Cursus Développeur Front End

Introduction Git et GitLab

M2I Formation 2022

Glodie Tshimini

PLAN

Glodie Tshimini : contact@tshimini.fr

PLAN

Installations

Annexe

Installations

Glodie Tshimini : contact@tshimini.fr

Environnement de travail

- Installation Git
  - Laisser les paramètres par défaut
  - Vous pouvez modifier l'éditeur
- Installation Visual Studio Code
- Création compte GitHub
- Création de compte GitLab

I.

Git, GitHub et GitLab

Qu’est-ce que Git ?

- Système de gestion de versions
- Permet
  - Plusieurs versions du projet
  - Plusieurs collaborateurs travaillent sur le même projet
  - Historique du projet
- Repose sur
  - Snapshots (sauvegarde de l’état du dépôt à un moment précis)
  - Commits
  - Dépôts
    - Local
    - Distant
- Système décentralisé
  - Plusieurs dépôts locaux liés à un dépôt distant

Qu'est-ce que GitHub ?

- Service en ligne qui héberge des dépôts *git* (dépôts distants)

- Crée en 2008
- Racheté par Microsoft en 2018

- Travail collaboratif
  - Entreprises
  - Écoles
  - Open Source

- Plusieurs offres
  - Gratuites (offre étudiante)
  - Payantes

Qu'est-ce que GitLab ?

- Crée en 2011
- Concurrent de GitHub
- Propose plus de fonctionnalités
  - Possible d'héberger ses dépôts distants sur son propre serveur privé.
  - Définition des rôles des utilisateurs (documentation permissions)
    - Guest
    - Developer
    - Maintener
  - CI/CD
  - Etc.

Source image troispointzero.fr

Fichiers spéciaux d'un dépôt Git

README.md

Documentation du projet

.gitkeep

Conserver un dossier vide

.gitignore

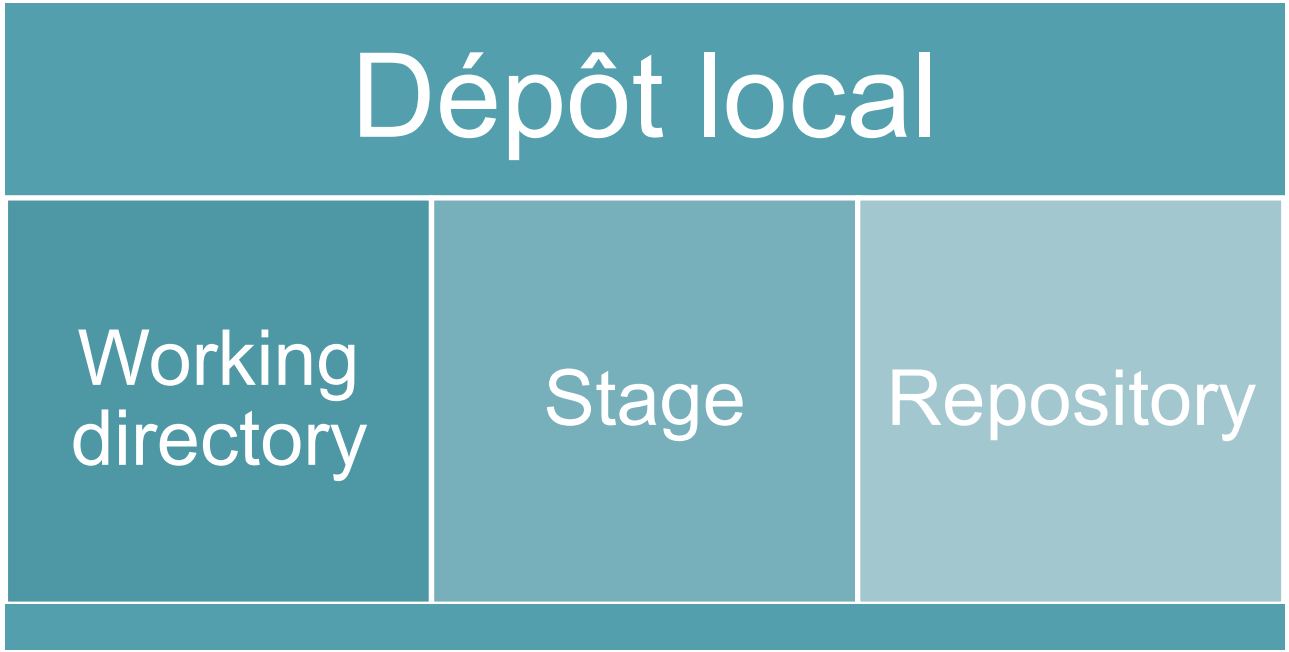
Liste fichiers et dossiers à ignorer

Fichier .gitignore

.gitignore

```
# Cache, temp and personal files

/.htaccess
*.log
npm-debug.log.*
.sass-cache/
/cache/*

#/img/*
/log/*
/upload/*
docs/phpdoc-sf/
#composer.lock
tests/Selenium/errorShots/
tests/Selenium/errorDumps/


admin134ntao8l/autoupgrade/*
admin134ntao8l/backups/*
admin134ntao8l/import/*

admin134ntao8l/import/*
!admin134ntao8l/import/.htaccess
!admin134ntao8l/import/index.php

themes/*/cache/*
```

3 sections de travail du dépôt local



II.

Commandes de base

Aide sur les commandes git

git {command} --help

- **{command}** à remplacer par une des commandes *git* que nous verrons tout au long de ce cours.

- La documentation complète sur la commande s'ouvrira via une page web ou une autre sortie selon la configuration de votre logiciel *git*.

Configurer et initialiser un dépôt git

Configuration minimale Git

- Au minimum
  - Nom et prénom
  - Email

```
Glodie@Glodie MINGW64 ~
$ git config --global user.name 'Glodie'

Glodie@Glodie MINGW64 ~
$ git config --global user.email 'contact@tshimini.fr'
```

- Autres
  - Editeur
  - Couleurs
  - Format de l'aide

À faire une seule fois sur votre ordinateur.

l'option global permet de vous identifier sur tous vos dépôts git de votre machine avec les informations fournies.

Créer un dépôt local

```
Glodie@Glodie MINGW64 /d/demo
$ git init
Initialized empty Git repository in D:/demo/.git/

Glodie@Glodie MINGW64 /d/demo (master)
$ ls -lah
total 44K
drwxr-xr-x 1 Glodie 197121 0 avr.  19 19:04 ./
drwxr-xr-x 1 Glodie 197121 0 avr.  19 19:04 ../
drwxr-xr-x 1 Glodie 197121 0 avr.  19 19:04 .git/
```

- Initialise un dépôt git
- Crée un **dossier caché *.git***

- Une fois le dépôt initialisé sur un répertoire de travail à l'aide de cette commande, vous n'avez plus besoin d'utiliser cette commande sur le projet en cours, hormis si vous avez supprimé le **dossier caché *.git*** .
- **Soyez vigilant à l'emplacement du dossier où vous allez initialiser votre dépôt.**

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 1

- Faites la configuration de git de manière globale en ajoutant votre nom, prénom et adresse e-mail de manière globale.

Liaison des dépôts

Ajouter un dépôt distant Git

- Synchroniser avec un dépôt distant (GitHub, GitLab, etc.)

git remote add origin {URL}

Remplacer {URL} par l'URL du dépôt distant
Origin est un **alias** de **l'URL**

Cloner un dépôt distant Git

- Cloner : Créer une copie d'un projet distant (depuis un dépôt distant) en local

git clone {URL}

Remplacer {URL} par l'URL du dépôt distant

Résumé lancement projet Git en local

Pas de dépôt local	Existence dépôt local et dépôt distant	Pas de dépôt local et existence d'un dépôt distant
git init	*git remote add origin {URL}*	*git clone {URL}*

Remplacer {URL} par l'URL de votre dépôt distant

Les branches

Gérer les branches d'un dépôt distant

Créer	Se déplacer sur une branche	Créer et se (dé)placer directement sur la nouvelle branche
git branch main	*git checkout main*	*git checkout -b feature/user*

Gérer les branches d'un dépôt distant

Lister toutes les branches	Renommer une branche	Supprimer une branche
git branch	*git branch -m old_name new_name*	*git branch -d feature/user*

```
Glodie@Glodie MINGW64 /e/formations/coderbase/poei-java-salesforce/1-git (main)
$ git branch
  feature/ex1-3
  feature/ex4
  feature/ex5
  feature/ex6
  feature/pratical-work/conflicts/merge/dev2/dev1
  feature/pratical-work/dev1
  feature/pratical-work/dev2
  feature/pratical-work/fix/conflicts/dev2/dev1
  feature/pratical-work/owner
* main
```

- Dans la liste de branches , * indique la branche courante

Bonnes pratiques branches

- **Anglais**

- Branche ***main*** (anciennement *master*)
  - Par défaut
  - **Protégé**
  - **Ne jamais travailler directement sur la branche *main* ou *master***

- Supprimer les branches inutiles après avoir effectué la fusion

Exercice

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

Exercice 2

- ❑ Créez un dépôt distant sur GitHub.
- ❑ Récupérez le dépôt distant en local.
- ❑ Créez et placez-vous sur une nouvelle branche nommée ***feature/exercises***.
- ❑ Créez un répertoire (dossier) sandwich.
- ❑ Créez un fichier à l'intérieur du répertoire sandwich nommé ***burger.md*** qui contient la liste des ingrédients qui compose un burger (un ingrédient par ligne).
- ❑ Dans le fichier README.md, ajoutez une photo (documentation ajout photo markdown).

Exemple de rendu final :

Steak

Salade

Tomate

Cornichon

Fromage

Versionner son code

Ajouter fichiers/dossiers dans l'index

Individuel	Tous (modifiés, supprimés et nouveaux)
git add fichier.txt dossier/	***git add --all***

- **Attention** avec l'option *--all*, soyez vigilant sur les fichiers qui seront ajoutés dans *l'index* après l'exécution de cette commande.

Commiter

En saisissant directement le message du commit sur ligne	À l'aide de l'éditeur configuré pour git (permet de saisir sur plusieurs lignes)
git commit -m "first commit"	***git commit***

- Avec l'éditeur
- Pour passer en mode insertion **Echap + i**
- Insérer votre message de commit
- Pour sortir et enregistrer votre message
- **Echap**
- **:x Entrez**

Commiter sans modifier l'historique

Sans modifier le message du dernier commit	En modifiant le message du dernier commit
git commit --amend --no-edit	***git commit --amend***

- ***--amend*** permet de modifier le message du dernier commit
- ***--amend --no-edit*** va affecter les fichiers dans l'index au précédent commit

Bonnes pratiques Commit

- **Anglais**
- Pas de caractères spéciaux
- **Motif des modifications**

git commit -m ‘update contact form, add GPDR requirements’

- Il existe des conventions de nommage

Convention de nommage Angular

Exercice

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

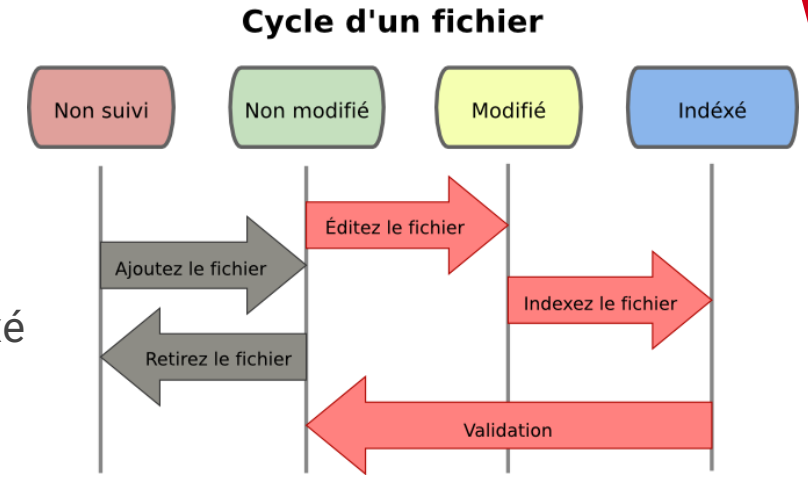
Exercice 3

- Préparez le fichier *burger.md* pour le commit.
- Commitez toutes les modifications survenues sur votre branche en cours.

État et historique du dépôt

4 états d'un fichier Git

- **Untracked**
  - Non versionné
- **Unmodified**
  - Versionné
- **Modified**
  - Versionné et modifié en attente d'être indexé
- **Staged**
  - Prise en compte lors d'une nouvelle version



Source image djibril.developpez.com

Etat du dépôt

git status

- Fichiers et/ou dossiers
  - Modifiés
  - Supprimés
  - Nouveaux (à ajouter dans l'index)
  - Ajoutés (présents) dans l'index

```
Glodie@Glodie MINGW64 /e/formations/coderbase/cda_2itech/0-interns/1-intro_git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        0-guide_installation/
        1-exercices/
        2-evaluation/
        README.md

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

Historique des commits

Détaillé	Sur une ligne	Graphique
git log	*git log --oneline*	*git log --graph*

```
commit b161417b432e4515f0e3b27e4e757bc35bd84739 (HEAD
Author: Glodie <contact@tshimini.fr>
Date:   Wed Dec 7 15:50:04 2022 +0100

    feat: exercise 7 bis interns proposals

commit 01c0bd16fa4178c3fca78fccd500b3dd677c8531
Author: Glodie <contact@tshimini.fr>
Date:   Wed Dec 7 14:28:47 2022 +0100

    feat: correction of exercise 7 both AlgoBox & JS

commit b3f23ab3fdb9c473b30eaa4a64831de0f995e572
Author: Glodie <contact@tshimini.fr>
Date:   Wed Dec 7 11:52:13 2022 +0100

    feat: basics JS

commit 43c1a6c7694248f644ff31dabdcdb515774b3568
Author: Glodie <contact@tshimini.fr>
Date:   Wed Dec 7 09:04:02 2022 +0100
```

```
*   commit c57287ff437ab009ff16e07c6ddb3dbd2bee88e4 (HEAD -> feature/front/full)
|\  Merge: eaf46cc 1417b4c
| | Author: glodie <glodie.tshimini@gmail.com>
| | Date:   Thu Dec 1 20:10:05 2022 +0100
| |
| |     Merge branch 'feature/front/jquery/ex6' into feature/front/full
| |
| * commit 1417b4cd8c67ae86ec3394d6f9e4ab40f362bf0d (origin/feature/front/jquery/ex
| | Author: glodie <glodie.tshimini@gmail.com>
| | Date:   Tue Nov 29 19:33:06 2022 +0100
| |
| |     feat: correction of exercise 6
| |
* |   commit eaf46cc0ba295e729faf7d93f6c225624556581e
|\ \  Merge: b52e9ed 758768b
| | | Author: glodie <glodie.tshimini@gmail.com>
| | | Date:   Thu Dec 1 20:09:37 2022 +0100
| | |
| | |     Merge branch 'feature/front/local-storage/ex4' into feature/front/full
| | |
| * | commit 758768bf0afd75205fd07e714e144772911dc2d1 (origin/feature/front/local-s
| |/  Author: glodie <glodie.tshimini@gmail.com>
| |   Date:   Mon Nov 28 21:51:59 2022 +0100
| |
| |       feat: correction of exercise4
| |
* |   commit b52e9ed54ec6248cadd1e3ea46dee856cdee69cf
|\ \  Merge: 9ec6626 26055a5
| | | Author: glodie <glodie.tshimini@gmail.com>
| | | Date:   Thu Dec 1 20:09:21 2022 +0100
| | |
| | |     Merge branch 'feature/front/async/ex3' into feature/front/full
| | |
```

Exercice

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

Exercice 4

- ❑ Créez quelques autres sandwichs (*hot_dog.md et jambon_beurre.md*).
- ❑ Modifiez le contenu de *burger.md* .
- ❑ Regardez l'état de votre dépôt.
- ❑ Ajoutez des photos.
- ❑ Effectuez au moins 5 modifications au total sur l'ensemble de vos fichiers.

  Chaque modification doit donner lieu à un commit.

  5 modifications = 5 *commits* différents.
- ❑ Regardez l'historique de vos *commits*.

Synchroniser les dépôts local et distant

Envoyer vos modifications vers dépôt distant

- Pusher (pousser) du dépôt local vers le dépôt distant *GitHub*

git push origin main

Récupérer mises à jour sans fusionner

- Récupérer en local les MAJ du dépôt distant *GitHub* sans fusionner

git fetch

Fusionner 2 branches

- Peut créer un commit de fusion automatique
- Se placer sur la branche de réception

Fusionner 2 branches	Annuler la fusion
git merge feature/newsletter	*git merge --abort*

Fusionner 2 branches

- Peut créer un commit de fusion automatique
- Fusionner 2 branches (depuis la branche *main* du dépôt distant)

git pull origin main

- ***Pull = fetch + merge***

Gestion des conflits

- Quand ?
  - **Fusion des branches**

- Pourquoi ?
  - **Modification du même fichier aux mêmes endroits dans les 2 branches**

- Comment résoudre le problème ?

1. **Choisir la version à conserver avec ces collaborateurs**
   - Version de la branche 1
   - Version de la branche 2
   - Les 2 versions
   - Une résultante
2. **Commiter la résolution du conflit**

Gestion des conflits

```
$ cat merge.txt
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> new_branch_to_merge_later
```

Source image Atlassian

Démonstration

Glodie Tshimini : contact@tshimini.fr

Résolution d’un conflit

Exercice

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

Exercice 5

- Depuis GitHub , modifiez le contenu de *burger.md depuis et faites un commit* .

- En local, faites une modification sur la même ou les mêmes lignes puis faites un commit.

- En local, sur la même branche, récupérez les modifications effectuées en ligne.

- Résolvez les conflits et commitez.

III.

Manipulation de l’historique

Revenir en arrière avec git checkout

Sans modifier l'historique (en tant que simple observateur)

État du dépôt tel qu'il était à partir d'un commit	Fichier tel qu'il était à partir d'un commit	Fichier tel qu'il est au niveau du HEAD (pointeur)
git checkout {id}	***git checkout {id} file.txt***	***git checkout file.txt***

- Pour quitter le mode spectateur, utilisez la commande ***git checkout {name_of_current_branch}***
- Remplacer {id} par l'identifiant de votre commit

Revenir en arrière avec git reset

- Modifie l’historique
- **Avant une publication (push)**
- Après avoir effectué un push, vous ne devez pas utiliser la commande reset pour modifier votre historique, **c’est trop tard.**

Revenir en arrière git reset

	git reset {id} --soft	*git reset {id} --mixed*	*git reset {id} --hard*
Supprime les commits après l'id indiqué	Oui	Oui	Oui
Conserve les modifications effectuées	Oui	Oui	Non
Conserve les fichiers modifiés dans l'index	Oui	Non	Non

Attention avec l'option **--hard**, toutes les modifications survenues après l'*id* du commit indiqué **seront perdues**

Revenir en arrière git revert

- Après avoir effectué un push et sans modifier l’historique
- Crée un commit
- Retirer les modifications introduites par un commit (défaire)

git revert {id}

- Remplacer {id} par l ’identifiant du commit

Démonstration

Glodie Tshimini : contact@tshimini.fr

Reset des commits

Exercice

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

Exercice 6

Reprendre l’exercice 5.

- ❑ Créez une nouvelle branche.
- ❑ Réalisez un reset d’un ou plusieurs *commits*.
- ❑ Regardez l’état de votre dépôt.
- ❑ Envoyez vos modifications sur votre dépôt distant.

Travaux pratiques

source des exercices : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

TP en binome : Attention au piège

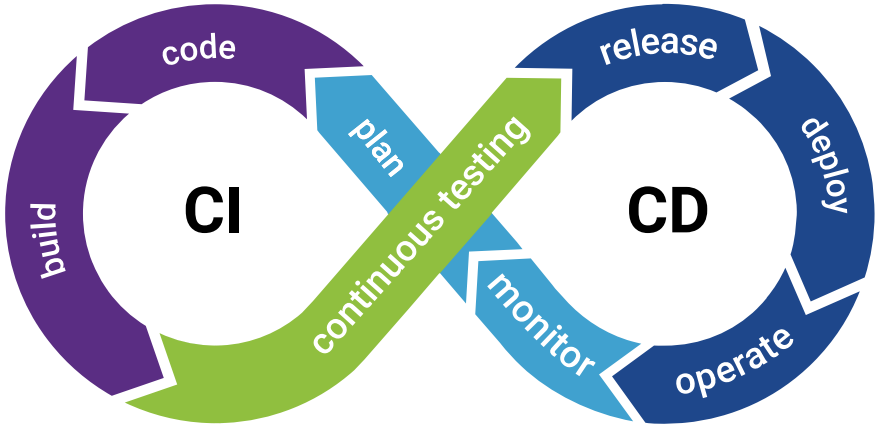
- Un membre du binôme, crée un dépôt distant sur GitHub.
  Invite l'autre développeur à travailler sur ce dépôt.
- En local, chaque développeur travail sur sa branche (2 devs = 2 branches différentes)
  Chacun crée les fichiers *vegan.md* et *vegetarien.md* à la racine du projet et sans vous concerter sur le contenu de vos fichiers, remplissez-les avec les ingrédients de votre choix.
- Envoyez vos modifications en ligne sur votre branche dédiée.
- Récupérez la branche de votre binôme.
- Fusionnez la branche de votre binôme avec la vôtre.
- Résolvez les conflits.
- Envoyez vos modifications en ligne (vos branches ne doivent pas contenir de conflit) sur votre branche.

IV.

CI/CD

Qu’est-ce que le CI/CD

- *Outil* permettant d’effectuer ***l’intégration*** et le ***déploiement*** en ***continu*** de son application.
- Automatisation de la mise en production de vos modifications dans l’environnement de production juste après un ***push.***



Source image synopsys

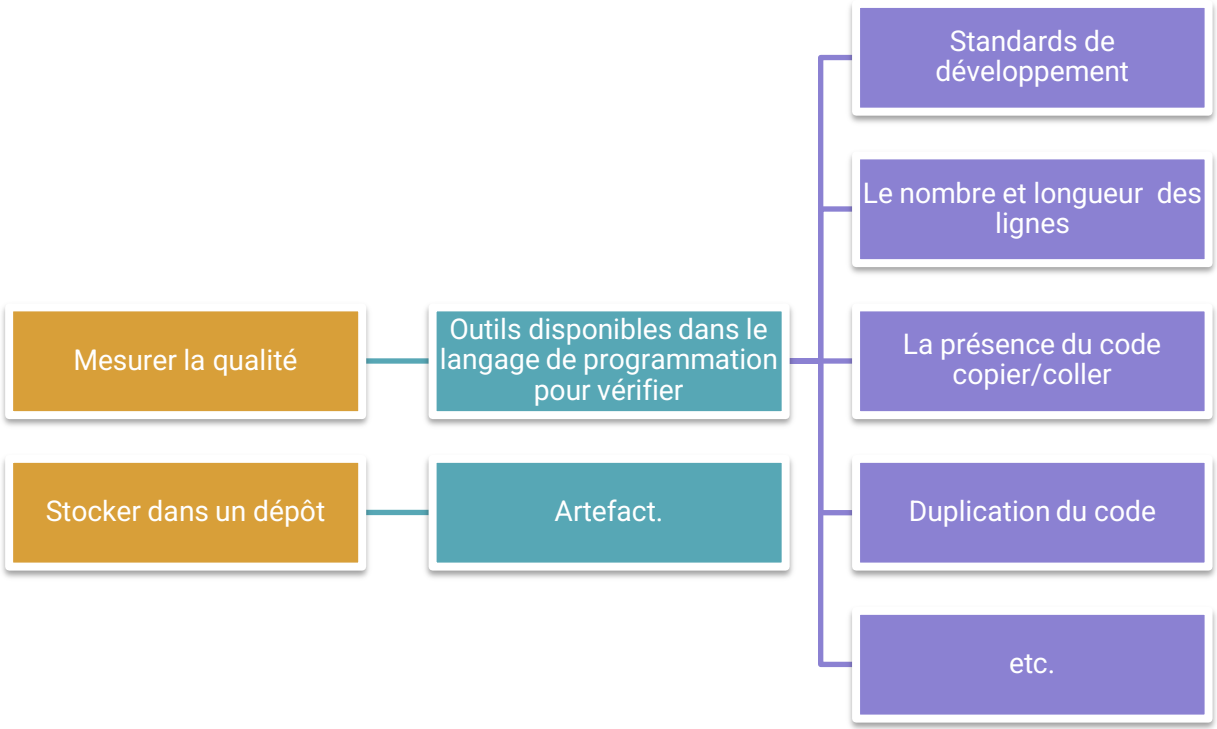
Étapes de l'intégration continue (CI)

1. Planifier les fonctionnalités à coder
2. Coder et compiler les nouvelles fonctionnalités
3. Tester
4. Mesurer la qualité
5. Stocker dans un artefact (fichier(s) d'archive crée(s) par *GitLab*)

Étapes intégration continue (CI)

- Planifier les fonctionnalités à coder
  - Création des issues sur GitLab
  - Ou Création des User Stories sur Jira
- Coder et compiler les nouvelles fonctionnalités
  - Utilisation d'un éditeur de code et de Git
  - Gestion des dépendances de votre projet
- Tester
  - Tests unitaires (minimum)

Étapes intégration continue (CI)

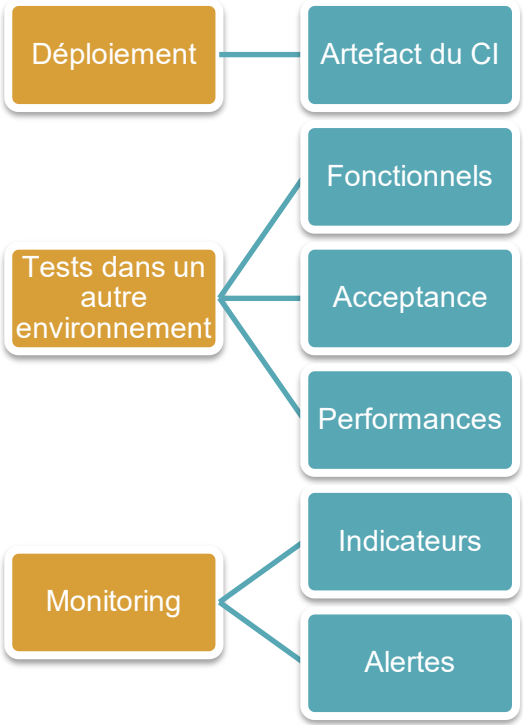


Déploiement continue (CD)

- Déployer le code dans un environnement de production à la même vitesse que l'intégration de façon **automatisé**.

- Généralement le *CI* ne va pas sans le *CD*.

Étapes du déploiement continue (CD)



Fichier .gitlab-ci.yml

- Gère les étapes du *CI*.
- Lance les différentes opérations dans un *pipeline*.
- Documentation GitLab-CI

Example configuration .gitlab-ci.yml

Nodejs.gitlab-ci.yml 1.24 KiB — Open in Web IDE — Lock — Re

```yaml
# You can copy and paste this template into a new `.gitlab-ci.yml` file.
# You should not add this template to an existing `.gitlab-ci.yml` file by using the `include:` keyword.
#
# To contribute improvements to CI/CD templates, please follow the Development guide at:
# https://docs.gitlab.com/ee/development/cicd/templates.html
# This specific template is located at:
# https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Nodejs.gitlab-ci.yml

# Official framework image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/node/tags/
image: node:latest

# Pick zero or more services to be used on all builds.
# Only needed when using a docker container to run your tests in.
# Check out: https://docs.gitlab.com/ee/ci/services/index.html
services:
  - mysql:latest
  - redis:latest
  - postgres:latest

# This folder is cached between builds
# https://docs.gitlab.com/ee/ci/yaml/index.html#cache
cache:
  paths:
    - node_modules/

test_async:
  script:
    - npm install
    - node ./specs/start.js ./specs/async.spec.js

test_db:
  script:
    - npm install
    - node ./specs/start.js ./specs/db-postgres.spec.js

deploy:
  stage: deploy
  script: echo "Define your deployment script!"
  environment: production
```

Source image GitLab

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 7

1-exercices/exercice7.md

V.

Workflow : Git Flow

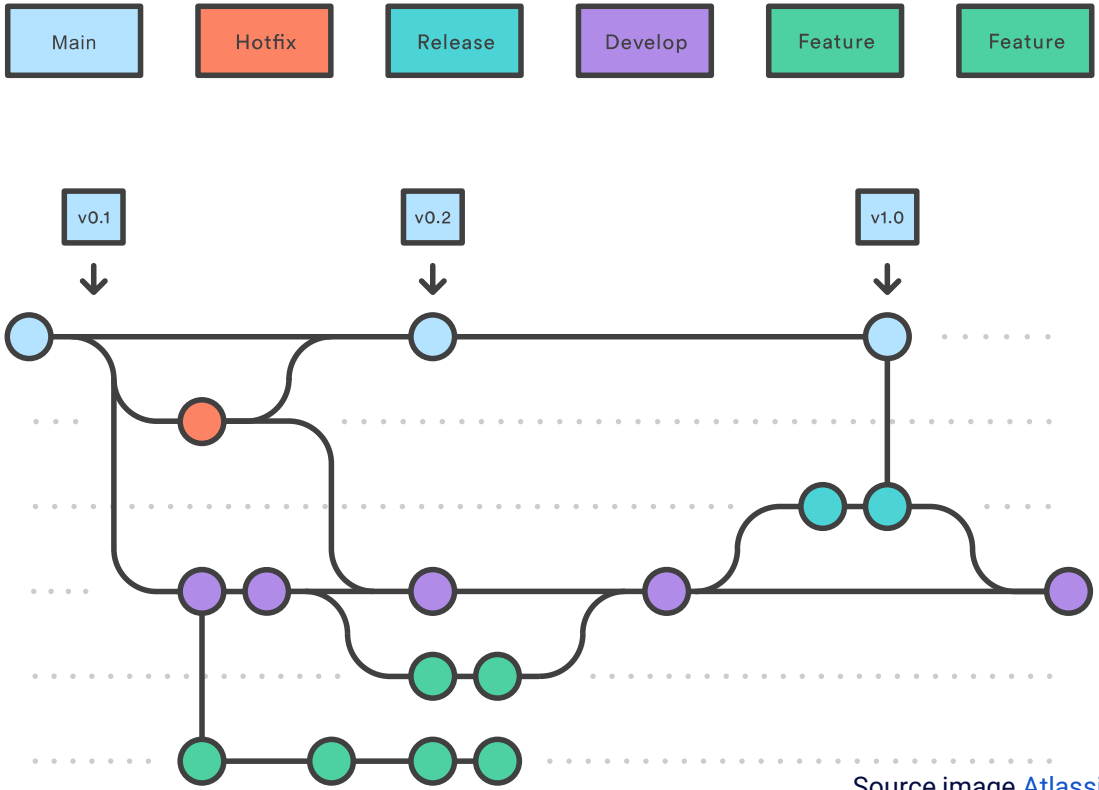
Git Flow : branches principales

- **2 branches principales.**
- La branche ***main*** : fonctionnement habituel.
- La branche ***develop*** : sert de branche d'intégration pour toutes les modifications. Elle est créée à partir de la *main*.

GitFlow : les autres branches

- Pour **chaque besoin** une **branche adapté** qui sera **crée à partir** de la branche ***develop.***
- Fonctionnalités : branche des *features* commence par ***feature/xxx.***
- Livraison : ***release/xxx*** (mergé dans *main* et *develop*).
- Maintenance : ***hotfix/xxx*** (elle se crée à partir de la branche *main*, elle est mergé dans la *main* et *develop*).

GitFlow : resumé



Source image Atlassian

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 8

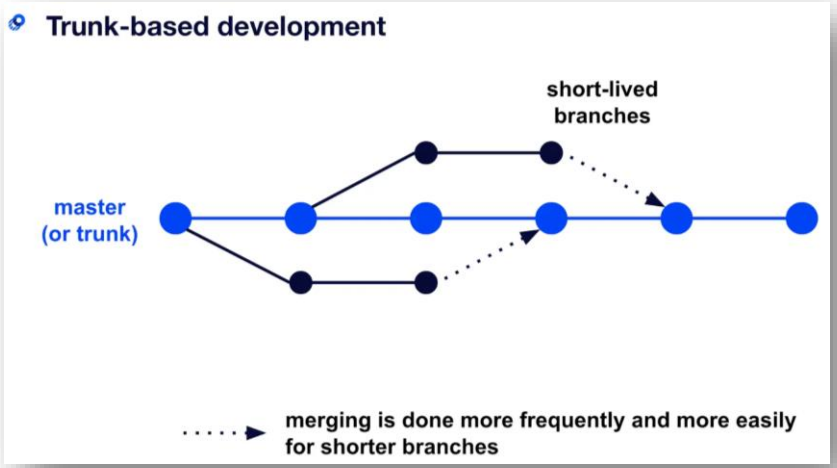
1-exercices/exercice8.md

VI.

Trunk-based Development

Principes

- 1 seule branche principale nommée *master* ou *main* ou *trunk*.
- C'est codé, on commit et on livre (merge immédiatement sur la branche principale).
- Anticiper les problèmes de fusion en amont avant la livraison finale.



Source image Optimizely

Principes

- Requiert des process fiables et solides au niveau de la qualité du code.
- Tests automatisés.
- Cycle de développement très court (tâche).
- Développement agile (*TDD*, *Code Review*).
- Développer en tenant compte de l'existant.

Annexe

Glodie Tshimini : contact@tshimini.fr

Annexe

- Convention de nommage des commits
  - Convention nommage commits Angular
- Pour aller plus vite, commandes Unix
  - Commandes de base console

Terminologie

source : les outils Devops Git, Jenkins, Docker, Ansible

Nicolas Haziza

Glodie Tshimini : contact@tshimini.fr

Terminologie Git

- **Repository (dépôt)** : dossier de travail contenant les fichiers et dossiers d'un projet.

- **Init** : initialiser un dépôt git en local.

- **Commit** : action d'enregistrer vos modifications dans l'historique du projet. Il est accompagné d'un commit message.

- **Status** : visualiser l'état du dépôt local.

Terminologie Git

- **Push** : permet d'envoyer les modifications commitées du depôt local vers le dépôt distant.
- **Pull** : récupérer les commits depuis le dépôt distant (ex: Github) en local.
- **Clone** : récupérer un dépôt distant en local.
- **Checkout** : visualiser ou revenir en arrière, également créer une branche et s'y déplacer.
- **Log** : visualiser l'historique des commits.

Terminologie Git

- **Merge** : action de fusionner deux versions (branches) d'un projet.
- **Fetch** : action de récupérer les modifications du dépôt distant en local sans effectuer directement la fusion.
- **Conflicts** (conflits) : modification(s) effectuée(s) sur les mêmes lignes d'un fichier dans 2 branches distinctes qui doivent être fusionnées.
- **Branch** : version alternative du projet.
- **Diff** : voir les différences entre 2 commits.

Terminologie Git

- **Rebase** : action de rembobiner pour modifier des commits précédents ou de mettre les commits d'une branche à la suite d'une autre branche (les 2 branches doivent avoir un ancêtre (commit) en commun). Pour ce dernier point, c 'est l'équivalent d'une fusion.
- **Tag** : attribuer un numéro de version avec un message à un commit.
- **Mv** : déplacer ou renommer un fichier.
- **Rm** : supprimer un fichier.

FIN.

Avec tous nos remerciements et toutes nos félicitations pour avoir suivi ce module.

Glodie Tshimini : contact@tshimini.fr
Christophe Guéroult : c.gueroult@coderbase.io