Cursus SALESFORCE

M2I Formations 2021

Christian Lisangola

GIT et GitHUB

C’est quoi GIT?

GIT un gestionnaire de versions, c’est-à-dire est un programme qui permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers.

Plus besoin de créer plusieurs dossiers avec de noms tels que : **V1**,**V2** ou encore **update1**, **update2**,etc.

Versioning

Le terme **versioning** fait référence à l'action de contrôler les versions.

Donc, GIT va nous permettre de faire du **versioning**.

Où utiliser GIT

Techniquement, git peut être utilisé partout, c'est-à-dire qu'il peut être utilisé pour gérer des version des fichiers word, excel, text,etc.

Mais, son objectif principale est orienté vers le versioning des fichiers faisant partie du code source d'un projet de développement(software).

Où utiliser GIT

Un gestionnaire de version nous permettra de faire principalement garder en mémoire 3 choses:

- ❏ Chaque modification de chaque fichier
- ❏ La raison de sa modification
- ❏ L’auteur de la modification

Utilisation de GIT

Git peut être utilisé dans de manière individuelle ou dans un environnement en équipe car dans les 2 cas il permet de :

- ❏ Revenir à une version précédente de votre code en cas de problème.
- ❏ Suivre l'évolution de votre code étape par étape.
- ❏ Travailler sur des petits incréments, ce qui facilite le retour en arrière en ayant connaissance de la dernière version stable du code.

En équipe, avec la particularité d'avoir

- ❏ plusieurs développeurs dans un même projet qui écrivent du code sans toucher ou supprimer les modifications apportées par d'autres
- ❏ Possibilité de mettre en commun le travail effectué par plusieurs membres de l'équipe où chacun travail sur des fonctionnalité différentes.

C’est quoi GIT?GitHub,Bitbucket,etc..

- **Git est un gestionnaire de versions**. Vous l'utilisez pour créer un dépôt local et gérer les versions de vos fichiers.

- **GitHub,Bitbucket,GitLab,etc.. sont des services en ligne** qui permettent héberger votre dépôt. Dans ce cas, on parle **de dépôt distant** puisqu’il n’est pas stocké sur votre machine.

  Grace au dépôt distant, vous pouvez collaborer avec d’autres développeurs, et si par exemple votre ordinateurs tombe en panne,pas d’inquiétude, car l’historique est délocalisé, ce qui permet de récupérer son code sans problème.

Github étant le plus utilisé et populaire, c’est ce que nous allons utiliser.

Préparation de l'environnement

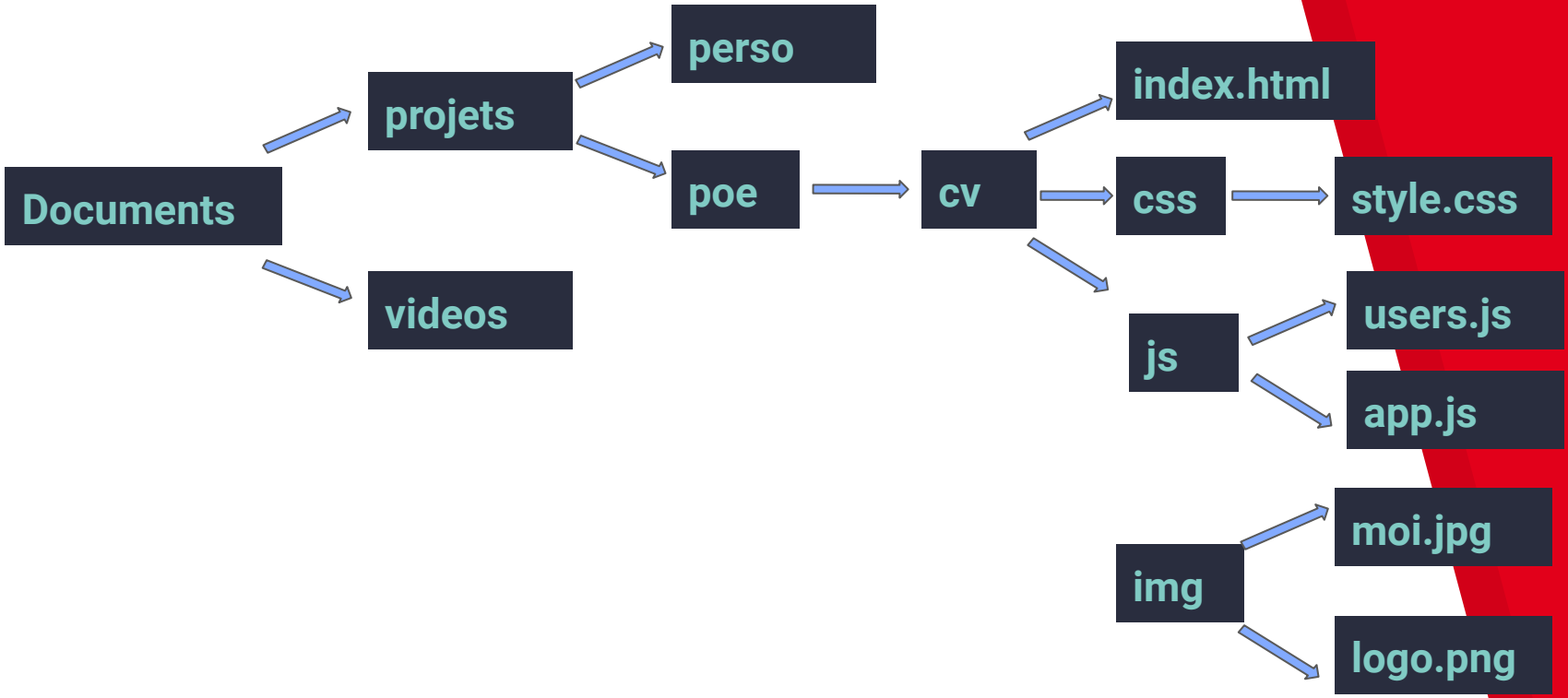
1. Création d'un compte github : https://github.com/
2. Installer vs-code : https://code.visualstudio.com/
3. Télécharger et installer Git en local : https://git-scm.com/
4. Tester le bon fonctionnement de GIT en local
5. Créer son premier dépôt distant

Revoir les base de la ligne de commande

Avec git, vous allez beaucoup manipuler la ligne de commande, donc il est nécessaire de revenir sur les notions de base de la ligne de command.

- **ls** : Elle permet de lister le contenu d'un répertoire
- **mkdir <nom_du_repertoire>** : permet de créer un répertoire ou dossier
- **rm <nom_du_fichier>** : Supprimer un fichier
- **rm -d <nom_du_dossier>** : Supprimer un dossier vide
- **rm -r <nom_du_dossier>** : Effectue un suppression récursive et permet la suppression des dossiers sous dossiers.
- **touch <nom_fichier>** : Permet de créer un fichier
- **cat <nom_ficher>** : Permet de voir ou lire le contenu d'un fichier
- **cd** : Permet de naviguer dans notre file system
- **pwd** : Permet de voir le chemin absolu vers le dossier où l'on se trouve
- **du -h** : Taille occupé par les dossiers(l'option -h c'est pour avoir un format d'affichage facile à comprendre).

Exercice


Documents
projets
perso
poe
videos
cv
index.html
css
style.css
js
users.js
app.js
img
moi.jpg
logo.png

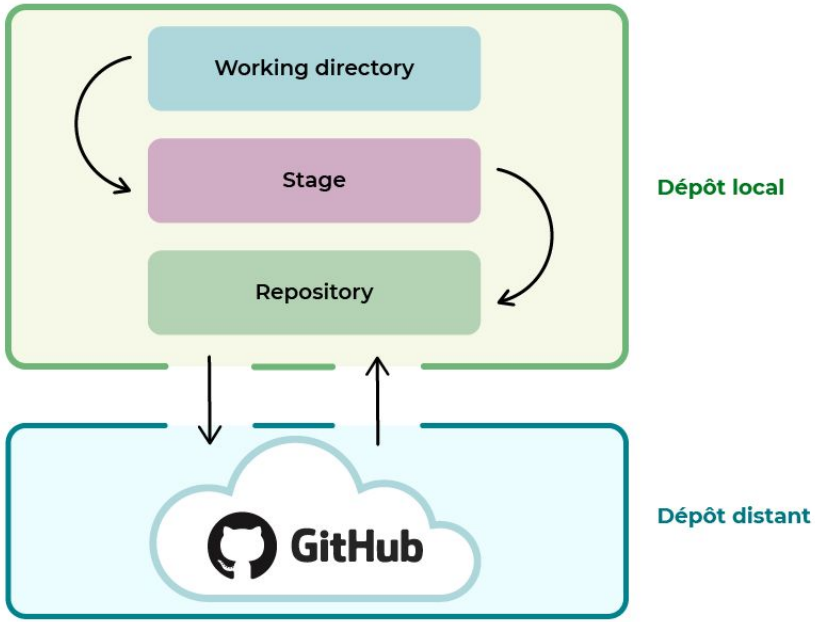
Configurer votre identité avec GIT

git config --global user.name "Christian Lisangola"

git config --global user.email "christian.lisangola@gmail.com"

git config --list : Pour voir toutes vos configurations

Fonctionnement du dépôt local



Working directory

Cette zone correspond au dossier du projet sur votre ordinateur.

Stage ou Index

C’est une zone intermédiaire entre le working directory et le repository local git.

C’est là que se trouve tous les fichiers modifiés que vous souhaitez voir apparaître dans votre prochaine **version de code**.

Le repository

c’est la zone où sont stockées les changements validé, c’est-à-dire, nos versions.

Commandes de base GIT : git init

git init : permet de créer un dépôt git local.

Commandes de base GIT : git add

git add : Ajouter nos fichiers dans le staging area

Utilisation :

git add mon_fichier.ext

git add mon_dossier

git add *.ext : Ajouter les fichier d'une extension spécifique.

git add -A ou git add* . : Ajouter tous les fichier

Annuler les change du Working Area

Ancienne approche

- Sur un seul fichier : git checkout <nom_fichier>
- Sur tous les fichiers : git checkout .

Approche récente:

- Sur un seul fichier : git restore <nom_fichier>
- Sur tous les fichiers : git restore .

Retirer les fichiers du stage

- Sur un seul fichier : git restore --staged <nom_fichier>
- Sur tous les fichiers : git restore --staged .

Commandes de base GIT : git status

git status : Montre l'état des fichier en montrant les changement qui sont en stage et ceux qui ne le sont pas encore.

Commandes de base GIT : git commit

git commit -m”Votre message” : C’est la commande de la création de version ou du versionnement proprement dit.

L’option -m* : Permet de définir un message qui décrit le motif du commit.

l’option -a(--all) : Quand on ajoute cette option,les fichiers sont ajoutés au stage et après faire un commit.

Commandes de base GIT : git log

git log : Cette commande vous permet de voir l'historique de vos changements

--oneline* : permet d'afficher l'historique avec une ligne par commit (plus lisible)

-n <nombre> : permet de sélectionner le nombre de commit à afficher

-p <fichier> : permet de voir l'historique des commits affectant un fichier en particulier

--author <nom_auteur> : permet de voir l'historique par rapport au nom de l'auteur

Commandes de base GIT : git checkout

git checkout <nom_branch> : C’est la commande qui vous permet de naviguer entre les branches.

git checkout <hash_commit> : Voir l’état du code à commit particulier.Tout changement effectué ici n’a aucun impact sur votre historique.

git checkout <hash_commit> <mon_fichier.html> : Permet de transformer le <fichier> tel qu'il était lors du <commit> et l'ajoute au staging.Si vous ne voulez pas revenir à ce code, il suffit de faire **git checkout main <mon_fichier>**.

Commandes de base GIT : git revert

La commande revert vous permet de défaire un commit.C’est-à-dire retirer tous les changement qui avaient été fait.Ceci ne permet donc pas de revenir à l’état du projet à cet instant sans rien perdre dans l’historique.

git revert <nom_commit>

Commandes de base GIT : git reset

Contrairement à la commande revert, la commande reset est très dangereuse, car elle nous permettre de revenir en arrière et donc modifier l'historique.

git reset HEAD <nom_fichier> : Supprimer des modification dans le stage.Si par exemple on a pas encore terminé le travail à faire sur un fichier, on peut le retirer du stage.

git reset : Va simplement retirer tous les fichiers du stage sans supprimer l'historique ou les modifications

Pour les 2 commandes ci-dessus, on peut utiliser une alternative récente de git qui est :

git restore --staged <nom_fichier> : Retirer un fichier du stage.En mettant un point à la place de **<nom_fichier>**, tous les fichiers modifiés sont retirés du stage.

Commandes de base GIT : git reset

git reset --soft <commit>: Revenir au commit tout en ajoutant tous les autres changements d'autres commits de l'historique dans le working Area pour que le travail ne soit pas perdu.

git reset --hard <commit> : Va non seulement supprimer l'historique pour revenir au commit, mais aussi toutes les modification du stage, et du working directory.IL ne va cependant pas toucher au untracked files.

Si vous avez fait un soft reset, vous pourrez revenir en arrière avec :

1. git reflog : Pour voir les traces de l'historique
2. git reset (--soft/--hard) ''HEAD@{1}": Choisir le commit sur lequel on souhaite de

Commandes de base GIT : git branch

git branch <nom_branch> : C'est la commande qui vous permet de créer une branche local.

Sans option **<nom_branch>** cette commande affiche les branches.

git checkout <nom_branch> : Se positionner sur une branch

git checkout -b <nom_branch> : Créer une branch et se placer dessus

git merge <nom_branch> : Fusionner le code avec celui d'une branch enfant

Commandes de base GIT : git rebase

Quand la branch parent a évolué et que vous souhaitez mettre à jour la branch enfant, vous pouvez faire un rebase:

git rebase <branch_parent>

Commandes de base GIT : git stash

git stash : stocke (ou stashe) temporairement les changements apportés à votre copie de travail pour que vous puissiez effectuer d'autres tâches, puis revenir et les réappliquer par la suite.Il peut aussi être utile si par erreur, vous aviez commencé à écrire du code dans une branche autre que celle qui est prévue pour le code.

Commandes de base GIT : git push

git push : C’est la commande qui vous permet d’envoyer vos changements dans le repo distant.

Exercice 1

Créer un répertoire sandwich.

Créer un fichier à l'intérieur 'burger.txt' qui contient la liste des ingrédients d'un burger (un par ligne), par exemple :

Steak

Salade

Tomate

Cornichon

Fromage

Exercice 2

- Vérifier l’état de votre dépôt local : git status
- Préparer burget.txt pour le commit : git add
- Commitez les modifications : git commit -m “message de commit”
- Vérifier l’état du dépôt local
- Vérifier la liste des changements prêt à l’envoi : git log

Exercice 3

- Créez quelques autres sandwichs (hot_dog.txt, jambon_beurre.txt, etc.), modifiez également le contenu de burger.txt. **Attention** : chaque modification doit être placée dans un commit dédié. Effectuez au moins 5 modifications en reprenant les étapes de l'exercice 2.

- Testez git log

Exercice 4

- Réalisez un push : git push
- Allez constater la mise à jour sur le repository

Travailler sur un projet avec plusieurs collaboratifs

- Créer un projet privé sur Github
- Inviter d'autres personnes en leur donnant des droits
- Cloner le projet
- Commencer à collaborer

Exercice 5

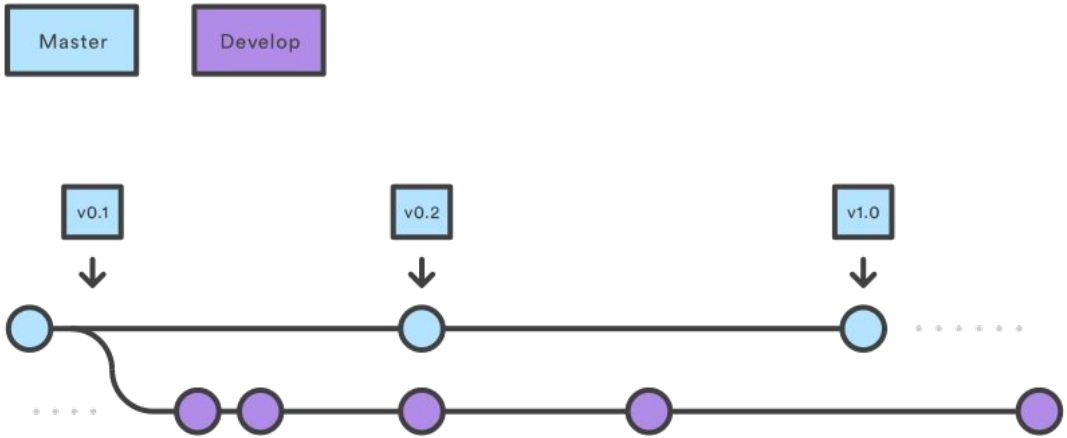
- ❏ Réalisez une modification dans une de vos recettes. **Attention** ne commitez pas
- ❏ Vérifier les modification via un git status puis git diff
- ❏ Réalisez un git reset <nom_du_fichier>
- ❏ Vérifier les modification via un git status puis git diff
- ❏ Réalisez un git checkout <nom_du_fichier>
- ❏ Réalisez un git status

Git : Le Gitflow

- Le Gitflow est un workflow de travail. Il décrit les processus d'utilisation de git dans une équipe (créé par Vincent Driessen).
- Le Gitflow représente un framework particulièrement efficace et adapté aux projets avec un cycle de livraison planifié et régulier.
- Ce worklow n'ajoute aucun concept ni aucune commande en dehors de ce qui est exigé pour le workflow d'organisation des branches

Gitflow : Develop & Master

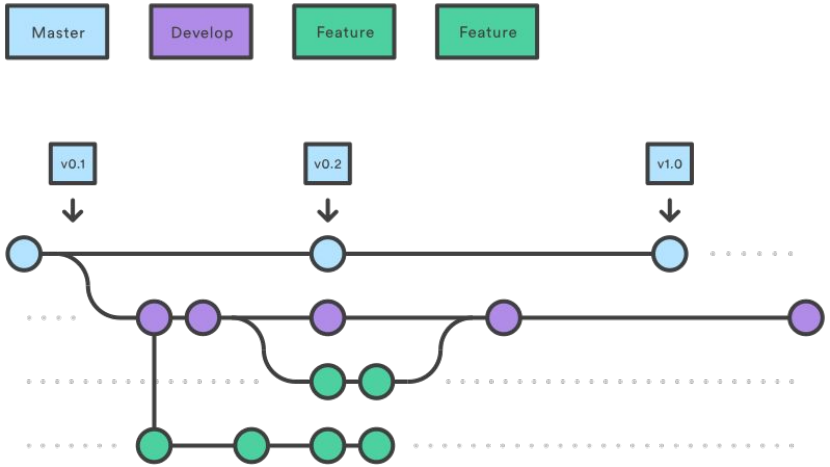
- Le Gitflow utilise deux branches principales au lieu d'une seule.
- La branche master stocke l'historique officiel des versions
- La branche develop sert de branche d'intégration pour les fonctionnalités



- Reprendre le projet des exercices précédent et créer une branche develop depuis master et pusher

Gitflow : Branche de fonctionnalité

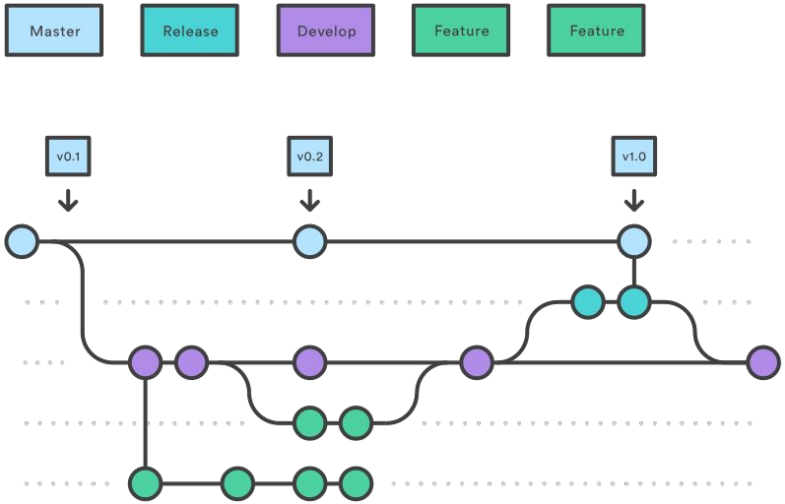
- Le Gitflow utilise une branche par fonctionnalité à développer
- Ces branches sont créées à partir de la branche develop
- Ces branches seront ensuite pushées en vue d'une intégration sur develop



- Créer une branche feature/xxx sur le projet, ajouter un sandwich et pushez sur le repository

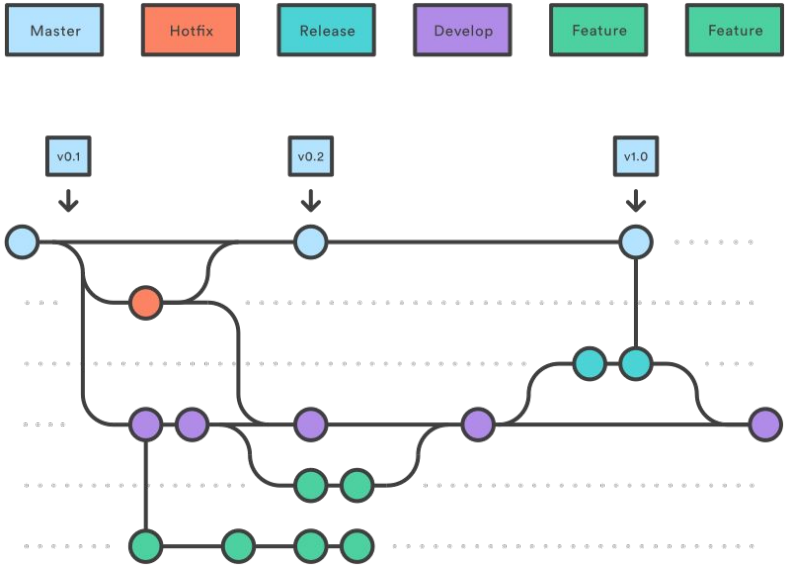
Gitflow : Branche de livraison

- Lorsque vous souhaitez préparer une livraison d'une version, vous allez créer une branche de livraison, souvent nommée release, depuis develop
- La création de cette branche débute un cycle de livraison :
  - Aucune fonctionnalité supplémentaire de pourra être ajoutée à cette branche
  - Dédiée à la corrections de bugs, documentation
- Une fois la branche prête elle sera mergée dans master et taguée d'un numéro de version



Gitflow : Branche de hotfix

- Les branches de maintenances (hotfix) sont utilisées pour appliqué
- Très similaire aux branches de release mais se base sur master (c'est la seule branche qui doit être créée depuis master après develop)
- Une fois la branche prête elle sera mergée dans master et taguée d'un numéro de version (correctif)



Gitflow : Le résumé

- Une branche de développement (develop) est créée à partir de master.
- Une branche de version (release) est créée à partir de develop.
- Les branches de fonctionnalité (feature) sont créées à partir de develop.
- Lorsqu'une branche de fonctionnalité (feature) est terminée, elle est mergée dans la branche de développement (develop).
- Lorsque la branche de version (release) est terminée, elle est mergée dans les branches de développement (develop) et principale (master).
- Si un problème est identifié dans la branche principale (master), une branche de maintenance (hotfix) est créée à partir de master.
- Une fois la branche de maintenance (hotfix) terminée, elle est mergée dans les branches de développement (develop) et principale (master).