Übung Grundlagen der Betriebssysteme (GBS)

Gruppenübung 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: Gruppenübung 1: Warmup

Gruppenübung 1: Warmup

- Benutzeraccount im Abgabesystem
- Shell - Die Schnittstelle zum Rechner
- Abgabeprotokoll
- Hello World und Make
- Verkettete Liste
- Backup
- Herumklettern im Dateisystembaum
- Inhalte aufzeigen

Woher bekomme ich einen Linux Computer?

- Verfügbare Linux Computer
  - Dein eigener Laptop
  - Der GBS-Server
  - Die GBS-VM

Woher bekomme ich einen Linux Computer?

- Verfügbare Linux Computer
  - Dein eigener Laptop
  - Der GBS-Server
  - Die GBS-VM
  - Leihgeräte für die Gruppenübungen

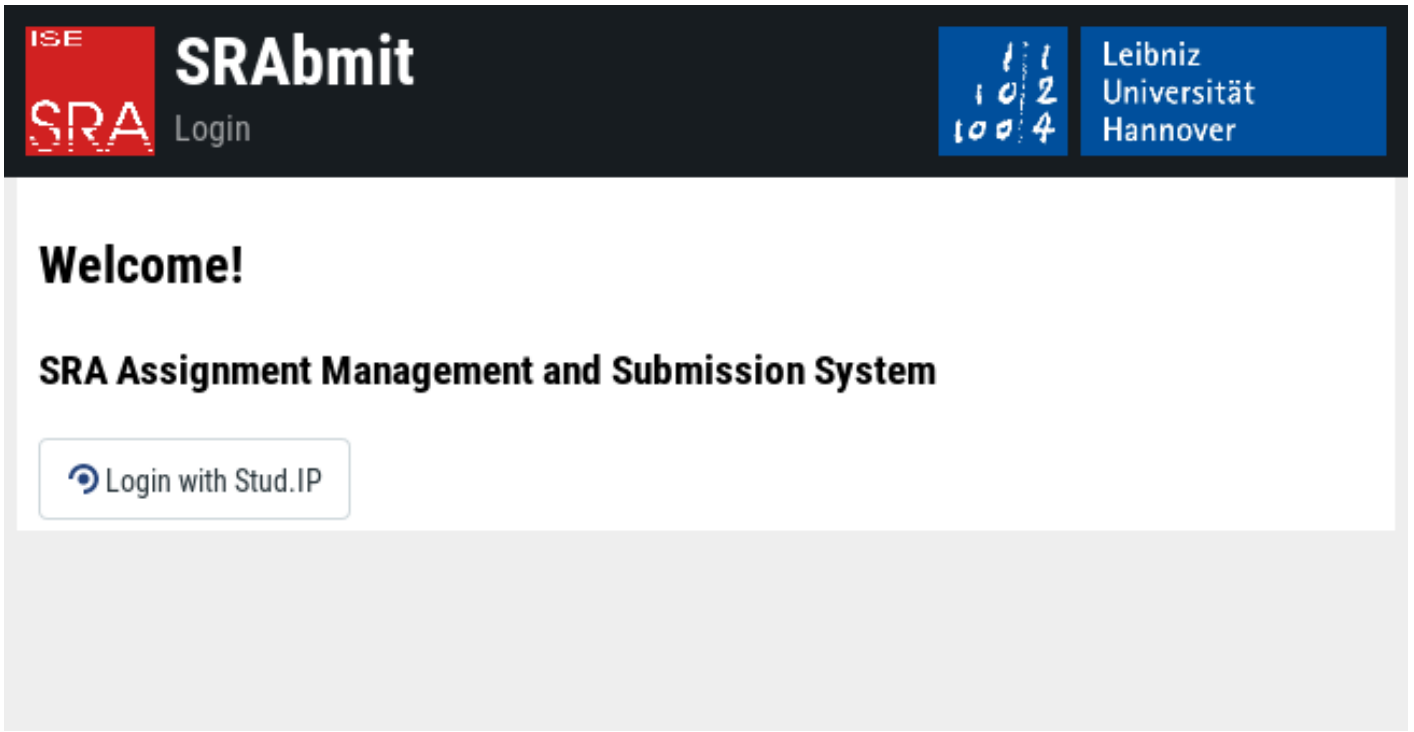
Woher bekomme ich einen Linux Computer?

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

Benutzeraccount im Abgabesystem

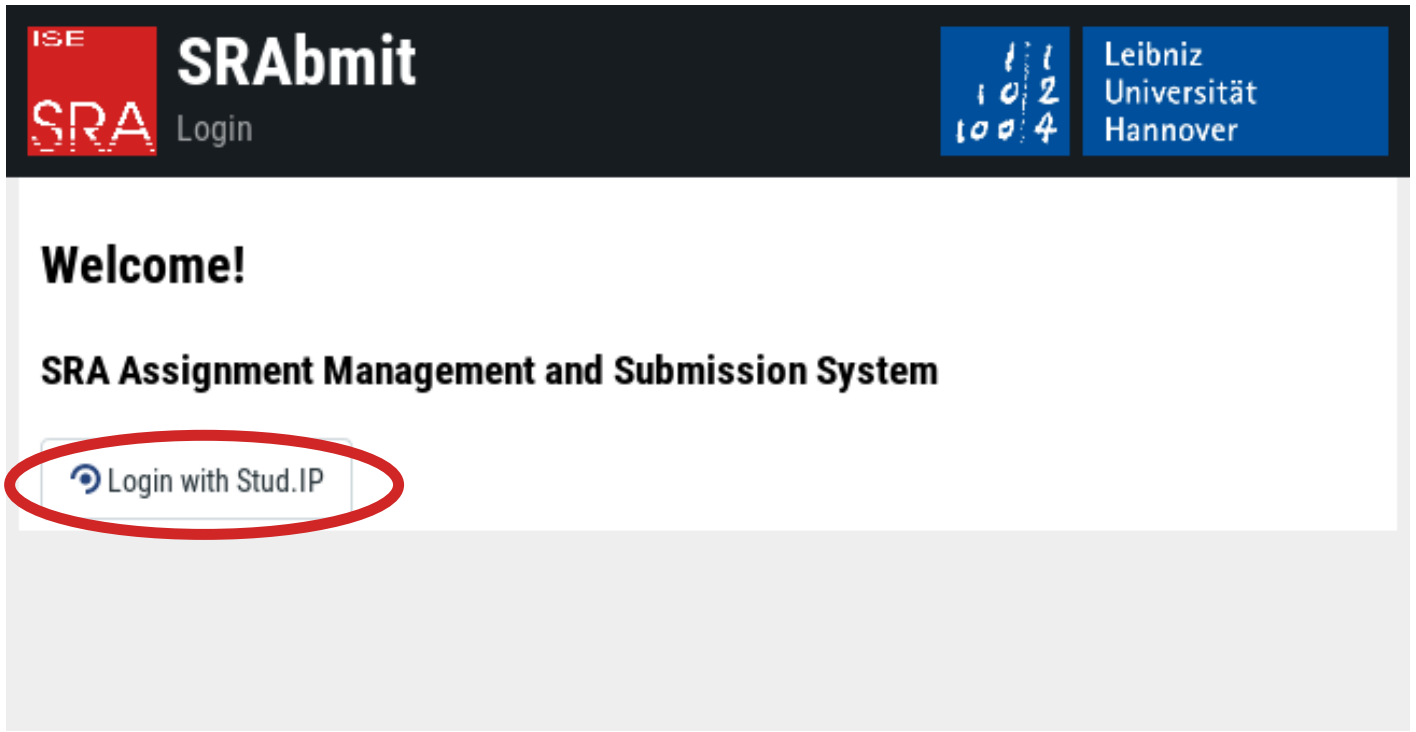
Account anlegen im Abgabesystem

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

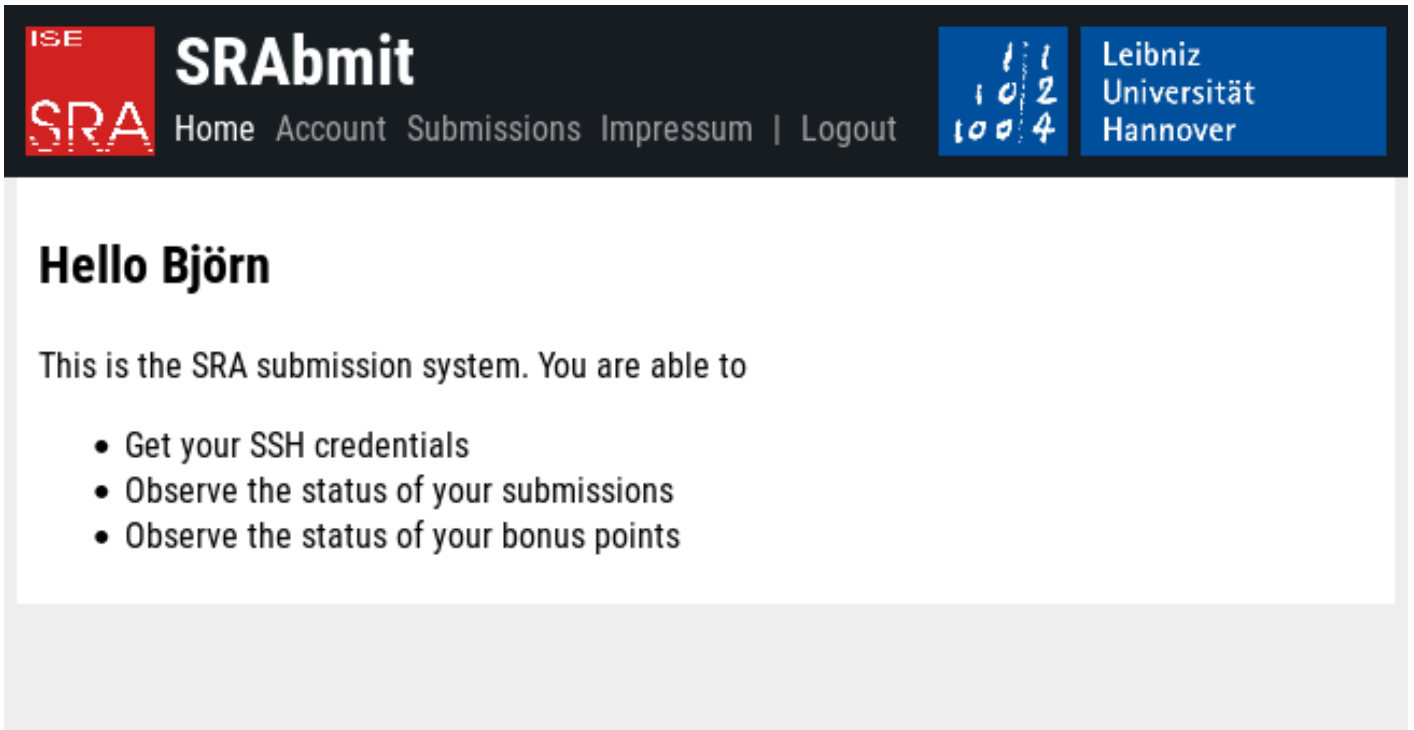


Account anlegen im Abgabesystem

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

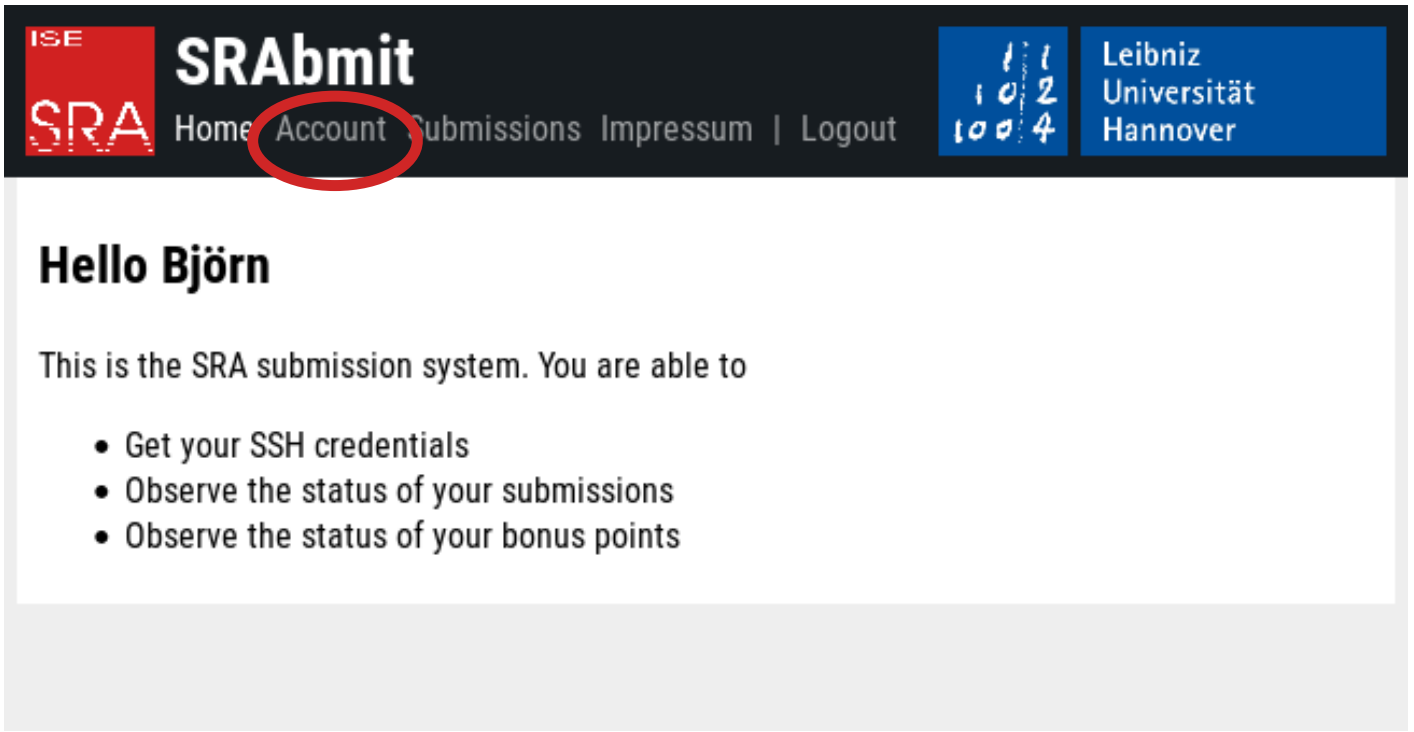
Account anlegen im Abgabesystem

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



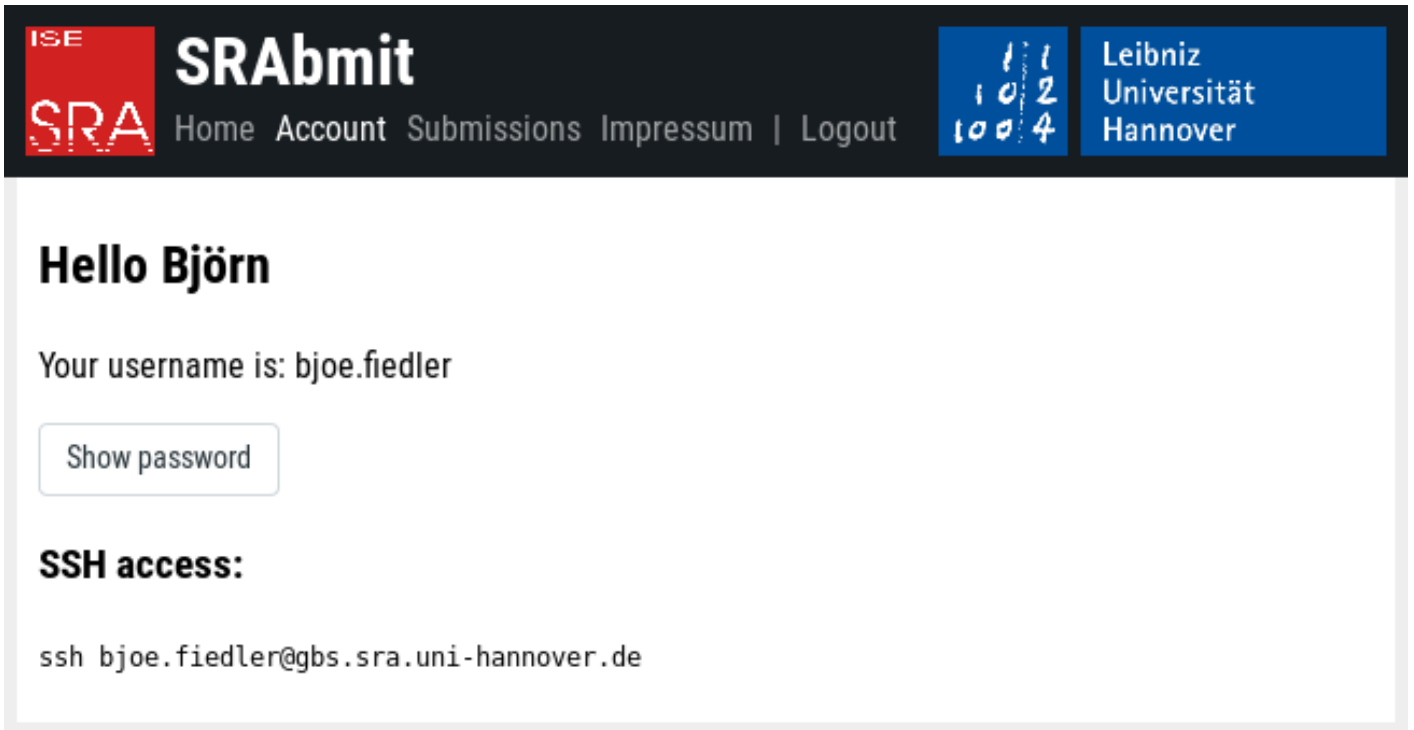
Account anlegen im Abgabesystem

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



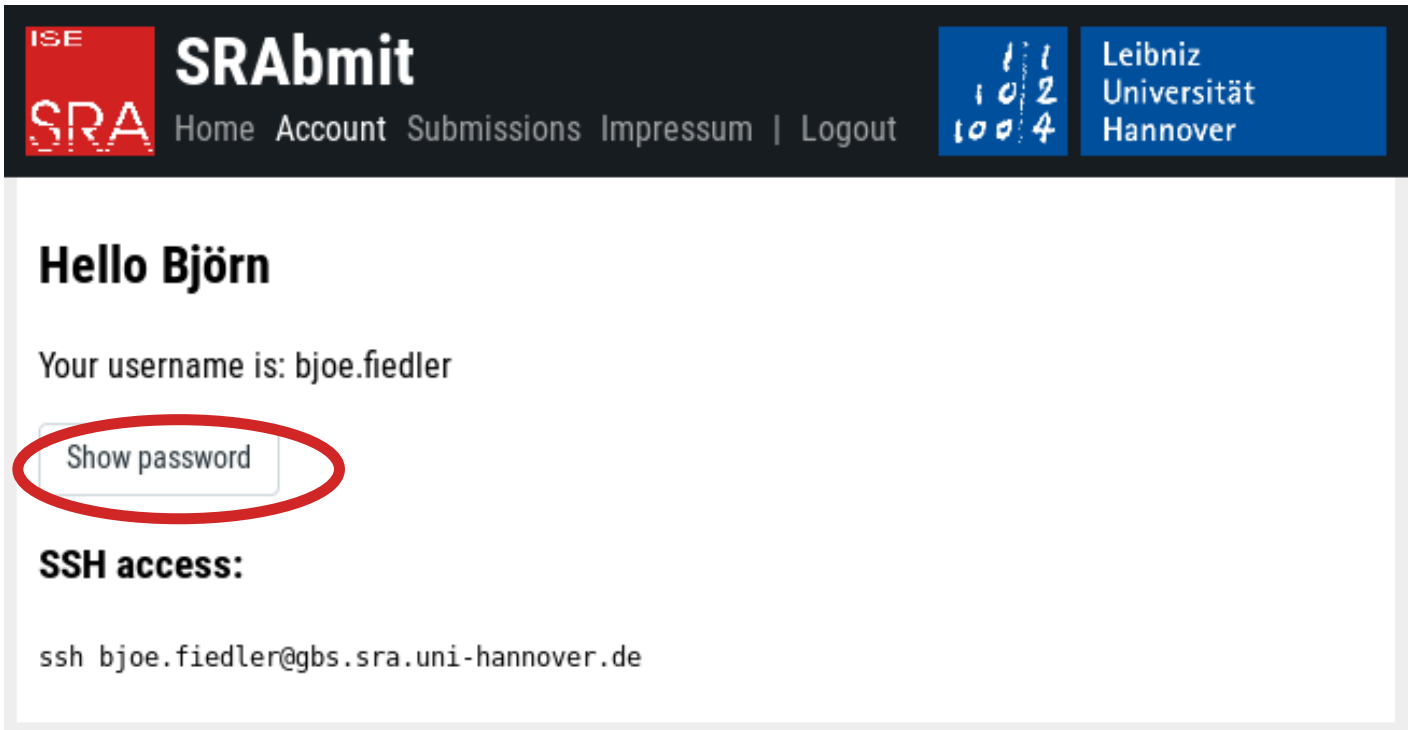
Account anlegen im Abgabesystem

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



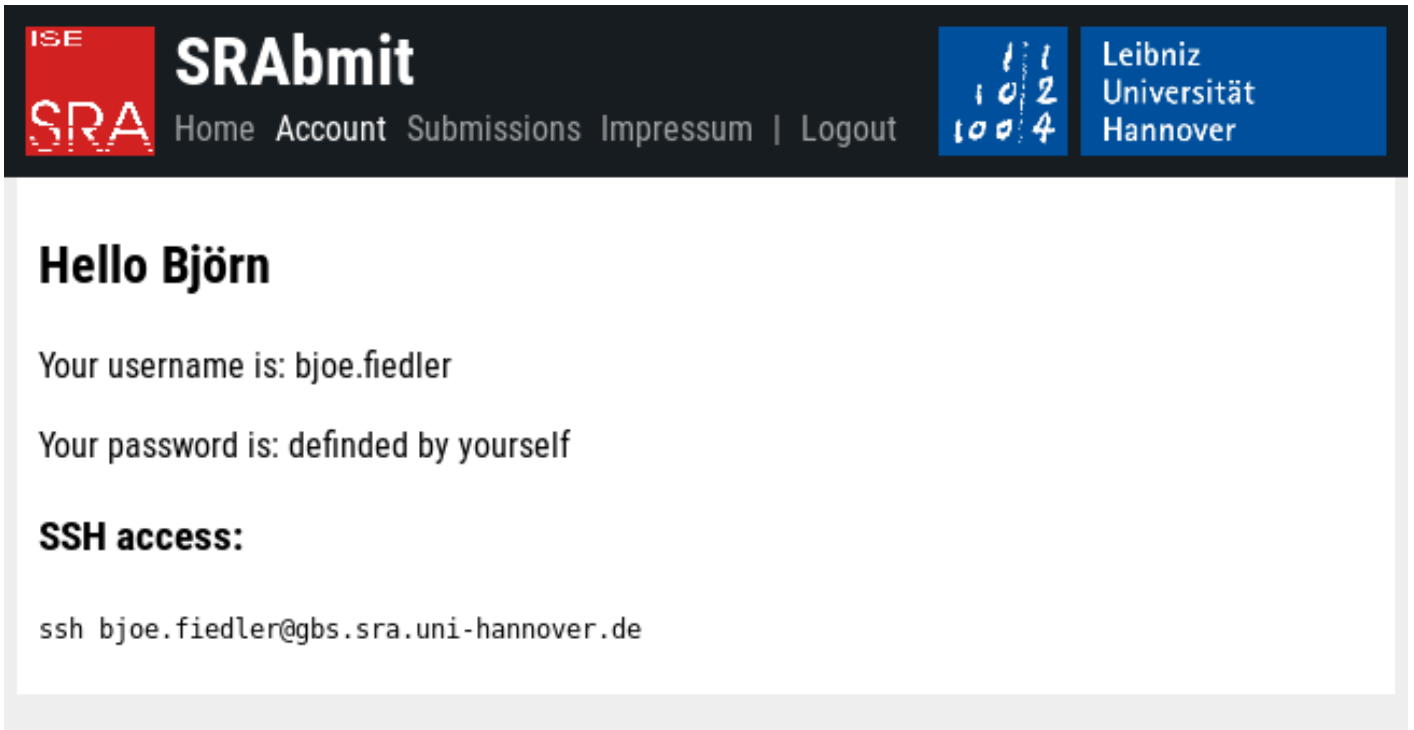
Account anlegen im Abgabesystem

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



Shell – Die Schnittstelle zum Rechner

Die Shell

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

bash 2019-09-06

Die Shell

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

```
$ echo foobar
```

bash 2019-09-06

Die Shell

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

```
$ echo foobar
foobar
$
```

bash 2019-09-06

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 2019-09-06

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 2019-09-06

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

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 2023-10-16

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

ssh 2023-10-16

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

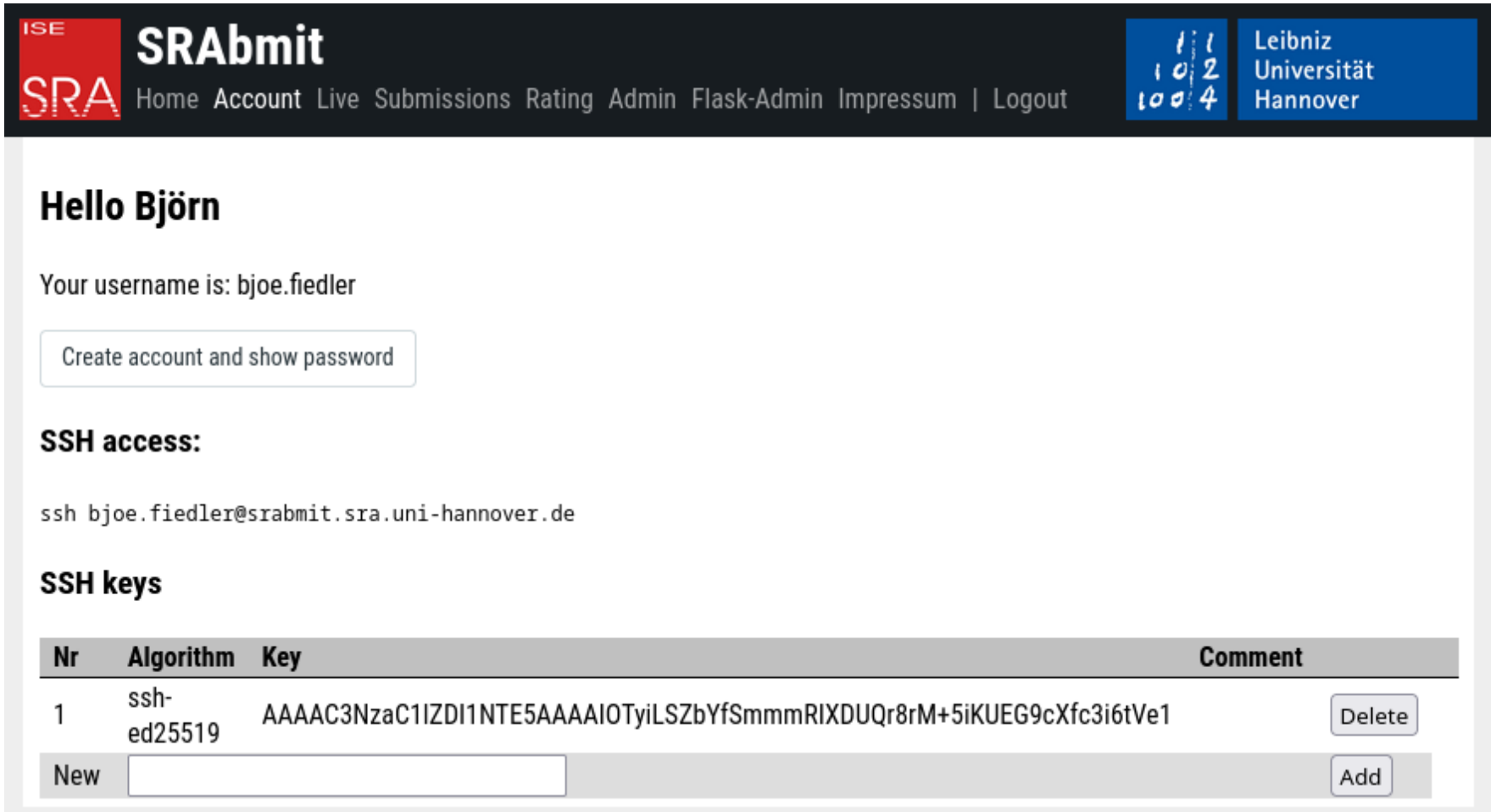
Eigenschaften

- 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



Passwort ändern

- Passwörter sollten geheim sein

↪ Das vorgegebene Passwort ändern

```
$ passwd
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
```

- `passwd` schreibt keine Zeichen auf die Konsole
- Das neue Passwort sicher aufbewahren

passwd 2019-09-06

Vereinfachungen für den Alltag mit SSH

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

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.

ssh_config 2023-10-13

Vereinfachungen für den Alltag mit SSH

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

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: Trage diese Informationen in die lokale Konfigurationsdatei ein.
- Wo ist die Datei: `~/.ssh/config`

ssh_config 2023-10-13

Vereinfachungen für den Alltag mit SSH

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

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: Trage diese Informationen in die lokale Konfigurationsdatei ein.
- Wo ist die Datei: `~/.ssh/config`

Beispiel: Definiere eine Verbindung *gbs*

```
Host gbs
  HostName gbs.sra.uni-hannover.de
  User bjoe.fiedler
  IdentityFile /path/to/my/sshkey
```

```
$ ssh gbs
```

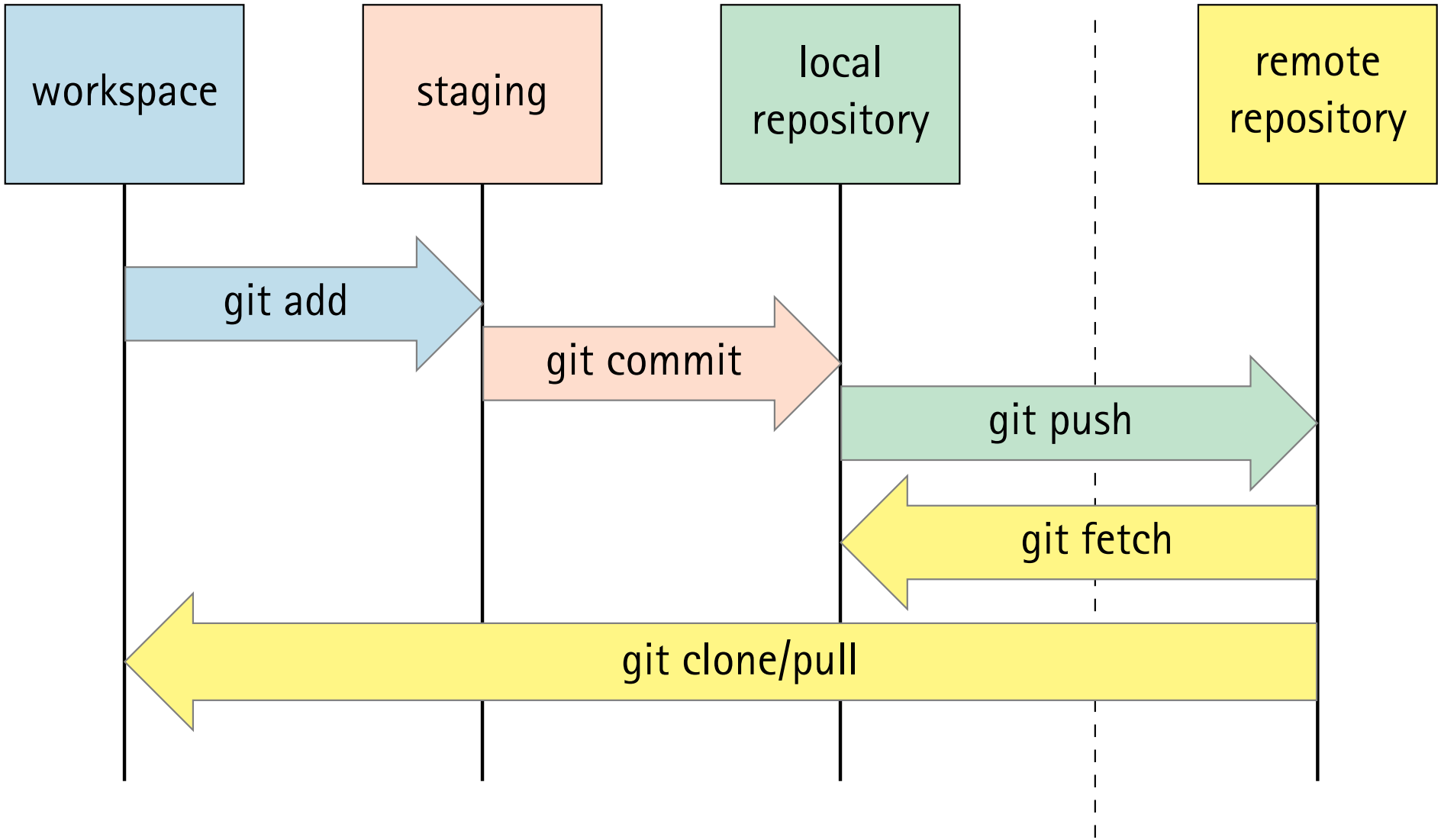
Git

- Git Versionsverwaltung
  - Versionierung von Dateien
  - dezentrale Struktur
  - meist offline
  - kann zentralisiert verwendet werden
  - Aufruf als `git cmd params...`
  - Speichert Zustand als Commit
  - Man kann zwischen Commits wechseln
  - Commits haben eine gerichtete Beziehung zueinander
  - Jedes Repository enthält die (gesamte) Historie

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

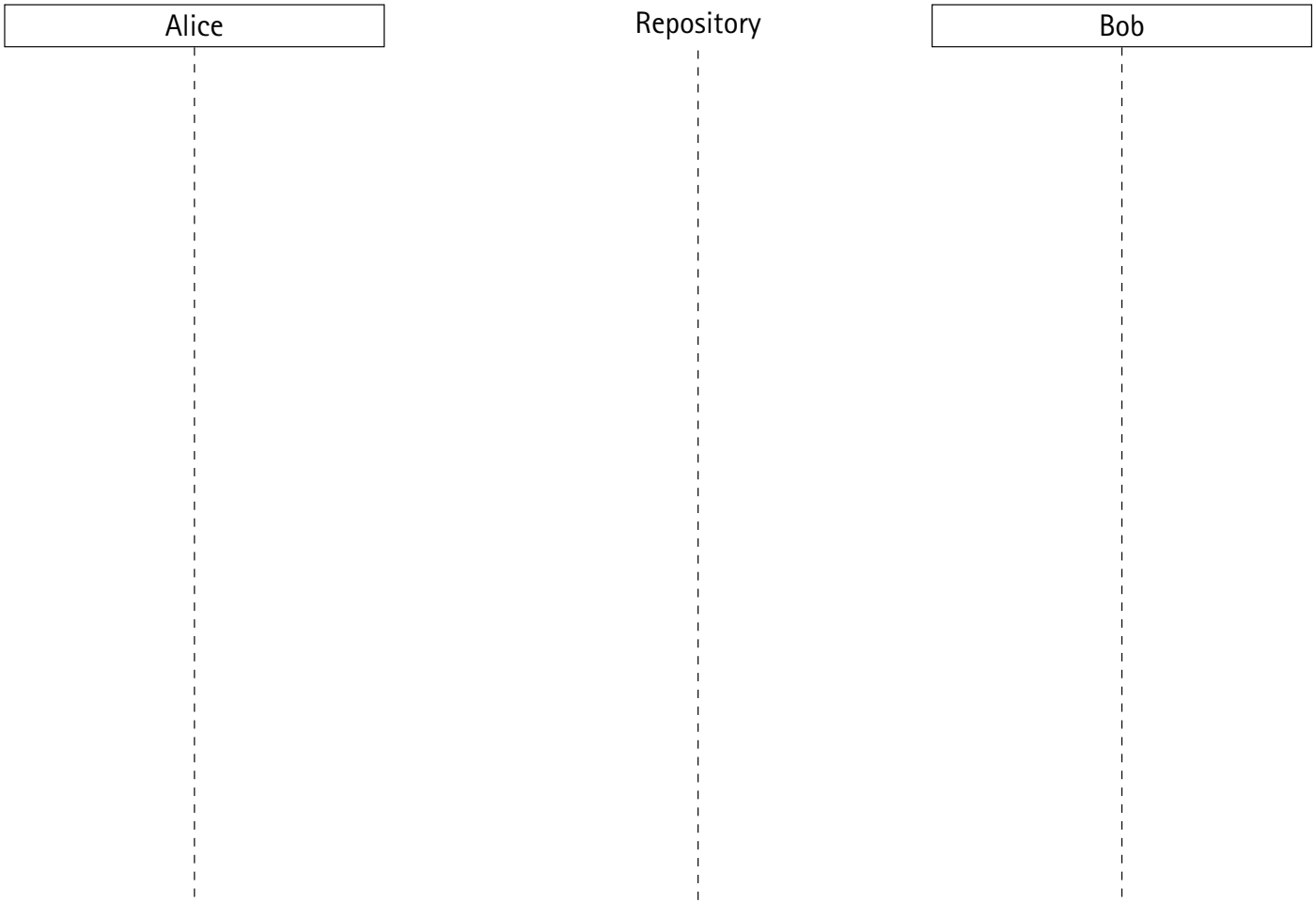
Git



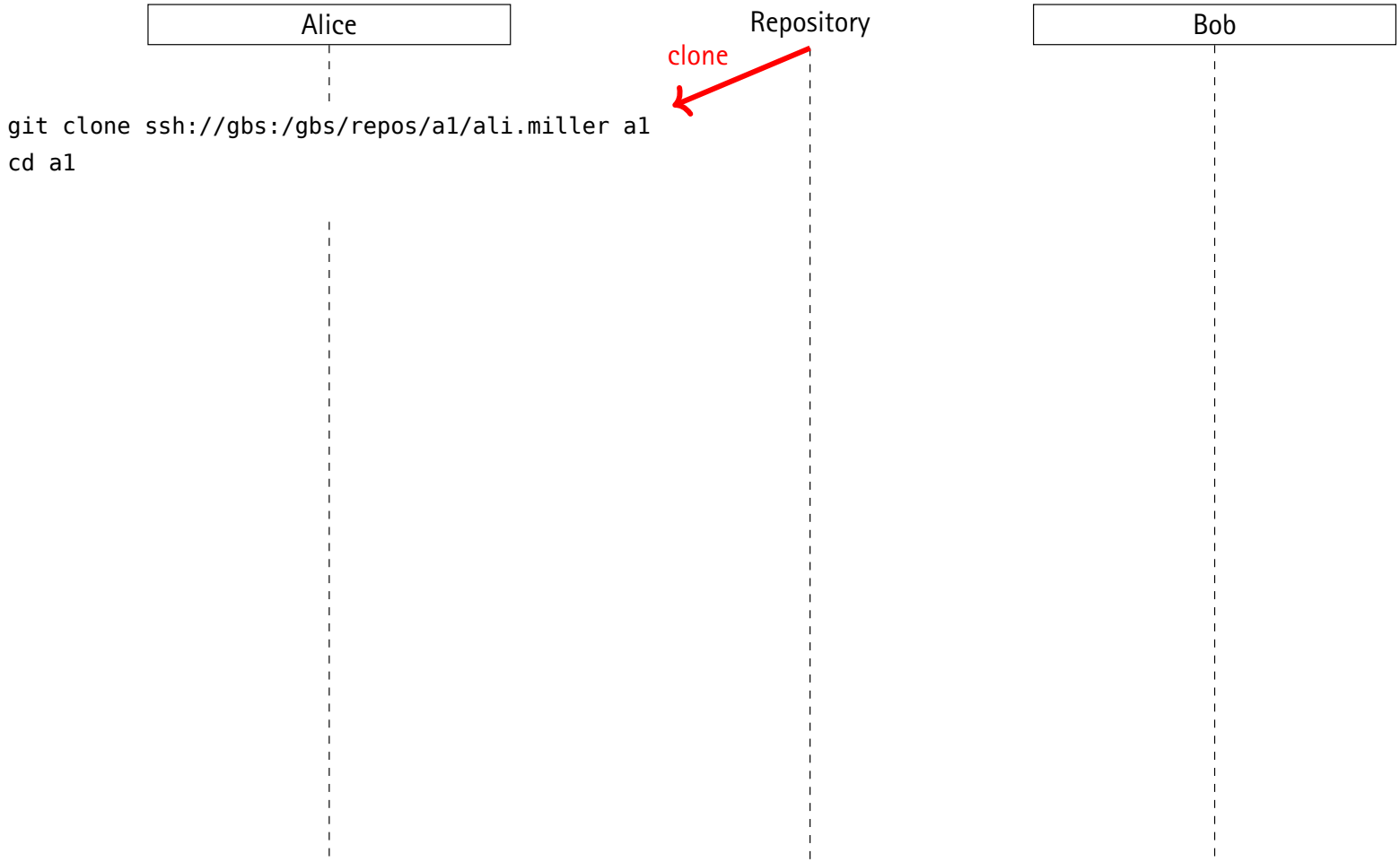
git 2020-10-06

Abgabeprotokoll

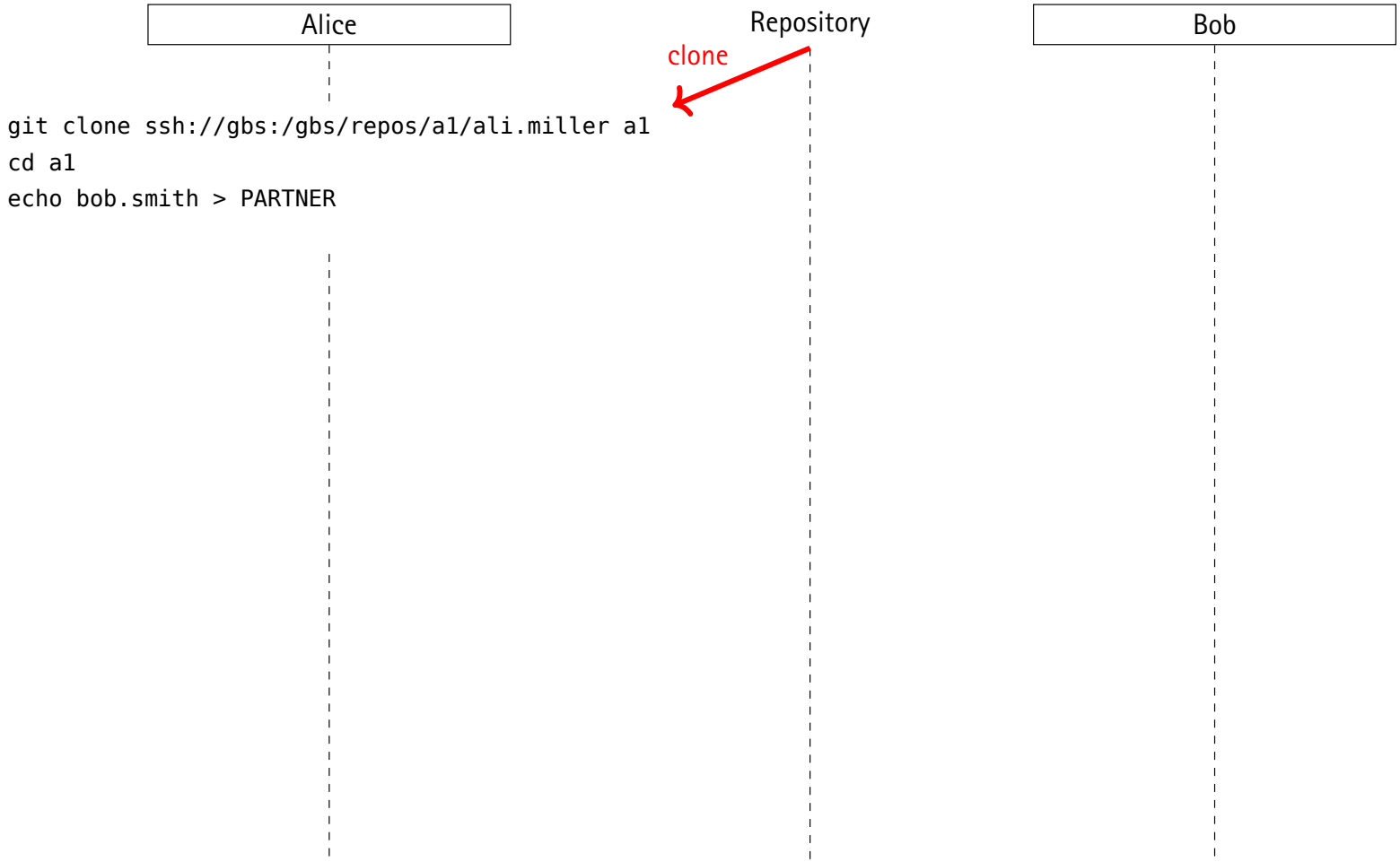
Abgabeprotokoll



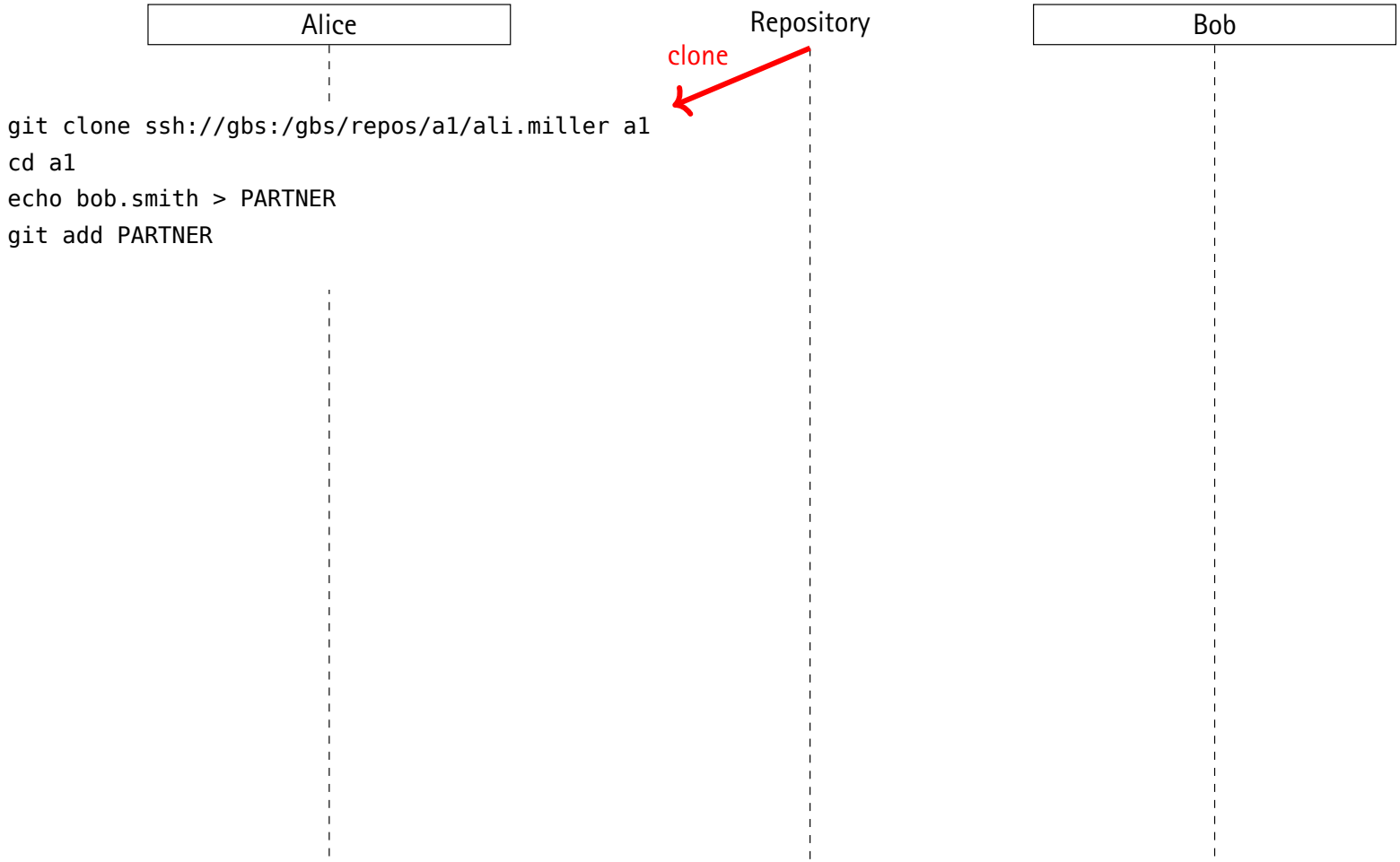
Abgabeprotokoll



Abgabeprotokoll

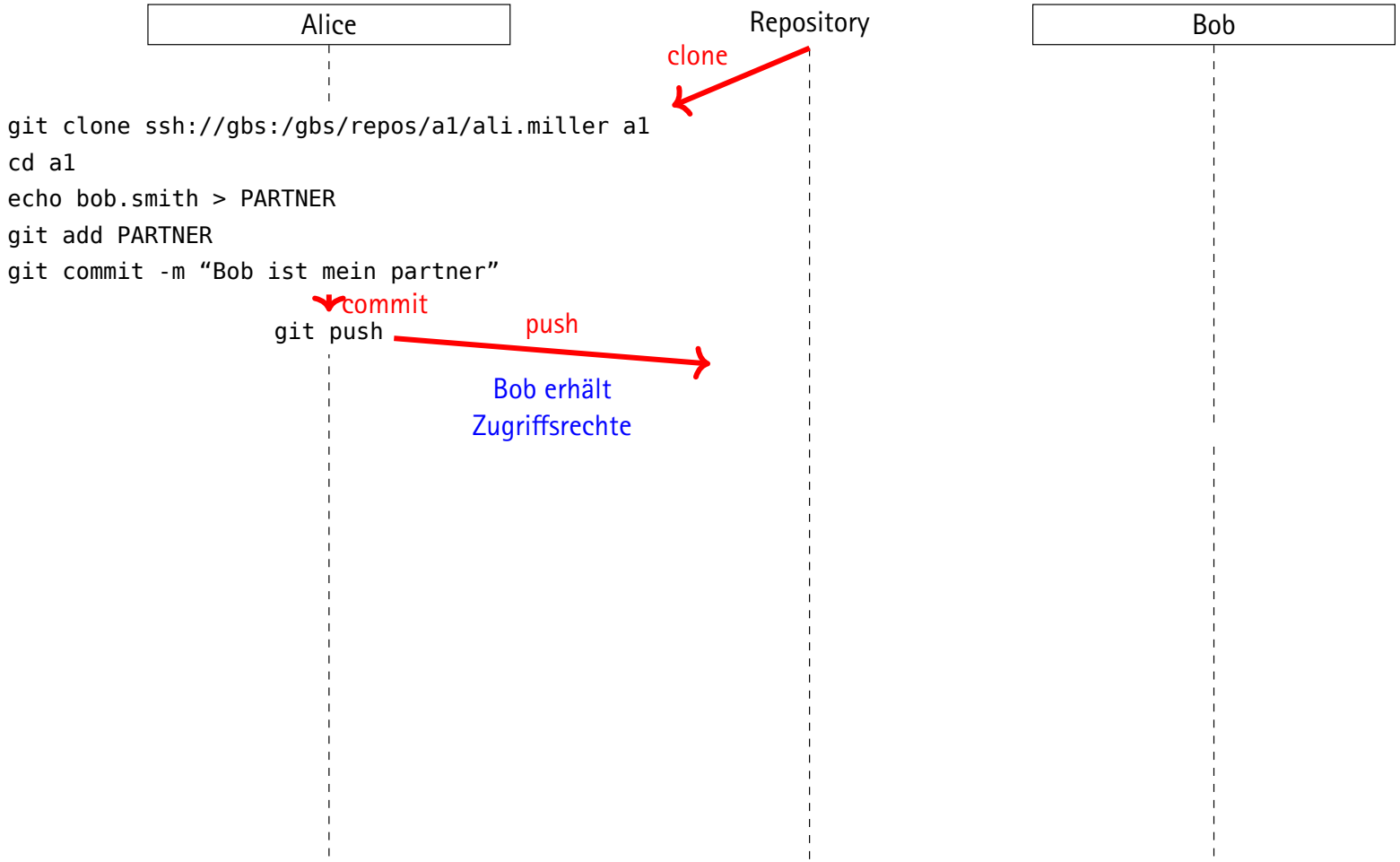


Abgabeprotokoll

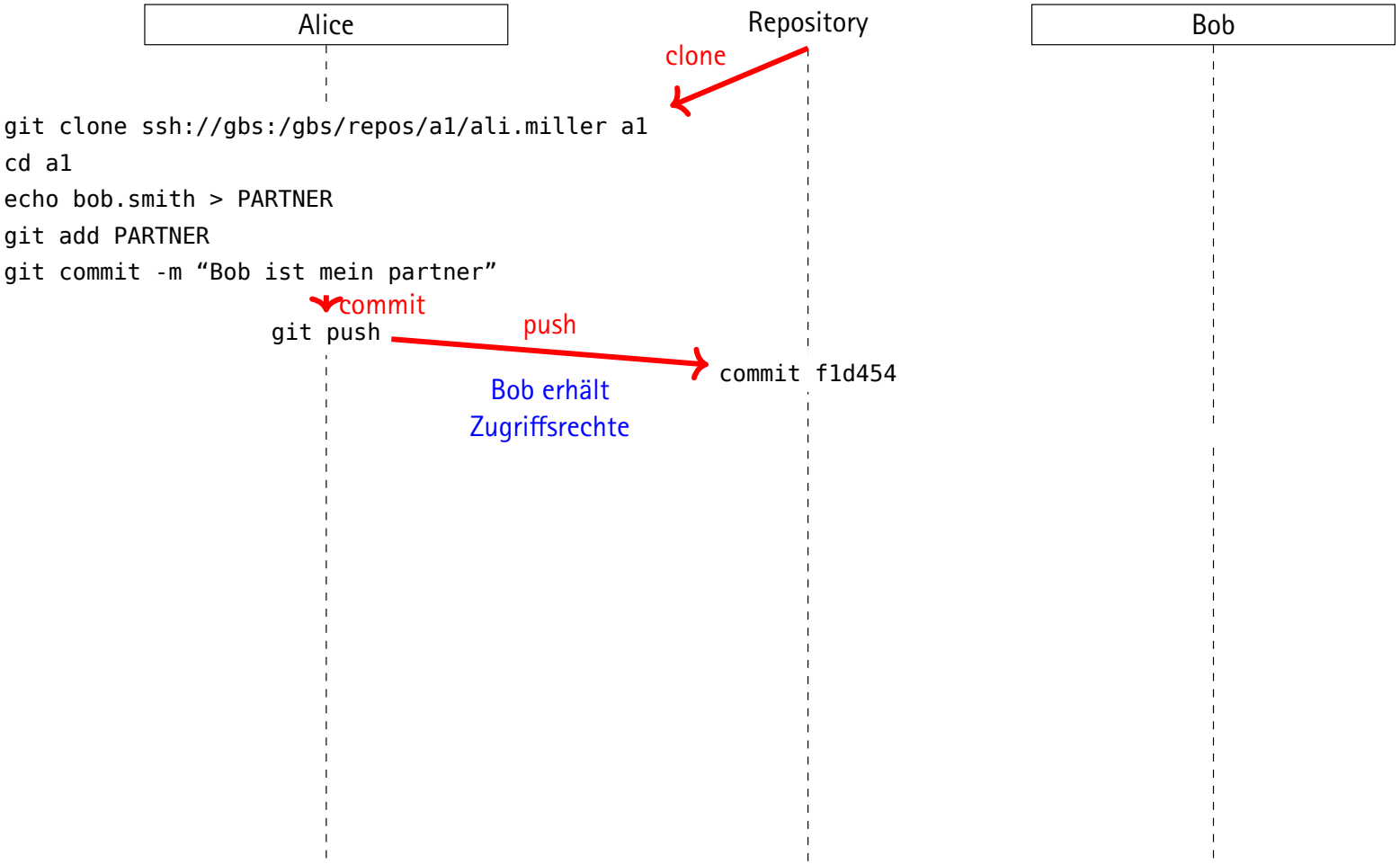


Abgabeprotokoll

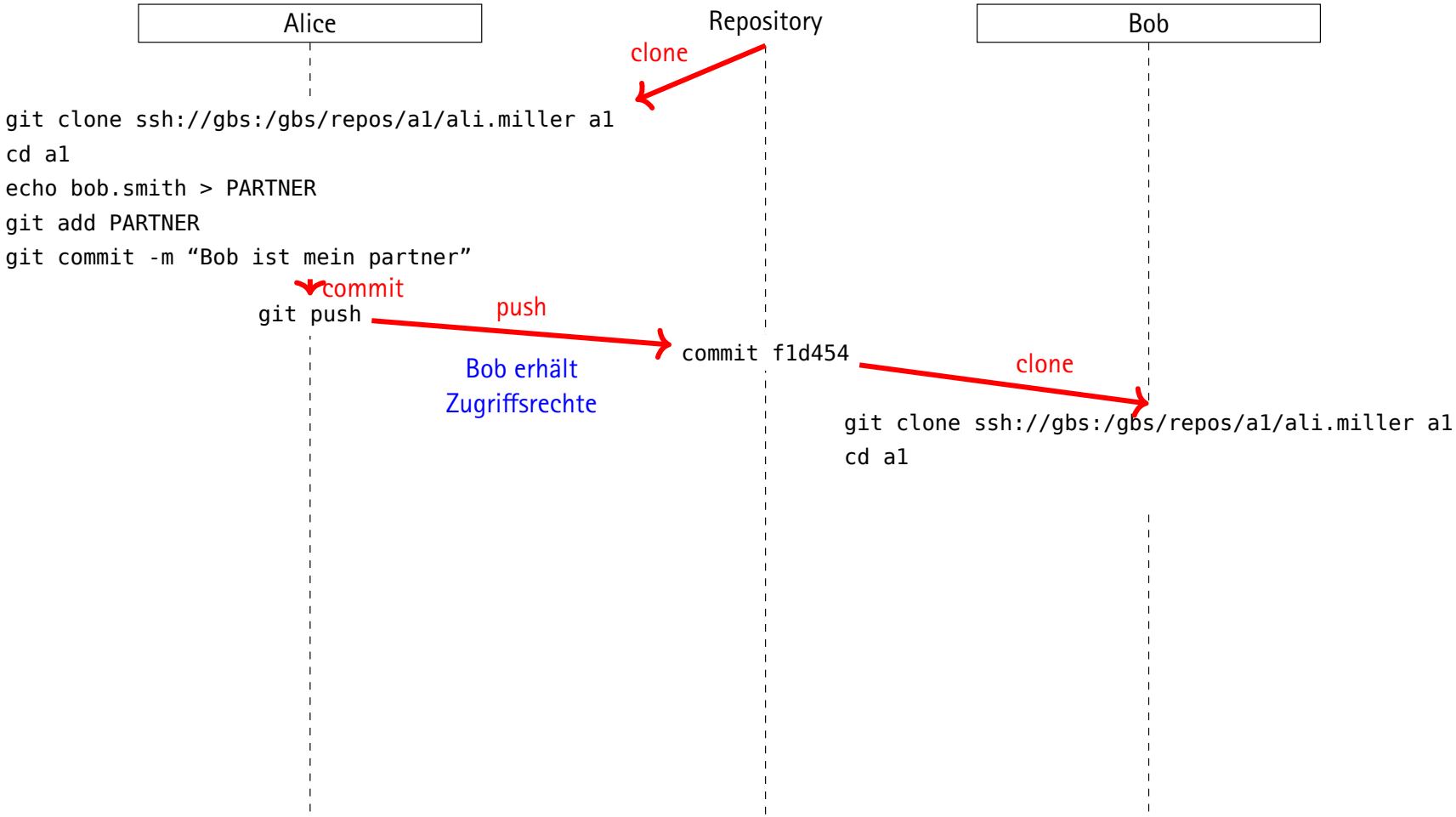
Abgabeprotokoll



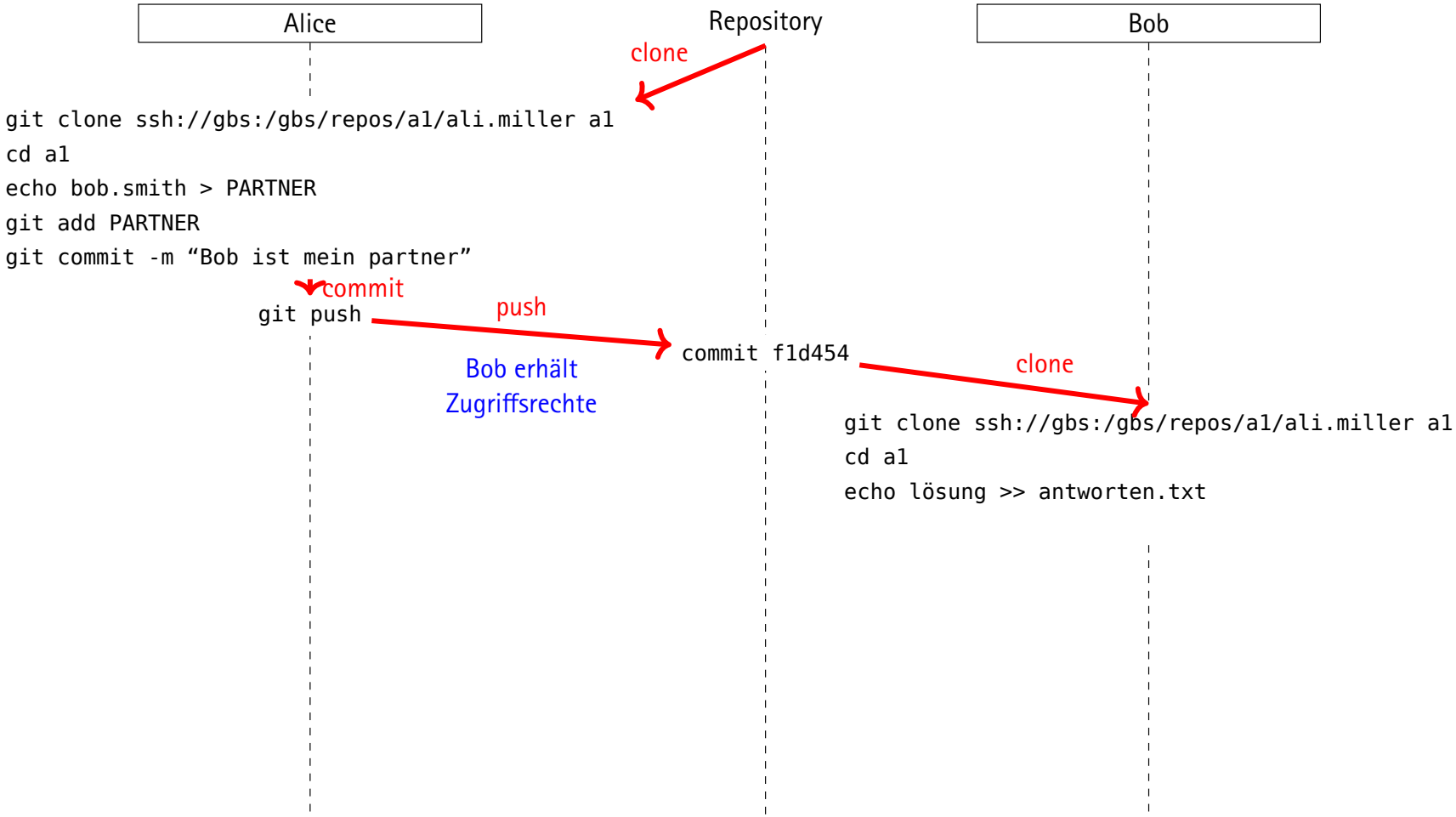
Abgabeprotokoll



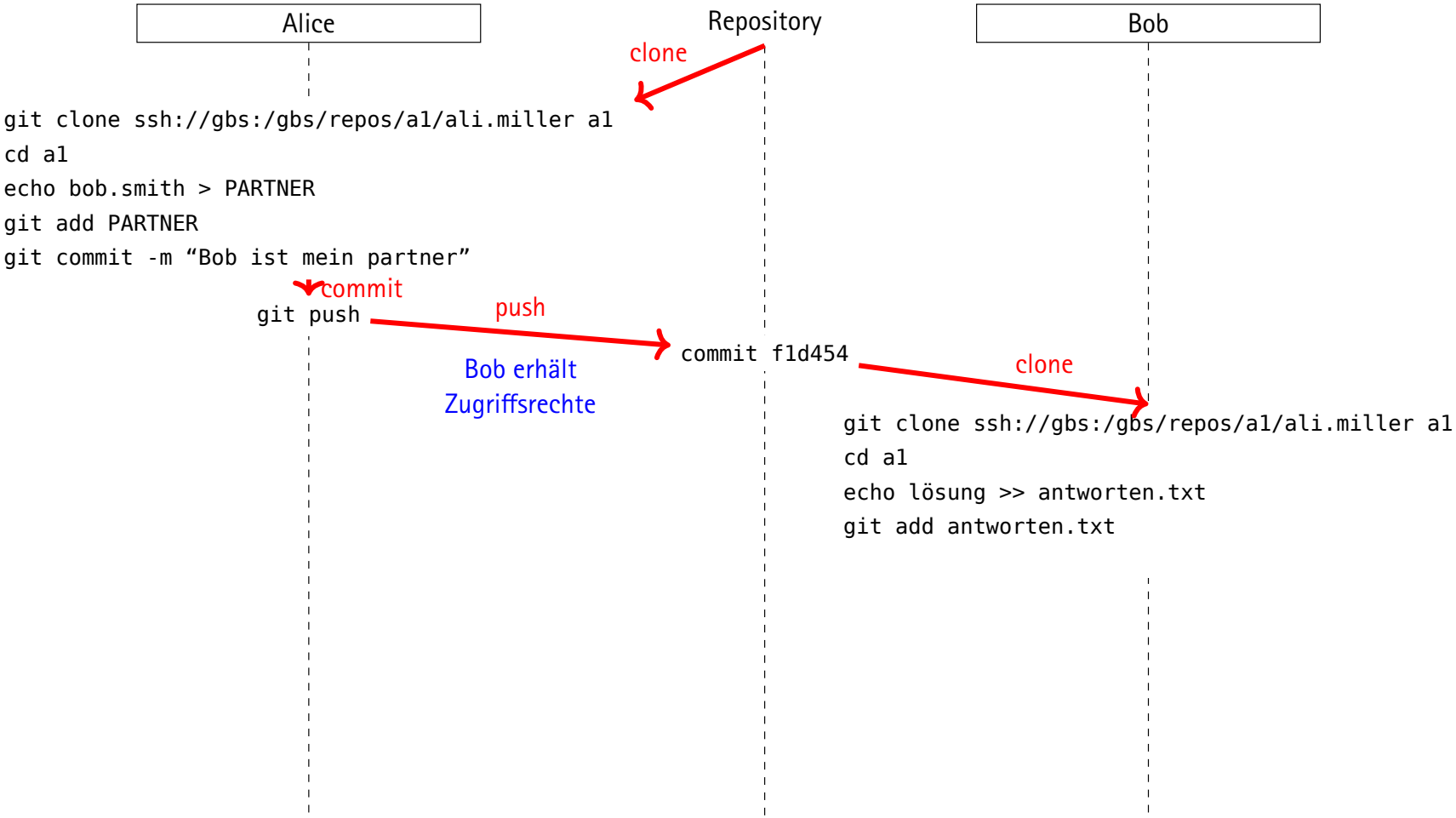
Abgabeprotokoll



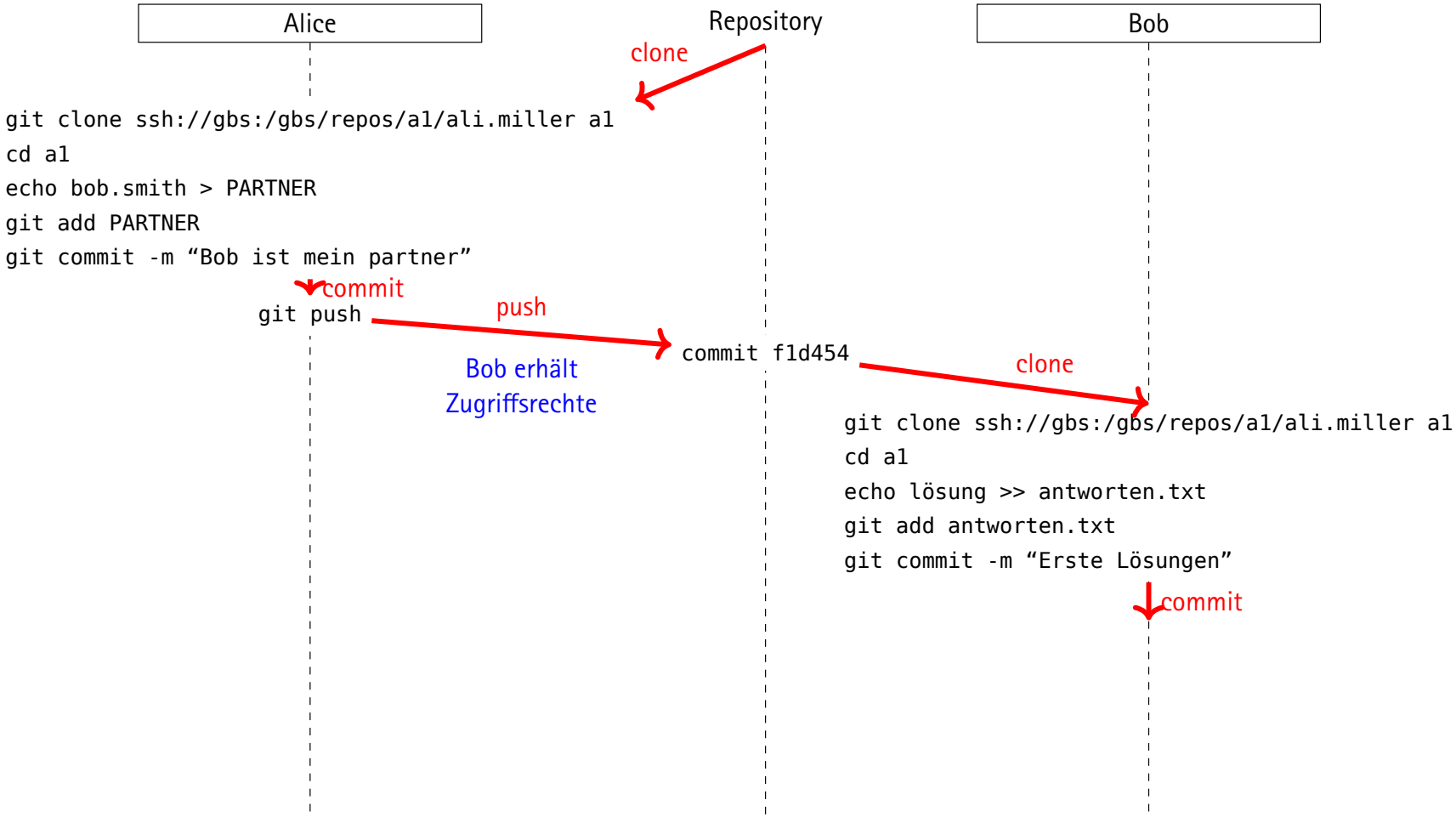
Abgabeprotokoll



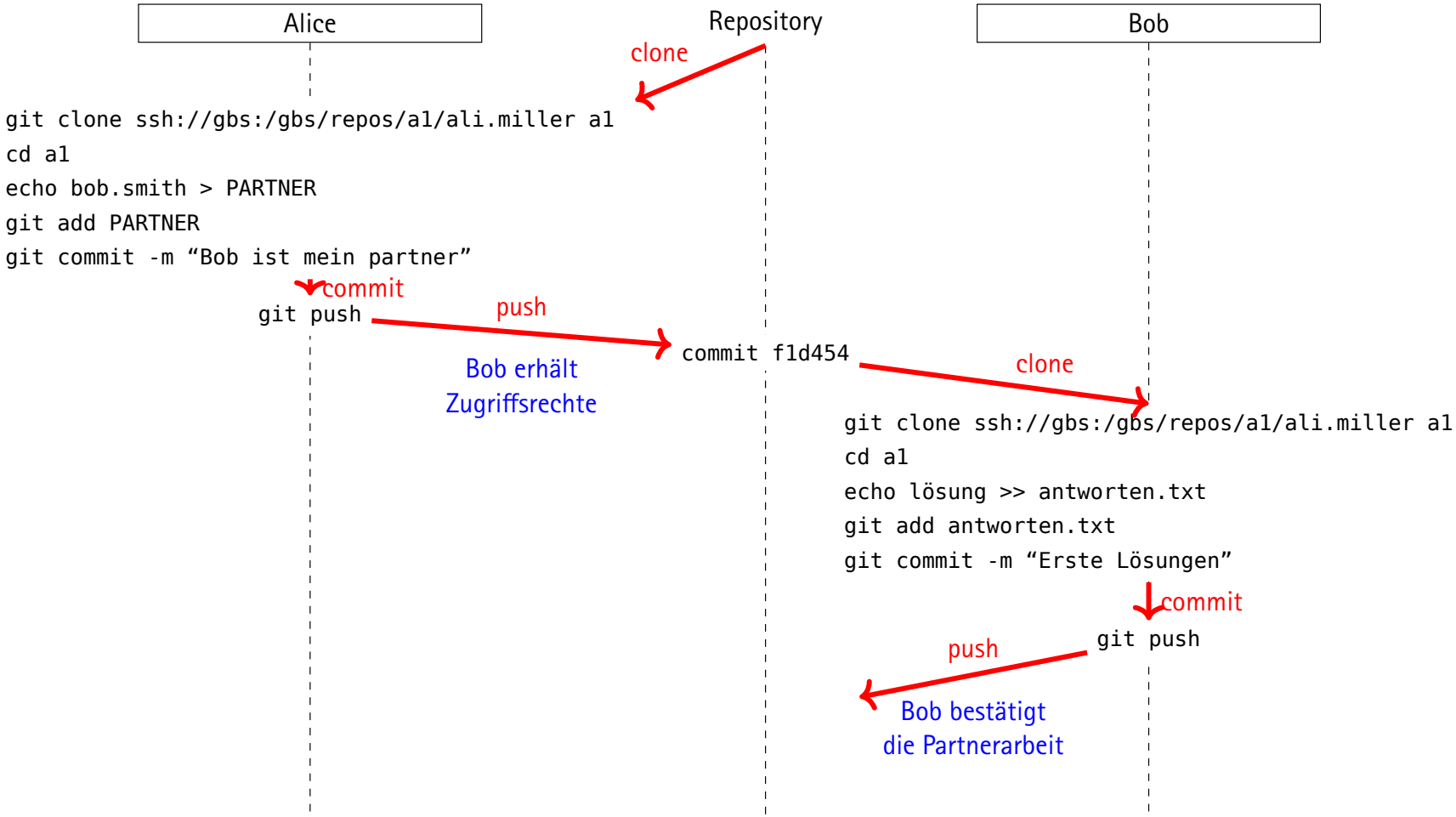
Abgabeprotokoll



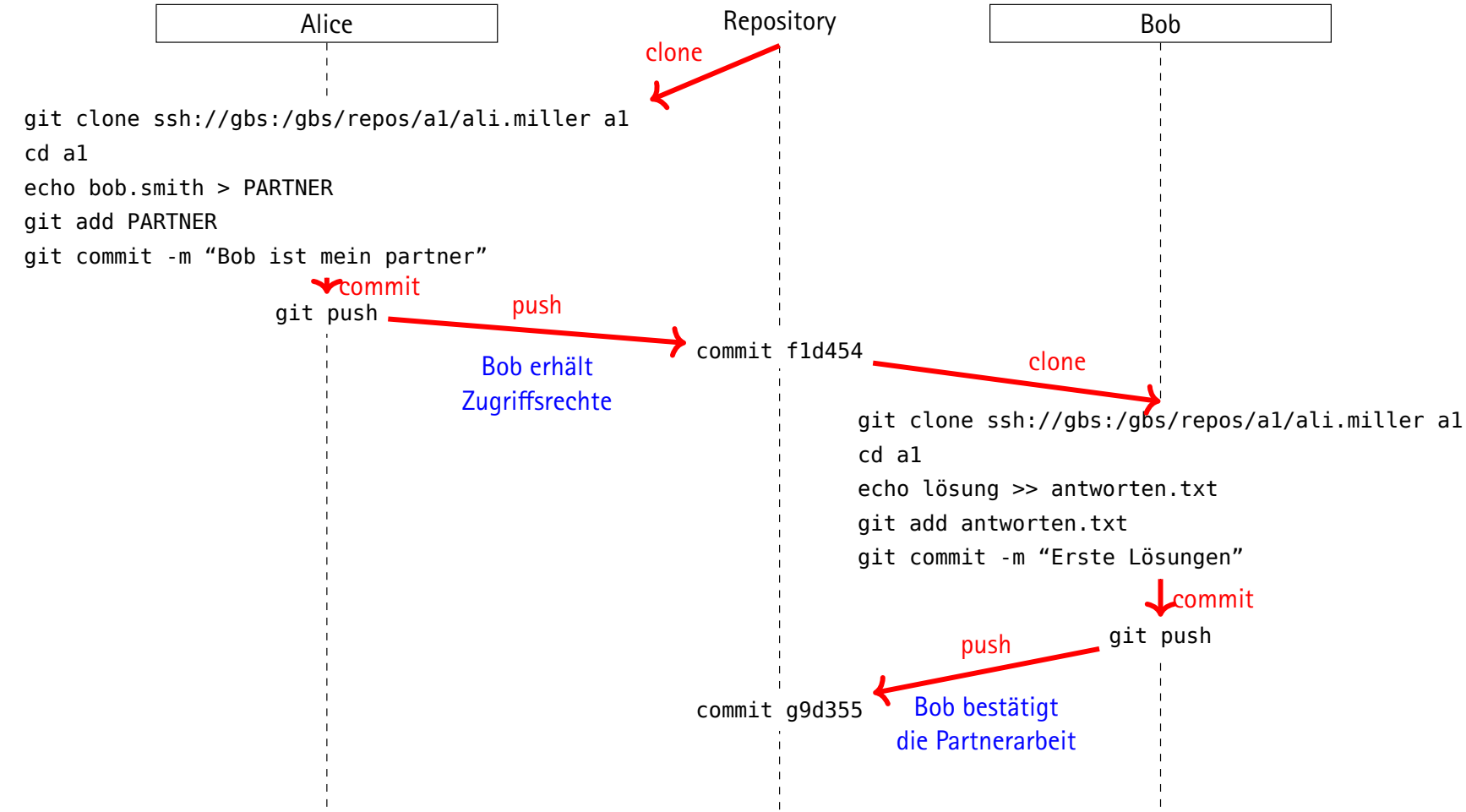
Abgabeprotokoll



Abgabeprotokoll



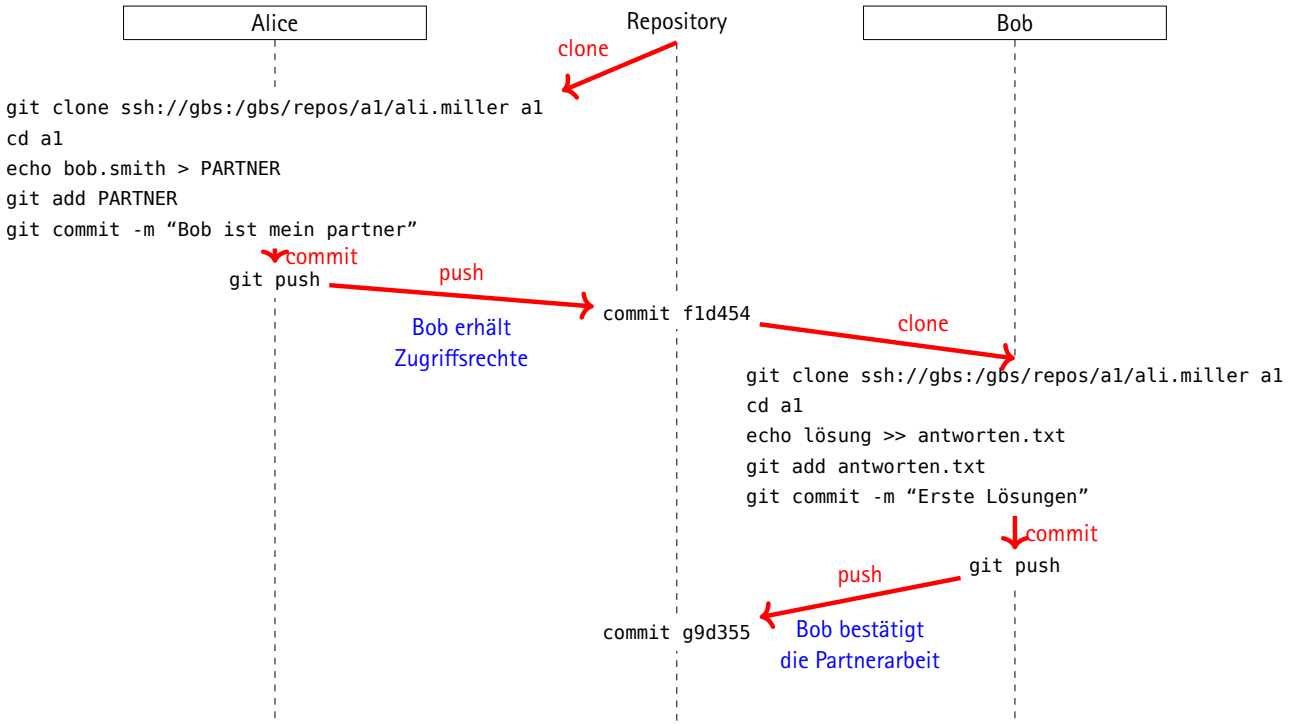
Abgabeprotokoll




```
git clone ssh://<user>@gbs.sra.uni-hannover.de:/gbs/repos/<aufgabe>/<alice.user> <zielordner>
```

Übung Abgabeprotokoll

- Abgabe mit Nachfolger in Namensliste
- Zwei Dummy-Abgaben
- Partner mit Namen vorn in der Liste ist Alice, hinten ist Bob
- `ssh://gbs:/gbs/repos/partnertest1/<username>/`
- `ssh://gbs:/gbs/repos/partnertest2/<username>/`



Hello World und Make

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

Make

- Makefile
  - Regeln zum Bauen von Objekten
  - Aufruf mit `make <target>`

Beispiel `$`

Make

- Makefile
  - Regeln zum Bauen von Objekten
  - Aufruf mit `make <target>`

Beispiel
```
$ make hello
cc hello.c -o hello
$
```

Make

- Makefile
  - Regeln zum Bauen von Objekten
  - Aufruf mit `make <target>`

Beispiel

```
$ make hello
cc hello.c -o hello
$ ls
hello.c hello
$
```

Make

- Makefile
  - Regeln zum Bauen von Objekten
  - Aufruf mit `make <target>`

Beispiel
```
$ make hello
cc hello.c -o hello
$ ls
hello.c hello
$ ./hello
Hello World
```

Verkettete Liste

Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

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

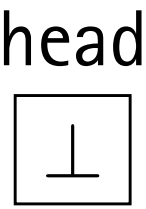
Aufgabenstellung: lilo — Einfach verkettete FIFO-Liste

- 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

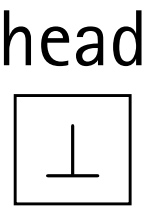
- Zielsetzung
  - Programmiere eine einfach verkettete Liste
  - Dynamische Speicherverwaltung und Umgang mit Zeigern
- Zu implementierende Schnittstelle:
  - `int list_append(list_t *list, int value)`:
    Fügt einen neuen, nicht-negativen Wert in die Liste ein.
    Erfolg: Rückgabe des eingefügten 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



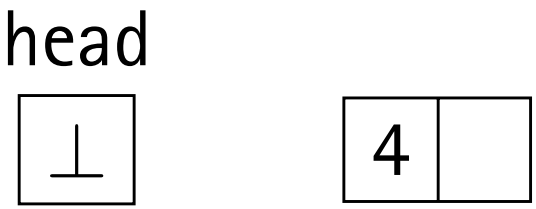
- Zustand: leer
- Operationen:

`lilo` - Funktionsweise



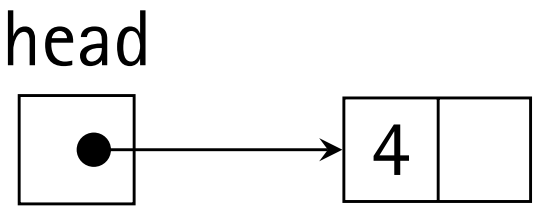
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



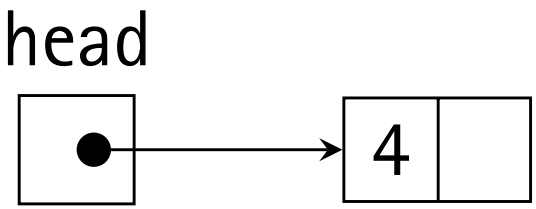
- Zustand: leer
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



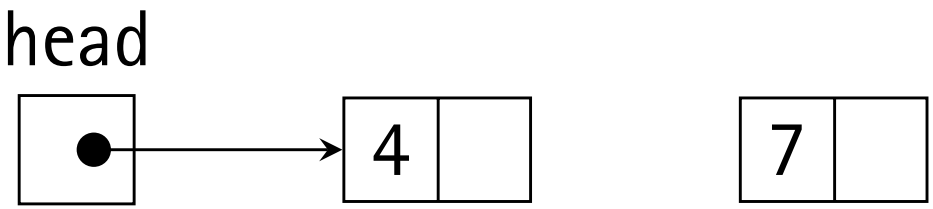
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)`

`lilo` - Funktionsweise



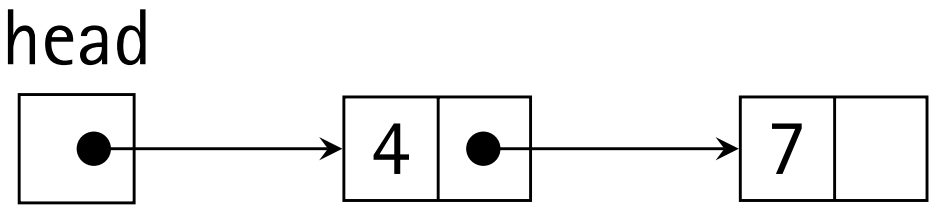
- Zustand: 1 Element
- Operationen:
  1. `list_append(head, 4)` =4

lilo - Funktionsweise



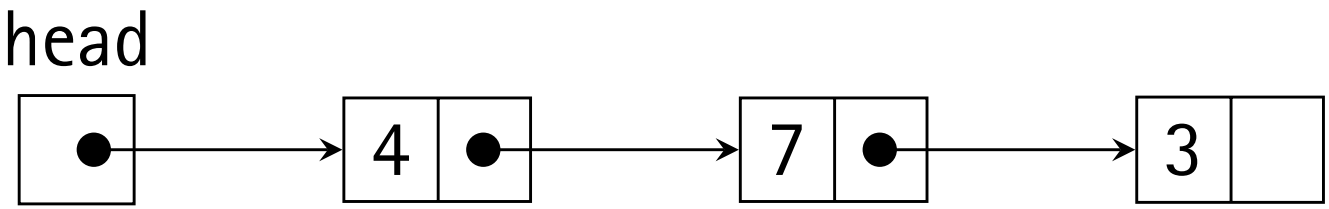
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)`

`lilo` - Funktionsweise



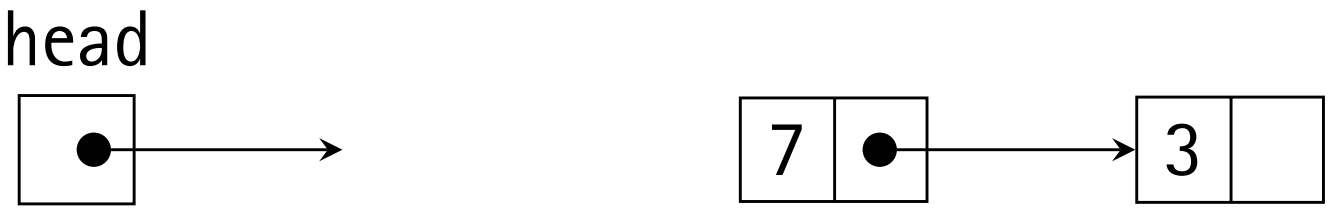
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7

lilo - Funktionsweise



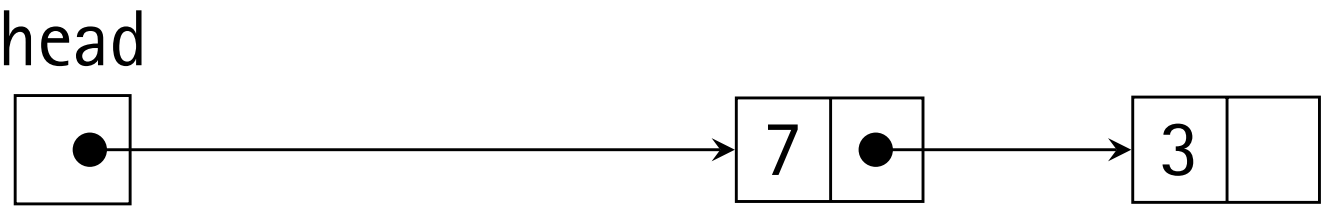
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3

lilo - Funktionsweise



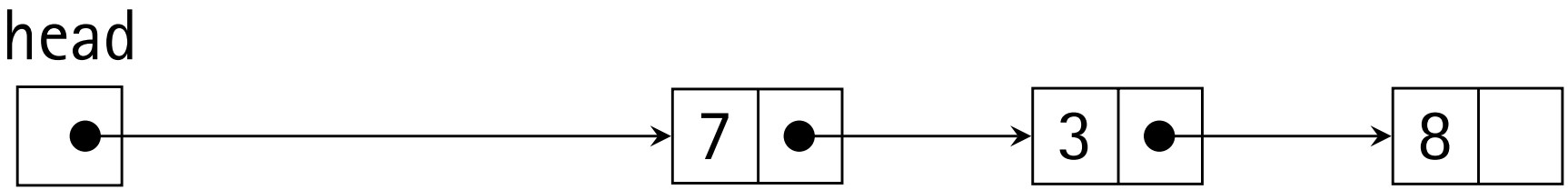
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

lilo - Funktionsweise



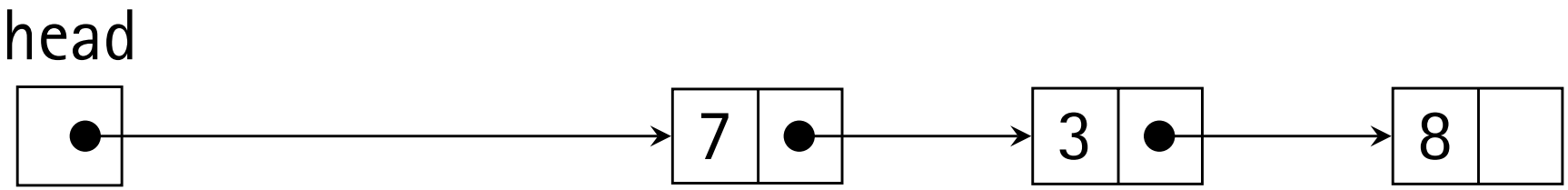
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4

`lilo` - Funktionsweise



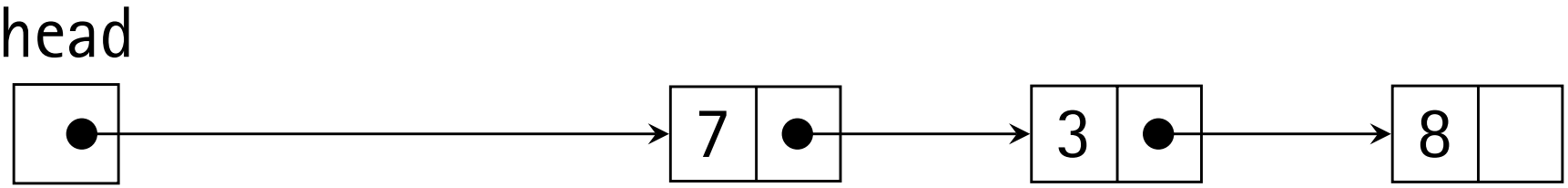
- Zustand: 3 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8

`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

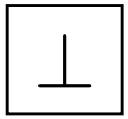
- Zustand: 2 Elemente
- Operationen:
  1. `list_append(head, 4)` =4
  2. `list_append(head, 7)` =7
  3. `list_append(head, 3)` =3
  4. `list_pop(head)` =4
  5. `list_append(head, 8)` =8
  6. `list_append(head, 8)` =-1
  7. `list_pop(head)` =7
  8. `list_append(head, -5)` =-1

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

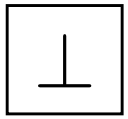
head



- 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

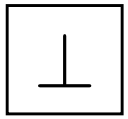
head



- 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;
```

Backup

Texteditoren

- Wahl des Texteditors ist beliebig
- Auf dem Server vorhanden:
  - emacs
  - vim
  - ne
  - nano

Typische Verwendung

```
nano <Datei>
```

Hello.c in nano

Hello World in nano öffnen

```
$ nano hello.c
```

Hello.c in nano

Hello World in nano öffnen

```
$ nano hello.c
```

```
GNU nano 3.1              code_examples/hello.c              Modified

#include <stdio.h>

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




^G Get Help ^O Write Out ^W Where Is ^K Cut Text  ^J Justify
^X Exit     ^R Read File ^\ Replace  ^U Uncut Tex ^T To Spell
```

Befehlsaufbau Allgemein

Im Terminal kann man jetzt Befehle eingeben:

```
$ echo
```

`echo` gibt den übergebenen Text unverändert wieder aus.

parameter 2019-09-06

Befehle mit einem Parameter

Dazu brauchen wir Parameter:

Muster

<Befehl> <Parameter>

```
$ echo foo
foo
```

Mehrere Parameter

Also einmal mit zwei Wörtern:

```
$ echo foo bar
foo bar
```

...und noch ein paar Leerzeichen mehr:

```
$ echo foo    bar
foo bar
```

parameter 2019-09-06

Quoting

```
Problem: $ echo foo    bar
foo bar
```

Mehrere Parameter werden durch Leerzeichen getrennt – wie viele Leerzeichen, spielt keine Rolle.
Durch *Quoting* kann man die Spezialbedeutung von Leerzeichen (und anderen Sonderzeichen) aufheben – der Text, der in Anführungszeichen steht, wird als ein einziger langer Parameter interpretiert.

```
Lösung: $ echo 'foo    bar'
foo    bar
```

Optionen

Je nach Befehl können auch verschiedene Optionen angegeben werden, um das Verhalten des Befehls zu verändern:

Muster

<Befehl> <Optionen> <Parameter>

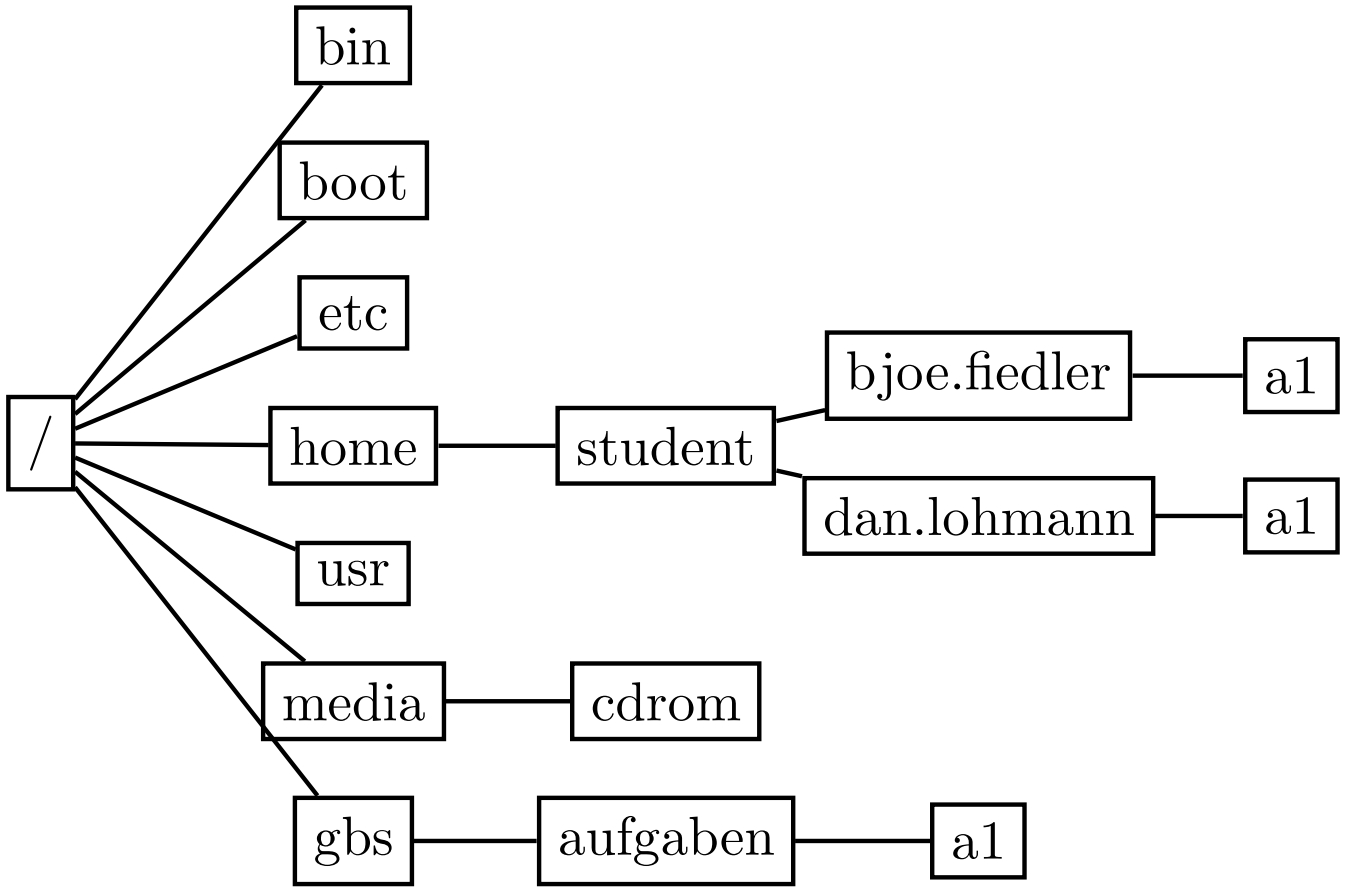
Bei `echo` bewirkt die Option `-n`, dass nach der Ausgabe keine neue Zeile angefangen wird.

```
$ echo -n foo
foo $ _
```

parameter 2019-09-06

Herumklettern im Dateisystembaum

Aufbau des Verzeichnisbaums



cd-pwd 2019-09-09

Wo sind wir überhaupt?

pwd

`pwd` (*print working directory*) gibt das aktuelle Verzeichnis aus.

```
$ pwd
/home/student/bjoe.fiedler
```

Verzeichniswechsel

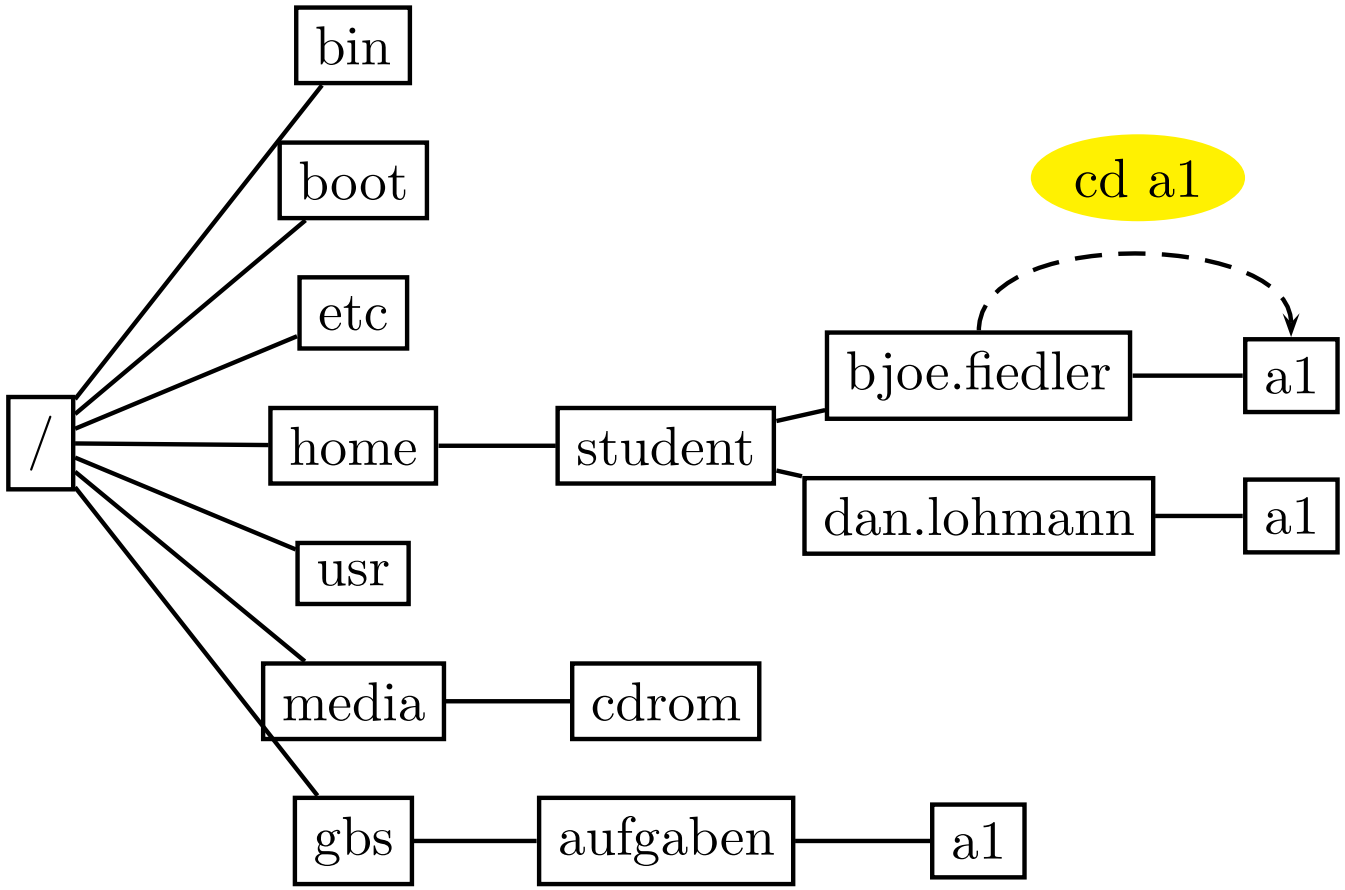
cd

Mit `cd` *(= change directory)* wechselt man zwischen Verzeichnissen.

Beispiele

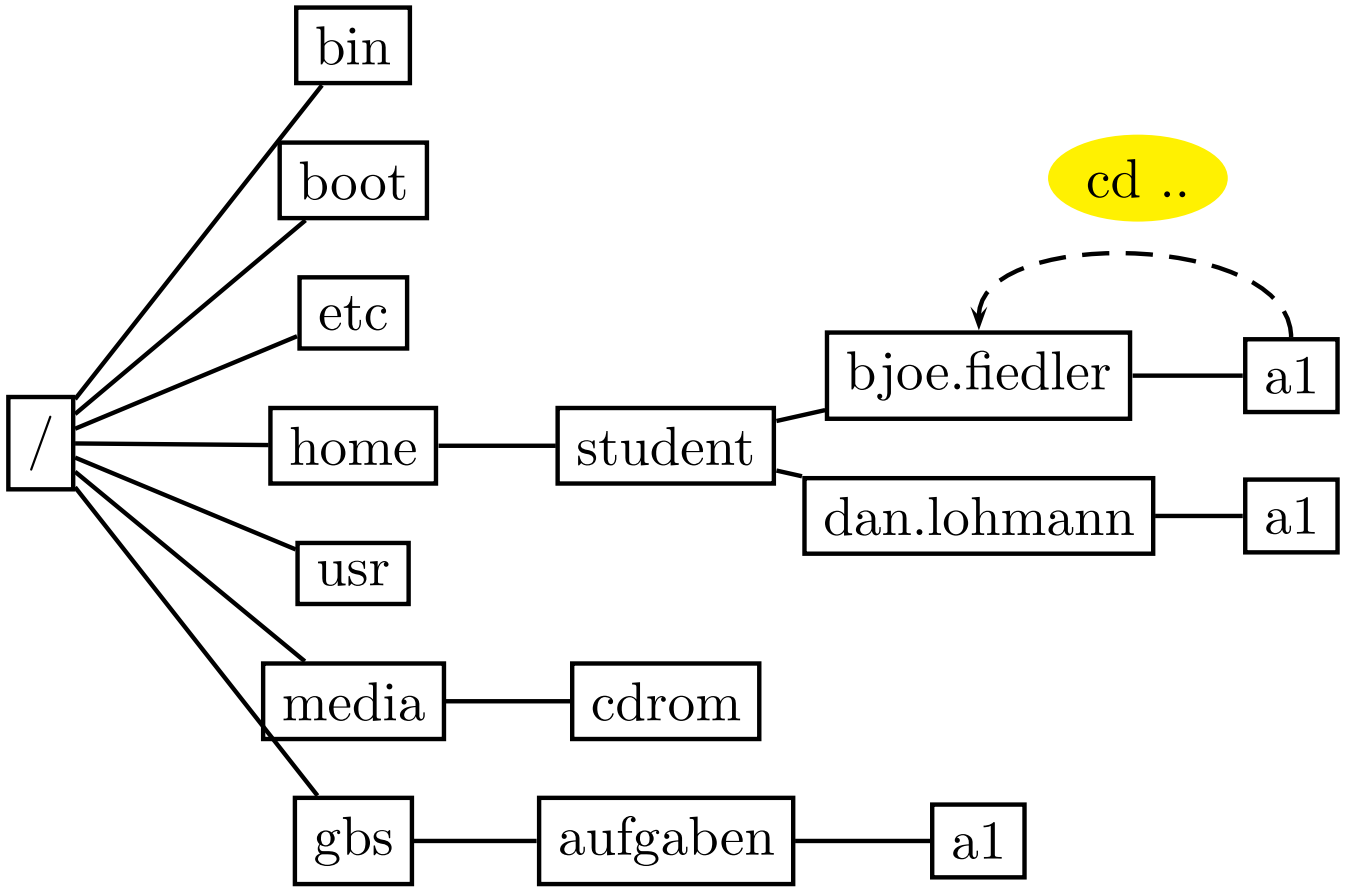
`cd bin`	–	wechselt in das Unterverzeichnis 'bin' im aktuellen Verzeichnis (*relativer Pfadwechsel*)
`cd /bin`	–	geht in das Verzeichnis 'bin' unterhalb des Root-Verzeichnisses `/` (*absoluter Pfadwechsel*)
`cd ..`	–	wechselt eine Verzeichnisebene nach oben
`cd ../testy`	–	wechselt eine Verzeichnisebene nach oben und darin in das Verzeichnis 'testy'
`cd`	–	geht in das *Home*-Verzeichnis
`cd -`	–	geht in das letzte besuchte Verzeichnis

Verzeichniswechsel



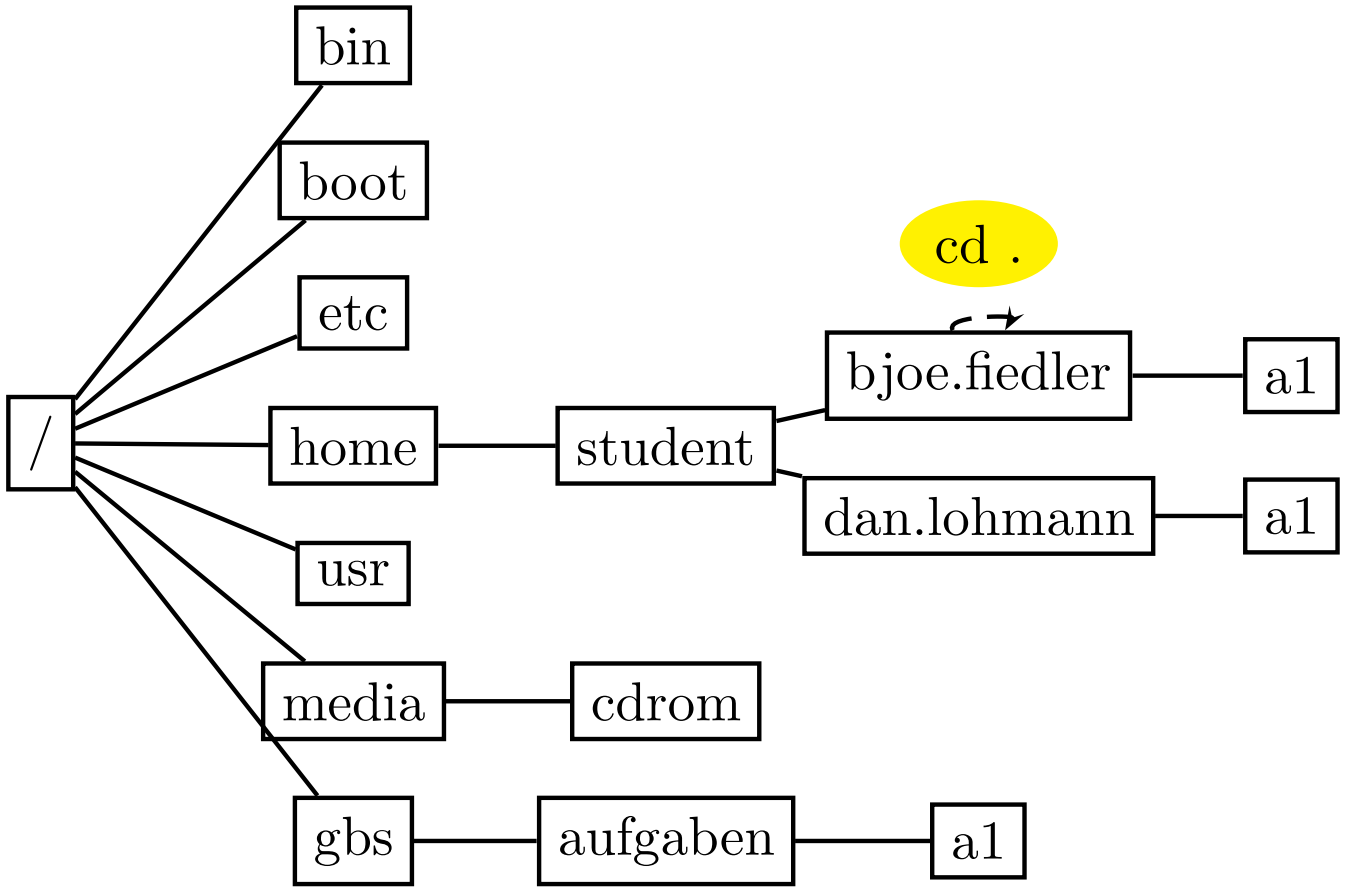
cd-pwd 2019-09-09

Verzeichniswechsel ins übergeordnete Verzeichnis



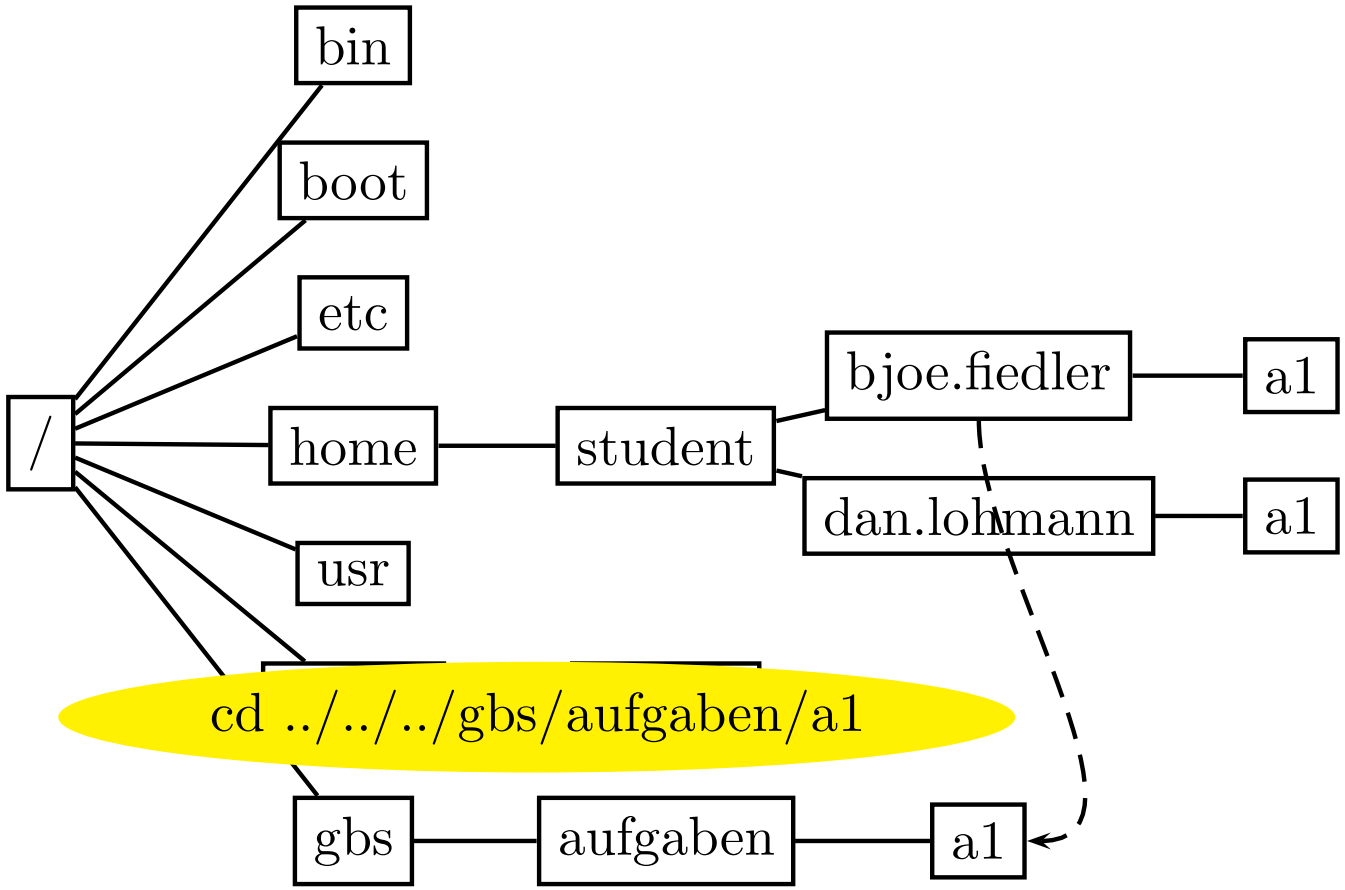
cd-pwd 2019-09-09

Verzeichniswechsel



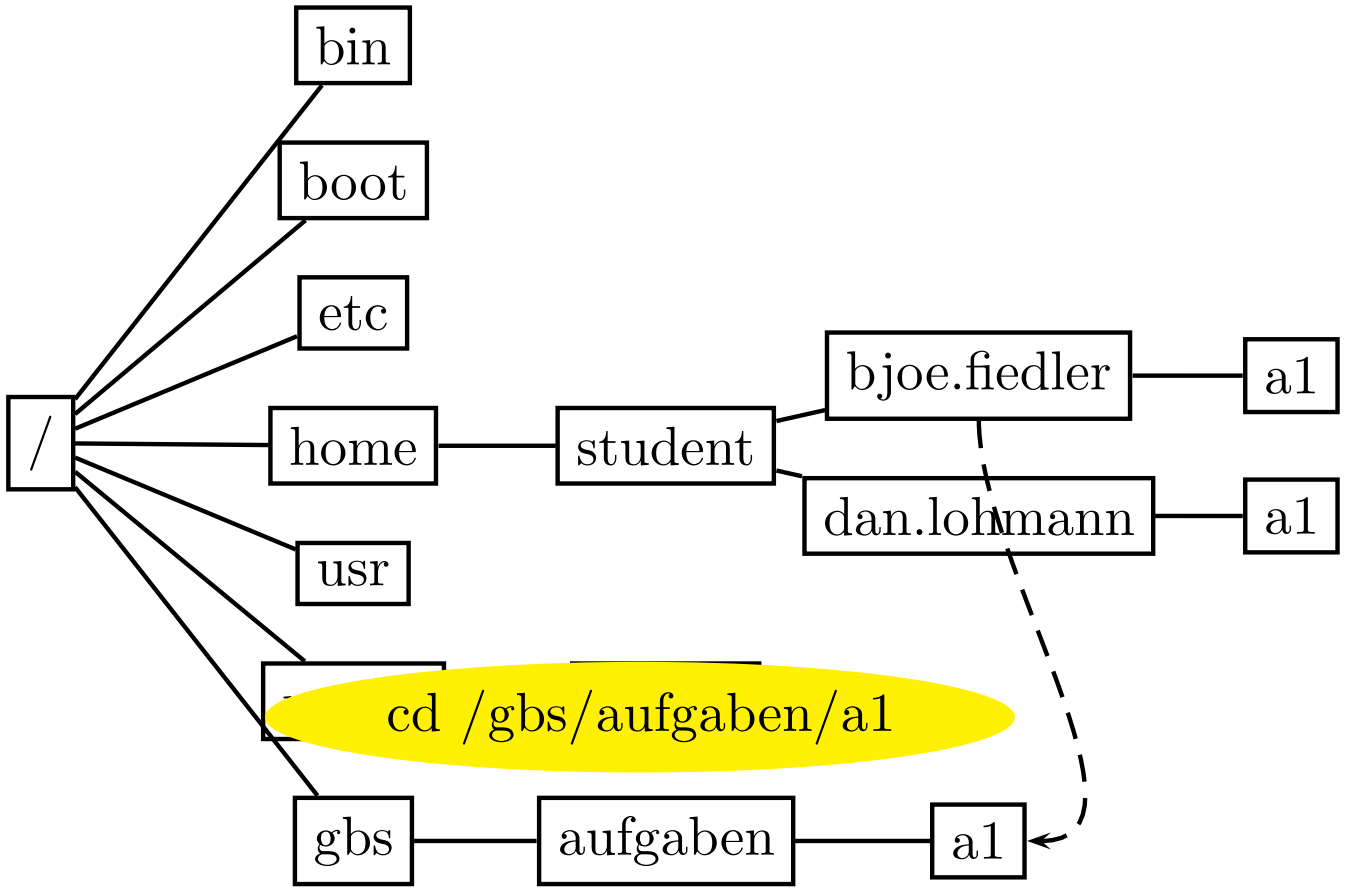
cd-pwd 2019-09-09

Relativer Verzeichniswechsel



cd-pwd 2019-09-09

Absoluter Verzeichniswechsel



cd-pwd 2019-09-09

Home

- Jeder Benutzer besitzt ein *Home*-Verzeichnis (`/home/student/<userlogin>`):
  - Es steht nur begrenzter Speicherplatz zur Verfügung
  - Dort liegen Konfigurationen und Nutzdaten
  - Kurzschreibweise fürs *Home*-Verzeichnis: ∼ (*Tilde-Zeichen*)

Der Befehl `quota` zeigt, wie viel Speicherplatz zur Verfügung steht.

cd-pwd 2019-09-09

Inhalte aufzeigen

Verzeichnisinhalt

ls

`ls` listet den Inhalt eines Verzeichnisses auf.

Beispiele

`ls`	–	listet Inhalt des aktuellen Verzeichnisses auf
`ls verzeichnis`	–	listet Inhalt des angegebenen Verzeichnisses auf
`ls -d verzeichnis`	–	gibt Informationen zum angegebenen Verzeichnis aus (nicht aber den Inhalt)
`ls -l`	–	ausführliche Verzeichnisauflistung (Dateigrößen, Rechte, Zeitstempel etc.)
`ls -a`	–	listet auch versteckte Dateien (Dateien, die mit einem Punkt beginnen) auf

Beispiele

Normales *ls* vs. *ls -a* vs. *ls -l*

```
$ ls
a.txt mein_bild.jpg
$ ls -a
.  ..  .bash_history  a.txt  mein_bild.jpg
$ ls -l
total 16
-rw-r--r-- 1 bjoern srastaff     0 Okt 2 11:29 a.txt
-rw-r--r-- 1 bjoern srastaff 12784 Okt 2 11:29 mein_bild.jpg
```

- *ls -a* zeigt wirklich alle Einträge des Verzeichnisses an!
- Einträge, die mit einem „.“ beginnen, werden normalerweise versteckt, z. B.:
  - „.“ ist immer das aktuelle Verzeichnis
  - „..“ ist immer das übergeordnete Verzeichnis
  - „.bash_history“ enthält z. B. Befehle, die früher eingegeben wurden
- *ls -l* zeigt ein Listenformat mit Infos, wie dem Änderungsdatum

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

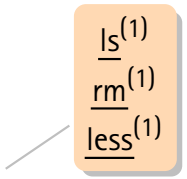
Man Sections

- 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 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