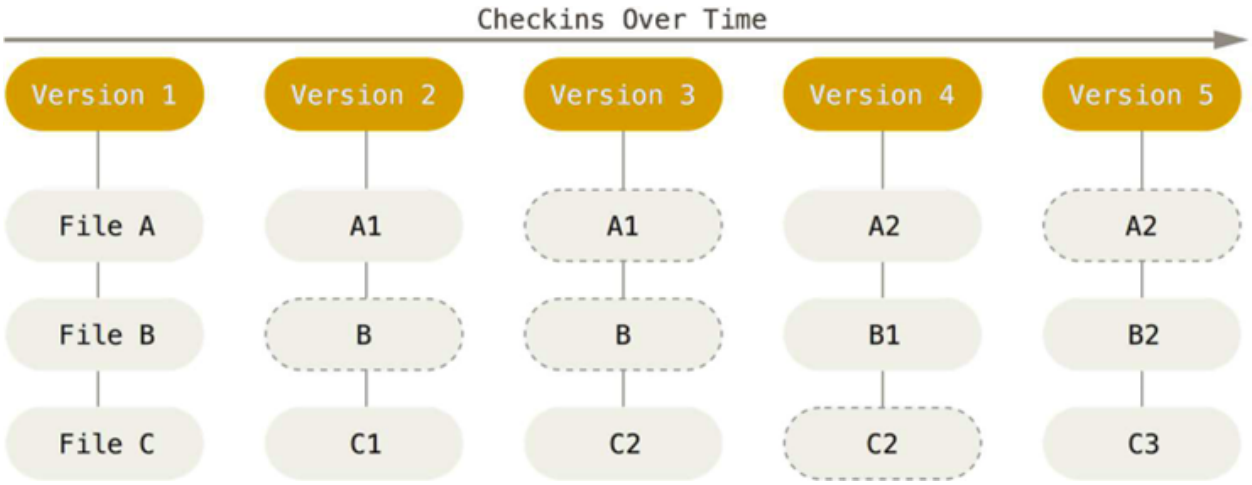
Présentation GIT

Git est né avec une dose de destruction créative et de controverse houleuse.

- Créé par ***Linus Trovalds***, le créateur de Linux en 2005.
- Nouveaux objectifs :
    - Vitesse
    - Conception simple
    - Support pour les développements non linéaires
    - Complètement distribué
    - Capacité à gérer efficacement des projets d'envergure tels que le noyau Linux

Avec Git, à chaque fois que vous validez ou enregistrez l'état du projet, il prend un instantané du contenu de votre espace de travail.



Presque toutes les opérations sont locales :

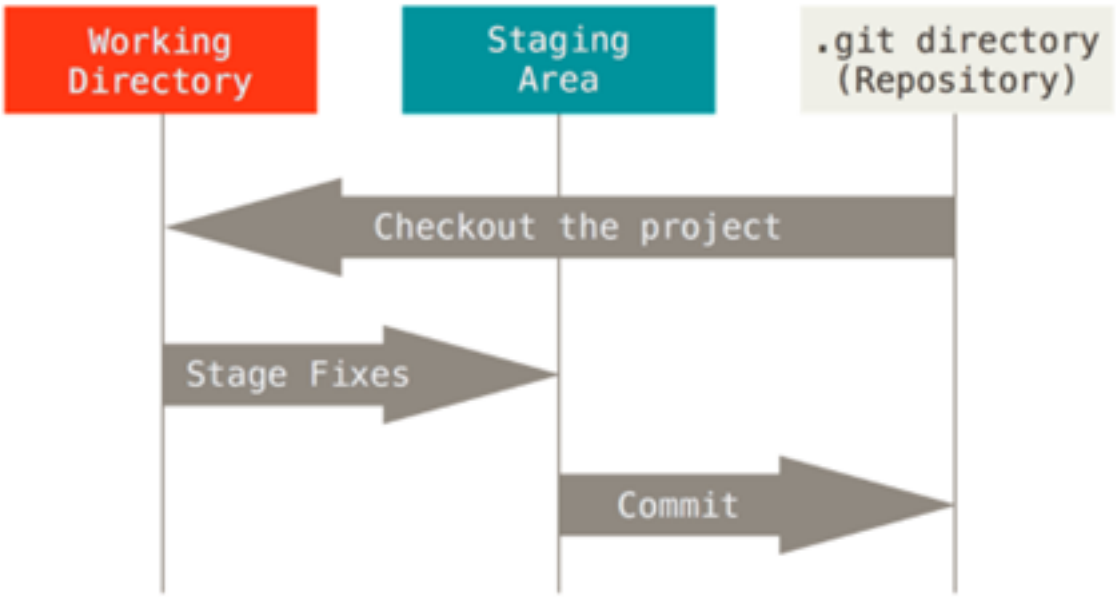
- La plupart des opérations de Git se font en local.
- Même pour chercher dans l'historique du projet.
- L'intégralité de la base de données est en locale.

Dans Git, tout est vérifié par une somme de contrôle avant d'être stocké et par la suite cette somme de contrôle, signature unique, sert de référence.

- Impossible de perdre un fichier dans Git sans que Git s'en aperçoive.
- Les sommes de contrôle sont en SHA-1.

Un fichier versionné sous Git peut-être sous 3 états :

- Validé, le fichier est sauvegardé en base.
- Modifié, le fichier est modifié, la modification n'est pas en base.
- Indexé, le fichier est modifié, et est prêt à être envoyé en base.



Git contient un outil '**git config'** pour modifier les paramètres de configuration

Git a 3 niveaux de configuration :

- Global au système : /etc/gitconfig, modifiable avec 'git config -- system'
- Global à l'utilisateur : ~/.gitconfig, modifiable avec 'git config --global'
- Global au projet : .git/config, modifiable avec 'git config'

Chaque niveau surcharge le précédent

Atelier GIT

Partie 1 : Les bases de Git

// Démarrer un dépôt Git et configurer l'identité

1. Créer f1.txt et src/f2.css
2. git init (observer le dossier caché **.git.** Un dossier .git apparait, l'initialisation est faite, mais aucun fichier n'est versionné !)
3. git config --global user.name "Sonia BEN AISSA"
4. git config --global user.email "benissasonia@gmail.com"

// Comment indiquer à git que le fichier déjà suivi a été modifié et qu'on veut l'ajouter à l'index

5. git add f1.txt
6. git status (Remarque : il y a un dossier n'est pas tracké)
7. git add **src** (Lorsqu'on ajoute un dossier, tous ses fichiers et sous répertoires sont ajoutés avec. Nous pouvons faire git add * (pour ajouter tous les fichiers et dossiers)
8. git status (Tous les fichiers et les dossiers sont traqués)
9. Modifier **f1.txt**
10. git status (git détecte une nouvelle modification). Il faut ajouter cette modification.
11. git add **f1.txt**
12. git status (Pour revérifier de nouveau)
13. git commit (Mettre le message : Initialiser le projet + Modification initiale)
14. Créer un fichier***. gitignore*** et saisir : *.exe
15. git status (La détection de ce nouveau fichier)
16. git add .gitignore
17. Créer un fichier test.exe (touch test.exe ou tout simplement copier-coller un exe)

18. git status (Git ignore cette test.exe parce que nous lui avons indiquer d'ignorer ce fichier dans le fichier de conf .gitignore, d'où le rôle de ce fichier)

19. git commit -m "Modification de f1.txt" (option -m pour saisir le message en ligne)

20. Créer un f3.txt

21. git add f3.txt

22. Modifer f3.txt

23. git commit -a -m "Mofication de f3.txt " (l'option -a joue le rôle de git add)

24. Modifier le fichier f1.txt et f3.txt

25. git status (détection de la modification de f1.txt et f3.txt)

26. git diff (pour visualiser la modification avant commit)

27. git commit -a -m "modification f1.txt f3.txt"

28. Modfier f2.css

29. git commit -a -v (Mettre le message « Update f2.css ». Il faut remarquer que nous pouvons observer les modifications en écrivant le message)// *Comment supprimer ou déplacer (renommer) des fichiers.*

30. Supprimer f3.txt

31. git status (le message de suppression est toujours là !! il faut éviter la suppression de cette façon)

32. git rm f3.txt

33. git status

34. git commit -m "Suppression f3.txt"

35. git mv f1.txt file1.txt (pour renommer un fichier)

    // Comment afficher l'historique de notre projet

36. git log (afficher tous les commits)

37. git log -p (afficher tous les commits en détails)

38. git log -2 (afficher les DEUX derniers commits)

39. git log -p -2 (afficher les DEUX derniers commits en détails)

40. git log –pretty=full (Tester avec fuller, short et oneline

// Jouer avec le formatage

41. git log --pretty=format:'%h %an %ar %s'

// Comment modifier un commit.

42. Modifier le fichier file1.txt
43. Ajouter file2.txt
44. git add file1.txt
45. git commit -m "modification f1.txt " (Nous avons oublié d'indexer file2)
46. git add file2.txt
47. git commit –amend (Cette commande va reprendre l'index du commit précédent. Il est possible de modifier uniquement le message du commit.)
48. git status
49. git log -1 -p (nous remarquons le dernier commit a été mis à jours)

// Comment désindexer un fichier déjà indexé.

50. Créer deux fichier file3.txt et file4.txt
51. git add *
52. git status (tous les fichiers sont indexés, par contre je veux désindexer file4.txt)
53. git reset HEAD file4.txt
54. git status (Nous remarquons le fichier file4.txt est désindexés)

// Comment réinitialiser un fichier modifié

55. Modifier le fichier file1.txt
56. git add file1.txt
57. git status
58. git diff
59. git checkout -- file1.txt (Pour revenir à l'ancien commit du fichier)

Partie 2 : Travailler avec des dépôts distants

Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants.

- Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise.

- Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture.

Nous allons voir comment :

- Ajouter des dépôts distants.

- Effacer des dépôts distants.

Pour visualiser les serveurs distants que vous avez enregistré, vous pouvez lancer la commande :

1. **git remote**.

- Si vous avez cloné notre projet, vous devez avoir au moins un dépôt distant : **origin**.

- Vous pouvez aussi spécifier '-v', qui vous montre l'URL que Git a stocké pour chaque nom court.

Pour ajouter des dépôts distants :

2. git remote add **origin** https://xxxxx.xxxxx.xxxxxxx (Nous pouvons appeler **origin** ou autre chose. Mais il est recommandé d'utiliser ce nom).

3. git remote rm origin (Pour supprimer un dépôt)

4. git remote rename origin myrepos (Pour renommer un dépôt)

Pour explorer le dépôt distant :

5. git fetch origin

   //Envoi et récupération des fichiers

   Pour envoyer le projet local au serveur distant, il faut saisir cette commande

6. git push -u origin --all
7. Ouvrir un nouveau terminal
8. git clone https://xxxxx.xxxxx.xxxxxxx (Pour cloner le dépôt dans un projet 2)
9. Accéder au teminal du projet 1
10. Modifier le fichier file1.txt
11. git status
12. git diff (pour comprendre la modification)
13. git commit -a -m "update file1.txt" (Valider et ajouter les modifications)
14. git push origin main (Pour envoyer les mises à jour au dépôt à la branche par défaut main)
15. Accéder au teminal du projet 2
16. git fetch origin (Une difference est affichée)
17. git pull origin main (Pour récupérer cette mise à jour)
18. git remote show origin (Pour avoir plus d'informations sur un dépôt distant)

    //Les étiquettes

    À l'instar de la plupart des VCS, Git donne la possibilité d'étiqueter un certain état dans l'historique comme important. Généralement, on utilise cette fonctionnalité pour marquer les versions d'un projet (v1.0.0, v1.0.42, v2.0.1, …)

19. Accéder au terminal du projet 1
20. git tag (Pour lister les étiquettes :)

    Créer des étiquettes annotées est simple avec Git :

21. git tag –a v1.1.0 –m 'Ma version 1.1.0'

    Puis pour afficher en détails votre nouveau tag :

22. git show v1.1.0
23. git push origin –tags

24. Accéder au terminal du projet 2

25. git pull origin –tags (Pour récupérer les tags d'un dépôt)

26. Modifier le file1.txt

27. git commit -a -m "Version MEP"

28. git tag -a v1.1.1 -m "Ma version 1.1.1"

29. git push

30. git push –tag

31. Observer le dépôt distant et explorer les tags.

32. git checkout origin (Pour switcher à la branche origine)

33. Afficher le fichier file1. Vous remarquez que le fichier est mis à jour selon la version "Origine"

34. git checkout v1.1.1 (Vous remarquez que le fichier est mis à jour selon la version 1.1.1)

Grâce à ce mécanisme, nous pouvons naviguer à travers les différentes versions de notre projet

Cahier des charges pour la location d'un vélo

1. Introduction Le présent cahier des charges a pour objectif de définir les besoins fonctionnels et non fonctionnels pour la conception et le développement d'une application de location de vélos.
2. Objectifs

- Offrir aux utilisateurs la possibilité de localiser, réserver et louer un vélo en toute simplicité.
- Assurer un processus de paiement sécurisé.
- Proposer une expérience utilisateur fluide et intuitive.

3. Cibles

- Les citadins qui cherchent un moyen de transport écologique.
- Les touristes qui veulent découvrir la ville à vélo.

Backlog

EPIC 1: Localisation des vélos

Nom	Description	Priorité	Classification	Acceptance Criteria
Feature 1.1: Afficher une carte	Intégration d'une carte pour visualiser la position des vélos disponibles.	Haute	Feature	La carte se charge sans erreur. Les points représentant les vélos sont visibles.

User Story 1.1.1: Charger la carte	En tant qu'utilisateur, je souhaite voir une carte de la ville pour localiser les vélos.	Haute	User Story	La carte s'affiche correctement et montre les zones de la ville.
Task 1.1.1.1: Intégrer l'API de la carte	Intégrer une API comme Google Maps ou OpenStreetMap.		Task	La carte se charge avec les données de l'API choisie.
Task 1.1.1.2: Afficher les points des vélos	Placer des points sur la carte pour chaque vélo disponible.		Task	Les vélos disponibles sont visibles sur la carte.

EPIC 2: Réservation de vélos

Nom	Description	Priorité	Classification	Acceptance Criteria
Feature 2.1: Processus de réservation	Permettre à l'utilisateur de réserver un vélo.	Haute	Feature	L'utilisateur peut sélectionner, réserver et obtenir la confirmation de sa réservation.

User Story 2.1.1: Sélectionner un vélo	En tant qu'utilisateur, je souhaite sélectionner un vélo sur la carte pour le réserver.	Haute	User Story	Je peux cliquer sur un point de vélo et voir ses détails.
Task 2.1.1.1: Interface de sélection	Créer une interface permettant la sélection d'un vélo.		Task	En cliquant sur un vélo, ses détails s'affichent.
Task 2.1.1.2: Backend de réservation	Créer une fonction backend pour gérer la réservation.		Task	La réservation est sauvegardée dans la base de données.

EPIC 3: Paiement sécurisé

Nom	Description	Priorité	Classification	Acceptance Criteria
Feature 3.1: Intégration d'une solution de paiement	Permettre à l'utilisateur de payer sa réservation en ligne.	Haute	Feature	L'utilisateur peut payer et recevoir une confirmation de paiement.

User Story 3.1.1: Entrer les détails de paiement	En tant qu'utilisateur, je souhaite entrer mes coordonnées bancaires pour payer ma réservation.	Haute	User Story	Je peux entrer mes coordonnées et effectuer le paiement.
Task 3.1.1.1: Choisir une plateforme de paiement	Sélectionner une plateforme de paiement comme Stripe ou PayPal.		Task	La plateforme de paiement est intégrée et fonctionnelle.
Task 3.1.1.2: Interface de paiement	Créer une interface pour l'entrée des détails de paiement.		Task	L'interface est intuitive et sécurisée.

Présentation de DevOps

https://learn.microsoft.com/fr-fr/training/modules/introduction-to-devops/

Présentation d'Azure DevOps

https://learn.microsoft.com/fr-fr/training/paths/evolve-your-devops-practices/

Support de cours

https://learn.microsoft.com/fr-fr/azure/devops/get-started/?view=azure-devops