

Verteilte Systeme – Übung

Christopher Eibel, Michael Eischer, Tobias Distler, Timo Hönig

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

www4.cs.fau.de

Sommersemester 2017



Überblick

Organisatorisches
Übung
Versionsverwaltung mit Git
Screen



VS-Übung (SS17)

Organisatorisches

0-2

Termine und Ansprechpartner

- Tafelübung
 - Donnerstag: 16:00–17:30 Uhr, Raum 0.031-113
- Rechnerübung
 - Mittwoch: 14:00–16:00 Uhr, CIP-Pool 02.151{a,b}-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: vs@i4.informatik.uni-erlangen.de



VS-Übung (SS17)

Organisatorisches – Übung

0-3

Übungsaufgaben

- Anmeldung (per WAFFEL)
 - <https://waffel.informatik.uni-erlangen.de/signup/?univisid=20545691>
- Bearbeitung
 - Persönliches Projektverzeichnis: `/proj/i4vs/<loginname>`
 - Bearbeitung in Gruppen
 - 3 Teilnehmer pro Gruppe
 - Festlegung der Gruppenzugehörigkeit: `/proj/i4vs/bin/vsgroups`
 - 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)



VS-Übung (SS17)

Organisatorisches – Übung

0-4

Organisatorisches

Übung

Versionsverwaltung mit Git

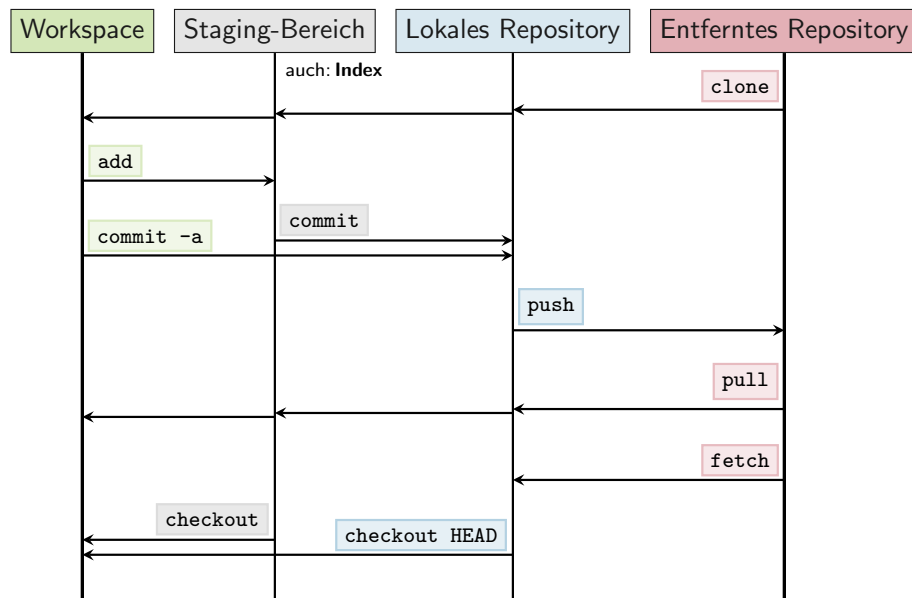
Screen



- 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
 - * „Settings“ → „Members“ auswählen
 - * 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:VS-SS17/gruppe0/vsue.git
```
 - URL steht in Gitlab auf der „Project“-Seite eines Repository
- 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 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.



- 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
```

```
...
```



- 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.



- 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 evtl. auftretenden Konflikts: siehe Folie 0-17



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)

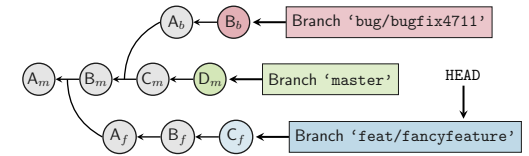
Entfernt

- > `git fetch`
 - Aktualisierung der Remote-Tracking-Branche (refs/remotes/)
- > `git pull`
 - Zustand aus entferntem Repository holen und in ein anderes Repository oder in einen lokalen Branch integrieren¹ (→ `≈ git fetch`, gefolgt von `git merge`)

¹<https://git-scm.com/docs/git-pull>

- 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

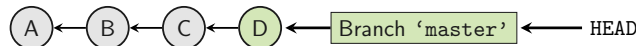
- Wechsel in bereits lokal verfügbaren Branch


```
> git checkout <local_branch>
```
- Remote-Branch in implizit neu erstellten lokalen Branch einspielen


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

Branches (2)

- Bedeutung von HEAD:

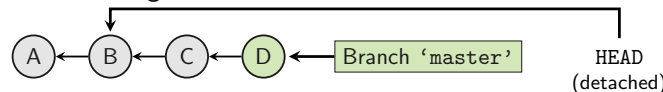


- Eine Art Zeiger auf den aktuell ausgewählten Branch


```
> cat .git/HEAD
ref: refs/heads/master
```
- Branch ist wiederum Zeiger auf aktuellsten Commit einer Verzweigung

- Losgelöster HEAD

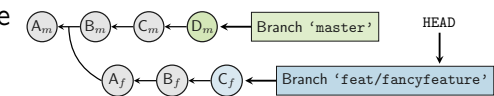
- Git sorgt normalerweise dafür, dass der HEAD-Zeiger automatisch weitergerückt wird (d. h., immer auf den aktuellen Branch zeigt)
- Ausnahme: HEAD zeigt auf bestimmten Commit → detached HEAD



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

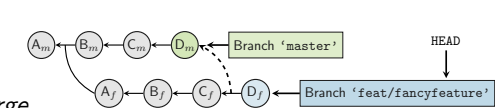
Branches (3)

- Irgendwann müssen verschiedene Zweige (wieder) vereint werden



- Prinzipiell zwei unterschiedliche Wege
- Das klassische Mergen → Mergen von <branch_src> in <branch_dst>:

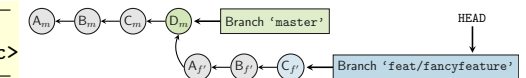
```
> git checkout <branch_dst>
> git merge <branch_src>
```



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

- Rebase

```
> git checkout <branch_dst>
> git rebase [-i] <branch_src>
```



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

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  
>>>>>> 39efa77d814d4aebfec37da8d252cfc80091907
```

Konflikt auflösen

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



Eclipse enthält Unterstützung für Git

Schritte zum Einrichten

1. Lokale Kopie des Repository erstellen

- Entweder mit `git clone`, siehe Folie 0-8
- oder direkt in Eclipse
 - * „File“ → „Import...“ → „Git“ → „Projects from Git“
 - * Anschließend „Clone URI“ auswählen und URL aus Gitlab einfügen
 - * Bei „Branch Selection“ auf weiter klicken
 - * Bei „Local Destination“ ggf. **Pfad** anpassen und diesen **merken**
 - * „Import using the New Project wizard“ auswählen

2. Als Projekt in Eclipse hinzufügen

- * Neues „Java“ → „Java Project“ auswählen
- * **„Use default location“ deaktivieren**
- * **Pfad des lokalen Repository eingeben**
→ Eclipse erkennt das git-Repository anschließend automatisch
- * Rest wie ohne Git

Git-Befehle sind nach Rechtsklick auf das Projekt über das „Team“ Untermenü verfügbar



„Push to Upstream“ ist nicht verfügbar

- Tritt bei bislang leerem Repository auf
- Dateien hinzufügen und einprüfen (`commit`)
- Einmalig „Push Branch 'master'...“ mit Standardeinstellungen verwenden



Programm zum Verwalten mehrerer virtueller Terminals

Erlaubt beliebiges Trennen und Fortsetzen von Sitzungen

Wichtige Tastatur-/Screen-Befehle:

<code>Ctrl+a c</code>	<code>screen</code>	Erstelle neues Fenster und wechsele zu diesem
<code>Ctrl+a Ctrl+a</code>	<code>other</code>	Zwischen letzten aktiven Fenstern wechseln
<code>Ctrl+a <num></code>	<code>select</code>	Springe zu Fenster <code><num></code>
<code>Ctrl+a "</code>	<code>windowlist</code>	Liste mit offenen Fenstern anzeigen
<code>Ctrl+a k</code>	<code>kill</code>	Schließe aktuelles Fenster
<code>Ctrl+a \</code>	<code>quit</code>	Schließe alle Fenster und beende Screen-Instanz
<code>Ctrl+a d</code>	<code>detach</code>	Screen-Sitzung trennen
<code>Ctrl+a [</code>	<code>copy</code>	Kopiermodus zum Scrollen, Verlassen: <code><ESC></code>
<code>Ctrl+a ?</code>	<code>help</code>	Tastaturbelegung mit Befehlen zeigen



- Programme laufen auch bei getrennter Sitzung weiter
- Screen-Sitzung im aktuellen Terminal fortsetzen
(wird gegebenenfalls von anderem Terminal getrennt)

```
$ screen -dr
```

- Bei mehreren Screen-Sitzungen
 - Auflisten laufender Sitzungen

```
$ screen -ls
There are screens on:
 16656.pts-145.faii48f (20.04.2015 10:35:06) (Attached)
 16457.pts-123.faii48f (20.04.2015 10:34:59) (Attached)
2 Sockets in /var/run/screen/S-stengel.
```

- Bestimmte Sitzung fortsetzen

```
$ screen -dr 16457.pts-123.faii48f
```



- Screen-Konfiguration (.screenrc): Textdatei mit Befehlen

```
startup_message off # Keine Hilfeseite beim Start
defscrollback 1500 # Max. 1500 Zeilen puffern
# Fenster starten
screen ssh faui04e java -cp ../local/jgroups.jar ReplicationServer 1
screen ssh faui04f java -cp ../local/jgroups.jar ReplicationServer 2
screen ssh faui04g java -cp ../local/jgroups.jar ReplicationServer 3
```

- Konfiguration beim Starten von Screen laden

```
$ screen -c screen-config.txt
```

- Fernsteuern aus anderem Terminal oder Shell-Script

```
# screen -X <Screen-Befehl> ...
$ screen -X screen ssh faui04g
```

- Vollständige Dokumentation: `man screen`

