

Übungen zu Systemprogrammierung 1

Üo – Einführung

Sommersemester 2025

Luis Gerhorst, Thomas Preisner, Tobias Häberlein, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Universität
Technische Fakultät



- 0.1 Allgemeines
- 0.2 Organisatorisches
- 0.3 Linux-Kenntnisse
- 0.4 Versionsverwaltung mit Git
- 0.5 SP-Abgabesystem

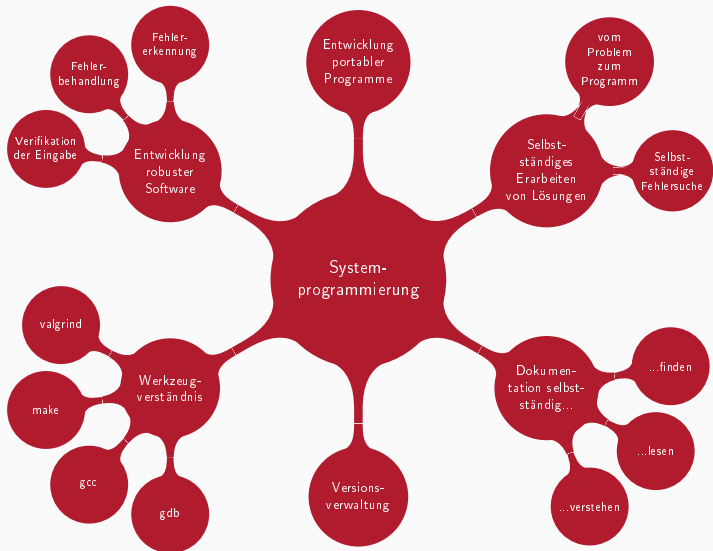
0.1 Allgemeines

0.2 Organisatorisches

0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

0.5 SP-Abgabesystem



Tafelübungen

- Vorstellung von Betriebssystemkonzepten und Werkzeugen
- Einführung in die Verwendung der Schnittstellen
- Erarbeiten eines kleinen Programmes (Demo)
- Besprechung der Abgaben und allgemeiner Fallstricke

Praktischer Teil – Aufgaben

- Arbeiten mit der Betriebssystemschnittstelle
- Fehlersuche und Fehlerbehebung
- Verwenden der vorgestellten Werkzeuge

Rechnerübungen

- Hilfestellung für Aufgaben



0.1 Allgemeines

0.2 Organisatorisches

0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

0.5 SP-Abgabesystem

- Ausgabe neuer Aufgaben in den Tafelübungen
 - Aufgabenstellung meist recht knapp
 - ⇒ nicht alles bis in letzte Detail spezifiziert
 - Gegebene Spezifikationen sind zwingend einzuhalten
- Selbstständiges Bearbeiten der Aufgaben
 - bei Problemen hilft z. B. ein Besuch in den Rechnerübungen
- Korrektur und Bewertung durch die Tutoren
 - Korrekturen werden elektronisch zur Verfügung gestellt
 - eigenes Ergebnis nach Login im *WAFFEL* einsehbar
 - Korrekturrichtlinien auf Webseite dokumentiert
- Übungspunkte können das Klausurergebnis verbessern
 - Abschreibtests
 - Vorstellen der eigenen Lösungen
 - **Anwesenheit in Besprechungsübungen für Bonuspunkte**
 - Notenbonus nur bei bestandener Klausur

- Bearbeitungszeitraum angegeben in Werktagen (Mo. bis Fr.)
 - Bearbeitungszeitraum beinhaltet Tag der Tafelübung
 - Feiertage und „Berg-Dienstag“ (nach Pfingsten) nicht enthalten
 - Abgabetermin kann per Skript erfragt werden
- plant mit **mindestens** 8–16 Stunden (in Worten: ein bis zwei **Tage**) Bearbeitungszeit pro Aufgabe
 - langer Bearbeitungszeitraum bietet Flexibilität bei der Arbeitsverteilung
 - Feedback über wirkliche Bearbeitungszeit erwünscht

- Mailingliste: `i4sp@cs.fau.de`
 - geht an alle Tutoren
 - Angelegenheiten, die nur die eigene Person/Gruppe betreffen
- Mailingliste: `i4sp-orga@cs.fau.de`
 - geht an die SP-Organisatoren
 - Fragen zur Organisation und zum Übungsbetrieb
- Rechnerübungen (siehe Homepage)
 - Hilfe bei konkreten Problemen (z. B. Quellcode kompiliert nicht)
 - **kein** Händchenhalten, während ihr die Tastatur bedient :)
- der korrigierende Tutor
 - Fragen zur Korrektur, vergessener Gruppenbonus
 - fälschlicherweise positiver Abschreibetest
- FSI-Forum: `https://fsi.cs.fau.de/forum/18`
 - inhaltliche Fragen zum Stoff oder den Aufgaben
 - allgemein alles, was auch für andere Teilnehmer interessant sein könnte

0.1 Allgemeines

0.2 Organisatorisches

0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

0.5 SP-Abgabesystem

- Grundkenntnisse zur Linux- und Shell-Nutzung werden vorausgesetzt
- Bei Bedarf:
 - Linux-Kurs der FSI Informatik (inkl. Aufzeichnung):
<https://fsi.cs.fau.de/linuxkurs>
 - Bei Problemen:
Fragen stellen in den ersten Wochen der Rechnerübungen
- CIP als Referenzsystem
 - Einführung zum CIP:
<https://wwwcip.cs.fau.de/documentation/overview.de.html>
 - Auflistung der Rechnerausstattung:
<https://wwwcip.cs.fau.de/cipPools/roomIndex.de.html>
 - Liste der SSH Host Keys:
<https://wwwcip.cs.fau.de/documentation/sshostkeys.de.html>



- Aufgeteilt in verschiedene *Sections*
 - 1 Kommandos
 - 2 Systemaufrufe
 - 3 Bibliotheksfunktionen
 - 3p POSIX-Funktionen
 - 5 Dateiformate (Spezielle Datenstrukturen etc.)
 - 7 Verschiedenes (z. B. Terminaltreiber, IP)
- Nach Möglichkeit Verwendung von POSIX-Manpages
 - Betriebssystemstandard für Unix-Systeme
 - Beinhalten sowohl Systemaufrufe als auch Bibliotheksfunktionen.
- **Achtung:** Manual-Pages unter Mac OS oft abweichend von Linux
⇒ CIP ist Referenzsystem!



- Angabe normalerweise mit *Section*: `printf(3p)`

```
$ man 3p printf
```

```
# man [section] begriff
```

- Suche nach *Sections*:

```
$ man -f <begriff>
```

- Suche nach Manual-Pages zu einem Stichwort:

```
$ man -k <stichwort>
```





0.1 Allgemeines

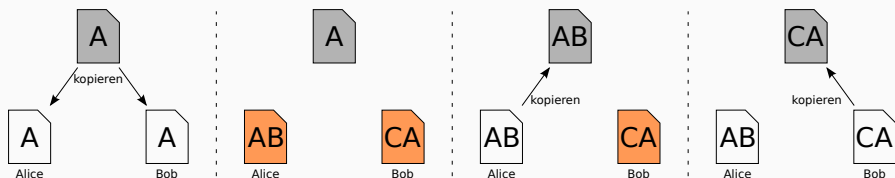
0.2 Organisatorisches

0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

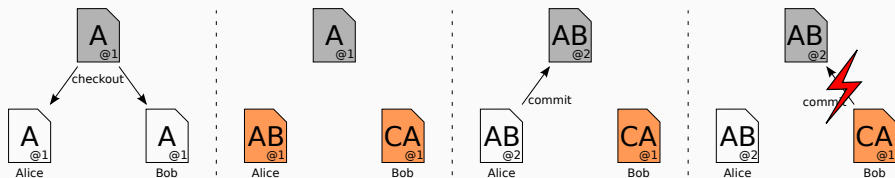
0.5 SP-Abgabesystem

- Gemeinsames Bearbeiten einer Datei kann zu Problemen führen:

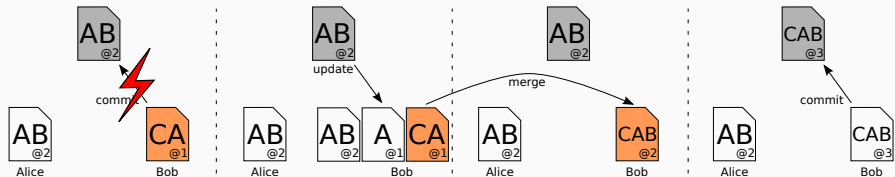


- Modifikationen werden nicht erkannt
- Änderungen von Alice gehen unbemerkt verloren

■ Versionsnummer zur Erkennung von Modifikationen



■ Entstandener Konflikt muss lokal gelöst werden





- Kommando: `git`
- Speichert Zusatzinformationen zu jeder Änderung
 - Name des Ändernden
 - Zeitpunkt
 - Kommentar
 - ...

⇒ identifiziert durch Commit-Hash
- Hilfe über Manpages (`man 1 git`) oder `git --help`
- SP-Abgabesystem verwendet Git

Tipp

Bei jedem Pushvorgang Nutzernamen + Passwort einzugeben? :-(
Stattdessen: SSH-Schlüssel einrichten!

→ <https://docs.gitlab.com/ee/user/ssh.html>



bfb76ad

main

1

```
~$ git clone gitlab.cs.fau.de:i4sp/  
ss23/test.git beispiel  
Cloning into 'beispiel'...
```





bfb76ad

main

1

Remote

```
~$ git clone gitlab.cs.fau.de:i4sp/
  ss23/test.git beispiel
Cloning into 'beispiel'...
~$ cd beispiel
```

Workspace

Repository



bfb76ad

main

1

Remote

```
~/beispiel $ touch README.md
```


README.md

Workspace

Repository



bfb76ad

main

1

Remote

```
~/beispiel $ touch README.md  
~/beispiel $ git add README.md
```

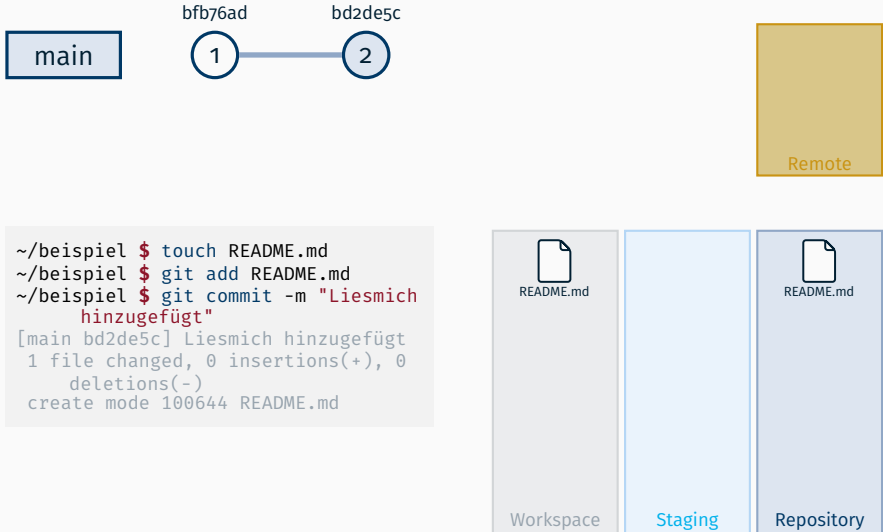

README.md

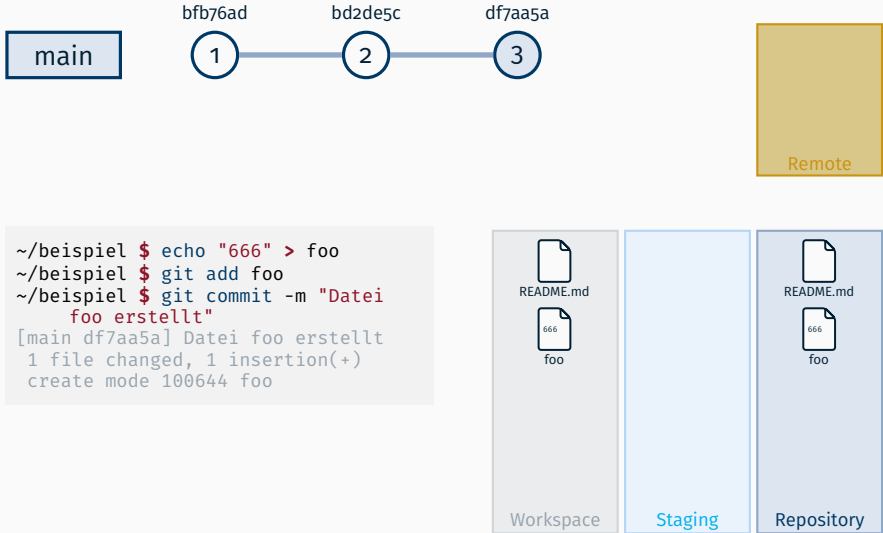
Workspace


README.md

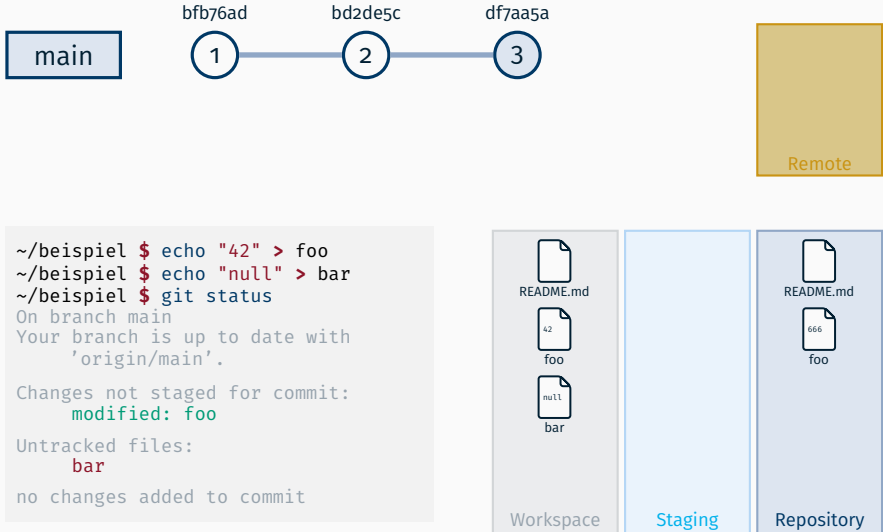
Staging

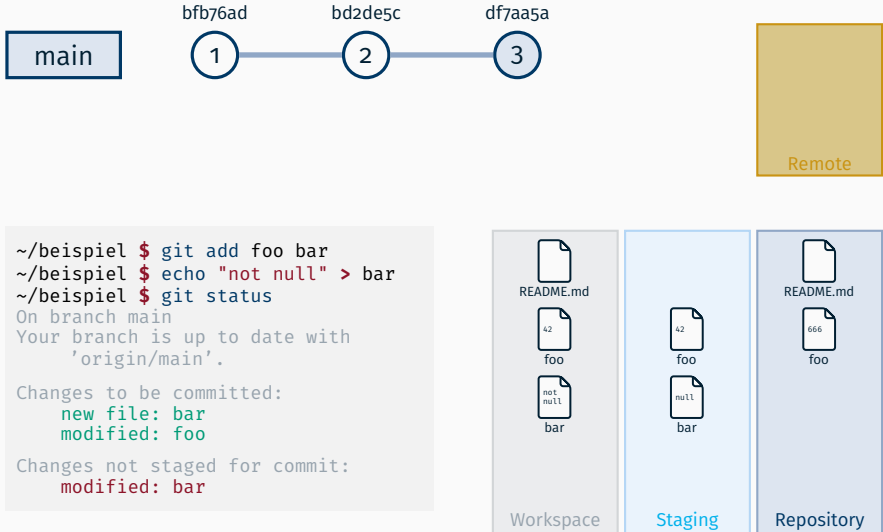
Repository

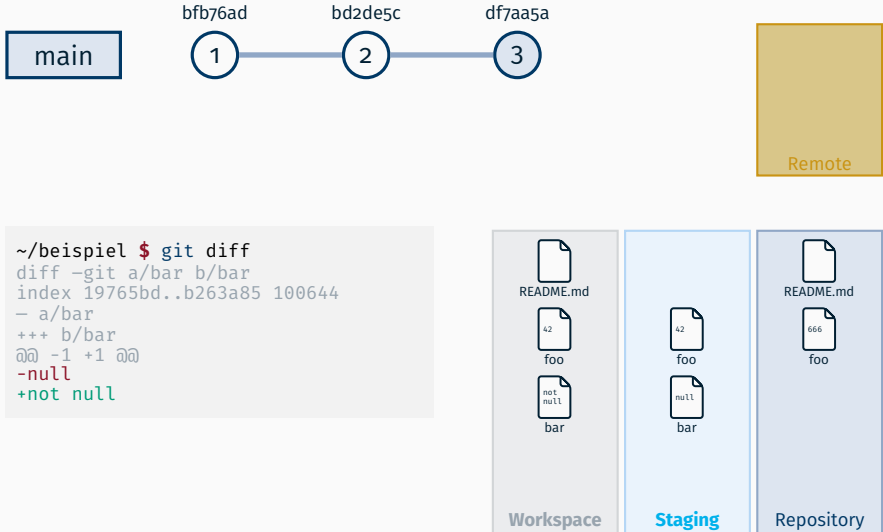


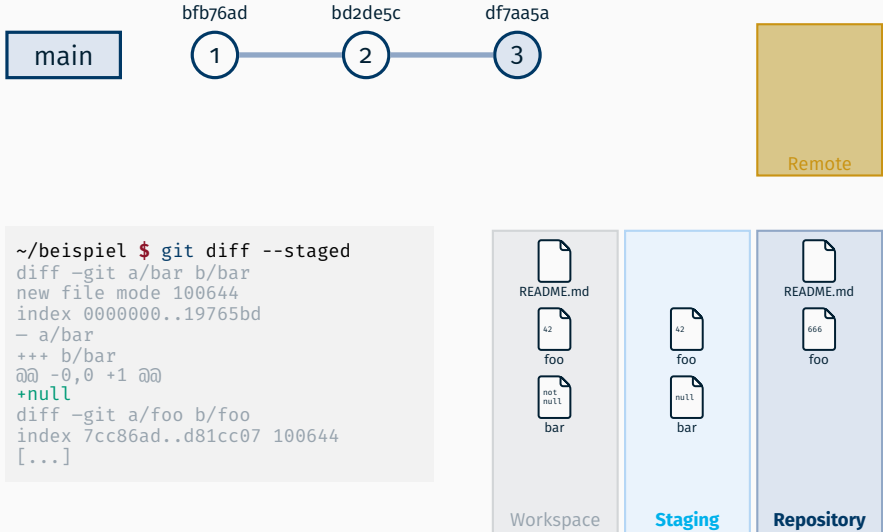


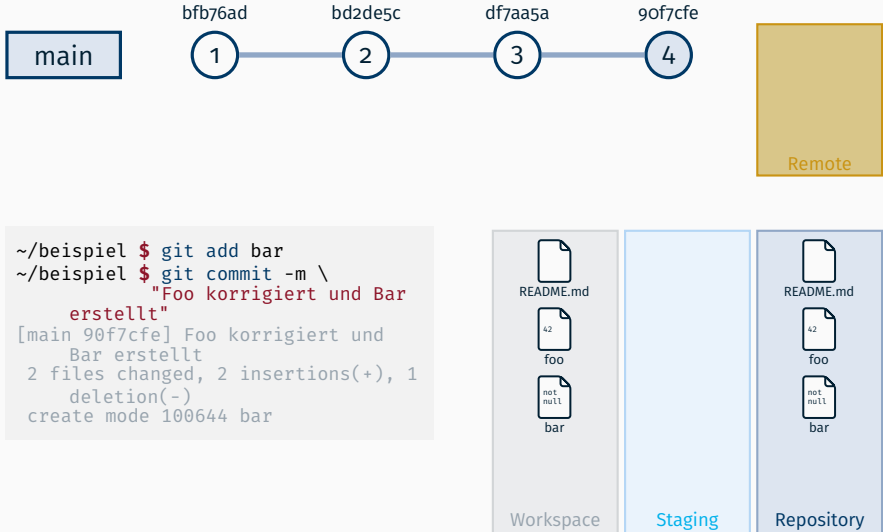
```
~/beispiel $ echo "666" > foo
~/beispiel $ git add foo
~/beispiel $ git commit -m "Datei
foo erstellt"
[main df7aa5a] Datei foo erstellt
1 file changed, 1 insertion(+)
create mode 100644 foo
```

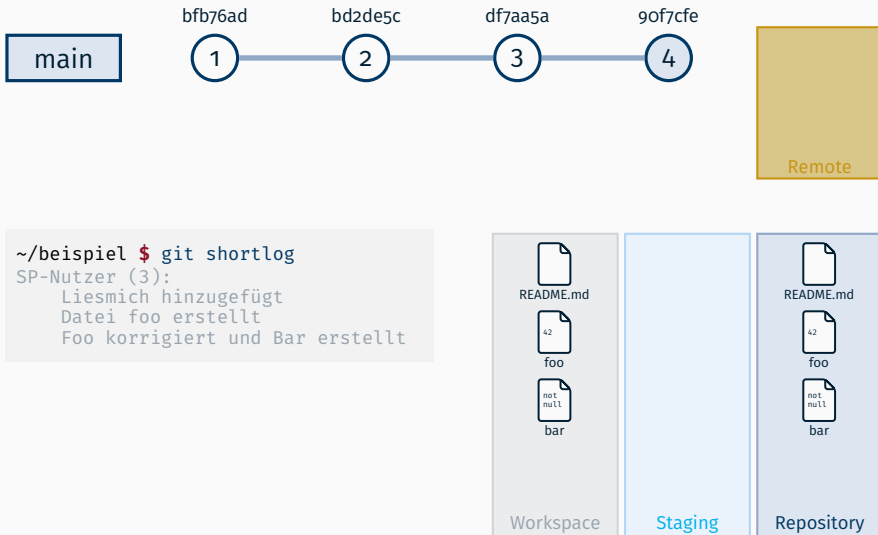


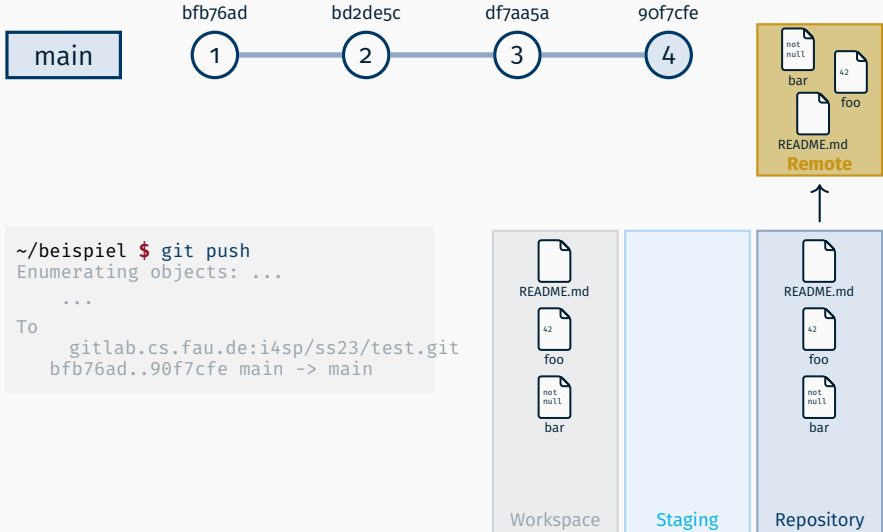


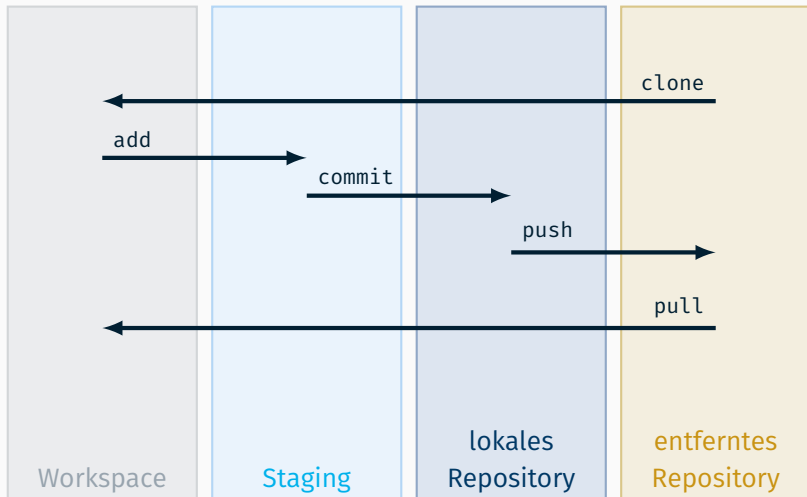














git add <file> Datei als Kandidat für nächsten *commit* markieren

git commit Änderungen versionieren

git diff unversionierte Änderungen anzeigen

git show neuste (versionierte) Änderungen anzeigen

git status Änderungen zum Vorgänger anzeigen

git log Historie anzeigen

git clone <url> initiales Kopieren von einer Quelle

git pull kurz für holen und zusammenfügen

git push in entfernte Quelle übertragen

man git-<cmd> Hilfe anzeigen, z.B. man git-add

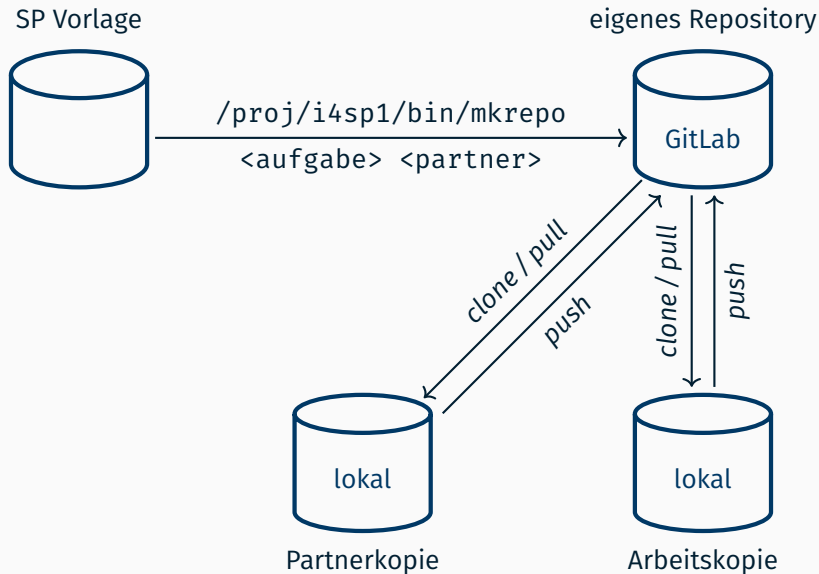
0.1 Allgemeines

0.2 Organisatorisches

0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

0.5 SP-Abgabesystem



- Für jede Aufgabe muss sich jede Übungsgruppe ein neues Vorgabe-Repository erstellen:
`/proj/i4sp1/bin/mkrepo <aufgabe> [<partner>]`
 - `aufgabe`: aktuell zu bearbeitende Aufgabe
 - bei Gruppenabgaben: Nutzerkennung des `partners`
- Repository enthält Vorgabe
(z.B. Programmgerüste und Beispieleingaben)
- Repository-Link:
`https://gitlab.cs.fau.de/i4sp/ss25/<TUEB>/<user>/<aufgabe>`
- Nutzung von Git zum Erstellen von lokalen Arbeitskopien

- Zum Abgabezeitpunkt wird der neueste Commit im Repository auf <https://gitlab.cs.fau.de/i4sp/ss25/...> eingesammelt
 - von dem Hauptbranch (main)
 - dieser Commit ist Abgabe der Übungsaufgabe
 - zu bewertende Änderungen stets mit **push** verbreiten
 - anderenfalls **keine Bewertung!**
 - bis zum Abgabezeitpunkt kann beliebig oft aktualisiert werden
- **Eigener** Abgabetermin kann per Skript erfragt werden

```
$ /proj/i4sp1/bin/deadline aufgabe0  
Dein Abgabezeitpunkt fuer die Aufgabe 0: epoche ist 01.01.1970  
um 17:30:00 Uhr
```



```
me@laptop ~ $ ssh student@cipterm0
# Repository anlegen
student@cipterm0 ~ $ /proj/i4sp1/bin/mkrepo epoche
# Lokale Kopie des Repositories anlegen
student@cipterm0 ~ $ git clone https://gitlab.cs.fau.de/i4sp/...
# Bearbeiten der Dateien
student@cipterm0 ~ $ cd aufgabe0
student@cipterm0 ~/epoche $ nano epoche.c
# Hinzufügen der Änderungen
student@cipterm0 ~/epoche $ git add epoche.c
student@cipterm0 ~/epoche $ git commit -m "add epoche.c"
# weitere Änderungen
student@cipterm0 ~/epoche $ micro epoche.c
student@cipterm0 ~/epoche $ git add epoche.c
student@cipterm0 ~/epoche $ git commit -m "bugfix in printf"
# Aktualisieren der Abgabe
student@cipterm0 ~/epoche $ git push
```

Selbst direkt ausprobieren! (1 Punkt)


aufgabe5

Project ID: 25215





Star
0

Fork
0


1 Commit

1 Branch

3 Tags

21 KB Project Storage

Topics:
 aufgabe5
rabenstein


init from 4fe7dd4e

Systemprogrammierung authored 2 weeks ago


a65f6391


main
aufgabe5 /
+

Find file
Web IDE
↓
Clone

- **Continuous Integration**
- Pipeline mit vordefinierten Checks/Tests
- Ausführung nach einem Push von neuen Commits
 - ⇒ Automatisiertes Testen
 - ⇒ Entdecken von Fehlern
 - ⇒ Überblick über Stand der Softwarefunktionalität



- Pipeline, bestehend aus Jobs und Stages
- Stages...
 - gruppieren mehrere Jobs
 - definieren, wann welche Jobs ausgeführt werden
 - z.B. build oder test Stage
- Jobs ...
 - definieren, was zu tun ist
 - werden von Runnern ausgeführt
- Tests ...
 - als Teilergebnisse von Jobs
 - liefern Rückmeldung, wie erfolgreich die Tests für den gegebenen Commit waren

Hands-On: Übersicht über mehrere Pipelines



i4sp > ... > rabenstein > aufgabe5 > Pipelines

All **1** Finished Branches Tags

Clear runner caches

CI lint

Run pipeline

Filter pipelines



Show Pipeline ID ▾

Status

Pipeline

Triggerer

Stages



🕒 00:00:11
📅 55 minutes ago

init from 4fe7dd4e
[#105574](#) main [a65f6391](#)
[latest](#)





failed

Pipeline #105574 triggered 57 minutes ago by  Eva Dengler

Retry

Delete

init from 4fe7dd4e



2 jobs for [main](#)

in 11 seconds and was queued for 2 seconds



latest



a65f6391 



No related merge requests found.

Pipeline

Needs

Jobs 2

Failed Jobs 1

Tests 8

build

test



build



test



Hands-On: Jobs einer Pipeline



failed Pipeline #105574 triggered 57 minutes ago by Eva Dengler

[Retry](#)[Delete](#)

init from 4fe7dd4e

🕒 2 jobs for [main](#)
in 11 seconds and was queued for 2 seconds

📄 [latest](#)

🔗 [a65f6391](#)

🔗 No related merge requests found.

Pipeline Needs **Jobs 2** Failed Jobs 1 Tests 8

Status	Job	Stage	Name	Duration	Coverage
failed	#897100 🔗 main → a65f6391	test	test	🕒 00:00:06 🕒 57 minutes ago	
passed	#897099 🔗 main → a65f6391	build	build	🕒 00:00:04 🕒 57 minutes ago	



Pipeline #105574 triggered 57 minutes ago by  Eva Dengler

Retry

Delete

init from 4fe7dd4e

🕒 2 jobs for [main](#)
in 11 seconds and was queued for 2 seconds

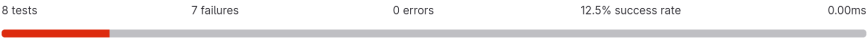
🚩 latest

🔗 [a65f6391](#) 

🔍 No related merge requests found.

Pipeline Needs Jobs 2 Failed Jobs 1 Tests 8

Summary



Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
test	0.00ms	7	0	0	1	8

Hands-On: Detailansicht der Tests einer Pipeline



< test

8 tests 7 failures 0 errors 12.5% success rate 0.00ms

Tests

Suite	Name	Filename	Status	Duration	Details
	calloc_calls_malloc		✗	0.00ms	View details
	calloc_is_zeroed		✗	0.00ms	View details
	calloc_too_much_fails		✗	0.00ms	View details
	free_valid_pointer_should_succeed		✗	0.00ms	View details
	malloc_max_size_succeeds		✗	0.00ms	View details
	realloc_calls_malloc		✗	0.00ms	View details
	realloc_too_much_fails		✗	0.00ms	View details
	malloc_too_much_fails		✓	0.00ms	View details



Scenario: student implementation creates the same output as the reference implementation

Given I have a directory “/tmp/creeper” with the following structure
... passed in 0.019s

name	type	
dir1/subdir1/	directory	
dir1/subdir1/super_source_file.c	file	
dir1/subdir1/secret_source_file.c	file	
dir1/subdir1/Makefile	file	
dir1/subdir2/	directory	
dir2/subdir2/	directory	

And I add “/tmp/creeper” to the list of paths ... passed in 0.000s

When I run creeper ... passed in 0.003s

And I run creeper's reference implementation ... passed in 0.003s

Then student and reference implementations create the same output

... failed in 0.000s

■ Merke: failed zeigt, welcher Schritt fehlgeschlagen ist

■ Keywords aus Behavior Driven Development (BDD)

⇒ Given: Vorbedingung

⇒ When: Ausführung

⇒ Then: Erwartetes Ergebnis

Testcases:

- Genaue Implementierung der Tests geheim
⇒ Informationen nur in der Namensbeschreibung und den Details
- Sollen als grobe Hinweise auf Fehler dienen
⇒ Angabe/Manpages genau lesen ;-)

HINWEIS

Die Jobs/Tests der Gitlab-CI ...

- sind lediglich als zusätzliche Hilfestellung gedacht
- haben **KEINEN** Anspruch auf Vollständigkeit / Korrektheit
 - Erfolgreiche Pipeline kann 0 Punkte bedeuten, fehlgeschlagene Pipeline aber auch volle Punktzahl