ETAPE 2

1. Pour une configuration unique cad pour un projet unique, on tape dans le terminal la commande suivante « git config user.name ’’ Nom d’Utilisateur’’ » et pour une configuration globale pour tous les projets « git config –global user.name ‘’nom d’utilisateur’’ »
2. Cela est important car cela permet **d’identifier les contributions (**chaque commis sera associé à mon nom d’utilisateur et mon adresse e-mail. Cela permet de savoir qui a fait quoi dans le projet ; suivre **Historique et suivi des modifications** (un bon suivi des versions permet de revenir sur l’historique du projet ; une Collaboration efficace

ETAPE 3 :

1. Comment faire ?

* Ouvrir le terminal
* Naviguer vers le répertoire concerné
* Initialiser Git avec la commande « git init » (Cette commande crée un sous repertoire « .git »)
* Vérifier l’initialisation grace a la commande « git status »
* OPTIONNEL
* Ajouter des élément à la zone de staging avec la commande « Git add . »
* Effectuer un premier commit pour sauvegarder l’état initial du projet « Initial commit »4

1. Cette commande crée un sous-répertoire « **.git »** dans ton dossier. Ce sous-répertoire contient tous les fichiers et métadonnées nécessaires à Git pour suivre les versions de ton projet.

ETAPE 4

/////

* Ajouter des élément à la zone de staging avec la commande « Git add . »
* Effectuer un premier commit pour sauvegarder l’état initial du projet « Initial commit »
* la commande « git status »

ETAPE 5 :

* On utilise la commande « git add . » ou « git add -A »
* La commande « git commit -m ‘’message à passer’’ »

ETAPE 6 :

Pour afficher l’historique des commits dans Git, tu peux utiliser la commande suivante :

git log

**Détails de la commande git log :**

* Cette commande affiche une liste des commits précédents, avec des informations telles que :
  + L'ID du commit (un hash unique).
  + Le nom de l’auteur du commit.
  + La date du commit.
  + Le message associé au commit.

**Options utiles pour git log :**

1. **Afficher un historique condensé** : Si tu souhaites un affichage plus condensé avec uniquement les informations essentielles (ID de commit abrégé, auteur et message), tu peux utiliser :

git log --oneline

Cela affiche l'historique des commits sur une seule ligne pour chaque commit, avec un ID abrégé.

1. **Limiter le nombre de commits affichés** : Si tu veux voir un nombre spécifique de commits, par exemple les 5 derniers, tu peux faire :

git log -n 5

Ici, -n 5 affiche les 5 derniers commits.

1. **Voir les différences avec chaque commit** : Si tu veux voir les changements effectués dans chaque commit (les différences dans les fichiers), tu peux utiliser :

git log -p

Cela affiche les modifications détaillées (le "diff") pour chaque commit.

1. **Personnalisation de l'affichage** : Pour un affichage encore plus visuel avec un graphe représentant les branches, tu peux utiliser cette commande :

git log --oneline --graph --all

Cela montre un graphe des branches avec les commits sous forme condensée.

ETAPE 7

* Création de branche : git branch nom de la branche
* Bacsculer de l’une à l’autre : git switch nom branche ou git chekout nom branche
* Les etapes sont les suivantes :
* Basculer sur la branche principale
* Fusionner la branche avec la branche principale (commande : git merge featureX)
* Résolution des conflits (si nécéssaire)
* Vérification

**Résumé des commandes :**

1. Bascule sur la branche principale :

git checkout main

1. Fusionne la branche featureX :

git merge featureX

1. Résous les conflits si nécessaire, puis valide avec :

git add nom\_du\_fichier

git commit

ETAPE 8 :

**1. Annuler les modifications non indexées (non ajoutées à la zone de staging)**

Si tu as modifié un fichier mais que tu ne l'as pas encore ajouté à la zone de staging avec git add, tu peux simplement restaurer le fichier à son état précédent (c'est-à-dire avant les modifications) avec la commande suivante :

git checkout -- nom\_du\_fichier

Ou, dans les versions récentes de Git, il est recommandé d'utiliser :

git restore nom\_du\_fichier

Cette commande annule toutes les modifications non enregistrées dans nom\_du\_fichier et le remet dans l’état où il était lors du dernier commit.

**2. Annuler les modifications dans la zone de staging**

Si tu as déjà ajouté le fichier à la zone de staging avec git add, mais que tu veux annuler ces changements pour revenir à l'état modifié non indexé, utilise cette commande :

git reset nom\_du\_fichier

Cette commande retire le fichier de la zone de staging tout en conservant les modifications locales (les modifications ne seront pas perdues, elles reviendront à l’état non indexé).

**3. Annuler toutes les modifications dans un fichier (même dans la zone de staging)**

Si tu veux annuler toutes les modifications, que le fichier soit dans la zone de staging ou non, pour le remettre tel qu'il était lors du dernier commit, utilise la commande suivante :

git restore --staged nom\_du\_fichier

git restore nom\_du\_fichier

* git restore --staged nom\_du\_fichier retire le fichier de la zone de staging (si nécessaire).
* git restore nom\_du\_fichier restaure le fichier tel qu'il était lors du dernier commit.

**4. Annuler toutes les modifications dans tout le projet**

Si tu veux annuler toutes les modifications sur tous les fichiers de ton projet (non stagés et non commités), utilise :

git restore .

Cela restaure **tous les fichiers modifiés** non indexés à leur état initial.

**5. Annuler un commit déjà effectué (avancé)**

Si tu veux annuler des modifications qui ont déjà été commités, tu peux utiliser :

git revert <hash\_du\_commit>

Cela crée un nouveau commit qui annule les changements du commit précédent.

ETAPE 9-10

**1. Créer un dépôt sur GitHub :**

1. Connecte-toi à [GitHub](https://github.com).
2. Dans le coin supérieur droit, clique sur le signe "plus" (+) et sélectionne **New repository** (Nouveau dépôt).
3. Remplis les champs :
   * **Repository name** : Donne un nom à ton dépôt (par exemple, mon-projet).
   * **Description (optionnel)** : Ajoute une description pour ton dépôt si tu le souhaites.
   * **Public ou Private** : Choisis si tu veux que ton dépôt soit public (visible par tous) ou privé.
   * **Initialize this repository with** : Tu peux cocher cette option si tu veux initialiser ton dépôt avec un fichier README.md, .gitignore, ou une licence. Si ton projet est déjà local, il est préférable de laisser cette option décochée.
4. Clique sur **Create repository**.

**2. Connecter ton projet local à ce dépôt GitHub :**

Si tu as déjà un projet local avec Git initialisé, voici comment le connecter au dépôt GitHub que tu viens de créer.

**a. Ouvre ton terminal et accède au projet local :**

cd /chemin/vers/ton/projet

**b. Initialise Git dans ton projet (si ce n'est pas déjà fait) :**

Si ton projet n'est pas encore initialisé avec Git :

git init

**c. Ajoute les fichiers à la zone de staging et fais un commit initial (si ce n’est pas déjà fait) :**

git add .

git commit -m "Initial commit"

**d. Ajoute le dépôt GitHub en tant que dépôt distant (remote) :**

Remplace URL\_DU\_DEPOT par l'URL HTTPS du dépôt que tu as créé sur GitHub (tu la trouveras sur la page de ton dépôt).

git remote add origin https://github.com/ton-utilisateur/mon-projet.git

**e. Pousse les fichiers locaux vers GitHub :**

git push -u origin main

Note : Si ta branche principale est master et non main, utilise master au lieu de main dans la commande ci-dessus.

**3. Vérifie sur GitHub :**

Une fois le push effectué, va sur la page du dépôt sur GitHub. Tu devrais voir les fichiers de ton projet apparaître.

**Résumé des commandes :**

1. **Initialise Git (si ce n'est pas fait)** :

git init

1. **Ajoute les fichiers à la zone de staging et fais un commit** :

git add .

git commit -m "Initial commit"

1. **Ajoute le dépôt distant GitHub** :

git remote add origin https://github.com/ton-utilisateur/mon-projet.git

1. **Pousse le projet vers GitHub** :

git push -u origin main

Etape 11 :

* Commande : git clone <https://github.com/ton-utilisateur/nom-du-depot.git>
* Commande : git fetch
* Git spécifiera les fichiers concernés par le conflit et demandera de les réglés manuellement

ETAPE 12

git pull. Elle combine les actions de git fetch et git merge en une seule étape :

git pull

**Détails pour git pull :**

* git pull : Cette commande récupère les modifications depuis le dépôt distant et fusionne immédiatement les changements dans ta branche locale. C'est une manière pratique de mettre à jour ta branche locale avec les dernières modifications du dépôt distant.

L’importance se trouve dans la facilitation du rassemblement des modifications de chacun permettant un travail plus rapide

Etape 13

 **Évitez les phrases longues et complexes.** Allez droit au but en décrivant clairement la modification apportée.

 **Utilisez des verbes d'action au début de la phrase.** Par exemple : "Ajout de", "Correction de", "Modification de".

 **ÊTRE spécifique.** Au lieu de "Correction de bug", précisez "Correction du bug #123 qui empêchait...".

 **Adoptez une convention de nommage cohérente.** Cela facilite la recherche et la compréhension des commits.

 **Utilisez des templates ou des modèles de messages.** Cela vous aide à structurer vos messages et à inclure les informations essentielles.

 **Expliquez pourquoi cette modification est nécessaire.** Quel problème résout-elle ?

 **Citez les tickets ou les issues associés.** Cela permet de tracer les modifications et de faciliter la revue de code.

 **Limitez la longueur des lignes.** Cela améliore la lisibilité.

 **Utilisez des phrases complètes.** Les abréviations peuvent être difficiles à comprendre.

 **Évitez les émojis.** Bien qu'ils puissent être amusants, ils peuvent ne pas être universels.

* ml

 **Convention de nommage claire:** Utilisez des noms de répertoires explicites et cohérents (par exemple, src pour le code source, tests pour les tests, docs pour la documentation).

 **Séparation des préoccupations:** Organisez votre code par fonctionnalités, modules ou types de fichiers pour faciliter la navigation.

 **Standardisation:** Adoptez une structure de répertoires standard au sein de votre équipe ou de votre projet pour assurer la cohérence.

ETAPE 14 :

* Git Flow est une méthodologie de gestion de branches Git qui propose un ensemble de conventions et de commandes pour structurer le développement d'un projet. Elle définit un ensemble de branches principales et des règles pour les créer, fusionner et supprimer.
* Git Flow est particulièrement adapté aux projets de taille moyenne à grande avec une équipe de développement importante et un cycle de release régulier. Cependant, pour les petits projets ou les projets avec un cycle de développement très rapide, des modèles plus simples comme GitHub Flow peuvent être plus adaptés.

ETAPE 16 :

* Git flow init