Übung Grundlagen der Betriebssysteme (GBS)

Tafelübung 1: Warmup

Daniel Lohmann, Björn Fiedler

Institute for Systems Engineering
System- und Rechnerarchitektur (SRA)

Wintersemester 2023

https://sra.uni-hannover.de/Lehre/WS23/V_GBS

Überblick: Tafelübung 1: Warmup

Tafelübung 1: Warmup

- Organisatorisches
- Semesterüberblick
- Ablauf und Inhalt der Übungen
- Allgemeine Anforderungen an Abgaben
- Abgabesystem
- SSH
- Abgabeprotokoll
- Linux Crashkurs
- Aufgabe 1 - Die Aufgabenstellung
- Hello World
- `lilo` - Verkettete Liste
- Funktionen für die Hausaufgabe
- C - Antipattern

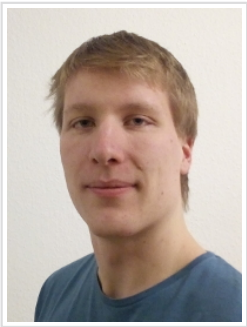
Organisatorisches

Beteiligte Personen

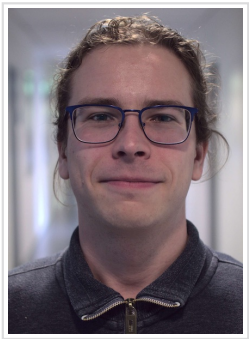
Vorlesung

Daniel Lohmann

Übung



Björn Fiedler



Tim-Marek Thomas

Tutoren Gruppenübungen

- Alena Radtke
- Tim Füchsel
- Nils Fuhler
- Matthias Wormann
- Paul Aumann
- Julian Helmsen

aufbau_der_veranstaltung 2023-10-13

Vorlesung

- Vermittlung der grundlegenden Konzepte und Prinzipien
  - Essenz der fundamentalen Betriebssystemkonzepte
  - Einordnung in den Kanon der Informatik
  - **Grundlage für die Übung**
- Termin: Freitags 10:00 - 11:30
- Ort: Multimedia-Hörsal: 3703 Raum 023
  und zeitgleich online via BBB
- Stud.IP: Vorlesung

aufbau_der_veranstaltung 2023-10-13

Vorlesungsskript

- Handout der Vorlesungsfolien wird über Stud.IP zur Verfügung gestellt
  - Als zusammenhängendes PDF-Dokument (Inhaltsverzeichnis, Querverweise)
  - Handout enthält (in geringem Umfang) zusätzliche Informationen
- Das Vorlesungsskript ist ein iteratives Projekt
  - Wird gelegentlich (vor der Vorlesung) aktualisiert
  - Es sind nicht direkt alle Kapitel verfügbar
  - Fehler gefunden? Mail an `gbs@sra.uni-hannover.de`

aufbau_der_veranstaltung 2023-10-13

Vorlesungsskript

- Handout der Vorlesungsfolien wird über Stud.IP zur Verfügung gestellt
  - Als zusammenhängendes PDF-Dokument (Inhaltsverzeichnis, Querverweise)
  - Handout enthält (in geringem Umfang) zusätzliche Informationen
- Das Vorlesungsskript ist ein iteratives Projekt
  - Wird gelegentlich (vor der Vorlesung) aktualisiert
  - Es sind nicht direkt alle Kapitel verfügbar
  - Fehler gefunden? Mail an `gbs@sra.uni-hannover.de`

Das Handout kann eine eigene Mitschrift nicht ersetzen!

- Ein Skript ist **kein** Lehrbuch.
- Folien sind nur die „Bildspur" – und auch davon nur ein Teil.

⇝ Skript als **Grundlage** für die eigene Mitschrift – **nicht Ersatz!**

aufbau_der_veranstaltung 2023-10-13

Übung

- Tafelübung
  - 2 parallele Partitionen
  - Termin A: Freitags 12:30 - 14:00, 14-tägig ab 13.10.2023
  - Termin B: Freitags 12:30 - 14:00, 14-tägig ab 20.10.2023
  - Ort: 3408 -220 (Hochhaus Keller, Appelstr 9A)
    und zeitgleich online via BBB
- Gruppenübungen
  - Termin: mehrere Termine, 14-täglich ab 16.10.2023
  - Ort: 3703 135 (SRA Seminarraum)
  - Zuordnung mittels Stud.IP Gruppen.
- Inhalte aus Tafel- und Gruppenübung
  - 6 Aufgaben mit Abgabe
  - Partnerarbeit
  - Bonus: bis zu 10% der Klausurpunkte zur Notenverbesserung
    - pro 10% der möglichen Übungspunkte 1% der möglichen Klausurpunkte als Bonus
    - dieser Bonus **hilft nicht** über die Bestehensgrenze
- Stud.IP: Übung

aufbau_der_veranstaltung 2023-10-13

Testat

- Termin: 12.01.2024
- Ort: Multimedia-Hörsal: 3703 Raum 023
- Zeit: Vorlesungstermin
- Bonus: bis zu **10% der Klausurpunkte**
  - Testat muss mit mindestens 50% der möglichen Testatpunkte „bestanden" sein
  - pro weitere 5% der möglichen Punkte gibt es 1 % der möglichen Klausurpunkte als Bonus
  - dieser Bonus **hilft auch** über die Bestehensgrenze der Klausur!

Voraussetzung für Wirkung auf Bestehensgrenze

Testat kann unter Klausurbedingungen in Präsenz durchgeführt werden.

aufbau_der_veranstaltung 2023-10-13

Klausur und Modulnote

- Klausur
  - Dauer: 90 Minuten, schriftlich
  - Termin: 1. März 2024 16:00 (ohne Gewähr)
- Modulnote (Basis: 100 Prozent der möglichen Klausurpunkte)

Klausurpunkte	0 — 100	Prozent
+ Testatbonus	0 — 10	Prozent
Summe > 50 Prozent? (Bestehensgrenze)		
Ja		Nein
+ Übungsbonus 0 — 10 Prozent		
Ergebnis	0 — 120	Prozent
(100 Prozent entsprechen der Note 1,0)		

Semesterüberblick

Wo bekomme ich Hilfe?

1. Übungspartner
2. Kommilitoninnen
3. Stud.IP Forum
4. Übungstutoren
5. Übungsleiter (Björn Fiedler) gbs@sra.uni-hannover.de
6. Dozent (Daniel Lohmann) gbs@sra.uni-hannover.de

Reihenfolge beachten!

- Fragen zum Inhalt werden **nur im Forum** beantwortet
  - Studentische Antworten sind hier **sehr gerne gesehen**
  - Es gibt keine falschen Fragen – und keine Nachteile durch Antworten
  - Trauen Sie sich auch zu antworten – man lernt eine Menge dabei!
- Wir moderieren, korrigieren und ergänzen nach Bedarf. **Nur Mut!**

aufbau_der_veranstaltung 2023-10-13

Semesterüberblick

Termine

Woche	Vorlesung	Übung A	Übung B
09.10.23	VL 1 Einführung	TÜ 1	
16.10.23	VL 2 Systemnahe Softwareentwicklung in C	GÜ 1	TÜ 1
23.10.23	VL 3 Grundlegende Konzepte	TÜ 2	GÜ 1
30.10.23	VL 4 Dateien und Dateisysteme	GÜ 2	TÜ 2
06.11.23	VL 5 Prozesse und Fäden	TÜ 3	GÜ 2
13.11.23	VL 6 Unterbrechungen, Systemaufrufe und Signale	GÜ 3	TÜ 3
20.11.23	VL 7 Prozesseinplanung	TÜ 4	GÜ 3
27.11.23	VL 8 Speicherbasierte Interaktion	GÜ 4	TÜ 4
04.12.23	VL 9 Betriebsmittelverwaltung, Synchronisation und Verklemmung	TÜ 5	GÜ 4
11.12.23	VL 10 Interprozesskommunikation	GÜ 5	TÜ 5
18.12.24	VL 11 Speicherorganisation	TÜ 6	GÜ 5
09.01.24	Testatklausur	GÜ 6	TÜ 6
16.01.24	VL 12 Speichervirtualisierung		GÜ 7
23.01.24		TÜ 7	TÜ 7
01.03.24	Klausur		

aufbau_der_veranstaltung 2023-10-13

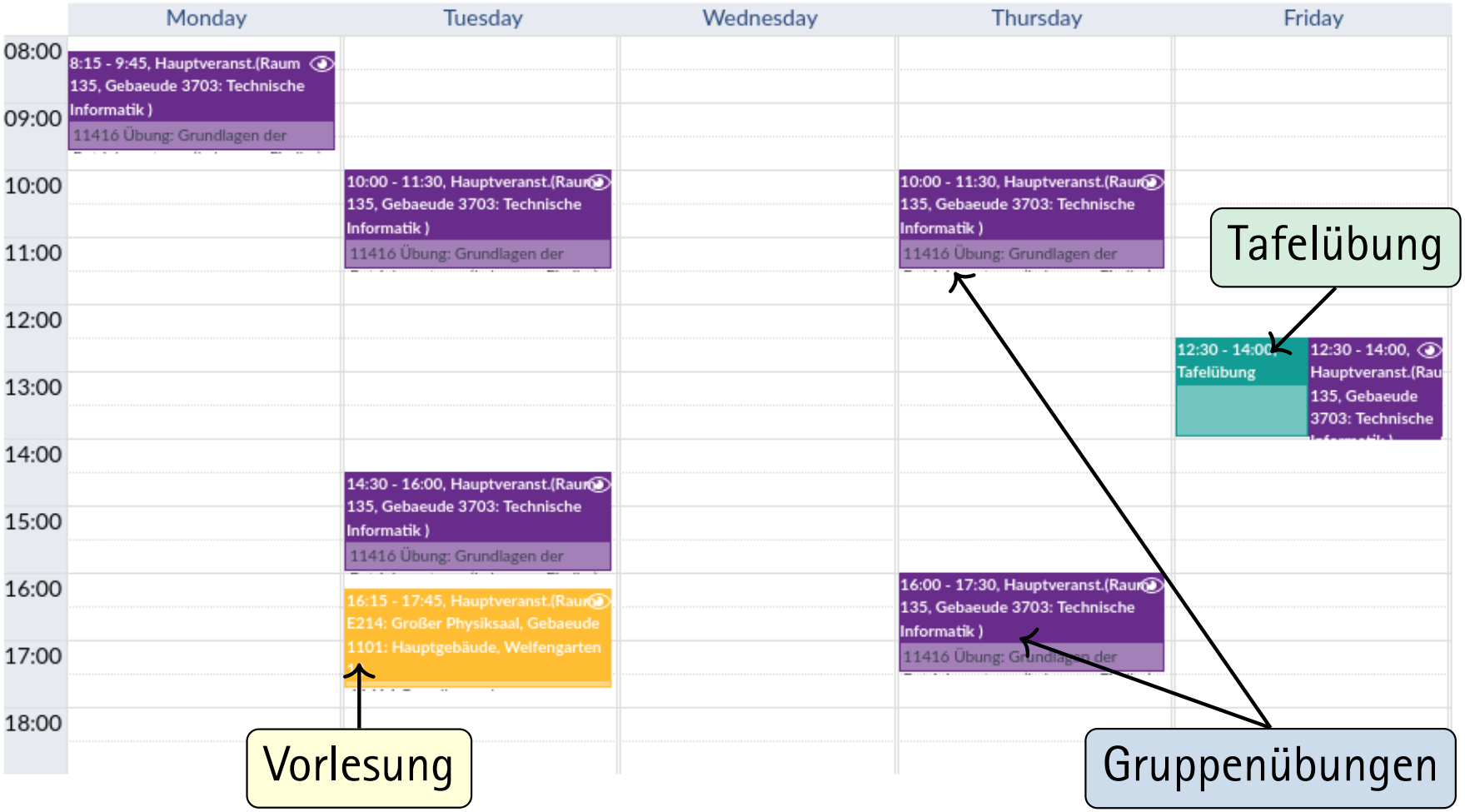
Ablauf und Inhalt der Übungen

Inhalt Tafel- und Gruppenübung

- Tafelübung
  - Vorstellung der neuen Aufgabe
  - Hinweise und Wissen zur Bearbeitung
  - Beispiellösung verwandter Aufgaben
  - Nachbesprechung Testatklausur
- Gruppenübung **mit Computer**
  - Hilfestellung bei der Bearbeitung
  - Fragemöglichkeit
  - Fachgdiskussion mit Kommilitonen und Tutoren

aufbau_der_veranstaltung 2023-10-13

Übungstermine



Wie wähle ich Gruppe und Partner?

- Partition A oder B?
  - Egal, sind identisch
  - Bitte gleichmäßig verteilen
- Welche Gruppenübung?
  - Wie es in deinen Stundenplan passt
  - Bitte gleichmäßig verteilen
- Partnerwahl?
  - Aus der selben Partition (Abgabetermin)
  - Praktischerweise aus der selben Gruppenübung
  - Kann für jede Abgabe variieren
  - Partnerbörse im Stud.IP Forum

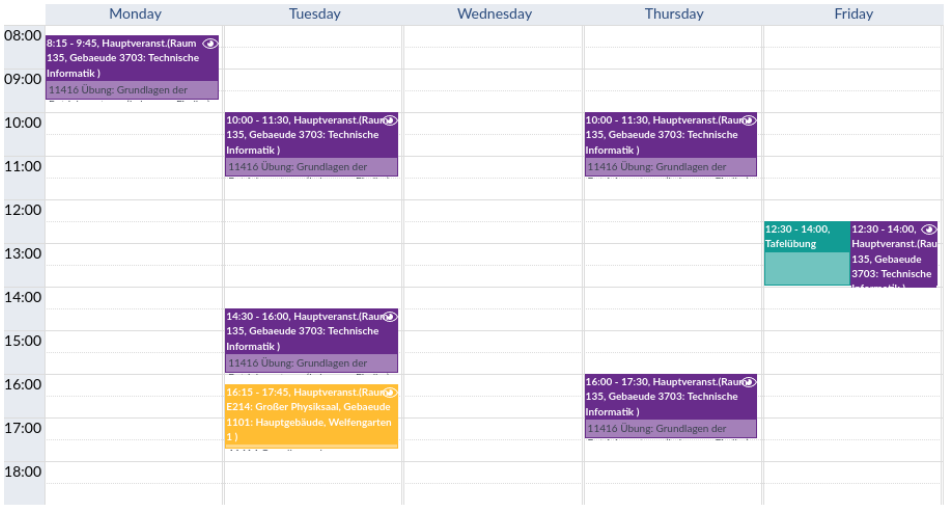
aufbau_der_veranstaltung 2023-10-13

Beispiel Gruppenwahl

- Schritt 1: Wahl der Partition A oder B

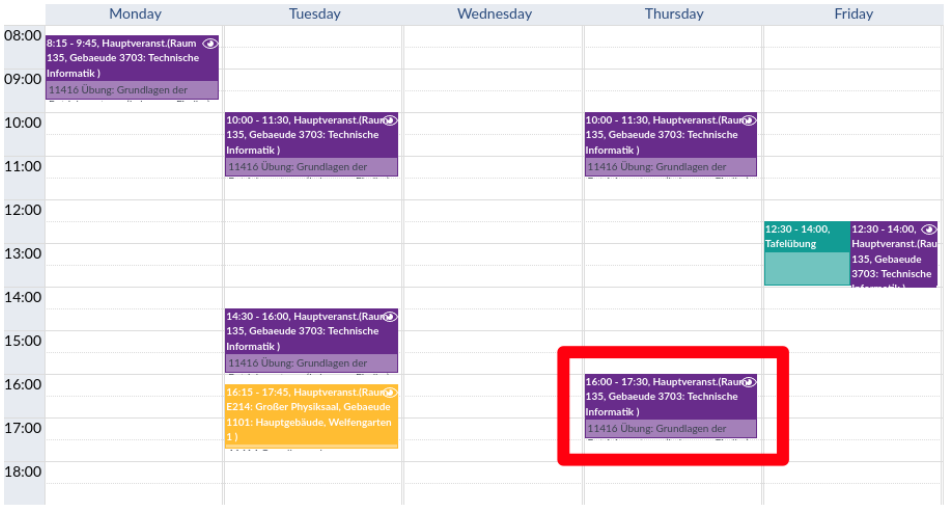
Beispiel Gruppenwahl

- Schritt 1: Wahl der Partition A oder B
- Schritt 2: Wahl der Übungsgruppe



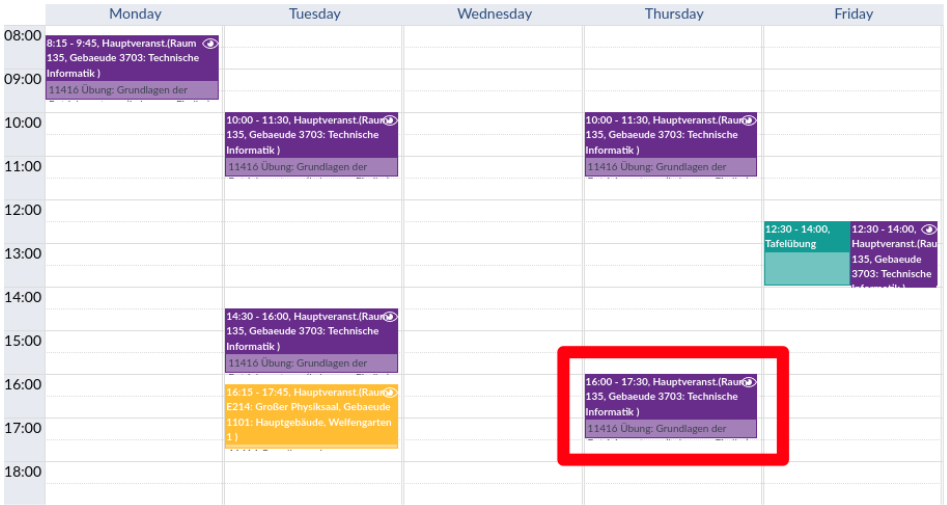
Beispiel Gruppenwahl

- Schritt 1: Wahl der Partition A oder B
- Schritt 2: Wahl der Übungsgruppe



Beispiel Gruppenwahl

- Schritt 1: Wahl der Partition A oder B
- Schritt 2: Wahl der Übungsgruppe



Resultierender Ablauf

- TÜ 1: 13.10.2023, 12:30
- GÜ 1: 19.10.2023, 16:00
- TÜ 2: 27.10.2023, 12:30
- GÜ 2: 02.11.2023, 16:00
- TÜ 3: 10.11.2023, 12:30
- GÜ 3: 16.11.2023, 16:00
- TÜ 4: 24.11.2023, 12:30
- GÜ 4: 30.11.2023, 16:00
- TÜ 5: 08.12.2023, 12:30
- GÜ 5: 14.12.2023, 16:00
- TÜ 6: 16.12.2024, 12:30
- GÜ 6: 12.01.2024, 16:00
- TÜ 7: 27.01.2024, 12:30

Abgabe der Hausaufgaben

- Partnerwahl:
  - `PARTNER`-Datei im Repo
- Abgabefrist:
  - Donnerstag vor der nächsten Tafelübung, 23:59 Uhr MEZ
- Wie oft darf ich abgeben?
  - Beliebig oft, die letzte fristgerechte Abgabe wird gewertet.
- Wie werde ich ein Paar?
  - Schau in deinen bestehenden Lerngruppen
  - Schau nach links und rechts
  - Geh in eine Gruppenübung
  - Frag im Forum
  - Frag **nicht** uns

Workload-Berechnung

- Standard-Workload für 5 LP-Veranstaltung (laut NHG): 125–150 h

12 VL	a	1,5 h	18	h
7 TÜ	a	1,5 h	10,5	h
6 GÜ	a	1,5 h	9	h
Testat			1	h
Klausur			1,5	h
		Kontaktzeit:	40	h

aufbau_der_veranstaltung 2023-10-13

Workload-Berechnung

- Standard-Workload für 5 LP-Veranstaltung (laut NHG): 125–150 h

12 VL	a	1,5 h	18 h
7 TÜ	a	1,5 h	10,5 h
6 GÜ	a	1,5 h	9 h
Testat			1 h
Klausur			1,5 h
		Kontaktzeit:	40 h

VL Vor-/Nachbereitung			6 h
6 Aufgaben	a	12 h	72 h
Testatvorbereitung			8 h
Klausurvorbereitung			14 h
		Selbststudium:	100 h

aufbau_der_veranstaltung 2023-10-13

Workload-Berechnung

- Standard-Workload für 5 LP-Veranstaltung (laut NHG): 125–150 h

12 VL	a	1,5 h	18 h
7 TÜ	a	1,5 h	10,5 h
6 GÜ	a	1,5 h	9 h
Testat			1 h
Klausur			1,5 h
		Kontaktzeit:	40 h

VL Vor-/Nachbereitung			6 h
6 Aufgaben	a	12 h	72 h
Testatvorbereitung			8 h
Klausurvorbereitung			14 h
		Selbststudium:	100 h

- Werte beziehen sich auf eine(n) **durchschnittliche(n) Studierende(n)**

aufbau_der_veranstaltung 2023-10-13

Workload-Berechnung

- Standard-Workload für 5 LP-Veranstaltung (laut NHG): 125–150 h

12 VL	a	1,5 h	18 h
7 TÜ	a	1,5 h	10,5 h
6 GÜ	a	1,5 h	9 h
Testat			1 h
Klausur			1,5 h
		Kontaktzeit:	40 h

VL Vor-/Nachbereitung			6 h
6 Aufgaben	a	12 h	72 h
Testatvorbereitung			8 h
Klausurvorbereitung			14 h
		Selbststudium:	100 h

- Werte beziehen sich auf eine(n) **durchschnittliche(n) Studierende(n)**
- Übungen sind **essentieller Teil** der Prüfungsvorbereitung
  - Etwa 90 von 140 Stunden Workload entfallen auf die Übung
  - Übungserfolg $\mapsto$ Klausurerfolg

aufbau_der_veranstaltung 2023-10-13

Workload-Berechnung

- Standard-Workload für 5 LP-Veranstaltung (laut NHG): 125–150 h

12 VL	a	1,5 h	18 h
7 TÜ	a	1,5 h	10,5 h
6 GÜ	a	1,5 h	9 h
Testat			1 h
Klausur			1,5 h
		Kontaktzeit:	40 h

VL Vor-/Nachbereitung			6 h
6 Aufgaben	a	12 h	72 h
Testatvorbereitung			8 h
Klausurvorbereitung			14 h
		Selbststudium:	100 h

- Werte beziehen sich auf eine(n) **durchschnittliche(n) Studierende(n)**
- Übungen sind **essentieller Teil** der Prüfungsvorbereitung
  - Etwa 90 von 140 Stunden Workload entfallen auf die Übung
  - Übungserfolg $\mapsto$ Klausurerfolg

↪ **Nehmen Sie die Übungsangebote wahr!**

aufbau_der_veranstaltung 2023-10-13

Welche Arten von Aufgaben erwarten mich?

- Theoriefragen
  - Abgabe in Textdatei `antworten.txt`
  - Beantwortung und Erklärung in **eigenen** Worten
  - Antworten bitte Stichpunktartig, keine Romane
  - Mögliche Kandidaten für Klausurfragen
    - Alte Klausuren sind auf der SRA Webseite verfügbar.

Welche Arten von Aufgaben erwarten mich?

- Theoriefragen
  - Abgabe in Textdatei `antworten.txt`
  - Beantwortung und Erklärung in **eigenen** Worten
  - Antworten bitte Stichpunktartig, keine Romane
  - Mögliche Kandidaten für Klausurfragen
    - Alte Klausuren sind auf der SRA Webseite verfügbar.

- Programmieraufgaben
  - Abgabe als Quellcode
  - Interaktion mit den Betriebssystem Schnitttstellen
  - Programmieren in C
  - Programmieren gegen POSIX/Linux

Woher bekomme ich einen Linux Computer?

- Verfügbare Linux Computer
  - Dein eigener Laptop
  - Der GBS-Server
  - Die GBS-VM
- Abgabekriterium für die Übung:
  - Es muss auf dem GBS-Server funktionieren
  - mitgelieferte Testfälle: `make test`

Allgemeine Anforderungen an Abgaben

Was muss mein Programm können?

- Welche Funktionen soll ich implementieren?
  - Siehe Aufgabenbeschreibung
  - Manche Teile sind als optional gekennzeichnet
- Wie kann ich die Funktionen prüfen?
  - In den Vorgaben sind Tests mit enthalten. Das Makefile enthält dazu passende Regeln:
    `make test`
- Muss ich auch die Fragen beantworten? Die Testfälle decken das nicht ab.
  - Ja auch die, die Tutoren bewerten die Antworten.

Welche Anforderungen muss der Code erfüllen?

- C-Sprachumfang konform zu ANSI C11
- Betriebssystemschnittstelle konform zu SUSv4

Welche Anforderungen muss der Code erfüllen?

- C-Sprachumfang konform zu ANSI C11
- Betriebssystemschnittstelle konform zu SUSv4
- **warnungs-** und **fehlerfrei** auf dem gbs-Server mit folgenden `gcc`-Optionen übersetzen:

  ```
  -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror
  ```

  - `-std=c11 -pedantic` erlauben nur ANSI-C11-konformen C-Quellcode
  - `-D_XOPEN_SOURCE=700` erlaubt nur SUSv4-konforme Betriebssystemaufrufe

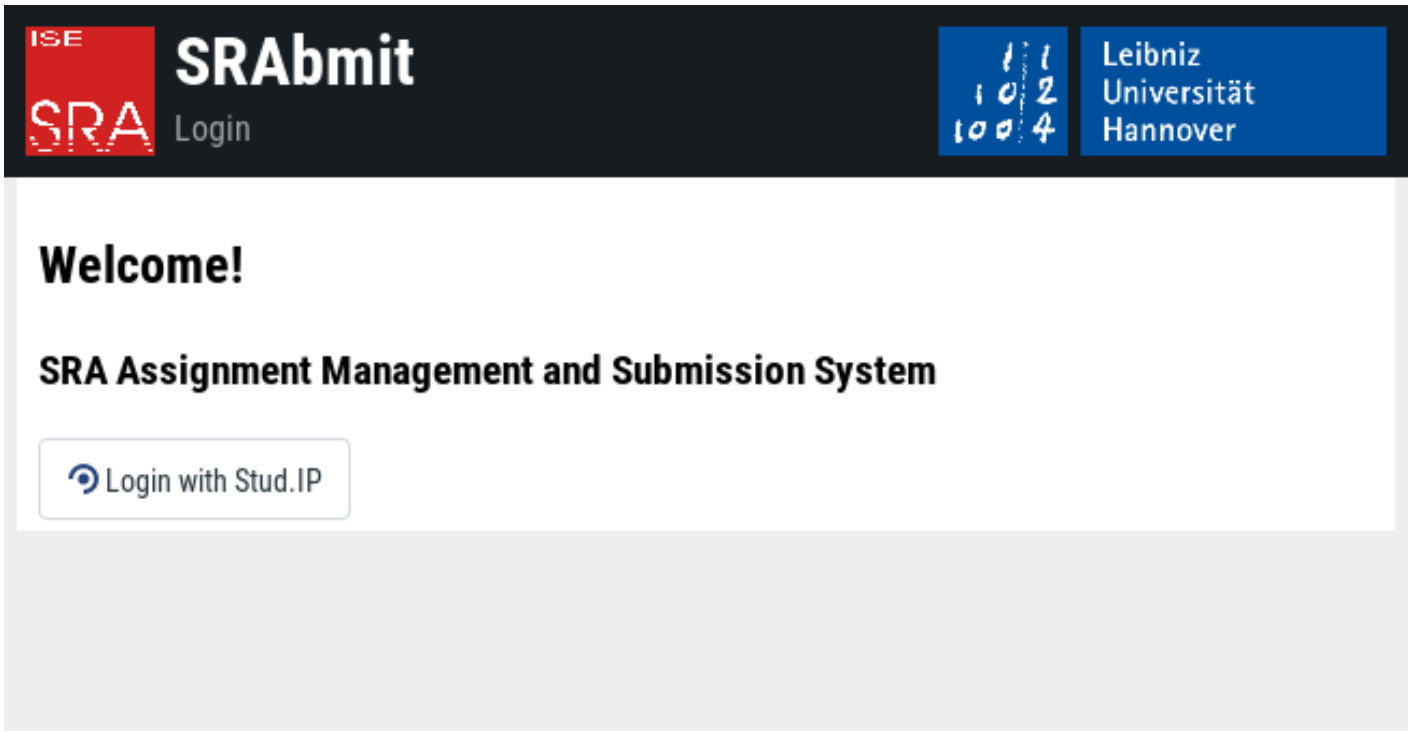
Abgabesystem

SRAbmit - Das Abgabesystem

- Zugang
  - Login über Stud.IP
  - Web: `https://gbs.sra.uni-hannover.de`
  - Shell: `ssh <username>@gbs.sra.uni-hannover.de`
- Funktionen
  - SSH Zugangsdaten
  - Status der Abgaben
  - Status des Klausurbonus

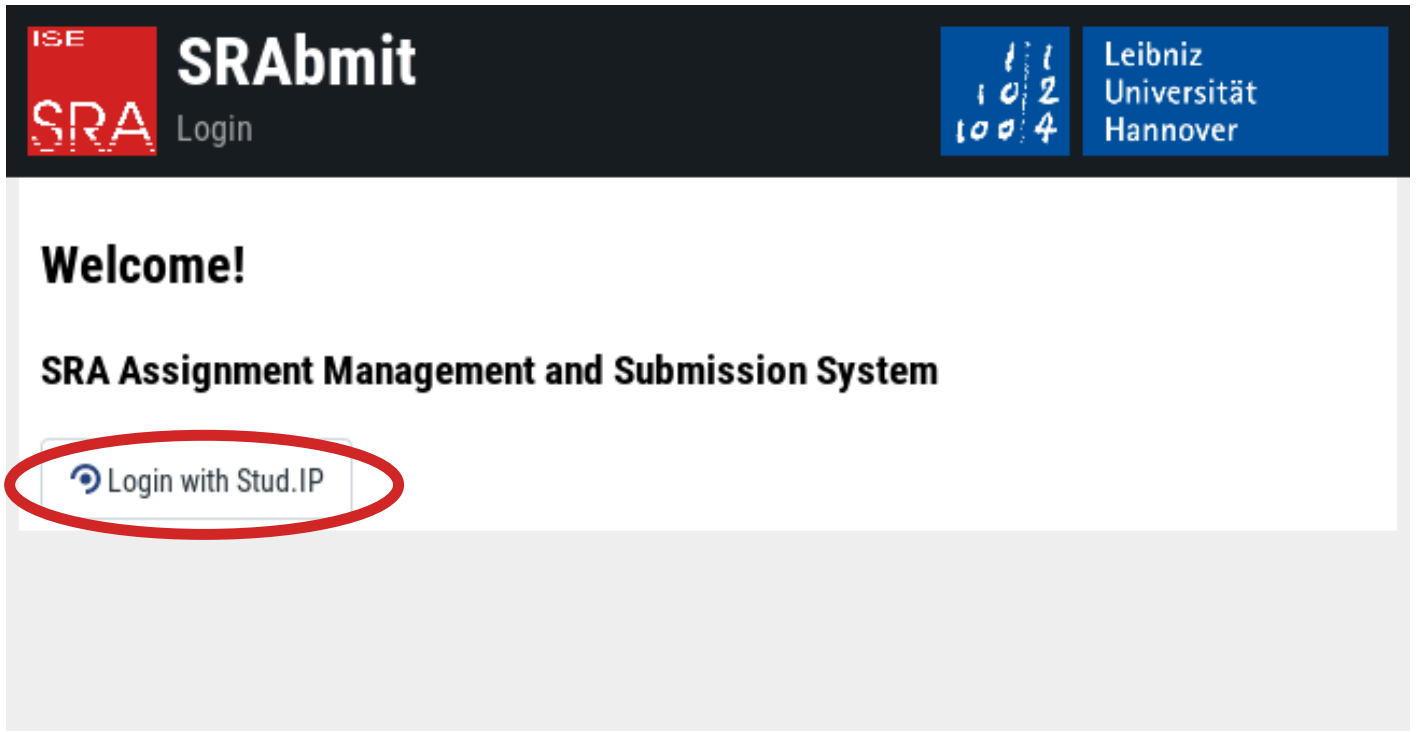
Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`



Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden

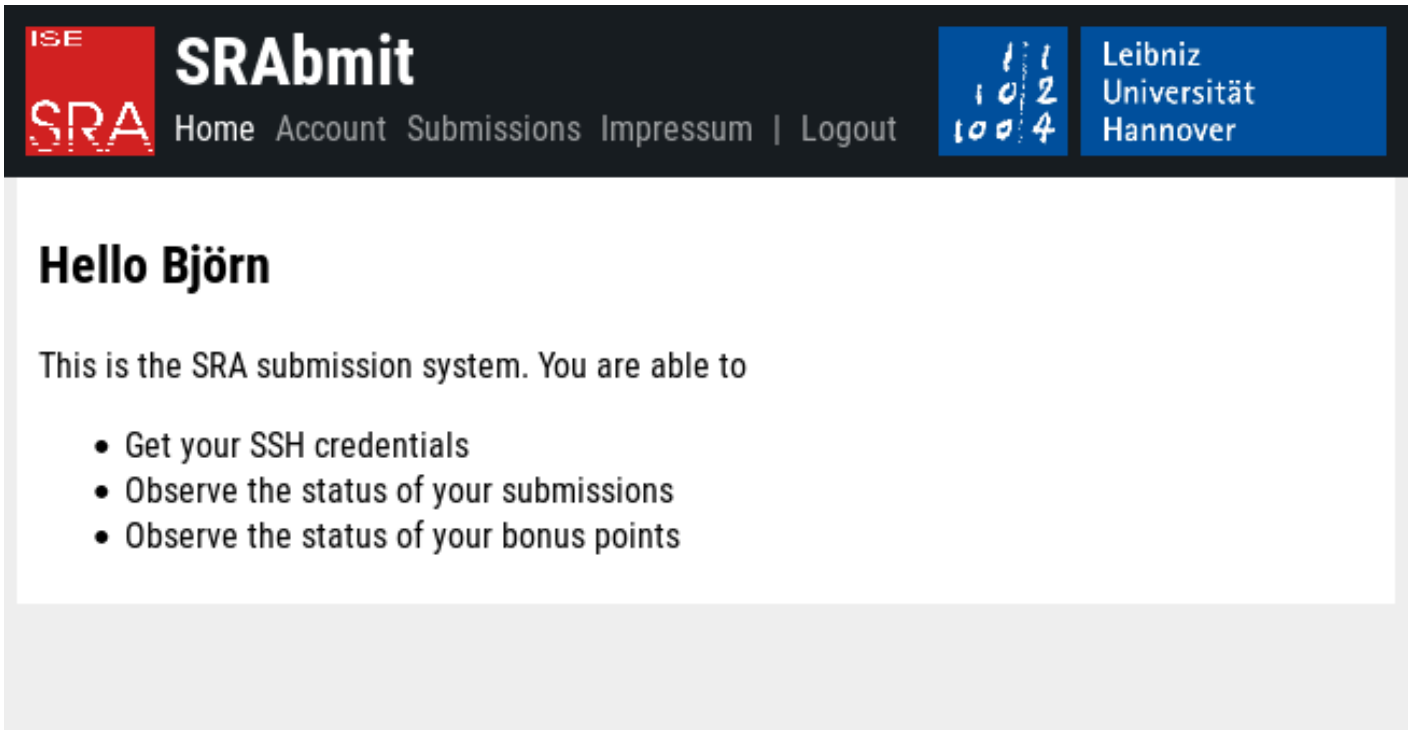


Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden

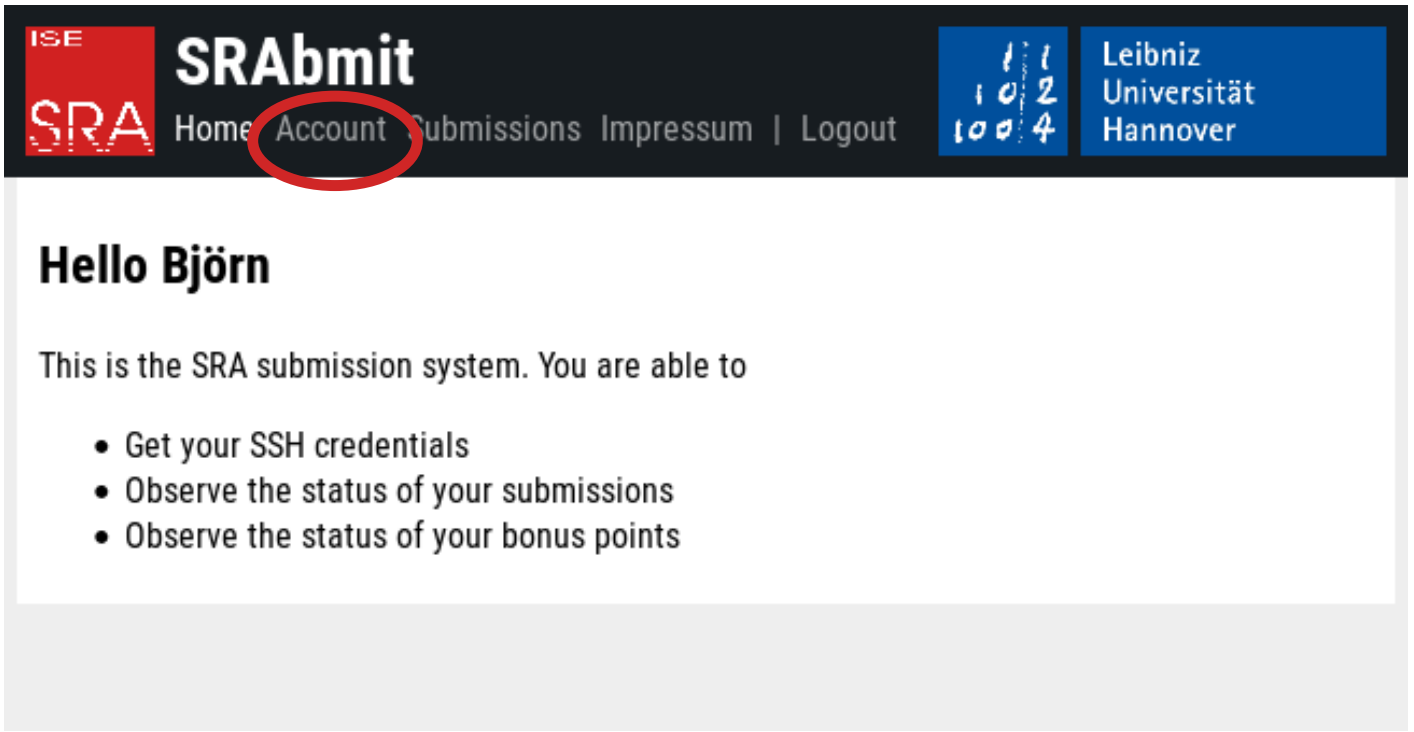
Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden



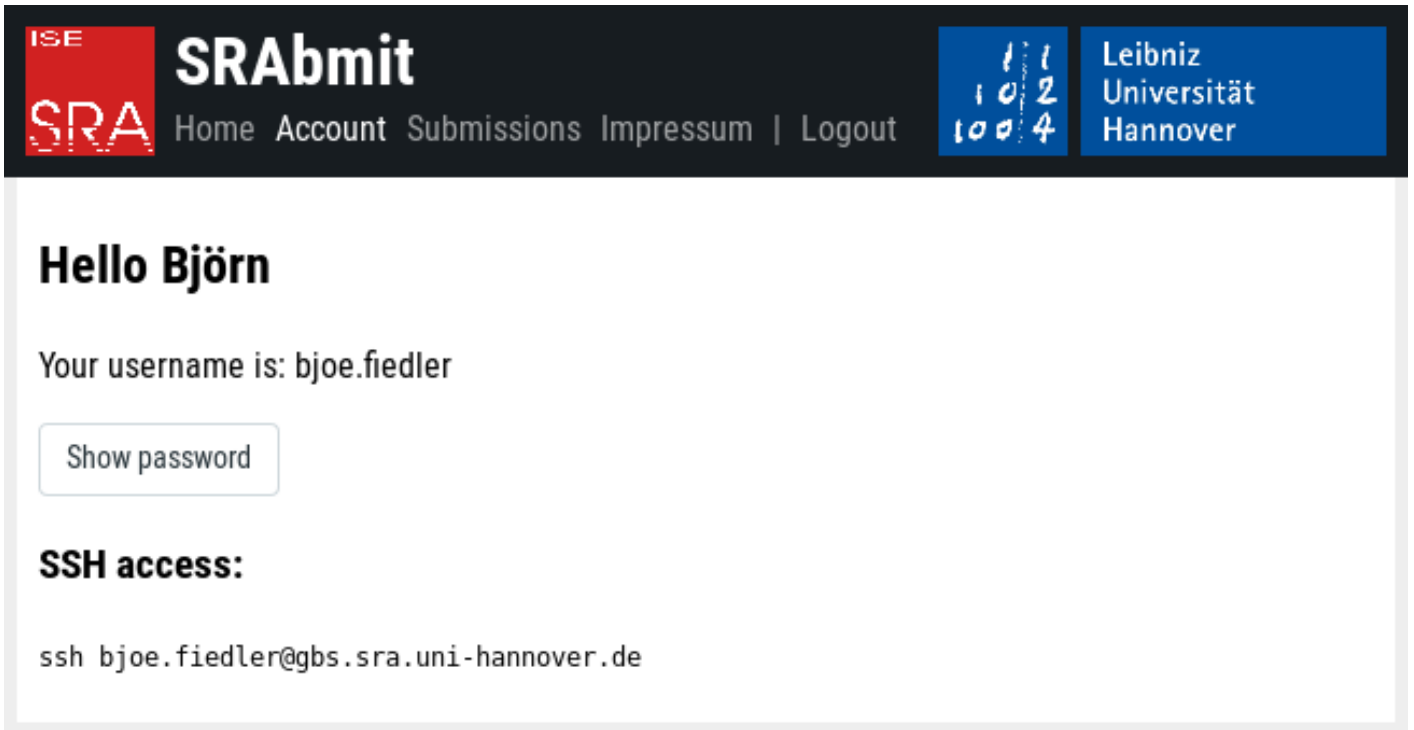
Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden
3. Benutzername anzeigen



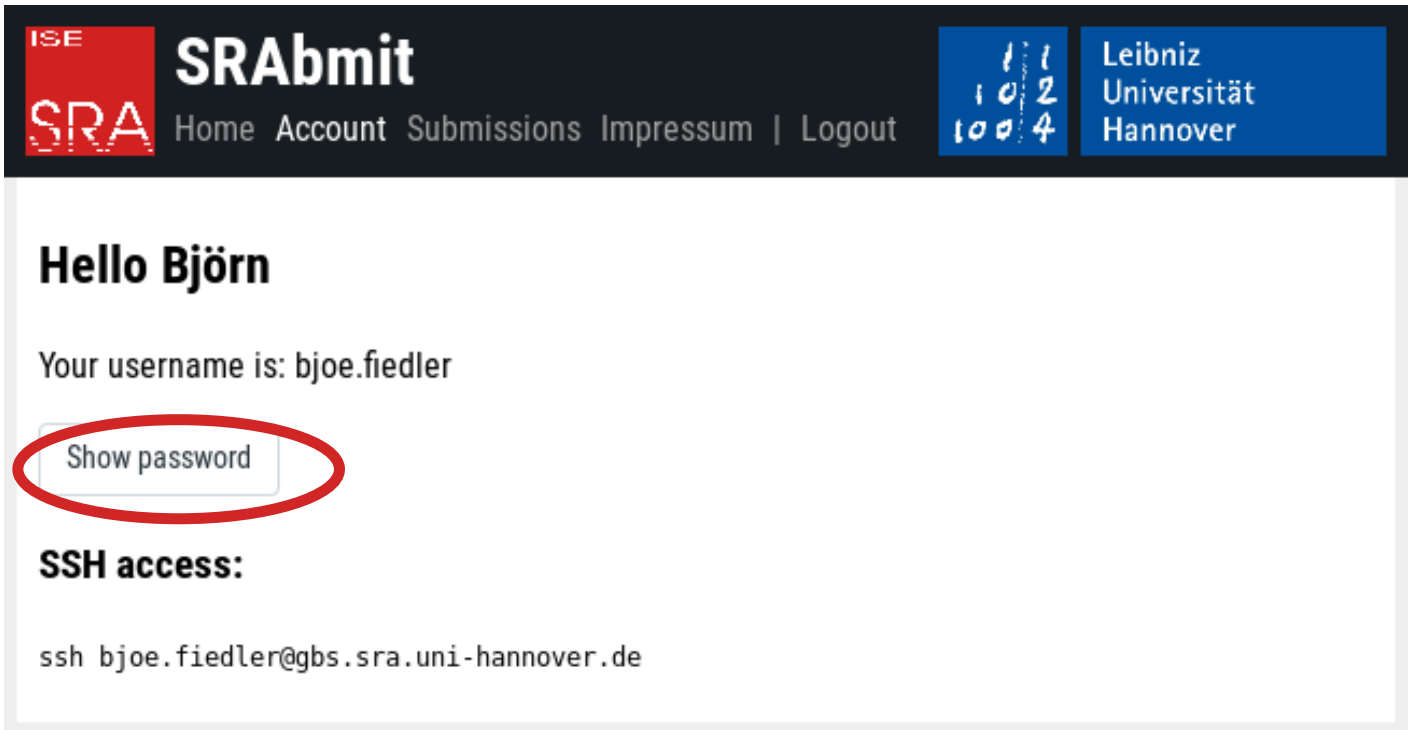
Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden
3. Benutzername anzeigen



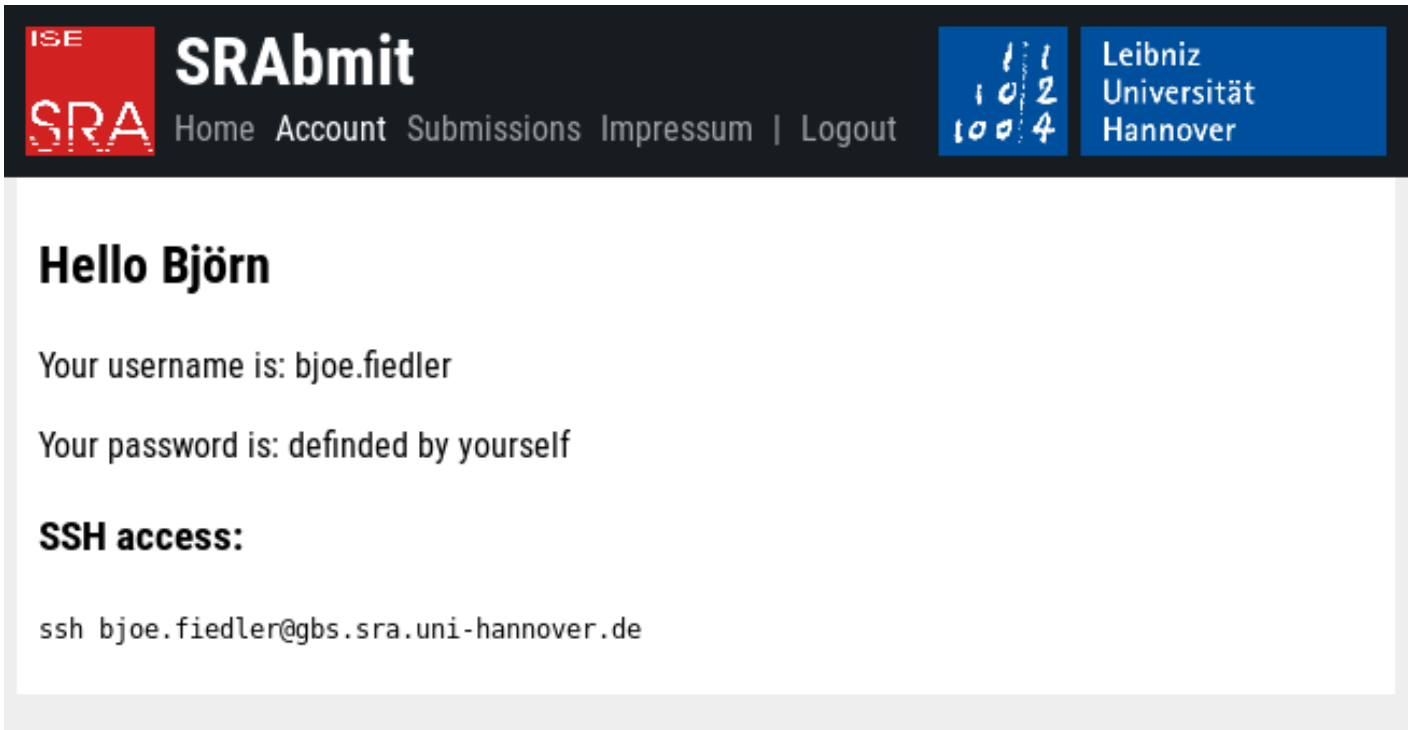
Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden
3. Benutzername anzeigen
4. Passwort anzeigen



Account anlegen im Abgabesystem

1. Webseite aufrufen: `https://gbs.sra.uni-hannover.de`
2. Über Stud.IP anmelden
3. Benutzername anzeigen
4. Passwort anzeigen



Tafelübung 1: Warmup

SSH

SSH Zugang

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem entfernten Rechner ausgeführt.

SSH Zugang

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem entfernten Rechner ausgeführt.

Verbinden mit dem GBS-Rechner

```
ssh <user>@gbs.sra.uni-hannover.de
```

- `<user>` ist dein Benutzername.
- `gbs.sra.uni-hannover.de` ist der entfernte Rechner.
- Zugangsdaten aus dem SRAbmit

Erster Login

```
$ ssh <user>@gbs.sra.uni-hannover.de
The authenticity of host 'gbs.sra.uni-hannover.de (130.75.33.185)'
    can't be established.
ED25519 key fingerprint is SHA256:9WwQeygESkw+nwmWhh/WklnN03ah3txzEYZa
Are you sure you want to continue connecting (yes/no)?
```

SSH-Fingerprint

- eindeutiger Fingerabdruck für jeden Rechner
- garantiert, dass man mit dem richtigen Rechner redet
- wird in `~/.ssh/known_hosts` gespeichert
- alle bekannten Fingerprints stehen in `/etc/ssh/ssh_known_hosts`

SSH Schlüssel

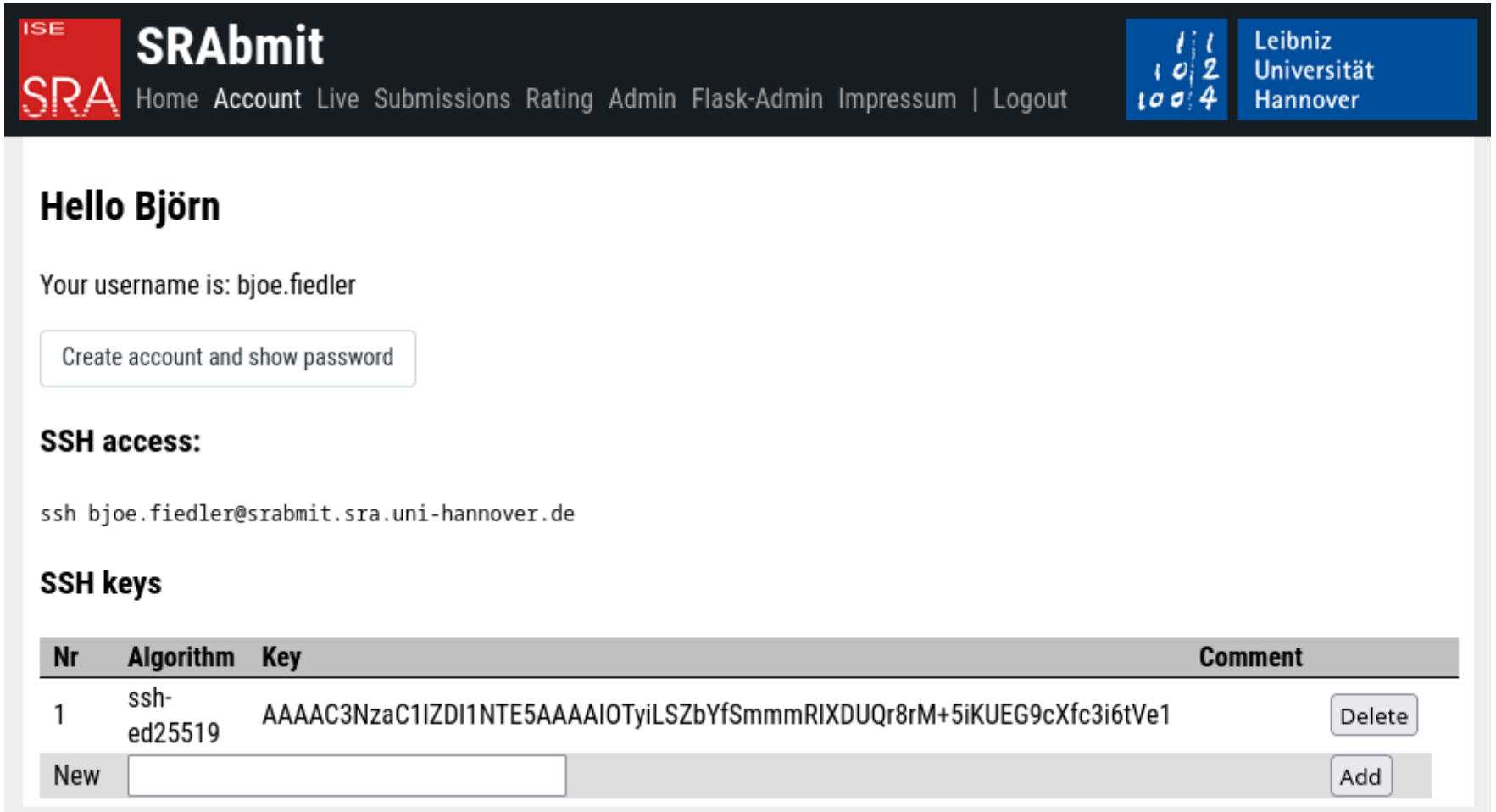
Eigenschaften

- Asymetrische Kryptographie
- Schlüssle**paar**: öffentlicher und privater Schlüssel

```
user@mymachine:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
The key fingerprint is:
SHA256:SMuCWAnwSrLqUny24iGKVSppCT79euc6sHBr5sbYyx4 user@mymachine
The key's randomart image is:
+--[ED25519 256]--+
|+                |
| o .             |
|..+   .          |
|o= . o o         |
|=.. o + S        |
|+o==o.           |
|=*XE+.           |
|*=+X+o .         |
|+oBB+o=.         |
+----[SHA256]-----+
user@mymachine:~$ cat testkey.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPO7lSMabfQq9eKvlTZTMYyYl1QEMhXsbx9Y1uiv2m7D user@mymachine
```

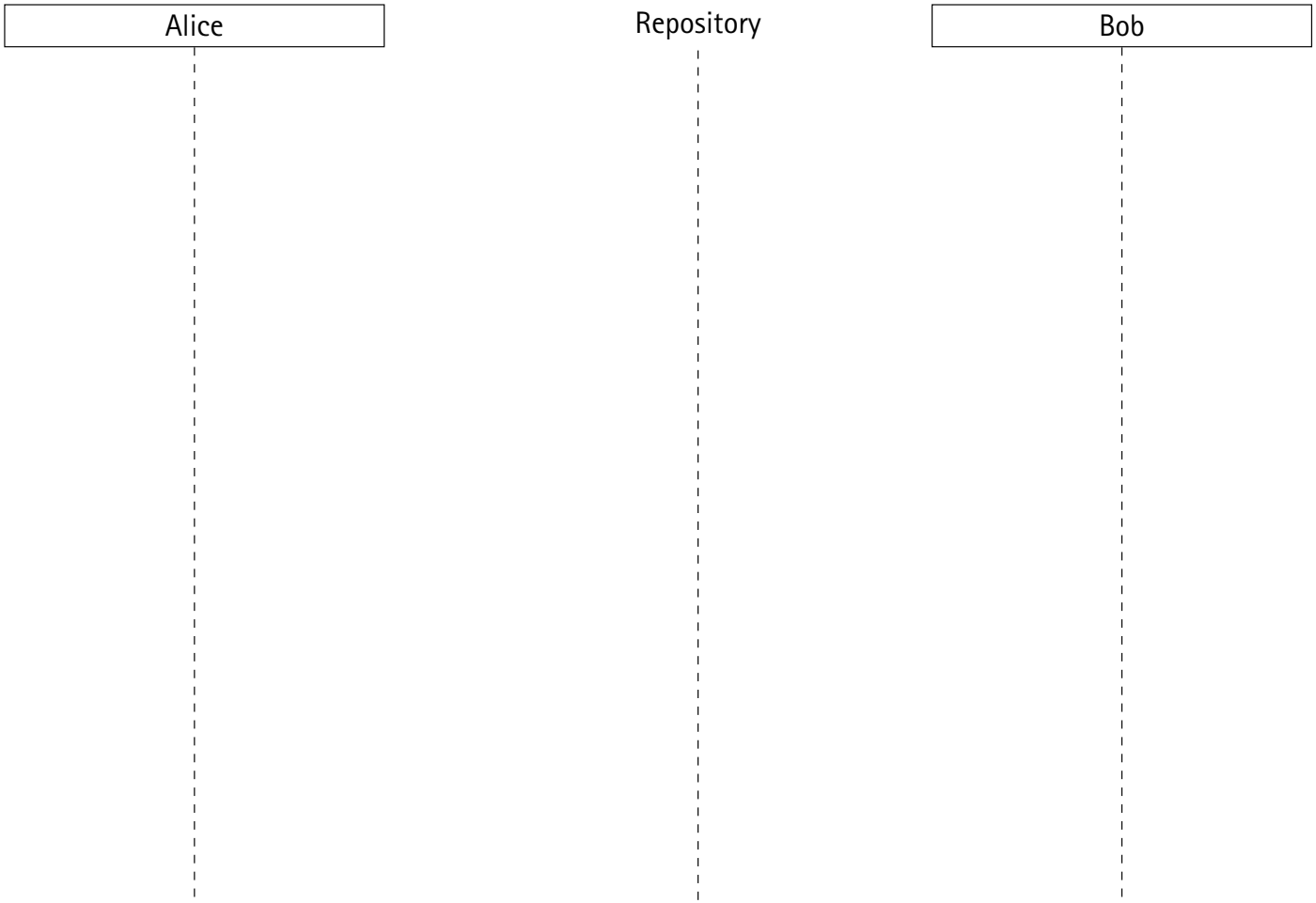
Nur einmal notwendig!

SSH Schlüssel in `srabmit` eintragen

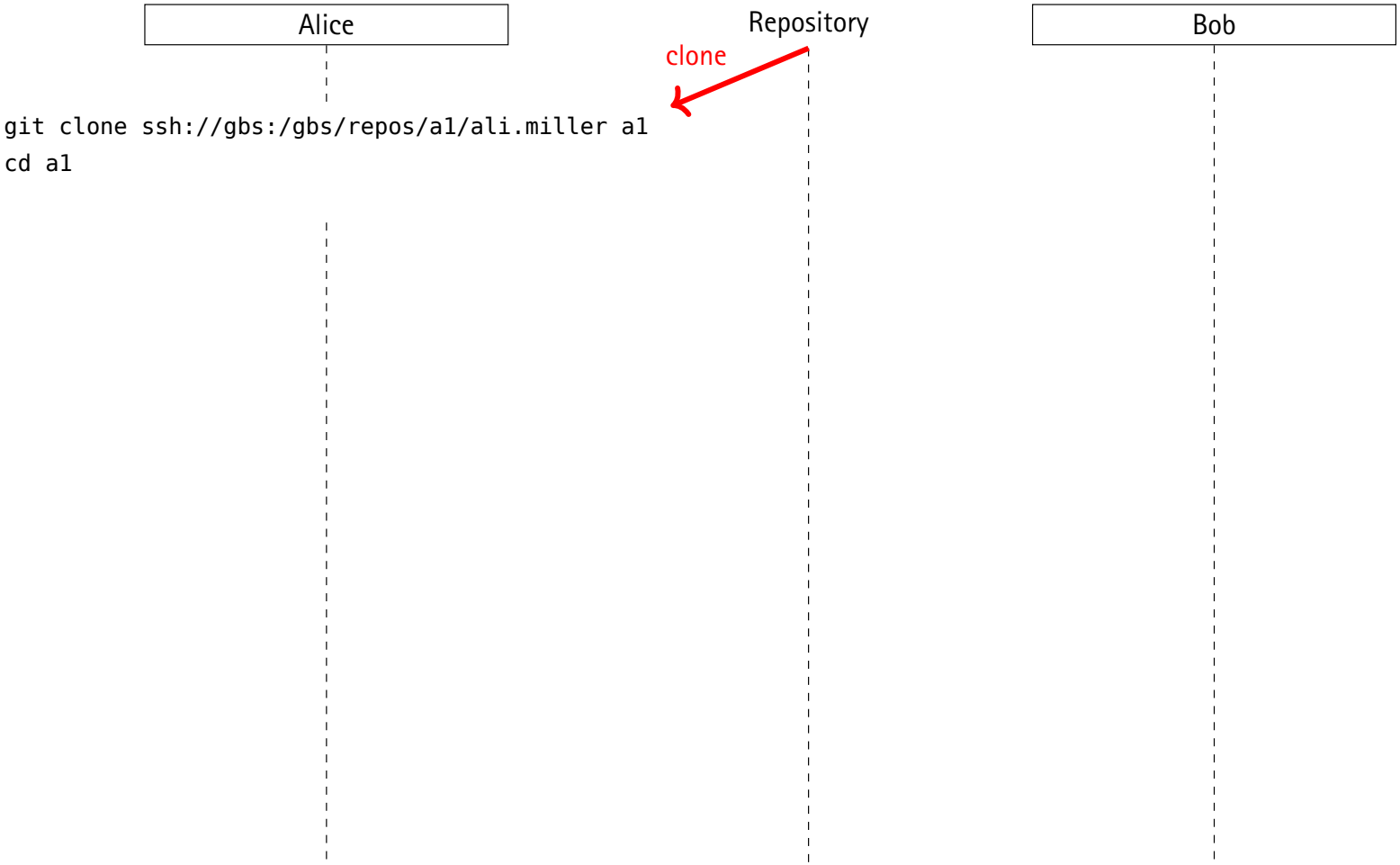


Abgabeprotokoll

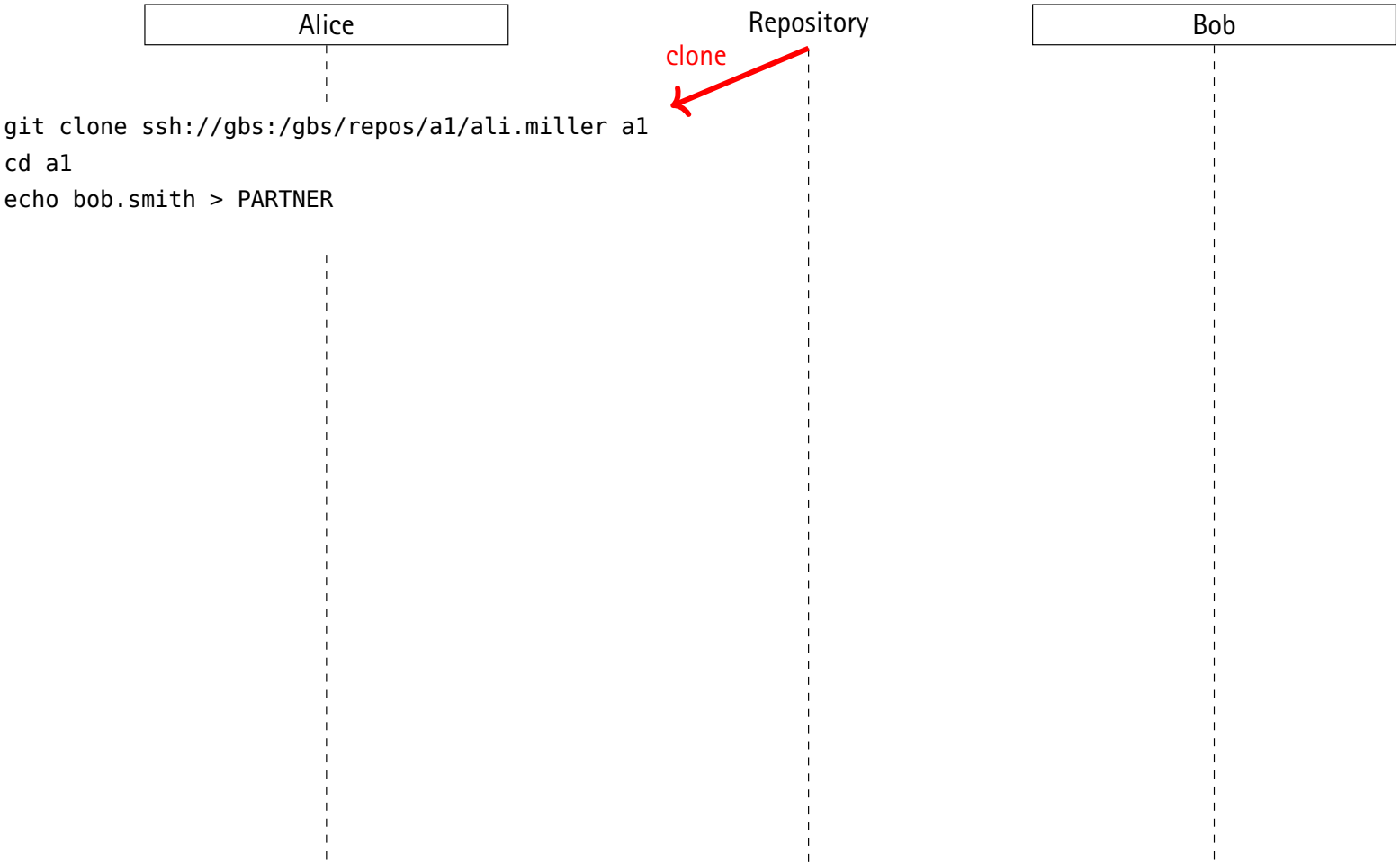
Abgabeprotokoll



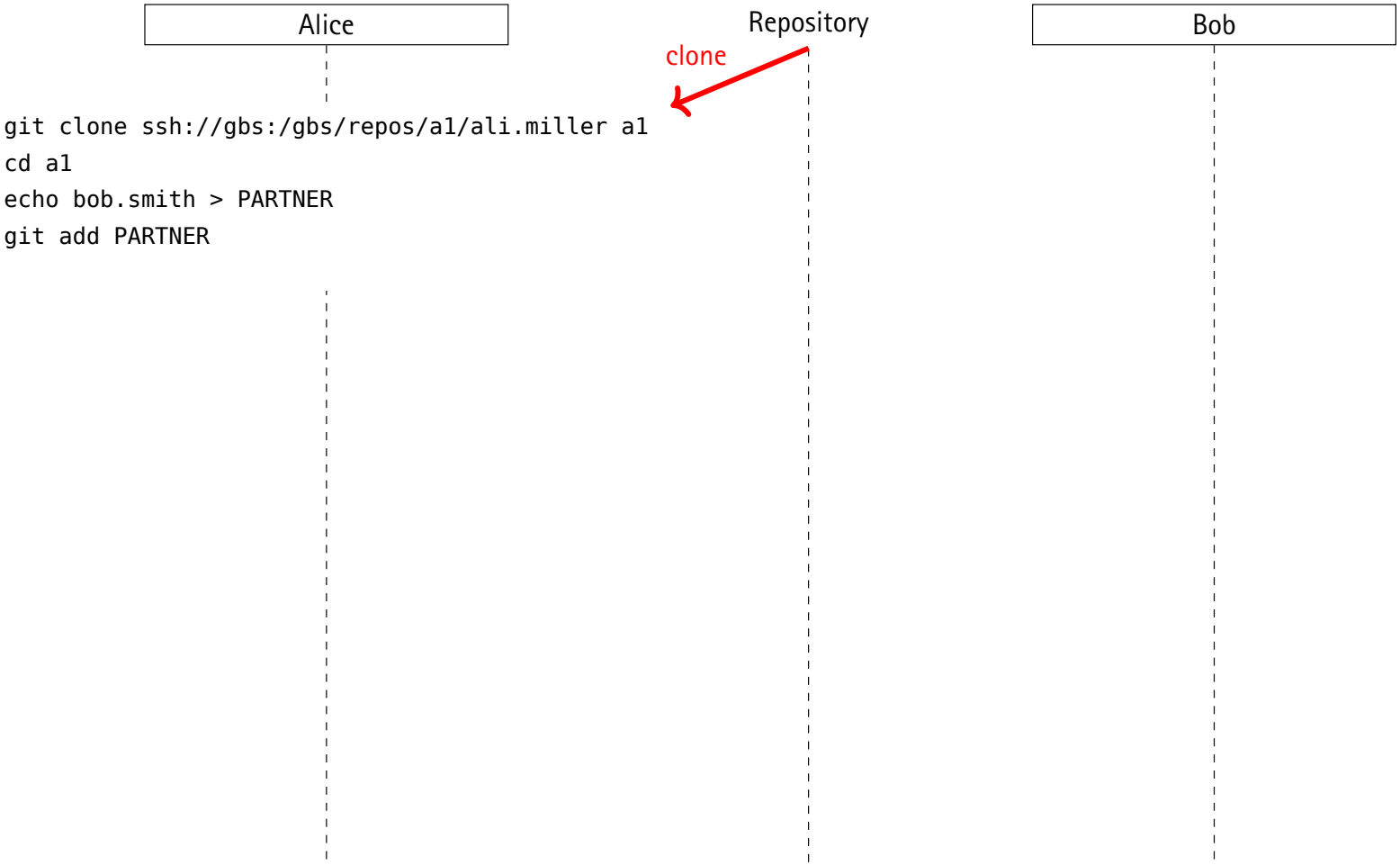
Abgabeprotokoll



Abgabeprotokoll

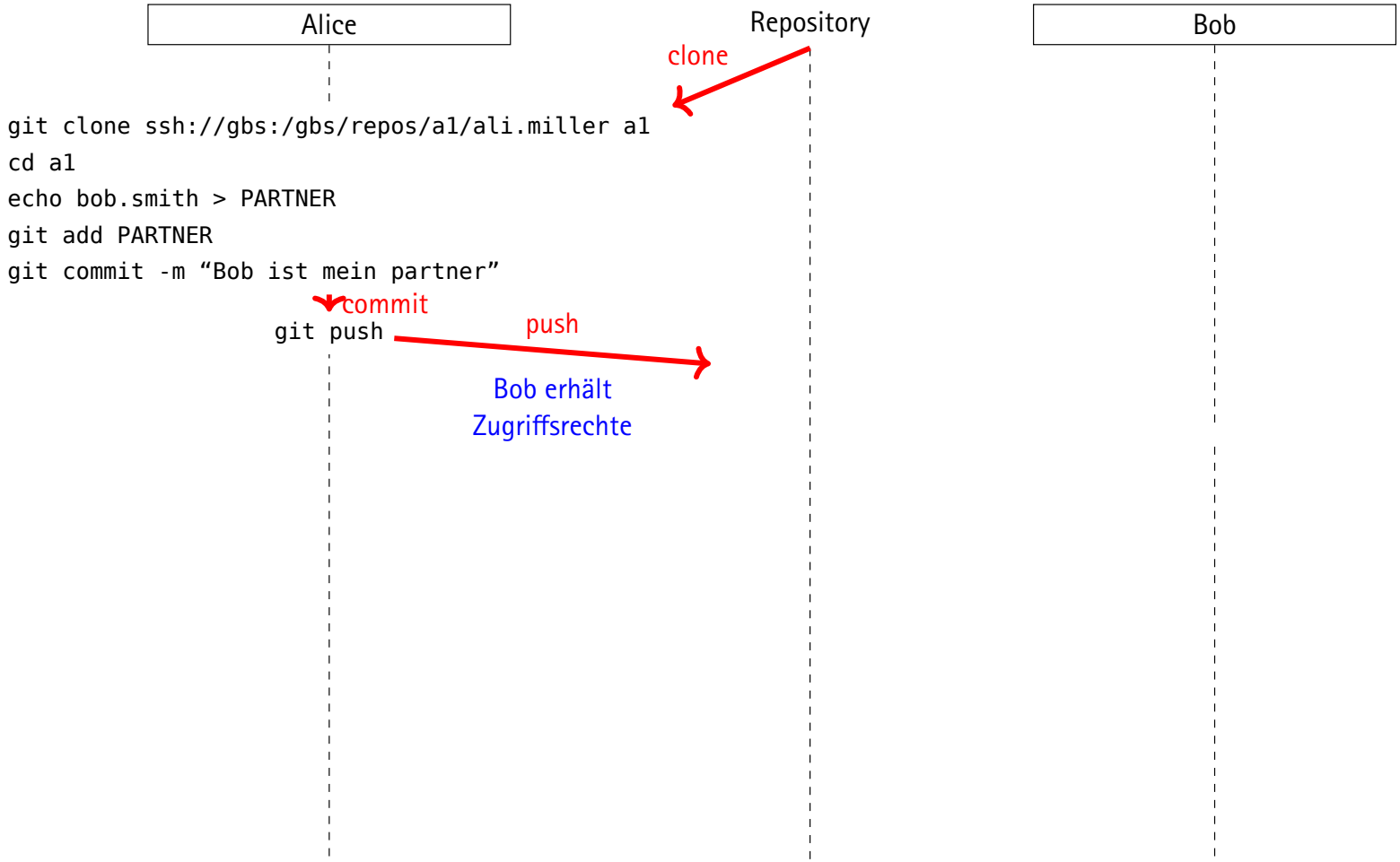


Abgabeprotokoll

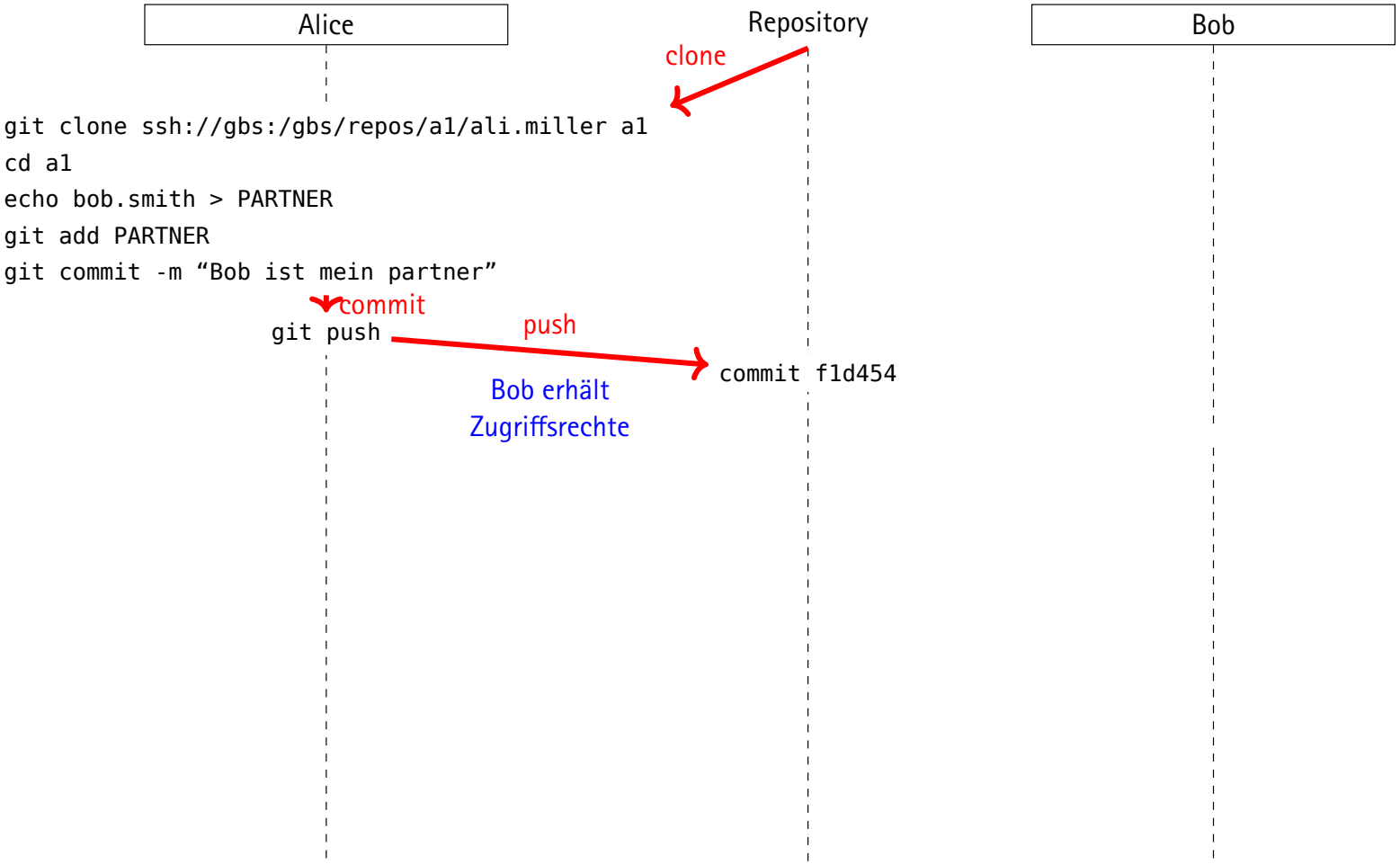


Abgabeprotokoll

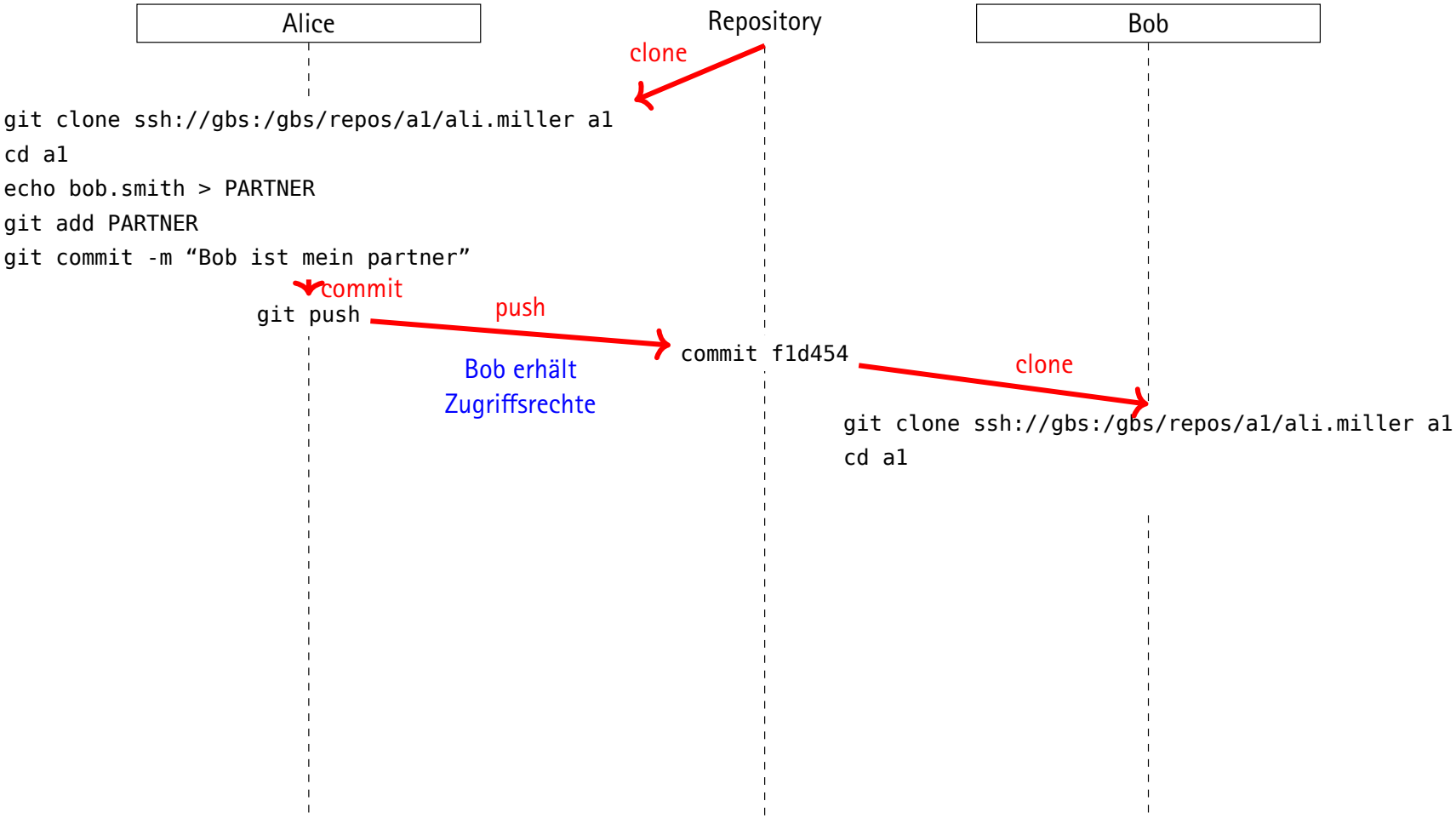
Abgabeprotokoll



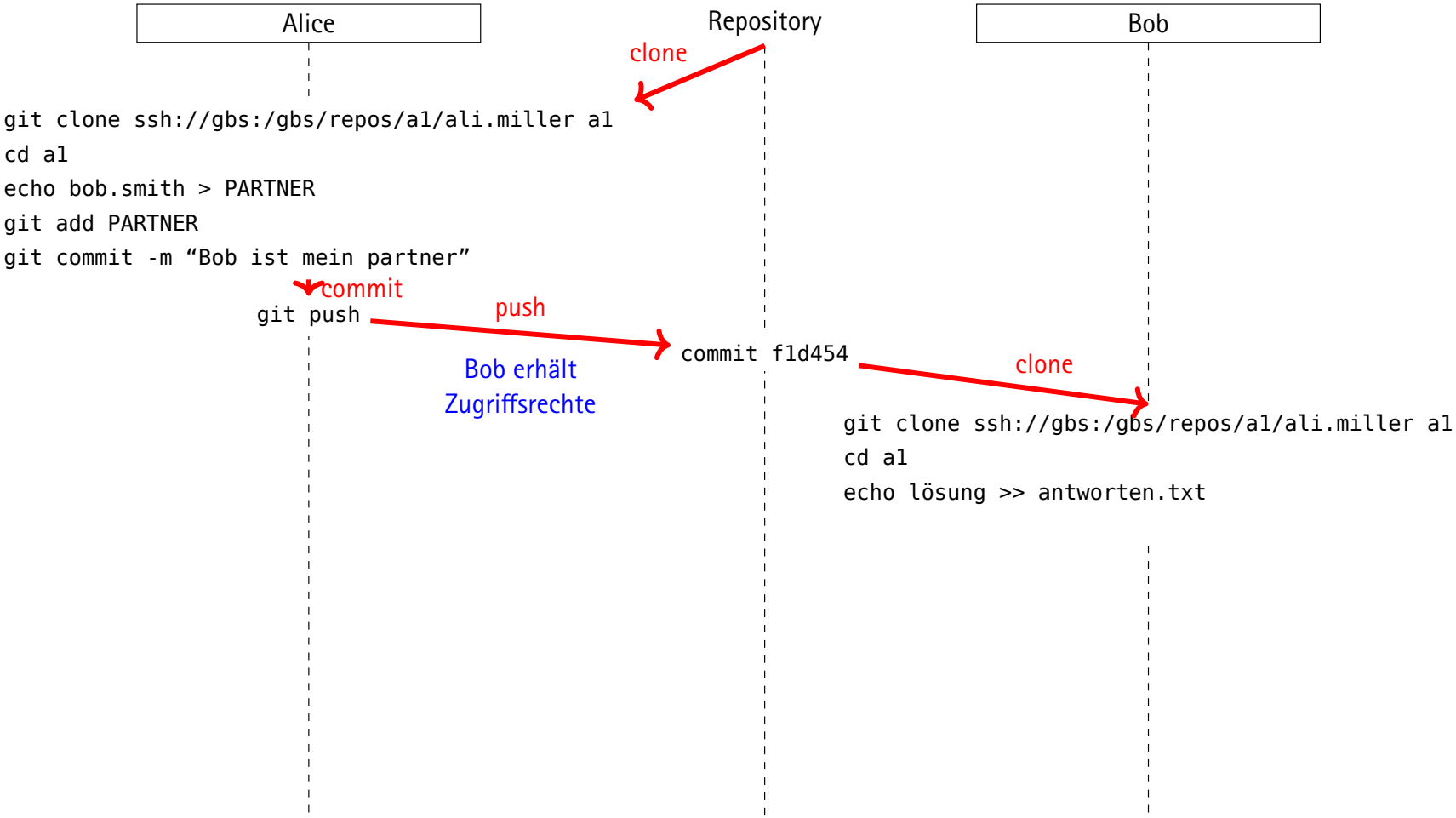
Abgabeprotokoll



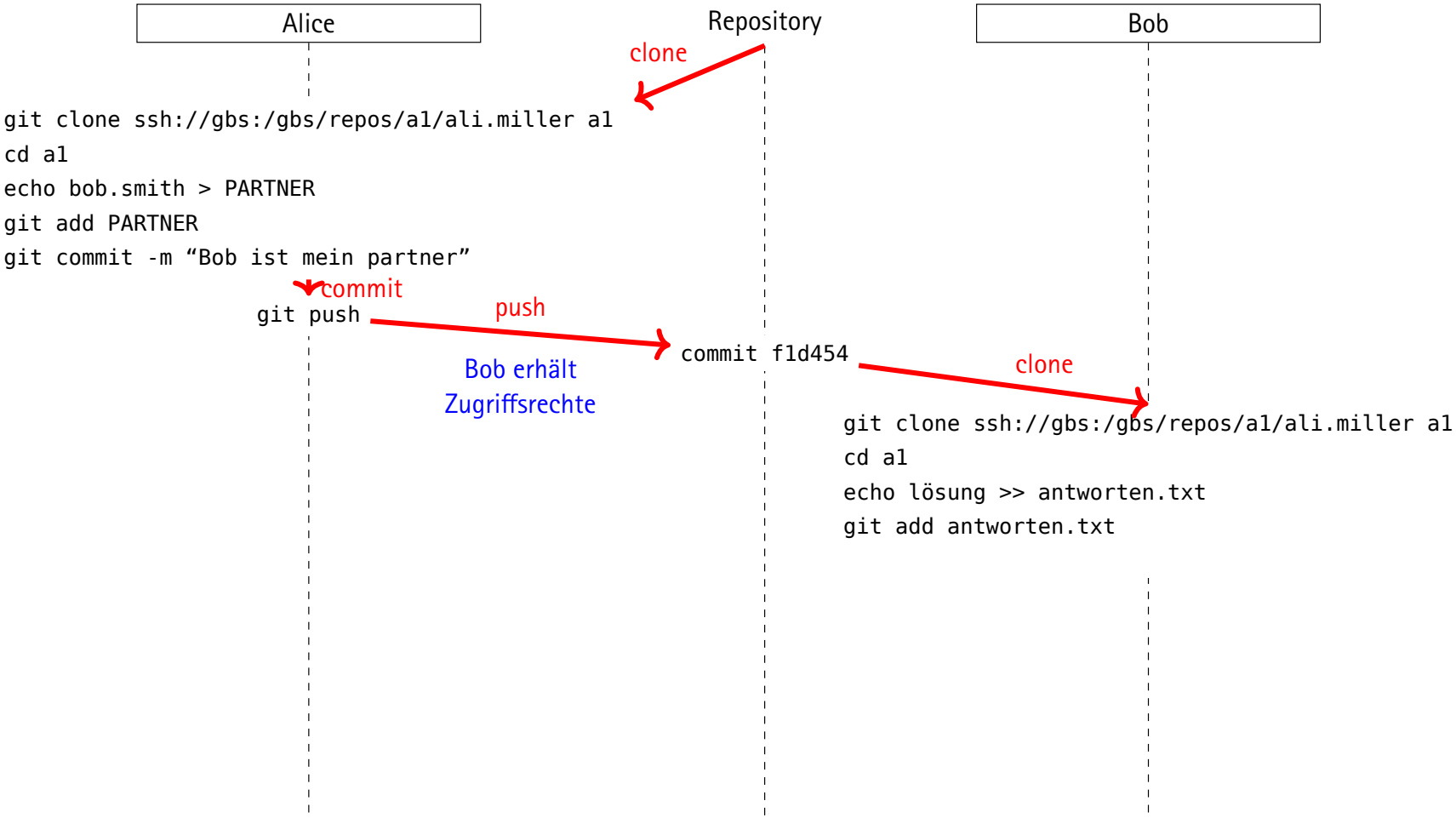
Abgabeprotokoll



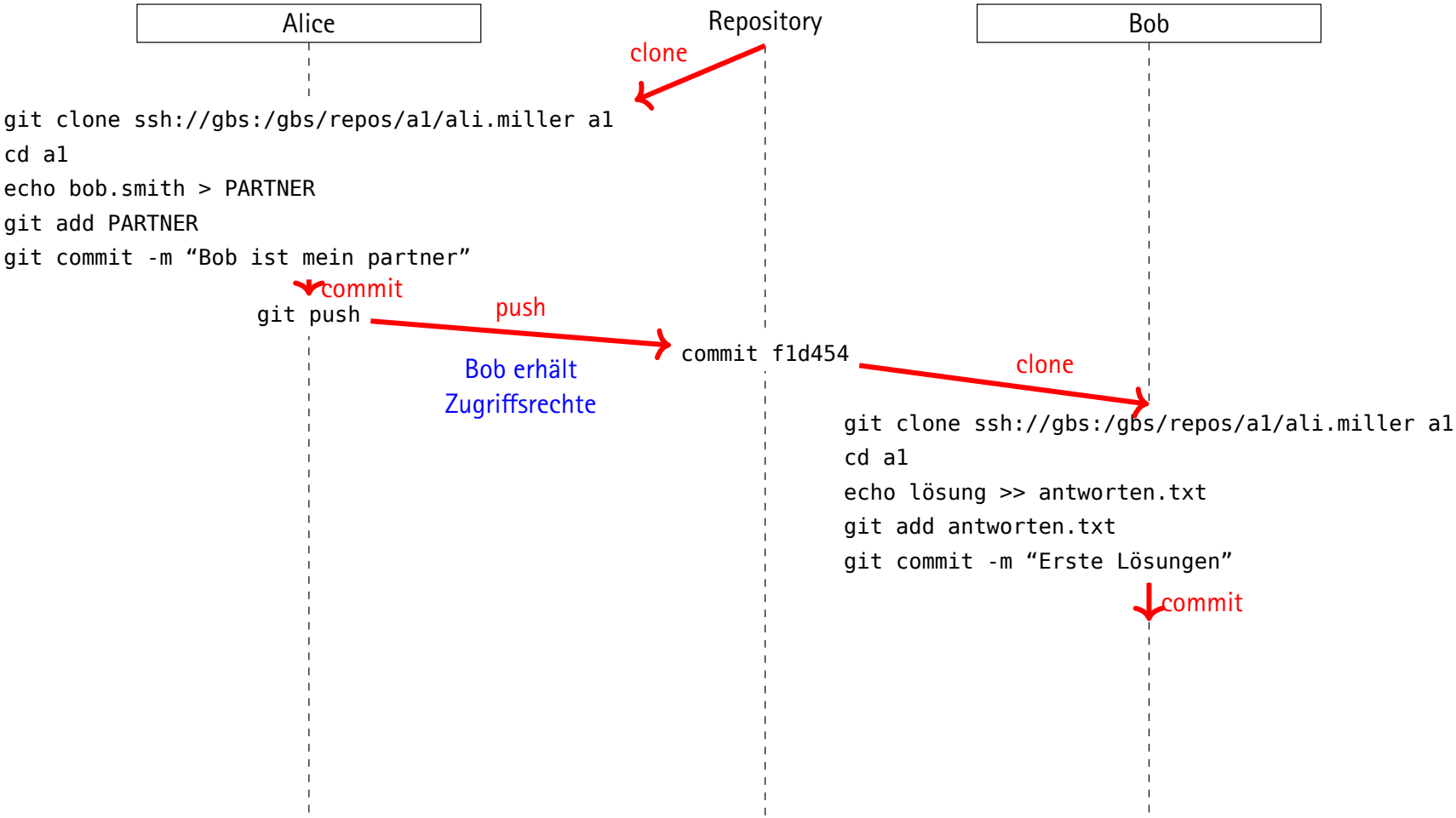
Abgabeprotokoll



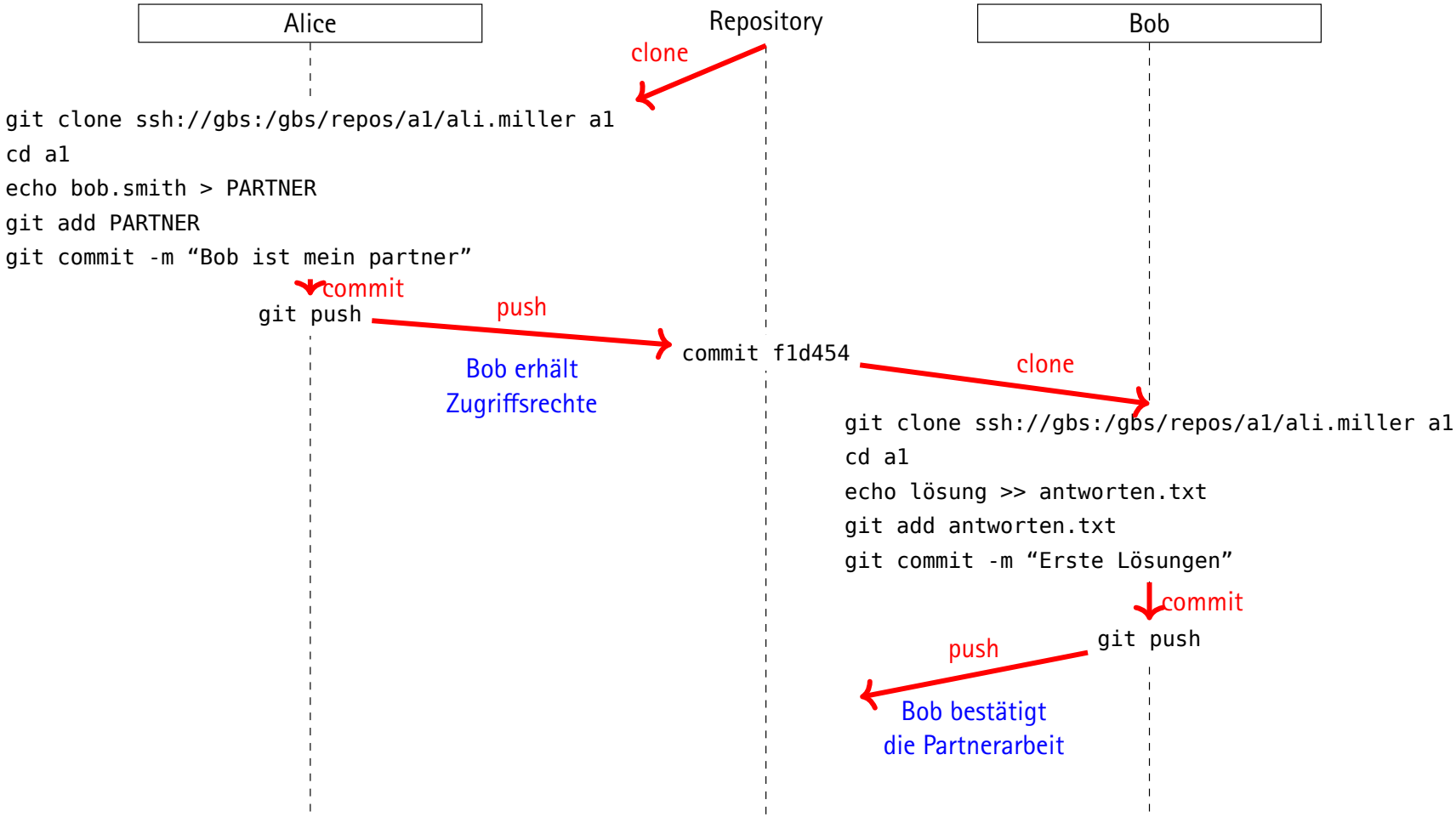
Abgabeprotokoll



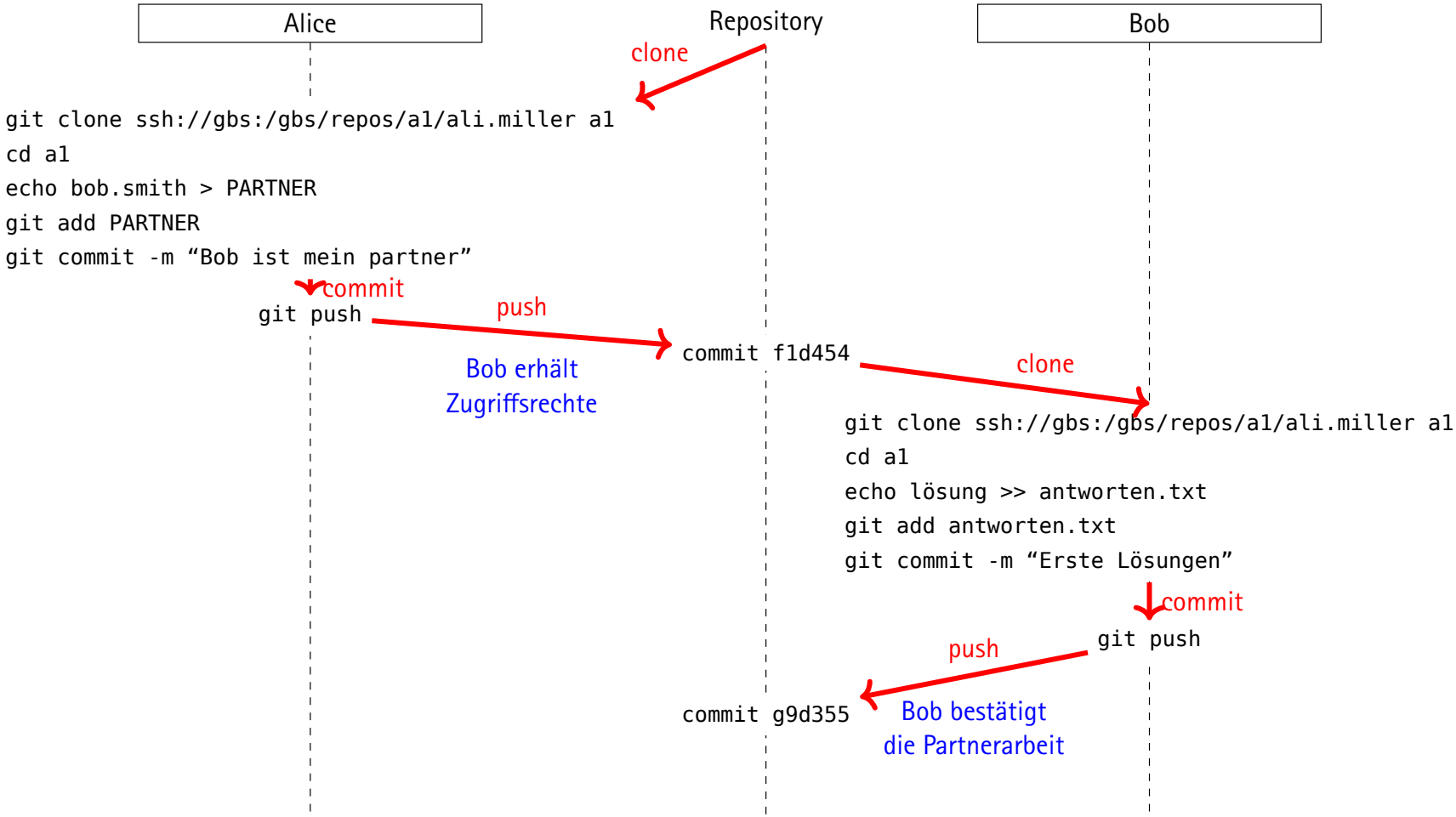
Abgabeprotokoll



Abgabeprotokoll



Abgabeprotokoll



Linux Crashkurs

Die Shell

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

Die Shell

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
```

Die Shell

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
foobar
$
```

Die Shell

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
foobar
$
```

- Steuerbefehle

CTRL-C	beendet Prozess
CTRL-Z	pausiert Prozess
bg	Prozess im Hintergrund fortsetzen
fg	Prozess im Vordergrund fortsetzen
CTRL-R	In Befehlshistorie suchen
↑ ↓	Befehlshistorie durchblättern
TAB	auto Vervollständigung

Die Shell

- Was ist die Shell?
  - Mediator zwischen Anwender und Computer
  - Interpreter für Befehle
  - Arbeitsumgebung
  - erlaubt Interaktion mit dem Betriebssystem

```
$ echo foobar
foobar
$
```

- Steuerbefehle

CTRL-C	beendet Prozess
CTRL-Z	pausiert Prozess
bg	Prozess im Hintergrund fortsetzen
fg	Prozess im Vordergrund fortsetzen
CTRL-R	In Befehlshistorie suchen
↑ ↓	Befehlshistorie durchblättern
TAB	auto Vervollständigung

bash verwenden

awk	Programmiersprache zum Bearbeiten von Textdaten
cat	Dateien lesen
cd	Verzeichnis wechseln
cp	Datei kopieren
cut	Ausschneiden von Spalten
date	Datum und Zeit anzeigen
echo	Ausgabe erzeugen
find	Dateien auflisten
grep	Dateien durchsuchen
head	Ausgabe der ersten Zeilen
kill	Prozesse töten
less	Seitenweise Ausgabe
ls	Ordnerinhalt auflisten
man	Hilfeseiten anzeigen
mv	Datei verschieben
nano	Einfacher Texteditor
ps	Prozesse anzeigen
pwd	Aktuelles Verzeichnis anzeigen
rm	Datei löschen
rmdir	Ordner löschen
sed	Manipulation von Textdaten
ssh	Shell auf entferntem Rechner
tail	Ausgabe der letzten Zeilen
tar	Archivierungswerkzeug
uniq	Gleiche Zeilen löschen
wc	Zeichen und Zeilen zählen

bash verwenden

awk	Programmiersprache zum Bearbeiten von Textdaten
cat	Dateien lesen
cd	Verzeichnis wechseln
cp	Datei kopieren
cut	Ausschneiden von Spalten
date	Datum und Zeit anzeigen
echo	Ausgabe erzeugen
find	Dateien auflisten
grep	Dateien durchsuchen
head	Ausgabe der ersten Zeilen
kill	Prozesse töten
less	Seitenweise Ausgabe
ls	Ordnerinhalt auflisten
man	Hilfeseiten anzeigen
mv	Datei verschieben
nano	Einfacher Texteditor
ps	Prozesse anzeigen
pwd	Aktuelles Verzeichnis anzeigen
rm	Datei löschen
rmdir	Ordner löschen
sed	Manipulation von Textdaten
ssh	Shell auf entferntem Rechner
tail	Ausgabe der letzten Zeilen
tar	Archivierungswerkzeug
uniq	Gleiche Zeilen löschen
wc	Zeichen und Zeilen zählen

bash 2019-09-06

manpages – das Hilfesystem unter Unix

Typische Verwendung

`man <Befehl>`

man echo

```
ECHO(1)                    User Commands                    ECHO(1)

NAME
       echo - display a line of text

SYNOPSIS
       echo [OPTION]... [STRING]...

DESCRIPTION
       Echo the STRING(s) to standard output.

       -n     do not output the trailing newline
```

man 2021-09-10

Bedienung von man

Die wichtigsten Tasten

- **Scrollen (zeilenweise)**: Pfeiltaste hoch/runter
- **Scrollen (seitenweise)**: Bild auf/ab
- **Suchen**: `/suchbegriff<ENTER>`
- **Nächster Treffer**: `n`
- **Vorheriger Treffer**: `N`
- **Beenden**: `q`

Tipp: Auch andere Befehle wie `less` lassen sich so bedienen!

Und wenn ich nicht weiß, welchen Befehl ich brauche?

apropos ist dein Freund!

`apropos <Suchbegriff>`

```
$ apropos rename
...
mv (1) - move (rename) files
prename (1) - renames multiple files
rename (2) - change the name or location of a file
...
```

Wenn die Anzeige zu lang wird, hilft `apropos <Befehl> | less` weiter.

man 2021-09-10

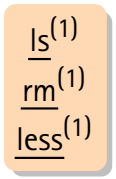
Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  4. Spezielle Dateien (gewöhnlich in /dev)
  5. Dateiformate und Konventionen, z. B. /etc/passwd
  6. Spiele
  7. Verschiedenes (incl. Makropaketen und Konventionen)
  8. Befehle für die Systemverwaltung (in der Regel nur für root)
  9. Kernel-Routinen [nicht Standard]

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  4. Spezielle Dateien (gewöhnlich in /dev)
  5. Dateiformate und Konventionen, z. B. /etc/passwd
  6. Spiele
  7. Verschiedenes (incl. Makropaketen und Konventionen)
  8. Befehle für die Systemverwaltung (in der Regel nur für root)
  9. Kernel-Routinen [nicht Standard]



man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  4. Spezielle Dateien (gewöhnlich in /dev)
  5. Dateiformate und Konventionen, z. B. /etc/passwd
  6. Spiele
  7. Verschiedenes (incl. Makropaketen und Konventionen)
  8. Befehle für die Systemverwaltung (in der Regel nur für root)
  9. Kernel-Routinen [nicht Standard]

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  4. Spezielle Dateien (gewöhnlich in /dev)
  5. Dateiformate und Konventionen, z. B. /etc/passwd
  6. Spiele
  7. Verschiedenes (incl. Makropaketen und Konventionen)
  8. Befehle für die Systemverwaltung (in der Regel nur für root)
  9. Kernel-Routinen [nicht Standard]

printf(3)
malloc(3)
sem_post(3)

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  4. Spezielle Dateien (gewöhnlich in /dev)
  5. Dateiformate und Konventionen, z. B. /etc/passwd
  6. Spiele
  7. Verschiedenes (incl. Makropaketen und Konventionen)
  8. Befehle für die Systemverwaltung (in der Regel nur für root)
  9. Kernel-Routinen [nicht Standard]

gittutorial(7)
ascii(7)
signal(7)

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  7. Verschiedenes (incl. Makropaketen und Konventionen)

```
$ man -f write
```

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  7. Verschiedenes (incl. Makropaketen und Konventionen)

```
$ man -f write
write (1) - send a message to another user
write (2) - write to a file descriptor
write (3posix) - write on a file
write (1posix) - write to another user
$
```

man 2021-09-10

Man Sections

- Aufteilung in Kategoriene
  1. Ausführbare Programme oder Shell-Befehle
  2. Systemaufrufe (Kernel-Funktionen)
  3. Bibliotheksaufrufe (Funktionen in Programmbibliotheken)
  7. Verschiedenes (incl. Makropaketen und Konventionen)

```
$ man -f write
write (1) - send a message to another user
write (2) - write to a file descriptor
write (3posix) - write on a file
write (1posix) - write to another user
$ man 2 write
```

man 2021-09-10

man 2 write

```
man 2 write
WRITE(2)                Linux Programmer's Manual                WRITE(2)

NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write() writes up to count bytes from the buffer starting at buf
       to the file referred to by the file descriptor fd.
```

man 2021-09-10

Wo bin ich und wo muss ich hin?

- Wo bin ich?
  - pwd

Wo bin ich und wo muss ich hin?

- Wo bin ich?
  - pwd
- Wie komme ich in mein Home-Verzeichnis?
  - cd

Wo bin ich und wo muss ich hin?

- Wo bin ich?
  - `pwd`
- Wie komme ich in mein Home-Verzeichnis?
  - `cd`
- Wie bekomme ich eine Arbeitskopie meines Abgaberepos?
  - `git clone ssh://gbs:/gbs/repos/<aufgabe>/<user> <Ziel>`

Wo bin ich und wo muss ich hin?

- Wo bin ich?
  - `pwd`
- Wie komme ich in mein Home-Verzeichnis?
  - `cd`
- Wie bekomme ich eine Arbeitskopie meines Abgaberepos?
  - `git clone ssh://gbs:/gbs/repos/<aufgabe>/<user> <Ziel>`
- Wo liegen die Vorgaben?
  - `/gbs/vorgaben/<aufgabe>`

Wo bin ich und wo muss ich hin?

- Wo bin ich?
  - `pwd`
- Wie komme ich in mein Home-Verzeichnis?
  - `cd`
- Wie bekomme ich eine Arbeitskopie meines Abgaberepos?
  - `git clone ssh://gbs:/gbs/repos/<aufgabe>/<user> <Ziel>`
- Wo liegen die Vorgaben?
  - `/gbs/vorgaben/<aufgabe>`
- Wie bekomme ich die Vorgaben in mein Repo?
  - Sie sind bereits in deinem Repo vorhanden

Kompilieren

- GCC - GNU Compiler Collection
  - `gcc -o <executable> <source1.c source2 ...>`
  - Übersetzt die angegebenen Quelldateien
  - Eingabe: source1.c, source2, ...
  - Ausgabe: Maschinenprogramm <executable>

- Verhalten des `gcc` kann durch Optionen beeinflusst werden

`-g` Erzeugt Debug-Symbole in der ausführbaren Datei

`-c` Übersetzt Quellcode in Maschinencode, erzeugt aber kein ausführbares Programm

`-Wall` aktiviert weitere Warnungen, die auf mögliche Programmierfehler hinweisen

`-Werror` gcc behandelt Warnungen wie Fehler

Gängige Compiler-Warnungen

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

Gängige Compiler-Warnungen

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.

Gängige Compiler-Warnungen

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.

man 3 printf

```
PRINTF(3)         Linux Programmer's Manual         PRINTF(3)

NAME
       printf,  fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf,
       vdprintf, vsprintf,  vsnprintf  -  formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);
```

Gängige Compiler-Warnungen

- Problem: fehlende Deklaration einer Funktion

```
test.c: In function 'main':
test.c:3:2: warning: implicit declaration of function 'printf'
```

  - Lösung für Bibliotheksfunktionen: Entsprechendes `#include` ergänzen.

    ↪ Manual-Page gibt Auskunft über den Namen der nötigen Headerdateien

  - Lösung für eigene Funktionen: Forward-Deklaration ergänzen.

```
int my_printf(const char *format, ...);
```

Gängige Compiler-Warnungen

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

gcc 2021-09-10

Gängige Compiler-Warnungen

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

```
fun.c: In function 'bar':
fun.c:7:1: warning: 'return' with a value, in function returning void
```

gcc 2021-09-10

Gängige Compiler-Warnungen

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

```
fun.c: In function 'bar':
fun.c:7:1: warning: 'return' with a value, in function returning void
```

  - in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung
  - in der Funktion, die keinen Wert zurückliefern soll muss der Wert vom `return` entfernt werden

Gängige Compiler-Warnungen

- Problem: inkonsistente Rückgabewerte

```
fun.c: In function 'foo':
fun.c:3:1: warning: 'return' with no value, in function returning non-void
```

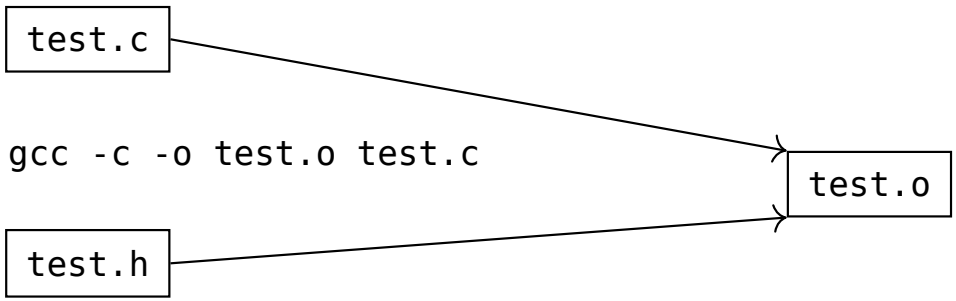
```
fun.c: In function 'bar':
fun.c:7:1: warning: 'return' with a value, in function returning void
```

  - in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung
  - in der Funktion, die keinen Wert zurückliefern soll muss der Wert vom `return` entfernt werden

- Lösung: richtige Verwendung der `return`-Anweisung.

Make

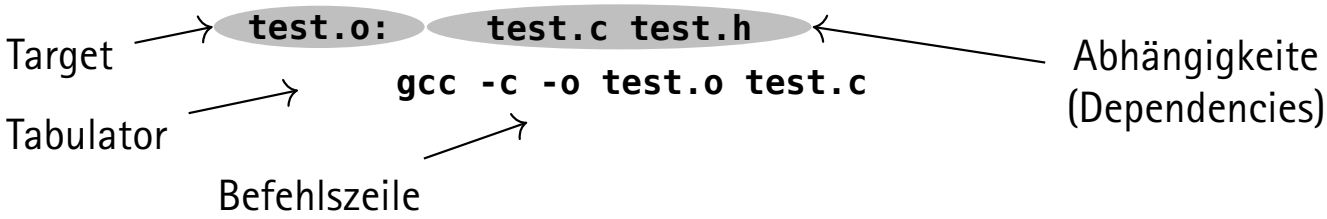
- GNU make
- Automatisierung des Übersetzungsprozess
  - `make <target>`
  - Führt alle nötigen Schritte aus, um <target> zu bauen
- Grundsätzlich: Erzeugung von Dateien aus anderen Dateien
  - für uns interessant: Erzeugung einer .o-Datei aus einer .c-Datei



- Ausführung von *Update*-Operationen (auf Basis der Modifikationszeit)

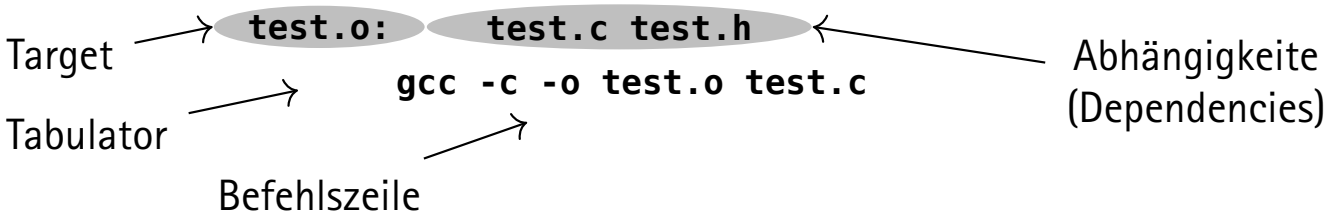
Funktionsweise

- Regeldatei mit dem Namen `Makefile`



Funktionsweise

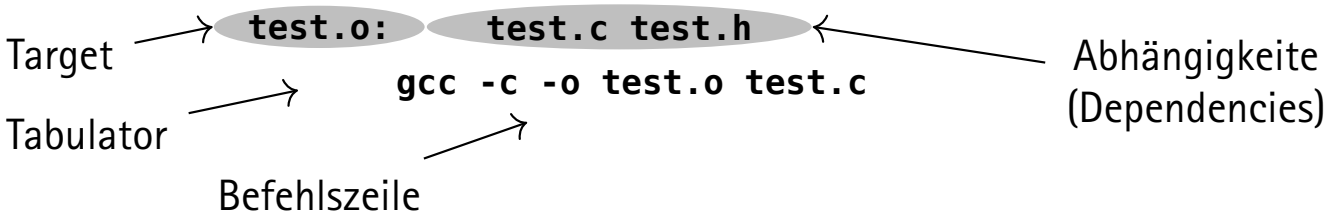
- Regeldatei mit dem Namen `Makefile`



  - Target (was wird erzeugt?)
    - Name der zu erstellenden Datei
  - Abhängigkeiten (woraus?)
    - Namen aller Eingabedateien (direkt oder indirekt)
    - Können selbst Targets sein
  - Befehlszeilen (wie?)
    - Erzeugt aus den Abhängigkeiten das Target

Funktionsweise

- Regeldatei mit dem Namen `Makefile`



  - Target (was wird erzeugt?)
    - Name der zu erstellenden Datei
  - Abhängigkeiten (woraus?)
    - Namen aller Eingabedateien (direkt oder indirekt)
    - Können selbst Targets sein
  - Befehlszeilen (wie?)
    - Erzeugt aus den Abhängigkeiten das Target
- zu erstellendes Target bei `make`-Aufruf angeben: `make test.o`
  - Falls nötig baut `make` die angegebene Datei neu
  - Davor werden rekursiv alle veralteten Abhängigkeiten aktualisiert
  - Ohne Target-Angabe bearbeitet `make` das erste Target im Makefile

Makros

- In einem Makefile können Makros definiert werden

```
SOURCE = test.c func.c
```

Makros

- In einem Makefile können Makros definiert werden

```
SOURCE = test.c func.c
```

- Verwendung der Makros mit `$(NAME)` oder `${NAME}`

```
test: $(SOURCE)
        gcc -o test $(SOURCE)
```

Makros

- In einem Makefile können Makros definiert werden

```
SOURCE = test.c func.c
```

- Verwendung der Makros mit `$(NAME)` oder `${NAME}`

```
test: $(SOURCE)
        gcc -o test $(SOURCE)
```

- Erzeugung neuer Makros durch Konkatenation

```
ALLOBJS = $(OBJS) hallo.o
```

Makros

- In einem Makefile können Makros definiert werden

```
SOURCE = test.c func.c
```

- Verwendung der Makros mit `$(NAME)` oder `${NAME}`

```
test: $(SOURCE)
        gcc -o test $(SOURCE)
```

- Erzeugung neuer Makros durch Konkatenation

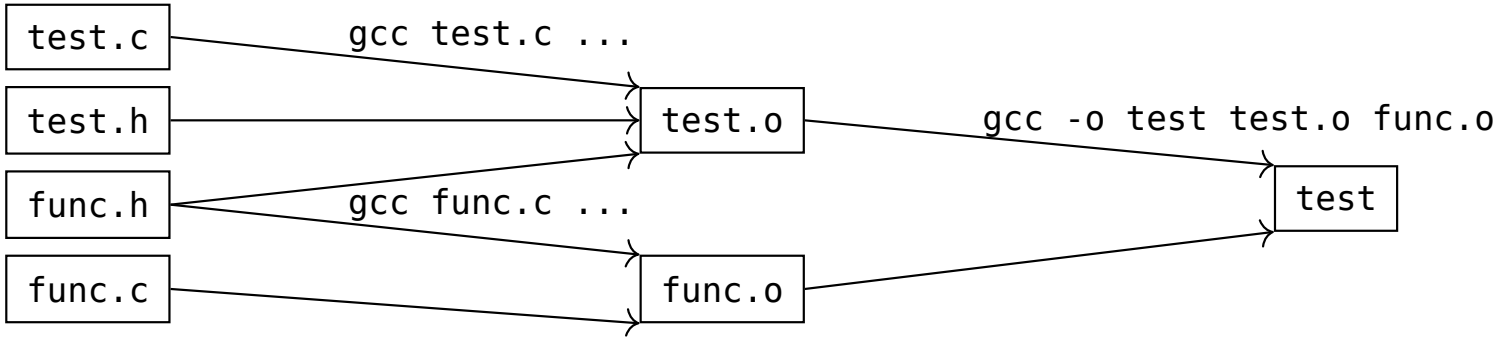
```
ALLOBJS = $(OBJS) hallo.o
```

- Gängige Makros:

`CC`	C-Compiler-Befehl
`CFLAGS`	Optionen für den C-Compiler

Schrittweises Übersetzen

- Rechner beim Erzeugen von ausführbaren Dateien „entlasten"



  - Zwischenprodukte verwenden und somit Übersetzungszeit sparen
  - Nur Produkte aktualisieren, deren Quellen sich verändert haben
- Ein mögliches `Makefile` hierzu

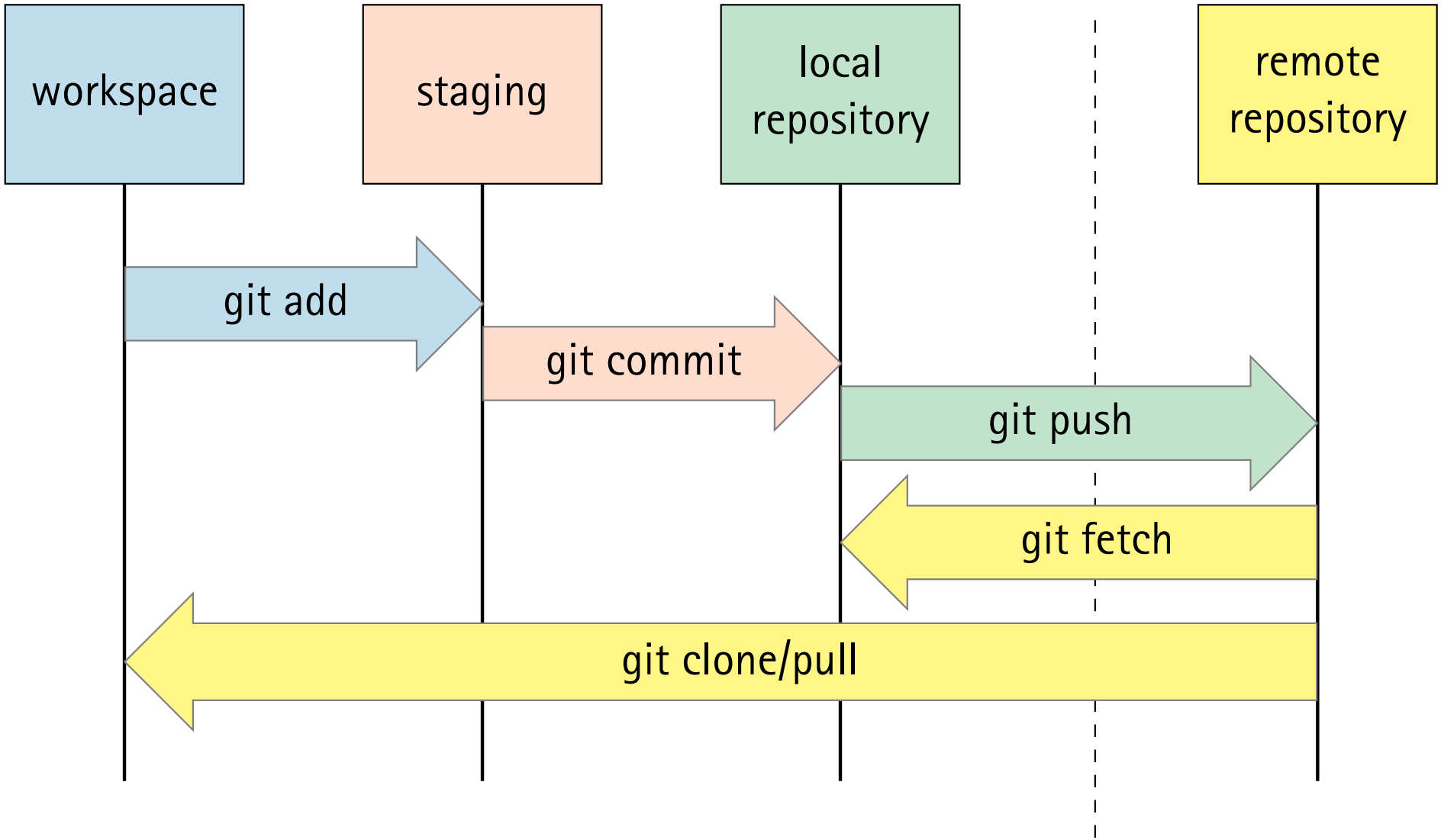
```
test: test.o func.o
    gcc -o test test.o func.o

%.o: %.c
    gcc -c -o $@ $^
```

Git Befehle (siehe git(1))

`git clone`	Eine Kopie eines Repositorys erzeugen
`git add`	Veränderungen zum Commiten vormerken
`git commit`	Veränderungen commiten
`git push`	Commits an Remote übertragen
`git pull`	Commits vom Remote holen
`git log`	Zeigt eine Liste der Commits
`git diff`	Zeigt aktuelle Modifikationen
`git remote list`	Zeigt die bekannten Remotes
`git remote add <name> <url>`	Fügt den neuen Remote <name> hinzu
`git pull <remote> <branch>`	Holt den branch vom angegebenen Remote
`git push <remote> <branch>`	Überträgt den aktuellen Branch an den angegebenen Remote und Branch
`git help <command>`	Zeigt die Hilfe für das Kommando
`tig`	Kommandozeilenbasierte interaktives Git Werkzeug

Git


workspace
staging
local repository
remote repository
git add
git commit
git push
git fetch
git clone/pull

Aufgabe 1 - Die Aufgabenstellung

Aufgabe 1 - Warmup

- Aufgabenteile
  - Abgabesystem
  - Kommandozeilenwerkzeuge
  - Git
  - Theoriefragen
  - `hello`: “Hello World”-Programm
  - `lilo`: Einfach verkettete Liste

Tafelübung 1: Warmup

Hello World

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

helloworld 2019-09-04

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

- Verwendete Systemfunktionen:

```c
int printf(const char *format, ...);
```

Formatierte Ausgabe von Zeichenketten und Daten. Bereitgestellt durch die C-Standardbibliothek. printf(3)

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

- Verwendete Systemfunktionen:

```c
int printf(const char *format, ...);
```

Formatierte Ausgabe von Zeichenketten und Daten. Bereitgestellt durch die C-Standardbibliothek. printf(3)

- Übersetzen und Ausführen:

```
$
```

helloworld 2019-09-04

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

- Verwendete Systemfunktionen:

```c
int printf(const char *format, ...);
```

Formatierte Ausgabe von Zeichenketten und Daten. Bereitgestellt durch die C-Standardbibliothek. printf(3)

- Übersetzen und Ausführen:

```
$ gcc -o hello hello.c
$
```

helloworld 2019-09-04

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

- Verwendete Systemfunktionen:

```c
int printf(const char *format, ...);
```

Formatierte Ausgabe von Zeichenketten und Daten. Bereitgestellt durch die C-Standardbibliothek. printf(3)

- Übersetzen und Ausführen:

```
$ gcc -o hello hello.c
$ ./hello
```

helloworld 2019-09-04

Hello World

- Minimalprogramm in C (`hello.c`):

```c
#include <stdio.h>

int main(int argc, char** argv){
  printf("Hello World\n");
}
```

- Verwendete Systemfunktionen:

```c
int printf(const char *format, ...);
```

Formatierte Ausgabe von Zeichenketten und Daten. Bereitgestellt durch die C-Standardbibliothek. printf(3)

- Übersetzen und Ausführen:

```
$ gcc -o hello hello.c
$ ./hello
Hello World
$
```

helloworld 2019-09-04

`lilo` – Verkettete Liste

Verkettete Liste

- Wann setzt man eine verkettete Liste ein?

Verkettete Liste

- Wann setzt man eine verkettete Liste ein?
- Anforderungsanalyse für verkettete Liste
  - Wieviele Listenelemente gibt es maximal?
  - Welche Lebensdauer muss ein Listenelement besitzen?
  - In welchem Kontext muss ein Listenelement sichtbar sein?

Verkettete Liste

- Wann setzt man eine verkettete Liste ein?
- Anforderungsanalyse für verkettete Liste
  - Wieviele Listenelemente gibt es maximal?
  - Welche Lebensdauer muss ein Listenelement besitzen?
  - In welchem Kontext muss ein Listenelement sichtbar sein?
- Wir brauchen einen Mechanismus, mit dem Listenelemente
  - in a-priori nicht bekannter Anzahl
  - zur Laufzeit des Programmes erzeugt und zerstört werden können

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

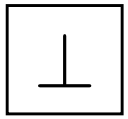
- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
- Zu implementierende Schnittstelle:
  - `int list_append(list_t *list, int value);`
    Fügt einen neuen, nicht-negativen Wert in die Liste ein.
    Erfolg: Rückgabe des eingefügten Werts
    Fehler (duplikat, negativ): Rückgabe -1
  - `int list_pop(list_t * list);`
    Entfernt den ältesten Wert in der Liste und gibt diesen zurück.
    Fehlerfall (leer): -1 wird zurückgeliefert

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
- Zu implementierende Schnittstelle:
  - `int list_append(list_t *list, int value);`
    Fügt einen neuen, nicht-negativen Wert in die Liste ein.
    Erfolg: Rückgabe des eingefügten Werts
    Fehler (duplikat, negativ): Rückgabe -1
  - `int list_pop(list_t * list);`
    Entfernt den ältesten Wert in der Liste und gibt diesen zurück.
    Fehlerfall (leer): -1 wird zurückgeliefert
- Keine Listen-Funktionalität in der `main()`-Funktion
  - Allerdings: Erweitern der `main()` zum Testen erlaubt und **erwünscht**
- Sollte bei der Ausführung einer verwendeten Funktion (z. B. malloc(3)) ein Fehler auftreten, sind keine Fehlermeldungen auszugeben.

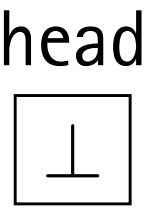
lilo - Funktionsweise

head



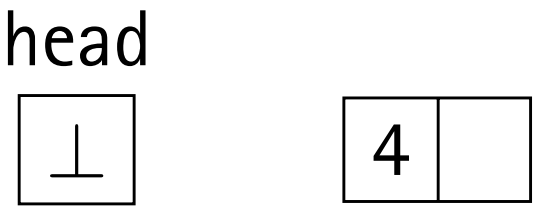
- Zustand: leer
- Operationen:

`lilo` - Funktionsweise



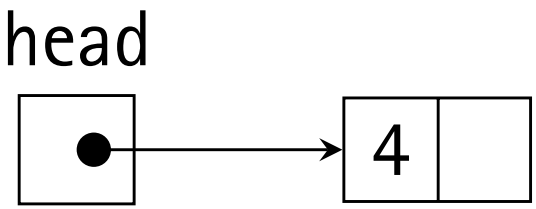
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

lilo - Funktionsweise



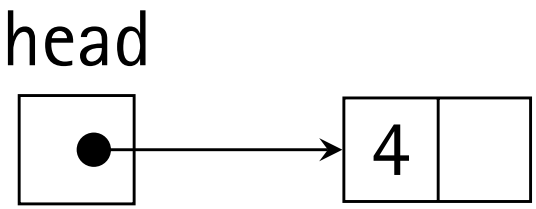
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



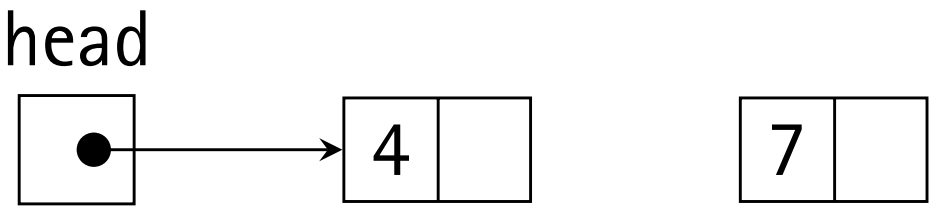
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)`

lilo - Funktionsweise



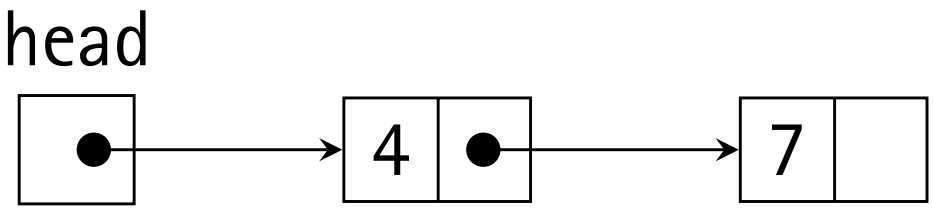
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)` =4

lilo - Funktionsweise



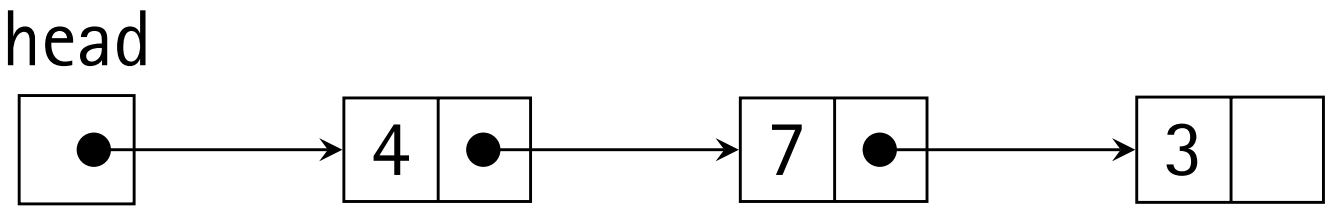
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)`

lilo - Funktionsweise



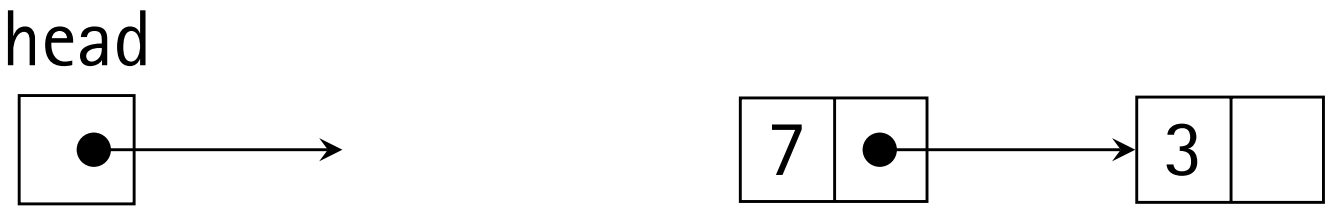
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7

lilo - Funktionsweise



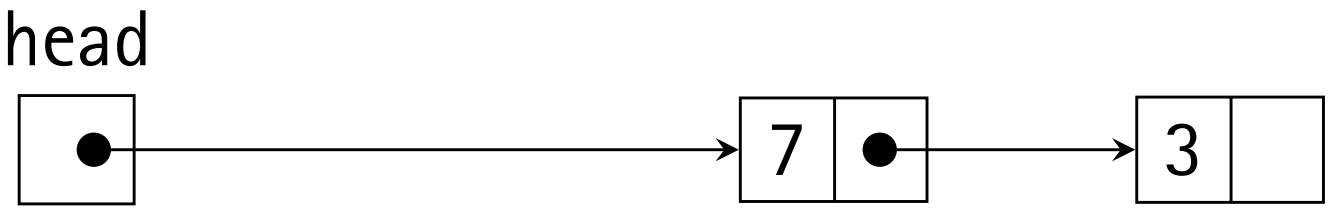
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3

lilo - Funktionsweise



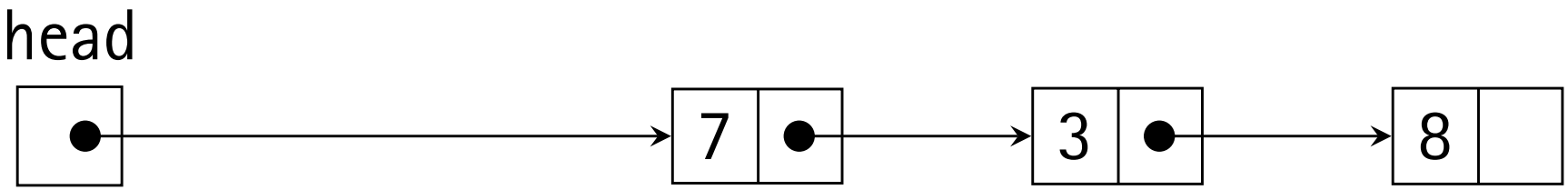
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

lilo - Funktionsweise



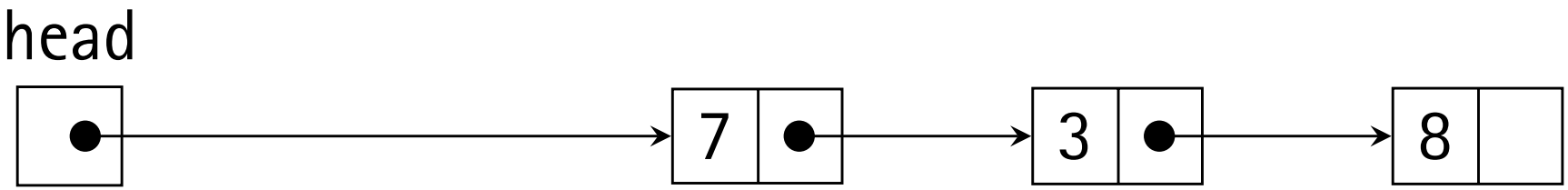
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

lilo - Funktionsweise



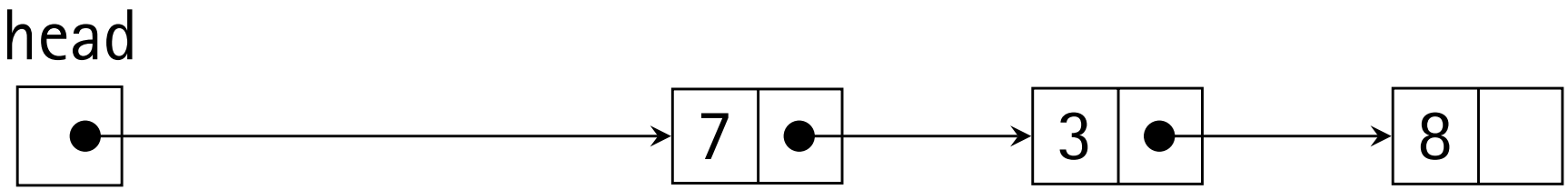
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8

lilo - Funktionsweise



- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)`

lilo - Funktionsweise



- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1

lilo - Funktionsweise

- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7

lilo - Funktionsweise

- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)`

lilo - Funktionsweise

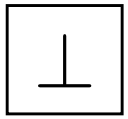
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1

lilo - Funktionsweise

- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3

lilo - Funktionsweise

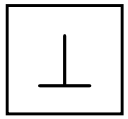
head



- Zustand: leer
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3
  10. `list_pop(head)` =8

lilo - Funktionsweise

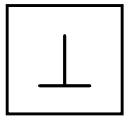
head



- Zustand: leer
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3
  10. `list_pop(head)` =8
  11. `list_pop(head)`

lilo - Funktionsweise

head



- Zustand: leer
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1
  9. `list_pop(head)` =3
  10. `list_pop(head)` =8
  11. `list_pop(head)` =-1

lilo - Datenstruktur

```
typedef struct list_element {

  ...

} list_element_t;

typedef struct list {

  ...

} list_t;
```

lilo - Datenstruktur

```
typedef struct list_element {
    Zeiger auf das nächste Listenelement
  ...
    Wert des Elements
} list_element_t;


typedef struct list {
    Zeiger auf das erste Listenelement
  ...
} list_t;
```

Funktionen für die Hausaufgabe

Funktionen für die Hausaufgabe

`int printf(const char *format, ...);`	Formatierte Ausgabe von Zeichenketten Kann auch in Dateien und `char[]` schreiben. printf(3)
`uid_t getuid(void);`	Liefert die User-ID des aktuellen Benutzers. getuid(2)
`gid_t getgid(void);`	Liefert die Group-ID des aktuellen Benutzers. getgid(2)
`void *malloc(size_t size);`	Allokiert `size` Bytes Speicher auf dem Heap. malloc(3)
`void free(void *ptr)`	Gibt den zuvor allokierten Speicher, auf den `ptr` zeigt, wieder frei. free(3)

- Fehler können wie in den Man-Pages beschrieben, auftreten und sind gemäß der Aufgabenstellung abzufangen
- Es soll keine Ausgabe von Fehlernachrichten geschehen.

C - Antipattern

C - So geht's **nicht**

- Problem:

```
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

C - So geht's **nicht**

- Problem: use after free

```
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

Der Speicher, auf den stack_elem zeigt, ist nach dem free() nicht mehr gültig!

C - So geht's **nicht**

- Problem: use after free

```
int stack_pop(stack_t *stack) {
  stack_elem elem* = stack->first;
  stack->first = stack->first->next;
  free(elem);
  return elem->value;
}
```

Der Speicher, auf den stack_elem zeigt, ist nach dem free() nicht mehr gültig!

- Lösung: Zwischenspeichern

```
int stack_pop(stack_t *stack) {
  stack_elem *top_of_stack = stack->first;
  int tos_value = top_of_stack->value;
  stack->first = stack->first->next;
  free(top_of_stack);
  return tos_value;
  }
```

C - So geht's **nicht**

```
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

C - So geht's **nicht**

```
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem: return local address

```
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

f liegt auf dem Stack im Aufrufrahmen der create_family Funktion.
Beim Verlassen der Funktion verliert f die Gültigkeit.
Der zurückgegebene Zeiger zeigt damit auf ungültigen Speicher!

C - So geht's **nicht**

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem: return local address

```c
struct family *create_family(int adults, int children, int pets) {
  struct family f;
  f.num_adults = adults;
  f.num_children = children;
  f.num_pets = pets;
  return &f;
}
```

f liegt auf dem Stack im Aufrufrahmen der create_family Funktion.
Beim Verlassen der Funktion verliert f die Gültigkeit.
Der zurückgegebene Zeiger zeigt damit auf ungültigen Speicher!

- Lösung: Dynamische Speicheranforderung

```c
struct family *create_family(int parents, int children, int pets) {
  struct family *f = malloc(sizeof(struct family));
  f->num_adults = adults;
  f->num_children = children;
  f->num_pets = pets;
  return f;
}
```

C - So geht's **nicht**

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

C - So geht's **nicht**

```c
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```c
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

C - So geht's ~~nicht~~ doch

```
struct family {
  int num_adults;
  int num_children;
  int num_pets;
}
```

- Problem:

```
int calculate_theme_park_costs(int parents, int children, int pets) {
  struct family f = {parents,children,pets};
  int overall_costs = 0;
  int bus_costs = compute_bus_tickets(&f);
  int ticket_costs = compute_ticket_costs(&f);
  overall_costs = bus_costs + ticket_costs;
  return overall_costs;
}
```

Kein Problem!
f „lebt“ länger, als der Funktionsaufruf dauert.

Gute Lösung für zeitlich begrenzten „dynamischen“ Speicher

Warum machen wir diese Übungen?

> *“ Was du mir sagst, das vergesse ich.*
> *Was du mir zeigst, daran erinnere ich mich.*
> *Was du mich tun lässt, das verstehe ich. ”*
>
> Konfuzius

Warum machen wir diese Übungen?

> *“ Was du mir sagst, das vergesse ich.*
> *Was du mir zeigst, daran erinnere ich mich.*
> *Was du mich tun lässt, das verstehe ich. ”*
>
> Konfuzius

Happy Coding :-)