Exercice pratique : Revenir en arrière et créer des tags

Petit exercice pratique...

5TTR 6TTR 6TQ Découverte

Objectifs

- Comprendre la différence entre revenir temporairement (`checkout`) et définitivement (`reset`).
- Apprendre à marquer une version du projet avec un **tag**.

Étapes de l'exercice

1. Créer un dépôt et faire quelques commits

```
mkdir projet-tags
cd projet-tags
git init
```

1. Crée un premier fichier :

```
echo "Version 1 : Base du projet" > fichier.txt
git add .
git commit -m "Commit 1 - Base du projet"
```

2. Ajoute une deuxième version :

```
echo "Version 2 : Ajout d'une fonctionnalité" >> fichier.txt
git commit -am "Commit 2 - Nouvelle fonctionnalité"
```

3. Ajoute une troisième version :

```
echo "Version 3 : Correction d'un bug" >> fichier.txt
git commit -am "Commit 3 - Correction"
```

2. Explorer l’historique

Affiche les commits sous forme compacte :

```
git log --oneline
```

👉 Note les 3 identifiants (hash) des commits.

---

3. Revenir à un commit précédent (lecture seule)

Reviens sur le premier commit :

```
git checkout <hash_du_commit1>
```

👉 Vérifie le contenu du fichier : il doit contenir uniquement "**Version 1 : Base du projet**".

Puis reviens sur la branche principale :

```
git checkout main
```

---

4. Créer des tags

1. Crée un tag `v1.0` sur le **premier commit** :

```
git tag v1.0 <hash_du_commit1>
```

2. Crée un tag `v2.0` sur le **deuxième commit** :

```
git tag v2.0 <hash_du_commit2>
```

3. Crée un tag `v3.0` sur le **troisième commit** (le plus récent) :

```
git tag v3.0
```

👉 Vérifie la liste :

```
git tag
```

---

5. Naviguer avec les tags

Teste la navigation avec `checkout` :

```
git checkout v1.0
cat fichier.txt
```

👉 Le fichier doit montrer la **Version 1**.

Puis :

```
git checkout v3.0
cat fichier.txt
```

👉 Tu retrouves la **Version 3**.

Questions de réflexion

1. Quelle est la différence entre `checkout <hash>` et `checkout v1.0` ?
2. Que se passe-t-il si tu supprimes un fichier après un `checkout` sur un ancien commit ?
3. Pourquoi est-il utile d'avoir des tags comme `v1.0`, `v2.0` au lieu de retenir les hash ?

👉 Cet exercice permet de **manipuler l'historique**, de comprendre le mode "lecture seule" (detached HEAD), et de voir l'intérêt des **tags pour marquer des versions stables**.

Correction détaillée – Exercice `checkout` & `tag`

Prérequis:

- Dépôt initialisé, 3 commits réalisés sur `fichier.txt` (Versions 1, 2, 3).

---

1) Historique compact

Commande

```
git log --oneline
```

Attendu (exemple, les hash varient)

```
a3f9c1d Commit 3 - Correction
7b2e4a9 Commit 2 - Nouvelle fonctionnalité
c18d0f3 Commit 1 - Base du projet
```

- Note les **hash** des 3 commits : `C3`, `C2`, `C1` (abréviations).

2) `checkout` temporaire sur un ancien commit

Commande

```
git checkout C1
```

Observation

- Git affiche un message du type "**You are in 'detached HEAD' state**".
- Tu n'es plus sur `main`, mais sur un **commit précis**.

Vérification

```
type fichier.txt   # (Windows)
# ou
cat fichier.txt    # (macOS/Linux)
```

Attendu

```
Version 1 : Base du projet
```

Revenir sur la branche

```
git checkout main
```

Attendu

- Retour sur `main`.
- Le contenu reflète le dernier commit (Version 1 + 2 + 3).

3) Création des tags

1. Tag sur le premier commit

```
git tag v1.0 C1
```

2. Tag sur le deuxième commit

```
git tag v2.0 C2
```

3. Tag sur le dernier commit (HEAD)

```
git tag v3.0
```

4. Lister les tags

```
git tag
```

Attendu

```
v1.0
v2.0
v3.0
```

4) Navigation avec des tags

Aller au tag v1.0

```
git checkout v1.0
cat fichier.txt
```

Attendu

```
Version 1 : Base du projet
```

> Remarque : encore **DETACHED HEAD** (normal quand on checkout un tag).

Aller au tag v3.0

```
git checkout v3.0
cat fichier.txt
```

Attendu

```
Version 1 : Base du projet
Version 2 : Ajout d'une fonctionnalité
Version 3 : Correction d’un bug
```

Revenir travailler normalement

```
git checkout main
```

5) Points de compréhension (réponses brèves)

Q1. Différence entre `checkout <hash>` et `checkout v1.0` ?

- Aucune différence fonctionnelle majeure : **les deux pointent un commit précis**.
- Le tag est un **alias lisible** et stable, plus facile à retenir qu’un hash.

Q2. Que se passe-t-il si tu modifies/supprimes un fichier après un `checkout` sur un ancien commit (detached HEAD) ?

- Tu peux modifier et **committer**, mais **tu n’es sur aucune branche**.
- Si tu veux conserver ce travail, **crée une branche** depuis là :

```
git switch -c correction-rapide
```

Q3. Pourquoi taguer (`v1.0`, `v2.0`) plutôt que mémoriser des hash ?

- Les tags :
  - **lisibles** (`v1.0` > `a3f9c1d`),
  - **stables** et **partageables** (versions, releases),
  - facilitent le support, les déploiements, les retours en arrière.

6) (Optionnel) Partager les tags sur GitHub

Un tag spécifique

```
git push origin v1.0
```

Tous les tags

```
git push --tags
```

7) Erreurs/Comportements attendus fréquents

- **Oubli : rester en detached HEAD** Symptôme : “Pourquoi je ne vois pas ma branche ?” → Revenir sur `main` avec `git checkout main`.
- **Tag égaré** Si `git tag` ne montre pas le tag : vérifier l’orthographe, la position (commit visé), ou refaire :

```
git tag -d v1.0
git tag v1.0 C1
```

- **Confusion reset/checkout**
  - `checkout` = **navigation** (temporaire).
  - `reset --hard` = **réécriture** de l’historique de la branche (destructif).

8) Critères de réussite (auto-évaluation)

- [ ] J’ai listé l’historique en `--oneline`.
- [ ] J’ai navigué sur un ancien commit et observé le **detached HEAD**.
- [ ] J’ai créé `v1.0`, `v2.0`, `v3.0` et je peux les lister.
- [ ] Je sais `checkout` un tag et vérifier le contenu du fichier.
- [ ] Je suis revenu sur `main` pour continuer à travailler.
- [ ] (Optionnel) J’ai poussé mes tags sur GitHub.