Spickzettel: Wie funktioniert Git technisch?

1. Git als Dateisystem

- Git ist **kein** klassisches Delta-basiertes System wie SVN.
- Arbeitet mit **Snapshots** statt Änderungen.
- Jeder Commit speichert einen kompletten Stand des Projekts.
- Unveränderte Dateien werden als Referenzen gespeichert → effizienter Speicherverbrauch.

2. Diffs & Komprimierung

- Git speichert Änderungen als **Blobs** (Binär-Objekte für Dateien).
- Referenzen auf frühere Objekte minimieren Speicherverbrauch.
- Unterschiedliche Methoden zur Speicherung:
  - **Lose Objekte**: Einzelne Commits als separate Dateien.
  - **Pack-Dateien**: Verdichtete Objekte für Effizienz (`.git/objects/pack`).

3. Datenstruktur in Git

- **Blobs:** Speichern Dateiinhalte.
- **Trees:** Speichern Verzeichnisse und enthalten Blobs und weitere Trees.
- **Commits:** Verweisen auf einen Tree und enthalten Metadaten.
- **Refs:** Zeigen auf Branches, Tags oder HEAD.

Beispiel: Ein Commit in Git

```
git cat-file -p HEAD
```

- Zeigt den Inhalt des letzten Commits an.

```
git cat-file -t <hash>
```

- Zeigt den Typ eines Objekts (`blob`, `tree`, `commit`).

4. Wie wird eine Version rekonstruiert?

- Ein **Commit-Objekt** enthält:
  - Referenz auf **Tree-Objekt** (Projektdateien zum Zeitpunkt des Commits).
  - Parent-Commit(s) für Historienverfolgung.
  - Autor, Datum & Commit-Message.
- Beim Checkout stellt Git aus Tree-Objekten den Zustand des Repos zu einem bestimmten Zeitpunkt wieder her.

```
git checkout <commit-hash>
```

- Rekonstruiert das Projekt zum gespeicherten Snapshot.

5. Garbage Collection & Optimierung

- Alte oder ungenutzte Objekte aufräumen:

```
git gc
```

- Pack-Dateien erzeugen, um Speicher zu optimieren:

```
git repack
```

Best Practices

- Git arbeitet effizient, wenn regelmäßig **Garbage Collection** (`git gc`) ausgeführt wird.
- **Pack-Dateien optimieren** Speicherplatz bei großen Repositories (`git repack`).
- Mit **`git cat-file`** und **`git ls-tree`** kann man tief in die Git-Objektstruktur schauen.