Spickzettel: Git Merging – Typen & Funktionsweise

1. Fast-Forward Merge

- **Wann?** Wenn der Zielbranch keine zusätzlichen Commits hat.
- **Was passiert?** Der Branch wird direkt an die Spitze des Zielbranches gesetzt.
- **Vorteil:** Saubere, lineare Historie.
- **Nachteil:** Keine expliziten Merge-Commits zur Nachverfolgung.

```
git checkout main
git merge feature-branch  # Automatisch Fast-Forward
```

2. Merge mit Commit (Three-Way Merge)

- **Wann?** Wenn beide Branches neue Commits haben.
- **Was passiert?** Ein neuer Merge-Commit wird erstellt.
- **Vorteil:** Zeigt, wann ein Merge stattfand.
- **Nachteil:** Kann eine nicht-lineare Historie erzeugen.

```
git checkout main
git merge --no-ff feature-branch  # Merge-Commit erzwingen
```

3. Squash Merge

- **Wann?** Wenn alle Commits eines Branches zu einem einzigen zusammengefasst werden sollen.
- **Was passiert?** Alle Commits werden zu einem einzigen kombiniert.
- **Vorteil:** Historie bleibt sauber.
- **Nachteil:** Einzelne Änderungen sind nicht mehr getrennt sichtbar.

```
git checkout main
git merge --squash feature-branch
```

4. Rebase statt Merge

- **Wann?** Wenn man eine saubere, lineare Historie erhalten will.
- **Was passiert?** Die Commits aus `feature-branch` werden neu auf `main` aufgesetzt.
- **Vorteil:** Keine unnötigen Merge-Commits.
- **Nachteil:** Kann problematisch sein, wenn der Branch bereits geteilt wurde.

```
git checkout feature-branch
git rebase main
```

5. Konflikte während eines Merges lösen

- Git stoppt den Merge und markiert Konflikte.

- Öffne die betroffenen Dateien und bereinige Konfliktstellen (<<<<<<<, =======, >>>>>>>).
- Danach:

```
git add <datei>
git commit -m "Merge-Konflikte gelöst"
```

Best Practices für Merging

- Regelmäßig `git pull --rebase` nutzen, um Konflikte frühzeitig zu vermeiden.
- Feature-Branches nicht zu lange ungemergt lassen.
- Bei Team-Projekten Merge-Strategie abstimmen (Fast-Forward oder Merge-Commit?).