Spickzettel: Git Hooks – Automatisierung von Workflows

1. Was sind Git Hooks?

- Skripte, die automatisch bei bestimmten Git-Ereignissen ausgeführt werden.
- Ermöglichen Automatisierung (z. B. Code-Formatierung, Tests, Benachrichtigungen).
- Werden lokal im Verzeichnis `.git/hooks/` gespeichert.

2. Verfügbare Hook-Typen

Hook	Wird ausgeführt bei...
`pre-commit`	Vor dem Commit (z. B. Code-Checks)
`prepare-commit-msg`	Vor dem Öffnen der Commit-Nachricht
`commit-msg`	Nach der Eingabe der Commit-Nachricht
`pre-push`	Vor dem Push zu einem Remote-Repo
`post-merge`	Nach einem erfolgreichen Merge
`pre-rebase`	Vor dem Start eines Rebase-Vorgangs

3. Neuen Hook erstellen

1. Zum `.git/hooks/`-Verzeichnis wechseln:

```
cd .git/hooks/
```

2. Neues Skript für einen Hook anlegen:

```
touch pre-commit
chmod +x pre-commit
```

3. Skript mit Inhalt füllen (z. B. `pre-commit` für Linter):

```
#!/bin/sh
npm run lint  # Führt den Linter vor dem Commit aus
```

4. Beispiel: `pre-commit` Hook für Formatierung

```
#!/bin/sh
black .  # Formatiert Python-Code mit Black
```

5. Beispiel: `commit-msg` Hook für Commit-Format

```
#!/bin/sh
if ! grep -q "#[0-9]" "$1"; then
  echo "Commit-Nachricht muss eine Ticket-Nummer enthalten!"
  exit 1
fi
```

6. Git Hooks für alle Repositories verwenden

- Standardmäßig sind Hooks **nur lokal** aktiv.
- Um Hooks projektübergreifend zu nutzen:

```
git config --global core.hooksPath ~/meine-hooks/
```

- Danach müssen Hooks im neuen Pfad (`~/meine-hooks/`) gespeichert werden.

Best Practices

- **Vor jedem Commit Code-Checks oder Tests einbinden** (`pre-commit`).
- **Hooks im Team synchronisieren** (`core.hooksPath` nutzen oder per Repository teilen).
- **Fehlertolerante Hooks schreiben**, um Entwicklungsfluss nicht zu blockieren.
- **Nicht zu viele Hooks setzen**, um unnötige Wartezeiten zu vermeiden.