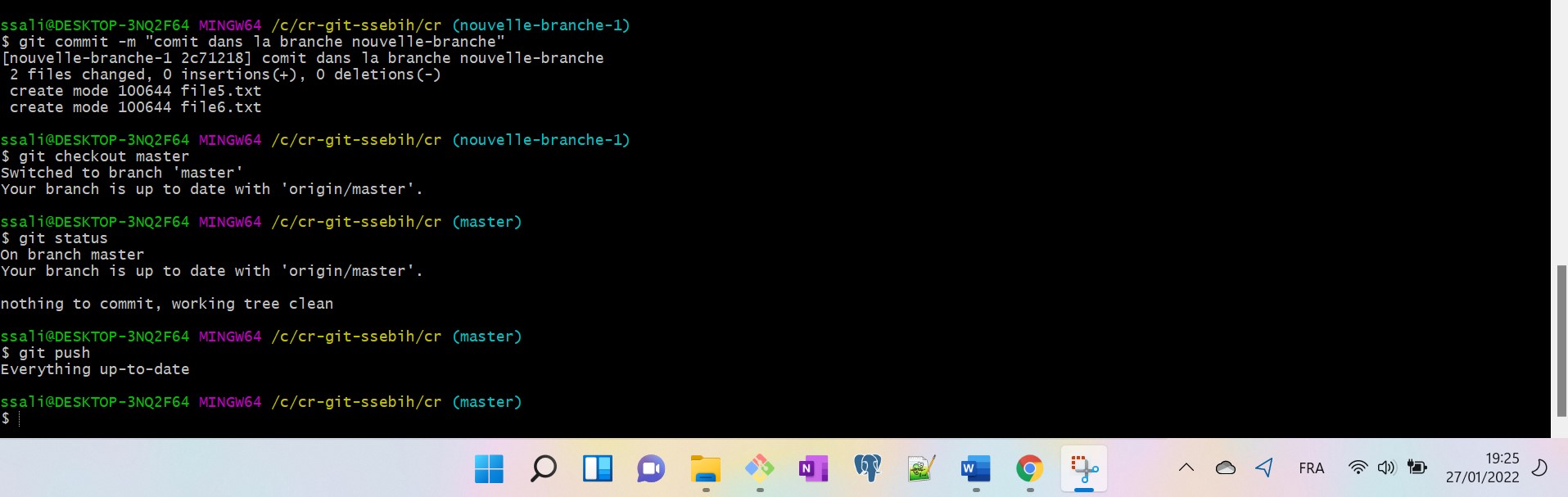
# Chapitre 3 : LES BRANCHES :

Pour créer une nouvelle branche dans un repository on utilise la commande : git nouvelle-branche-1

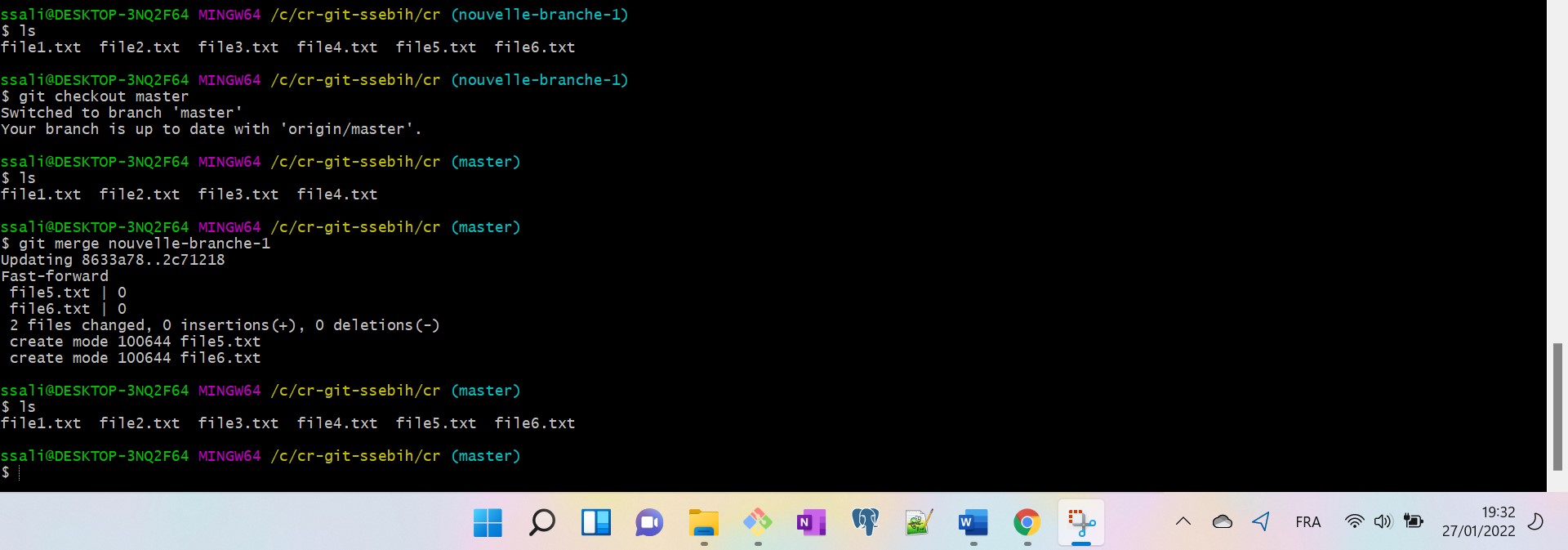
Pour connaitre le nom de toutes les branches et la branche sur laquelle on travaille (en verte sur la console)



* On a créé file5 et file6 dans la branche : nouvelle-branche-1. On les a stagés avec git add . on a changé de branche vers master par la commande git checkout master et on les a remarqués qu’ ils nous suivent vers la branche master on peut **donc conclure** si le fichier n’est pas comitté sur un branche il reste toujours volatile et il nous pour chaque changement de branche 🡺 git commit est la seule façon de lier un fichier à une branche.
* Si on va vers la branche master et on fait un git status git nous renseigne qu’il n’existe aucune modification à signaler la preuve que les fichier 5 et 6 sont comittés et liés à la branche : nouvelle-branche-1.

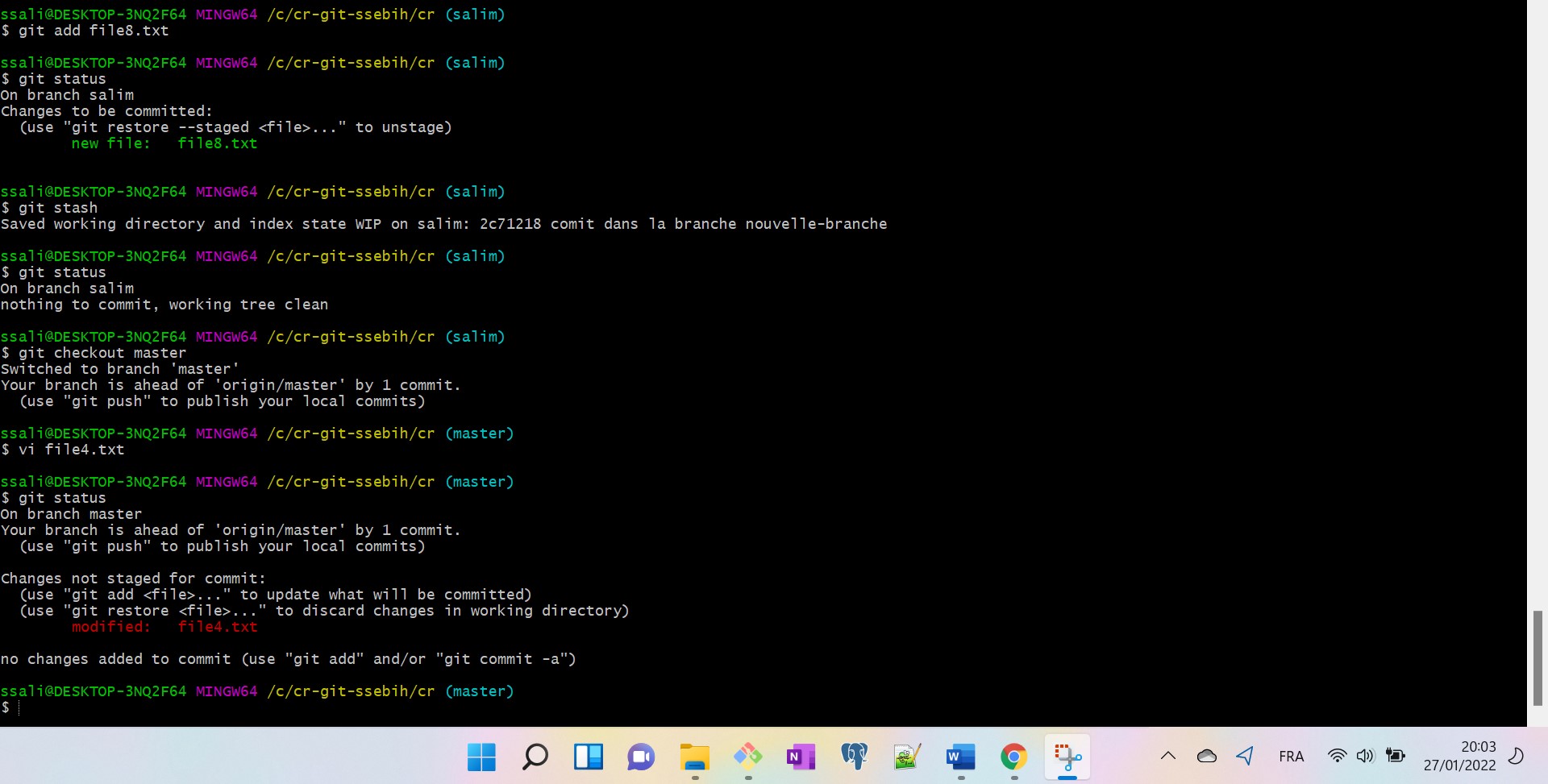


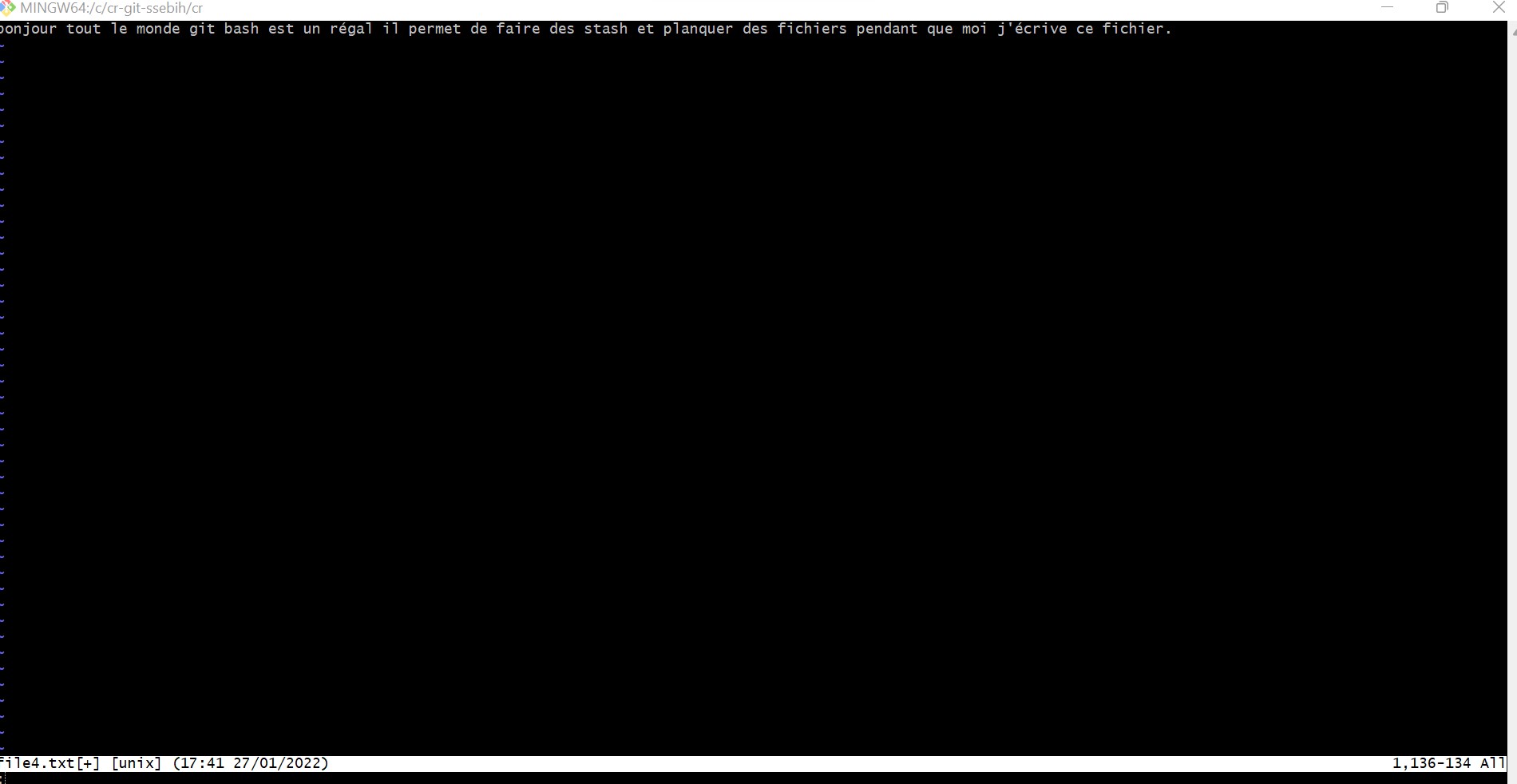
Maintenant si on veut récupérer le contenu de la nouvelle branche « nouvelle-branche-1 » dans master on doit se positionner sur la branche master et on applique git merge nouvelle-branche-1



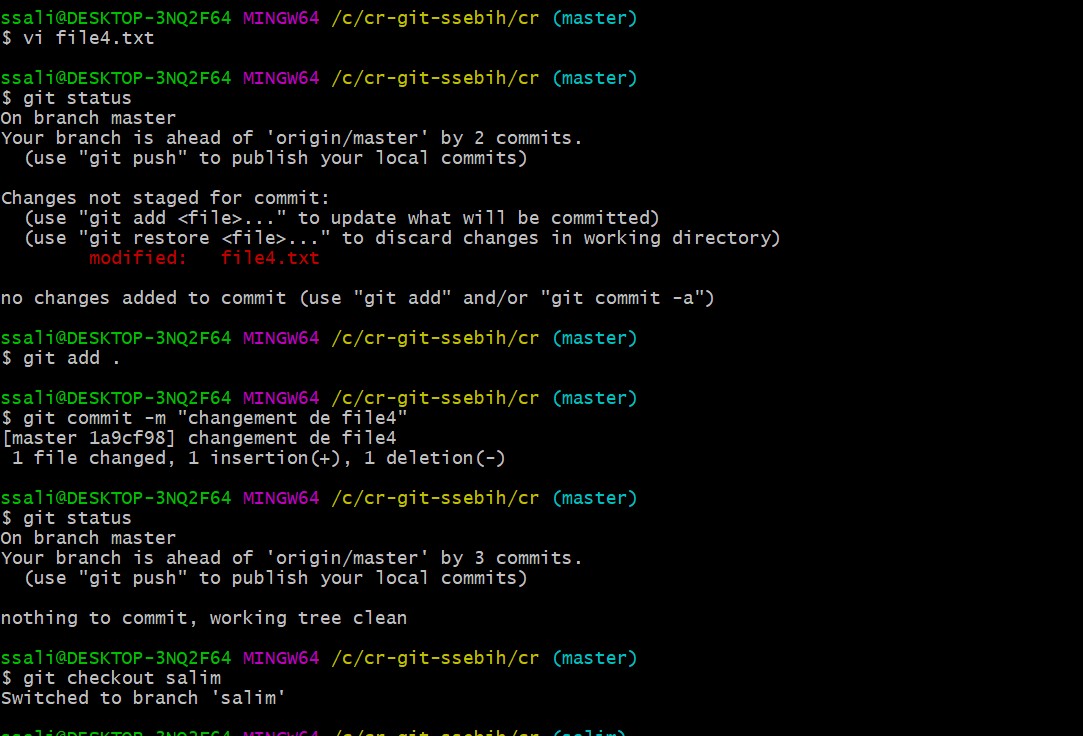
Si on veut créer une nouvelle branche « salim » et aller travailler directement sur elle on applique la commande suivante :

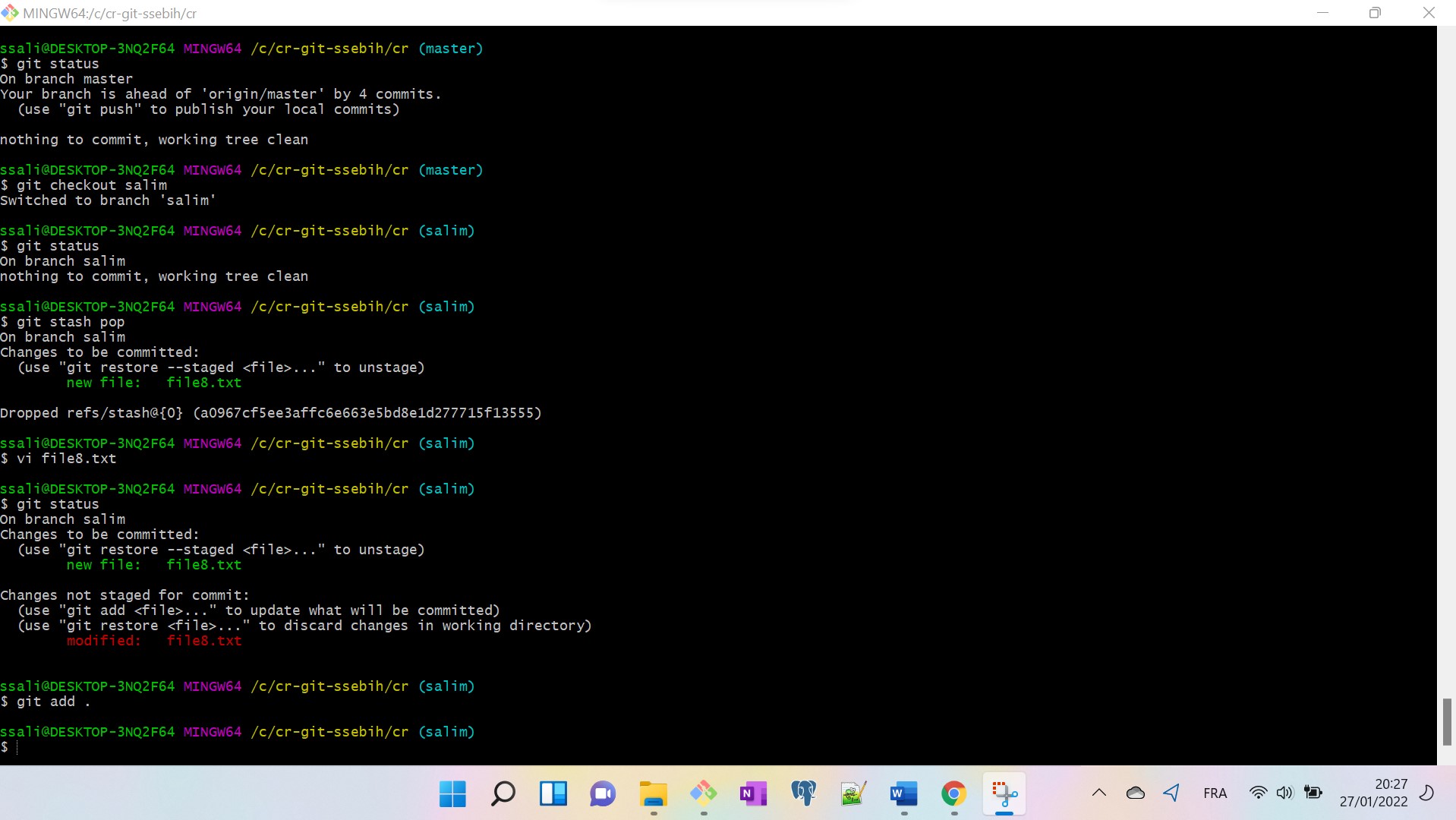
Git checkout -b « salim » et on va créer un fichier file8.txt on commence à travailler dessus et tout d’un coup on sera obligé de revenir sur la branche master pour faire des corrections sur le fichier file4.txt c’est urgent alors qu’on ne peut pas comitté parce que le fichier 8 n’est pas encore fini mais on peut ajouter (add). A ce moment la on utilise la commande git stash après git add.





Une fois les modification sur le fichier file4.txt sur la branche réalisée je le comitt puis je reviens maintenant sur la branche « salim » pour continuer à travailler sur le fichier file8.txt.





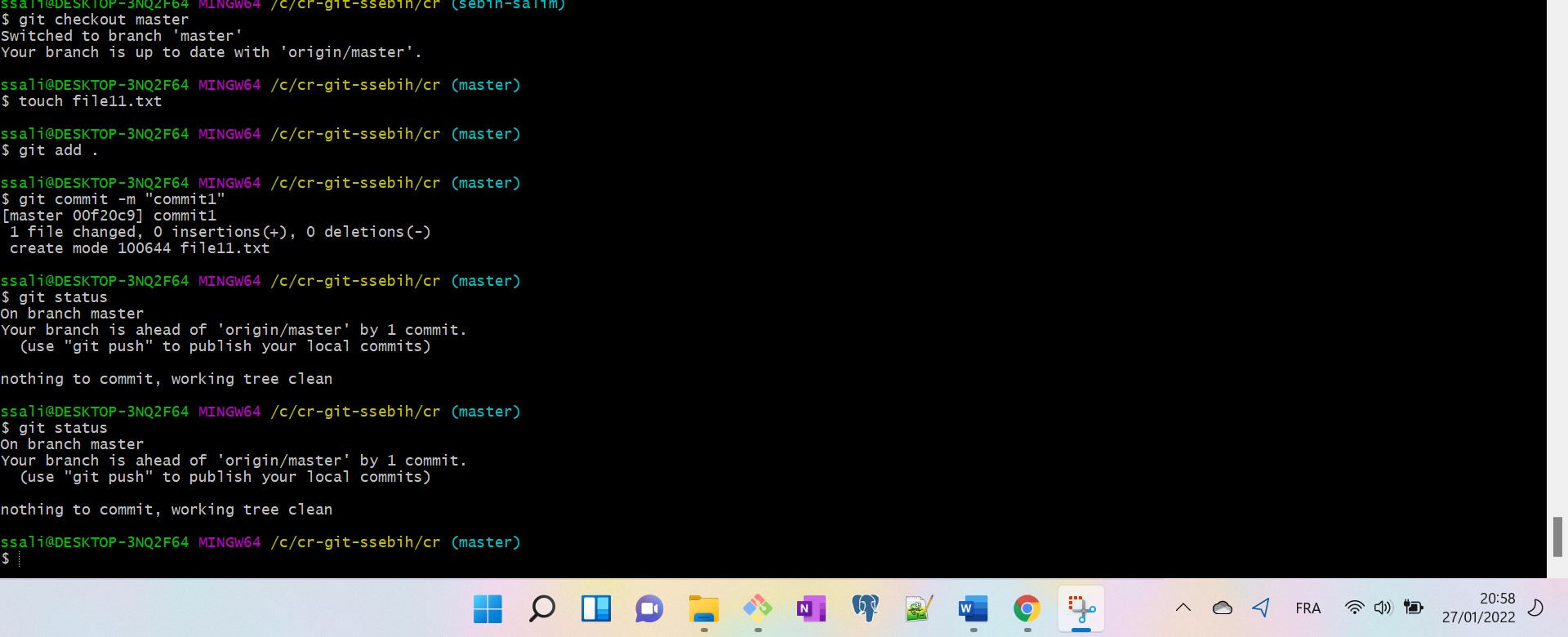
En effet on constate que git stash est une commande extrêmement importante dans git car elle permet quand même d’avoir un statut intermédiaire entre commit et add car si on ne fait que git add le fichier file8 nous suivra sur l’autre branche alors que l’autre branche ça se trouve est en production et on devra committe le fichier file4.txt après correction et on va tout fausser par cette manœuvre car le fichier file8.txt est une version très primitive et non finie. Et si on ne le garde pas dans la branche « salim » alors on aurait perdu un travail de quelques heures voir de quelques journées, surtout on ne peut le comitter car vu son primitif et non fini.

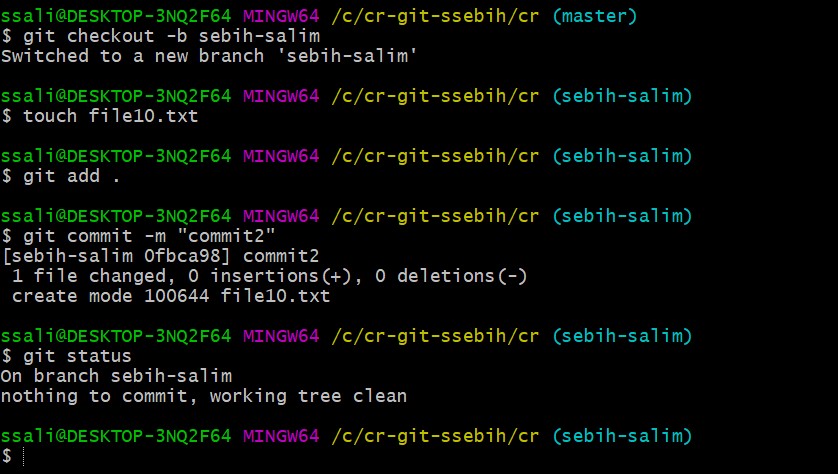
**NB :** j’ai constaté que si on est sur branche 1 et on ouvre un fichier qui se trouve sur branche 2 avec la commande de linux vi on aura une copie de ce fichier dans la branche 1 et si on fait on fait merge il y aura un conflit de données car on aura deux fois le fichier qu’on ouvert, il faut donc faire le nécessaire « mise à jour de fichier » avant de merger.

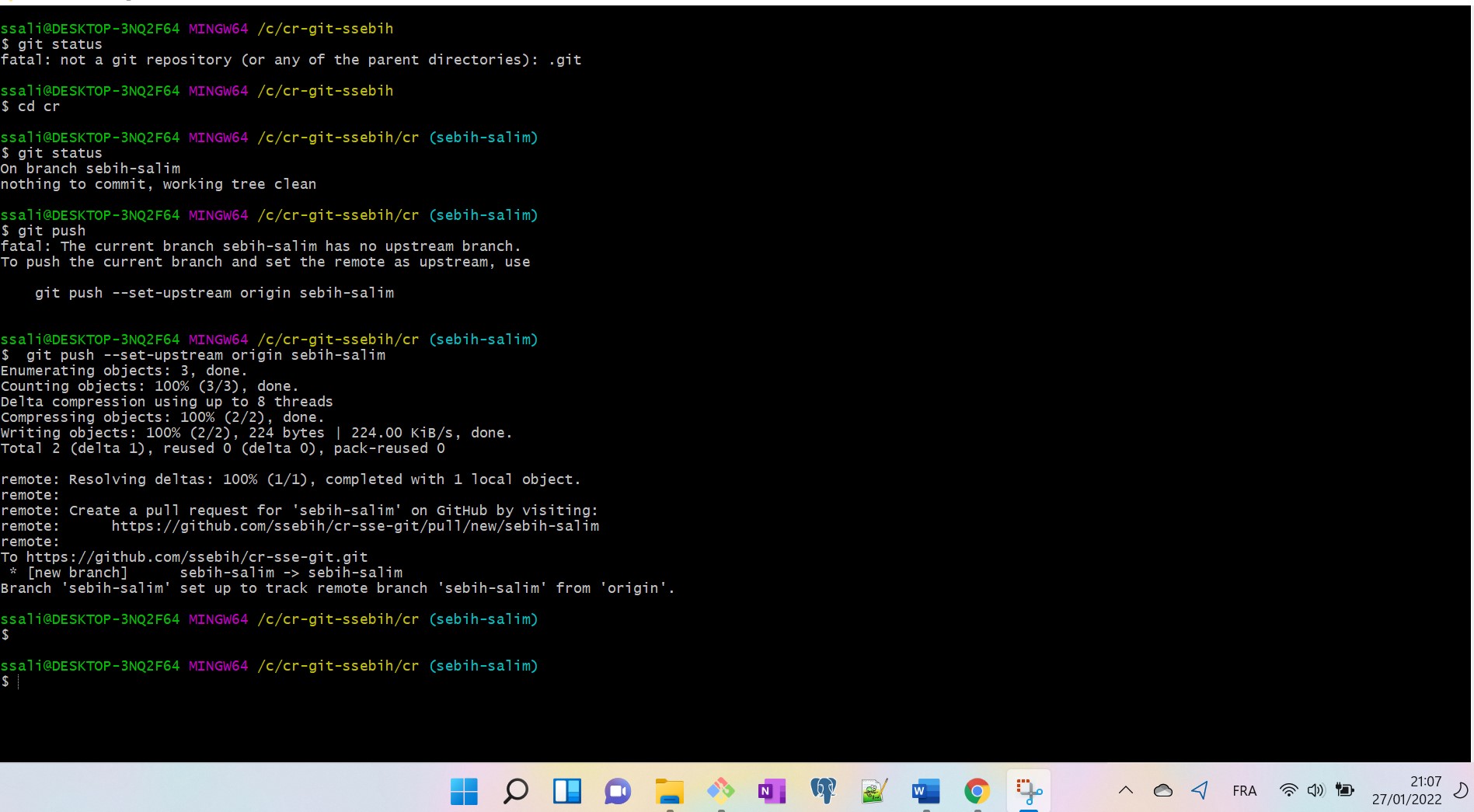
* Maintenant on essaie de faire un push sur deux commit qui se trouvent dans deux branches différentes.

Branche master ----commit 1

Branche salim-sebih----comit 2







On constate que il est impossible de faire des push sur des commit réalisés sur des branches différentes et chaque push concerne les commit réalisés sur la même branche.