Middleware – Cloud Computing – Übung

Tobias Distler, Christopher Eibel,
Michael Eischer, Timo Hönig

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.cs.fau.de

Wintersemester 2016/17

Überblick

Organisatorisches
- Übung
- Versionsverwaltung mit Git

Termine und Ansprechpartner

- Tafelübung
  - Dienstag: 14:15–15:45 Uhr
  - Raum 0.031-113
- Rechnerübungen
  - Mittwoch: 16:00–18:00 Uhr (ab 26.10.)
  - CIP-Pool: Raum 00.153-113
  - Betreuung (nur) bei Bedarf
- Ansprechpartner
  - Christopher Eibel Raum 0.045 `ceibel@cs.fau.de`
  - Michael Eischer Raum 0.045 `eischer@cs.fau.de`
  - Tobias Distler Raum 0.039 `distler@cs.fau.de`
  - Timo Hönig Raum 0.050 `thoenig@cs.fau.de`
  - Alle: `mw@i4.informatik.uni-erlangen.de`

Übungsaufgaben

- Anmeldung (per WAFFEL)

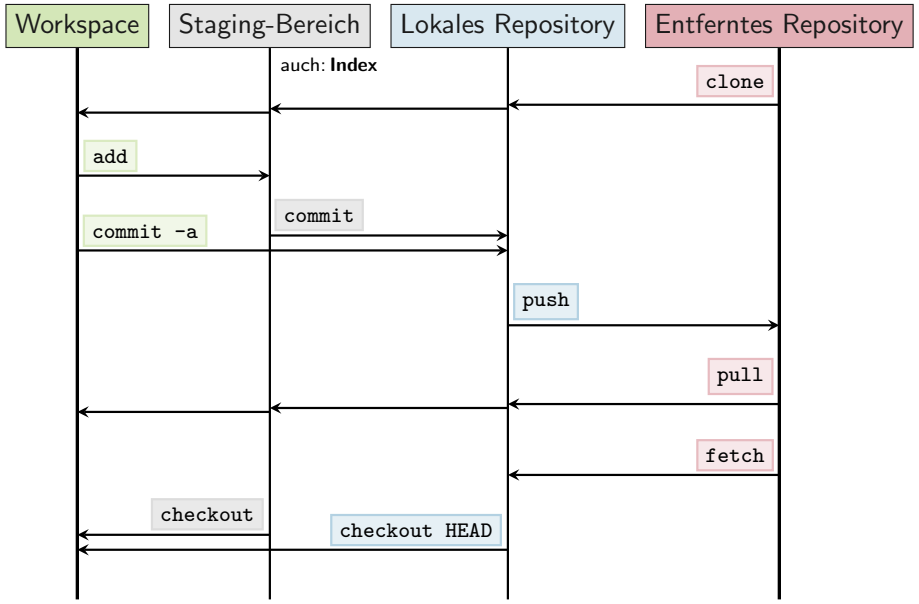
  `https://waffel.informatik.uni-erlangen.de/signup/?univisid=40740560`
- Bearbeitung
  - Persönliches Projektverzeichnis: `/proj/i4mw/<loginname>`
  - Bearbeitung in Gruppen
    - 3 Teilnehmer pro Gruppe
    - Festlegung der Gruppenzugehörigkeit: `/proj/i4mw/bin/mwgroups`
      (Neben der Gruppennummer werden außerdem die Zugangsdaten für das in Aufgabe 2 benötigte OpenStack per E-Mail mitgeteilt.)
  - Empfehlung: Git-Repository für die gesamte Gruppe → siehe Folie 0–6 ff.
- Abgabe
  - Präsentation der eigenen Implementierung
  - Falls eine Präsentation am Abgabetermin nicht möglich sein sollte: Rechtzeitige Mitteilung an Übungsleiter (per Mail / persönlich)

Überblick

GitLab: Repository erstellen

- Gilt für alle
  - Benutzerkonto erstellen: `https://gitlab.cs.fau.de` → „Sign up“
  - Öffentliche(n) SSH-Schlüssel hinzufügen:
    → Oben rechts auf das Profil-Logo und auf „Profile Settings“ klicken
    → Reiter „SSH Keys“ auswählen
    → Einen oder mehrere SSH-Schlüssel hinzufügen
    (siehe auch: `https://gitlab.cs.fau.de/help/ssh/README`)
- Nur durch ein Mitglied der Gruppe durchzuführen
  - Projekt für Gruppe erstellen
    → Auf „New Project“-Button klicken
    * **„Visibility Level“ = Private**
  - Gruppenmitglieder hinzufügen
    → Projekt bzw. Repository auswählen
    * Rechts auf ⚙ → „Members“
    * Namen der Gruppenmitglieder eingeben
    * Auswahlbox: „role permission“ (statt „Guest“) auf „Master“ setzen
    * Auf „Add to project“-Button klicken

Überblick über den Git-Arbeitsablauf



Einrichten eines Repository

`git clone/git config`

- Erstellen einer lokalen Arbeitskopie über ein entferntes Repository
  - Befehl:
    ```
    > git clone <URL>
    ```
  - Beispiel: `git clone` über SSH (SSH-Schlüssel nötig, siehe Folie 0–6)
    ```
    > git clone git@gitlab.cs.fau.de:MWCC-WS16/gruppe0/mwue.git
    ```
- Konfiguration
  - Befehl:
    ```
    > git config
    ```
  - Konfiguration systemweit (`--system`), benutzer- (`--global`) oder Repository-spezifisch (`.git/config`) möglich
  - E-Mail-Adresse und Name festlegen
    ```
    > git config --global user.name "Max Mustermann"
    > git config --global user.email max@mustermann.de
    ```
  - ... [→ `man 1 git-config`]

Dateien hinzufügen und entfernen

`git add/rm`

- Dateien werden zunächst nur dem Index (→ Staging-Bereich) hinzugefügt oder davon entfernt
- Es wird nur der **aktuelle** Zustand hinzugefügt
- Änderung(en) hinzufügen

```
> git add <file(s)-to-add>
```

- Datei(en) entfernen

```
> git rm <file(s)-to-remove>
```

- Änderungen werden erst beim nächsten Commit wirksam (d.h. in das lokale Repository übertragen) → siehe nächste Folie ff.

Änderungen einprüfen (1)

`git status`

- Auswirkungen des nächsten Commits überprüfen:

```
> git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   README.md
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# Makefile
```

- Commit-Aufbau: Commit-ID, Autor, Datum, Commit-Nachricht

```
commit f8ceebed8d581cab736350c055b072db148987cd
Author: Christopher Eibel <ceibel+gitlab@cs.fau.de>
Date:   Tue Aug 9 15:23:51 2016 +0200

Add initial README file

...
```

Änderungen einprüfen (2)

`git diff`

- Unterschiedliche Ausprägungen
  - Standardverhalten: Diff zwischen Workspace und Index

```
> git diff
```

  - Diff zwischen Workspace und einem bestimmten Commit

```
> git diff <commit>
```

  - Diff zwischen zwei bestimmten Commits
    (zeigt Änderungen von `revision_1` ggü. `revision_0`)

```
> git diff <revision_0> <revision_1> [<filename>]
```

- Unterschiede zu Dateien in einem Remote-Branch

```
> git diff <local_branch> <remote_branch>
```

Zum Beispiel: Unterschied von lokalem Branch 'master' zu Zustand von 'master' im entfernten Repository: `local_branch := master` u. `remote_branch := origin/master`
→ ! Vorheriges `git fetch` (siehe Folie 0–13) ratsam.

Änderungen einprüfen (3)

`git commit`

- Mit `commit` übernommene Änderungen sind zunächst nur im **lokalen** Repository sichtbar
  - Kompletten Index oder nur bestimmte Datei(en) übernehmen

```
> git commit [<file(s)-to-commit>]
```

  - Alle modifizierten Dateien übernehmen

```
> git commit -a
```

  - Commit-Nachricht direkt per Kommandozeile übergeben

```
> git commit -m <commit_message>
```

  - Vorherigen Commit modifizieren

```
> git commit --amend
```

- Commits vom lokalen in das **entfernte** Repository einprüfen

```
> git push [[remote_name] [branch_name]]
```

↪ Auflösung eines auftretenden Konflikts: siehe Folie 0–17

Änderungen ausprüfen

`git checkout/pull/fetch`

- Lokal
  - `> git checkout <branch>`

  → Aktuellen Stand aus dem Zweig `<branch>` übernehmen
  → Übernahme in den Workspace **und** in den Index
  → Operation ist „safe“, verwirft also keine Änderungen
  (führt aber zu evtl. notwendiger Konfliktauflösung, siehe Folie 0–17)

- Entfernt
  - `> git fetch`

  → Aktualisierung der Remote-Tracking-Branches (`refs/remotes/`)
  - `> git pull`

  → Zustand aus entferntem Repository holen und in ein anderes Repository oder in einen lokalen Branch integrieren[1]
  (→ ≃ `git fetch`, gefolgt von git merge)

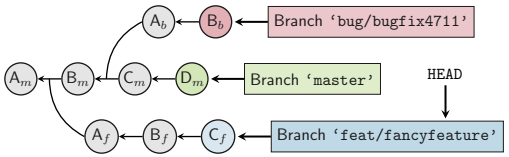
[1] `https://git-scm.com/docs/git-pull`

Branches (1)

Grundlegendes

- Für jedes neue Feature wird üblicherweise ein neuer Branch erstellt
- Anzeigen aller (auch entfernter) Branches

```
> git branch -a
  master
* feat/fancyfeature
  bug/bugfix4711
  remotes/origin/master
```



- Neuen lokalen Branch erstellen (ausgehend vom gegenwärtigen `HEAD`)

```
> git checkout -b <new_branch_name>
```

- Branch wechseln
  - Wechsel in bereits lokal verfügbaren Branch

```
> git checkout <local_branch>
```

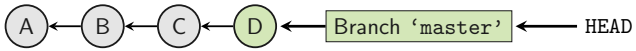
  - Remote-Branch in implizit neu erstellten lokalen Branch einspielen

```
> git checkout -b <local_branch> <remote_branch>
```


Branches (2)

`HEAD`

- Bedeutung von `HEAD`:

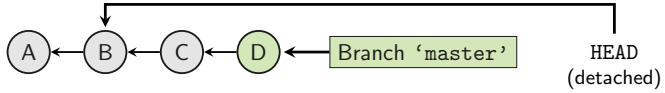


  - Eine Art Zeiger auf den aktuell ausgewählten Branch

```
> cat .git/HEAD
ref: refs/heads/master
```

  - Branch ist wiederum Zeiger auf aktuellsten Commit einer Verzweigung

- Losgelöster `HEAD`
  - Git sorgt normalerweise dafür, dass der `HEAD`-Zeiger automatisch weitergerückt wird (d. h., immer auf den aktuellen Branch zeigt)
  - Ausnahme: `HEAD` zeigt auf bestimmten Commit → `detached HEAD`

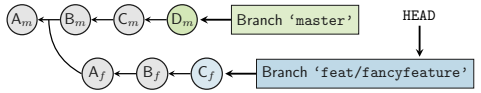


  - ❗ „detached“ bedeutet, dass folgende Commits keinem (benannten) Branch zugeordnet sind und bei Wechsel zu anderem Commit/Branch verlorengehen

Branches (3)

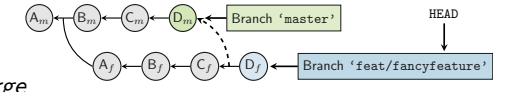
`git merge` vs. `git rebase`

- Irgendwann müssen verschiedene Zweige (wieder) vereint werden



- Prinzipiell zwei unterschiedliche Wege
  - Das klassische Mergen → Mergen von `<branch>` in `<other_branch>`:

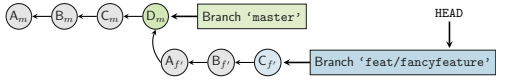
```
> git checkout <other_branch>
> git merge <branch>
```



    - Einfacher Fall: *fast-forward merge*
    - Fall mit eventuell notwendiger Konfliktauflösung: *3-way merge*

  - Rebase

```
> git checkout <other_branch>
> git rebase [-i] <branch>
```



    - Interaktives Rebase (`-i`): Historie neu schreiben
    - ❗ Sollte i. A. nicht auf öffentlichem Branch angewendet werden

Konfliktbewältigung (1)

- Konflikt feststellen

```
> git pull
[...]
   1b09b5d..39efa77  master -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

> cat README.md
<<<<<<< HEAD
TODO: Structure and fill this README.
=======
## Synopsis

## Installation
>>>>>>> 39efa77d814d4aebfecd37da8d252cfc80091907
```

- Konflikt auflösen

```
> $(EDITOR) README.md
> git add README.md
> git commit
```

Konfliktbewältigung (2)

-