

# Partie 4 : Mettre en œuvre des outils de gestion de versions et de mesure de la qualité du code

**Filière : Développement digital – option web full stack**

**Module : Approche agile**

Rédigé par : M. Haij Oussama & M. Goumih Mohamed

Révisé par: Mme Laouija Soukaina

Adapté par : Mme YOUALA Asmae

# PARTIE 4

## Mettre en œuvre des outils de gestion de versions et de mesure de la qualité du code

Dans ce module, vous allez :

- Manipuler les outils de gestion de versions (Git/Gitlab) .
- Manipuler l'outil de mesure de la qualité du code (SonarQube)



**30 heures**



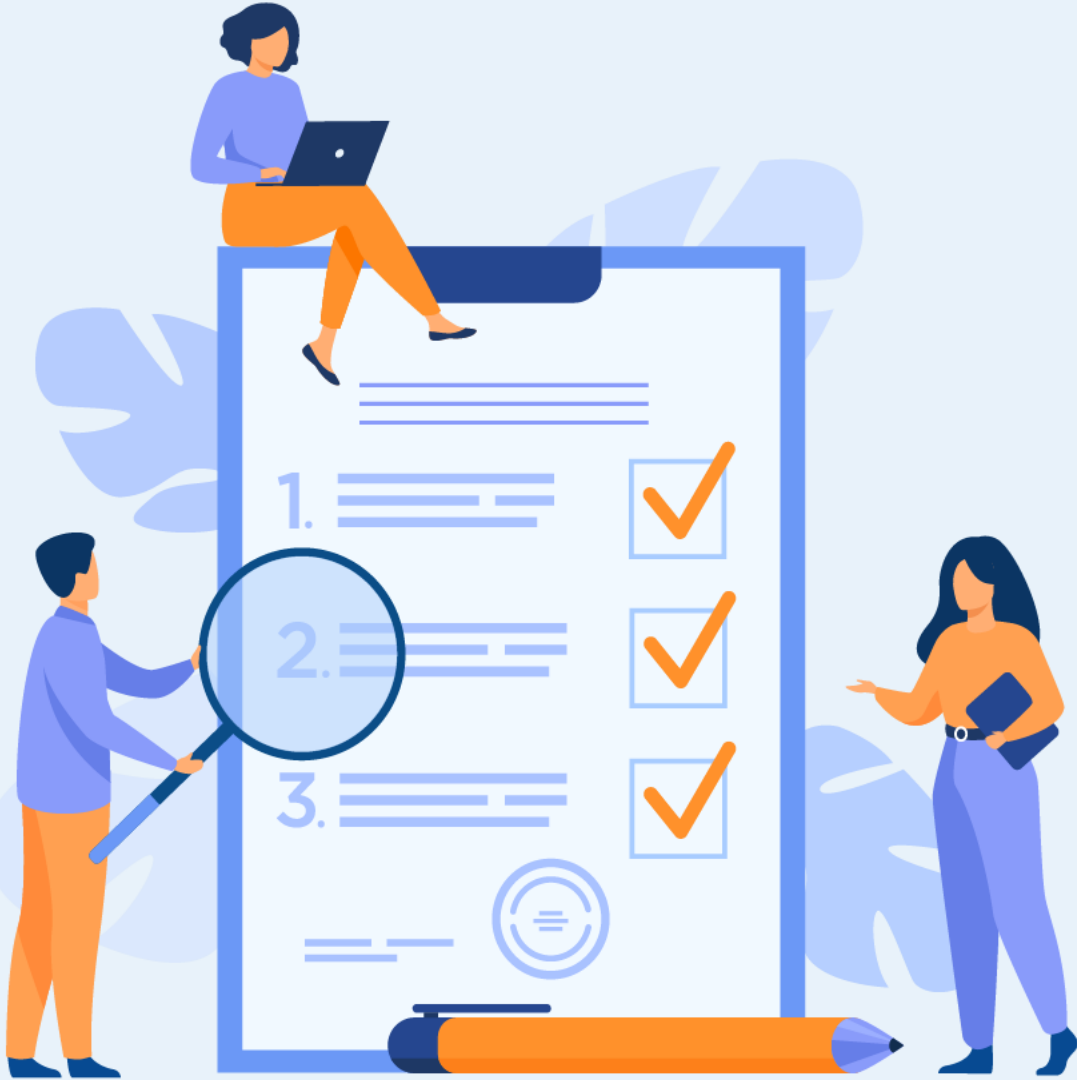
# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

Ce que vous allez apprendre dans ce chapitre :

- Présentation des outils existants de gestion
- Présentation de Git et Gitlab et comparaison entre les deux
- Présentation des fonctionnalités de Gitlab
- Installation et manipulation de Git.
- Manipulation des commandes de base de Git (Git bash)
- Notion de branches et gestion des conflits de fusion avec Git

**15 heures**



# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. **Intérêt de la gestion de version et présentation des outils existants de gestion de versions**
2. Présentation de Git
3. Présentation de Gitlab
4. Manipulation des dépôts avec Gitlab
5. Gestion des conflits de fusion avec Git/GitLab
6. Comparaison Git vs Gitlab

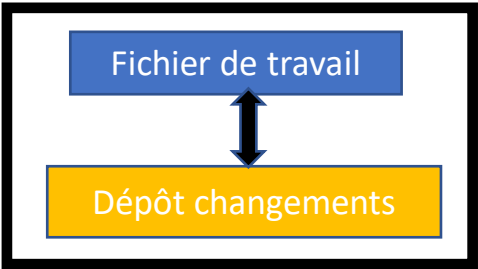


# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

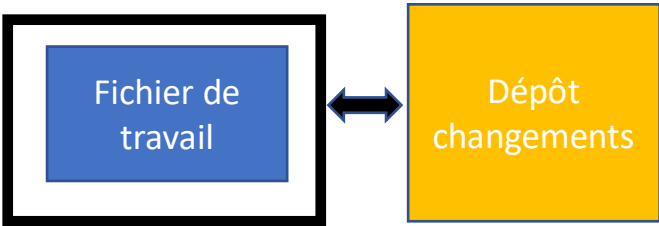
## Gestion de versions

### C'est quoi la gestion de versions?

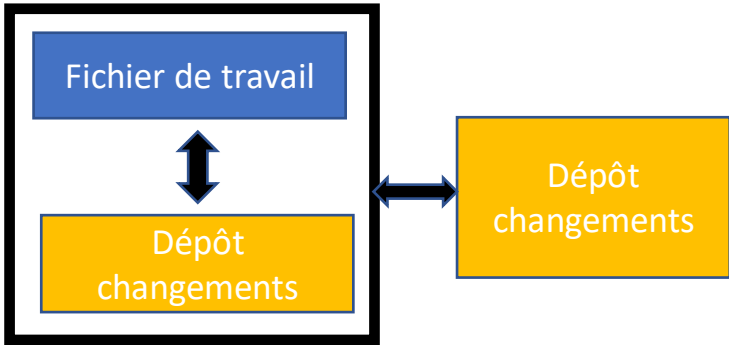
- La gestion de versions également appelé historique de versions ou contrôle de versions (en anglais *version control system* (VCS)) se réfère, dans le milieu numérique, au stockage de plusieurs versions de fichier(s) afin de pouvoir tracer l'évolution chronologique à travers les changements apportés d'une version à l'autre.
- Elle peut s'appliquer à un fichier individuel (exemple : un document texte) ou à plusieurs fichiers d'un projet; elle peut se faire au niveau individuel ou dans des groupes
- La gestion de versions se fait aussi à la base de plateformes collaboratives privées ou open source, comme par exemple [GitHub](#) ou [GitLab](#).
- On peut diviser les systèmes de gestion de versions en trois catégories :
  - Gestion de versions locale,
  - Gestion de versions centralisée,
  - Gestion de versions distribuée ou décentralisée.



Dans un système de gestion de versions **locale**, le *dépôt*(code ou code de source) avec les changements se trouve physiquement sur la même machine qui stocke les fichiers tracés



Dans la gestion de versions **centralisée**, le *dépôt* qui contient les informations sur les changements se trouve sur une autre machine par rapport aux fichiers de travail. Le cas de figure le plus commun consiste à garder le *dépôt* des changements sur un serveur centralisé, tandis que les différents ordinateurs individuels des personnes qui participent au projet ne gardent que la dernière version des fichiers de travail.



La gestion de versions **distribuée** combine la gestion de version locale et centralisée en créant **deux dépôts** des changements :

1. Le premier se trouve sur la même machine des fichiers de travail
2. Le deuxième se trouve dans une autre machine, souvent un serveur ou une plateforme *cloud* comme [GitHub](#) ou [GitLab](#), qui s'occupe de centraliser les changements

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Gestion de versions

### gestion de versions distribuées

- La gestion distribuée est notamment l'un des éléments principaux des projets open source, comme elle est représentée dans l'image ci-bas
- La gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. De ce fait, il existe plusieurs dépôts pour un même logiciel.
- Ce système est très utilisé par les logiciels libres.
- Par exemple, **GNU Arch** et **Git** sont des logiciels de gestion de versions décentralisée.



Exemple de gestion de versions distribuée source <https://edutechwiki.unige.ch/>

- Dans ce schéma qui utilise **Git** et **GitHub** comme exemple (le même principe peut-être appliqué à d'autres systèmes), le dépôt local se trouve dans un dossier **.git** sur l'ordinateur d'une personne. Ce dépôt local est partagé en open source à travers un dépôt central qui se trouve sur la plateforme *cloud* **GitHub**. Ce dépôt central peut être utilisé par plusieurs personnes pour :
  - Contribuer au même projet: envoyer des changements qui sont ensuite incorporés dans le dépôt central et peuvent donc être ensuite propagés à tout dépôt local connecté,
  - Utiliser ce dépôt central comme base pour un autre projet.

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Gestion de versions

### Exemples d'outils de gestion de versions

- **Wiki** : appelée dans ce contexte souvent historique des versions, est l'une des fonctionnalités principales caractérisant tout système qui s'inspire du principe wiki. Des systèmes de ce type peuvent se trouver à plusieurs endroits, avec des fonctionnalités légèrement différentes, comme par exemple dans :
  - **Moodle** : une plateforme pédagogique très utilisée dans la formation et l'enseignement,
  - **GitHub** : dans lequel le wiki est souvent utilisé comme documentation ou moyen d'organisation entre collaborateurs,
  - **Tous les sites qui sont basés sur un moteur wiki**, comme par exemple MediaWiki, le logiciel open-source à la base de Wikipédia et également de EduTech Wiki.
- **Logiciels de traitement de texte** : plusieurs logiciels de traitement de texte, surtout dans les versions en ligne et collaboratives, permettent de retracer l'historique des versions : OFFICE ,GOOGLE DOCS..
- **CVS (Concurrent Versioning System)** : fonctionne sur un principe centralisé, de même que son successeur SVN (Subversion).
- **Logiciels de SCM décentralisés** : sont apparus plus récemment : Mercurial et surtout Git, que nous utiliserons dans la suite de ce chapitre. Ce sont également des logiciels libres.
- **TFS (Team Foundation Server)** de Microsoft: TFS est une solution payante qui fonctionne de manière centralisée.

# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. Intérêt de la gestion de version et présentation des outils existants de gestion de versions
2. **Présentation de Git**
3. Présentation de Gitlab
4. Manipulation des dépôts avec Gitlab
5. Gestion des conflits de fusion avec Git/GitLab
6. Comparaison Git vs Gitlab





# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Présentation de git

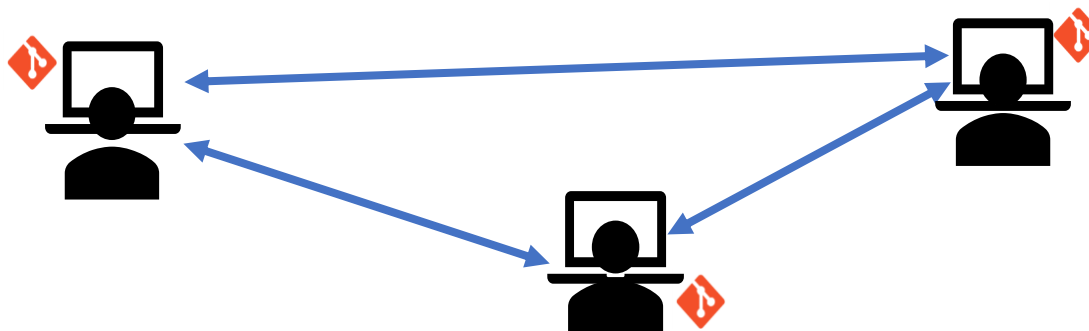
### C'est quoi git?

- **Git** est un logiciel libre de gestion de versions. C'est un outil qui permet d'archiver et de maintenir les différentes versions d'un ensemble de fichiers constituant souvent le code source d'un projet logiciel, il est multi-langages et multi-plateformes. **Git** est devenu à l'heure actuelle un quasi-standard .



logo de Git (Le nom "Git" se prononce comme dans guitare)

- **Git** rassemble dans un **dépôt (repository ou repo)** l'ensemble des données associées au projet. Il fonctionne de manière décentralisée: tout dépôt Git contient l'intégralité des données (code source, historique, versions, etc).
- Chaque participant au projet travaille à son rythme sur son dépôt local. Il existe donc autant de dépôts que de participants.
- **Git** offre des mécanismes permettant de synchroniser les modifications entre tous les dépôts.

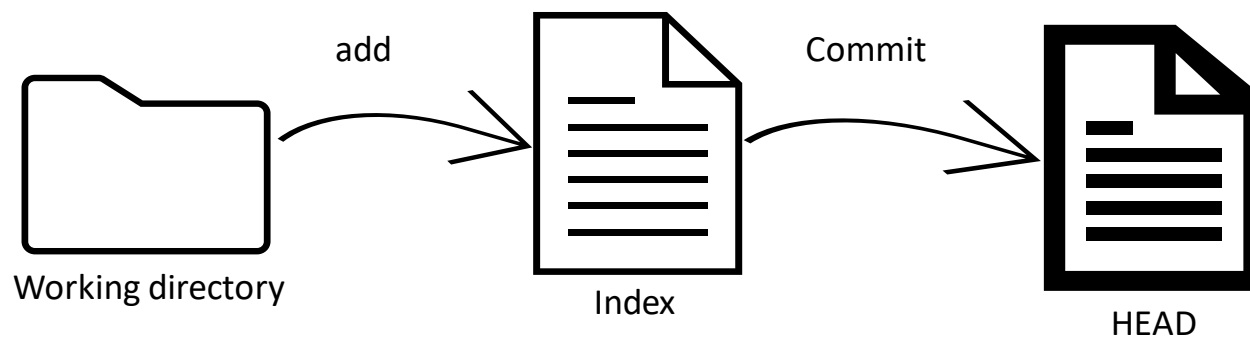


# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Présentation de git

### Les dépôts avec git:

- **Version** : contenu du projet à un moment de son cycle de vie.
- **Dépôt**(repository) : l'historique du projet, contenant toutes ses versions.
- **Branche** (branch) = variante d'un projet.
- Un dépôt Git correspond physiquement à un ensemble de fichiers rassemblés dans un répertoire **.git**. Sauf cas particulier, il n'est pas nécessaire d'intervenir manuellement dans ce répertoire.
- Lorsqu'on travaille avec Git, il est essentiel de faire la distinction entre trois zones :
  - Le **répertoire de travail** (*working directory*) :correspond aux fichiers actuellement sauvegardés localement.
  - L'**index** ou *staging : area* est un espace de transit.
  - **HEAD** : correspond aux derniers fichiers ajoutés au dépôt.



# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

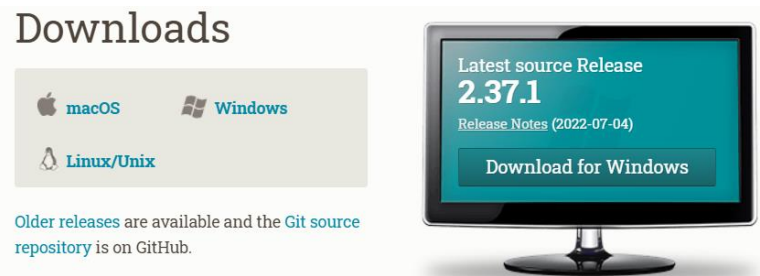
## Installation de git

### Installation git?

1.Vérifier dans le terminal que git est bien installé: **git version**.

```
C:\Users\<user>>git version
git version 2.37.1.windows.1
```

2.Si ce n'est pas le cas, installez-le en suivant les recommandations de votre système d'exploitation où en le téléchargeant depuis [git-scm.com/downloads](https://git-scm.com/downloads), puis redémarrez votre terminal.



### Git Bash

Lancer git après l'installation et tapez la commande **git** pour voir les différentes commandes git et leurs rôles

```
DELL@DESKTOP-01D0J3U MINGW64 ~/Desktop
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      [--super-prefix=<path>] [--config-env=<name>=<envvar>]
      <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index
```

# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Commandes git

### Principales commandes git

Les principales commandes git sont :

- **git init** : crée un nouveau dépôt vide à l'emplacement courant.
- **git status** : affiche les différences entre le répertoire de travail, l'index et HEAD.
- **git add** : ajoute des fichiers depuis le répertoire de travail vers l'index.
- **git commit** : ajoute des fichiers depuis l'index vers HEAD.
- **git log** : affiche l'historique des commits.
- **git clone** : clone un dépôt existant local ou distant.
- **git pull** : récupère des modifications depuis un dépôt distant vers HEAD.
- **git push** : publie des modifications depuis HEAD vers un dépôt distant.

## 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

### Commandes git

#### Exemple des principales étapes pour utiliser les commandes git localement

1. Ouvrez git bash ;
2. Déplacez vous vers un dossier de votre choix à l'aide de la commande **cd**: **ex. cd E:/**
3. Créez un nouveau dossier : **mkdir mon\_projet**
4. Placez vous dans ce dossier: **cd mon\_projet**
5. Initialisez le dossier avec : **git init**
6. Créer un fichier texte d'exemple: **echo « Bonjour tout le monde » > file.txt**
7. Vous pouvez proposer un changement (l'ajouter à l'**Index**) en exécutant les commandes **git add <fichier>** ou **git add \*** .
8. Pour valider ces changements, utilisez **git commit -m 'Message de validation'** .
9. Maintenant, vous pouvez envoyer vos changements vers le serveur distant : en utilisant les commandes 7 et 8 à chaque changement .
10. Pour afficher l'historique des commit, utilisez **git log** ou **git log --oneline**

# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Gestion des branches avec git

### Branches

Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits. La branche par défaut dans Git s'appelle master. Au fur et à mesure des validations, la branche master pointe vers le dernier des commits réalisés. À chaque validation, le pointeur de la branche master avance automatiquement.

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche *master* est la branche par défaut quand vous créez un dépôt. Utilisez les autres branches pour le développement et fusionnez ensuite à la branche principale quand vous avez fini.

# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

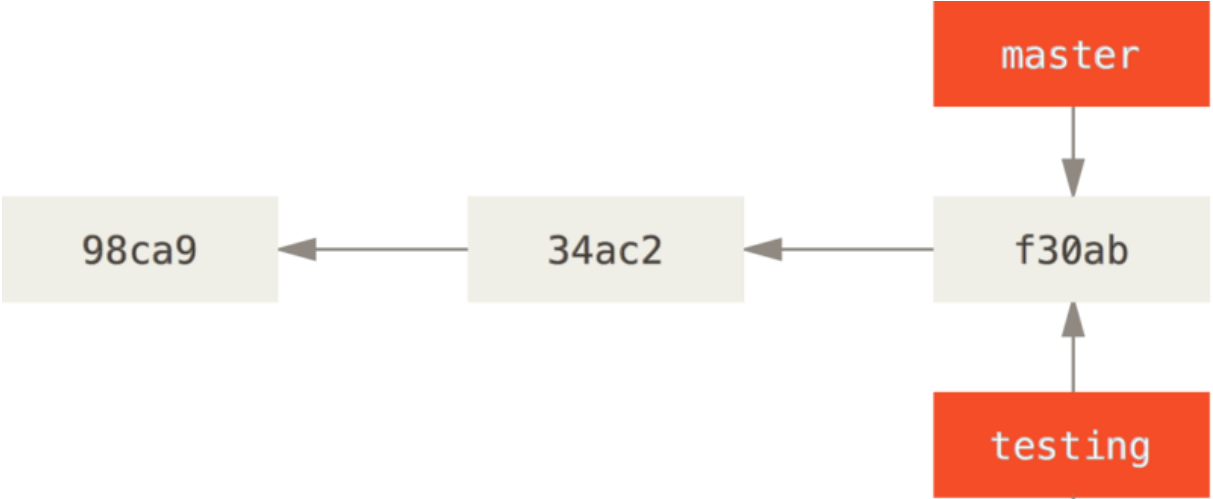
## Gestion des branches avec git

### Créer une nouvelle branche

Que se passe-t-il si vous créez une nouvelle branche ? Eh bien, cela crée un nouveau pointeur pour vous. Supposons que vous créez une nouvelle branche nommée **testing**. Vous utilisez pour cela la commande `git branch` :

```
$ git branch testing
```

Cela crée un nouveau pointeur vers le **commit** courant.



Deux branches pointant vers la même série de *commits*

# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

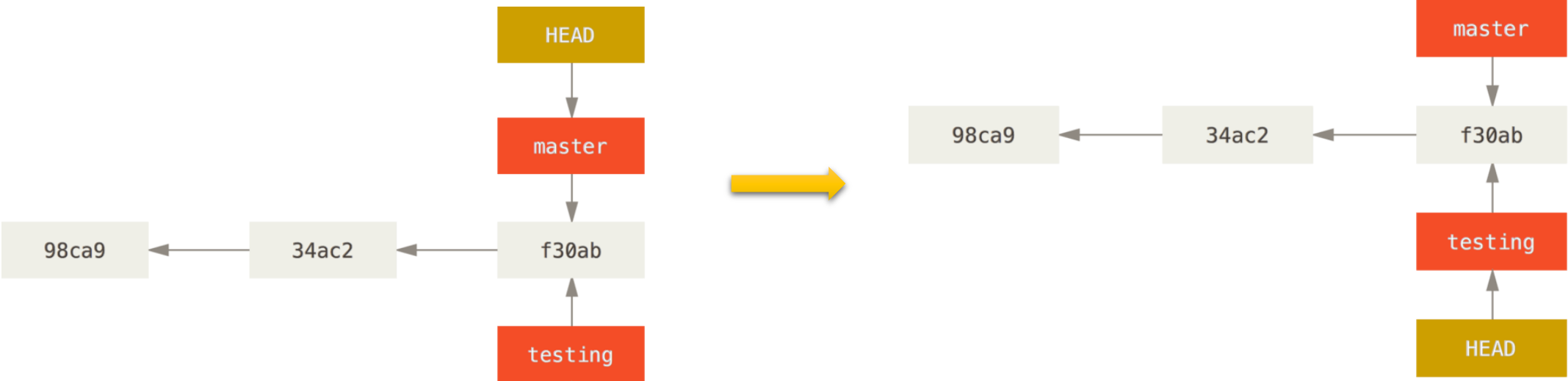
## Gestion des branches avec git

### Basculer entre les branches

- Comment Git connaît-il alors la branche sur laquelle vous vous trouvez ? Il conserve à cet effet un pointeur spécial appelé **HEAD**. Dans Git, il s'agit simplement d'un pointeur sur la branche locale où vous vous trouvez.
- Dans ce cas, vous vous trouvez toujours sur master. En effet, la commande *git branch* n'a fait que créer une nouvelle branche — elle n'a pas fait basculer la copie de travail vers cette branche.
- Pour basculer sur une branche existante, il suffit de lancer la commande *git checkout*.
- Basculons sur la nouvelle branche testing :

**\$ git checkout testing**      ou      **\$ git switch testing**

Cela déplace HEAD pour le faire pointer vers la branche testing.





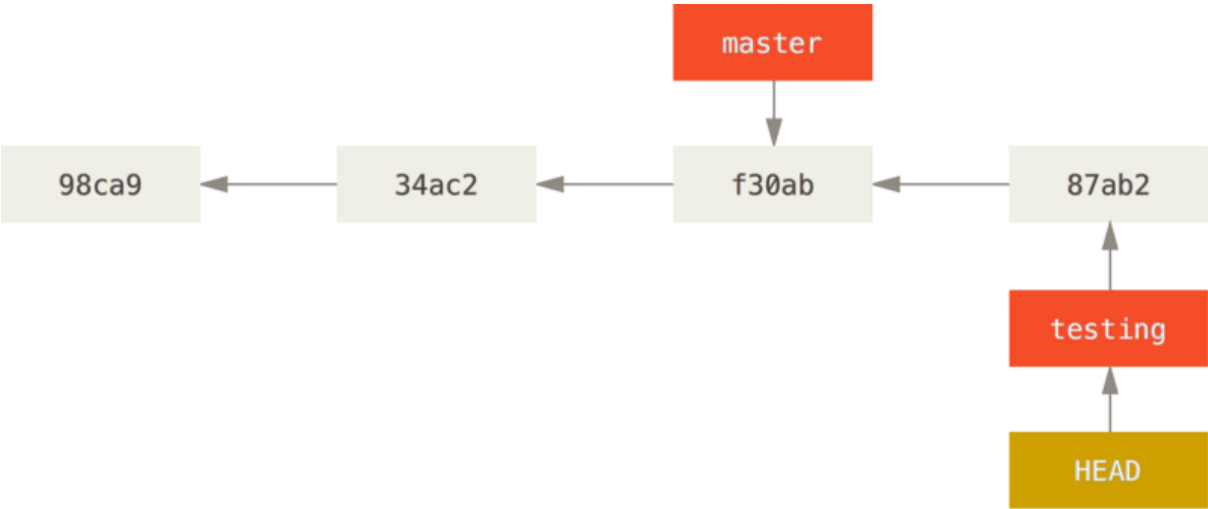
# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Gestion des branches avec git

### Basculer entre les branches

```
$ vi test.rb
$ git commit -a -m 'made a change'
```

Si on effectue un changement et on fait un commit, La branche `testing` avance tandis que la branche `master` pointe toujours sur le **commit** sur lequel vous étiez lorsque vous avez lancé la commande `git checkout` pour changer de branche.



# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

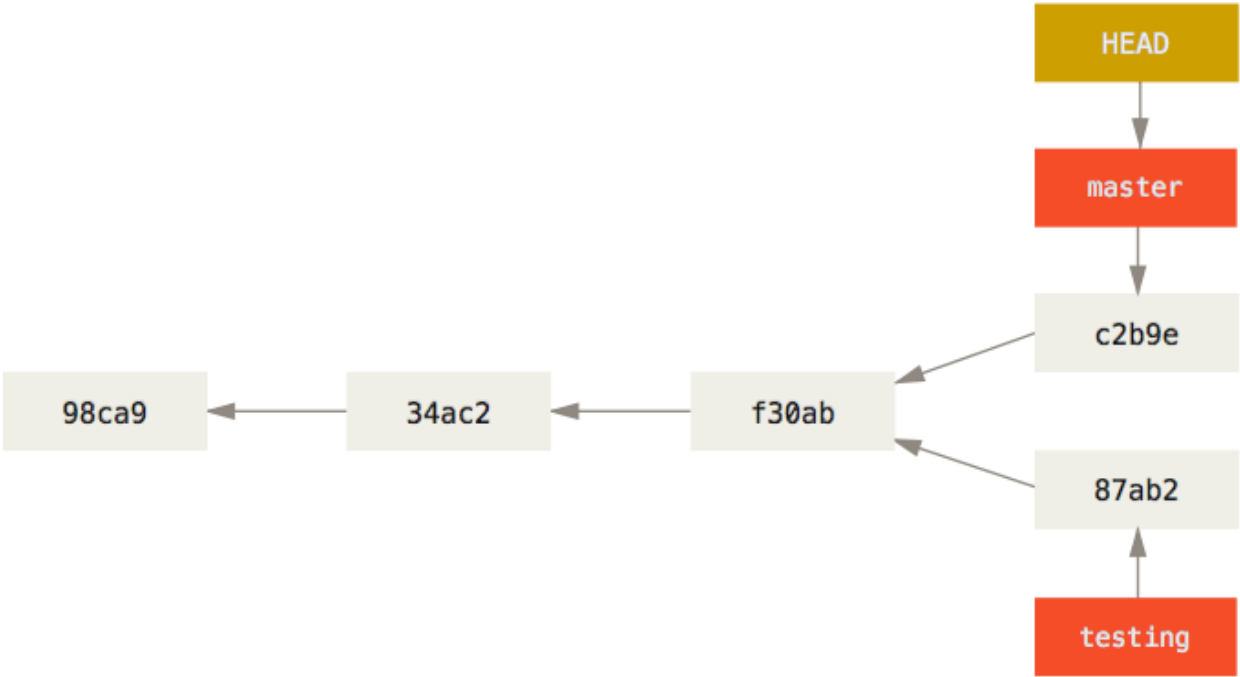
## Gestion des branches avec git

### Basculer entre les branches

Rebasculant vers la branche master et effectuant des changements:

```
$ git checkout master
$ vi test.rb
$ git commit -a -m 'made other changes'
```

Ces deux modifications sont isolées dans des branches séparées : vous pouvez basculer d’une branche à l’autre et les fusionner quand vous êtes prêt.



## 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

### Gestion des branches avec git

#### Créer une branche et basculer dessus

Il est habituel de créer une nouvelle branche et de vouloir basculer sur cette nouvelle branche en même temps — ça peut être réalisé en une seule opération avec :

```
git checkout -b <nouvelle-branche>.
```

Depuis Git version 2.23, on peut utiliser **git switch** au lieu de **git checkout** pour :

- Basculer sur une branche existante : **git switch testing-branch**,
- Créer une branche et basculer dessus : **git switch -c nouvelle-branche**;  
*le drapeau -c signifie créer, vous pouvez aussi utiliser le drapeau complet --create,*
- Revenir sur votre branche précédemment extraite : **git switch**

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

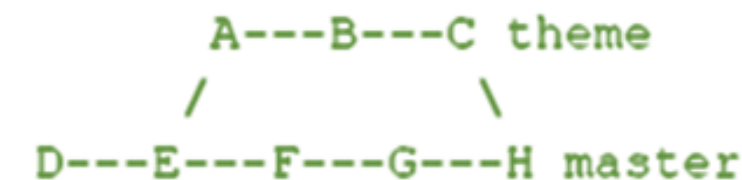
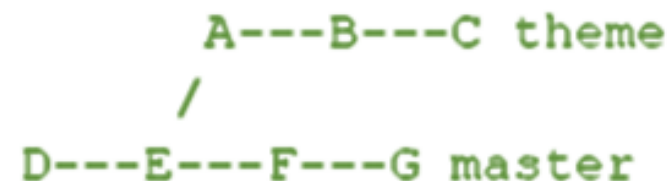
## Fusionner les branches

### Fusionner les branches

**git-merge** - Fusionne deux ou plusieurs historiques de développement ensemble

- Intègre les modifications des commits nommés (depuis le moment où leur historique a divergé de la branche actuelle) dans la branche actuelle.

Supposons que l'historique suivant existe et que la branche actuelle est master :



- La branche "**theme**" s'est écartée de **master** (c'est-à-dire E) jusqu'à son commit actuel (**C**);
- "**git merge theme**" jouera les modifications apportées à la branche "**theme**" par dessus **master**;
- Elle enregistrera le résultat dans un nouveau commit comprenant les noms des deux parents et un message de validation de l'utilisateur décrivant les modifications.

**git merge -abort:** ne peut être exécutée qu'après que la fusion ait entraîné des conflits. Elle annulera le processus de fusion et tentera de reconstruire l'état antérieur à la fusion;

## 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

### Fusionner les branches

#### Conflits lors de fusion :

Un conflit de fusion intervient lorsque l'on tente de fusionner deux branches qui modifient la même partie d'un même fichier. Dans ce cas, git va intégrer les deux versions dans le même fichier puis laisser le développeur décider du contenu final de cette partie.

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Fusionner les branches

### Exemple de conflit et sa résolution :

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git
$ git init
Initialized empty Git repository in E:/Projects/projet_git/.git/

asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ echo "ligne A" > file.txt

asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ git commit -m 'premier ajout du file.txt'
[master (root-commit) e9af967] premier ajout du file.txt
1 file changed, 1 insertion(+)
create mode 100644 file.txt

asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ git checkout -b fixbug
Switched to a new branch 'fixbug'
```

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Fusionner les branches

### Exemple de conflit et sa résolution :

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (fixbug)
$ echo "ligne B" > file.txt
```

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (fixbug)
$ git commit -a -m 'Modification du file.txt'
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[fixbug 718cf8a] Modification du file.txt
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ git branch
master
* fixbug
```

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (fixbug)
$ git checkout master
Switched to branch 'master'
```

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/projects/projet_git (master)
$ echo "ligne C" > file.txt
```

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

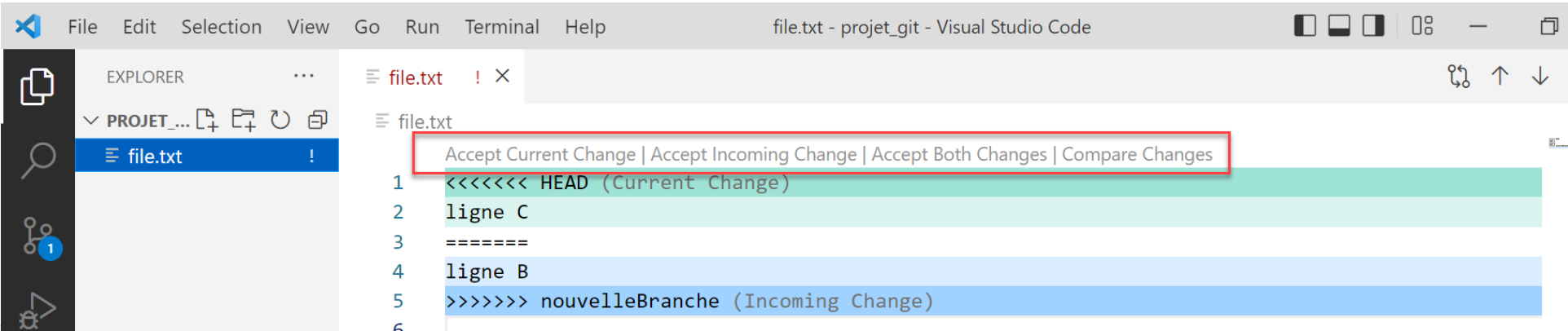
## Fusionner les branches

### Exemple de conflit et sa résolution :

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/projects/projet_git (master)
$ git commit -a -m 'Modification du file.txt à partir du master'
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[master c02d770] Modification du file.txt à partir du master
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/projects/projet_git (master)
$ git merge fixbug
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/projects/projet_git (master|MERGING)
$ code .
```





# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

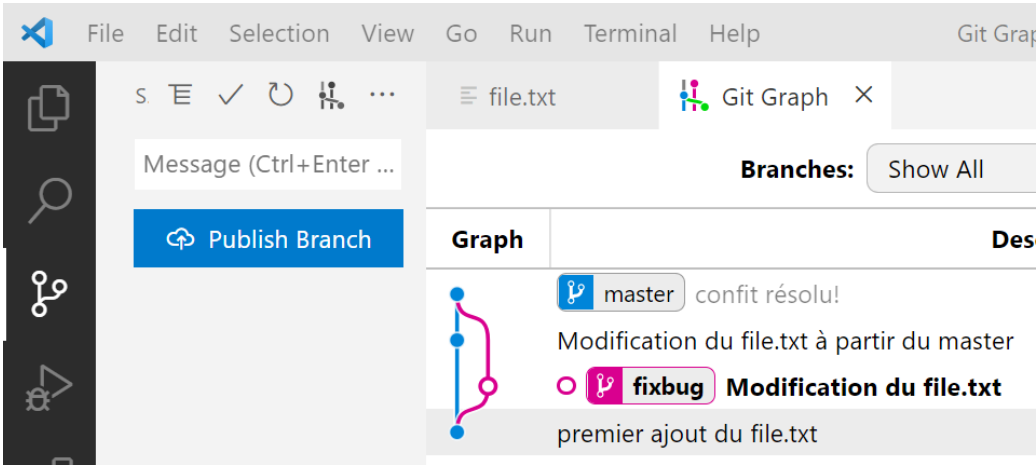
## Fusionner les branches

### Exemple de conflit et sa résolution :

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/projects/projet_git (master|MERGING)
$ git commit -a -m 'confit résolu!'
[master 77f3c56] confit résolu!
```

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/projects/projet_git (master)
$ git log --oneline
77f3c56 (HEAD -> master) confit résolu!
c02d770 Modification du file.txt à partir du master
718cf8a (fixbug) Modification du file.txt
e9af967 premier ajout du file.txt
```

A l'aide de l'extension **git graph** (sous **visual studio code**), on peut afficher graphiquement l'historique des commits et des branches



# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. Intérêt de la gestion de version et présentation des outils existants de gestion de versions
2. Présentation de Git
- 3. Présentation de Gitlab**
4. Manipulation des dépôts avec GitlabGestion des conflits de fusion avec Git/GitLab
5. Comparaison Git vs Gitlab



# 01-Manipuler les outils de gestion de versions

(Git/Gitlab) :

## Présentation de Gitlab

### C'est quoi Gitlab?

**Gitlab** est une plateforme open source et collaborative de développement basé sur **Git**. **Gitlab** permet d'héberger des projets web, du code, et de la documentation.

### Fonctionnalités de Gitlab :

- L'interface de **GitLab** reste très similaire à celle de **GitHub**. Toutefois, **GitLab** propose des options pour le moins pratiques :
  - Gestion de projet
  - Planification / priorisation
  - Build
  - Test logiciel
  - Sécurité applicative
  - Gestion des configurations
  - Monitoring
  - Intégration et déploiement continu, etc.
- Pour un emploi ergonomique, **GitLab** se situe sur une machine virtuelle, elle-même hébergée sur un serveur web. Cet outil de plateforme collaborative s'appuie sur une base de données.
- L'interface d'administration, notamment pour la création de comptes utilisateurs, passe par une configuration en ligne :
  - Création / suppression de dépôts.
  - Enregistrement et gestion des droits d'accès aux dépôts .
  - Outils graphiques pour visualiser et éditer : graphe des commits, branches, tags, fichiers, etc.
  - Documentation des projets (fichiers README..)
  - Outils de communication de projets (issues)
  - Mécanisme de fork et merge requests (comme sur github).

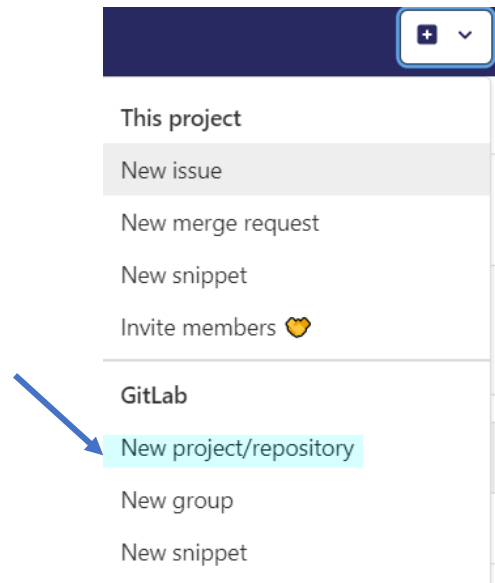
# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Création de compte



### Etapas pour créer un compte sur Gitlab

- Ouvrir l'interface web **GitLab** sur : <https://gitlab.com>
- Si vous ne l'avez pas encore fait, ajoutez un mot de passe à votre compte **GitLab**, depuis la page de paramètres de votre compte **GitLab**
- Après la connexion avec le compte **GitLab** vous aurez l'interface suivante :
  - Pour créer un projet cliquer sur **create blank project** ou **newProject/repository**



#### Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

#### Create new project

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Création de compte



### Etapas pour créer compte sur Gitlab

- Donner un nom à votre groupe et projet :

#### Create or import your first project

Projects help you organize your work. They contain your file repository, issues, merge requests, and so much more.

Create

Import

Group name

fullstack

Project name

projet1

Your project will be created at:

https://gitlab.com/fullstack38/projet1

You can always change your URL later

☒ Include a Getting Started README

Recommended if you're new to GitLab

Create project

- Le nom du projet sera aussi le nom du dépôt correspondant => Choisissez “public”, afin que le dépôt soit accessible en lecture aux autres stagiaires, puis cliquer sur **create project**

Project name

projet1

Project URL

https://gitlab.com/ fullstack38

Project slug

projet1

Want to organize several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Description format

Project deployment target (optional)

Select the deployment target

Visibility Level ?

☐ Private  
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☒ Public  
The project can be accessed without any authentication.

Project Configuration

☒ Initialize repository with a README  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ Enable Static Application Security Testing (SAST)  
Analyze your source code for known security vulnerabilities. [Learn more.](#)

Create project

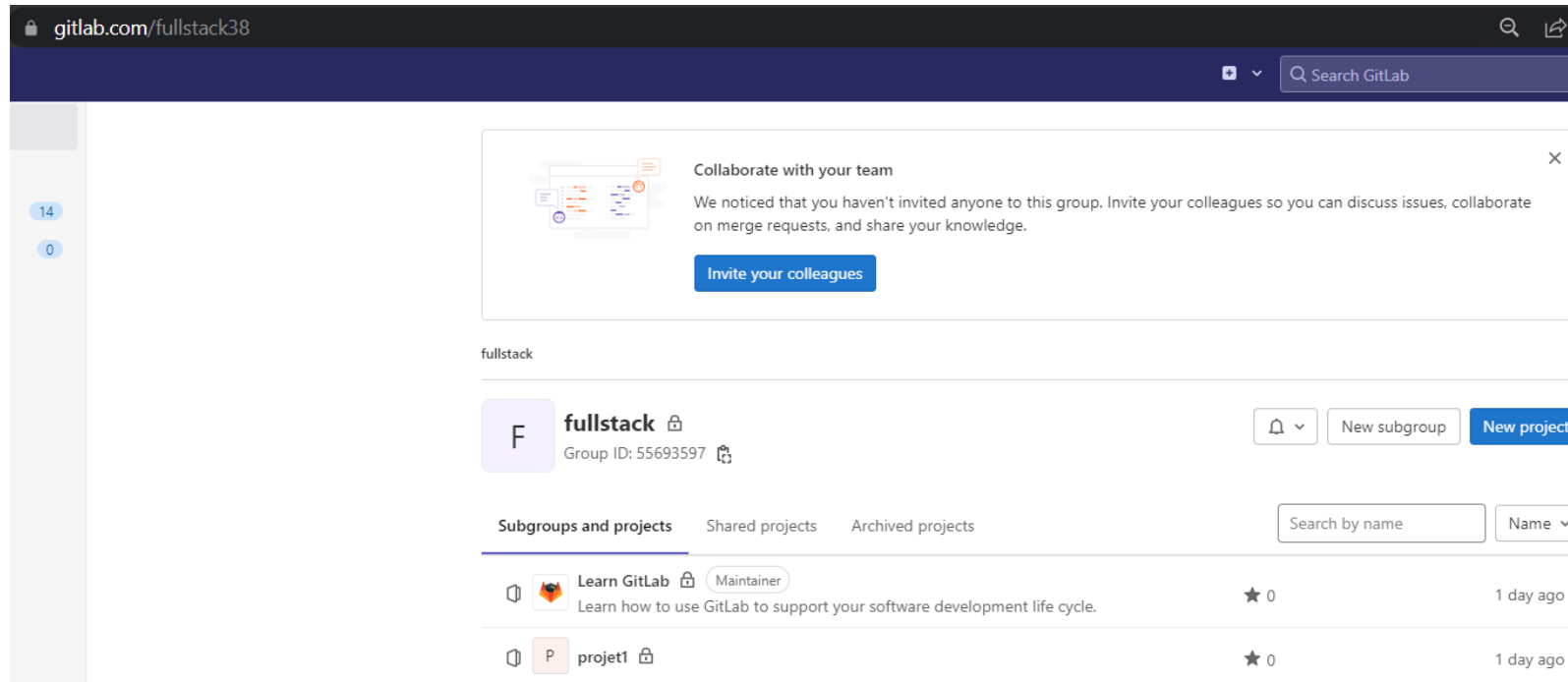
Cancel

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) : Création de compte



## Etapas pour créer compte sur Gitlab

- Vous accéder à votre projet via l'url : **gitlab.com/groupeuprojet** :



# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. Intérêt de la gestion de version et présentation des outils existants de gestion de versions
2. Présentation de Git
3. Présentation de Gitlab
- 4. Manipulation des dépôts avec Gitlab**
5. Gestion des conflits de fusion avec Git/GitLab
6. Comparaison Github vs Gitlab



# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Création dépôts sur Gitlab

1. Dans le terminal:

- Téléchargez le dépôt sur votre disque-dur: `$ git clone <url>` (remplacer <url> par l'adresse HTTPS de votre dépôt, celle qui finit par .git)
- puis entrez dans le dossier de ce dépôt: `$ cd nom_du_dépôt`, exemple:

```
git clone https://gitlab.com/fullstack38/fsdev.git
```

```
C:\Users\DELL>git clone https://gitlab.com/fullstack38/fsdev.git
Cloning into 'fsdev'...
```

```
C:\Users\DELL>cd fsdev
```



## 33

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Création dépôts sur Gitlab

4. Le fichier **README.md** devrait maintenant être visible depuis la page web du dépôt, sur **GitLab** :

main

fsdev /

+


Find file

Web IDE


↓


↓


Clone


 ajout du fichier README.md  
mohamed goumih authored 1 minute ago


5bcf9fa4





 README


 Add LICENSE


 Add CHANGELOG


 Add CONTRIBUTING

 Add Kubernetes cluster

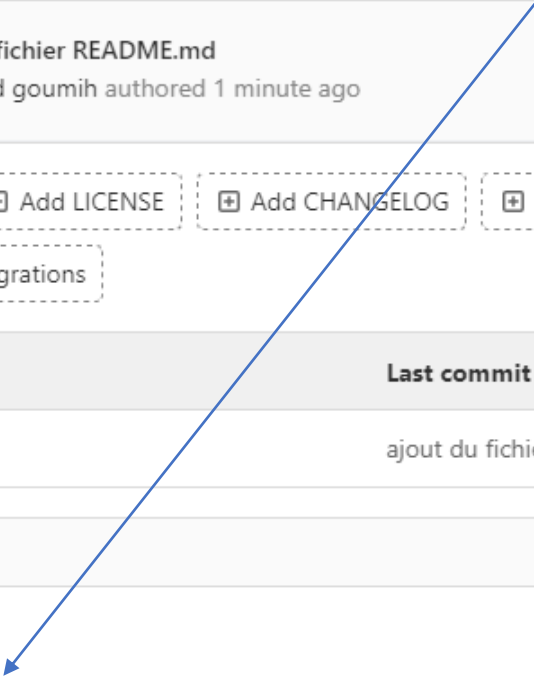
 Set up CI/CD

 Configure Integrations

Name	Last commit	Last update
 README.md	ajout du fichier README.md	1 minute ago

 README.md

"Bonjour"



# 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

## Gestion branches avec git

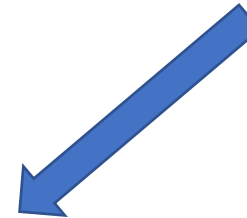


### Gestion branches avec commandes git

créer une nouvelle branche nommée B1 et passer dessus pour l'utiliser  
**git checkout -b B1**  
retourner sur la branche principale  
**git checkout master**  
et supprimer la branche  
**git branch -d B1**  
une branche n'est *pas disponible pour les autres* tant que vous ne l'aurez pas  
envoyée vers votre dépôt distant  
**git push origin <branch>**



pour mettre à jour votre dépôt local depuis les dernières validations,  
exécutez la commande  
**git pull**  
dans votre espace de travail pour *recupérer* et *fusionner* les changements  
distants.  
pour fusionner une autre branche avec la branche active (par exemple  
master), utilisez  
**git merge <branch>**  
dans les deux cas, git tente d'auto-fusionner les changements.  
Malheureusement, ça n'est pas toujours possible et résulte par  
des *conflits*. Vous devez alors régler ces *conflits* manuellement en éditant  
les fichiers indiqués par git. Après l'avoir fait, vous devez les marquer  
comme fusionnés avec  
**git add <nomdufichier>**  
après avoir fusionné les changements, vous pouvez en avoir un aperçu  
en utilisant  
**git diff <source\_branch> <target\_branch>**



vous pouvez annuler les changements locaux en utilisant cette commande  
**git checkout -- <nomdufichier>**  
cela remplacera les changements dans votre arbre de travail avec le dernier contenu du HEAD. Les changements  
déjà ajoutés à l'index, aussi bien les nouveaux fichiers, seront gardés.  
Si à la place vous voulez supprimer tous les changements et validations locaux, récupérez le dernier historique  
depuis le serveur et pointez la branche principale locale dessus comme ceci  
**git fetch origin**  
**git reset --hard origin/master**

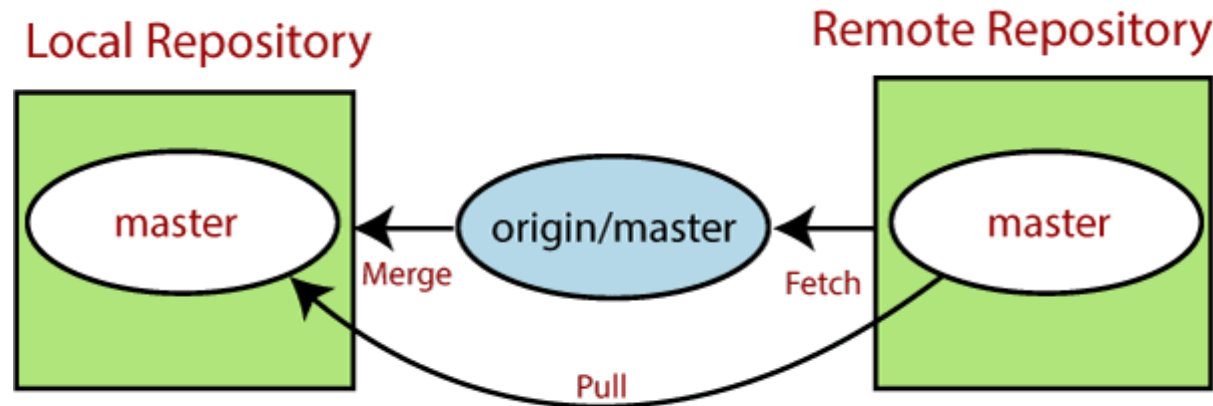
## 01-Manipuler les outils de gestion de versions(Git/Gitlab) :

### Gestion branches avec git



#### La commande git fetch:

La commande `git fetch` permet de récupérer les modifications présentes sur le serveur distant que vous n'avez pas encore sur votre copie en local. Cette commande va alors mettre à jour dans votre dépôt local, l'ensemble des références distantes (branches, tags, ...) et récupérer les commits associés.

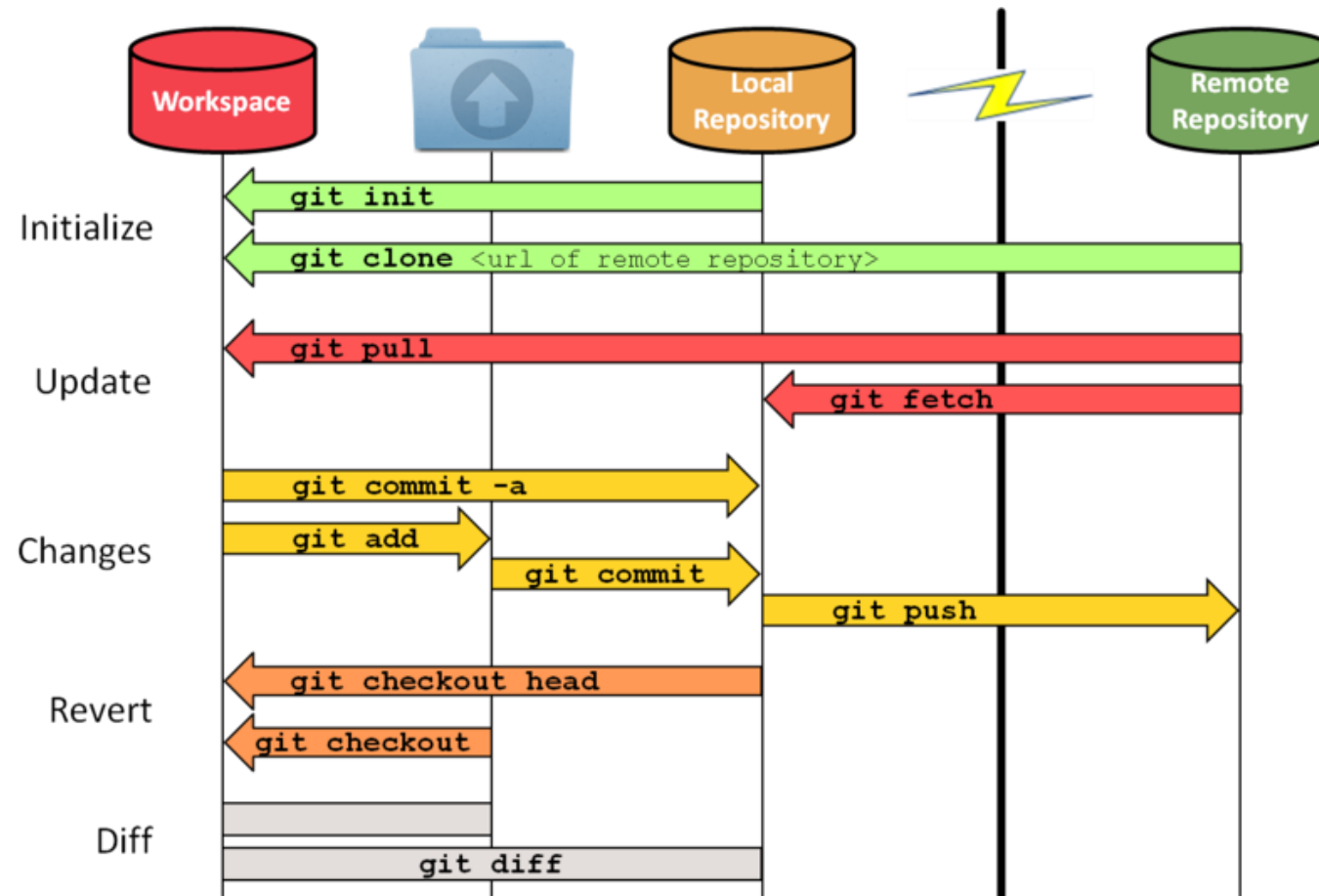


# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Récapitulatif des commandes git



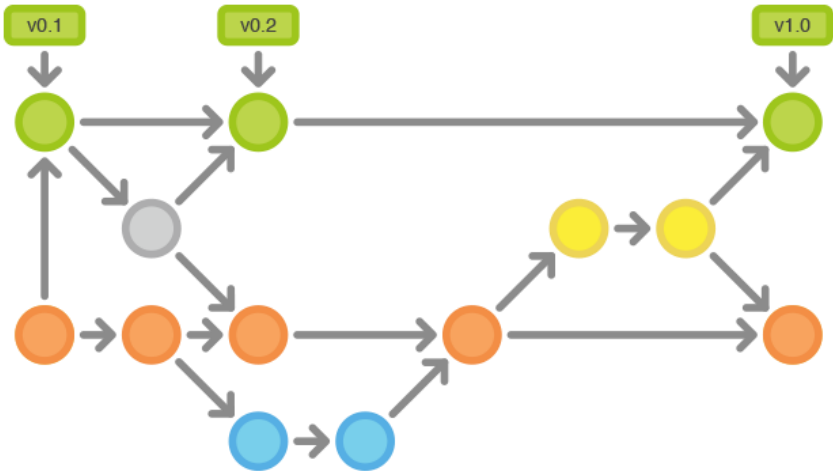
# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Git tag

- Les **tag** sont des références qui pointent vers des points spécifiques de l'historique de Git.
- Le **tag** est généralement utilisé pour capturer un point dans l'historique qui est utilisé pour une version release (exp. : v1.0.1).
- Un tag est une branche qui ne change pas.
- Contrairement aux branches, les tag ne gardent plus d'historique des prochains commits.



Créer un tag

**git tag** <tagname>

git tag v1.0

Créer un tag annoté : stocke des informations supplémentaires concernant la personne qui a crée le tag, la date ...

**git tag -a** <tagname> **-m** <message>

git tag -a v1.4 -m "my version 1.4"

Lister les tags

**git tag**

Partager un tag avec le repertoire distant

**git push** <remote> <tagname>

git push origin v1.4

Supprimer un tag

**git tag -d** <tagname>

git tag -d v1.4

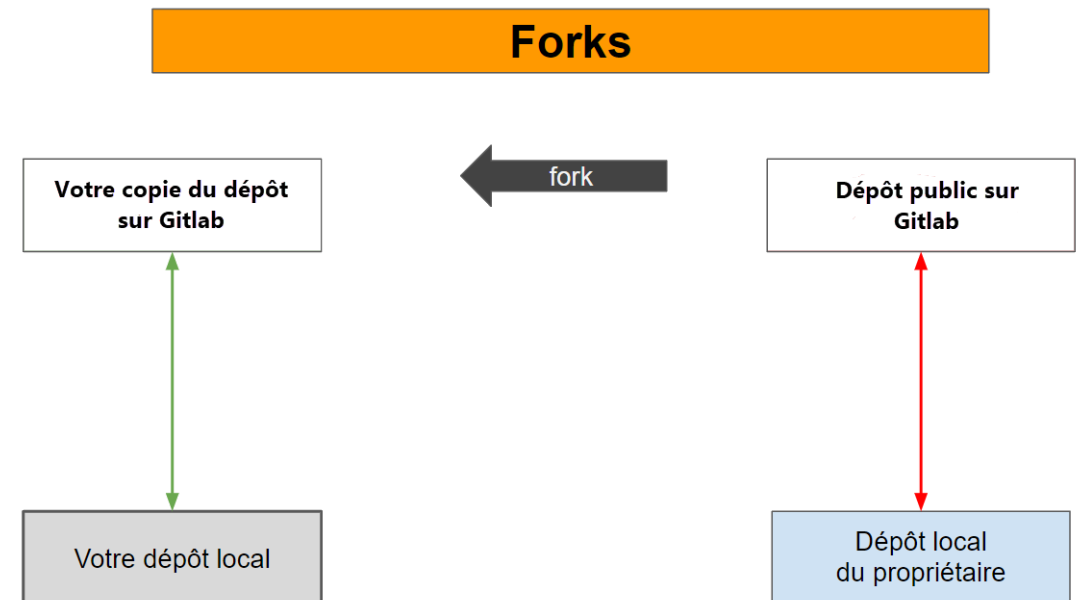
# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Le fork avec Gitlab : Contribution à un projet

- Si vous souhaitez contribuer à un projet existant sur lequel vous n'avez pas le droit de pousser, vous pouvez dupliquer (fork) ce projet. Cela signifie que Gitlab va faire pour vous une copie personnelle du projet. Elle se situe dans votre espace de nom et vous pouvez pousser dessus.
- **Un fork** : est une copie d'un dépôt. Ceci est utile lorsque vous souhaitez contribuer au projet de quelqu'un d'autre ou démarrer votre propre projet basé sur le sien.
- **fork** n'est pas une commande dans Git, mais quelque chose d'offert dans GitLab et d'autres référence de versions.



# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Le fork avec Gitlab : Contribution à un projet

- Créons un nouveau projet public ;
  - Le projet sera accessible par tout le monde et sans aucune authentification;
  - N'importe qui peut cloner le projet pour en créer une copie local ;
- 
- **Pourtant;** l'erreur suivante sera affichée si on essaie de faire un push sur ce projet public !  
(Après l'avoir cloner)

```
asmae@DESKTOP-PGQ50JJ MINGW64 /e/js2 (master)
$ git push
remote: Permission to [redacted] denied to [redacted]
la.
fatal: unable to access 'https://git[redacted].g
it/': The requested URL returned error: 403
```

New project > Create blank project

Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Project name

demo\_dev

Project URL

https://gitlab.com/ devowfs\_adarissa /

Project slug

demo\_dev

Want to organize several dependent projects under the same namespace? [Create a group.](#)

Project deployment target (optional)

Select the deployment target

Visibility Level ⓘ

☐ Private

☒ Public

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

The project can be accessed without any authentication.

Project Configuration

☒ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ Enable Static Application Security Testing (SAST)

Analyze your source code for known security vulnerabilities. [Learn more.](#)

Create project Cancel



# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

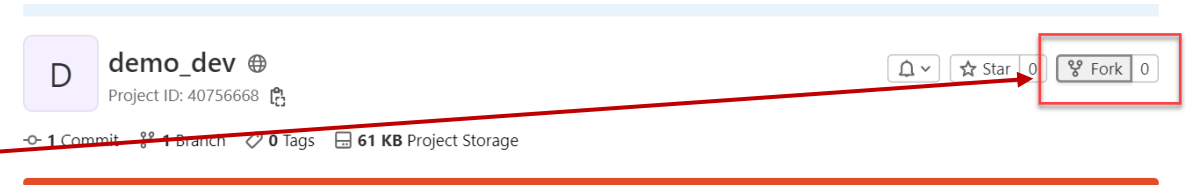
## Manipulation des dépôts avec Gitlab



### Le fork avec Gitlab : Contribution à un projet

Afin de pouvoir contribuer à un projet distant, on peut utiliser l'option fork de gitlab:

- En cliquant sur fork, la nouvelle fenêtre « fork project » apparaît;
- On peut indiquer, sur cette page, le nom de notre fork, son namespace, sa visibilité ...



#### Fork project

A fork is a copy of a project. Forking a repository allows you to make changes without affecting the original project.

Project name  
demo\_dev

Project URL  
https://gitlab.com/asmae.youala

Project slug  
demo\_dev

Want to organize several namespaces? [Create a group](#)

Project description (optional)

Namespaces  
devowfs\_adarissa  
asmae.youala

Visibility level [?](#)

☐ Private  
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

☐ Internal  
The project can be accessed by any logged in user.

☐ Public  
The project can be accessed without any authentication.

Fork project

Cancel

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Le fork avec Gitlab : Contribution à un projet



Sur le répertoire de travail, créons un accès à notre fork en lançant la commande suivante:

```
$ git remote add ma_copie https://gitlab.com/asmae.youala/demo_dev.git
```

*Un nom de mon choix*

*L'adresse du projet copié (forked project)*

The screenshot shows the GitLab interface for a project named 'demo\_dev'. The 'Forked from' link is highlighted with a red box. The 'Clone with HTTPS' option is also highlighted with a red box, and a red arrow points to the 'Copy URL' button. The 'Clone with SSH' option is also visible.

demo\_dev   
Project ID: 40757405 


1 Commit 1 Branch 0 Tags 113 KB Project Storage



Forked from [devowfs\\_adarissa / demo\\_dev](#)


main demo\_dev / +


Upload New File  
Asmae Youala authored 8 minutes ago


+ Add README + Add LICENSE + Add CHANGELOG + Add CONTRIBUTING + Set up CI/CD + Configure Integrations

Name	Last commit
 index.html	Upload New File

Find file Web IDE  Clone 

Clone with SSH  
git@gitlab.com:asmae.youala/demo\_d 

Clone with HTTPS  
https://gitlab.com/asmae.youala/de 

Open in your IDE  Copy URL

Visual Studio Code (SSH)  
Visual Studio Code (HTTPS)  
IntelliJ IDEA (SSH)  
IntelliJ IDEA (HTTPS)

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Le fork avec Gitlab : Contribution à un projet

Afin d'afficher les répertoires à distance du projet (cloned project + forked project), utilisez la commande suivante :

```
$ git remote -v
```

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/demo_dev (main)
$ git remote -v
ma_copie      https://gitlab.com/asmae.youala/demo_dev.git (fetch)
ma_copie      https://gitlab.com/asmae.youala/demo_dev.git (push)
origin https://gitlab.com/devowfs_adarissa/demo_dev (fetch)
origin https://gitlab.com/devowfs_adarissa/demo_dev (push)
```

*origin: c'est le projet original en lecture seule*

*ma\_copie: c'est mon fork avec un accès en lecture et en écriture*

Pour pousser les modifications vers le projet fork, il suffit de lancer la commande :

```
$ git push ma_copie
```

Nous pouvons renommer l'un des répertoires en utilisant la commande :

```
$ git remote rename ancien_nom nouveau_nom
```

```
asmae@DESKTOP-PGQ5OJJ MINGW64 /e/demo_dev (main)
$ git remote rename ma_copie fullstack

asmae@DESKTOP-PGQ5OJJ MINGW64 /e/demo_dev (main)
$ git remote -v
fullstack      https://gitlab.com/asmae.youala/demo_dev.git (fetch)
fullstack      https://gitlab.com/asmae.youala/demo_dev.git (push)
origin https://gitlab.com/devowfs_adarissa/demo_dev (fetch)
origin https://gitlab.com/devowfs_adarissa/demo_dev (push)
```

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Merge Request avec Gitlab

- **Merge Request** (Les demandes de fusion) : (**Pull Request** en github): sont la façon dont vous vérifiez les modifications du code source dans une branche. Lorsque vous ouvrez une demande de fusion, vous pouvez visualiser et collaborer sur les changements de code avant la fusion.
- Vous pouvez créer une Merge Request à partir de la liste des demandes de fusion.
- Sur le menu gauche
  1. Dans le menu de gauche de votre projet (forked), sélectionnez **Merge Requests**.
  2. puis, sélectionnez **new merge request**.
  3. Sélectionnez une branche source et cible, puis Comparez les branches et continuez.
  4. Remplissez les champs et sélectionnez **create merge request**.

Asmae Youala > demo\_dev > Merge requests > New

### New merge request

**Source branch**  
asmae.youala/demo\_dev main

**Target branch**  
devowfs\_adarissa/demo\_dev main

Upload New File  
Asmae Youala authored 13 hours ago 38bb8d9d

Compare branches and continue

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Merge Request avec Git/gitlab

- Vous pouvez créer une Merge Request en exécutant des commandes Git sur votre ordinateur local.

1. Créer une branche:

`git checkout -b my-new-branch`

2. Créer ou modifier vos fichiers, puis les mettre dans le stage et les valider :

`git add .`

`git commit -m "message de validation"`

3. Poussez votre branche vers GitLab :

`git push -u origin my-new-branch`

4. GitLab vous invite avec un lien direct pour créer une demande de fusion :

[Copiez le lien et collez-le dans votre navigateur.](#)

# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Manipulation des dépôts avec Gitlab



### Collaborer sur un dépôt Gitlab

- Par défaut seul le créateur d'un dépôt peut y ajouter des **commits**.
- Si un autre développeur souhaite contribuer au dépôt, il existe deux manières de procéder:
  - **forker** le dépôt puis proposer une contribution à l'aide d'un **pull request(merge request)**;
  - ou obtenir la permission d'ajouter des **commits** directement dans le dépôt.

### Etapes pour ajouter un développeur dans une équipe Gitlab:

1. Depuis la barre latérale de la page du dépôt, cliquer sur "**Paramètres**",
2. Cliquer sur "Membres",
3. Taper le ou les noms d'utilisateurs (ou adresse email) du/des développeurs à ajouter,
4. Sélectionner le rôle (ou niveau de permissions) à donner à ce(s) développeur(s),
5. Vérifier que le(s) développeur(s) est/sont bien capables d'ajouter et pusher un commit dans la branche de travail du dépôt (ex: master).

**Note:** pour qu'un développeur aie le droit de pusher des **commits** dans votre dépôt, il faut lui donner le rôle de "**Maintainer**". Le rôle "**Developer**" ne suffit pas.

### Etapes pour ajouter un commit dans le dépôt d'un autre développeur

1. Utiliser git clone pour importer le dépôt de l'autre stagiaire
2. S'assurer qu'on a bien les dernières mises à jour (git pull)
3. Créer un nouveau fichier dans le dépôt local, puis l'ajouter à l'index (git add)
4. Créer un commit contenant ce fichier (git commit) puis l'envoyer sur le dépôt de l'autre stagiaire (git push)
5. Dans l'interface web de GitLab, aller sur le projet de votre camarade, puis cliquez sur "commits" pour vérifier que votre commit apparaît bien dans la liste.

# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. Intérêt de la gestion de version et présentation des outils existants de gestion de versions
2. Présentation de Git
3. Présentation de Gitlab
4. Manipulation des dépôts avec Gitlab
- 5. Gestion des conflits de fusion avec Git/GitLab**
6. Comparaison Github vs Gitlab



# 01-Manipuler les outils de gestion de versions (Git/Gitlab) :

## Gestion conflits avec Gitlab

### Définition d'un conflit de fusion

Un conflit de fusion intervient lorsque l'on tente de fusionner deux branches qui modifient la même partie d'un même fichier. Dans ce cas, git va intégrer les deux versions dans le même fichier puis laisser le développeur décider du contenu final de cette partie.

### Etales à suivre pour causer un conflit de fusion :

1. Créer un nouveau dépôt sur **GitLab**
2. Cloner ce dépôt localement avec `git clone`
3. Dans le répertoire du dépôt local, créer un fichier **README.md** contenant un texte
4. Créer un commit initial sur la branche master et l'envoyer sur **GitLab** avec `git add`, `git commit` puis `git push`
5. Créer une branche **branche1** à partir de master, avec `git checkout -b`
6. Dans le **README.md** de cette branche, modifier le premier texte, créer un commit, puis envoyer les modifications de cette branche sur **GitLab**
7. Revenir à la branche master avec `git checkout`
8. Créer une branche **branche2** à partir de master (comme dans l'étape 5)
9. Dans le **README.md** de cette branche, modifier le texte différemment de celui saisi à l'étape 6, créer un commit, puis envoyer les modifications de cette branche sur **GitLab**
10. Revenir à la **branche master**
11. Fusionner **branche1** dans master, avec `git merge`
12. Fusionner **branche2** dans master
13. Taper `git status` pour voir le message du conflit

### Etales à suivre pour résoudre un conflit

Pour résoudre ce conflit, il va falloir:

1. ouvrir le fichier **README.md** dans son éditeur de code,
2. constater comment git représente le conflit, et la source de chaque version,
3. éditer le fichier pour ne conserver que la version finale souhaitée,
4. puis créer un commit.
5. Après la résolution du conflit de fusion, taper `git status` pour voir le résultat
6. supprimer les branches **branche1** et **branche2**, dans votre dépôt local, et dans le dépôt distant associé (sur **GitLab**).



# CHAPITRE 1

## Manipuler les outils de gestion de versions (Git/Gitlab)

1. Intérêt de la gestion de version et présentation des outils existants de gestion de versions
2. Présentation de Git
3. Présentation de Gitlab
4. Manipulation des dépôts avec Gitlab
5. Gestion des conflits de fusion avec Git/GitLab
6. **Comparaison Github vs Gitlab**



# 01-Manipuler les outils de gestion de versions

## Git/Gitlab :

### Comparaison Github et Gitlab

#### Quelques Fonctionnalités semblables entre gitlab et github

**GitHub** et **GitLab** reposent tous les deux sur **Git** et ses commandes, la **migration** d'une plateforme à l'autre est possible sans grandes difficultés. L'importation des repositories, **wikis**, **pull requests** et **issues** est facile en règle générale. En revanche, il existe quelques différences entre **GitHub** et **GitLab**, comme l'illustre le tableau ci-dessous :

Fonctionnalité	GitLab	GitHub
Git	✓	✓
Version auto-hébergée	✓	✓ (avec plan d'entreprise)
Intégration et livraison continues	✓	✓ (avec une application tierce)
Documentation basée sur le Wiki	✓	✓
Aperçu des modifications du code	✓	✓
Suivi des problèmes	✓	✓
Examen du code	✓	✓
Cessionnaires d'émissions multiples	✓ (Plan payant)	✓ (uniquement le dépôt public sur le plan gratuit)
Conseils de gestion de projet	✓	✓
Discussions d'équipe	✓	✓
Suivi du temps	✓	✓ (Avec App)
Outils de sécurité et de conformité	✓	✓
Test de performance de charge	✓ (Plan payant)	✓ (Avec App)
Test de performance des navigateurs	✓ (Plan payant)	✓ (Avec App)
Itérations et planification des sprints (y compris Burndown Chart)	✓ (Plan payant)	✓ (Avec App)
Dépendances des problèmes	✓ (Plan payant)	✓

# 01-Manipuler les outils de gestion de versions

## Git/Gitlab :

### Comparaison Github et Gitlab

#### Quelques différences majeures entre gitlab et gitlab

GitHub	GitLab
Les issues peuvent être suivies dans plusieurs repositories	Les issues ne peuvent pas être suivies dans plusieurs repositories
Repositories privés payants	Repositories privés gratuits
Pas d'hébergement gratuit sur un serveur privé	Hébergement gratuit possible sur un serveur privé
Intégration continue uniquement avec des outils tiers (Travis CI, CircleCI, etc.)	Intégration continue gratuite incluse
Aucune plateforme de déploiement intégrée	Déploiement logiciel avec Kubernetes
Suivi détaillé des commentaires	Pas de suivi des commentaires
Impossible d'exporter les issues au format CSV	Exportation possible des issues au format CSV par e-mail
Tableau de bord personnel pour suivre les issues et pull requests	Tableau de bord analytique pour planifier et surveiller le projet
Interface graphique un peu compliqué	Interface graphique clair et facile

# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

### Ce que vous allez apprendre dans ce chapitre :

- Notions des métriques de la qualité du code
- Présentation des outils existants de mesure de la qualité du code
- Présentation et installation de SonarQube
- Analyse de code source avec SonarQube à travers la génération des rapports

**15 heures**



# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

1. **Notions des métriques de la qualité du code**
2. Présentation de SonarQube
3. Installation de SonarQube
4. Analyse de code source avec SonarQube
5. Intégration de SonarQube avec les outils ALM et Configuration



## 02-Manipuler l'outil de mesure de la qualité du code(SonarQube) :

### Notions de métriques

#### Définitions

- **Une métrique** : est une caractéristique (ou une propriété) d'une application.
- **Métrique logicielle** : mesure d'une propriété d'un logiciel (par exemple le nombre de lignes de codes).
- **Exemples de métriques** :
  - Lignes de codes
  - Nombre de méthodes par classe
  - Couplage afférent/efférent
  - Niveau d'abstraction
  - Indice Instabilité
  - Complexité cyclomatique(compter le nombre de points de décision (if, case, while, ...)).
- **Pas de métrique "absolue"** : la pertinence de chaque métrique dépend du projet et surtout de l'interprétation qui en est faite.

## 02-Manipuler l'outil de mesure de la qualité du code(SonarQube) :

### Notions de métriques

#### Définitions

- On distingue deux types de qualités :
  - La **qualité du code (qualité structurelle du logiciel )** : concerne la manière avec laquelle une fonctionnalité est implémentée, et inclut entre autres la robustesse, la maintenabilité, la lisibilité ou encore l'évolutivité.
  - La **qualité d'un logiciel (qualité fonctionnelle)**: concerne le résultat final.
- la **mesure de la qualité du code** peut en partie être mesurée automatiquement par des outils d'analyse statique du code source. Ces outils sont disponibles pour la plupart des langages et proposent de **nombreuses métriques** et conventions standards.
- La **qualité du code** peut être également évaluée par d'autres critères comme :
  - L'architecture qui inclut la maintenabilité, l'évolutivité, la performance et la pertinence
  - La documentation,
  - La portabilité et la sécurité.
  - La fiabilité
  - Le nombre de bugs connus.
- Tous ces critères sont fortement dépendants du contexte comme la spécificité du logiciel ou programme au niveau fonctionnel, son avenir mais aussi d'autres facteurs importants comme le marché et ses utilisateurs.

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Exemples d'outils

#### Exemples d'outil pour analyser le code:

---

**Cobertura**

---

**Crap4J**

---

**PMD**

---

**FindBugs**

---

**Eclipse Metrics**

---

**Jdepend**

---

**Review Board**

---

**Crucible**

---

**Raxis**

---

**Technologies RIPS**

---

**PVS-Studio**

---

**Kiuwan**

---

**CodeScene**

---

**Expert visuel**

---

**Veracode**

---

**CodeSonar**

---

**SonarQube**



# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

1. Notions des métriques de la qualité du code
- 2. Présentation de SonarQube**
3. Installation de SonarQube
4. Intégration de SonarQube avec les outils ALM et Configuration
5. Analyse de code source avec SonarQube



## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Présentation de SonarQube

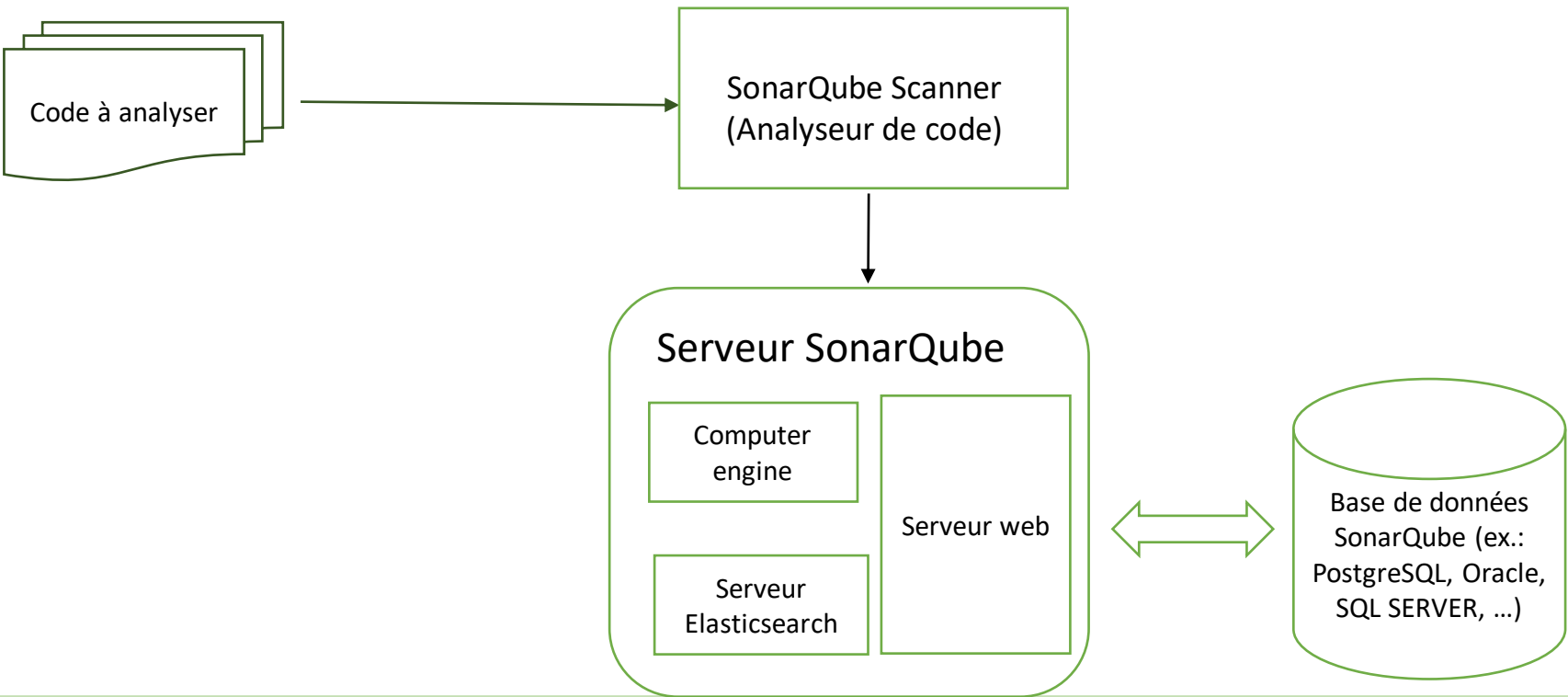
#### C'est quoi l'outil SonarQube ?

- **SonarQube** est un logiciel open source de mesure de la qualité du code source de projets de développement.
- Il permet d'obtenir des informations sur la qualité au niveau du projet, du fichier ou d'un module et donne des indications sur chaque problème de qualité détecté et le temps de remédiation.
- Son périmètre est le code source, le design ainsi que les tests unitaires, il supporte plus d'une vingtaine de langages de programmation, dont **C/C++,C#,Java, Python, PHP, JavaScript..**

# 02-Manipuler l’outil de mesure de la qualité du code (SonarQube) : Présentation de SonarQube

## Architecture de SonarQube ?

- D'un point de vue architectural, Sonar est composé de plusieurs couches :
  - Un exécuteur dont le but sera de lancer un certain nombre d'outils d'analyse, et d'en agréger les résultats ;
  - Le serveur web qui permet la navigation et la consultation des analyses réalisées sur les projets ;
  - Une base de données, qui stocke et historise les informations sur les projets surveillés par Sonar ;



## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Présentation de SonarQube

#### Les types de métriques avec SonarQube?

- **SonarQube** génère un rapport consultable via un navigateur sur :
  - ✓ Densité des commentaires
  - ✓ Taux de couverture des tests unitaires
  - ✓ des conventions de nommage
  - ✓ Respect des règles de codage et des bonnes pratiques
  - ✓ Détection de bogues
  - ✓ Détection de code mort
  - ✓ Détection de code dupliqué
  - ✓ Complexité, complexité cognitive)
  - ✓ Score de maintenabilité, fiabilité et sécurité du code (complexité cyclomatique)
  - ✓ Dette technique (estimation du temps nécessaire pour fixer tous les problèmes détectés )
  - ✓ Sonar couvre les 7 axes de la qualité du code : **architecture & design , documentation ,respect des standards de codage ,non duplication du code , tests unitaires ,complexité ,bogues potentiels.**
- Il est possible d'automatiser l'analyse avec un serveur d'intégration continue (**ex. Jenkins, Travis**).

# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) : Présentation de SonarQube

## Tableau du bord avec SonarQube?

- **SonarQube** classe les défauts logiciels selon **3** catégories :
  - Les **bugs** : anomalies évidentes du code. Ils impactent la **fiabilité** (reliability) de l'application.
  - Les **vulnérabilités** : faiblesses du code pouvant nuire au système. Elles impactent la **sécurité** de l'application.
  - Les **code smells** : anti-patterns (ou anti-patterns). Ils impactent la **maintenabilité** de l'application ,en général les défauts pratiques dans le code d'un programme comme code dupliqué , répété , commentaires non nécessaires ...
- **SonarQube** affiche un tableau de bord résumé à l'exécution d'un projet montrant les scores de maintenabilité, fiabilité et sécurité, le taux de couverture des tests, le taux de duplications, la taille du projet ainsi que les langages des sources.



Exemple tableau du bord

02-Manipuler l’outil de mesure de la qualité  
du code (SonarQube) :  
Exemples de métriques avec Sonarqube



1. Scores de maintenabilité, fiabilité et sécurité (MFS):

Score	Fiabilité	Sécurité	Maintenabilité
A	0 bug	0 vulnérabilité	0,00<=ratio dette technique<=0,05
B	Au moins 1 bug mineur	Au moins 1 vulnérabilité mineure	0,06<=ratio dette technique<=0,1
C	Au moins 1 bug majeur	Au moins 1 vulnérabilité majeure	0,11<=ratio dette technique<=0,20
D	Au moins 1 bug critique	Au moins 1 vulnérabilité critique	0,21<=ratio dette technique<=0,50
E	Au moins 1 bug bloquant	Au moins 1 vulnérabilité bloquante	0,51<=ratio dette technique<=1

Tableau montrant comment calculer les scores de MFS

(\*) **Le ratio de la dette technique** est le ratio entre le coût pour remédier aux problèmes de type code smell et le coût de développement de l’application.

La formule de calcul est la suivante : **coût total de remédiation des issues / (coût pour développer une ligne de code x nombre de lignes de code)**

L’idée est de déterminer si réécrire l’application est plus rentable que corriger tous les problèmes. Lorsque le ratio est trop grand, il est préférable de réécrire l’application de zéro plutôt que d’essayer de réduire la dette en corrigeant les problèmes.

Le nombre de minutes pour remédier à chaque issue est stocké dans la base de données.

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Exemples de métriques avec Sonarqube

#### 2. Mesures pour la documentation

- Quelques variables utilisés :
  - **Comment lines** = nombre de lignes contenant des commentaires de documentation ou du code commenté, les lignes de commentaires non significatives sont ignorées (ligne vide ou avec le caractère spécial \*)
  - **Comments (%)** = densité de commentaire calculée par la formule :
 
$$\text{Comment lines} / (\text{Lines of code} + \text{Comment lines}) * 100$$

Densité de 50% = autant de lignes de code que de commentaires  
 Densité de 100% = le fichier ne contient que des commentaires
  - **Commented-out LOC** = nombre de lignes de code commentées.

#### 3. Mesures pour la duplication du code :

- Les variables utilisés pour le calcul :
  - **Duplicated blocks** = nombre de blocs de lignes dupliqués. Un bloc de code est considéré comme dupliqué s'il y a au moins 100 tokens successifs dupliqués sur 10 lignes de code (PHP, JavaScript)
  - **Duplicated files** = nombre de fichiers impliqués dans des duplications
  - **Duplicated lines** = nombre de lignes impliquées dans des duplications
  - **Duplicated lines (%)** = densité de duplication calculée par la formule :  $\text{Duplicated lines} / \text{Lines} * 100$

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Exemples de métriques avec Sonarqube

#### 4. Mesures pour les tests :

- Les variables utilisés pour faire cette mesure :
  - **Nombre de tests unitaires** (unit tests)
  - **nombre de tests unitaires** qui ont échoué (unit test errors)
  - **nombre d'exception** lors de l'exécution (unit test failures)
  - **nombre de lignes non couvertes** (uncovered lines)
  - **nombre de conditions non couvertes** (uncovered conditions)
  - **couverture** : calcul =  $(CT + CF + LC) / (2*B + EL)$

*CT : condition évaluées à true au moins une fois*

*CF : conditions évaluées à false au moins une fois*

*LC : lignes couvertes*

*B : nombre total de conditions*

*EL : nombre total de lignes exécutables*

#### 5.Mesures de complexité :

- Les variables utilisés pour faire cette mesure :
  - **Complexity** = Complexité totale du projet
  - **Complexity / method** = Complexité moyenne par fonction
  - **Complexity / file** = Complexité moyenne par fichier
  - **Complexity / class** = Complexité moyenne par classe
- Le calcul de la complexité d'une méthode se base sur la complexité cyclomatique. Elle est minimum de 1 (1 pour la méthode). Elle est incrémentée de 1 pour chaque if, case, for, while, throw, catch, return (sauf s'il est en dernière instruction de la fonction), &&, ||, opérateur conditionnel ternaire



# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

1. Notions des métriques de la qualité du code
2. Présentation de SonarQube
- 3. Installation de SonarQube**
4. Intégration de SonarQube avec les outils ALM et Configuration
5. Analyse de code source avec SonarQube



# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) : Installation de sonarqube

## Installation de SonarQube :

### 1. Prérequis:

- **Prérequis côté serveur** : installer JDK sur <https://www.oracle.com/java/technologies/downloads/>
- **Prérequis de base de données:** Postgres ou Oracle ou SqlServer
- **Prérequis côté client** : navigateur web récent
- **JavaScript activé.**

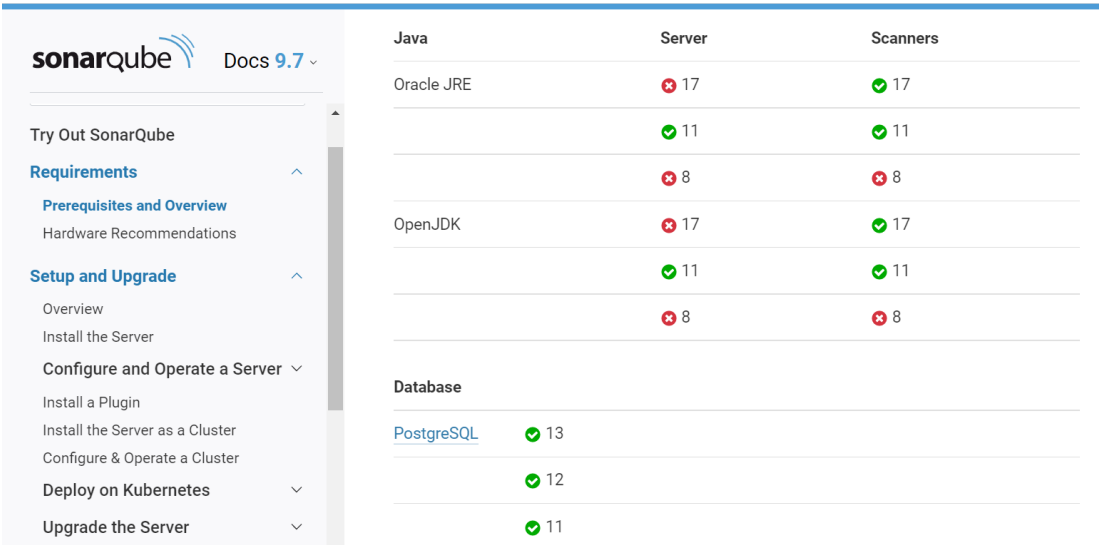
On peut vérifier les prérequis ainsi que leurs versions valables en visitant ce lien :

<https://docs.sonarqube.org/latest/requirements/requirements/>

### 2. Télécharger **Sonarqube community** sur:

<https://www.sonarqube.org/downloads/>

### 3. Décompressez le fichier dans n'importe quel dossier (le nom du dossier de ne doit commencer par un chiffre)



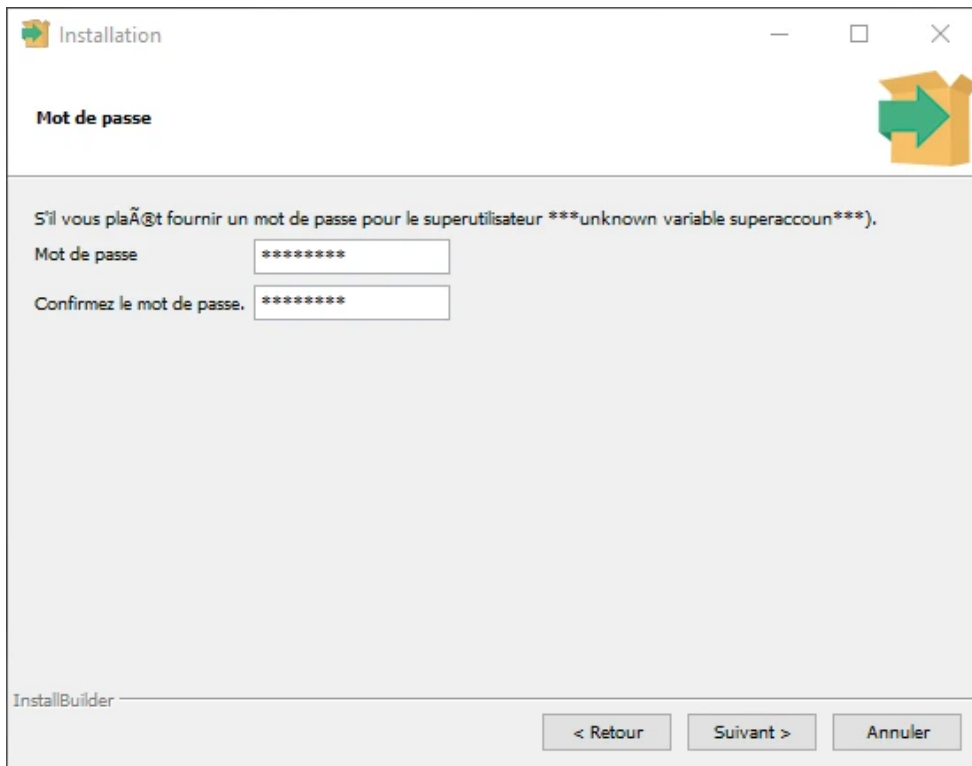
Java	Server	Scanners
Oracle JRE	✗ 17	✓ 17
	✓ 11	✓ 11
	✗ 8	✗ 8
OpenJDK	✗ 17	✓ 17
	✓ 11	✓ 11
	✗ 8	✗ 8
<b>Database</b>		
<a href="#">PostgreSQL</a>	✓ 13	
	✓ 12	
	✓ 11	

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Installation de sonarqube

#### Installation et configuration du serveur de bases de données Postgres :

En suivant les instructions de l'installation, configurez le mot de passe (ex.:12345) ainsi que le port du SGBD (ex.: 5432) :



**Mot de passe**

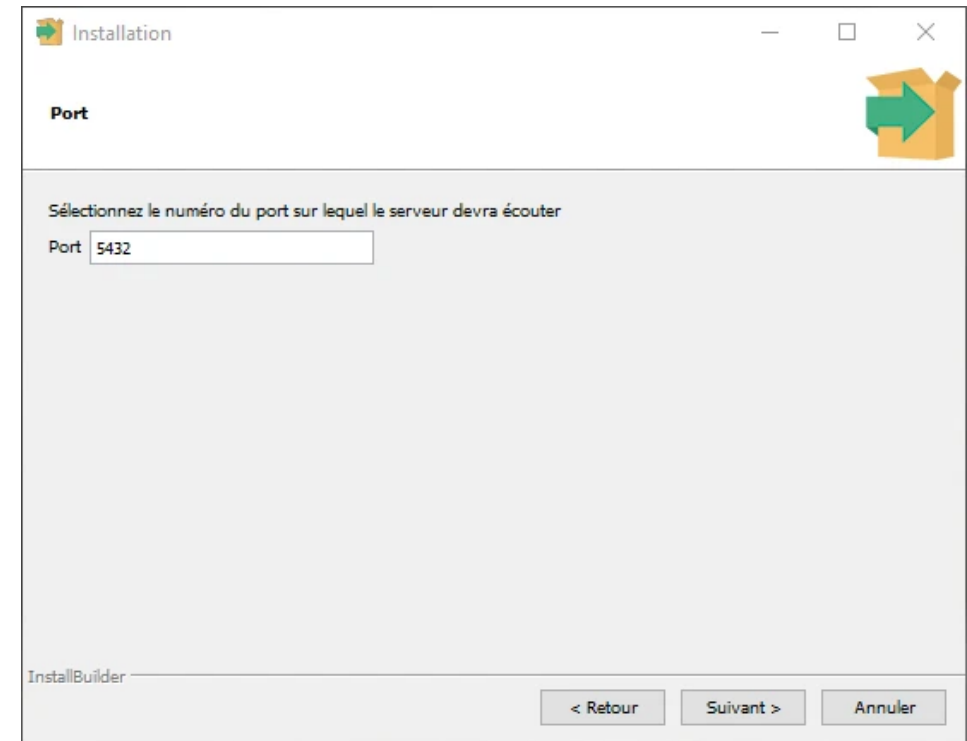
S'il vous plaît fournir un mot de passe pour le superutilisateur \*\*\*unknown variable superaccoun\*\*\*).

Mot de passe

Confirmez le mot de passe.

InstallBuilder

< Retour Suivant > Annuler



**Port**

Sélectionnez le numéro du port sur lequel le serveur devra écouter

Port

InstallBuilder

< Retour Suivant > Annuler

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Installation de sonarqube

#### Configuration de SonarQube :

1. Après avoir décompressé le fichier zip téléchargé: sonarqube-9.7.1.62043.zip (dans notre cas)
2. Accédez au dossier config et ouvrez le fichier « sonar.properties »:
  - a. Décommentez et modifiez les lignes suivantes, ainsi:

```
sonar.jdbc.username=postgres
sonar.jdbc.password=12345
.....
sonar.jdbc.url=jdbc:postgresql://localhost:5432/postgres
... .
sonar.web.host=0.0.0.0
.....
sonar.web.port=9000
... .
```

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Installation de sonarqube

#### Démarrage de SonarQube :

1. Dans le dossier bin choisir le dossier relative à votre système d'exploitation

) sonarqube-9.5.0.56709 > bin

Nom	Modifié le	Type	Taille
jsw-license	10/06/2022 06:14	Dossier de fichiers	
linux-x86-64	10/06/2022 06:14	Dossier de fichiers	
macosx-universal-64	10/06/2022 06:14	Dossier de fichiers	
windows-x86-64	10/06/2022 06:14	Dossier de fichiers	

2. Cliquer sur **startSonar.bat** pour le démarrer :

Nom	Modifié le	Type
lib	06/11/2022 12:34	Dossier de fichiers
SonarService.bat	28/10/2022 06:26	Fichier de commande
StartSonar.bat	28/10/2022 06:26	Fichier de commande

3. Aller dans un navigateur et taper l'url **localhost:9000** pour afficher l'interface admin de **SonarQube**:

4. Connectez avec les identifiants suivants:

- login: **admin**
- password: **admin**

# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

1. Notions des métriques de la qualité du code
2. Présentation de SonarQube
3. Installation de SonarQube
- 4. Analyse de code source avec SonarQube**
5. Intégration de SonarQube avec les outils ALM et Configuration

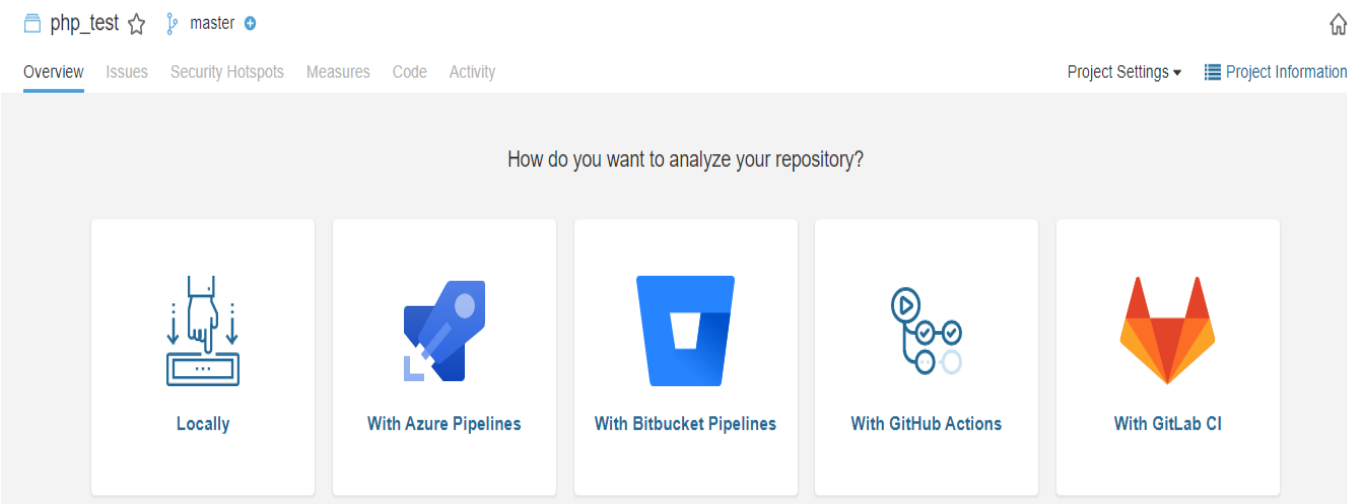


# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

- Maintenant que vous êtes connecté à votre instance **SonarQube** locale, analysons un projet :
  - Cliquez sur le bouton **Créer un nouveau projet** (vous pouvez choisir entre **locally** ou **importer des logiciels des versions**)
  - Donnez à votre projet une **clé de projet** et un **nom complet**



### Create a project

All fields marked with \* are required

**Project display name \***

✓

Up to 255 characters. Some scanners might override the value you provide.

**Project key \***

✓

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '\_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Set Up





# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

4.Par la suite ,choisir le langage de votre projet et le système d'exploitation :

php\_test ☆ master +

Overview Issues Security Hotspots Measures Code Activity

Analyze your project

We initialized your project on SonarQube, now it's up to you to launch analyses!

1 Provide a token

2 Run analysis on your project

What option best describes your build?

Maven

Gradle

.NET

Other (for JS, TS, Go, Python, PHP, ...)

php\_test ☆ master +

Overview Issues Security Hotspots Measures Code Activity

1 Provide a token

2 Run analysis on your project

What option best describes your build?

Maven

Gradle

.NET

Other (for JS, TS, Go, Python, PHP, ...)

What is your OS?

Linux

Windows

macOS

# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

5.Ensuite ,télécharger et installer **SonarScanner** , selon votre **SE** sur : <https://docs.sonarqube.org/latest/analysis/scan/sonarscanner>

### SonarScanner

By [SonarSource](#) | GNU LGPL 3 | [Issue Tracker](#)

**4.7**

[Show more versions](#)

2022-02-22

Ease import of custom certificates with the Docker image,  
update embedded JRE 11

[Linux 64-bit](#) [Windows 64-bit](#) [Mac OS X 64-bit](#) [Docker](#)

[Any \(Requires a pre-installed JVM\)](#) [Release notes](#)

The SonarScanner is the scanner to use when there is no specific scanner for your build system.

**Le SonarScanner** est un outil utilisé par **SonarQube** pour scanner votre projet vers **SonarQube** pour que ce dernier puisse lire le langage de programmation utilisé.

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

Analyser le code avec SonarQube

### Analyser un projet avec SonarQube : Configurer sonnar-scanner

1. Décompressez le fichier téléchargé dans un répertoire de votre choix ;
2. Sous le dossier « **config** », accédez au fichier « **sonar-scanner.properties** » et décommentez la ligne suivante :

```
#----- Default SonarQube server  
sonar.host.url=http://localhost:9000
```

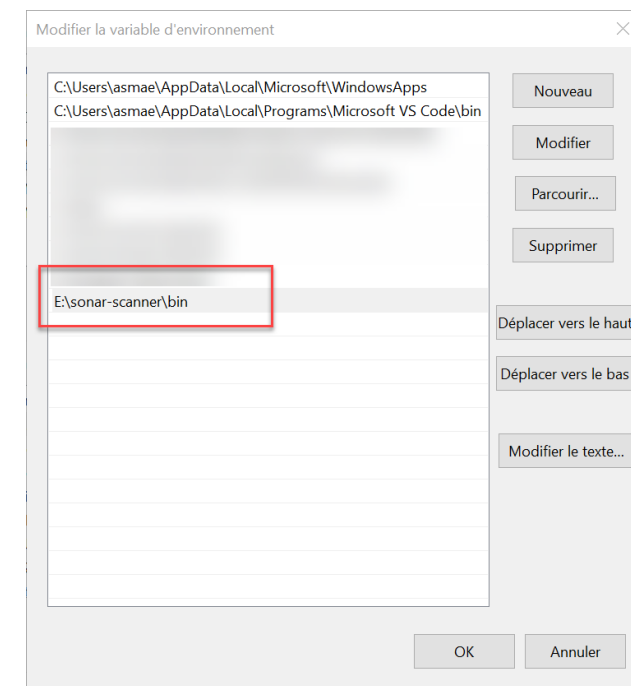
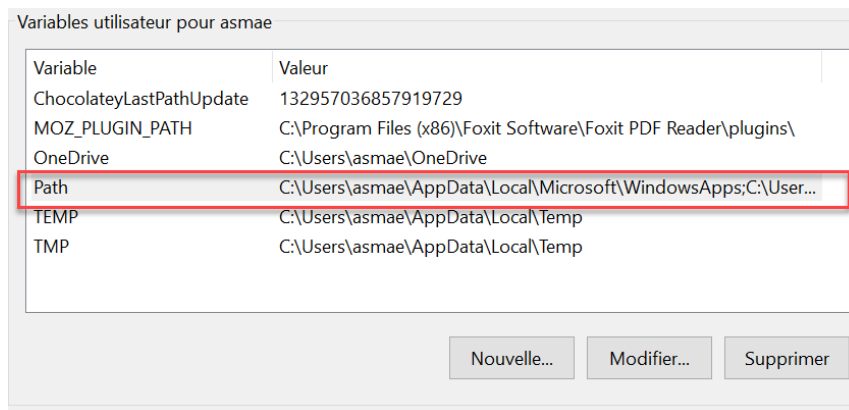
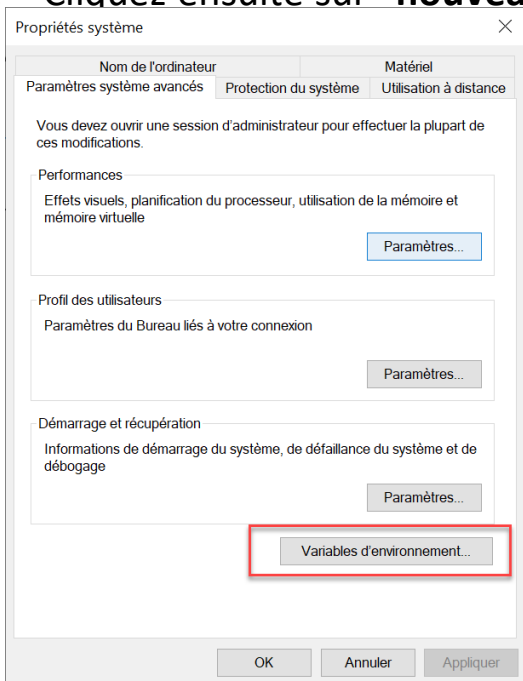
# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube : Configurer sonnar-scanner

Afin de faciliter l'appel au sonar scanner, il vaut mieux l'ajouter au path windows, pour cela :

1. Bouton doit sur l'icone windows, puis accédez à **"système"**
2. Sur le menu à droite, accédez à **"Paramètres avancés du système"**
3. Cliquez sur **"Variables d'environnement"**
4. Sélectionnez **"path"** puis cliquez sur **"modifier"**
5. Cliquez ensuite sur **"nouveau"** et tapez le chemin vers le dossier bin de sonnar-scanner;



# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube : Configurer sonnar-scanner

Pour commencer l'analyse du projet:

1. Ajouter un nouveau fichier à votre projet; le fichier devra prendre le nom : **sonar-project.properties** et doit avoir le contenu suivant :

```
# must be unique in a given SonarQube instance
```

```
sonar.projectKey=php_test
```

La propriété **Sonar.projectkey** devra avoir **le nom de votre projet**

```
# --- optional properties ---
```

```
# defaults to project key
```

```
#sonar.projectName=My project
```

```
# defaults to 'not provided'
```

```
#sonar.projectVersion=1.0
```

```
# Path is relative to the sonar-project.properties file. Defaults to .
```

```
#sonar.sources=.
```

```
# Encoding of the source code. Default is default system encoding
```

```
#sonar.sourceEncoding=UTF-8
```

# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

Copiez la commande proposée par Sonarqube et collez la dans le terminal de visual studio :

1 Provide a token

Analyze "php\_test":sqp\_5e2aa4d2f808a5c67fb8314960b2439d62bb63b3

2 Run analysis on your project

What option best describes your build?

Maven

Gradle

.NET

Other (for JS, TS, Go, Python, PHP, ...)

What is your OS?

Linux

Windows

macOS

Download and unzip the Scanner for Windows

Visit the [official documentation of the Scanner](#) to download the latest version, and add the `bin` directory to the `PATH` environment variable

Execute the Scanner

Running a SonarQube analysis is straightforward. You just need to execute the following commands in your project's folder:

```
sonar-scanner.bat -D"sonar.projectKey=php_test" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=sqp_5e2aa4d2f808a5c67fb8314960b2439d62bb63b3"
```

Copy

Please visit the [official documentation of the Scanner](#) for more details.

EXPLORER

GESTIONVOITURES\_MVC

.scannerwork

assets

Controller

Model

View

index.php

sonar.properties

sonar.properties

sonar.properties

# must be unique in a given SonarQube instance

sonar.projectKey=php\_test

# --- optional properties ---

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

cmd

Microsoft Windows [version 10.0.19044.2130]

(c) Microsoft Corporation. Tous droits réservés.

C:\xampp\htdocs\PHP\gestionVoitures\_mvc>sonar-scanner.bat -D"sonar.projectKey=php\_test" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=sqp\_5e2aa4d2f808a5c67fb8314960b2439d62bb63b3"

# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

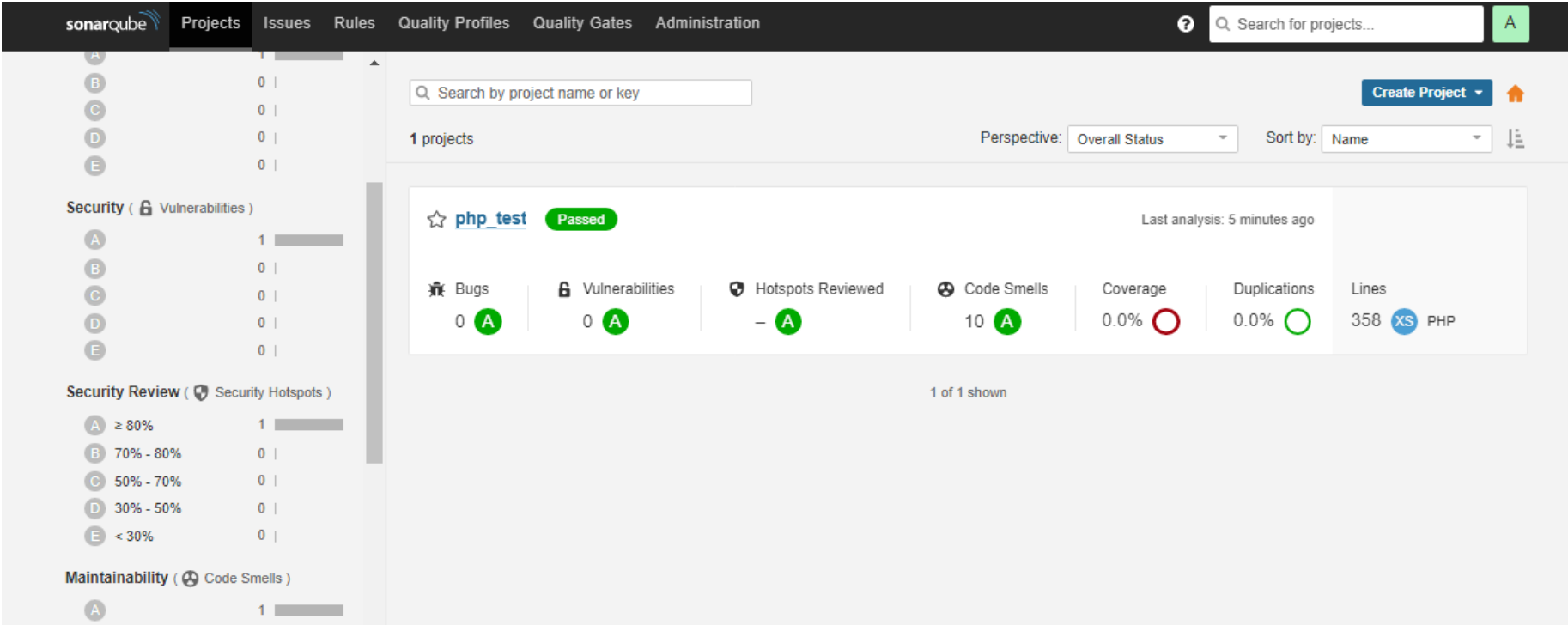
## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

8. A la fin de l'analyse un url s'affiche pour voir les résultats ,copier le et accéder via un navigateur :

More about the report processing at <http://localhost:9000/api/ce/task?id=AYImYtoQQCeb5V2Rt8oU>  
 Analysis total time: 1:35.634 s

Le tableau du bord de l'analyse du code s'affiche:

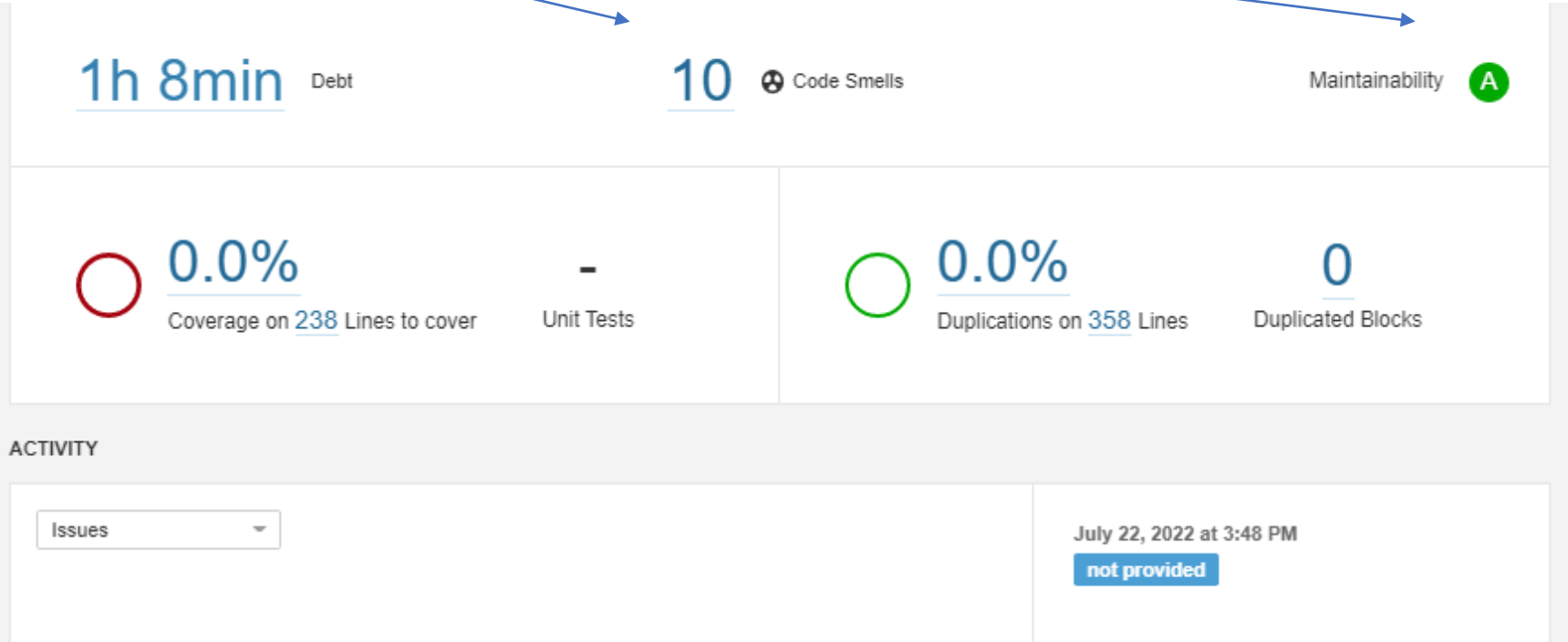


# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

Analyser le code avec SonarQube

## Analyser un projet avec SonarQube:

- On peut accéder à chaque métrique en cliquant sur son nom :
- Les numéros sont le nombre d'erreurs dans le code





# 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

## Analyser le code avec SonarQube

### Analyser un projet avec SonarQube:

- On peut voir les erreurs dans le code et avec la suggestion de correction:

```
body {
    font-size: 17px;
    font-family: arial;
```

**Unexpected missing generic font family** Why is this an issue?

Bug Major Open Not assigned 1min effort [Comment](#)

```
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
```

**Define and throw a dedicated exception instead of using a generic one.** Why is this an issue? 16 minutes ago L11 [🔗](#)

Code Smell Major Open Not assigned 20min effort [Comment](#) [🔗](#) cwe, error-handling

```
$colors = [
    1 => 'red',
    2 => 'green',
    3 => 'blue',
];

// echo $colors[1];
```

**Remove this commented out code.** Why is this an issue? 16 minutes ago L36 [🔗](#)

Code Smell Major Open Not assigned 5min effort [Comment](#) [🔗](#) unused

php2 [/index.html](#) [🔗](#) [See all is](#)

```
<!DOCTYPE html>
<html>
```

**Add "lang" and/or "xml:lang" attributes to this "<html>" element** Why is this an issue? 6 minutes ago L2 [🔗](#)

Bug Major Open Not assigned [Comment](#) [🔗](#) accessibility, wcag2-a

# CHAPITRE 2

## Manipuler l'outil de mesure de la qualité du code (SonarQube)

1. Notions des métriques de la qualité du code
2. Présentation de SonarQube
3. Installation de SonarQube
4. Analyse de code source avec SonarQube
5. **Intégration de SonarQube avec les outils ALM et Configuration**



## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

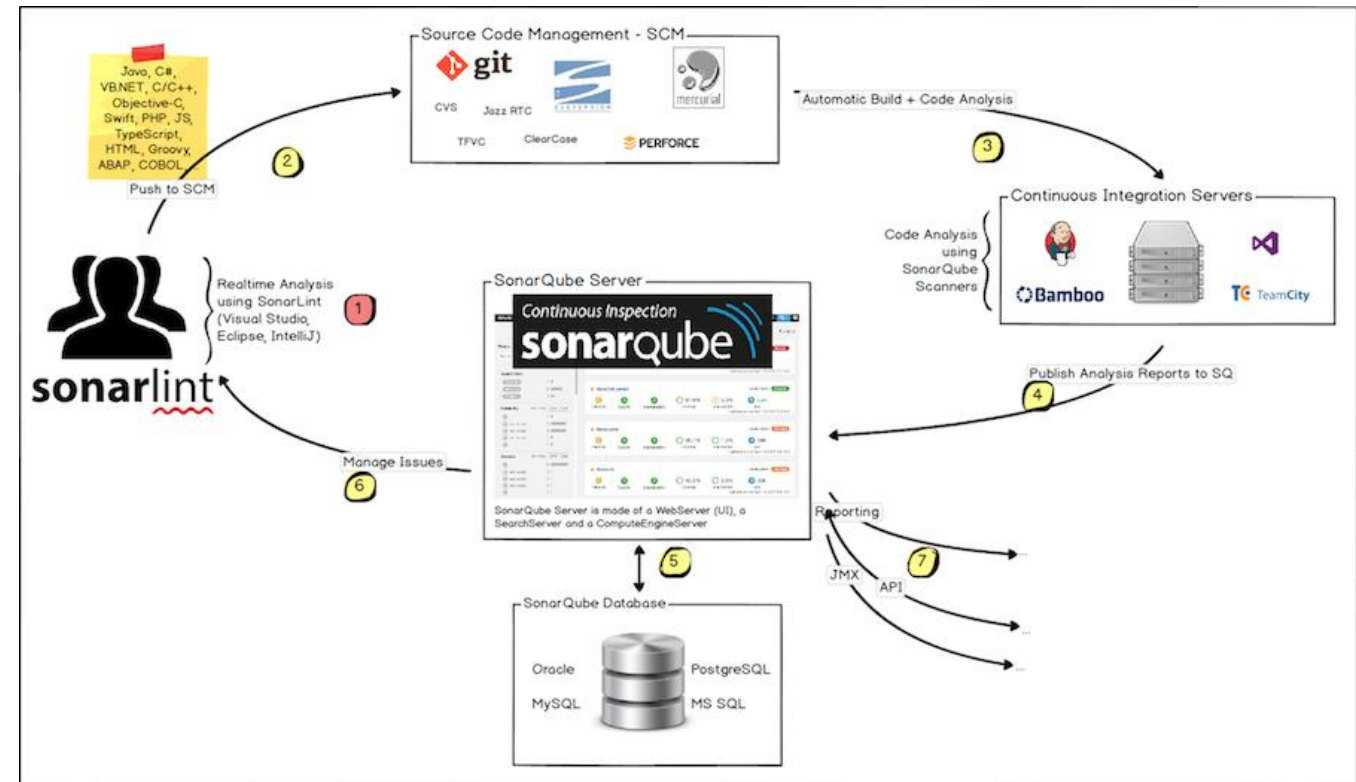
### Intégration de SonarQube avec les outils ALM

#### Intégration de SonarQube avec les outils ALM :

Le schéma suivant montre comment **SonarQube** s'intègre à d'autres outils **ALM**<sup>1</sup> dans un grand projet d'équipe où les différents composants de **SonarQube** sont utilisés

#### <sup>1</sup> L'application life cycle

**management** ou **ALM** :est le processus global de gestion du cycle de vie d'un logiciel. C'est l'ensemble des moyens nécessaires au développement ou à la maintenance d'une application. Cela concerne les activités d'ingénierie logicielle comme les activités de gestion de projet.



Source : <https://docs.sonarqube.org/>

## 02-Manipuler l'outil de mesure de la qualité du code (SonarQube) :

### Intégration de SonarQube avec les outils ALM

#### Intégration avec Application Life cycle management:

- Les rôles dans le schéma selon les numéros :

1. Les développeurs codent dans leurs IDE et utilisent **SonarLint** pour exécuter une analyse locale.
2. Les développeurs poussent leur code dans leur **SCM** préféré : **github, gitlab...**
3. Le **serveur d'intégration continue** déclenche une construction automatique et l'exécution du **SonarScanner** requis pour exécuter l'analyse **SonarQube**.
4. Le rapport d'analyse est envoyé au serveur **SonarQube** pour traitement.
5. **SonarQube** Server traite et stocke les résultats du rapport d'analyse dans la base de données **SonarQube** et affiche les résultats dans l'interface utilisateur.
6. Les développeurs examinent, commentent, contestent leurs problèmes pour gérer et réduire leur dette technique via l'interface utilisateur **SonarQube**.
7. Les managers reçoivent les rapports de l'analyse. Les opérateurs utilisent des API pour automatiser la configuration et extraire les données de **SonarQube**. Les opérateurs utilisent JMX pour surveiller le serveur **SonarQube**.

**SonarLint** est une extension IDE gratuite et open source qui identifie et vous aide à résoudre les problèmes de qualité et de sécurité pendant que vous codez. Comme un correcteur orthographique, SonarLint corrige les défauts et fournit des commentaires en temps réel et des conseils de correction clairs pour fournir un code propre dès le départ. Il supporte presque tous les IDE comme vscode ,eclipse ,JetBrains..

- Quelques règles avant la configuration de SonarQube :

- La plate-forme **SonarQube** ne peut pas avoir plus d'un serveur **SonarQube** et plus d'une base de données .
- Pour des performances optimales, chaque composant (serveur, base de données, analyseurs) doit être installé sur une machine distincte et la machine serveur doit être dédiée.
- Les **SonarScanners** évoluent en ajoutant des machines.
- Toutes les machines doivent être synchronisées.
- Le serveur **SonarQube** et la base de données **SonarQube** doivent être situés sur le même réseau.
- Les **SonarScanners** n'ont pas besoin d'être sur le même réseau que le serveur **SonarQube**.
- Il n'y a pas de communication entre les **SonarScanners** et la base de données **SonarQube**.