Spickzettel: Git Submodule – Externe Repositories einbinden

1. Was sind Git Submodule?

- Submodule ermöglichen das Einbinden externer Repositories innerhalb eines Git-Projekts.
- Nützlich für geteilte Bibliotheken oder Abhängigkeiten.
- Submodule behalten ihre eigene Versionshistorie und sind nicht direkt Teil des Hauptrepositories.

2. Submodule zu einem Repository hinzufügen

```
git submodule add <repo-url> <pfad>
```

- Beispiel:

```
git submodule add https://github.com/example/library.git libs/library
```

3. Submodule initialisieren & aktualisieren

Falls ein Repository mit Submodulen geklont wurde, sind die Submodule zunächst leer.

```
git submodule init    # Submodule initialisieren
git submodule update  # Submodule herunterladen
```

4. Alle Submodule auf neuesten Stand bringen

```
git submodule update --remote
```

- Holt die neuesten Änderungen aus dem Submodule-Repository.

5. Submodule aus Repository entfernen

```
git submodule deinit -f <pfad>
rm -rf .git/modules/<pfad>
git rm -f <pfad>
```

6. Submodule in einem anderen Branch aktualisieren

```
git checkout <branch>
git submodule update --remote --merge
```

7. Submodule als eigenständiges Repository bearbeiten

Falls du Änderungen direkt im Submodule machen möchtest:

```
cd <pfad-zum-submodule>
git checkout -b feature-branch
git commit -m "Änderung im Submodule"
git push origin feature-branch
```

- Anschließend im Hauptprojekt das Submodule-Commit aktualisieren:

```
git add <pfad-zum-submodule>
git commit -m "Submodule aktualisiert"
```

Best Practices

- Submodule nur verwenden, wenn eine echte Trennung nötig ist (z. B. externe Abhängigkeiten).
- Nach dem Klonen eines Repositories immer `git submodule update --init` ausführen.
- Vor dem Wechseln zwischen Branches sicherstellen, dass alle Submodule aktuell sind.
- Änderungen im Submodule immer zuerst in dessen Repository committen, bevor das Hauptprojekt aktualisiert wird.