جامعة الحسن الثاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

FSBM
FACULTE DES SCIENCES BEN M'SICK
UNIVERSITE HASSAN II DE CASABLANCA

MODULE :

Le Cloud Computing & DevOps

Pr. F. Benabbou
Master DSBD
Faculté des Sciences Ben M'Sik Casablanca

TABLE OF CONTENTS

01

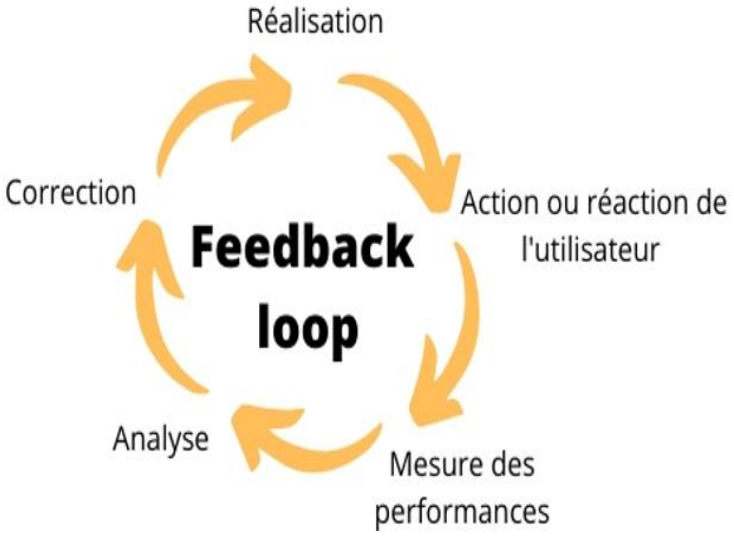
Développement Continue

Développement Continue

- Le développement continu est la première étape du cycle de vie DevOps et implique la planification et le codage des logiciels.
- À ce stade, le processus de développement complet est décomposé en cycles de développement plus petits.
- Cette phase est essentielle pour déterminer la vision de l'ensemble du cycle de développement et permettre aux développeurs de comprendre clairement ce qu'ils attendent d'un projet.

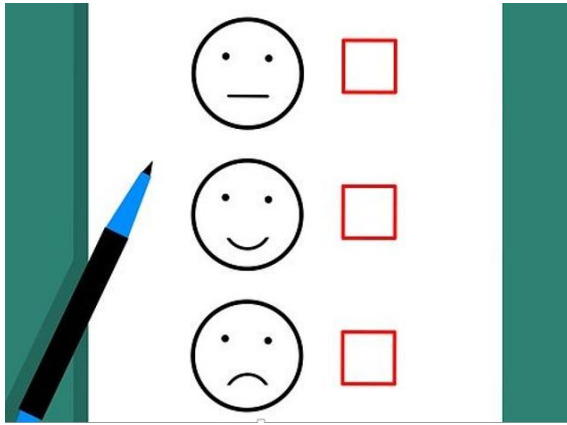
Développement Continue

- Dans la culture DevOps, la communication continue avec les utilisateurs est primordiale.
- Grace aux feedbacks des utilisateurs les logiciels sont constamment améliorés, mis à jour et déployés de manière itérative.
- Le développement continu est un processus dans lequel les logiciels ou les applications sont fréquemment améliorés, avec une plus grande rapidité et une meilleure réactivité aux besoins des utilisateurs et aux évolutions du marché.



Développement Continue

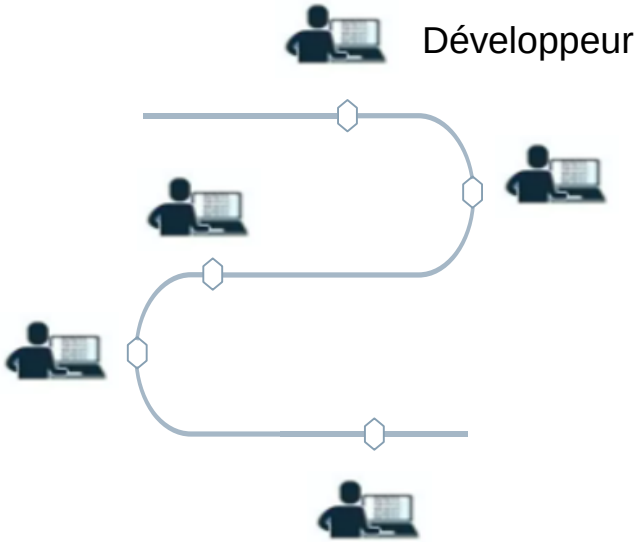
- Les utilisateurs finaux fournissent des retours essentiels qui guident l'évolution des applications, et permettent leur réajustent.
- L'analyse des données d’utilisation, l’évaluations des fonctionnalités permettent d’obtenir des informations précieuses sur l’avis des utilisateurs sur l’usage des applications produites
- Dans un environnement de développement continu, les tests sont automatisés et exécutés chaque fois qu'une modification est effectuée dans le code.
- Les tests peuvent être vus comme des "feedbacks techniques" sur la qualité du code développé.
- si un test échoue, il faut corriger les erreurs rapidement



Projet avec multi-contributeurs

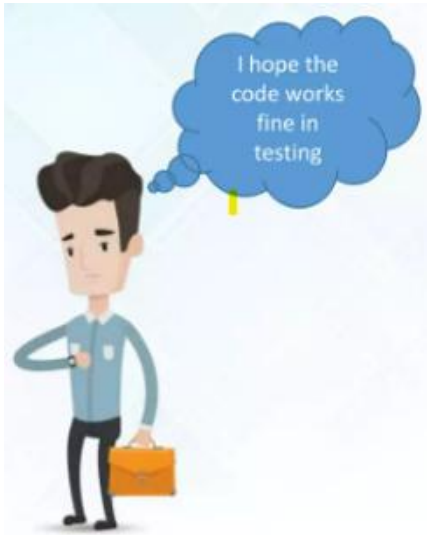
- Projet X : 5 développeurs, chacun travaille sur une fonctionnalité, comment regrouper les fonctionnalité et les intégrer dans le même projet sans perte de code

Projet divisé contenant plusieurs fonctionnalités

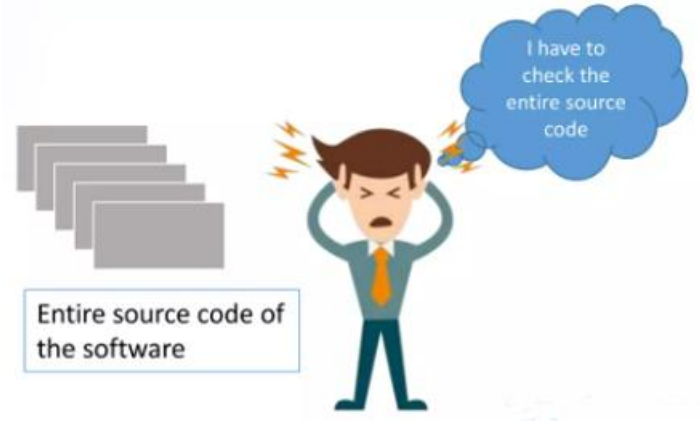


Projet avec multi-contributeurs

Développeur



Intégrateur



- Difficulté d'intégration
- Si une erreur est décelée, tout le code source du projet doit être analysé pour le corriger

Projet avec multi-contributeurs

- Sur quel ordinateur est stockée la copie « officielle » du projet ?
- Comment lire/écrire les modifications apportées par les diiferent membres de l’équipe?
- Commet gérer les autorisations pour les fichiers ?
- Que se passe-t-il si des membres essayent de modifier le même fichier ?
- Que se passe-t-il s’il y a eu erreur et un fichier important est corrompu ?
- Existe-t-il un moyen de conserver des sauvegardes des fichiers chaque membre du projet ?
- Comment savoir sur quel code chaque coéquipier travaille ?

Projet avec multi-contributeurs

- **Risque de conflit**
  - Lorsque plusieurs développeurs travaillent sur la même code projet, les risques de conflits sont élevés, surtout s'ils travaillent sur les mêmes fichiers ou fonctions.
  - Des conflits peuvent facilement survenir lorsque deux développeurs tentent de modifier la même partie du code.
- **Risque de perte**
  - Les développeurs doivent souvent gérer manuellement les modifications apportées à leur code
  - si quelqu'un supprime ou écrase accidentellement des parties critiques du code, le travail peut être perdu et il n'y a pas de moyen de le récupérer.
- **Difficulté de gérer différentes versions**
  - Durant l'avancement du projet, les développeurs peuvent avoir besoin de maintenir et de travailler sur différentes versions de l'application

Projet avec multi-contributeurs

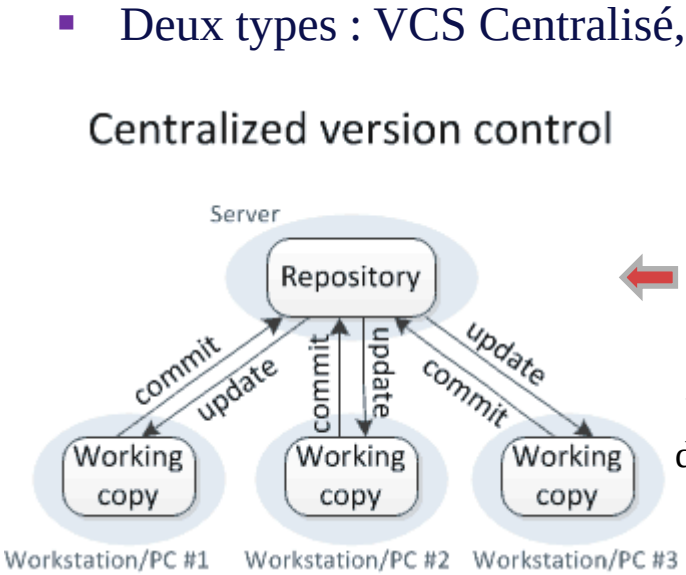
- **Pas de retour en arrière ou de récupération facile**
  - Il est souvent difficile de revenir à une dernière version stable en cas de problème.
  - les erreurs ou les problèmes introduits par un membre pourraient devenir très difficiles à résoudre, car l'historique des modifications n'est pas conservé.
- **Manque de visibilité sur les modifications**
  - S’il n'y a pas d'enregistrement automatique des modifications effectuées, de leur date et de leur raison, il est donc difficile de suivre l'évolution du projet ou de comprendre les raisons de certaines modifications.
- **Risque Constructions et déploiements incohérents**
  - il devient difficile de s'assurer que tous les développeurs travaillent avec la même version du code, ce qui entraîne des incohérences dans les constructions et des problèmes de déploiement.

Contrôle de versions

- Dans le développement Continue:
  - Il y a un besoin de méthodes pour conserver les versions actuelles de tous les fichiers et les sauvegardes des versions antérieures
  - D’avoir une copie partagée de tous les fichiers de code à laquelle tous les utilisateurs peuvent accéder
  - De voir quels fichiers ont été modifiés par d'autres, quand et de visualiser ces modifications et connaître les raisons
  - De gérer les conflits lorsque plusieurs utilisateurs modifient le même fichier
  - Se protéger lorsque les choses tournent inévitablement mal.
  - Etc.

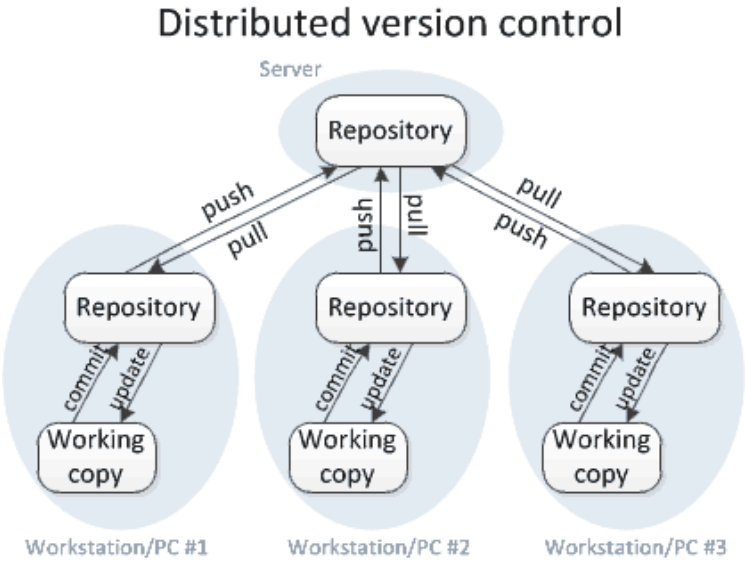
Solution : Contrôle de versions

- De nombreux systèmes de contrôle des versions (VCS) sont conçus et utilisés spécialement pour les projets de génie logiciel
- Deux types : VCS Centralisé, VCS distribué



1. Aucun dépôt ne fait Autorité
2. Un seule dépôt faisant autorité

Un **repository** est un espace de stockage où le code source d'un projet est conservé, versionné, et suivi.



Exemples: Concurrent Version System (CVS)
Subversion (SVN)

Exemples: Git, Mercurial

Système de Contrôle de versions

- Git est un système de contrôle de version distribué et open source, conçu pour la rapidité et l'efficacité.
- Git a été inventé par Linus Torvalds, le créateur du noyau Linux, en 2005.
- L’histoire de Git est étroitement liée à l'évolution du développement du noyau Linux et aux problèmes rencontrés avec le système de gestion de version précédent.

Système de Contrôle de versions

- Repository local :
  - C’est un dépôt Git sur la machine locale d'un développeur.
  - Il contient une copie complète du code source, ainsi que de son historique complet.
  - Un repository local permet aux développeurs de travailler de manière autonome et de synchroniser leurs modifications avec un repository distant lorsqu’ils le souhaitent.
- Repository distant :
  - Un repository Git stocké sur un serveur ou une plateforme comme GitHub, GitLab, ou Bitbucket.
  - Il permet aux développeurs de partager leur travail, de collaborer, et de gérer des projets à grande échelle.
  - Le repository distant sert souvent de source principale pour les équipes qui collaborent.

Concepts de base

- **Version**
  - C’est un dépôt Git sur la machine locale d'un développeur.
- **Commit**
  - Un repository Git stocké sur un serveur ou une plateforme comme GitHub, GitLab, ou Bitbucket.
  - Il permet aux développeurs de partager leur travail, de collaborer, et de gérer des projets à grande échelle.
- **Fusion (Merge)**
  - consiste à combiner les modifications effectuées sur une branche dans une autre, pour intégrer les fonctionnalités développées dans des branches secondaires à la branche principale.
- **Branche** est une ligne indépendante de développement qui permet de travailler sur différentes fonctionnalités ou corrections sans affecter la version principale du projet.

Concepts de base: branche (Branch)

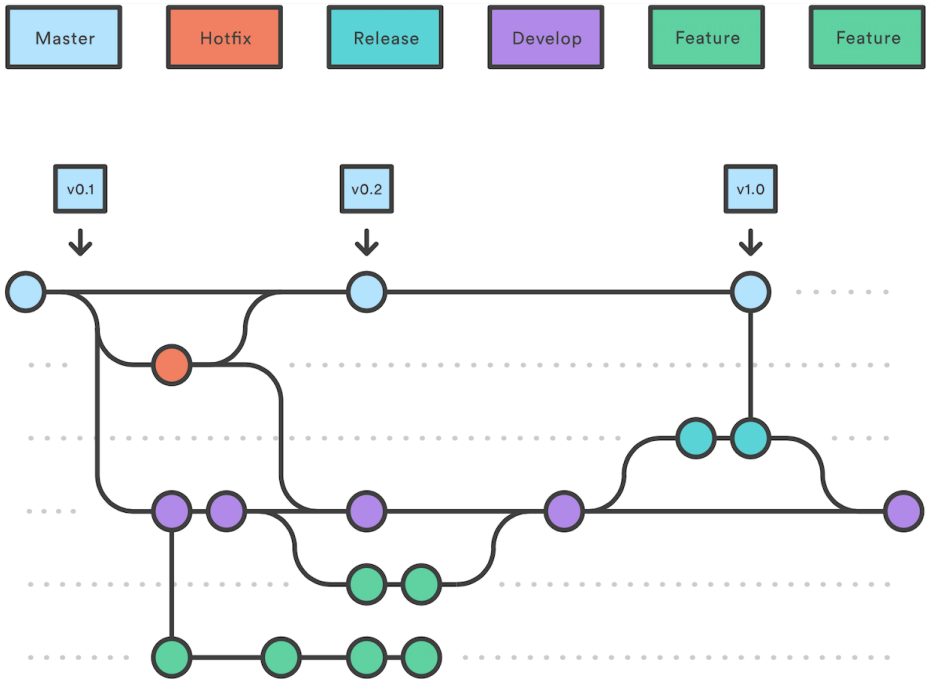
- Il y a deux types
  - Branche **principale** (main ou master)
    - ✓elle doit être la branche la plus stable.
    - ✓Chacune des modifications (effectuée avec commit) de cette branche est une nouvelle **version**
    - ✓ les développeurs ne travaillent jamais directement à partir de la branche master
  - Branches **secondaires** : personnalisables, elles sont utilisées pour des développements spécifiques (ex. develop, feature, release).

Concepts de base: branche (Branch)

- Git ne propose pas directement des branches avec des nom spécifiques, les noms et les structures des branches sont totalement personnalisables.
- Afin de structurer le développement un modèle de workflow peut être utilisé appelé Git Flow.
- Git Flow est une convention de gestion des branches, qui automatise ce processus.

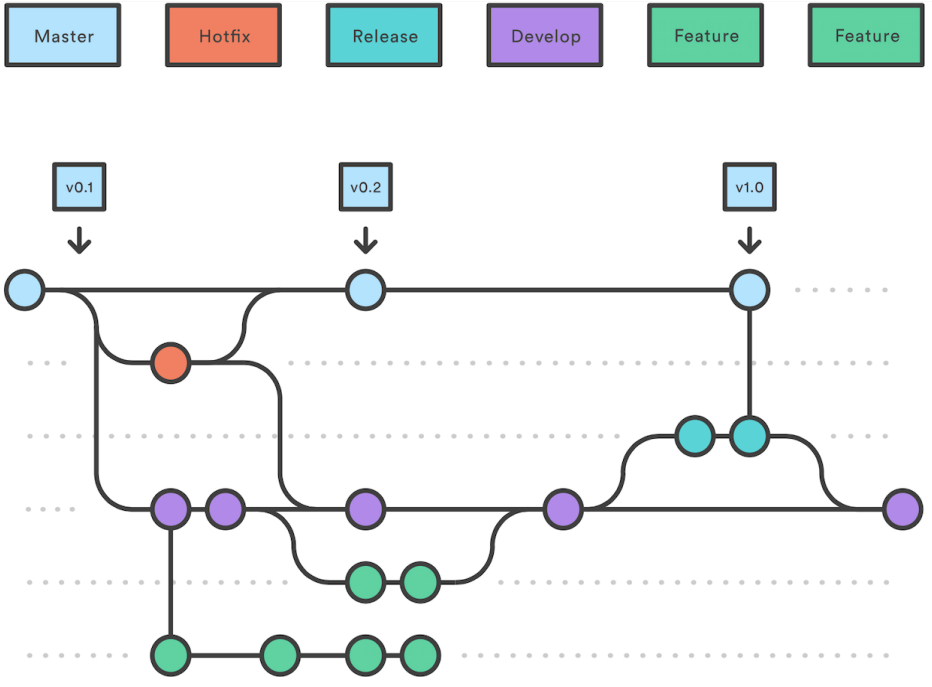
Concepts de base: branche (Branch)

- La branche **Develop** contient l'historique complet du code et intègre toutes les modifications, mineures ou majeures.
- Pour garantir sa stabilité, seules des corrections de bugs ou des améliorations mineures doivent y être effectuées directement.
- Lors du développement de nouvelles **fonctionnalités**, des conflits peuvent survenir lorsque plusieurs développeurs modifient les mêmes parties du code.
- C’est là qu’intervient la branche **feature**.



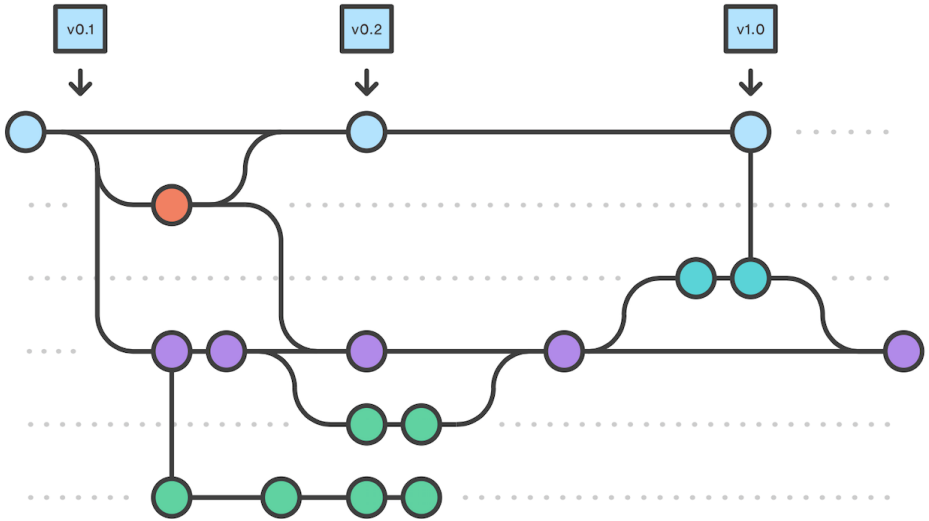
Concepts de base: branche (Branch)

- La branche **Feature** est utilisée pour développer de nouvelles fonctionnalités et réduire les conflits, car ils ne surviennent qu'au moment de fusionner (merge) la branche Feature dans Develop.
- La branche **Release** sert à stabiliser le code avant une mise à jour majeure.
  - Une fois assez de fonctionnalités ajoutées à Develop, une branche Release est créée pour corriger les bugs sans ajouter de nouvelles fonctionnalités.
  - Une fois stable, elle est **fusionnée** à la fois dans Master et Develop pour garantir la cohérence des corrections,



Concepts de base: branche (Branch)

- La branche **Hotfix** est destinée à corriger rapidement des bugs critiques directement sur la branche Master.
- Elle est peu utilisée mais essentielle pour des correctifs urgents.

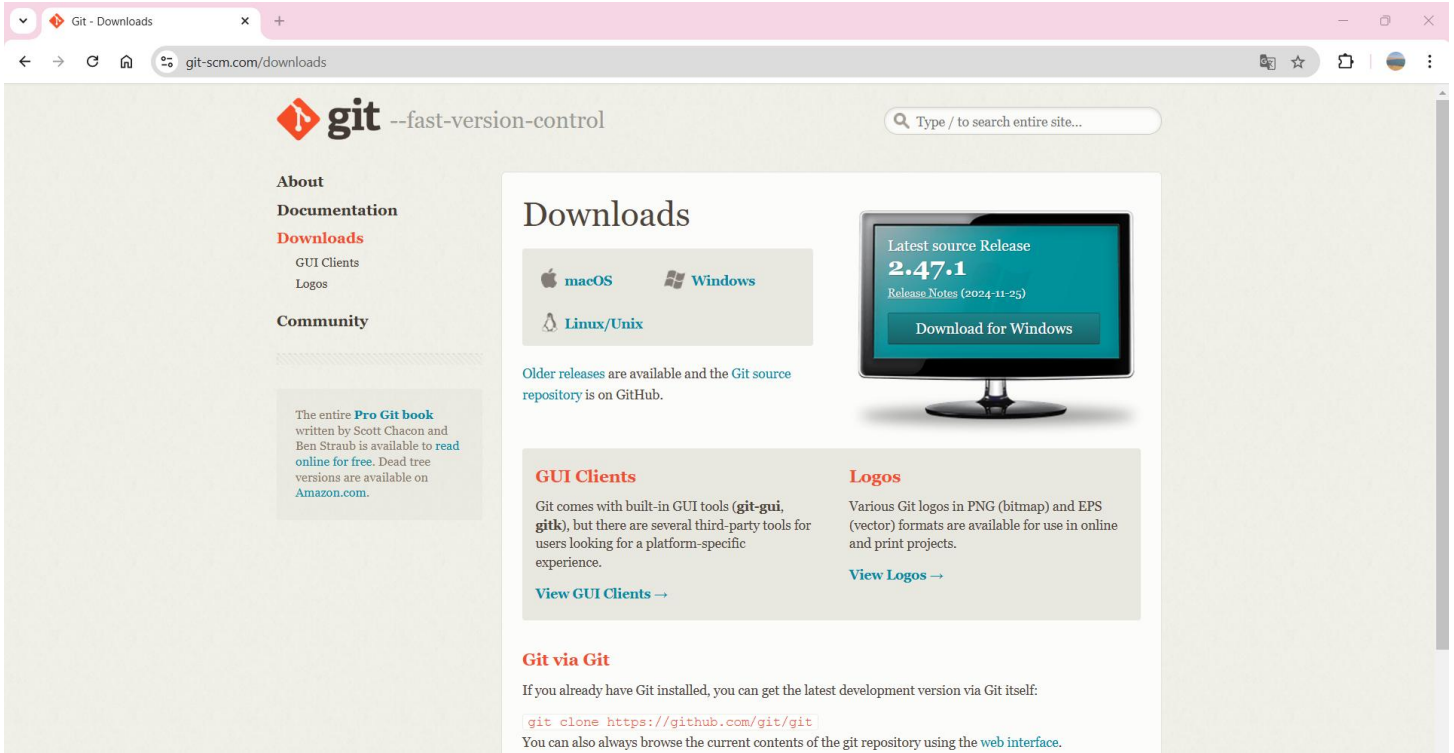


Concepts de base: branche (Branch)

- La branche **Hotfix** est destinée à corriger rapidement des bugs critiques directement sur la branche Master.
- Elle est peu utilisée mais essentielle pour des correctifs urgents.

Installer git

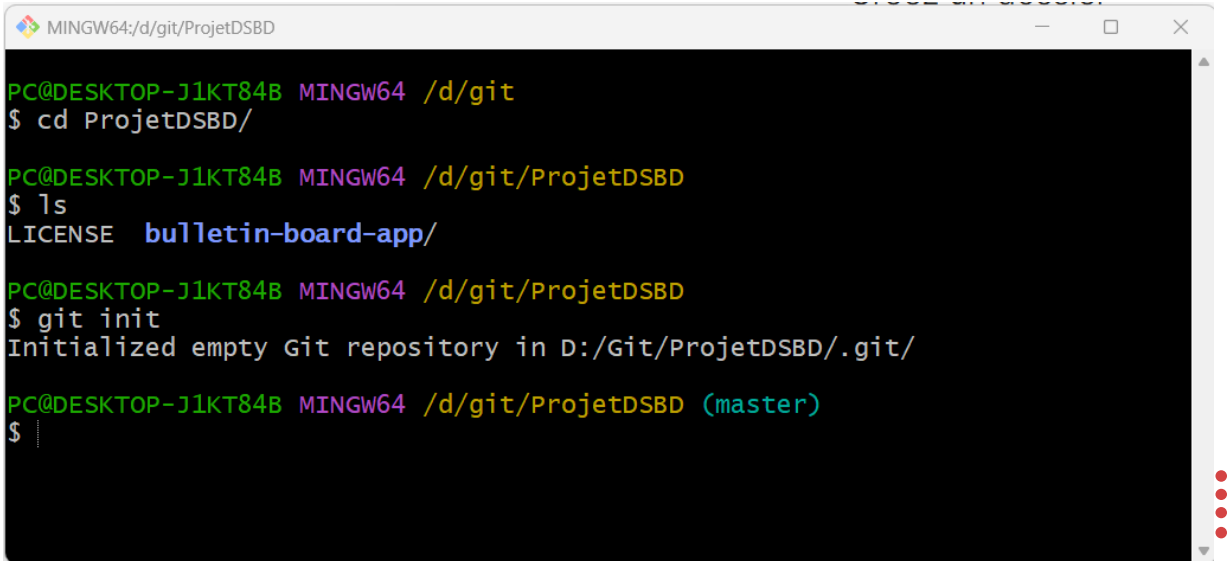
http://git-scm.com/downloads



Comment ca marche?

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

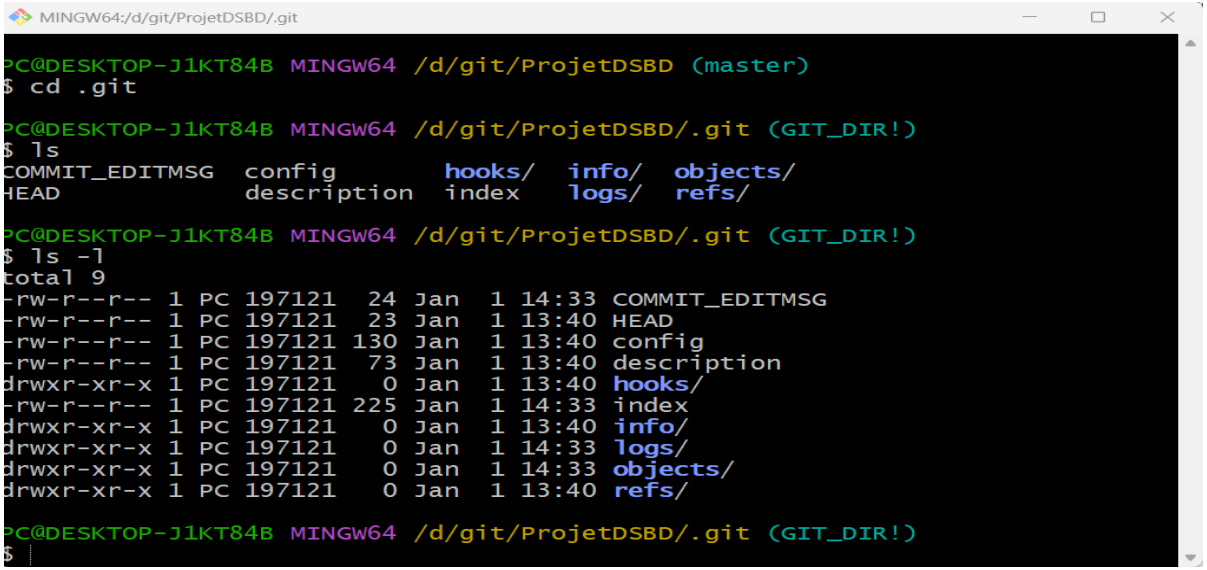
$ git init



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

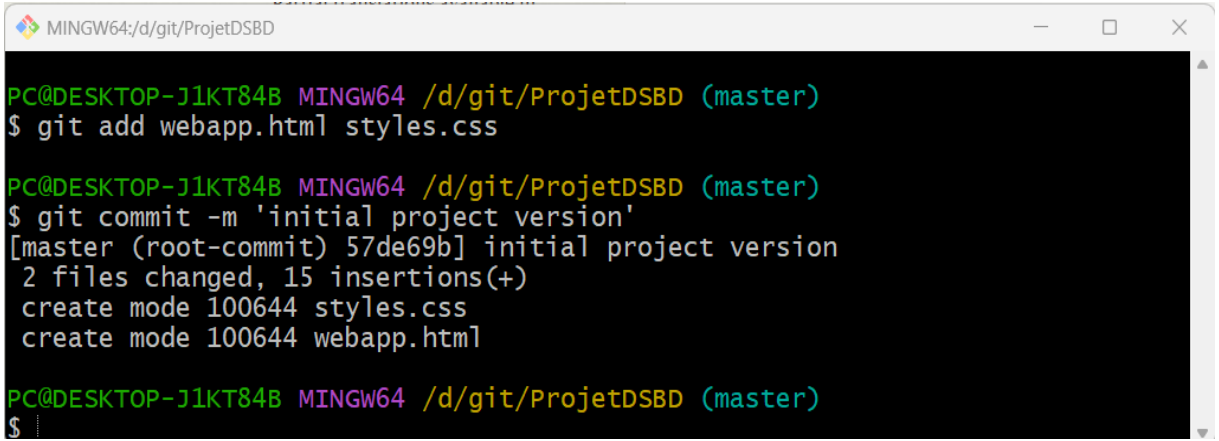
- Un fichier caché est crée .git qui contient tous les fichiers nécessaires au dépôt



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

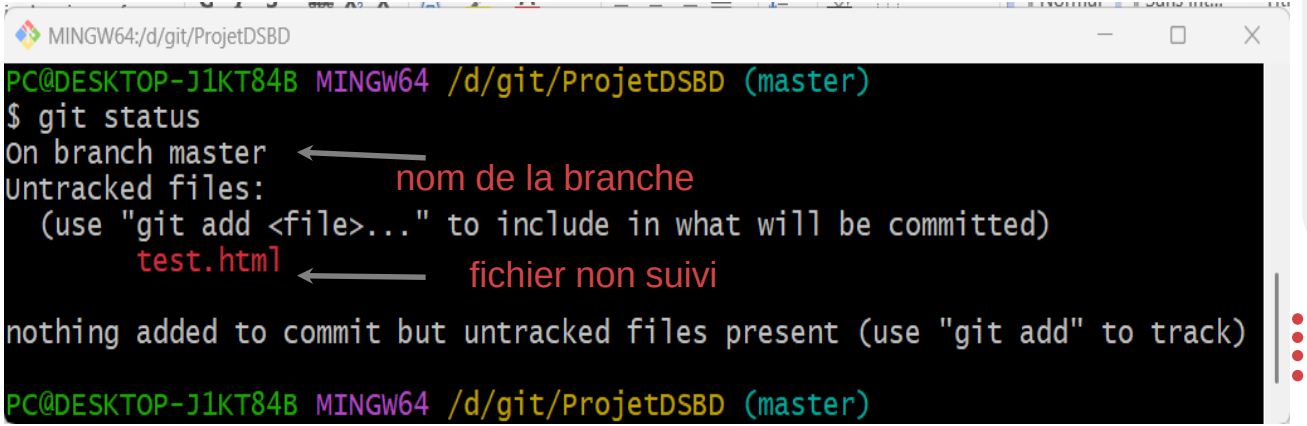
$ git add *source*
$ git commit



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

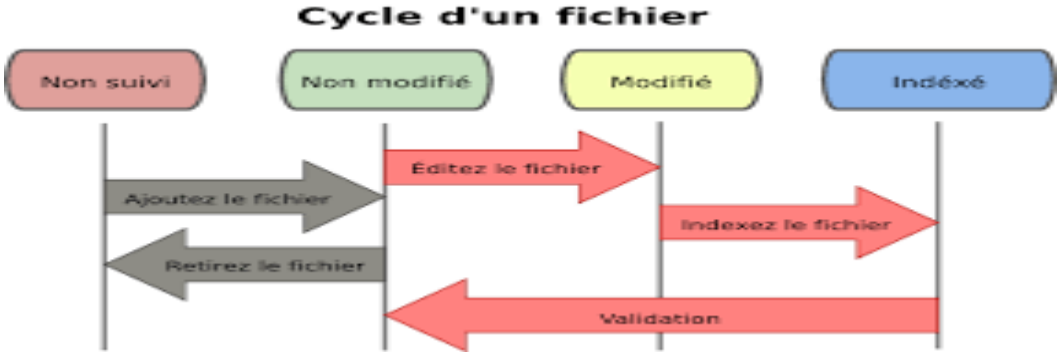
- Ajouter un fichier quelconque dans le répertoire
- La command « status » permet de voir l'état des fichiers suivis et non suivis.



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

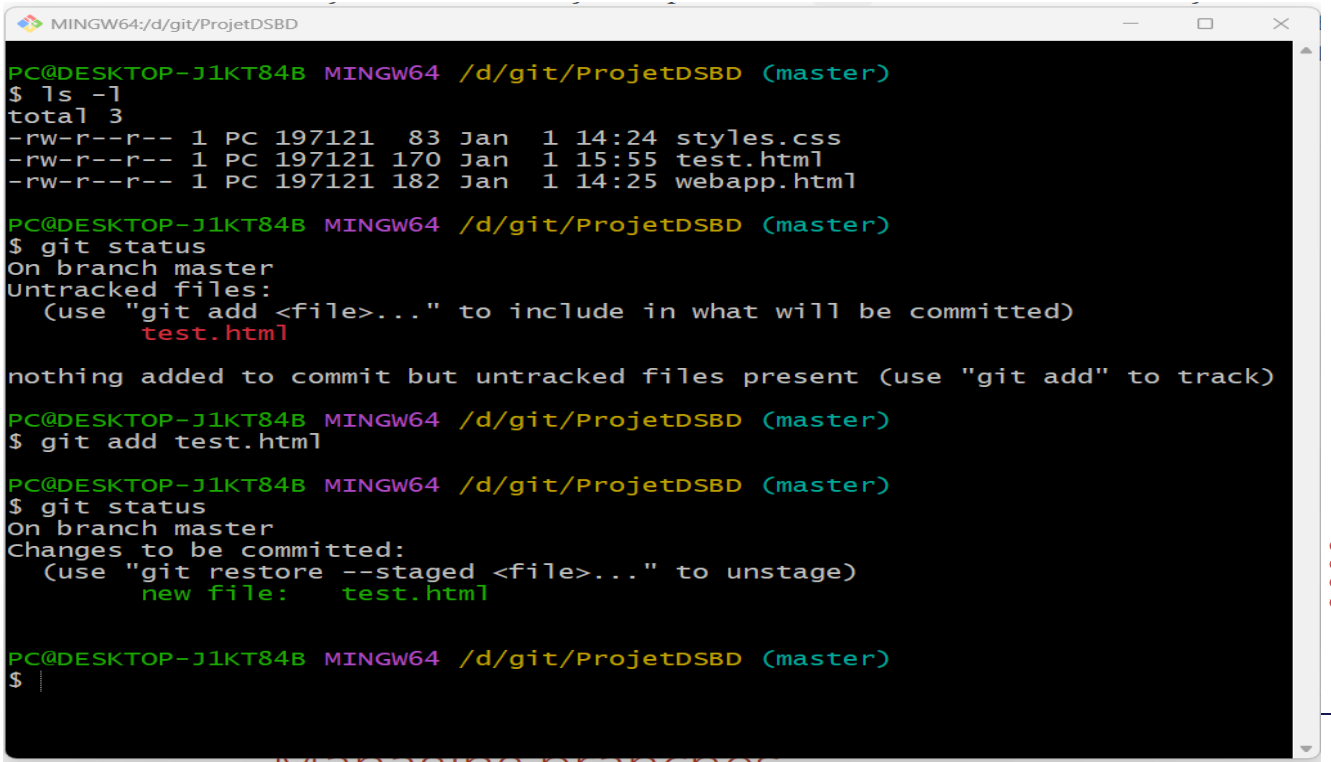
- **Untracked** (Non suivi) : Le fichier existe dans le répertoire de travail mais n'est pas encore suivi par Git.
- **Staged** (Indexé) : Le fichier a été ajouté à l'index avec la commande git add mais pas encore inclus dans un commit.
- **Non modifié ou Commited**: Le fichier a été enregistré dans l'historique du dépôt (dépôt local). Il est maintenant versionné et sa version actuelle est suivie par Git.
- **Modified** : Fichier est modifié mais non indexé.



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Exemple avec le fichier test.html.



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Annuler l'indexation du fichier test.html.

```
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ git reset

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.html

nothing added to commit but untracked files present (use "git add" to track)

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$
```

Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- git reset --soft HEAD~1 : annule le dernier commit mais conserve les changements dans l'index (comme s'ils étaient toujours staged).
- git reset –hard :Toutes les modifications non sauvegardées sont perdues.

```
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ git reset

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.html

nothing added to commit but untracked files present (use "git add" to track)

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$
```

Commandes Git

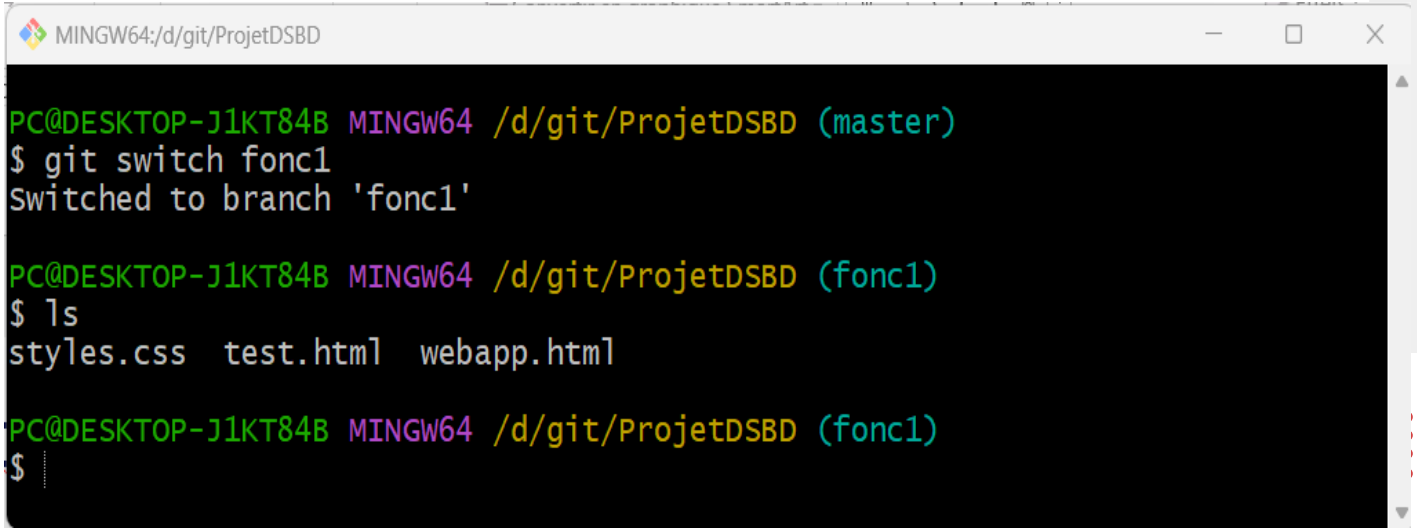
- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Créer une nouvelle branche : $git branch *nom*

Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

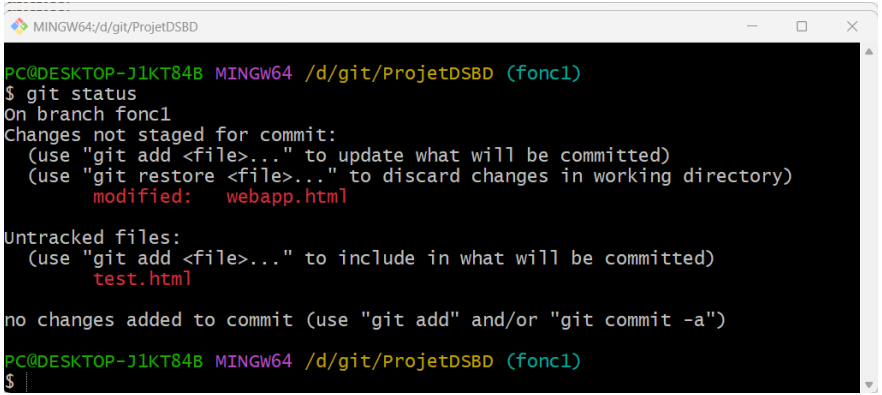
- Pour aller sur la branche fonc1on utilise : $ git switch fonc1



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Nous allons éditer et modifier le fichier webapp



-a permet de de ne pas utiliser le git add

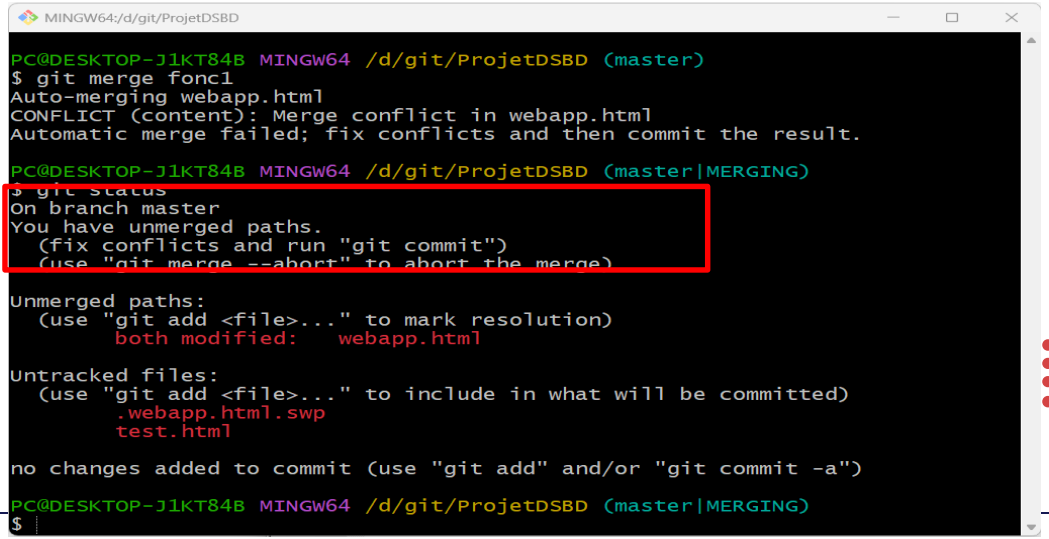
- Nous allons faire un commit

```
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (fonc1)
$ git commit -a -m "version0.1 avec un tableau"
[fonc1 52ddfff] version0.1 avec un tableau
 1 file changed, 7 insertions(+), 1 deletion(-)

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (fonc1)
$
```

Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

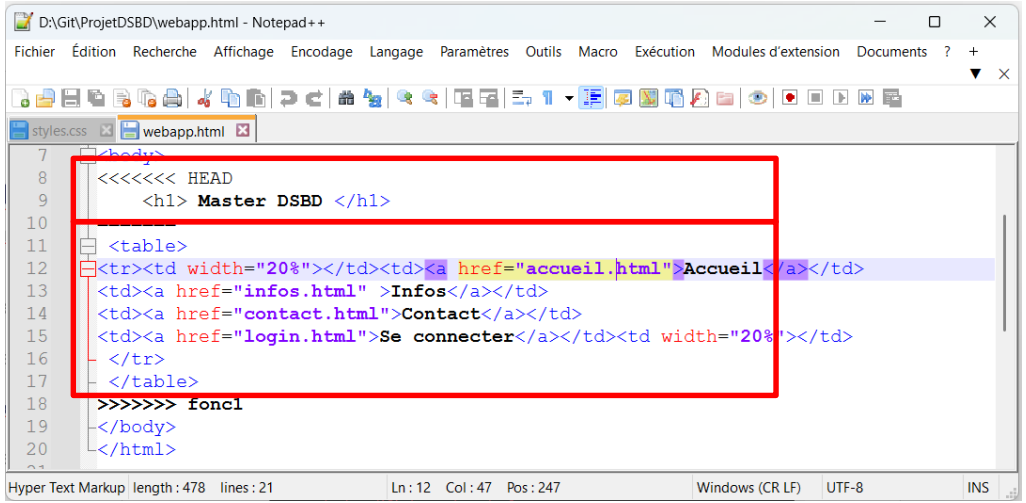
- Nous avons deux branches ont divergentes, avec des modifications différentes sur le même fichier.
- On va essayer de fusionner les changements effectués dans fonc1 avec master.



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- À ce niveau il faut résoudre le conflit et faire un add, commit



Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Revenir à la branche principale et modifier le fichier webapp

Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

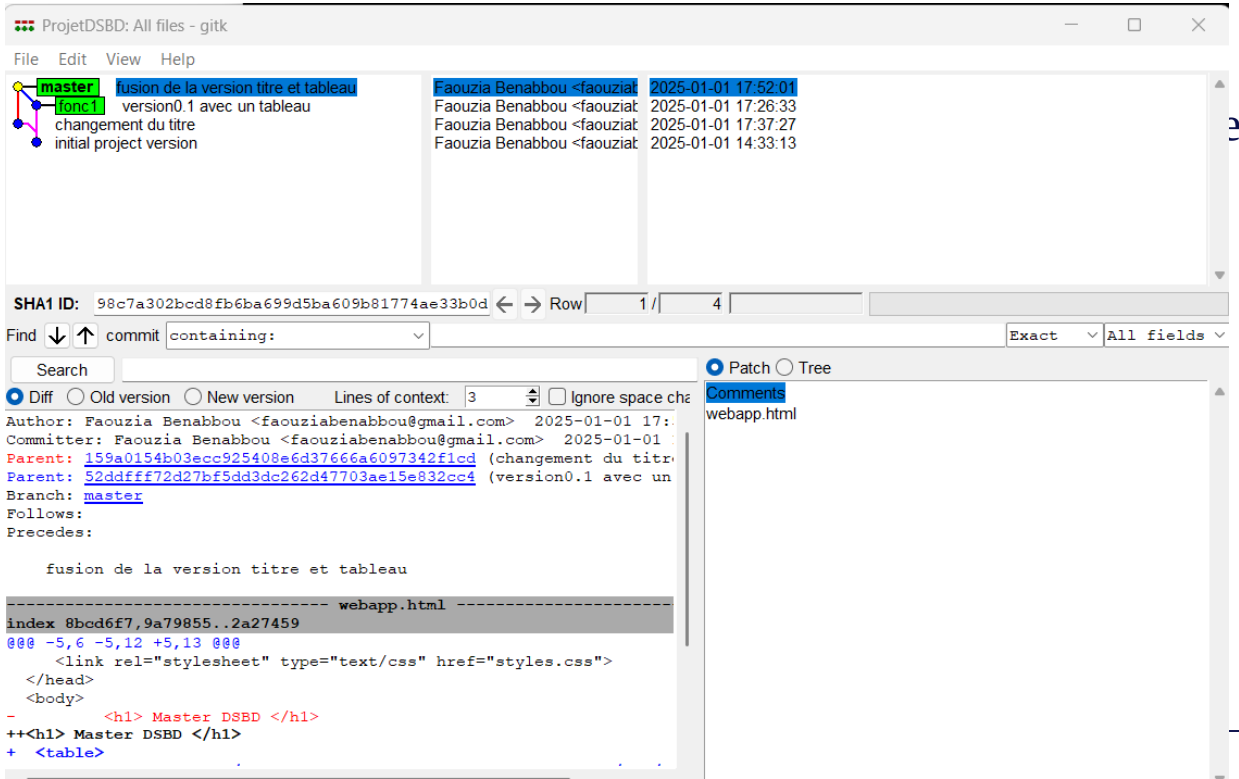
- La Log est un outil important pour analyser l'historique

$ git log --oneline --graph --decorate --all

- Cela affiche les commits sous forme abrégée, avec une vue graphique, les noms des branches et de tous les commits.

Commandes Git

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo



Travailler en équipe

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- git clone <URL_du_dépôt>

```
MINGW64:/d/git/ProjetDSBD

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ git clone https://github.com/faouziabenabbou/Syst-me-expert-Diagnos
tic.git
Cloning into 'Syst-me-expert-Diagnostic'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 13 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0
)
Receiving objects: 100% (13/13), 16.75 KiB | 816.00 KiB/s, done.
Resolving deltas: 100% (3/3), done.

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD (master)
$ ls
Syst-me-expert-Diagnostic/  styles.css  test.html  webapp.html
```

GitHub : travailler en équipe

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Modifier (mineur) le fichier mycin.pl

```
MINGW64:/d/git/ProjetDSBD/Syst-me-expert-Diagnostic
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$ vim mycin.pl

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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:   mycin.pl

no changes added to commit (use "git add" and/or "git commit -a")

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$
```

```
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$ git commit -a -m "Amélioration de l'intéraction"
[main 1e19620] Amélioration de l'intéraction
 1 file changed, 1 insertion(+), 1 deletion(-)

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$
```

GitHub : travailler en équipe

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

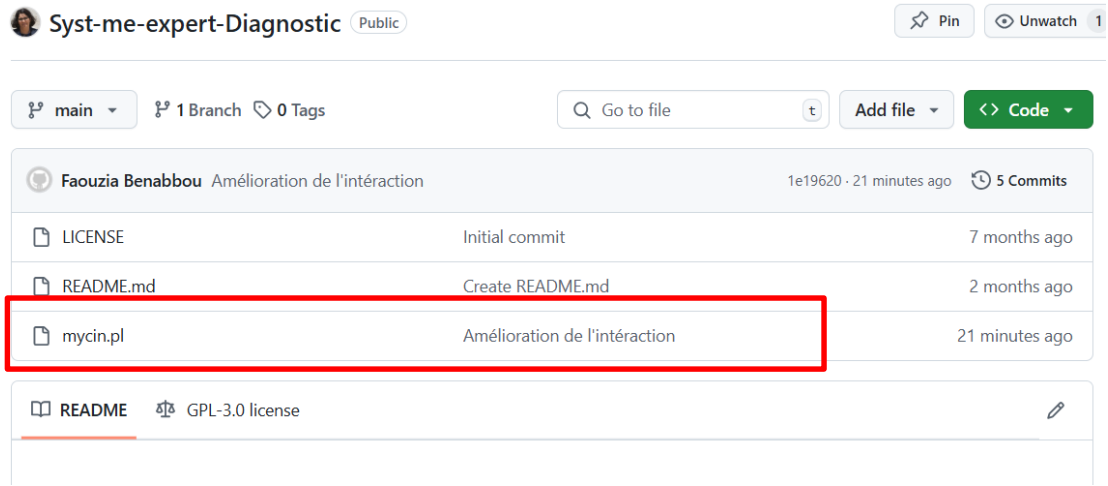
- Utiliser “$git push origin main” pour pousser les modification sur github (j’ai désactivé la verification par ssh!)

```
MINGW64:/d/git/ProjetDSBD/Syst-me-expert-Diagnostic
PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$ git config --global http.sslVerify false

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$ git push origin main
warning: ----------------- SECURITY WARNING ----------------
warning: | TLS certificate verification has been disabled! |
warning: ---------------------------------------------------
warning: HTTPS connections may not be secure. See https://aka.ms/gcm/tlsverify for more info
rmation.
info: please complete authentication in your browser...
warning: ----------------- SECURITY WARNING ----------------
warning: | TLS certificate verification has been disabled! |
warning: ---------------------------------------------------
warning: HTTPS connections may not be secure. See https://aka.ms/gcm/tlsverify for more info
rmation.
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/faouziabenabbou/Syst-me-expert-Diagnostic.git
   979cc22..1e19620  main -> main

PC@DESKTOP-J1KT84B MINGW64 /d/git/ProjetDSBD/Syst-me-expert-Diagnostic (main)
$
```

GitHub : travailler en équipe

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- On vérifie sur GitHub



GitHub : travailler en équipe

- Créer un nouveau repo local
- Add et Commit
- Cycle de vie
- Branche/fusion
- Inspecter/Log
- Cloner un Repo

- Les collaborateurs doivent executer un pull pour avoir la dernière version modifiée.

$git pull origin main