TP : Git - Gestion de version et Collaboration

---

1. Créer un dépôt sur GitHub UI

Objectif : Initialiser le projet via l'interface GitHub

Exercice 1.1 : Créer un nouveau repository

Vérification GitHub UI :

- ☐ Connecté sur https://github.com
- ☐ Cliquer sur l'icône "+" en haut à droite → "New repository"
- ☐ Remplir les champs :
  - **Repository name** : mon-projet-git
  - **Description** : TP complet Git et GitHub
  - **Public** (pour pouvoir le voir facilement)
  - ☐ Cocher "Add a README file"
  - ☐ Cocher "Add .gitignore" → sélectionner "Node"
  - ☐ Cocher "Choose a license" → MIT
- ☐ Cliquer "Create repository"

Exercice 1.2 : Observer la structure initiale

Vérification GitHub UI :

- ☐ Voir le dépôt créé avec 3 fichiers : README.md, .gitignore, LICENSE
- ☐ Onglet "Code" affiche les fichiers
- ☐ Onglet "Commits" : voir le commit initial "Initial commit"
- ☐ Onglet "Branches" : voir la branche "main" seule

---

2. Cloner le dépôt et configuration locale

Objectif : Récupérer le code sur la machine locale et le configurer

Exercice 2.1 : Cloner le repository

Vérification GitHub UI :

- ☐ Sur le repo, cliquer le bouton vert "Code" (en haut à droite)
- ☐ Copier l'URL HTTPS : `https://github.com/VOTRE_USER/mon-projet-git.git`

Commandes :

```
git clone https://github.com/VOTRE_USER/mon-projet-git.git
cd mon-projet-git
```

Vérification locale :

- ☐ Voir les fichiers : README.md, .gitignore, LICENSE
- ☐ Exécuter : `git log --oneline` → voir "Initial commit"
- ☐ Exécuter : `git branch -a` → voir "main" et "origin/main"

Exercice 2.2 : Vérifier la configuration du remote

```
git remote -v
git config --list | grep user
```

Résultat attendu :

```
origin  https://github.com/VOTRE_USER/mon-projet-git.git (fetch)
origin  https://github.com/VOTRE_USER/mon-projet-git.git (push)
```

Exercice 2.3 : Configurer l'utilisateur Git (si nécessaire)

```
git config user.name "Votre Nom"
git config user.email "votre.email@example.com"
git config --global user.name "Votre Nom"  # Globalement
git config --global user.email "votre.email@example.com"
```

Vérification :

```
git config user.name
git config user.email
```

---

3. Premiers changements et synchronisation

Objectif : Ajouter du contenu et le pousser vers GitHub

Exercice 3.1 : Modifier le README et committer

```
echo "## Objectif du projet
Apprendre Git et GitHub en pratique." >> README.md

git status
git add README.md
git commit -m "docs: ajouter objectif au README"
git log --oneline
```

Vérification GitHub UI :

- ☐ Le commit n'apparaît PAS encore sur GitHub (il est local)
- ☐ Exécuter : `git push origin main`

Exercice 3.2 : Pousser vers GitHub

```
git push origin main
```

Vérification GitHub UI :

- ☐ Rafraîchir https://github.com/VOTRE_USER/mon-projet-git
- ☐ README.md est mis à jour avec le nouvel objectif
- ☐ Onglet "Commits" affiche le nouveau commit
- ☐ Cliquer sur le commit pour voir l'auteur et la diff

Exercice 3.3 : Créer les fichiers du projet

```
echo "# Configuration du projet" > config.js
echo "console.log('App started');" > app.js
echo "body { margin: 0; }" > style.css

git add .
git commit -m "feat: créer structure initiale du projet"
git push origin main
```

Vérification GitHub UI :

- ☐ Les 3 fichiers apparaissent dans l'onglet "Code"
- ☐ Onglet "Commits" : voir les 2 derniers commits
- ☐ Voir le graphique en haut : "main" avec 2 commits

---

4. Créer et gérer des branches

Objectif : Utiliser des branches courtes et les fusionner

Exercice 4.1 : Créer une branche feature localement

```
git switch main
git pull origin main
git switch -c feat/authentification

echo "// Authentification module" > auth.js
git add auth.js
git commit -m "feat(auth): créer module d'authentification"
```

Vérification locale :

```
git branch -a
git log --oneline --graph --all
```

Exercice 4.2 : Pousser la branche vers GitHub

```
git push origin feat/authentification
```

Vérification GitHub UI :

- ☐ GitHub affiche une notification "feat/authentification had recent pushes"
- ☐ Onglet "Branches" : voir "feat/authentification" et "main"
- ☐ Voir "1 commit ahead of main"
- ☐ Bouton "Compare & pull request" est visible

Exercice 4.3 : Créer d'autres branches avec conventions

```
git switch main
git pull origin main
git switch -c fix/style-responsive
```

```
echo "/* Mobile responsive */" >> style.css
git commit -am "fix(ui): rendre design responsive"
git push origin fix/style-responsive

git switch main
git switch -c chore/update-dependencies

echo "Updated: npm packages" > CHANGELOG.md
git commit -am "chore: mettre à jour dépendances"
git push origin chore/update-dependencies
```

Vérification GitHub UI :

- ☐ Onglet "Branches" affiche 4 branches
- ☐ Chaque branche affiche le nombre de commits ahead
- ☐ Chaque branche a son propre historique

---

5. Créer des Pull Requests (PR)

Objectif : Proposer des changements via PR et les réviser

Exercice 5.1 : Créer une PR via GitHub UI

Vérification GitHub UI :

- ☐ Aller sur "Pull requests" → "New pull request"
- ☐ Sélectionner : base: main ← compare: feat/authentification
- ☐ Cliquer "Create pull request"
- ☐ Remplir :
  - **Title** : feat(auth): ajouter module d'authentification
  - **Description** : Ajoute un module d'authentification réutilisable. Fixes #1 (ou laisser vide)
- ☐ Cliquer "Create pull request"

Exercice 5.2 : Examiner la PR

Vérification GitHub UI :

- ☐ Onglet "Conversation" : voir la description et les détails
- ☐ Onglet "Commits" : voir le(s) commit(s) de la branche
- ☐ Onglet "Files changed" : voir les fichiers modifiés
  - ☐ Voir le diff : rouge (suppression) et vert (ajout)
  - ☐ Cliquer sur "+" pour ajouter un commentaire en ligne

Exercice 5.3 : Ajouter des commentaires sur la PR

Vérification GitHub UI :

- ☐ Dans "Files changed", cliquer "+" sur une ligne de auth.js
- ☐ Ajouter un commentaire : "Bonne implémentation !"
- ☐ Cliquer "Comment"
- ☐ Voir le commentaire apparaître dans "Conversation"

Exercice 5.4 : Apporter des modifications suite aux retours

```
# Localement
git switch feat/authentification
echo "// Improved error handling" >> auth.js
git commit -am "fix(auth): améliorer gestion des erreurs"
git push origin feat/authentification
```

Vérification GitHub UI :

- ☐ La PR se met à jour automatiquement
- ☐ Onglet "Commits" : voir 2 commits maintenant
- ☐ Voir "X new commits pushed"
- ☐ Onglet "Files changed" : voir tous les changements cumulés

---

6. Fusion des Pull Requests

Objectif : Fusionner une PR proprement et mettre à jour la branche locale

Exercice 6.1 : Fusionner via GitHub UI

Vérification GitHub UI :

- ☐ Dans la PR, voir "This branch can be merged"
- ☐ Cliquer "Merge pull request"
- ☐ Choisir le type de merge : "Create a merge commit"
- ☐ Confirmer le message : "Merge pull request #1 from VOTRE_USER/feat/authentification"
- ☐ Cliquer "Confirm merge"

Vérification GitHub UI :

- ☐ Voir "Pull request successfully merged and closed"
- ☐ Onglet "Branches" : feat/authentification affiche "Delete branch"
- ☐ Onglet "Code" : main contient maintenant auth.js

Exercice 6.2 : Synchroniser localement

```
git switch main
git pull origin main
git log --oneline --graph --all
```

Résultat attendu :

- ☐ main est à jour avec le merge commit
- ☐ Voir le commit de fusion dans l'historique
- ☐ Voir auth.js dans les fichiers locaux

Exercice 6.3 : Nettoyer les branches locales

```
git branch -a
git branch -d feat/authentification  # Supprimer localement
git push origin --delete feat/authentification  # Supprimer sur GitHub
```

Vérification GitHub UI :

- ☐ Onglet "Branches" : feat/authentification a disparu

---

7. Résolution de conflits

Objectif : Créer, identifier et résoudre un conflit de merge

Exercice 7.1 : Créer un conflit intentionnellement

```
# Branche 1
git switch -c feat/version-v1
echo "Version 1.0.0" > VERSION.txt
git commit -am "feat: ajouter numéro version v1"
git push origin feat/version-v1

# Revenir et créer une autre branche
git switch main
git pull origin main
git switch -c feat/version-v2
echo "Version 2.0.0" > VERSION.txt
git commit -am "feat: ajouter numéro version v2"
git push origin feat/version-v2
```

Exercice 7.2 : Créer une PR avec conflit

Vérification GitHub UI :

- ☐ Créer PR : feat/version-v2 → main
- ☐ GitHub affiche "Can't automatically merge"
- ☐ Voir le message d'avertissement en rouge

Exercice 7.3 : Résoudre via l'éditeur web GitHub

Vérification GitHub UI :

- ☐ Cliquer "Resolve conflicts"
- ☐ Voir l'éditeur de conflit avec marqueurs :

  ```
  <<<<<<< feat/version-v2
  Version 2.0.0
  =======
  Version 1.0.0
  >>>>>>> main
  ```

- ☐ Éditer pour garder "Version 2.0.0"
- ☐ Supprimer les marqueurs `<<<<<<<`, `=======`, `>>>>>>>`
- ☐ Cliquer "Mark as resolved"
- ☐ Cliquer "Commit merge"

Vérification GitHub UI :

- ☐ "This branch can be merged" réapparaît
- ☐ Cliquer "Merge pull request"

Exercice 7.4 : OU Résoudre localement

```
git fetch origin
git switch feat/version-v1
git merge origin/main
# Voir le conflit
git status
# Éditer VERSION.txt : choisir la version correcte
nano VERSION.txt  # Garder une seule version
git add VERSION.txt
git commit -m "Merge: résoudre conflit VERSION"
git push origin feat/version-v1
```

Vérification GitHub UI :

- PR se met à jour automatiquement
- "Can be merged" visible
- Merge et fermer la PR

---

8. Tags et Releases

Objectif : Marquer les versions et créer des releases

Exercice 8.1 : Créer un tag localement et le pousser

```
git switch main
git pull origin main
git tag -a v1.0.0 -m "Release initiale - Authentification"
git push origin v1.0.0
```

Vérification GitHub UI :

- Onglet "Releases" : voir le tag v1.0.0
- Cliquer sur le tag pour voir les détails

Exercice 8.2 : Créer une Release via GitHub UI

Vérification GitHub UI :

- "Releases" → "Create a new release"
- Tag version : sélectionner "v1.0.0" ou créer "v1.1.0"
- Remplir :
  - **Release title** : Version 1.0.0 - Authentification
  - **Description** :

    ```
    ## Features
    - Ajouter module d'authentification
    - Design responsive
    - Mises à jour des dépendances
    ```

- "Set as a pre-release" (décocher si version finale)
- Cliquer "Publish release"

Exercice 8.3 : Récupérer les tags localement

```
git fetch origin --tags
git tag -l
git show v1.0.0
git log --oneline --decorate --all
```

Résultat attendu :

- ✅ Voir tous les tags locaux
- ✅ Tags affichés dans le log avec `(tag: v1.0.0)`

---

9. Gestion des fichiers ignorés

Objectif : Configurer .gitignore et vérifier l'effet

Exercice 9.1 : Observer le .gitignore existant

```
cat .gitignore
```

Exercice 9.2 : Modifier .gitignore et tester

```
# Ajouter des lignes
echo "*.env" >> .gitignore
echo ".DS_Store" >> .gitignore
echo "node_modules/" >> .gitignore

# Créer des fichiers ignorés
touch .env.local
touch .DS_Store
mkdir -p node_modules
touch node_modules/package.json

git status  # Ne doit voir aucun des fichiers ci-dessus
```

Exercice 9.3 : Committer le .gitignore mis à jour

```
git add .gitignore
git commit -m "chore: ajouter patterns au .gitignore"
git push origin main
```

Vérification GitHub UI :

- ✅ .gitignore est mis à jour sur GitHub
- ✅ Les fichiers .env.local, .DS_Store, node_modules n'apparaissent pas

---

10. Gestion des Issues et traçabilité

Objectif : Créer des issues et les lier aux commits/PR

Exercice 10.1 : Créer une issue via GitHub UI

Vérification GitHub UI :

- Onglet "Issues" → "New issue"
- Remplir :
  - **Title** : Ajouter page de profil utilisateur
  - **Description** :

    ```
    ## Description
    Les utilisateurs doivent pouvoir voir leur profil

    ## Tasks
    - [ ] Créer route /profile
    - [ ] Créer template profil.html
    - [ ] Ajouter lien dans navbar
    ```

  - **Labels** : enhancement, feature
  - **Assignee** : vous-même
- Cliquer "Submit new issue"
- Noter le numéro : #2 (ou autre)

Exercice 10.2 : Créer une branche depuis l'issue

Vérification GitHub UI :

- Dans l'issue, voir "Create a branch"
- Cliquer dessus, GitHub crée automatiquement une branche

Localement :

```
git fetch origin
git switch issue-2-ajouter-page-de-profil  # Branche créée
# OU créer manuellement
git switch -c feat/user-profile
```

Exercice 10.3 : Lier commits à l'issue

```
echo "// User profile page" > profile.html
git commit -am "feat(profile): créer page profil utilisateur - fixes #2"
git push origin feat/user-profile
```

Vérification GitHub UI :

- Créer PR
- Dans la description : "fixes #2"
- GitHub reconnaît le lien
- Après merge : l'issue #2 se ferme automatiquement

---

11. Lire l'historique et déboguer

Objectif : Maîtriser les commandes de lecture d'historique

Exercice 11.1 : Visualiser l'historique complet

```
git log --oneline --graph --all --decorate
git log --oneline --graph --decorate main..origin/main
```

Exercice 11.2 : Trouver qui a modifié une ligne

```
git blame profile.html
git blame style.css | head -5
```

Exercice 11.3 : Examiner un commit spécifique

```
git show <sha-du-commit>
git show <sha>:profile.html  # Voir le fichier à ce commit
```

Exercice 11.4 : Chercher dans l'historique

```
git log --grep="authentification"  # Par message
git log -S "auth.js"  # Par contenu de fichier
git log --author="Votre Nom"
```

---

12. Annuler et corriger

Objectif : Maîtriser les techniques d'annulation

Exercice 12.1 : Corriger le dernier commit

```
echo "error" >> profile.html
git commit -am "wip: test"

# Corriger le message
git commit --amend -m "feat(profile): amélioration profil"
git log --oneline  # SHA change
```

Exercice 12.2 : Retirer du staging

```
echo "secret" > .env
git add .env profile.html
git restore --staged .env
git status  # .env n'est plus staged
```

Exercice 12.3 : Restaurer un fichier

```
echo "erreur" > profile.html
git restore profile.html  # Récupérer la version du repo
cat profile.html  # Contenu restauré
```

Exercice 12.4 : Revert pour l'historique public

```
git revert <sha-d-un-ancien-commit>
git log --oneline  # Nouveau commit inverse visible
git push origin main
```

---

13. Synthèse - Workflow complet

Étape	GitHub UI	Commandes Git	Vérification
1. Créer repo	New repository	-	README, .gitignore, LICENSE
2. Cloner	Copy URL	`git clone <url>`	Fichiers locaux présents
3. Branche	Voir "Branches"	`git switch -c feat/nom`	`git branch -a`
4. Commits	Voir dans "Commits"	`git commit -m "..."`	`git log --oneline`
5. Push	Voir notification	`git push origin feat/nom`	Branche visible sur GitHub
6. PR	"New PR"	GitHub UI	"Compare & pull request"
7. Review	"Files changed"	Ajouter commits : `git push`	PR se met à jour
8. Merge	"Merge PR"	`git switch main && git pull`	main est à jour
9. Tag	"Releases"	`git tag -a v1.0.0`	Tag visible
10. Issue	"New issue"	`git commit -m "fixes #123"`	Issue fermée après merge

Checklist finale :

- [ ] Repo cloné localement avec 3 fichiers initiaux
- [ ] Au moins 3 branches créées avec conventions (feat/, fix/, chore/)
- [ ] Chaque branche pushée vers GitHub
- [ ] PR créées et fusionnées
- [ ] Historique visible sur GitHub "Network"
- [ ] Conflits résolus (au moins 1)
- [ ] Tags et releases créés (au moins v1.0.0)
- [ ] Issues créées et fermées automatiquement
- [ ] .gitignore fonctionne (fichiers ignorés n'apparaissent pas)
- [ ] Tous les commits ont un auteur correct