**1. Création d'un dépôt Git local**

La commande git init est utilisée pour initialiser un nouveau dépôt Git dans un répertoire.

Cela transforme ce répertoire en dépôt Git,

ce qui permet à Git de commencer à suivre les fichiers et leurs modifications dans ce projet.

2. Vérifier l'état du dépôt

Montre comment la commande git status affiche l'état actuel du dépôt, notamment les fichiers non suivis ou modifiés.

* **Exercice** : vérifier l'état du dépôt après avoir créé exercicesGit.html.

git status

**3. Ajouter des fichiers au suivi Git**

* **git add** permet d'**ajouter des fichiers ou des modifications** dans la **zone de staging** (ou index) de Git **Exercice** : Ajouter le fichier créé au dépôt et le préparer pour un commit.

git add exercicesGit.html

git reset exercicesGit.html # Enlever du staging

git add . # Ajouter tous les fichiers

**4. Effectuer un commit**

**git commit -m** permet d'**enregistrer les modifications** qui ont été ajoutées à la zone de staging (via git add) dans le dépôt local, en y associant un message descriptif.

**5. Ajouter d'autres fichiers et faire des commits**

* **Exercice** : Créer deux nouveaux fichiers et effectuer des commits.
* touch deuxièmeFichier.html

git add deuxièmeFichier.html

git commit -m "Ajout du deuxièmeFichier.html"

touch troisiemeFichier.txt

git add troisiemeFichier.txt

git commit -m "Ajout de troisiemeFichier.txt"

**6. Historique des commits**

**git log** affiche l'**historique des commits** dans un dépôt Git. Elle montre une liste de tous les commits effectués, avec des informations comme l'identifiant du commit (SHA-1), l'auteur, la date et le message de commit

**7. Gestion des branches**

* **git branch** permet de gérer les **branches** dans Git. Une branche est comme une version parallèle de ton projet, ce qui permet de développer des fonctionnalités ou de corriger des bugs sans affecter la branche principale (main ou master).
* **git branch** <nom-de-branche> créer une nouvelle branche
* Pour basculer vers cette nouvelle branche, tu dois utiliser **git checkout** ou **git switch**

**git branch** affiche toutes les branches locales. La branche actuelle est marquée par un astérisque (\*).

* git branch cours-git/premiere-branche

git checkout cours-git/premiere-branche # ou git switch

**8. Fusion des branches**

Présente la commande git merge pour fusionner les modifications d'une branche dans une autre.

* **Exercice** : Basculer sur la branche principale et fusionner les changements de la nouvelle branche.

git checkout master

git merge cours-git/premiere-branche

**9. Supprimer et renommer des fichiers**

Utilise git rm pour supprimer des fichiers et git mv pour les renommer, en expliquant que ces commandes enregistrent automatiquement les changements pour un commit.

* **Exercice** : Supprimer un fichier, en renommer un autre, puis faire un commit.

git rm quatre.pdf

git commit -m "Suppression de quatre.pdf"

git mv exercices.html exercicesVide.html

git commit -m "Renommage de exercices.html en exercicesVide.html"

**10. Dépôts distants**

Explique comment lier un dépôt local à un dépôt distant avec git remote add et comment pousser les modifications avec git push.

* **Exercice** : Ajouter un dépôt distant et pousser les commits.

git remote add origin https://github.com/saloumCDA03/cours-git.git

git push -u origin master

**11. Mise à jour depuis un dépôt distant**

Montre comment synchroniser les modifications du dépôt distant avec git pull.

* **Exercice** : Tirer les modifications d'une branche distante.

git pull origin master

**12. Fichier .gitignore**

Le fichier **.gitignore** permet de spécifier les fichiers ou répertoires que Git doit ignorer lors du suivi des changements (c’est-à-dire, ne pas inclure dans l’indexation ni dans les commits). Cela permet d'exclure les fichiers temporaires, sensibles, ou spécifiques à un environnement de développement, afin qu'ils ne soient pas partagés dans le dépôt.

Pour ignorer un fichier spécifique, inscris simplement son nom.

Pour ignorer un répertoire et tout son contenu, utilise un / à la fin de son nom

Pour ignorer tous les fichiers d'une certaine extension, utilise un \* pour faire correspondre tous les fichiers de ce type : \*.log

Les lignes qui commencent par un # sont des commentaires et ne sont pas interprétées

**Envoyer son dépôt local sur GitHub**

Pour envoyer ton dépôt local sur GitHub pour la première fois, tu dois lier ton dépôt local à un dépôt distant sur GitHub et pousser tes fichiers. Voici les étapes détaillées pour le faire :

**1. Créer un dépôt sur GitHub**

1. Connecte-toi à [GitHub](https://github.com/).
2. Clique sur le bouton **New** (en haut à droite) ou accède directement à la page de création d'un nouveau dépôt en suivant [ce lien](https://github.com/new).
3. Donne un nom à ton dépôt (par exemple, cours-git), ajoute une description si tu le souhaites.
4. Choisis si ton dépôt doit être **public** ou **privé**.
5. Ne coche pas l'option pour initialiser le dépôt avec un fichier README (si tu le fais, il peut y avoir des conflits lorsque tu tenteras d'envoyer ton dépôt local).
6. Clique sur **Create repository**.

**Lier ton dépôt local à GitHub**

Assure-toi que tu as déjà initialisé un dépôt Git localement (si ce n’est pas déjà fait, utilise git init dans ton dossier local).

Dans ton terminal, dans le dossier de ton projet local, lie ton dépôt local à ton dépôt GitHub avec la commande suivante :

git remote add origin https://github.com/ nomd'utilisateur /nom de ton dépôt.

**Pousser le dépôt local sur GitHub**

Pour envoyer ton dépôt local vers GitHub, utilise la commande suivante pour pousser la branche principale (habituellement master ou main) vers le dépôt distant :

git push -u origin master

**-u** : Établit une relation de suivi entre ta branche locale et la branche distante, ce qui te permet d'utiliser simplement git push à l'avenir sans spécifier origin master.

Si la branche par défaut s'appelle **main** au lieu de **master**, remplace master par main dans la commande.

**Authentification avec GitHub**

GitHub demandera une authentification.

Tu devras utiliser un **Personal Access Token (PAT)** à la place du mot de passe.

voici comment en créer un :

1. Va dans [tes paramètres GitHub](https://github.com/settings/tokens).
2. Sous **Developer settings**, sélectionne **Personal access tokens** et clique sur **Generate new token**.
3. Choisis les permissions nécessaires (par exemple, accès aux dépôts).
4. Copie le token généré et utilise-le comme mot de passe lorsque Git te demande d'authentifier l'envoi vers GitHub.

**Cloner un dépôt distant sur ta machine locale**

Un **dépôt distant** est une version d'un projet Git hébergé sur un serveur distant, comme GitHub, GitLab, ou Bitbucket.

**a. Trouver l'URL du dépôt distant**

1. Connecte-toi à la plateforme où se trouve le dépôt distant (par exemple, GitHub).
2. Va dans le dépôt que tu souhaites cloner.
3. Clique sur le bouton **Code** (ou Clone) puis copie l'URL. Il peut s'agir d'une URL **HTTPS** ou **SSH**.
   * **Exemple d'URL HTTPS** : https://github.com/username/repository.git
   * **Exemple d'URL SSH** : git@github.com:username/repository.git

**b. Cloner le dépôt**

Ouvre ton terminal ou ton invite de commande.

Navigue vers le répertoire où tu souhaites cloner le dépôt en utilisant la commande cd :

Utilise la commande git clone suivie de l'URL du dépôt distant pour cloner le dépôt sur ta machine locale :

git clone https://github.com/username/repository.git

Une fois le clonage terminé, va dans le répertoire du projet cloné, Tu devrais maintenant avoir une copie complète du dépôt distant sur ta machine, y compris toutes les branches, les fichiers et l'historique des commits.

**Pour** **Vérifier les branches disponibles** :

git branch -a : affiche la liste des branches locales et distantes disponibles dans ton dépôt Git.

**Récupérer un dépôt distant appartenant à un autre utilisateur**

Si tu veux **récupérer un dépôt distant appartenant à un autre utilisateur** et l'ajouter à ton propre compte GitHub, **faire un fork** du dépôt pour l'ajouter à ton propre compte GitHub.

**Forker le dépôt distant pour travailler sur ton propre compte GitHub.**

Accède au dépôt GitHub que tu veux forker.

En haut à droite de la page du dépôt, tu verras un bouton Fork. Clique dessus.

Cela créera une copie du dépôt sur ton propre compte GitHub.

**La collaboration avec Git**

La collaboration avec Git est essentielle pour travailler efficacement en équipe sur un même projet.

**Créer un dépôt central** : Le projet doit avoir un dépôt GitHub central, géré par un mainteneur ou une équipe. Ce dépôt sera le point de référence pour tous les contributeurs.

**Ajouter des collaborateurs** : Si les membres de l'équipe ont un accès direct, le propriétaire peut les ajouter en tant que collaborateurs avec des droits d'écriture sur le dépôt.

Va sur la page principale du dépôt où tu veux ajouter des collaborateurs. Clique sur l'onglet **"Settings",** **Sous la section "Collaborators", clique sur "Add people".**

**Cloner le dépôt sur ta machine locale.**

Le collaborateur peut cloner le dépôt pour y travailler.

**Travailler avec des branches** : Chaque membre de l'équipe doit créer une branche pour chaque nouvelle fonctionnalité ou correction de bug. Cela permet de séparer le travail et d'éviter les conflits de code.

**Récupérer les dernières modifications du dépôt principal**

Avant de commencer à travailler, assure-toi que ton dépôt local est à jour par rapport au dépôt principal : git pull origin main

Une fois que tu as terminé ton travail sur une branche, crée une Pull Request pour proposer tes modifications. Les autres membres de l'équipe ou les mainteneurs pourront réviser le code avant de le fusionner dans la branche principale. Sur GitHub, clique sur "Pull Requests" puis sur "New Pull Request", choisis ta branche et demande à fusionner dans main ou master.