



Git et GitHub

Introduction

Par Richard BONNAMY



Introduction

Git – Présentation et historique

➤ Présentation

- Git est un logiciel de type VCS décentralisé: **V**ersion **C**ontrol **S**ystem
- Un **VCS** permet de gérer l'historique d'un système de fichiers (v1, v2, etc..) :
 - Faire une nouvelle version, revenir à un état précédent, comparer 2 versions, etc..
- Git est un logiciel en ligne de commande

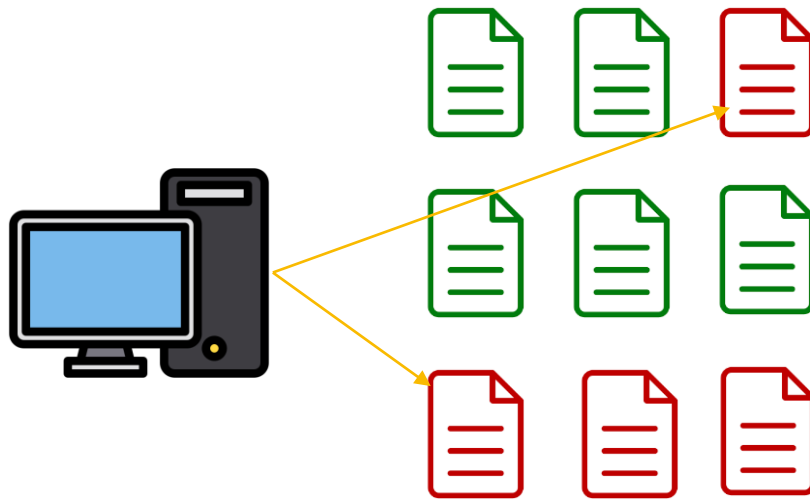
➤ Historique

- Développé en 2005 par Linus Torvalds, le créateur de Linux.

Git – A quoi ça sert un VCS ?

Problématique 1

- Vous travaillez sur une **application comportant des centaines de fichiers**
- Vous avez **modifié plusieurs fichiers** et suite à ces modifications **l'application ne fonctionne plus**
- Vous n'arrivez pas à faire un **retour arrière** (CTRL+Z ne fonctionne pas pour les fichiers qui ont été fermés puis réouverts).
- Vous êtes dans une **situation embêtante...**



Comment faire un retour arrière ?

Git – A quoi ça sert un VCS ?

Problématique 2

- Vous avez **livré au client** une application.
- Le client vous demande de corriger un bug.
- **Problème** : entre temps vous avez fait des tas de modifications sur l'application.
- Comment faire ? Vous n'allez pas livrer des fonctionnalités non testées
- Vous êtes dans une **situation embêtante...**

Git – Solution 1 : sans VCS

- **Sans VCS** vous pouvez **copier/coller** votre système de fichiers dès que vous êtes dans un état stable ou que vous livrez.
- Mais pour être efficace vous aurez besoin d'un logiciel permettant de comparer 2 versions de votre système de fichiers, de réaliser des transferts de fichiers, etc.
- Comment collaborer à plusieurs avec ce genre de solution ?

Git – Solution 2: avec un CVS

- **Un VCS, comme Git**, propose de faire des versions (ou **commits**) du système de fichiers.
- En cas de difficulté vous pouvez revenir à n'importe quelle version (commit).
- Pour ce faire Git permet de **transformer n'importe quel système de fichiers** et tout son contenu en **dépôt Git**.
- Vous pouvez avoir autant de dépôts Git que vous voulez sur un ordinateur donné.
- **Chaque dépôt a son propre cycle de vie.**

Git – A quoi ça sert ? Avantages

- **Versions incrémentales** : plus économique qu'un copier/coller d'un système de fichiers
 - Imaginez lorsque vous n'avez modifié que quelques dizaines de lignes parmi plusieurs milliers de fichiers.
- Permet de **versionner, comparer**, faire des **retours arrière, gérer des conflits** sur des fichiers, etc.
- Il permet également de réaliser des **branches** afin de permettre le travail d'équipe (cf. GitHub).
- Un **écosystème** s'est développé autour de Git avec notamment des plateformes comme **GitHub**.

Git – VCS centralisé vs décentralisé

- Il existe des **VCS centralisés** comme **Subversion**.
 - Inconvénient : lorsque le serveur central tombe vous ne pouvez plus faire de versions, de comparaison, etc.
- **Git est décentralisé.**
 - Cela signifie que même si le serveur central tombe, vous avez votre dépôt en local sur lequel vous pouvez réaliser toutes les opérations décrites précédemment
- Vous avez dit **serveur central** pour Git ? **Mais à quoi ça sert ?**

Git – Comment travailler à plusieurs ?

- Comment faire pour travailler à plusieurs sur un projet ?
 - Vous les remontez sur une plateforme de partage de fichiers comme Google Drive ?
 - Vous vous les envoyer par mail ?
- Comment gérez-vous les versions que chacun fait sur son poste ?
- Comment gérez-vous les modifications concurrentes sur des fichiers ?
- **Solution** : avoir un serveur central pour le partage.

GitHub - Présentation

- **GitHub** est un **service cloud** d'hébergement de **dépôts Git**
- **GitHub** permet de gérer des dépôts publics ou privés
- **GitHub** repose sur Git.
- **GitHub** permet de collaborer en équipe sur des dépôts.
- **GitHub** a des concurrents, ce n'est pas la seule plateforme qui propose ce genre de services : Bitbucket, GitLab notamment.
- **GitHub** a été créé en 2008 et racheté par Microsoft en 2018 pour 7,5 Mds de \$

Créer son compte GitHub ?

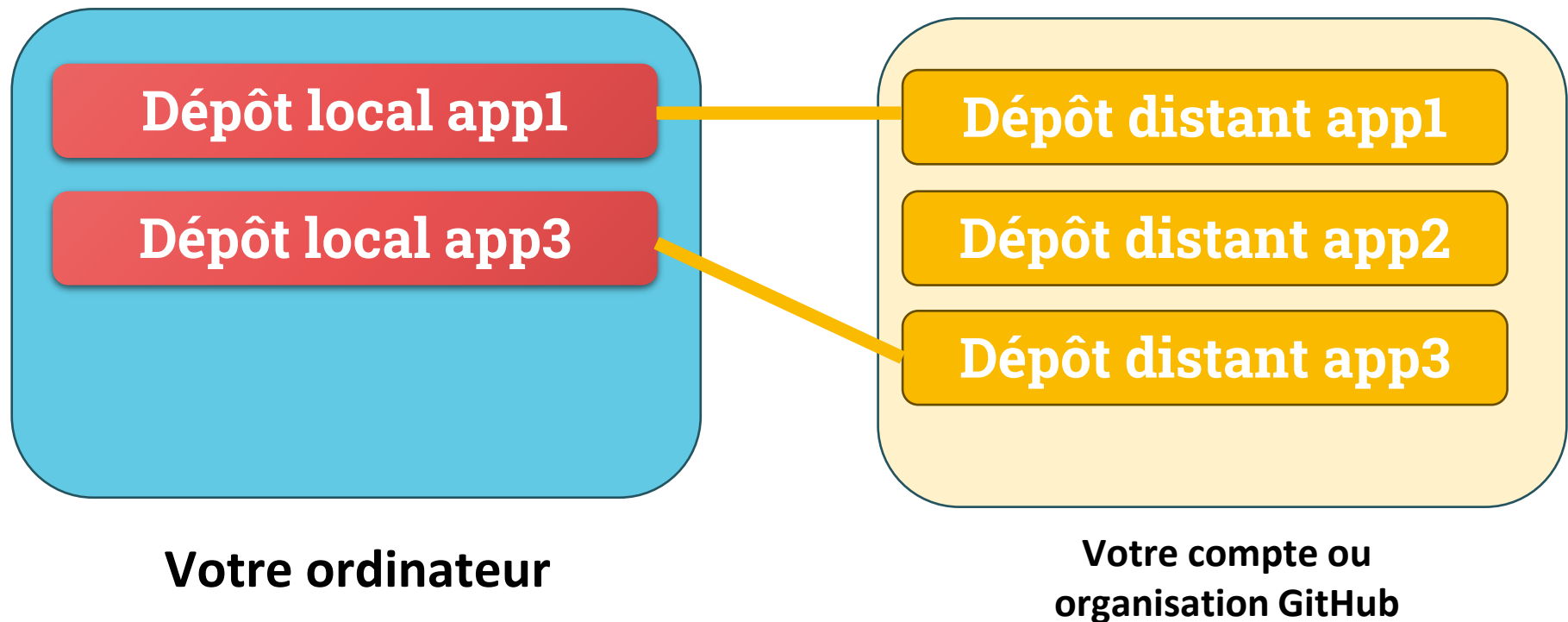
- Vous pouvez créer **gratuitement** un **compte GitHub**
- Vous pouvez également créer des **organisations** GitHub regroupant **plusieurs utilisateurs GitHub** pour partager des dépôts, travailler ensemble.
- Le couple Git/GitHub est **décentralisé**.

Comment travailler avec GitHub ?

- La **décentralisation** signifie de pouvoir :
 - Travailler sur un dépôt local (votre ordinateur) hors connexion
 - Synchroniser avec le dépôt distant si besoin.
- Pour que cela fonctionne il faut que le **dépôt local "connaisse" son dépôt distant** correspondant :
 - Est-il sur GitHub ? Sur Bitbucket ? Sur Gitlab ?
 - Sur quel compte est-il ?
 - A-t-il le même nom ?
- Un dépôt sur **GitHub** a une **URL**:
 - Une commande git permet de lier un dépôt local avec un dépôt distant (via son URL).

Git et GitHub

➤ Schématiquement :



- C'est **Git** qui va vous permettre d'échanger des informations entre vos dépôts locaux et vos dépôts distants.

Les services offerts par GitHub

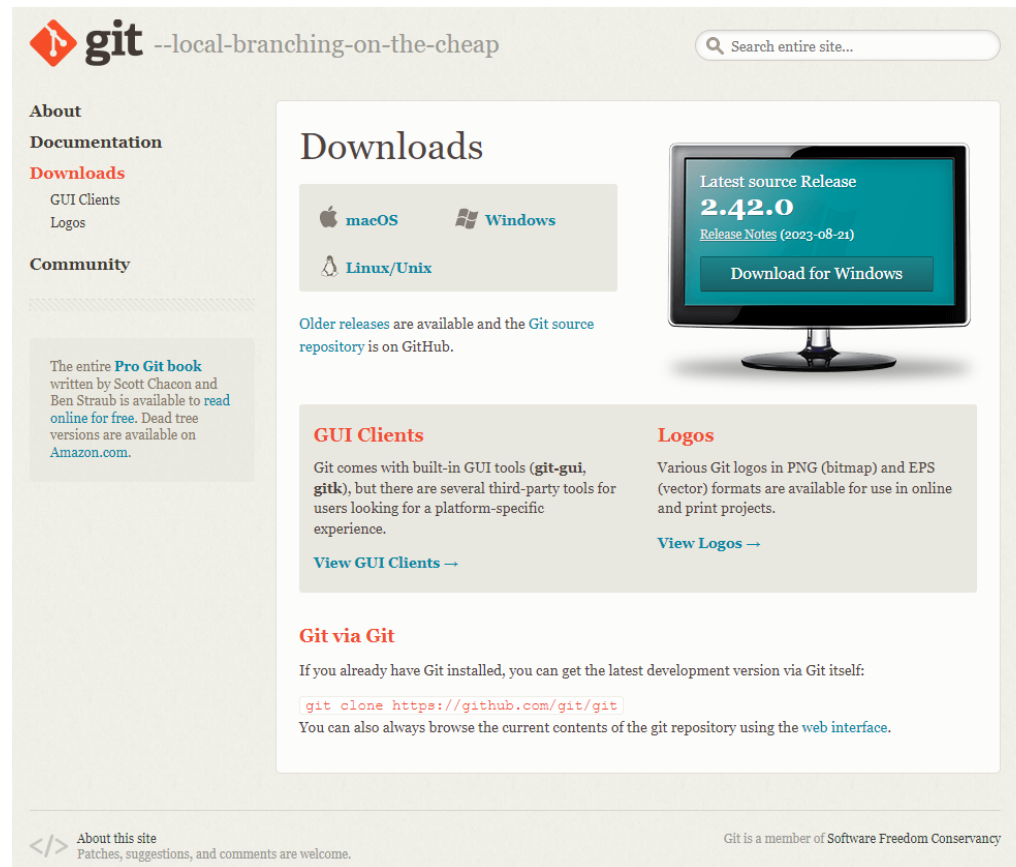
- C'est **gratuit** avec un **nombre illimité de dépôts** publics et privés
- **Intégration continue** avec GitHub (limitation si compte gratuit)
 - GitHub actions : préparer par exemple une archive de livraison pour un client.
- Hébergement de packages (limitation si compte gratuit)
 - si vous développez vos propres librairies et que vous voulez vous en servir pour d'autres projets.
 - Peut servir dans des dépendances MAVEN (vous verrez le cours).
- Vous pouvez **gérer des projets** depuis cette plateforme
- Vous pouvez **copier sur votre compte un dépôt public** dont vous n'êtes pas propriétaire via un mécanisme de **Fork**.
- Vous pouvez **contribuer à un projet forké** via un mécanisme de "**Pull Request**" que va recevoir le propriétaire.



Installation git bash

Git - Installation

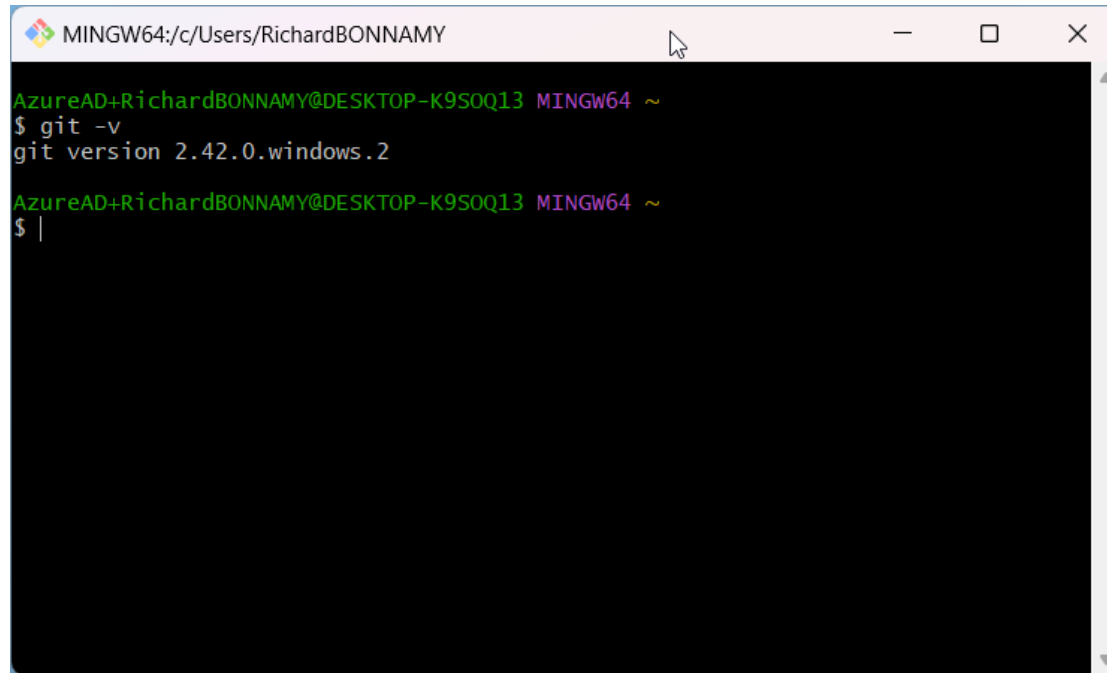
- Depuis cette URL : <https://git-scm.com/downloads>



- Téléchargez et installez l'application Git.

Git - Vérifier l'installation de git

- Démarrez **Git Bash** ou ouvrez votre terminal.
- **Git Bash** permet d'**exécuter de nombreuses commandes** en plus des commandes Git.
- Tapez la commande : **git -v**



```
MINGW64:/c/Users/RichardBONNAMY
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 ~
$ git -v
git version 2.42.0.windows.2
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 ~
$ |
```



Création compte
GitHub

Comment travailler avec GitHub ?

- **Créez votre compte GitHub**
- Il y aura sans doute une double authentification à mettre en place avec un numéro de portable.

Communication avec GitHub et SSH

- **GitHub** demande aujourd'hui à ce que les échanges reposent sur le **protocole SSH** et non plus https.
- Historiquement le protocole SSH (Secure Shell) a d'abord été utilisé pour administrer des serveurs UNIX ou Linux à distance.
- Il remplace progressivement Telnet à partir de 1995, date de sa création.
- SSH 2.0 depuis 2006.

Parenthèse : SSH et chiffrement

- **SSH** repose au démarrage de la connexion sur un chiffrement basé sur des clés asymétriques (clé publique / clé privée)
 - Ensuite cela se complique un peu, avec l'échange de clés symétriques pour chiffrer le reste de la connexion SSH
- Il existe plusieurs algorithmes de génération de clés : rsa (2048 bits), rsa-4096 (4096 bits), DSA (1024 bits), ed25519 qui est considéré comme plus sûr que RSA-4096
- A noter qu'aujourd'hui les chiffrements basés sur des clés de 1024 bits ne sont plus considérés comme sûrs.
 - En 2016 déjà on évoquait des procédés pour casser de telles clés comme le **crible algébrique** (Number Field Sieve en anglais).
 - En 2016, une IA aurait réussi à casser des clés RSA sur 2048 bits
 - ➔ Utilisez **ed25519**

Générer une paire de clés SSH

- Vous allez donc devoir générer une paire de clés SSH
- Sous **Windows** les paires sont dans le répertoire `C:/Users/monCompte/.ssh`
- Sous **MacOS** les clés sont dans le répertoire `/Users/monCompte/.ssh`
- **.ssh** est un répertoire caché.
- Vérifiez si vous posséder déjà des clés **rsa** ou **ed25519**. Dans ce cas il est préférable de les réutiliser pour GitHub. Allez directement diapo 24.
- Voici à quoi ressemblent les clés si elles existent :

```
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 464 Dec 11 2022 id_ed25519
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 103 Dec 11 2022 id_ed25519.pub
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 2635 Sep 23 2022 id_rsa
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 592 Sep 23 2022 id_rsa.pub
```

Générer la paire de clés

- Ouvrez **Git Bash**
- Tapez la commande suivante en remplaçant **mon_adresse_mail_github** par le mail de votre compte GitHub :

ssh-keygen -t ed25519 -C "mon_adresse_mail_github"

- **Ignorez** les questions suivantes en appuyant sur Entrée:

```
Generating public/private ecdsa key pair.  
Enter file in which to save the key (/c/Users/RichardBONNAMY/.ssh/id_ecdsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:
```

- **La commande génère 2 fichiers:**

id_ed25519

Clé privée

id_ed25519.pub

Clé publique (.pub)

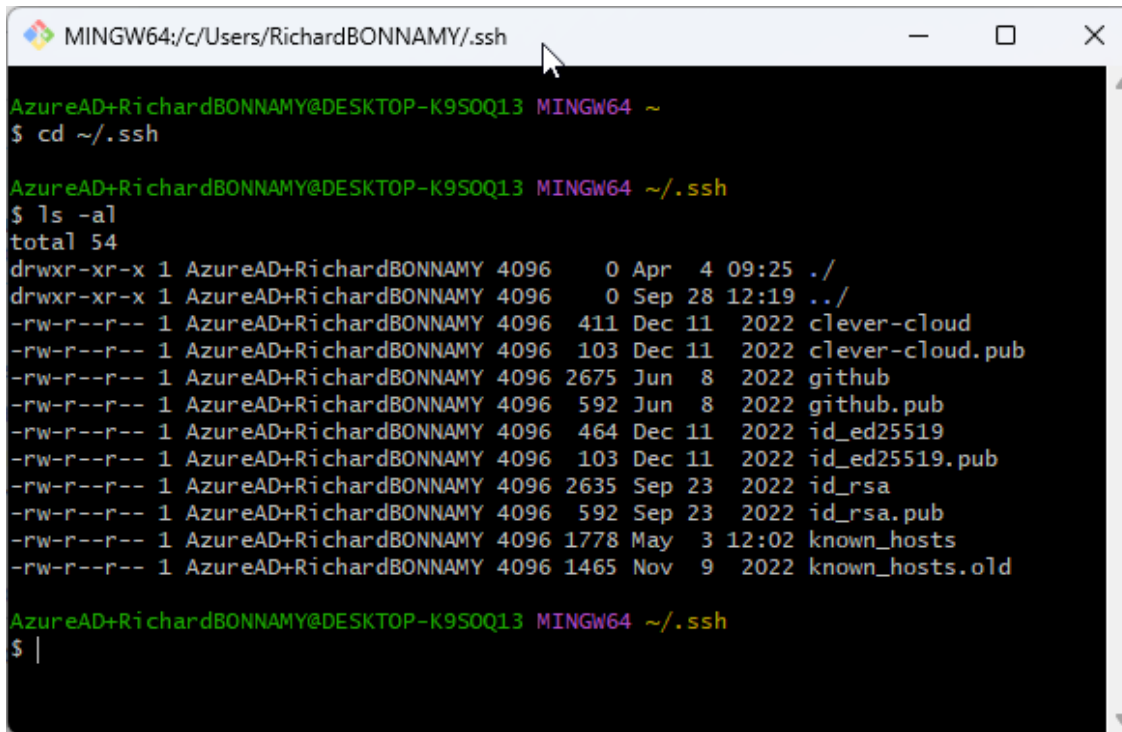
ssh-keygen – quelques explications

ssh-keygen -t ed25519 -C "mon_adresse_mail_github"

- **ed25519** est un algorithme de génération de clés.
 - A noter que GitHub supporte les 2 types de clés: **ed25519** ou **rsa**
 - L'option **-C** permet de passer un commentaire aidant à identifier la clé
- La commande **ssh-keygen** vous demande des infos facultatives.
- Vous pouvez modifier le nom du couple de fichiers dans lesquels seront hébergées les clés..
 - Si vous saisissez une passphrase (mot de passe), vous serez contraints de saisir cette passphrase à chaque commande **git vers ou depuis le dépôt distant**

Que faire de la clé publique ?

- Allez dans le répertoire **.ssh**
 - Vous pouvez faire **cd ~/.ssh**
- Puis tapez **ls -al** pour avoir la liste des fichiers



```
MINGW64:/c:/Users/RichardBONNAMY/.ssh
AzureAD+RichardBONNAMY@DESKTOP-K950Q13 MINGW64 ~
$ cd ~/.ssh

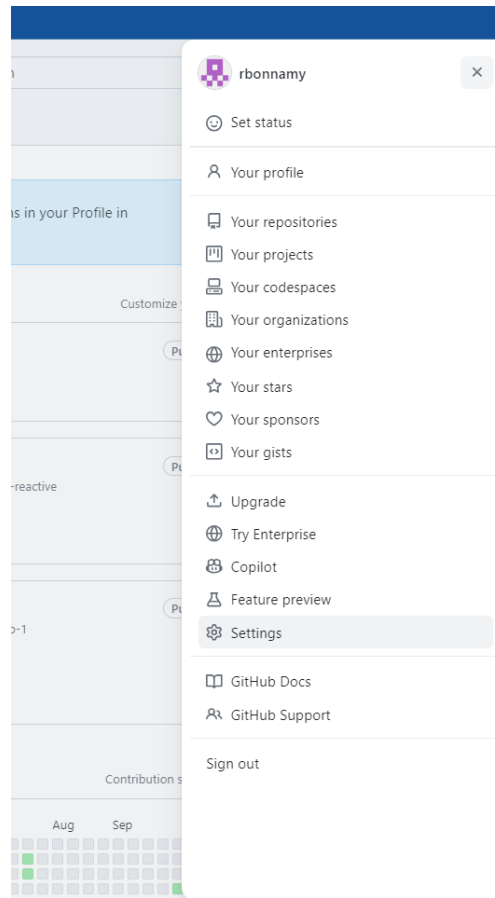
AzureAD+RichardBONNAMY@DESKTOP-K950Q13 MINGW64 ~/.ssh
$ ls -al
total 54
drwxr-xr-x 1 AzureAD+RichardBONNAMY 4096  0 Apr  4 09:25 ./
drwxr-xr-x 1 AzureAD+RichardBONNAMY 4096  0 Sep 28 12:19 ../
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  411 Dec 11 2022 clever-cloud
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  103 Dec 11 2022 clever-cloud.pub
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 2675 Jun  8 2022 github
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  592 Jun  8 2022 github.pub
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  464 Dec 11 2022 id_ed25519
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  103 Dec 11 2022 id_ed25519.pub
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 2635 Sep 23 2022 id_rsa
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096  592 Sep 23 2022 id_rsa.pub
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 1778 May  3 12:02 known_hosts
-rw-r--r-- 1 AzureAD+RichardBONNAMY 4096 1465 Nov  9 2022 known_hosts.old

AzureAD+RichardBONNAMY@DESKTOP-K950Q13 MINGW64 ~/.ssh
$ |
```

- Le fichier à éditer s'appelle **id_ed25519.pub** (pub comme public)

Que faire de la clé publique

- Allez sur votre compte GitHub
- Cliquez sur l'icône de votre profil en haut à droite et sélectionnez settings



Que faire de la clé publique

- Sélectionnez la rubrique **SSH and GPG keys**
- Cliquez sur le bouton **New SSH Key**
- Donnez un **nom** à votre clé SSH, par exemple mon_PC
- Saisissez **l'intégralité** de votre fichier **.pub** dans la zone "**Key**"
- Nous pouvons maintenant passer à la suite du cours.



Commandes de base

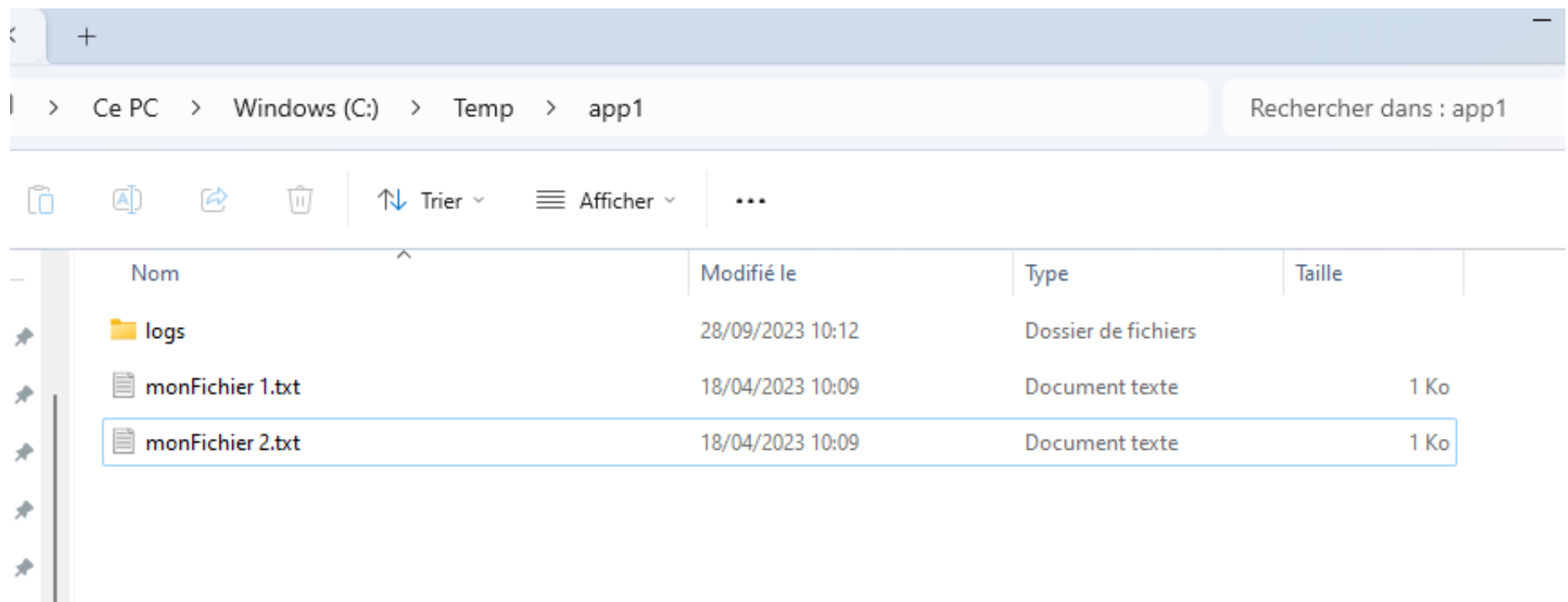
Git – Créer un dépôt local

- Pour **créer un dépôt local** il faut d'abord **choisir le système de fichiers cible**.
- Cela peut être un répertoire vide ou un répertoire contenant déjà des fichiers qu'on veut gérer avec Git.
- La création d'un dépôt Git pour un système de fichiers n'a pas d'impact pour vos fichiers.
- Cela va créer un répertoire **.git** à la racine de votre système de fichiers.
- Ne **supprimez jamais** ce répertoire **.git** ou vous perdrez tout l'historique.
- **Détaillons le process étape par étape.**

Git – Etape 1 : choix du système de fichiers

➤ Etape 1 : choix du système de fichiers

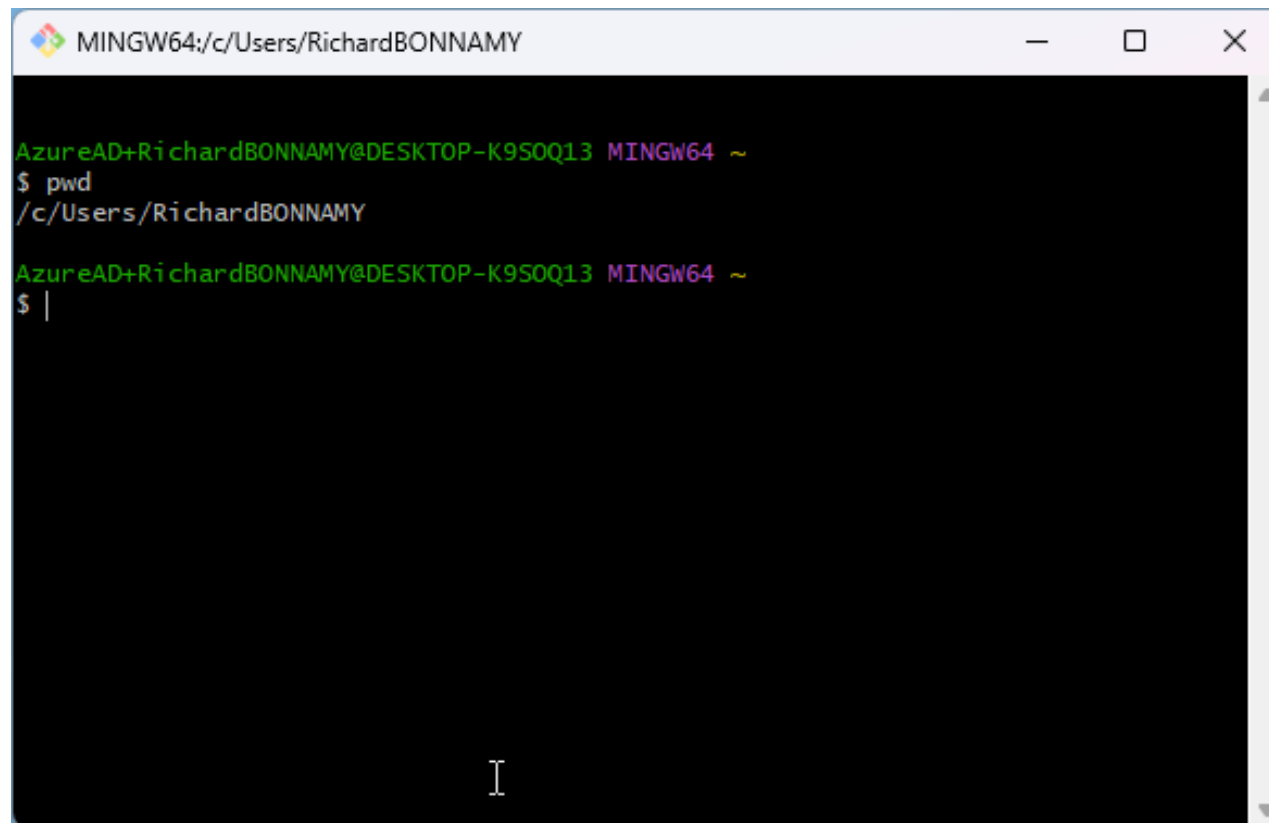
- Imaginons que je veuille créer un dépôt Git pour C:/temp/**app1**
- **app1** contient un répertoire logs et 2 fichiers. Le répertoire logs ne contient qu'un seul fichier.



Git – Ouvrir git bash

➤ Etape 2 : ouvrir Git Bash et regarder où on est

- Lorsqu'on ouvre **Git Bash**, par défaut on pointe sur le répertoire home.
- Lançons la commande **pwd** (UNIX - print working directory) pour le vérifier :



```
MINGW64:/c/Users/RichardBONNAMY

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 ~
$ pwd
/c/Users/RichardBONNAMY

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 ~
$ |
```

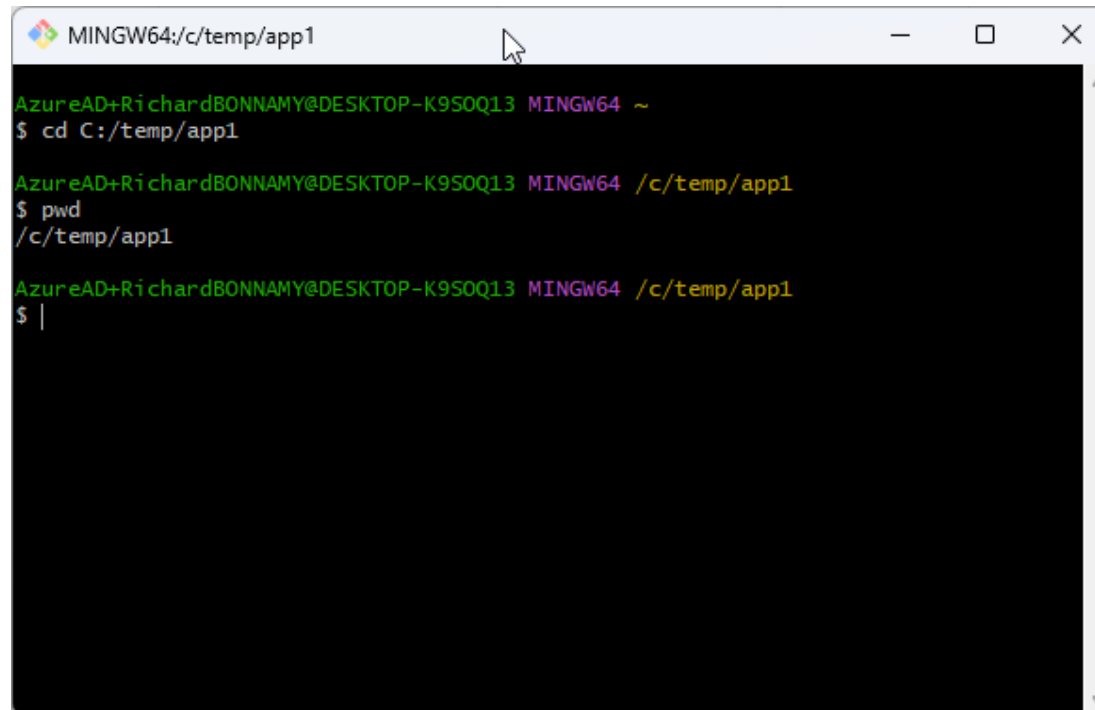

Git – Changer de répertoire

➤ Etape 3 : changer de répertoire de travail pour aller dans le répertoire cible

- Ma cible est : **C:/temp/app1**
- Je vais exécuter la commande suivante pour aller dans le **répertoire app1** :

cd C:\temp\app1

- La **commande cd** permet de **changer de répertoire** (UNIX - cd: change directory)

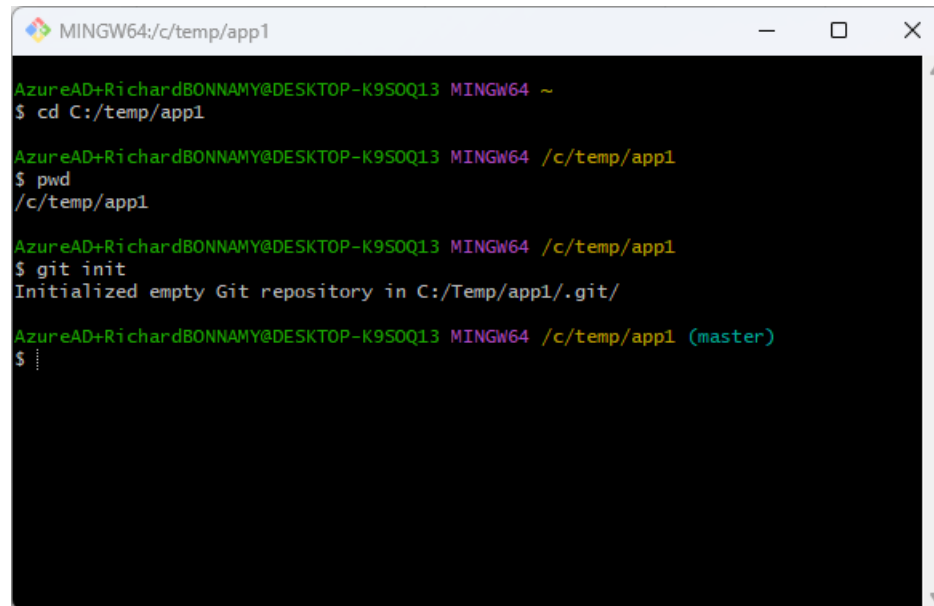


```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 ~
$ cd C:/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1
$ pwd
/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1
$ |
```

Git – Utilisez git init

➤ Exemple – Etape 3 : initialiser le dépôt

- Une fois dans app1, exécutez la commande **git init**
- **Attention il faut le faire dans app1 !!!**
- Si vous le faites dans **temp** alors **vous initialisez un dépôt pour gérer tout ce qui est dans temp**



```
MINGW64:/c/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 ~
$ cd C:/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1
$ pwd
/c/temp/app1

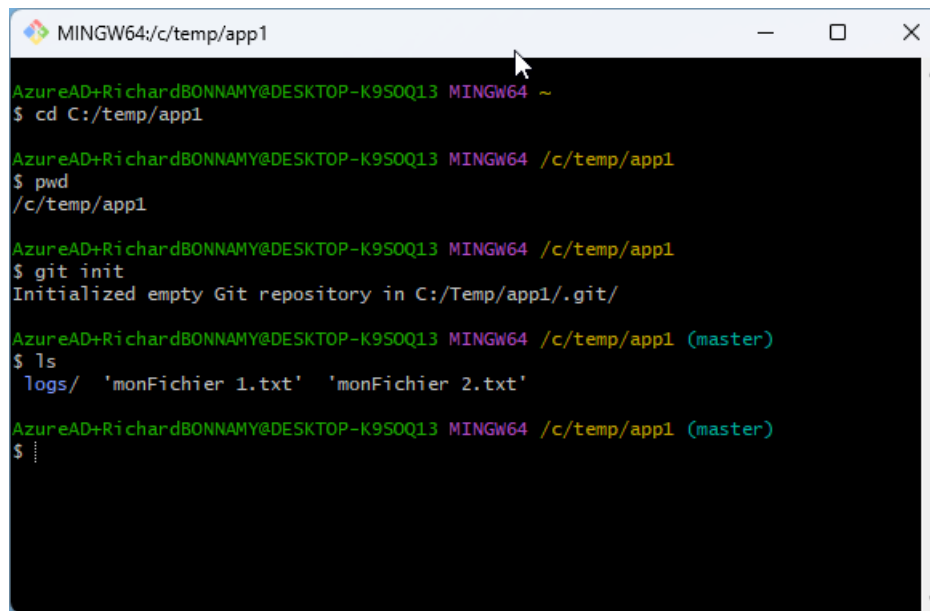
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1
$ git init
Initialized empty Git repository in C:/Temp/app1/.git/

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$
```

Git – Vérifiez le contenu de votre dépôt

➤ Etape 3 : visualiser le contenu du répertoire app1

- Tapez la commande **ls** pour lister les fichiers et répertoires



```
MINGW64/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 ~
$ cd C:/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1
$ pwd
/c/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1
$ git init
Initialized empty Git repository in C:/Temp/app1/.git/

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ ls
logs/  'monFichier 1.txt'  'monFichier 2.txt'

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$
```

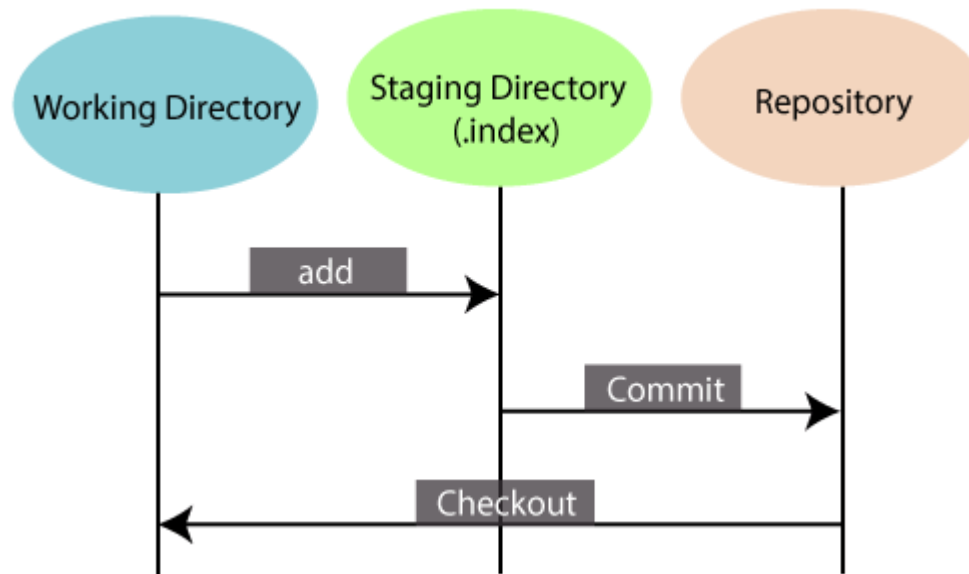
- On ne voit pas le répertoire **.git**
- Utilisez maintenant la commande ls avec l'option **-a** : **ls -a**

Git – Si vous vous êtes trompé de répertoire

- Que faire si vous avez créé le dépôt git au mauvais niveau ?
- Pas de panique, **ouvrez votre explorateur de fichiers**
- Allez dans le répertoire concerné et supprimer le répertoire **.git**
- Faire le ménage est indispensable si vous avez créé un dépôt au mauvais niveau.
- Par exemple si vous avez créé un **répertoire .git** dans **temp** et un autre **répertoire .git** dans **temp/app1**, git peut dysfonctionner.

Git – Comment versionner mon dépôt ?

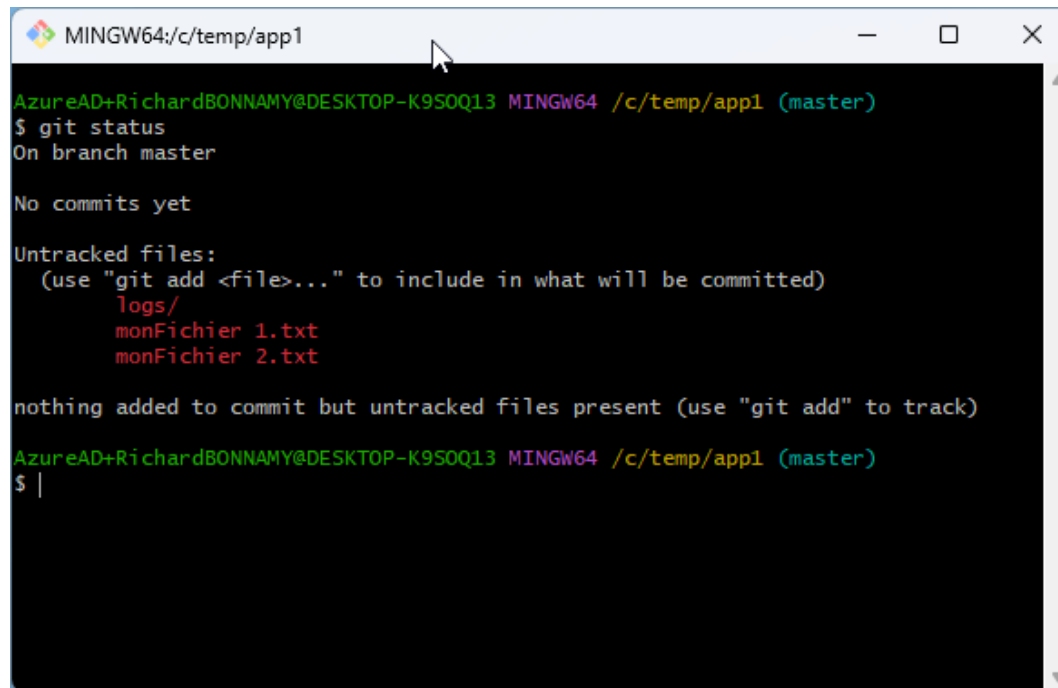
- Avant de réaliser une version (i.e. commit) vous devez sélectionner ce que vous voulez commiter.
- git demande à sélectionner les fichiers que vous voulez inclure à votre version.
- En effet il serait inconcevable de commiter systématiquement tout ce qui est présent dans le système de fichiers.
- Il va donc falloir **gérer** les fichiers à inclure dans votre version.



Git – Comment faire une version de mon nouveau dépôt ?

- La première chose qu'on va faire c'est de regarder le statut des différents fichiers et répertoires avec la commande :

git status



```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master

No commits yet

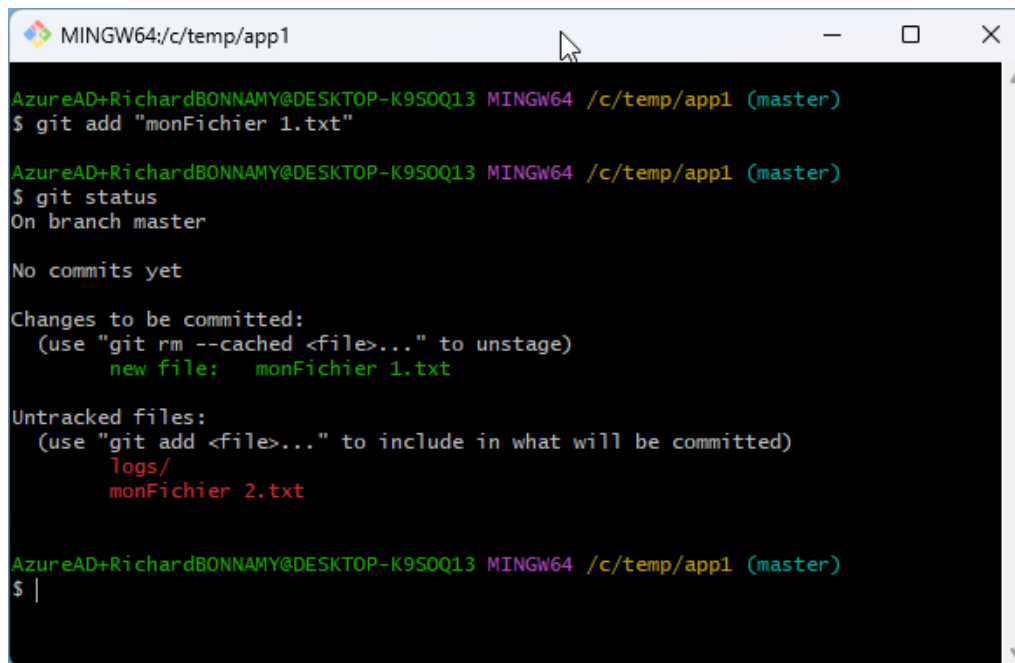
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    logs/
    monFichier 1.txt
    monFichier 2.txt

nothing added to commit but untracked files present (use "git add" to track)
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ |
```

- Nos fichiers sont "rouges" avec le statut "**untracked files**" qui correspond au statut par défaut de toute nouvelle ressource.

Git – Comment faire une version de mon nouveau dépôt ?

- La commande pour ajouter une ressource est : **git add *nom_ressource***
- **Exemple** : `git add "nomFichier 1.txt"`



```
MINGW64:/c/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ git add "monFichier 1.txt"

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   monFichier 1.txt

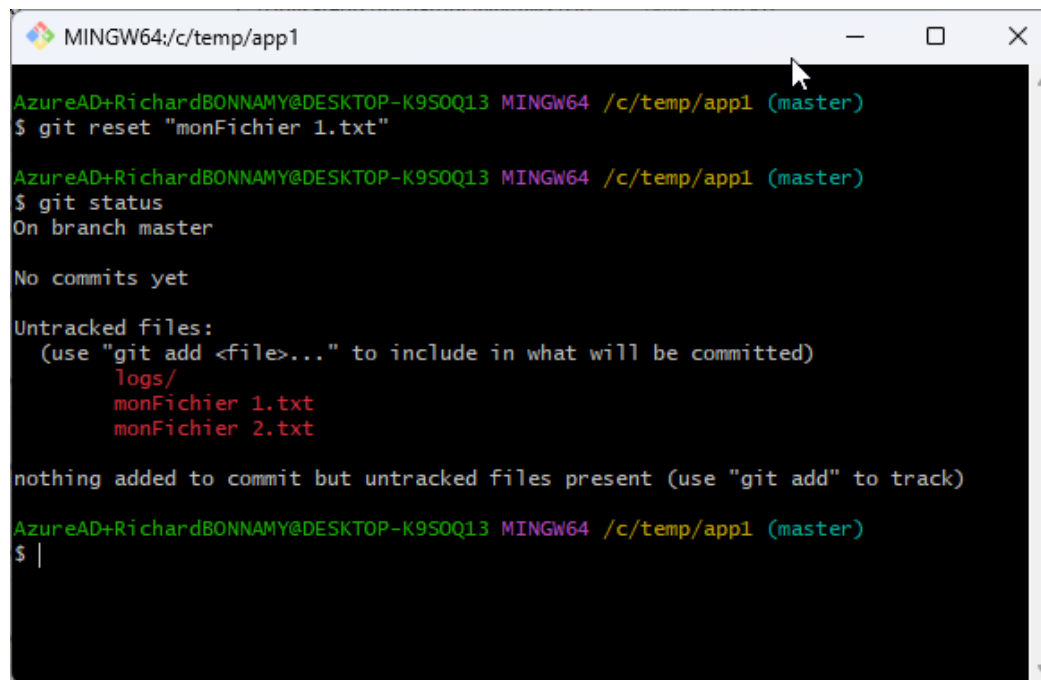
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        logs/
        monFichier 2.txt

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ |
```

- Lorsque je refais un **git status** on voit que **monFichier 1.txt** est passé en **vert** dans la rubrique "**Changes to be committed**".
- A ce stade si je fais un commit, seul ce fichier sera inclus dedans.

Git – Sortir un fichier de la rubrique "Changes to be committed"

- Si vous renoncez à commiter un fichier, utilisez : **git reset *nom_ressource***
- **Exemple** : `git reset "nomFichier 1.txt"`



```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ git reset "monFichier 1.txt"

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master

No commits yet

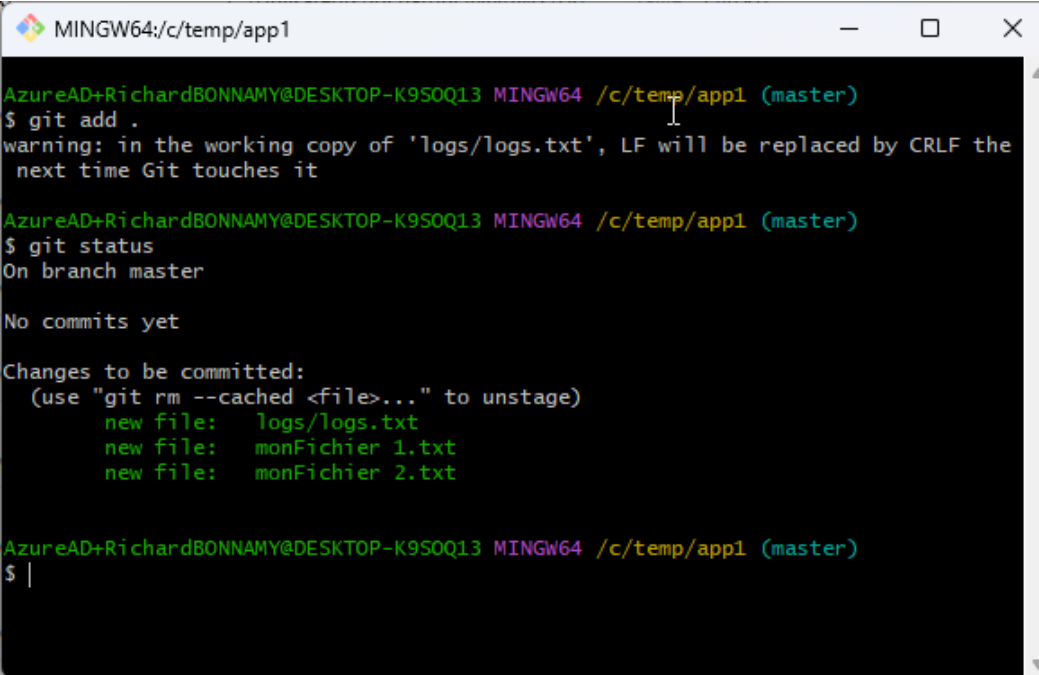
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    logs/
    monFichier 1.txt
    monFichier 2.txt

nothing added to commit but untracked files present (use "git add" to track)
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ |
```

- Lorsque je refais un **git status** on voit que **monFichier 1.txt** est repassé en rouge dans la rubrique "Untracked files".

Git – Ajoutons toutes les ressources

- Pour ajouter l'ensemble des ressources: **git add .**



```
MINGW64:/c/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git add .
warning: in the working copy of 'logs/logs.txt', LF will be replaced by CRLF the
next time Git touches it

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master

No commits yet

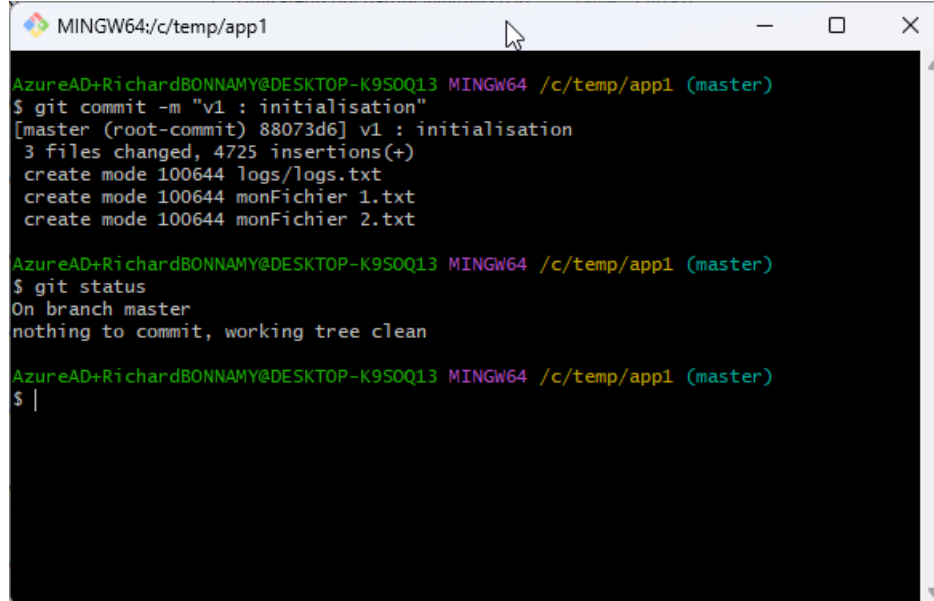
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   logs/logs.txt
        new file:   monFichier 1.txt
        new file:   monFichier 2.txt

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ |
```

- Avec cette commande toutes les ressources de l'arborescence sont ajoutées
- Lorsque je refais un **git status** on voit que **toutes les ressources** sont désormais dans la rubrique **"Changes to be committed"**

Git – Réaliser la version (enfin)

- Avec git, on parle de **commit** pour désigner une **version**.
- La commande que vous allons utiliser est : **git commit -m "Message"**
- Il est obligatoire d'associer un message à un commit.



```
MINGW64:/c/temp/app1

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git commit -m "v1 : initialisation"
[master (root-commit) 88073d6] v1 : initialisation
3 files changed, 4725 insertions(+)
create mode 100644 logs/logs.txt
create mode 100644 monFichier 1.txt
create mode 100644 monFichier 2.txt

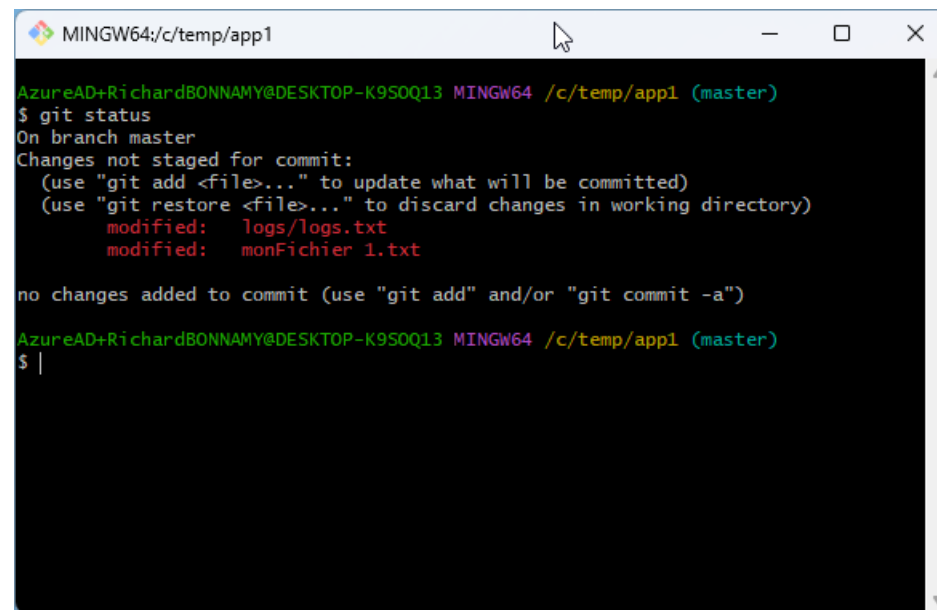
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master
nothing to commit, working tree clean

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ |
```

- Lorsque je refais un **git status** on voit le message "nothing to commit" s'afficher.

Git – Modifions maintenant 2 fichiers

- On va maintenant faire une modification dans **monFichier 1.txt** et dans **logs.txt**
- Refaisons un **git status**



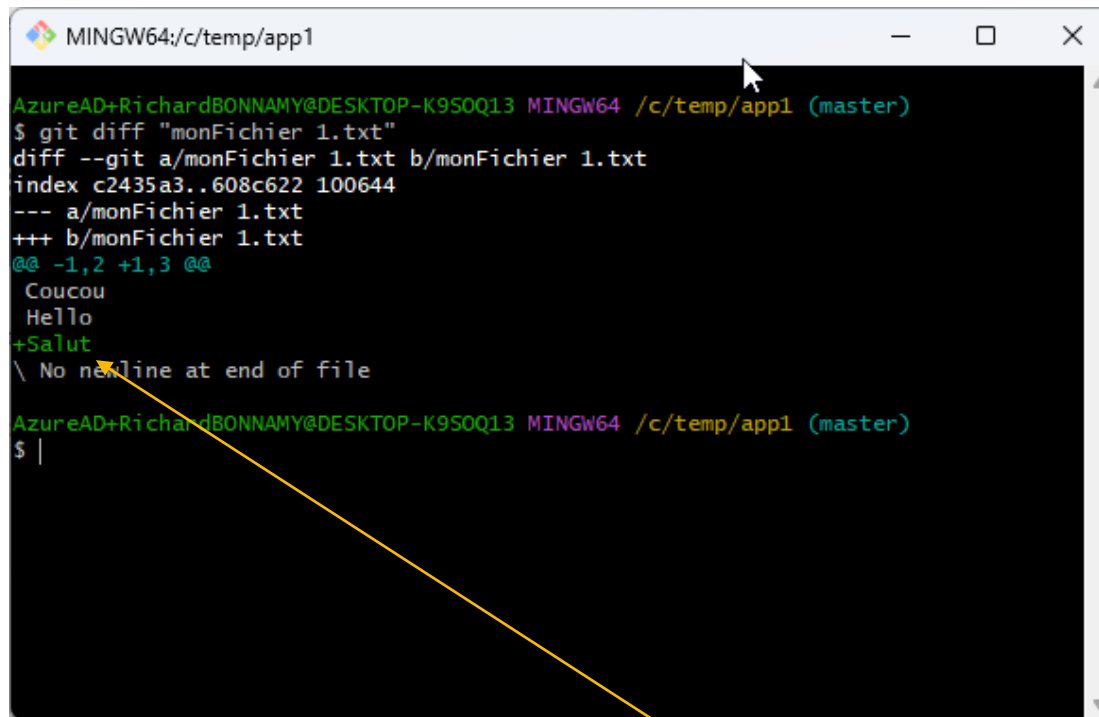
```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K950Q13 MINGW64 /c/temp/app1 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   logs/logs.txt
        modified:   monFichier 1.txt

no changes added to commit (use "git add" and/or "git commit -a")
AzureAD+RichardBONNAMY@DESKTOP-K950Q13 MINGW64 /c/temp/app1 (master)
$ |
```

- Dans ce cas les fichiers apparaissent au statut **modified** et non plus **untracked**.
- On note également que pour l'instant ils ne sont pas sélectionnés pour faire partie du prochain commit.

Git – Regardons les différences

- Je veux voir quelles sont les différences entre **monFichier 1.txt** actuel et celui du commit précédent
- Faisons un : **git diff "monFichier 1.txt"**



```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ git diff "monFichier 1.txt"
diff --git a/monFichier 1.txt b/monFichier 1.txt
index c2435a3..608c622 100644
--- a/monFichier 1.txt
+++ b/monFichier 1.txt
@@ -1,2 +1,3 @@
  Coucou
  Hello
+Salut
\ No newline at end of file

AzureAD+RichardBONNAMY@DESKTOP-K9SOQ13 MINGW64 /c/temp/app1 (master)
$ |
```

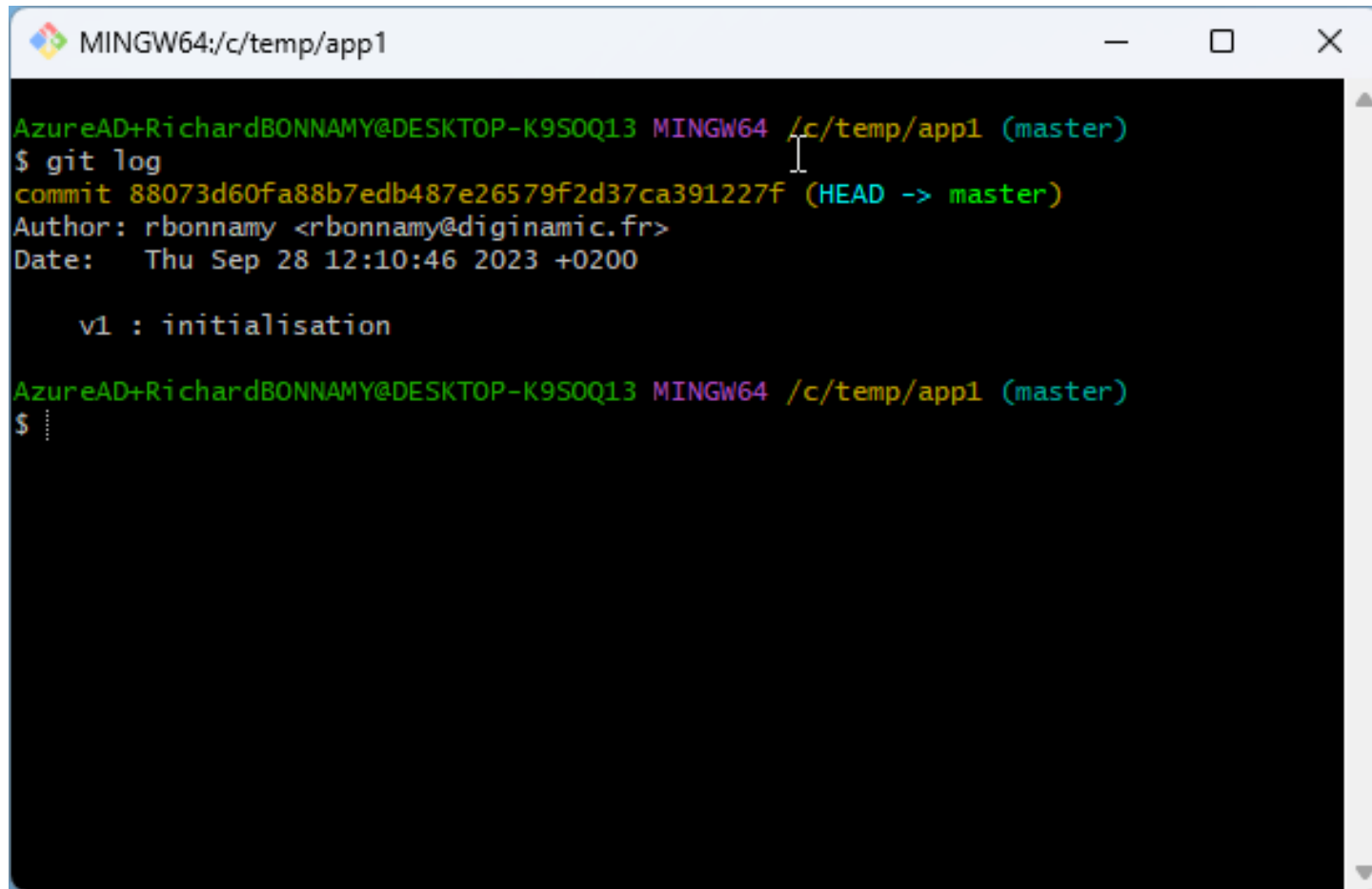
- On voit qu'une nouvelle ligne est apparue "+Salut"

Git – Regardons les différences

- Si je tape la commande **git diff** cela affiche les différences sur l'ensemble des ressources modifiées.
- Il existe de nombreuses options sur le git diff.

Git – Voir l'historique des commits

- Je peux avoir l'historique des commits avec : **git log**



```
MINGW64:/c/temp/app1
AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$ git log
commit 88073d60fa88b7edb487e26579f2d37ca391227f (HEAD -> master)
Author: rbonnamy <rbonnamy@diginamic.fr>
Date: Thu Sep 28 12:10:46 2023 +0200

    v1 : initialisation

AzureAD+RichardBONNAMY@DESKTOP-K9SQ13 MINGW64 /c/temp/app1 (master)
$
```



Ignorer certaines
ressources

Comment ignorer certaines ressources

➤ Problématique

- Je travaille sur un projet Java.
- Je peux commiter les fichiers sources mais pas les fichiers compilés.
- Je veux ignorer certains fichiers générés par mon IDE (outil de développement).

➤ Solution

- Demander à GIT d'ignorer certains fichiers, certaines extensions de fichiers ou même répertoires et contenus.
- Pour cela il faut créer un fichier **.gitignore** à la racine du projet.
 - **Exemple** : dans C:/temp/app1

Les patterns d'exclusion

➤ Les patterns d'exclusion par l'exemple

Exemple	Description
*.class	Exclusion de tous les fichiers d'extension .class
init.cfg	Exclusion du fichier init.cfg
/init.cfg	Exclusion du fichier init.cfg situé à la racine
config/	Exclusion du répertoire config
.idea/init.xml	permet d'exclure le fichier init.xml situé dans le répertoire .idea
*.log!important.log	Exclusion de tous les fichiers .log à l'exception de important.log



Travailler avec
GitHub

Travailler avec GitHub

➤ Pourquoi travailler avec GitHub ?

- Pour avoir des dépôts sécurisés et accessibles de partout (changement d'ordinateur, ordinateur en panne, etc.)
- Pour travailler en équipe

➤ Prérequis

- Vous devez avoir un dépôt local sur votre ordinateur
- Vous devez synchroniser régulièrement votre dépôt local avec votre dépôt distant pour :
 - Remonter les modifications que vous avez effectués
 - Récupérer les modifications effectuées par les membres de votre équipe.

➤ Comment synchroniser un dépôt local avec un dépôt distant ?

Cas n°1 : dépôt distant inexistant

- **Cas n°1:** le dépôt existe sur votre ordinateur mais pas sur GitHub. Vous voulez remonter ce dépôt sur GitHub
 - **Etape 1 :** Vous devez **créer le dépôt sur GitHub** avec le même nom que le dépôt local si possible.
 - Le nom du dépôt local = nom du répertoire
 - **Etape 2 :** Vous **ouvrez Git Bash** et vous utilisez la commande de changement de répertoire : **cd C:/temp/app1**
 - **Etape 3 :** Vous tapez la commande qui va permettre de lier votre dépôt local à votre dépôt distant: la commande **git remote**

git remote add origin <https://github.com/DiginamicFormation/app1.git>

git remote add origin <git@github.com:DiginamicFormation/app1.git>

Cas n°1 : quelques explications complémentaires

git remote add **origin** *mon_URL*

- **remote** signifie **distant**
- **add** va permettre de lier le dépôt local au dépôt distant. **add** ne fonctionnera pas si le dépôt local est déjà lié à un dépôt distant.
- Il existe donc **d'autres options** que **add** comme **set-url** qui permet de modifier le dépôt distant.
- **origin** est un **alias** que vous donnez à l'URL
<https://github.com/DiginamicFormation/app1.git>
- Cet alias permettra d'éviter de retaper l'URL pour les commandes ultérieures.
- Evitez de changer d'alias pour chaque dépôt. **origin** est une convention

Cas n°2 : dépôt local inexistant (1/2)

➤ **Cas n°2:** le dépôt existe sur GitHub mais pas sur votre ordinateur:

- Dans ce cas on n'utilise pas **git init**, on va utiliser un **git clone**.
- **git clone** va créer le dépôt en local et télécharger les sources.

git clone <https://github.com/DiginamicFormation/ressources-atelier.git>

- **git clone** va également effectuer la commande git remote puisqu'il a tous les éléments pour le faire. Par défaut l'alias utilisé pour l'URL est origin.

Cas n°2 : dépôt local inexistant (2/2)

➤ Les étapes:

- **Etape 1 :** Vous récupérez le lien du dépôt qui vous intéresse sur GitHub.
 - Exemple : <https://github.com/DiginamicFormation/ressources-atelier.git>
- **Etape 2 :** Vous choisissez un **répertoire d'accueil** de votre dépôt sur votre ordinateur **par exemple C:/temp**
- **Etape 3 :** Vous ouvrez Git Bash et vous utilisez la commande de changement de répertoire : **cd C:/temp**
- **Etape 4 :** Vous tapez la commande qui va permettre de télécharger le dépôt en local :

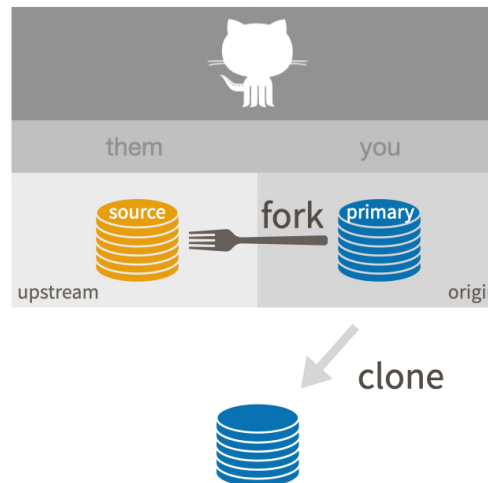
git clone <https://github.com/DiginamicFormation/ressources-atelier.git>

Les commandes de synchronisation

- **Consultez les dernières modifications sur le dépôt distant**
 - **git fetch** permet de consultez les dernières modifications sans les récupérer
- **Récupérez les dernières modifications sur le dépôt distant**
 - **git pull** permet de consultez les dernières modifications sans les récupérer
- **Envoyer les dernières modifications de votre dépôt local vers le distant**
 - **git push** permet d'envoyer les derniers commits vers le dépôt distant

Fork : Récupérez un projet sur un autre compte GitHub

- Vous pouvez cloner n'importe quel dépôt public trouvé sur GitHub
- Mais, si vous faites cela, vous ne pourrez pas synchroniser votre dépôt local avec le dépôt distant qui ne vous appartient pas.
- Vous devez faire un **fork** afin de **copier le dépôt distant** qui vous intéresse **sur votre compte GitHub**.
- Ensuite vous clonez à partir de votre compte GitHub
- **L'avantage du fork** est que vous pouvez proposer des commits que vous avez réalisés au dépôt d'origine avec un mécanisme de **pull requests**.





Travailler avec
des branches

Les branches

➤ Présentation

- Par défaut tout le code source est sur une **branche par défaut** qui s'appelle **master**.
- Vous pouvez très bien travailler avec git en ignorant ce mécanisme de branche mais quand on travaille à plusieurs c'est assez utile
- Une branche est une copie du master, sur laquelle une personne va pouvoir travailler en autonomie sans être perturbé.

➤ Bonne pratique d'utilisation

- Dépend de l'entreprise
- Il est recommandé de créer une branche à partir du master dès qu'on veut travailler sur une nouvelle fonctionnalité.
- Le propriétaire de la branche doit intégrer tous les jours les modifications réalisées sur le master.
- Régulièrement le propriétaire de la branche transfère ses modifications sur le master : phase de relecture par les pairs

Créer une branche

- La création d'une branche se fait via la commande :

git branch *nom_branche*

Changer de branche

- Le changement de branche se fait via la commande suivante:

git checkout *nom_branche*

Merger une branche sur le master

- **Etape 1: commiter** ce qui a été réalisé sur la **branche *nom_branche***
- **Etape 2: revenir** sur la **branche master**

git checkout master

- **Etape 3 :** une fois sur le master, lancer la commande de **merge**

git merge nom_branche

Identifier des conflits

➤ Qu'est-ce qu'un conflit ?

- Si quelqu'un a modifié un fichier sur le master et que vous avez également modifié ce fichier sur votre branche, l'opération de merge va générer un conflit.

➤ Comment identifier un conflit ?

- Tout d'abord utilisez la commande **git status** pour identifier les fichiers en conflits
- Exemple :

```
$ git status
> # On branch branch-fct001
> # You have unmerged paths.
> #   (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> #   (use "git add <file>..." to mark resolution)
> #
> # both modified:      monFichier 1.txt
> #
> no changes added to commit (use "git add" and/or "git commit -a")
```

Editer le fichier et identifier le conflit

- **Editer le fichier en conflit et chercher les marqueurs de conflit**
- **Exemple :**

```
Salut
<<<<<< HEAD
Bonjour tout le monde
Quel plaisir de se revoir
=====
Bonjour à tous
Quel plaisir de se revoir
>>>>>> branche_fct001
Au-revoir
A bientôt
Bla bla
```

- **La première partie** : située entre <<<<<< HEAD et ===== fait référence aux modifications commitées sur la branche master
- **La seconde partie** : située entre ===== et >>>>>> branch-fct001 fait référence aux modifications commitées sur la branche **branch-fct001**.

Résoudre un conflit

➤ Pour résoudre le conflit :

- éditer le fichier,
- choisissez les modifications à conserver : soit celles du master, soit celles de la branche, ou un mix des 2, à vous de voir.
- Supprimez les marqueurs de conflit
- Et enfin committez.
- **Exemple** : ici on a conservé les modifications effectuées sur la branche

```
Salut  
Bonjour à tous  
Quel plaisir de se revoir  
Au-revoir  
A bientôt  
Bla bla
```

Quelques commandes et points complémentaires

- La branche master est-elle obligatoire ?
 - Lorsque vous initialisez un dépôt, la branche principale s'appelle master par défaut
 - Vous pouvez parfaitement cloner un dépôt trouvé sur GitHub et dans lequel la branche principale s'appelle main.
 - On peut la renommer en main par exemple
- Comment renommer une branche ?

git branch -m master main

- Comment supprimer une branche ?

git branch -d ma_branche

FIN

MERCI DE VOTRE ATTENTION !