Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks
Reliable System Software

Übung Betriebssysteme (BS)

Tafelübung 4: Dateisystem

Sören Tempel

Wintersemester 2025

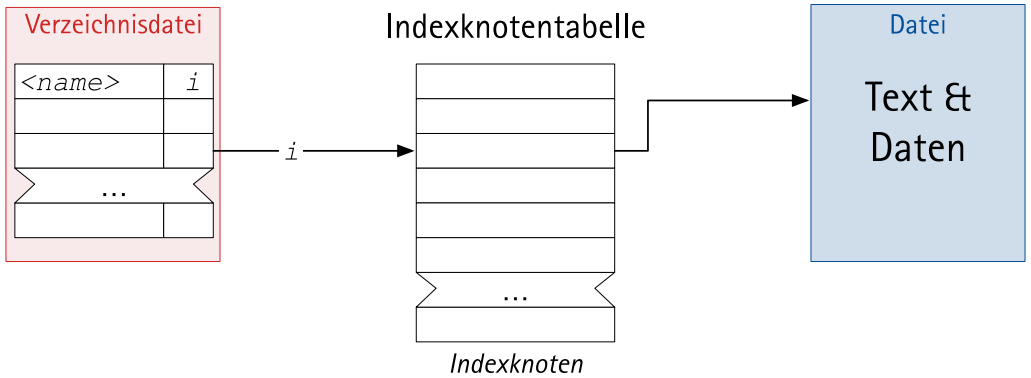
03t-datei 2025-11-21

Tafelübung 4: Dateisystem

- Rechteverwaltung im Dateisystem
- Dateisysteminteraktion
- Dateisysteminteraktion aus der Shell
- Wildcards
- Dateisysteminteraktion aus C
- Argumentparser
- crawl

Systemsicht: Datenstrukturen

Dateisystem (*file system*)



- Die **Indexknotentabelle** (*inode table*) ist ein statisches Feld (*array*) von Indexknoten (*inodes*) und die **zentrale Datenstruktur.**
  - Indexknoten: Deskriptor des Objektes (Datei)
  - Indexknotennummer: **Eindeutige Referenz** des Objektes
- Ein **Verzeichnis** (*directory*) ist eine **Abbildungstabelle**, es übersetzt symbolisch repräsentierte Namen in Indexknotennummern
  - Spezielle Datei der Namensverwaltung des Betriebssytems
- Eine **Datei** (*file*) ist eine abgeschlossene Einheit zusammenhängender Daten beliebiger Repräsentation, Struktur und Bedeutung.

Rechteverwaltung im Dateisystem

Wofür brauchen wir Rechte?

Technische Universität Braunschweig

Auf einem Mehrbenutzersystem wie der Referenzplattform `x1` mit vielen Anwendern sind sinnvolle Restriktionen essenziell:

- Man will seine *privaten Dokumente* vor fremden Augen schützen.
- Nur Administratoren sollen die *Konfiguration der Rechner* verändern können.
- *Kryptografie* setzt die bedingungslose Geheimhaltung der verwendeten Schlüssel voraus!

Wofür brauchen wir Rechte?

Auf einem Mehrbenutzersystem wie der Referenzplattform `x1` mit vielen Anwendern sind sinnvolle Restriktionen essenziell:

- Man will seine *privaten Dokumente* vor fremden Augen schützen.
- Nur Administratoren sollen die *Konfiguration der Rechner* verändern können.
- *Kryptografie* setzt die bedingungslose Geheimhaltung der verwendeten Schlüssel voraus!

Lösung: Der „Rechtsstaat“

- Für jede Datei und jedes Verzeichnis werden Berechtigungen vermerkt.
- Nur wer die entsprechenden Rechte besitzt, kann auf ein bestimmtes Objekt zugreifen.

Ein paar Grundlagen

Technische Universität Braunschweig

Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (hier: *mitarb*).
- Man kann in mehr als einer Gruppe sein.

Ein paar Grundlagen

Technische Universität Braunschweig

Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (hier: *mitarb*).
- Man kann in mehr als einer Gruppe sein.

Einteilung der Benutzer

Pro Objekt im Dateisystem sind die Rechte für drei Klassen von Benutzern gespeichert:

User	Diesem *Benutzer* „gehört“ die Datei / das Verzeichnis. Er darf Dateiberechtigungen vergeben.
Group	Die Datei / das Verzeichnis ist dieser *Gruppe* zugeordnet.
Others	Alle anderen.

Rechte auf Dateien

Technische Universität Braunschweig

Und welche Berechtigungen hat eine Datei / ein Verzeichnis?

- `ls -l` zeigt eine ausführliche Ausgabe.
- Dabei zeigt die erste Spalte die Rechte an.
- Die dritte und vierte Spalte geben den Eigentümer bzw. die Eigentümergruppe an.

```
$ ls -l
-rw-r--r-- 1 tempel mitarb 97 Oct 7 14:38 datei
-rwxr-x--- 1 tempel mitarb 84 Oct 12 14:39 programm
```

Rechte auf Dateien

Technische Universität Braunschweig

```
$ ls -l
- rw- r-- r--  1 tempel mitarb 97 Oct 7 14:38 datei
- rwx r-x ---  1 tempel mitarb 84 Oct 7 14:39 programm
d rwx r-x r-x  2 tempel mitarb 40 Oct 7 14:37 verzeichnis
```

Und was heißt das jetzt?

- Das erste Zeichen zeigt den Typ an (z. B. `d` für ein Verzeichnis oder `-` für normale Dateien).
- Die nächsten drei Zeichen zeigen die Rechte für den **User**.
- Das zweite Zeichentripel zeigt die Rechte für die **Group**.
- Und die verbleibenden drei Zeichen die Rechte für den Rest der Welt (**Others**).

Rechte auf Dateien

Technische Universität Braunschweig

```
$ ls -l
- rw- r-- r--  1 tempel mitarb 97 Oct 7 14:38 datei
- rwx r-x ---  1 tempel mitarb 84 Oct 7 14:39 programm
d rwx r-x r-x  2 tempel mitarb 40 Oct 7 14:37 verzeichnis
```

r? w? x?

- `r` = lesbar (read)
- `w` = schreibbar (write)
- `x` = ausführbar (execute)

ausführbare Dateien

`$ ./programm` führt Programm im aktuellen Verzeichnis aus

Rechte ändern

Technische Universität Braunschweig

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod      foo.bar
```

Das `mode`-Argument setzt sich zusammen aus drei Teilen

Rechte ändern

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g    foo.bar
```

Das `mode`-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?

u	Benutzer
g	Gruppe
o	Rest
a	alle

Rechte ändern

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+   foo.bar
```

Das `mode`-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?		**Welche Aktion?**	
u	Benutzer	+	Rechte geben
g	Gruppe	-	Rechte wegnehmen
o	Rest	=	Rechte setzen
a	alle		

Rechte ändern

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+rx foo.bar
```

Das `mode`-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?		**Welche Aktion?**		**Welche Rechte?**	
`u`	Benutzer	`+`	Rechte geben	`r`	lesen
`g`	Gruppe	`-`	Rechte wegnehmen	`w`	schreiben
`o`	Rest	`=`	Rechte setzen	`x`	ausführen
`a`	alle				

Änderungen im Rechtesystem

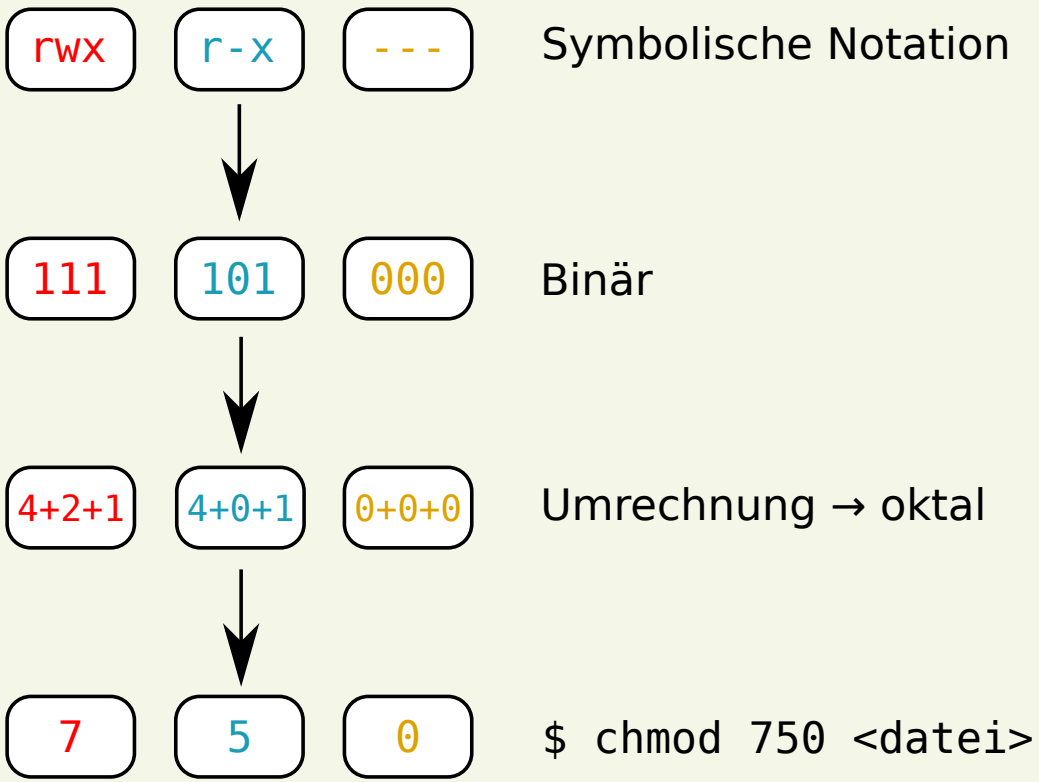
Technische Universität Braunschweig

```
$ chmod u+r datei

$ chmod go-rwx datei

$ chmod a+rx datei

$ chmod u=rwx,g=rx,o= datei
```

Änderungen im Rechtesystem

Aber ich kann besser mit Zahlen als mit Zeichen!

Kein Problem:



Rechte auf Verzeichnissen

Technische Universität Braunschweig

```
$ ls -l
- rw- r-- r--  1 tempel mitarb 97 Oct 7 14:38 datei
d rwx r-x r-x  2 tempel mitarb 40 Oct 7 14:37 verzeichnis
```

Aber Moment! Wie können Verzeichnisse ausführbar sein?

Für Verzeichnisse gilt:

- Lesbar: Inhalt des Verzeichnisses kann aufgelistet werden.
  (z. B. mit `ls`)
- Schreibbar: Weitere Verzeichnisse und Dateien können angelegt bzw. *gelöscht* werden.
- Ausführbar: Verzeichnis kann „betreten“ werden (⇒ Kein Zugriff auf darin befindliche Dateien und Verzeichnisse).

Rechte auf Verzeichnissen

Technische Universität Braunschweig

```
$ ls -l verzeichnis/
-rw------- 1 tempel mitarb 97 Oct 7 14:38 dokument
drwx------ 1 tempel mitarb 97 Oct 7 14:38 ordner

$ chmod -R a+rx verzeichnis/
```

Was passiert jetzt in `verzeichnis`?

Rechte auf Verzeichnissen

Technische Universität Braunschweig

```
$ ls -l verzeichnis/
- rw- --- --- 1 tempel mitarb 97 Oct 7 14:38 dokument
d rwx --- --- 1 tempel mitarb 97 Oct 7 14:38 ordner

$ chmod -R a+rx verzeichnis/
```

Was passiert jetzt in `verzeichnis`?

```
$ ls -l verzeichnis/
- rwx r-x r-x 1 tempel mitarb 97 Oct 7 14:38 dokument
d rwx r-x r-x 1 tempel mitarb 97 Oct 7 14:38 ordner
```

Ups…

Rechte auf Verzeichnissen

Technische Universität Braunschweig

Stattdessen:

```
$ ls -l verzeichnis/
-rw------- 1 tempel mitarb 97 Oct 7 14:38 dokument
drwx------ 1 tempel mitarb 97 Oct 7 14:38 ordner

$ chmod -R a+rX verzeichnis/

$ ls -l verzeichnis/
-rw-r--r-- 1 tempel mitarb 97 Oct 7 14:38 dokument
drwxr-xr-x 1 tempel mitarb 97 Oct 7 14:38 ordner
```

`chmod -R +X`

- Setzt das `x`-Recht nur dort, wo schon für irgendeinen Benutzer `x`-Rechte eingetragen sind.
- Also normalerweise nur bei Verzeichnissen und Programmdateien.

Verzeichnis abdichten

Technische Universität Braunschweig

Neu erstellte Verzeichnisse sind standardmäßig für alle Nutzer les- und betretbar. Dies kann dies problematisch sein.

```
$ mkdir /tmp/he29heri
$ ls -ld /tmp/he29heri
drwxr-xr-x 2 he29heri mitarb 4096 Oct 5 15:35 /tmp/he29heri
```

- Andere Benutzer können den Inhalt dieses Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Ihr seid für die Sicherheit eurer Daten selber verantwortlich!

rechte 2025-11-21

Verzeichnis abdichten

Neu erstellte Verzeichnisse sind standardmäßig für alle Nutzer les- und betretbar. Dies kann dies problematisch sein.

```
$ mkdir /tmp/he29heri
$ ls -ld /tmp/he29heri
drwxr-xr-x 2 he29heri mitarb 4096 Oct 5 15:35 /tmp/he29heri
```

- Andere Benutzer können den Inhalt dieses Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Ihr seid für die Sicherheit eurer Daten selber verantwortlich!

Auf Nummer sicher gehen:

```
$ chmod 700 /tmp/he29heri
```

Änderungen im Rechtesystem

Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur `root`!

Änderungen im Rechtesystem

Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur `root`!

Eigentümergruppe ändern

- `chgrp <group> <Datei|Verzeichnis>`
- `chgrp -R <group> <Datei|Verzeichnis>` (rekursiv)

Dateisysteminteraktion

Wie Interagiere ich mit dem Dateisystem?

Technische Universität Braunschweig

- Aus dem C Programm
  - open(), read(), write(), close(), rename()
  - lstat(), stat()
  - opendir(), readdir(), closedir()
  - creat(), mkdir(), link(), symlink()
  - rmdir(), unlink()
  - Offene Dateien als repräsentiert als Dateideskriptor
- Von der Shell
  - cat, mv
  - ls
  - touch, mkdir, ln
  - rmdir, rm

Dateisysteminteraktion aus der Shell

Anzeige von Textdateien

Technische Universität Braunschweig

Zum Anzeigen von Textdateien gibt es den Befehl `cat`.

Typische Verwendung

cat <Datei>

```
$ cat elementare-befehle.tex
\begin{frame}
\frametitle{manpages -- das Hilfesystem unter Unix}
...
```

cat-less 2024-10-18

Anzeige von Textdateien (2)

Technische Universität Braunschweig

Hilfe, so schnell kann ich nicht lesen!

Wie kann ich die Anzeige verlangsamen?

`cat` gibt eingelesene Datei komplett aus, egal wie groß diese ist.
Seitenweise Anzeige: `less`.

Typische Verwendung

```
less <Datei>
```

Achtung!

- `cat` und `less` können nur Textdateien sinnvoll anzeigen.
- Falls nach der Ausgabe einer Binärdatei nur noch seltsame Zeichen dargestellt werden, hilft der Befehl `reset`.

cat-less 2024-10-18

`rm` – Löschen

Technische Universität Braunschweig

`rm`

`rm` löscht Dateien und Verzeichnisse

Beispiele

`rm foo.pdf`	–	löscht die Datei 'foo.pdf'
`rm -r Mails/`	–	löscht das Verzeichnis 'Mails' und alle darin enthaltenen Dateien und Unterverzeichnisse
`rm -rf wichtig/`	–	löscht das Verzeichnis 'wichtig' mit allen darin enthaltenen Dateien und Unterverzeichnissen, ohne nachzufragen – auch falls diese schreibgeschützt sind!

Achtung!

`rm` löscht **ohne** Nachfrage und **ohne** Umweg über den Papierkorb!

loeschen 2024-08-12

`mkdir`, `rmdir` – Verzeichnisse erstellen und entfernen

`mkdir`

`mkdir foo`	legt ein Verzeichnis 'foo' im aktuellen Verzeichnis an

`rmdir`

`rmdir foo`	löscht das Verzeichnis 'foo' aus dem aktuellen Verzeichnis ('foo' muss leer sein)

loeschen 2024-08-12

`mv` – Verschieben

Aufbau

mv <Quelle> <Ziel>

Beispiele

`mv alt neu`	–	benennt die Datei 'alt' in 'neu' um (geht auch für Verzeichnisse)
`mv foo dinge/`	–	verschiebt die Datei 'foo' aus dem aktuellen Verzeichnis in das Verzeichnis 'dinge'

cp – Kopieren

Technische Universität Braunschweig

Aufbau

cp <Quelle> <Ziel>

Beispiele

`cp bsp bspkopie`	–	kopiert die Datei 'bsp' nach 'bspkopie' (im aktuellen Verzeichnis)
`cp bsp test/`	–	kopiert die Datei 'bsp' in das Verzeichnis 'test'
`cp -v bsp test/`	–	…mit Ausgabe der einzelnen Kopieraktionen
`cp -a test/ test2`	–	erstellt eine Kopie des Verzeichnisses 'test' mit dem Namen 'test2'
`cp -a /verz .`	–	erstellt eine Kopie des Verzeichnisses '/verz' im aktuellen Verzeichnis

`find` – Suche nach Dateien

Technische Universität Braunschweig

Rekursive Dateisuche in Verzeichnisstrukturen nach bestimmten Kriterien.

Aufruf

```
$ find [dir] <filter1> <filter2> ...
```

Häufig benutzte Filter:

-name, -iname	sucht mit Wildcards nach Dateinamen, mit *i* case-insensitive (Groß-/Kleinschreibung egal)
-type [f|d]	sucht nur nach bestimmten Dateityp, *f* für Files, *d* für Directories

`find` – Suche nach Dateien

Technische Universität Braunschweig

```
$ find . -name '*.pdf'
./AuD/uebungen/blatt01.pdf
./studbesch_ws1314.pdf

$ find Musik/ -iname '*.mp3'
./Musik/Deep_Purple/Made_in_Japan/Highwaystar.MP3
```

`grep` – Suchen in Dateien

Technische Universität Braunschweig

Sucht in der Standard-Eingabe (*stdin*) oder in Dateien nach Zeilen, die auf einen regulären Ausdruck passen, und gibt passende Zeilen auf der Standard-Ausgabe (*stdout*) aus.
Der einfachste reguläre Ausdruck umfasst nur das Suchwort selbst.

Aufruf

```
grep <pattern> [file1 file2 ...]
```

```
$ grep ssh /etc/services
ssh     22/tcp     # SSH Remote Login Protocol
ssh     22/udp
```

Tipp

```
grep -i <pattern>
```

Option *-i* zum Suchen ohne Beachtung der Groß-/Kleinschreibung.

Reguläre Ausdrücke

Technische Universität Braunschweig

- ... erlauben Beschreibung von Mustern für Zeichenketten
- Zeichen Mengen
  - [a-z] Kleinbuchstaben von a bis z
  - [:digit:] Ziffern
- Wiederholungen
  - ? 0 oder 1 mal
  - * Beliebig oft, auch 0 mal
  - {m,n} m bis n mal
- Kontrollfluss
  - | Alternative
  - () Gruppierung
- Ausführliche Beschreibung siehe regex(7), grep(1)
- Bei Angabe auf der Shell muss auf Maskierung geachtet werden
  - Sonst interpretiert die Shell fälschlicherweise Teile

regex 2024-08-12

Reguläre Ausdrücke

Technische Universität Braunschweig

- ... im C-Programm (siehe regex(3))
- Zweischrittiges Verfahren

1. Vorbereiten

```
int regcomp(regex_t *preg, const char *regex, int cflags);
```

2. Anwenden

```
int regexec(const regex_t *preg, const char *string, size_t nmatch,
            regmatch_t pmatch[], int eflags);
```

Wildcards

Wildcards

```
$ ls
linuxkurs2016.aux linuxkurs2016.log linuxkurs2016.nav
linuxkurs2016.pdf linuxkurs2016.tex linuxkurs2016.toc
linuxkurs2017.aux linuxkurs2017.log linuxkurs2017.nav
linuxkurs2017.pdf linuxkurs2017.tex linuxkurs2017.toc
```

Wildcards

Technische Universität Braunschweig

```
$ ls
linuxkurs2016.aux linuxkurs2016.log linuxkurs2016.nav
linuxkurs2016.pdf linuxkurs2016.tex linuxkurs2016.toc
linuxkurs2017.aux linuxkurs2017.log linuxkurs2017.nav
linuxkurs2017.pdf linuxkurs2017.tex linuxkurs2017.toc
```

Wie werde ich nur die ganzen Dateien vom letzten Jahr los?

```
$ rm linuxkurs2016.aux linuxkurs2016.log linuxkurs2016.nav …
```

Geht das nicht einfacher?!

Wildcards

Technische Universität Braunschweig

Aber natürlich.

Platzhalter

Die *bash* erlaubt den Einsatz von Platzhalterzeichen („Wildcards“).

- `*` steht für beliebig viele (oder auch keine) Zeichen
- `?` steht für genau ein Zeichen

Wildcards

Technische Universität Braunschweig

Aber natürlich.

Platzhalter

Die *bash* erlaubt den Einsatz von Platzhalterzeichen („Wildcards“).

- `*` steht für beliebig viele (oder auch keine) Zeichen
- `?` steht für genau ein Zeichen

Zurück zum Beispiel:

```
$ rm linuxkurs2016*
```

`linuxkurs2016*` steht demnach für alle Dateinamen, die mit `linuxkurs2016` beginnen:

`linuxkurs2016*` ⇝ `linuxkurs2016.aux linuxkurs2016.log …`

Platzhalter II

Es geht auch noch etwas komplizierter:

- `[123]` steht für genau eines der Zeichen zwischen den eckigen Klammern: `1 2 3`
- `[!123]` steht für ein Zeichen, das nicht zwischen den Klammern steht: z.B. `a 4 J _`
- `[a-d]` steht für ein Zeichen aus dem angegebenen Bereich: `a b c d`
- `{1,2,abc}` steht der Reihe nach für *alle* der angegebenen Strings (unabhängig davon, ob eine Datei mit dem Namen existiert)

Beispiele

```
$ ls
hand sand band
$ ls [hbr]and
hand band
```

```
$ wget http://www.example.net/folien{0,1,2,3,4}.pdf
```

Lädt die Dateien `folien0.pdf`, `folien1.pdf`, ... vom Server herunter

```
$ pdftk folien*.pdf cat output allefolien.pdf
```

... und baut die heruntergeladenen Dateien `folien0.pdf`, `folien1.pdf`, `folien2.pdf`, ... zu einer großen PDF-Datei zusammen.

Hinweis

Technische Universität Braunschweig

Der *-Platzhalter bezieht sich nur auf nicht-versteckte Dateien!

```
$ ls -a
.       ..      .bash_history  a.txt  mein_bild.jpg
$ rm *
$ ls -a
.       ..      .bash_history
```

Achtung!

`rm .*` würde `.` theoretisch zu `..` expandieren!
(die meisten `rm`-Versionen überprüfen das allerdings intern)

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc						
t1.tar						
t2.txt						
test.c						
test2.c						
tx.map						

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc						
t1.tar						
t2.txt						
test.c	X					
test2.c	X					
tx.map						

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc		X				
t1.tar						
t2.txt						
test.c	X	X				
test2.c	X	X				
tx.map						

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc		X				
t1.tar						
t2.txt						
test.c	X	X				
test2.c	X	X	X			
tx.map						

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc		X				
t1.tar				X		
t2.txt						
test.c	X	X				
test2.c	X	X	X			
tx.map				X		

Fun with Wildcards

Technische Universität Braunschweig

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						
attest.doc		X				
t1.tar				X		
t2.txt						
test.c	X	X				
test2.c	X	X	X			
tx.map				X	X	

Fun with Wildcards

	test*	***test***	**test?.***	**t[1x].***	**t[!12].***	**.text***
.text.c						X
attest.doc		X				
t1.tar				X		
t2.txt						
test.c	X	X				
test2.c	X	X	X			
tx.map				X	X	

Wildcards auswerten

Technische Universität Braunschweig

- ... mit der Funktion fnmatch(3)
  - Prüft, ob der String `string` zum Wildcard-Muster `pattern` passt

```
int fnmatch(const char *pattern, const char *string, int flags);
```

Wildcards auswerten

Technische Universität Braunschweig

- ... mit der Funktion fnmatch(3)
  - Prüft, ob der String `string` zum Wildcard-Muster `pattern` passt

```
int fnmatch(const char *pattern, const char *string, int flags);
```

- Flags (0 oder bitweises Oder von ein oder mehreren der Werte)
  - `FNM_PATHNAME`: Ein Slash in `string` wird nur von einem Slash-Zeichen in `pattern` getroffen, nicht von einem Wildcard-Zeichen

Wildcards auswerten

Technische Universität Braunschweig

- ... mit der Funktion fnmatch(3)
  - Prüft, ob der String `string` zum Wildcard-Muster `pattern` passt

```
int fnmatch(const char *pattern, const char *string, int flags);
```

- Flags (0 oder bitweises Oder von ein oder mehreren der Werte)
  - `FNM_PATHNAME`: Ein Slash in `string` wird nur von einem Slash-Zeichen in `pattern` getroffen, nicht von einem Wildcard-Zeichen
  - `FNM_PERIOD`: Ein führender Punkt in einer Pfadkomponente muss von einem korrespondierenden Punkt in `pattern` getroffen werden

Wildcards auswerten

Technische Universität Braunschweig

- ... mit der Funktion fnmatch(3)
  - Prüft, ob der String `string` zum Wildcard-Muster `pattern` passt

```
int fnmatch(const char *pattern, const char *string, int flags);
```

- Flags (0 oder bitweises Oder von ein oder mehreren der Werte)
  - `FNM_PATHNAME`: Ein Slash in `string` wird nur von einem Slash-Zeichen in `pattern` getroffen, nicht von einem Wildcard-Zeichen
  - `FNM_PERIOD`: Ein führender Punkt in einer Pfadkomponente muss von einem korrespondierenden Punkt in `pattern` getroffen werden
  - vollständige Beschreibung: glob(7), bash(3), fnmatch(3)

Wildcards auswerten

- ... mit der Funktion fnmatch(3)
  - Prüft, ob der String `string` zum Wildcard-Muster `pattern` passt

```
int fnmatch(const char *pattern, const char *string, int flags);
```

- Flags (0 oder bitweises Oder von ein oder mehreren der Werte)
  - `FNM_PATHNAME`: Ein Slash in `string` wird nur von einem Slash-Zeichen in `pattern` getroffen, nicht von einem Wildcard-Zeichen
  - `FNM_PERIOD`: Ein führender Punkt in einer Pfadkomponente muss von einem korrespondierenden Punkt in `pattern` getroffen werden
  - vollständige Beschreibung: glob(7), bash(3), fnmatch(3)
- Die Shell wertet die Wildcards aus
  - Quoting ermöglicht die Weitergabe als Parameter

```
mein_programm --suchmuster '*.pdf'
```

Dateisysteminteraktion aus C

Elementaroperationen auf Dateien

Technische Universität Braunschweig

- Dateien bearbeiten: open(2)/creat(2), read(2), write(2), close(2)

`int open(const char *path, int flags, mode_t mode ↳ );`	öffnet bestehende Datei; gibt **Dateideskriptor** zurück
`int creat(const char *path, mode_t mode);`	erzeugt Datei + Verzeichniseintrag; gibt **Dateideskriptor** zurück
`ssize_t read(int fd, void *buf, size_t count);`	liest aus geöffneter Datei (Deskriptor `fd`); gibt Anzahl der gelesen Bytes zurück (0 bei Dateiende)
`ssize_t write(int fd, void *buf, size_t count);`	schreibt in geöffnete Datei (Deskriptor `fd`); gibt Anzahl der gelesen Bytes zurück (0 bei Dateiende)
`int close(int fd);`	schließt geöffnete Datei (Deskriptor `fd`)

Dateiinformationen auslesen

- stat(2)/lstat(2) liefern Datei-Attribute aus dem i-node
- Unterschiedliches Verhalten bei Symlinks:
  - stat(2) folgt Symlinks (rekursiv) und liefert Informationen übers Ziel
  - lstat(2) liefert Informationen über den Symlink selber
- Funktions-Prototypen

```
int stat(const char *path, struct stat *buf);

int lstat(const char *path, struct stat *buf);
```

  - `path`: Dateiname
  - `buf`: Zeiger auf Puffer zum Speichern der Dateiinformationen

Dateiinformationen auslesen

- stat(2)/lstat(2) liefern Datei-Attribute aus dem i-node
- Unterschiedliches Verhalten bei Symlinks:
  - stat(2) folgt Symlinks (rekursiv) und liefert Informationen übers Ziel
  - lstat(2) liefert Informationen über den Symlink selber
- Funktions-Prototypen

```
int stat(const char *path, struct stat *buf);

int lstat(const char *path, struct stat *buf);
```

  - `path`: Dateiname
  - `buf`: Zeiger auf Puffer zum Speichern der Dateiinformationen
- Für uns relevante Strukturkomponenten der `struct stat`:
  - `mode_t st_mode`: Dateimode, u. a. Zugriffs-Bits und Dateityp
    - Zur Bestimmung des Dateitypes gibt es u. a. folgende Makros (inode(7)): `S_ISREG`, `S_ISDIR`, `S_ISLNK`
  - `off_t st_size`: Dateigröße in Bytes

Dateiinformationen auslesen (Forts.)

Attribute auflisten (*i-node* auslesen): lstat(2)

```
int lstat(const char *pathname, struct stat *buf);
```

liest Attribute (*i-node*-Daten)

```
struct stat {
  dev_t     st_dev;         // device ID containing file
  ino_t     st_ino;         // i-node number
  mode_t    st_mode;        // protection
  nlink_t   st_nlink;       // number of hard links
  uid_t     st_uid;         // user ID of owner
  gid_t     st_gid;         // group ID of owner
  dev_t     st_rdev;        // device ID (if special file)
  off_t     st_size;        // total size, in bytes
  blksize_t st_blksize;     // blocksize for filesystem IO
  blkcnt_t  st_blocks;      // number of blocks allocated
  struct timespec st_atim; // time of last access
  struct timespec st_mtim; // time of last modification
  struct timespec st_ctim; // time of last status change
};
```

Ergebnis von lstat(2): *i-node*-Datenstruktur des Betriebssystems (VFS)

Elementaroperationen auf Verzeichnisse

Verzeichnis auflisten: opendir(3), readdir(3), closedir(3)

`DIR *opendir(const char *name);`	öffnet ein Verzeichnis für readdir(3)
`struct dirent *readdir(DIR *dirp);`	liest nächsten Verzeichniseintrag; gibt `NULL` zurück, falls am Ende angekommen
`int closedir(DIR *dirp);`	schließt das mit opendir(3) geöffnete Verzeichnis
`struct dirent { ino_t d_ino; // i-node number ... char d_name[256]; // filename };`	Verzeichniseintrag kann weitere Elemente beinhalten, aber nur `d_ino` und `d_name` sind in POSIX spezifiziert

```c
#include <stdio.h>
#include <dirent.h>

int main() {
  DIR* dir = opendir( "." );    // öffne aktuelles Verzeichnis
  struct dirent* e = NULL;
  while(  (e = readdir(dir)) )  // lese nächsten Eintrag bis  NULL
    printf("%s\n", e->d_name);  //   gebe ihn aus
  closedir( dir );              // schließe Verzeichnis
  return 0;
}
```

Beispiel:
Auflisten aller Namen im aktuellen Verzeichnis.

Verzeichnisinhalte auslesen

Technische Universität Braunschweig

```c
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

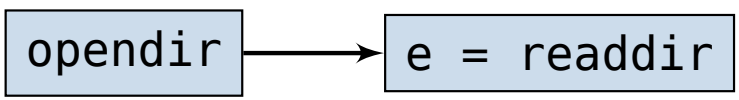
- Die `DIR`-Struktur ist ein Iterator und speichert die jeweils aktuelle Position

`opendir`

Verzeichnisinhalte auslesen

```
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

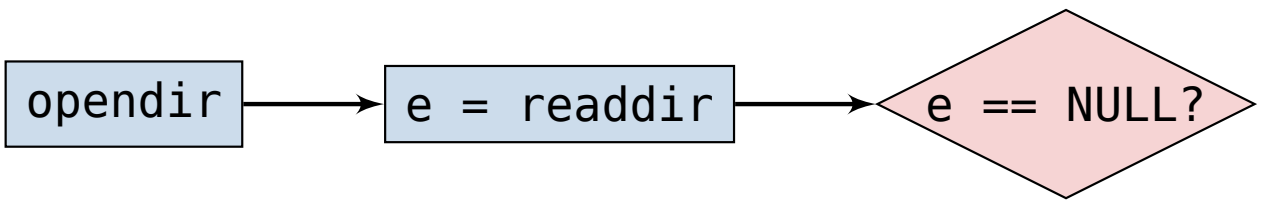
- Die **`DIR`**-Struktur ist ein Iterator und speichert die jeweils aktuelle Position
- readdir(3) liefert einen Verzeichniseintrag und setzt den **`DIR`**-Iterator auf den Folgeeintrag



Verzeichnisinhalte auslesen

```
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

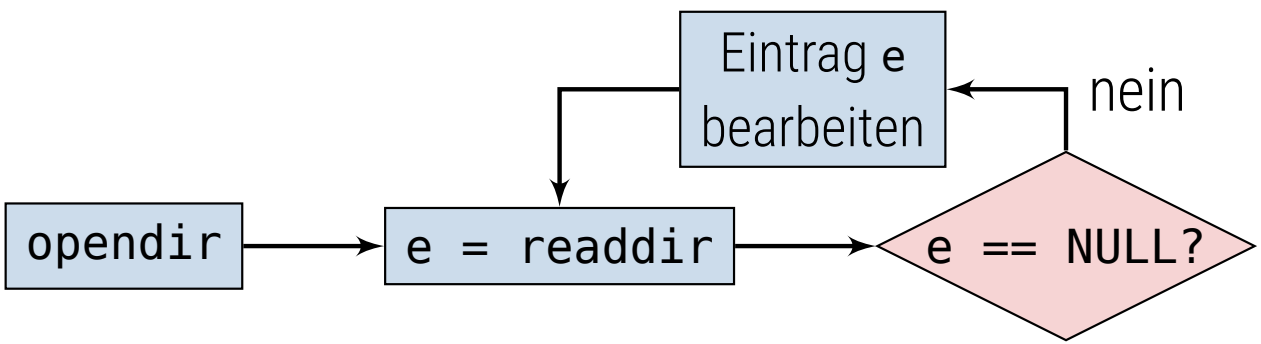
- Die `DIR`-Struktur ist ein Iterator und speichert die jeweils aktuelle Position
- readdir(3) liefert einen Verzeichniseintrag und setzt den `DIR`-Iterator auf den Folgeeintrag
  - Rückgabewert `NULL` im Fehlerfall oder wenn EOF erreicht wurde
    - bei EOF bleibt `errno` unverändert, im Fehlerfall wird `errno` entsprechend gesetzt
    - am besten vor jedem `readdir` Aufruf `errno` zurücksetzen



Verzeichnisinhalte auslesen

```
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

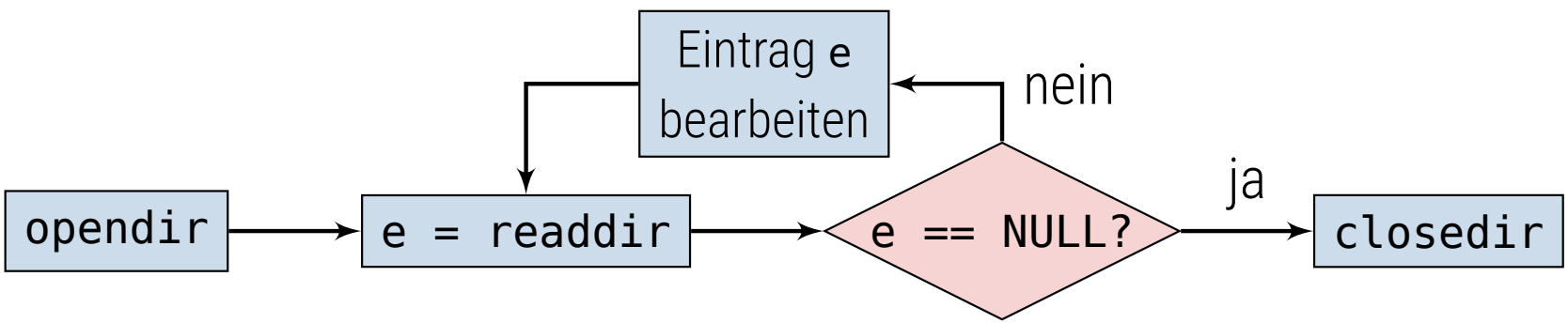
- Die `DIR`-Struktur ist ein Iterator und speichert die jeweils aktuelle Position
- readdir(3) liefert einen Verzeichniseintrag und setzt den `DIR`-Iterator auf den Folgeeintrag
  - Rückgabewert `NULL` im Fehlerfall oder wenn EOF erreicht wurde
    - bei EOF bleibt `errno` unverändert, im Fehlerfall wird `errno` entsprechend gesetzt
    - am besten vor jedem `readdir` Aufruf `errno` zurücksetzen



Verzeichnisinhalte auslesen

```
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

- Die `DIR`-Struktur ist ein Iterator und speichert die jeweils aktuelle Position
- readdir(3) liefert einen Verzeichniseintrag und setzt den `DIR`-Iterator auf den Folgeeintrag
  - Rückgabewert `NULL` im Fehlerfall oder wenn EOF erreicht wurde
    - bei EOF bleibt `errno` unverändert, im Fehlerfall wird `errno` entsprechend gesetzt
    - am besten vor jedem `readdir` Aufruf `errno` zurücksetzen
- closedir(3) gibt die belegten Ressourcen nach Ende der Bearbeitung frei



Diskussion der Schnittstelle von readdir(3)

Technische Universität Braunschweig

- Der Speicher für die zurückgelieferte `struct dirent` wird von den Bibliotheksfunktionen selbst angelegt und beim nächsten `readdir`-Aufruf auf dem gleichen `DIR`-Iterator potentiell wieder verwendet!
  - werden Daten aus der `dirent`-Struktur länger benötigt, müssen sie vor dem nächsten `readdir`-Aufruf kopiert werden
- Konzeptionell schlecht
  - aufrufende Funktion arbeitet mit Zeiger auf internen Speicher der `readdir`-Funktion
- In nebenläufigen Programmen nur bedingt einsetzbar
  - man weiß evtl. nicht, wann der nächste `readdir`-Aufruf stattfindet

Vergleich: readdir(3) und stat(2)

- Die problematische Rückgabe auf funktionsinternen Speicher wie bei readdir(3) gibt es bei stat(2) nicht

Vergleich: readdir(3) und stat(2)

- Die problematische Rückgabe auf funktionsinternen Speicher wie bei readdir(3) gibt es bei stat(2) nicht
- Grund: stat(2) ist ein Systemaufruf – Vorgehensweise wie bei readdir(3) wäre gar nicht möglich
  - readdir(3) ist komplett auf Ebene 3 implementiert (Teil der Standard-C-Bibliothek/Laufzeitbibliothek)
  - stat(2) ist (nur) ein Systemaufruf(-stumpf), die Funktion selbst ist Teil des Betriebssystems (Ebene 2)

Vergleich: readdir(3) und stat(2)

- Die problematische Rückgabe auf funktionsinternen Speicher wie bei readdir(3) gibt es bei stat(2) nicht
- Grund: stat(2) ist ein Systemaufruf – Vorgehensweise wie bei readdir(3) wäre gar nicht möglich
  - readdir(3) ist komplett auf Ebene 3 implementiert (Teil der Standard-C-Bibliothek/Laufzeitbibliothek)
  - stat(2) ist (nur) ein Systemaufruf(-stumpf), die Funktion selbst ist Teil des Betriebssystems (Ebene 2)
- der logische Adressraum auf Ebene 3 (Anwendungsprogramm) ist nur eine Teilmenge von dem logischen Adressraum auf Ebene 2 (Betriebssystemkern)
  - Betriebssystemspeicher ist für Anwendung nicht sichtbar/zugreifbar
  - Funktionen der Ebene 2 können keine Zeiger auf ihre internen Datenstrukturen an Ebene 3 zurückgeben

Argumentparser

Wie komme ich an die Argumente?

Technische Universität Braunschweig

```
tempel@x1:~$ ls -a -l /etc
```

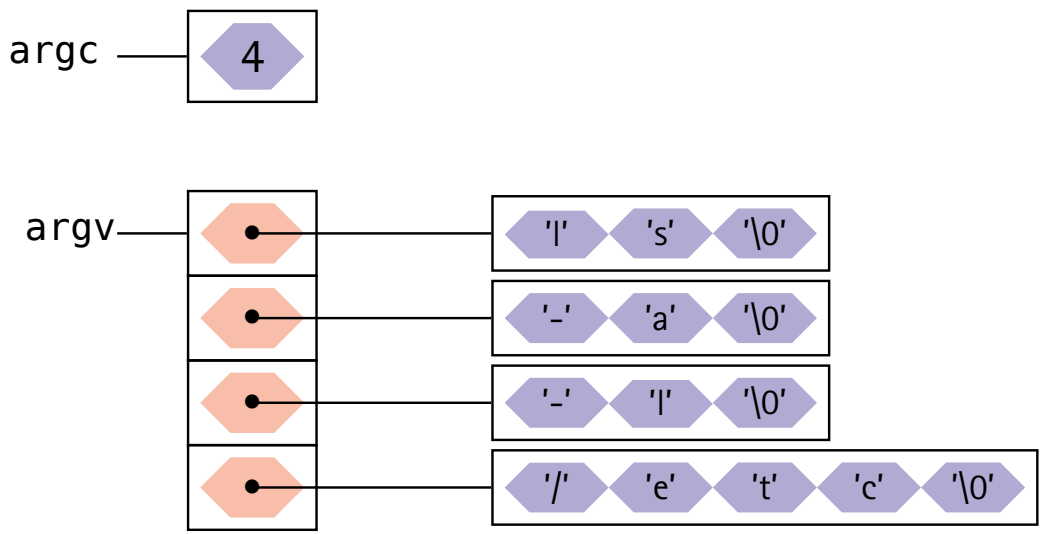
```c
int main(int argc, char *argv[])
```

Wie komme ich an die Argumente?

```
tempel@x1:~$ ls -a -l /etc
```

```
int main(int argc, char *argv[])
```

- `argc` Anzahl der Argumente
- `argv` Feld mit Zeigern auf die Argumente



Parser für Argumente

Technische Universität Braunschweig

- Schnittstelle:

`int initArgumentParser(int argc, char* argv[]);`	Initialisierung des Argumentenparsers. Prüft die Argumente auf plausibilität und bereitet die folgenden Aufrufe vor.
`char* getCommand(void);`	Liefert den Namen des aufgerufenen Programms
`char* getValueForOption(char* keyName);`	Liefert den Wert der benannten Option. Optionen können als Schlüssel-Wert-Paare der Form `-key=value` übergeben werden.
`int getNumberOfArguments(void);`	Liefert die Anzahl der angegebenen Argumente.
`char* getArgument(int index);`	Liefert das entsprechende Argument.

```
command [arg]... [-option=value]...
```

crawl

crawl

Technische Universität Braunschweig

```
crawl path... [-maxdepth=n] [-name=pattern] [-type={d,f}]
[-size=[+|-]n] [-line=string]
```

- Angegebenes Verzeichnis rekursiv durchsuchen
- Auflisten der passenden Dateien (und Zeilen)
- Kriterien:

-name	(Wildcard-)Pattern auf Dateinamen
-size	Vergleich der Dateigröße
-type	Typ der aufzulistenden Dateien
-line	Regulärer Ausdruck, anzuwenden auf Zeilen der Dateien
-maxdepth	Maximale Suchtiefe

crawl

Technische Universität Braunschweig

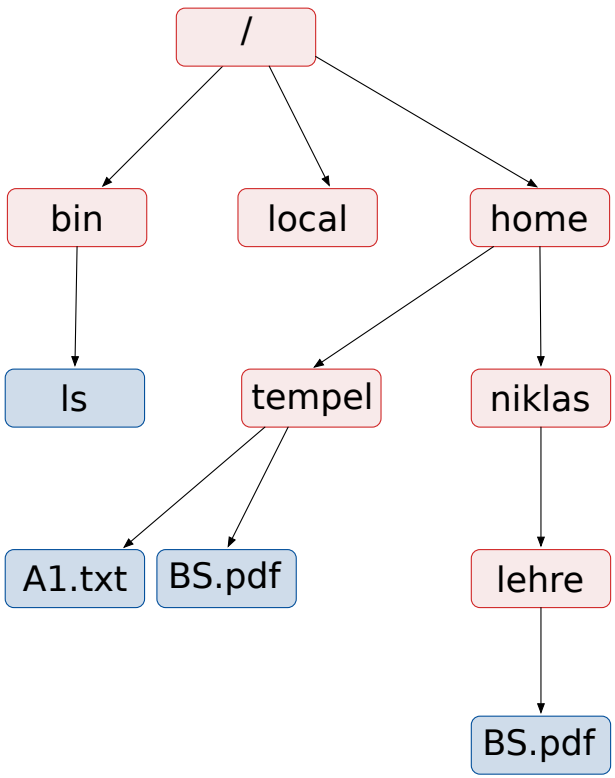
```
crawl path... [-maxdepth=n] [-name=pattern] [-type={d,f}]
[-size=[+|-]n] [-line=string]
```

- Angegebenes Verzeichnis rekursiv durchsuchen
- Auflisten der passenden Dateien (und Zeilen)
- Kriterien:

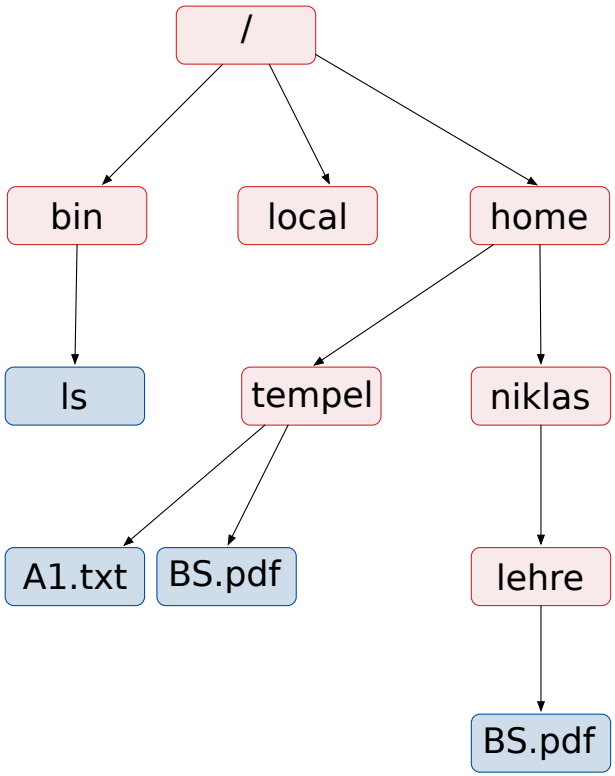
-name	(Wildcard-)Pattern auf Dateinamen
-size	Vergleich der Dateigröße
-type	Typ der aufzulistenden Dateien
-line	Regulärer Ausdruck, anzuwenden auf Zeilen der Dateien
-maxdepth	Maximale Suchtiefe

```
tempel@x1:/ibr/courses/ws2526/bs/blatt3$ ./crawl . -name="*log"
./testdir/resources/utility/srvLog/2014-03-03.log
./testdir/resources/utility/srvLog/2014-03-02.log
./testdir/resources/utility/ProgrammAufrufe.log
...
```

Dateibaum

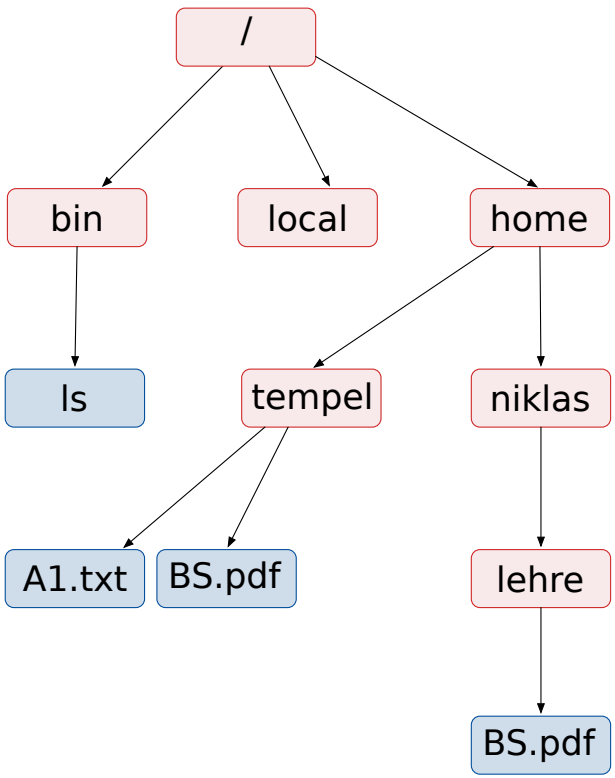


Dateibaum



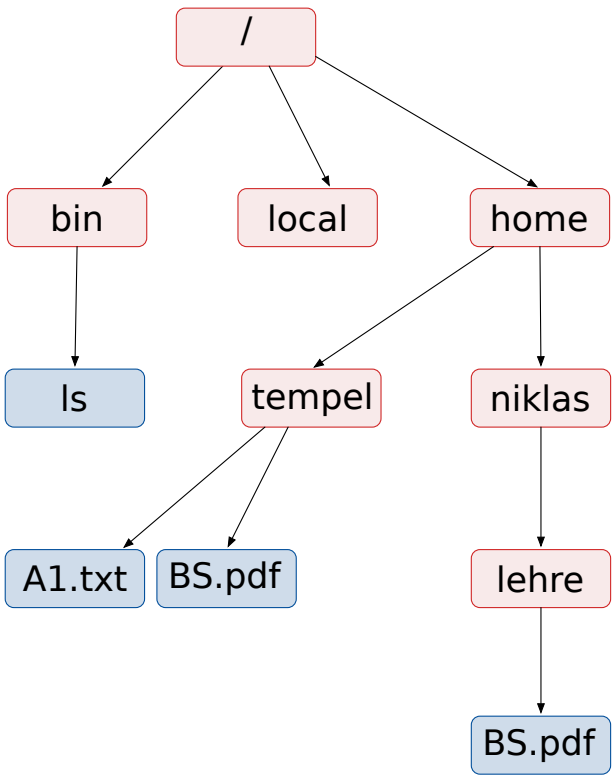

```
$ ./crawl /
```

Dateibaum



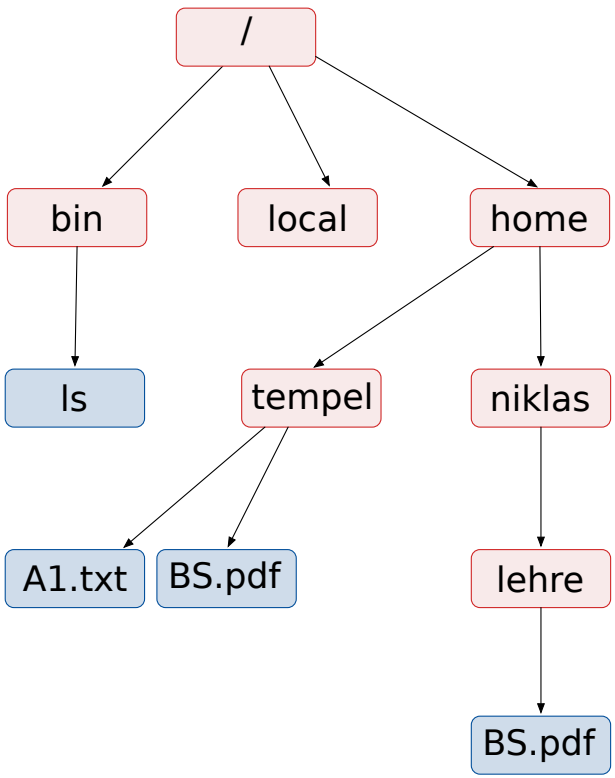

```
$ ./crawl /
/
```

Dateibaum



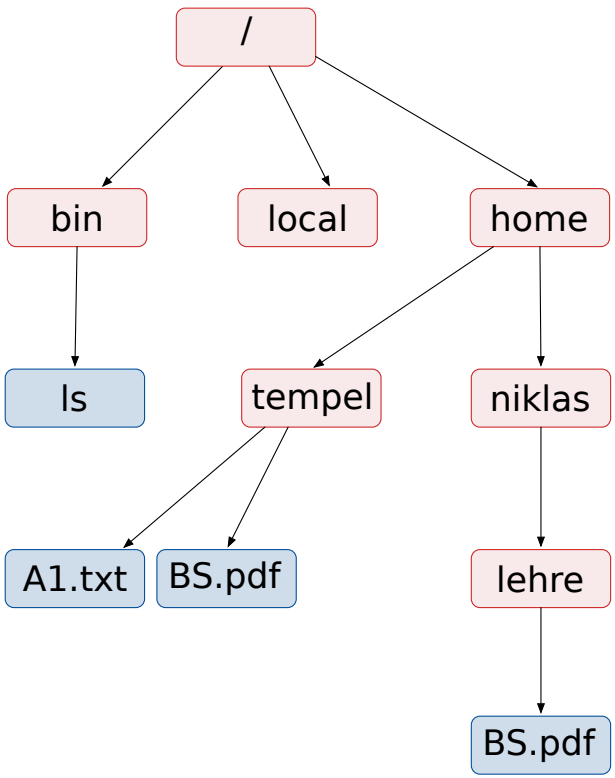

```
$ ./crawl /
/
/bin
```

Dateibaum



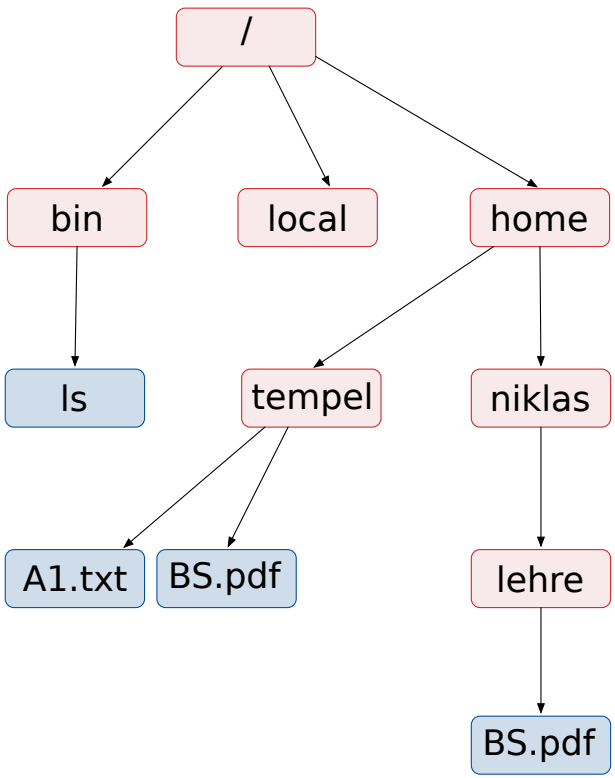

```
$ ./crawl /
/
/bin
/bin/ls
```

Dateibaum



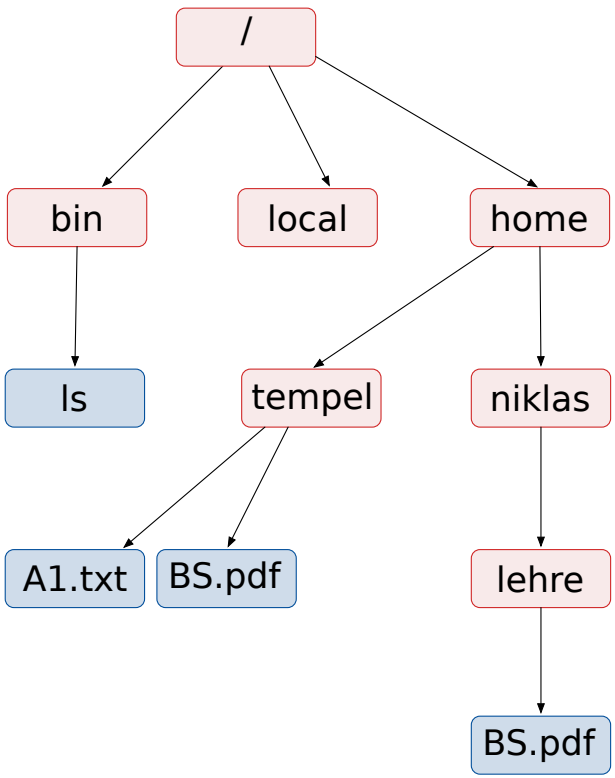

```
$ ./crawl /
/
/bin
/bin/ls
/local
```

Dateibaum



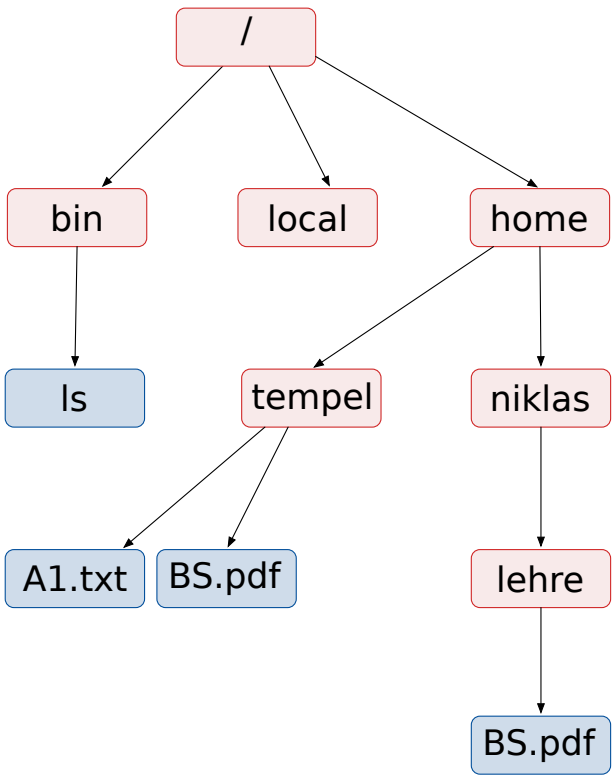

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
```

Dateibaum



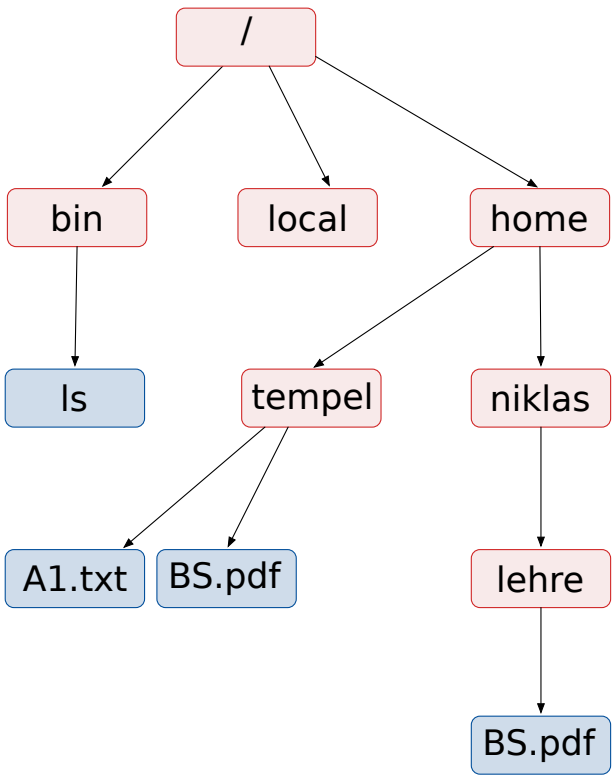

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
```

Dateibaum



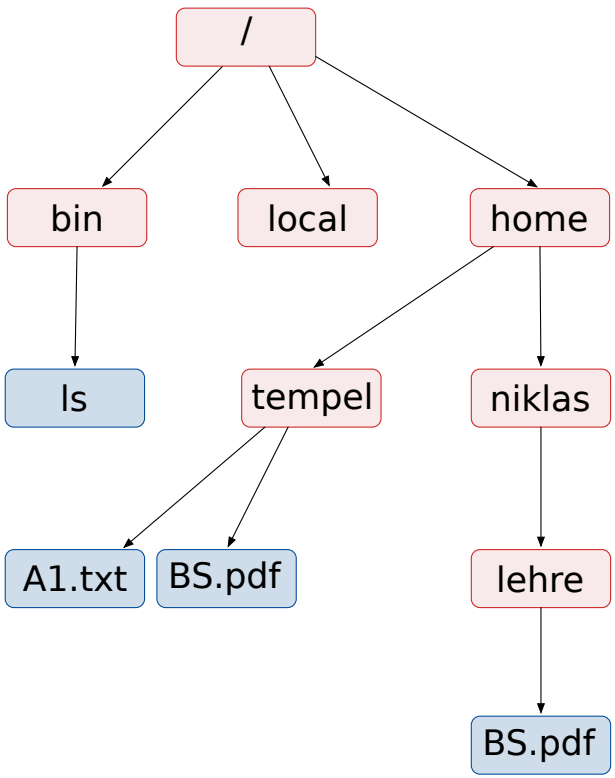

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
```

Dateibaum



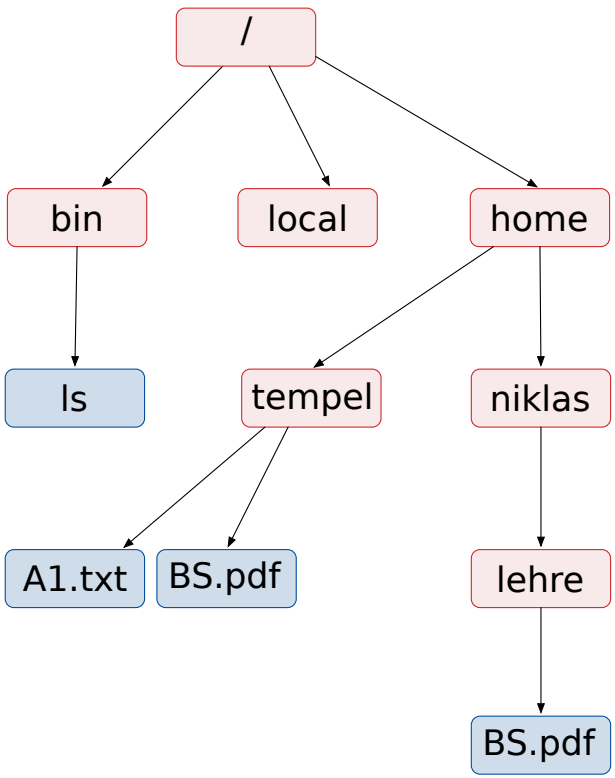

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
/home/tempel/BS.pdf
```

Dateibaum



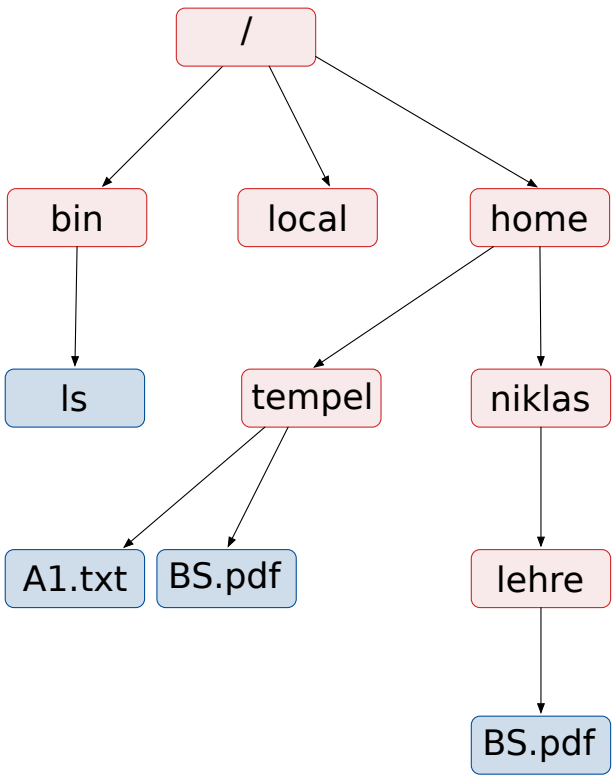

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
/home/tempel/BS.pdf
/home/niklas
```

Dateibaum



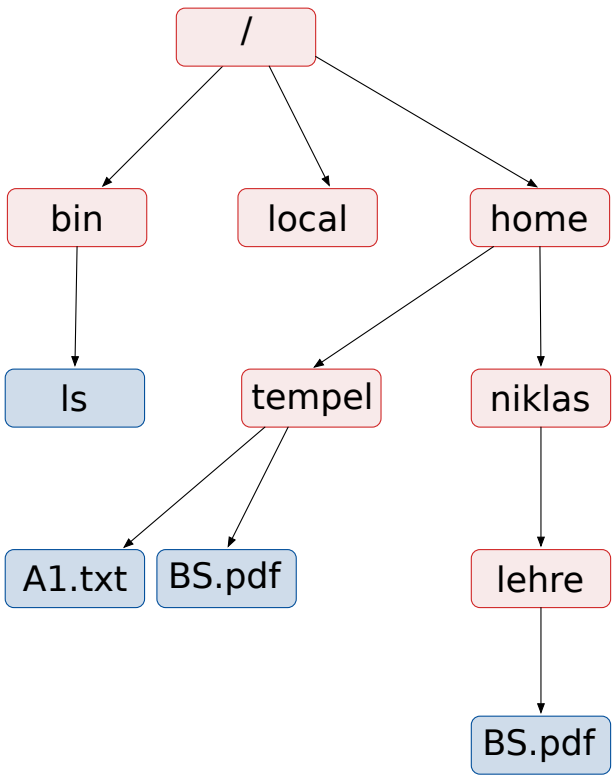

```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
/home/tempel/BS.pdf
/home/niklas
/home/niklas/lehre
```

Dateibaum




```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
/home/tempel/BS.pdf
/home/niklas
/home/niklas/lehre
/home/niklas/lehre/BS.pdf
```

Dateibaum




```
$ ./crawl /
/
/bin
/bin/ls
/local
/home
/home/tempel
/home/tempel/A1.txt
/home/tempel/BS.pdf
/home/niklas
/home/niklas/lehre
/home/niklas/lehre/BS.pdf
```