Basics der Versionsverwaltung mit Git (lokale Repos)

Carsten Gips (HSBI)

Versionsverwaltung mit Git: Typische Arbeitsschritte

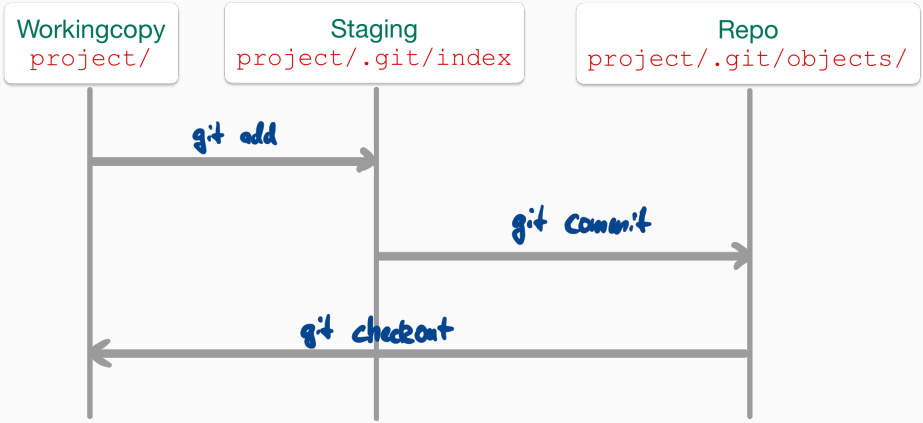
1. Repository anlegen (oder clonen)

2. Dateien neu erstellen (und löschen, umbenennen, verschieben)
3. Änderungen einpflegen (“committen”)
4. Änderungen und Logs betrachten
5. Änderungen rückgängig machen
6. Projektstand markieren (“taggen”)

7. Entwicklungszweige anlegen (“branchen”)
8. Entwicklungszweige zusammenführen (“mergen”)

9. Änderungen verteilen (verteiltes Arbeiten, Workflows)

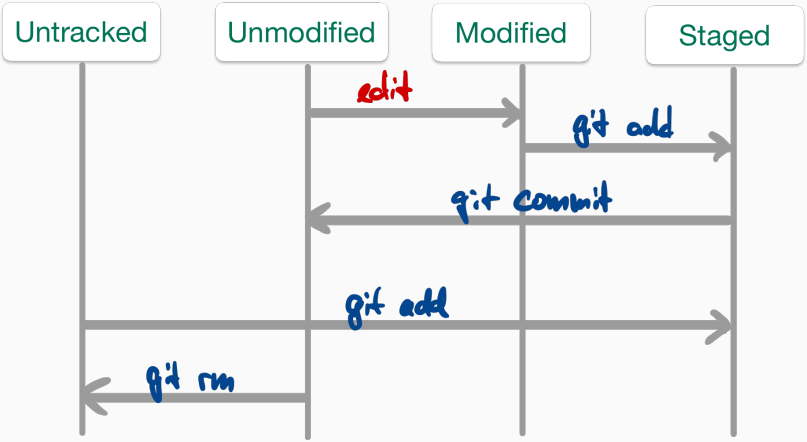
Dateien unter Versionskontrolle stellen



Abfrage mit `git status`

Konsole

Änderungen einpflegen



- Abfrage mit: `git status`
- "Staging" von modifizierten Dateien: `git add <file>`
- Committen der Änderungen im Stage: `git commit`

Letzten Commit ergänzen

- `git commit --amend -m "Eigentlich wollte ich das so sagen"`
- `git add <file>; git commit --amend`

Konsole

Commits betrachten

- Liste aller Commits: `git log`
  - `git log -<n>` oder `git log --since="3 days ago"`
  - `git log --stat`
  - `git log --author="pattern"`
  - `git log <file>`
- Inhalt eines Commits: `git show`

Konsole

Änderungen und Logs betrachten

- `git diff [<file>]`

  Änderungen zwischen Workingcopy und letztem Commit (ohne Stage)

- `git diff commitA commitB`

  Änderungen zwischen Commits

- Blame: `git blame <file>`

  Wer hat was wann gemacht?

Konsole

Dateien ignorieren: *.gitignore

```
# Compiled source #
*.class
*.o
*.so

# Packages #
*.zip

# All directories and files in a directory #
bin/**/*
```

man 5 gitignore

Zeitmaschine

- Änderungen in Workingcopy rückgängig machen
  - Änderungen nicht in Stage: `git checkout <file>` oder `git restore <file>`
  - Änderungen in Stage: `git reset HEAD <file>` oder `git restore --staged <file>`

- Datei aus altem Stand holen:
  - `git checkout <commit> <file>`, oder
  - `git restore --source <commit> <file>`
- Commit verwerfen, Geschichte neu: `git revert <commit>`

- Stempel (Tag) vergeben: `git tag <tagname> <commit>`
- Tags anzeigen: `git tag` und `git show <tagname>`

Konsole

Wann und wie committen?

Jeder Commit stellt einen Rücksetzpunkt dar!

- Kleinere “Häppchen” einchecken: ein Feature oder Task
- Logisch zusammenhängende Änderungen gemeinsam einchecken
- Projekt muss nach Commit compilierbar sein
- Projekt sollte nach Commit lauffähig sein

Schreiben von Commit-Messages: WARUM?!

```
Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to about
72 characters or so.  In some contexts, the first line is treated
as the subject of an email and the rest of the text as the body.
The blank line separating the summary from the body is critical
(unless you omit the body entirely); tools like rebase can get
confused if you run the two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too
  - Typically a hyphen or asterisk is used for the bullet, preceded
    by a single space, with blank lines in between, but conventions
    vary here
```

Quelle: “A Note About Git Commit Messages” by Tim Pope on tbaggery.com

Wrap-Up

- Änderungen einpflegen zweistufig (`add`, `commit`)
- Status der Workingcopy mit `status` ansehen
- Logmeldungen mit `log` ansehen
- Änderungen auf einem File mit `diff` bzw. `blame` ansehen
- Projektstand markieren mit `tag`
- Ignorieren von Dateien/Ordnern: Datei `.gitignore`

LICENSE

Exceptions

- Citation "*A Note About Git Commit Messages*" by Tim Pope on tbaggery.com