Rechnernetze und Betriebssysteme

Aufgabenblatt 1

***B*ourne-*a*gain *sh*ell**

Steven Illg

März / April 2019

Prof. Dr.-Ing. Martin Hübner

Hartmut Schulz
Daniel Sarnow

Dieses Dokument wurde mit dem Overleaf-LaTeX-Editor erstellt

1 Erste Erfahrungen mit der bash – Shell

a) Shellbefehle ausführen

1. Mit dem nachfolgenden Befehl können alle Datei- und Ordnernamen im aktuellen Arbeitsverzeichnis ausgegeben werden. Der Befehl entspricht einer Abkürzung, die aus den ersten drei Buchstaben von *»**dir**ectories«* besteht.

```
1    dir # offiziell von Windows, unter Linux nicht empfohlen
```

Die Verwendung von `dir` ist jedoch nicht die einzige Möglichkeit. Ein weiteres Beispiel ist *»**list**«*, dessen Ausgabe identisch mit der von `dir` ist.

```
1    ls # identisch mit dir, ofizieller Linux Befehl
```

Sowohl an `dir`, als auch an `ls`, können *Optionen* angehängt werden, um eine angepasste Ausgabe zu erzeugen. So kann diese nach Dateityp gefiltert oder in *Langform* mit Metadaten ausgegeben werden (siehe Abbildung 1 unten).

```
1    # liefert alle Datei-Informationen in Langform
2    dir -l || ls -l
3    # liefert Dateien mit der Endung .txt in Kurz- bzw. Langform
4    dir [-l] *.txt || ls [-l] *.txt
```

Abbildung 1 zeigt die Ausgabe von `dir`, die identisch mit der von `ls` ist und von `ls -l *.sh`, die identisch mit `dir -l *.sh` ist. Anstelle von `*.sh` kann auch ein expliziter Dateiname angegeben werden (wie `ls -l showme.sh`), um sich die Datei-Informationen über eine bestimmte Datei ausgeben zu lassen.

```
abr561@shell-18:~$ dir
Desktop     heute.txt  platz.txt      prozessbaum.txt
frename.sh  pf.sh      processes.txt  showme.sh
abr561@shell-18:~$ ls -l *.sh
-rw-r--r-- 1 abr561 students 336 Mär 24 04:01 frename.sh
-rw-r--r-- 1 abr561 students  73 Mär 26 10:27 pf.sh
-rw-r--r-- 1 abr561 students 445 Mär 27 12:48 showme.sh
```

Abbildung 1: Ausgabe von `dir` und `ls -l *.sh`

Hinweis:

Hinter dem salopp verwendeten Begriff *»Befehl«* steckt etwas mehr, als es in dieser Ausarbeitung deutlich wird. Genau genommen stellt die Verwendung – wie zum Beispiel von `ls` – einen *»Built-in«* Befehl dar, der vergleichbar mit einem Programmaufruf ist. Die Shell analysiert also den Befehl und führt diesen anschließend selbst aus. Andernfalls würde ein Befehl durch einen von der Shell ausgelösten *System Call* als neues Programm ausgeführt werden.

2. Um eine Datei mit dem Namen `heute` zu erstellen, die das aktuelle Datum und die aktuelle Uhrzeit enthält, ist nur ein Shellbefehl notwendig. Realisieren lässt sich dies mit einer sogenannten *Ausgabeumlenkung*. Das bedeutet, dass die gewünschte Ausgabe nicht auf der Standardausgabe erscheint, sondern in eine separate Datei im aktuellen Arbeitsverzeichnis gespeichert werden soll.

```
1    date > heute.txt || > heute.txt date
```

Mit dem Befehl `date` wird die aktuelle Systemzeit ausgelesen und mit dem `>` Zeichen in eine Textdatei `heute.txt` umgelenkt und abschließend im aktuellen Arbeitsverzeichnis gespeichert. Der Dateiname *»heute«* ist optional, daher wären auch andere Dateinamen an dieser Stelle vorstellbar, wie *»today«* oder *»current_systemtime«*, aber auch nur ein *»h«*. Bei der Wahl des Dateinamen muss weder auf reservierte Schlüsselworte geachtet werden, noch ist eine Dateiendung zwingend notwendig. Ist der Dateiname aber bereits im Arbeitsverzeichnis vorhanden, wird die Datei ohne Rückmeldung überschrieben. Es wäre also auch der nachfolgende Befehl für eine lesbare Datei möglich.

```
1    date > pwd
```

Der `>` Operator ist nicht zu verwechseln mit dem `>>` Operator, der Strings oder ganze Ausgaben unten an die angegebene Datei anhängt.

Nachdem die Datei erstellt wurde, kann mit `dir` oder `ls` überprüft werden, ob die Datei (hier `heute.txt`) tatsächlich im aktuellen Arbeitsverzeichnis abgelegt wurde. Möchte man die Datei hingegen öffnen, hat man gleich mehrere Möglichkeiten, die im Folgenden kurz vorgestellt werden.

```
1    cat heute.txt # Textdatei auf Standardausgabe ausgeben
2    less heute.txt # Textdatei in einem Pager anzeigen
3    more heute.txt # Textdatei Seitenweise anzeigen
4    nano heute.txt # öffnet die Textdatei im Nano-Texteditor
```

`cat` Ursprünglich war der `cat` Befehl zum Zusammenfügen von Textdateien gedacht, daher kommt der Name auch von *»con**cat**enate«*. Häufig wird dieser Befehl aber zum anzeigen von kurzen Textdateien verwendet[1].

Die Abbildung 2 zeigt den Inhalt der Datei `heute.txt`, der mit dem `cat` Befehl auf der Standardausgabe (auch *»Stdout«*) ausgegeben wurde.

```
abr561@shell-18:~$ cat heute.txt
Mi 27. Mär 13:12:20 CET 2019
```

Abbildung 2: Inhalt der Datei `heute.txt`

[1] Vgl. Eintrag zu `cat` auf [UbuUs], abgerufen am 27. März 2019

`less` Mit `less` öffnet sich ein Pager, der den Inhalt einer Textdatei anzeigt. Im vergleich zu `more` bietet `less` den Vorteil, dass man sich mit Hilfe der Tasten [↑] oder [↓] frei im Dokument nach oben oder unten bewegen kann, was bei langen Texten hilfreich sein kann. Geschlossen wird der Pager mit der [Q] Taste und nicht mit [Strg]+[X], wie es bei `nano` der Fall ist. Ist dieser geschlossen, verschwindet der Inhalt der Textdatei auch auf der Standardausgabe, was bei der Verwendung von `more` nicht der Fall ist.

`more` Mit `more` öffnet sich der Inhalt einer Textdatei seitenweise auf der Standardausgabe. Das Anzeigen mit `less` wird zwar als bequemere Alternative zu `more` bezeichnet, jedoch ist `less` nicht wie `more` auf allen UNIX-Systemen vorhanden[1]. Hat die Textdatei mehr Zeilen als die Fensterhöhe, so muss die Datei entweder mit der [⏎] Taste eine Zeile oder mit der [ ] Taste eine Bildschirmseite weiter aufgeklappt werden. Mit [B] kann eine Bildschirmseite zurück geblättert werden und mit [F] wird eine Bildschirmseite übersprungen. Mit [S] wird nur eine Zeile übersprungen.

Wie viel des Textes bereits angezeigt wird, gibt der prozentuale Fortschritt an, der unten links am Fenster steht. Wird also genau die Hälfte des Textes angezeigt, wird dies mit `-More- (50%)` angegeben. Die seitenweise Ausgabe kann wie `less` mit der [Q] Taste abgebrochen oder beendet werden.

`nano` Mit dem Befehl `nano` öffnet sich ein separater Texteditor, in dem die Textdatei eingesehen, bearbeitet und abgespeichert werden kann. Als Alternative zu `nano` kann die Anwendung `vi` verwendet werden.

3. Um eine Datei mit dem Namen `platz` zu erstellen, die alle Informationen über das aktuelle Dateisystem abspeichert – inklusive der Belegung in Prozent und mit optimierten Größenangaben – bedarf es auch hier nur eines Befehls, der dem aus der vorherigen Teilaufgabe sehr ähnlich ist.

Möchte man also eine Einschätzung über die verbrauchten Speicherkapazitäten auf dem aktuellen System erhalten, wird idealerweise der *»**d**isk **f**ree«* Befehl verwendet. Mit Hilfe des `>` Zeichen wird die Abfrage, wie in der vorherigen Teilaufgabe bereits erklärt, in eine Textdatei umgelenkt und abschließend gespeichert. Der vollständige Befehl sieht folgendermaßen aus.

```
1    df -h > platz.txt
```

[1] Vgl. Eintrag zu `more` auf [UbuUs], abgerufen am 27. März 2019

Bei diesem Aufruf gibt es bis auf den `df` Befehl und die verwendete Option *»-h«* also nichts Neues. Das `-h` hinter dem `df` Befehl steht ausgeschrieben für *»human readable«* und gibt die Größe der verbrauchten Speicherkapazität in ein *für Menschen lesbares* Format an.[1] Wie sich die Ausgabe verändert, würde man an dieser Stelle auf das `-h` verzichten, soll die Abbildung 4 demonstrieren. Diese Abbildung begrenzt sich auf die ersten fünf Zeilen der Textdatei. Der hier verwendete `head` Befehl wird in der nächsten Teilaufgabe erläutert.

Die Abbildung 3 zeigt den Inhalt der `platz.txt` in seiner vollen Pracht. In der Spalte `Size` wird der verfügbare Gesamtspeicher in Kibibyte `K`, Mebibyte `M`, Gibibyte `G`, Tebibyte `T` und so weiter angegeben. In der Spalte `Used` steht der tatsächlich verbrauchte Speicherplatz und in der Spalte `Avail` der freie Speicherplatz. Die Spalte `Use %` gibt den prozentualen Speicherverbrauch an.

```
abr561@shell-18:~$ more platz.txt
Filesystem                     Size  Used Avail Use% Mounted on
udev                           464M     0  464M   0% /dev
tmpfs                           99M  1,4M   98M   2% /run
/dev/sda2                       16G  7,7G  7,3G  52% /
tmpfs                          493M     0  493M   0% /dev/shm
tmpfs                          5,0M     0  5,0M   0% /run/lock
tmpfs                          493M     0  493M   0% /sys/fs/cgroup
filerfe:/haw/service/vcs       4,4T  1,1T  3,4T  24% /srv
filerfe:/haw/user/others       4,4T  3,2T  1,2T  73% /home/others
filerfe:/haw/user/students     4,4T  3,2T  1,2T  73% /home/students
filerfe:/haw/user/staff        4,4T  3,2T  1,2T  73% /home/staff
filerfe:/haw/user/prof         4,4T  3,2T  1,2T  73% /home/prof
filerfe:/haw/service/projects  4,4T  1,1T  3,4T  24% /srv/www/projects
/dev/loop3                      91M   91M     0 100% /snap/core/6350
tmpfs                           99M     0   99M   0% /run/user/23905
/dev/loop0                      91M   91M     0 100% /snap/core/6405
tmpfs                           99M     0   99M   0% /run/user/33706
tmpfs                           99M     0   99M   0% /run/user/2056
/dev/loop2                      92M   92M     0 100% /snap/core/6531
tmpfs                           99M     0   99M   0% /run/user/24157
tmpfs                           99M     0   99M   0% /run/user/2052
tmpfs                           99M     0   99M   0% /run/user/24525
tmpfs                           99M     0   99M   0% /run/user/19313
tmpfs                           99M     0   99M   0% /run/user/24377
tmpfs                           99M     0   99M   0% /run/user/32190
```

Abbildung 3: Inhalt der Datei `platz.txt`

Betrachten wir die Spalte `Avail` oder `Available` etwas genauer: Wie bereits erwähnt gibt diese Spalte an, wie viel Speicher noch verwendet werden kann, abhängig von der Gesamtspeicherkapazität, die in der Spalte `Size` angegeben wird. Somit sind in `udev` in Zeile 2 noch `464M` verfügbar von ingesamt `464M`. Es sind also `0%` belegt. In Zeile 13 hingegen sind in `/dev/loop3` noch `0M` verfügbar. Hier sind 100% der verfügbaren Speicherkapazität verbraucht.

[1] Vgl. Eintrag zu `df` auf [UbuUs], abgerufen am 27. März 2019

Würde dem `df` Befehl nicht die Option `-h` folgen, würde der verfügbare, verbrauchte bzw. freie Speicherplatz in den Spalten `Size`, `Used` und `Avail` nicht in `K`, `M`, `G`, `T` und so weiter[1] ausgegeben werden. Wie die Ausgabe für die ersten fünf Zeilen der `platz.txt` dann aussehen würde, soll die Abbildung 4 zeigen. In Zeile 4 auf Abbildung 3 wurde beispielsweise bei `/dev/sda2` = `Size 16G` angezeigt. Ohne `-h` zeigt Abbildung 4 für dieselbe Zeile `16445308` an.

```
abr561@shell-18:~$ df | head -5
Filesystem                    1K-blocks       Used  Available Use% Mounted on
udev                             474196          0     474196   0% /dev
tmpfs                            100964       1368      99596   2% /run
/dev/sda2                      16445308    8018356    7571864  52% /
tmpfs                            504816          0     504816   0% /dev/shm
```

Abbildung 4: Ausgabe von `df | head -5`

4. Sollen die ersten fünf Zeilen einer Datei ausgegeben werden – wie die, aus der in der vorherigen Teilaufgabe erstellten Textdatei `platz.txt` – hat man mehrere Möglichkeiten, die im Folgenden kurz vorgestellt werden.

– **Die empfohlene Variante**

Mit dem unten stehenden Befehl `head -5 <file>` werden nur die ersten Zeilen von $1, 2, \ldots, n$ mit $n \geq 1$ ausgegeben. Die Option `-n` (hier `-5`) bedeutet umgangssprachlich: *»Liefere die obersten (`head`) $-n$ (mit $n = 5$) Zeilen der Textdatei ... (hier `platz.txt`)«*. Es werden also nur die ersten fünf Zeilen der Textdatei `platz.txt` verlangt und ausgegeben.

Wird für $n = 0$ angegeben, werden entsprechend keine Zeilen ausgegeben.

```
1    head -5 platz.txt
```

Die Abbildung 5 zeigt die Ausgabe des Befehls `head -5 platz.txt`.

```
abr561@shell-18:~$ head -5 platz.txt
Filesystem                    Size  Used Avail Use% Mounted on
udev                          464M     0  464M   0% /dev
tmpfs                          99M  1,4M   98M   2% /run
/dev/sda2                      16G  7,7G  7,3G  52% /
tmpfs                         493M     0  493M   0% /dev/shm
```

Abbildung 5: Ausgabe von `head -5 platz.txt`

[1] Freundlich gerundet lässt sich sagen: `M` = Megabyte, `G` = Gigabyte, `T` = Terabyte, ... und so weiter.

– **Geht auch, ist aber umständlich**

Statt `head -5` ist auch `head -n 5` möglich. Die Ausgabe ist dieselbe, daher ist diese Variante umständlicher, da `-n 5` zu `-5` vereinfacht werden kann.

```
1    head -n 5 platz.txt || head -n -20 platz.txt
```

Aber es geht noch umständlicher:

Wird die Option `-n` im Befehl integriert, beschränkt sich die Angabe der maximal auszugebenden Zeilen nicht mehr auf alle Zeilen mit $n \geq 1$. Somit werden auch Zeilen mit $n < 0$ möglich. Auf diese Weise können $-1, -2, \ldots, -n$ Zeilen einer Textdatei für die Ausgabe *»abgeschnitten«* werden. Der Befehl `head -n -5 <file>` führt also dazu, dass die untersten fünf Zeilen der Textdatei nicht ausgegeben werden.

Mit `head -n -20 <file>` werden die untersten 20 Zeilen nicht ausgegeben. Die Textdatei `platz.txt` hat insgesamt 25 Zeilen. Da die untersten 20 Zeilen nicht ausgegeben werden, bleiben $25 - 20 = 5$ Zeilen übrig. Somit erfüllt dieser Befehl zwar die Anforderung, ist aber sehr umständlich.

Ausßerdem setzt die hier vorgestellte Variante voraus, dass die Anzahl aller Zeilen in der Textdatei bekannt ist. Dementsprechend gilt die `-20` nur für die hier vorgestellte Textdatei `platz.txt` und muss für jede andere Textdatei angepasst werden. Zum Glück: Einfacher ist es mit `head -5`.

– **Mit *»stream editor«***

Eine weitere Möglichkeit, mit der sich diese Ausgabe realisieren lässt, ist der *»**s**tream **ed**itor«* Befehl `sed`. Die Option `-n` bedeutet hier nicht *n Zeilen ausgeben*, sondern meint die Verhinderung der automatischen Ausgabe der Ergebnisse. Die Ausgabe erfolgt nur über das Kommando `p`.[1]

```
1    sed -n 1,5p platz.txt
```

Wird bei der Benutzung des `head` Befehls keine maximal auszugebende Zeilenanzahl angegeben, wird der Standardwert von $n = 10$ Zeilen verwendet.

```
1    head palatz.txt # Standardwert mit n=10 Zeilen
```

[1] Vgl. Eintrag zu `sed` auf [UbuUs], abgerufen am 28. März 2019

Zu einem `head` gehört auch ein `tail`

Die Benutzung von `tail` verläuft analog zum `head` Befehl. Wird die `-n` Option im Befehl integriert, hat das Vorzeichen – also ob man `tail -n 5` oder `tail -n -5` benutzt – keinen Effekt auf die Ausgabe. Bei der Verwendung von `head` ist dies nicht der Fall und führt zu einer Abtrennung der untersten fünf Zeilen.

```
1    tail -5 platz.txt
2    tail -n -5 platz.txt || tail -n 5 platz.txt # identisch
```

Die Abbildung 6 zeigt abschließend die Benutzung des `tail` Befehls.

```
abr561@shell-18:~$ tail -5 platz.txt
tmpfs                           99M     0   99M   0% /run/user/2052
tmpfs                           99M     0   99M   0% /run/user/24525
tmpfs                           99M     0   99M   0% /run/user/19313
tmpfs                           99M     0   99M   0% /run/user/24377
tmpfs                           99M     0   99M   0% /run/user/32190
```

Abbildung 6: Ausgabe von `tail -5 platz.txt`

5. Um eine Datei mit dem Namen `prozessbaum` zu erstellen, die alle aktuellen Prozesse in einer übersichtlichen Baumdarstellung beinhaltet, wird von Bash die praktische Funktion *»**p**roces**s** **tree**«* mitgeliefert. Für jeden Prozessnamen soll zusätzlich die jeweilige Prozess-ID angegeben werden. Dafür wird lediglich die Option `-p` am Befehl angehängt. Das `-p` steht für `PID`, also *Process-ID*.[1]

```
1    pstree -p > prozessbaum.txt
```

Die Abbildung 7 zeigt zwei Ausschnitte von Prozessbäumen im Vergleich.

```
abr561@shell-18:~$ pstree -p | head -5
systemd(1)-+-VGAuthService(1045)
           |-accounts-daemon(1036)-+-{accounts-daemon}(1173)
           |                       `-{accounts-daemon}(1246)
           |-agetty(1435)
           |-atd(940)
abr561@shell-18:~$ pstree | head -5
systemd-+-VGAuthService
        |-accounts-daemon---2*[{accounts-daemon}]
        |-agetty
        |-atd
        |-cron
```

Abbildung 7: Ausgabe von `pstree -p | head -5` und `pstree | head -5`

[1] Vgl. Eintrag zu `pstree` auf [UbuUs], abgerufen am 10. April 2019

Der obere Ausschnitt auf Abbildung 7 zeigt einen Prozessbaum, bei dem die Prozess-ID an den jeweiligen Prozessnamen anhängt (gut zu erkennen an der Zahl innerhalb der runden Klammer, wie `systemd(1)` oder `atd(940)`).

Der unter Ausschnitt zeigt hingegen einen Prozessbaum, bei dem keine einzige Prozess-ID angegeben wird. Hier wurde die Option `-p` also nicht verwendet. Außerdem ist zu erkennen, dass die Kindknoten vom Prozess `accounts-daemon` dieselbe Bezeichnung haben, wie der Elternknoten. Ohne die IDs wird der komplette Teilbaum mit `---2*[{accounts-deaemon}]` in einer Menge zusammengefasst. Die Option `-p` bewirkt also auch einen detaillierteren Prozessbaum.

6. Um sich die Datei `prozessbaum.txt` seitenweise ausgeben zu lassen, muss der Befehl `more` verwendet werden. Dieser wurde auf Seite 5 bereits beschrieben.

```
1    more prozessbaum.txt
```

Die Abbildung 8 zeigt den Prozessbaum, der zu 26% mit `more` geöffnet ist.

```
abr561@shell-18:~$ more prozessbaum.txt
systemd(1)-+-VGAuthService(1244)
           |-accounts-daemon(1040)-+-{accounts-daemon}(1152)
           |                       `-{accounts-daemon}(1218)
           |-agetty(1487)
           |-atd(1215)
           |-cron(1084)
           |-dbus-daemon(1088)
           |-lvmetad(524)
           |-lxcfs(1235)-+-{lxcfs}(25072)
           |             |-{lxcfs}(25078)
           |             |-{lxcfs}(24128)
           |             |-{lxcfs}(24129)
           |             |-{lxcfs}(24130)
           |             |-{lxcfs}(24131)
           |             |-{lxcfs}(24132)
           |             |-{lxcfs}(24134)
           |             |-{lxcfs}(24135)
           |             `-{lxcfs}(24136)
           |-master(12240)-+-pickup(6839)
           |               `-qmgr(12251)
           |-networkd-dispat(1051)---{networkd-dispat}(1231)
           |-nscd(1132)-+-{nscd}(1159)
           |            |-{nscd}(1160)
           |            |-{nscd}(1161)
--More--(26%)
```

Abbildung 8: Ausgabe von `more prozessbaum.txt`

7. Um sich alle Dateinamen sowie Rechte- und Größeninformationen der Dateien im aktuellen Arbeitsverzeichnis anzeigen zu lassen, können bereits bekannte Elemente aus den vorherigen Teilaufgaben 1 und 3 verwendet werden.

```
1    ls -lh
```

Optionen können miteinander verknüpft werden. So entsteht aus `-l` und `-h` die kombinierte Option `-lh`. Es können auch mehr als zwei Optionen verknüpft werden, sodass `ls -lahm` entstehen würde – sofern solch eine Abfrage benötigt wird. In diesem Fall wurden nur die nachfolgenden zwei Optionen gebraucht.

-l Diese Option wurde auf Seite 3 vorgestellt und mit Beispielen erläutert. Sie bedeutet: *»Datei-Informationen in Langform ausgeben«*.

-h Diese Option wurde auf Seite 6 vorgestellt und bedeutet *»human readable«*.

```
abr561@shell-18:~$ ls -lh
total 32K
drwx------ 3 abr561 students  512 Mär 27 14:29 Desktop
-rw-r--r-- 1 abr561 students  336 Mär 24 04:01 frename.sh
-rw-r--r-- 1 abr561 students   30 Mär 27 13:12 heute.txt
-rw-r--r-- 1 abr561 students  913 Apr  3 21:58 pb.txt
-rw-r--r-- 1 abr561 students   73 Apr 15 18:26 pf.sh
-rw-r--r-- 1 abr561 students 1,7K Mär 27 13:35 platz.txt
-rw-r--r-- 1 abr561 students   84 Mär 24 15:18 processes.txt
-rw-r--r-- 1 abr561 students 3,7K Apr 10 22:06 prozessbaum.txt
-rw-r--r-- 1 abr561 students  445 Mär 27 12:48 showme.sh
```

Abbildung 9: Ausgabe von `ls -lh`

Der Inhalt einer Datei-Information in Langform sieht wie folgt aus.

- **Rechte** – diese geben die Lese- (`r` für *read*), Schreib- (`w` für *write*) und Ausführungsrechte (`x` für *execute*) an. Unterschieden wird in: `-rw-` Eigentümer, `r--` Gruppe und `r--` alle anderen. Diese formen das Muster `-rw-r--r--`.
- **Hardinks** – hierbei handelt es sich um Einträge im Dateisystem mit einem Namen, der auf den tatsächlichen Speicherplatz einer Datei zeigt.[1]
- **Eigentümer** – hier `abr561` und **Gruppe** – hier `students`
- **Dateigröße** – die Datei `prozessbaum.txt` ist `3,7K` groß
- **Änderungsdatum** – die Datei `prozessbaum.txt` wurde das letzte Mal am `10 Apr` diesen Jahres um `22:06` geändert
- **Dateiname** – zum Beispiel `heute.txt` oder `platz.txt`

[1] Vgl. Eintrag zu `ln` auf [UbuUs], abgerufen am 17. April 2019

8. Um sich alle Dateinamen im aktuellen Arbeitsverzeichnis anzeigen zu lassen, die nur mit dem Buchstaben *»p«* beginnen, kann ebenfalls der `ls` Befehl mit der Option `-d` und dem regulären Ausdruck `p*` verwendet werden.

```
1    ls -d p*
```

Die Option `-d` steht für *»**d**irectory«* und bewirkt, dass nur die Namen der Verzeichnisse und nicht deren Inhalte aufgelistet werden. Das `p*` entspricht einem regulären Ausdruck, der für Namen steht, die mindestens aus einem *»p«* bestehen, gefolgt von einer beliebigen Zeichenkette oder nichts.

Die Abbildung 10 zeigt die Ausgabe von `ls -d p*` und somit alle Dateien im Arbeitsverzeichnis, die mit *»p«* beginnen. Zum Vergleich ist im unteren Abschnitt die Ausgabe von `ls -d *p` zu sehen, die alle Dateien im Arbeitsverzeichnis zeigt, die mit *»p«* enden. In diesem Fall war dies nur eine Datei.

```
abr561@shell-18:~$ ls -d p*
pf.sh  platz.txt  processes.txt  prozessbaum.txt
abr561@shell-18:~$ ls -d *p
Desktop
```

Abbildung 10: Ausgabe von `ls -d p*` und `ls -d *p`

Die nachfolgende Abbildung 11 soll veranschaulichen, wie die Ausgabe beim Verzicht auf die Option `-d` aussehen würde. Die aufgelisteten Dateien sind identisch mit denen, die in dem Verzeichnis `Dekstop` enthalten sind. Ohne die Option würde sich die Ausgabe also nicht nur auf das aktuelle Arbeitsverzeichnis beschränken, sondern auch auf alle Unterverzeichnisse.

```
abr561@shell-18:~$ ls *p
konsole.desktop  MatLab.mat          Office.desktop   Support.desktop
MatLab           myComputer.desktop  Printer.desktop  SuSE.desktop
```

Abbildung 11: Ausgabe von `ls *p`

9. Um ein neues Verzeichnis mit dem Namen `meindir` zu erstellen, wird der *»**m**a**k**e **dir**ectory«* Befehl verwendet. Wird das Verzeichnis wie im nachfolgenden Beispiel erstellt, ist dieses leer. Es ist zu beachten, dass für Verzeichnisnamen dieselben Regeln gelten, wie für Dateinamen, die auf Seite 4 bereits vorgestellt wurden. Außerdem kann eine Überprüfung mit `ls` oder `dir` nützlich sein, um zu kontrollieren, ob das Verzeichnis erfolgreich erstellt wurde.

```
1    mkdir meindir
```

10. Um die Datei `prozessbaum.txt` in das Unterverzeichnis `meindir` zu verschieben, kann der *»**mov**e«* Befehl verwendet werden. Beim Verschieben muss beachtet werden, dass das angegebene Verzeichnis existiert bzw. richtig geschrieben wird. Ist dies nicht der Fall, wird die Datei lediglich in den Namen des vermeintlichen Verzeichnisses umbenannt und nicht verschoben.

```
1  # mv <file> <target_directory>
2  mv prozessbaum.txt meindir
```

Möchte man sicher sein und mögliche Fehler vermeiden, die später schwer zu finden sind, kann auch der Pfad des Zielverzeichnisses angegeben werden.

```
1  # ausgehend vom Homeverzeichnis
2  mv prozessbaum.txt $HOME/meindir
3  mv prozessbaum.txt ~/meindir
4
5  # ausgehend vom aktuellen Arbeitsverzeichnis
6  mv prozessbaum.txt ./meindir
```

11. Um alle Dateipfade von Dateien im aktuellen Arbeitsverzeichnis und dessen Unterverzeichnisse zu durchsuchen – die mit dem Buchstaben *»p«* beginnen – und anschließend ausgeben zu lassen, wird der *»**find**«* Befehl benötigt.

```
1  find . -name "p*"
```

Der `.` nach dem Befehl definiert das Startverzeichnis, ab dem die Suche beginnen soll. Ein Verzeichnispfad wäre ebenfalls denkbar, jedoch steht der Punkt für das *aktuelle Arbeitsverzeichnis*. Die Option `-name` entspricht einer Option und ist keine Verknüpfung aus vier. Diese bewirkt, dass nach Dateinamen gesucht wird – unter Berücksichtigung der Groß- und Kleinschreibung. Anders als bei der Teilaufgabe 8 muss hier `"p*"` in Anführungsstrichen stehen. Mit `-name` kann mit `find . -name showme.sh` auch nach einer bestimmten Datei gesucht werden. In diesem Fall sind die Anführungsstriche nicht mehr notwendig.

12. Um in das in der Teilaufgabe 9 erstellte Unterverzeichnis `meindir` zu wechseln, kann der *»**c**hange **d**irectory«* Befehl unter Berücksichtigung der Groß- und Kleinschreibung verwendet werden. Mit dem `cd` Befehl kann auch in ein beliebiges Verzeichnis oder Unterverzeichnis von jedem anderen Verzeichnis aus gewechselt werden, sofern der Pfad angegeben wird. Für den Wechsel in das Unterverzeichnis `meindir` reicht der nachfolgende Befehl völlig aus.

```
1  cd meindir
```

Wird ein Verzeichnis beim Aufruf des `cd` Befehls nicht gefunden, wird der Hinweis zurückgegeben, dass jenes Verzeichnis nicht existiert.

```
1  cd <foreign_directory>
2  -bash: cd: <foreign_directory>: No such file or directory
```

13. Um die Textdatei `prozessbaum.txt` in eine neue Textdatei mit dem Namen `pb.txt` zu kopieren, kann der *»**co**py«* Befehl verwendet werden. Mit diesem Befehl können Dateien aber auch ganze Verzeichnisse kopiert werden.

```
1  cp prozessbaum.txt pb.txt
```

Die Kopie darf nicht den identischen Dateiname haben, wie die zu kopierende Datei. Sollte dies doch geschehen, wird der Kopiervorgang abgebrochen und mit einem Hinweis quittiert, die nachfolgend zu sehen ist.

```
1  cp pb.txt pb.txt
2  cp: 'pb.txt' and 'pb.txt' are the same file
```

Beim Kopiervorgang von Verzeichnissen ist zu beachten, dass stets die Option `-r` für *»**r**ecursive«* angegeben werden muss. Hierbei spielt es keine Rolle, ob das Verzeichnis leer ist, Dateien oder Unterverzeichnisse enthält.

```
1  cp meindir -r new_meindir
```

14. Um die in der Teilaufgabe 5 erstellten Datei `prozessbaum.txt` aus dem Verzeichnis `meindir` zu löschen, kann der *»**rem**ove«* Befehl verwendet werden. Dieser ist gut geeignet, um Dateien oder komplette Verzeichnisse zu löschen.

```
1  rm prozessbaum.txt # löschen einer Datei
```

Der Vollständigkeit halber wird abschließend das Löschen eines Verzeichnisses gezeigt. Dieser Vorgang ist nicht analog zum Löschen einer Datei. Hier muss der *»**rem**ove **-r**ecursive«* oder der *»**rem**ove **dir**ectory«* Befehl verwendet werden.

```
1  # fehlerhaftes Löschen eines Verzeichnisses mit Fehlermeldung
2  rm meindir
3  rm: cannot remove 'meindir': Is a directory
4
5  # richtiges Löschen eines Verzeichnisses
6  rm -r meindir || rmdir meindir
```

b) Fragen zur bash – Shell

– *Was enthalten die nachfolgenden Umgebungsvariablen?*

* **`$HOME`**

  Die Umgebungsvariable `$HOME` beinhaltet den Pfad zum Homeverzeichnis. Nur hier hat ein Benutzer volle Schreib- und Leserechte, daher sollten alle Daten hier abgespeichert werden. Außerdem ist zu beachten, dass Linux anders als Windows *case sensitive* ist. Das bedeutet, dass die Dateien `heute.txt` und `Heute.txt` zwei völlig unterschiedliche Dateien sind.

  Wird nur `$HOME` eingegeben, wird der nachfolgende Hinweis geliefert.

```
1  $HOME
2  -bash: /home/students/abr561: Is a directory
```

  Wird die Umgebungsvariable `$HOME` in Kombination mit dem `echo` Befehl eingegeben, dann wird die Information, welche die Umgebungsvariable hält, ausgelesen und auf der Standardausgabe geliefert.

```
1  echo $HOME
2  /home/students/abr561
```

* **`$PATH`**

  Wird die Umgebungsvariable `$PATH` in Kombination mit dem `echo` Befehl eingegeben, dann werden alle Pfade geliefert, welche die Umgebungsvariable hält. Außerdem ist `$PATH` erweiterbar (siehe Seite 22).

```
1  echo $PATH
2  /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
       /bin:/usr/games:/usr/local/games:/snap/bin
```

  Die Pfade werden durch ein »:« getrennt dargestellt. Aufgestückelte ergibt dies neun Einzelpfade, welche `$PATH` zu diesem Zeitpunkt enthält.

```
1  /usr/local/sbin
2  /usr/local/bin
3  /usr/sbin
4  /usr/bin
5  /sbin
6  /bin
7  /usr/games
8  /usr/local/games
9  /snap/bin
```

* **`$UID`** und **`$USER`**

  Die Umgebungsvariable `$UID` enthält die Benutzerkennungsnummer (auch *»user identifier«*). Der UID steht die GID gegenüber, also die Gruppen-Erkennungsnummer (auch *»group identifier«*).

  Die Umgebungsvaribale `$USER` enthält den aktuellen Benuzternamen, der stets mit `echo $USER` abgefragt werden kann. Zusätzlich steht dieser immer als erstes vor der Eingabeaufforderung: `abr561@shell-18:~$` ...

```
1  echo $UID
2  32190
3
4  echo $USER
5  abr561
```

– *Was bewirkt der Befehl `cd ~`? Gibt es eine einfachere Alternative?*

Mit dem Befehl `cd ~` wird in das Homeverzeichnis gewechselt. Eine einfachere Alternative wäre der Aufruf von `cd` ohne weitere Angaben die darauf folgen.

```
1  cd # Wechsel in das Homeverzeichnis
```

Neben den Aufruf `cd` oder von `cd ~` gibt es noch zwei weitere Möglichkeiten, die aber umständlicher sind, als die zwei bisherigen Varianten.

```
1  cd $HOME # Wechsel mit Umgebungsvariable
2  cd /home/students/abr561 # Wechsel mit Pfadangabe
```

– *Welche Funktion haben die nachfolgenden Tastatureingaben im Terminal, sofern die Eingabzeile (auch Prompt genannt) leer ist?*

* ↑ oder ↓

  Mit Hilfe der Pfeiltasten hat man die Möglichkeit, durch vergangene Befehle *durchzublättern*, die man bereits eingegeben hat.

  Mit ↑ blättert man vom neusten zum ältesten Befehl nach oben. Man erreicht also den chronologisch ältesten Befehl, je höher man kommt. Dieser Effekt hat auch bestand, nachdem das Terminal erneut geöffnet wurde.

  Mit ↓ blättert man vom ältesten wieder zurück zum neusten Befehl nach unten. Man erreicht also den chronologisch neusten Befehl, je tiefer man kommt. Ist die Zeile wieder leer und hat man den zuletzt eingegebenen Befehl übersprungen, so hat diese Taste keinen weiteren Effekt mehr.

* Strg + D

  Mit dieser Tastenkombination wird die Eingabeaufforderung geschlossen und ist somit vergleichbar mit Alt + F4 bzw. ⌘ + Q für offene Fenster.

2 Shell-Skripte

a) Um ein Skript zu schreiben, welches alle aktuell laufenden Prozesse nach einem übergebenen String durchsucht, werden unterschiedliche Elemente benötigt, die nachfolgend kurz erklärt und anschließend in einem fertigen Skript gezeigt werden.

- **`#!/bin/bash`**

  Diese Zeile bewirkt, dass das Skript als *bash-Skript* interpretiert wird. Daher wird sie auch als *»Shebang«* oder *»Magic Line«*[1] bezeichnet und meint die Shell, mit der das Skript ausgewertet wird. Das *Shebang* steht stets in Zeile 1.

- **`inputstring=$1`**

  Diese Zeile entspricht einer Variablendeklaration und bewirkt, dass eine Variable mit dem Namen `inputstring` innerhalb des Skripts eingeführt wird. Außerdem wird der Variablen mit `=` der Wert `$1` zugewiesen. `$1` enthält zu diesem Zeitpunkt noch keinen Wert, erst wenn das Skript ausgeführt wird. Bei der Ausführung muss ein String angehängt werden, der an die Variable übergeben werden kann. Dieser ist an der Position `$1`. Der an der zweiten Position würde mit `$2` deklariert werden und der an der dritten mit `$3` und so weiter.

- **`ps -ef`**

  Der *»**p**rocesse**s**«* Befehl liefert eine Momentaufnahme aller laufenden Prozesse. Da dieser Befehl im Skript integriert ist und mit einem Pipe[2] mit anderen Befehlen verknüpft wird, gewährleistet es immer die aktuellste Momentaufnahme aller Prozesse, ohne eine separate Datei zu erzeugen. Kurz zu den Optionen:

  `-e` Diese Option wählt *»alle«* Prozesse aus (identisch mit der Option `-A`).

  `-f` Diese Option erzeugt eine detailliertere Ausgabe.

- **`grep $inputstring`**

  Der *»**grep**«*[3] Befehl durchsucht Dateien oder Ausgabe nach bestimmten Textstücken. Das Suchmuster entspricht einem regulären Ausdruck und die Ergebnisse werden auf der Standardausgabe geliefert oder in eine Datei gespeichert.

  Der String, nach dem gesucht werden soll, wird nach dem `grep` Befehl notiert. Da bereits eine Variable mit dem Namen `inputstring` deklariert und mit einem übergebenen String initialisiert wurde, kann mit `$inputstring` auf diese Variable und dementsprechend auch auf den String zugegriffen werden.

---

[1] Vgl. [Sch16]: *»Ein erstes Shell-Skript, der Skript-Start und das Shebang«*, Min.: 4:55 bis 6:07

[2] Auch *Befehlsverkettung* genannt mit Hilfe des `|` Symbols

[3] Das Programm `grep` steht für *»**g**lobal search for a **r**egular **e**xpression and **p**rint out matched lines«*

- **`grep -v "grep"`**

  Diese Anweisung ist für dieses Sktipt nicht relevant, verschönert aber die Ausgabe. Die *»invert«* Option `-v` bewirkt, dass die Suchergebnisse *umgkehrt* werden. Es werden also alle Ergebnisse *nicht* angezeigt, die einen Treffer auf eine Suchanfrage haben. Damit wird die Suchanfrage *»grep«* zwar gefunden, aber in der Ausgabe nicht als aktuell laufender Prozess angezeigt.

- **`if (( $# == 1 )) then ... else ... fi`**

  Eine `if`-Abfrage ummantelt den Code, die mit `$#` die Anzahl der übergebenen Parameter überprüft. Diese muss genau eins entsprechen, andernfalls wird ein Satz mit der korrekten Benutzung des Skripts zurückgegeben.

Ein fertiges Skript `pf.sh` könnte für diese Anforderung folgendermaßen aussehen.

```
1  #!/bin/bash
2
3  inputstring=$1
4
5  if (( $# == 1 ))
6  then
7      ps -ef | grep $inputstring | grep -v "grep"
8  else
9      echo "Usage: pf STRING"
10 fi
```

Wird das Skript mit dem übergebenen String `init` ausgeführt, werden als Momentaufnahme die nachfolgenden Zeilen und Werte ausgegeben (siehe Abbildung 12).

```
abr561@shell-18:~$ ./pf.sh init
root         1     0  0 Apr12 ?        00:00:29 /sbin/init maybe-ubiquity
abr561   14254 14246  0 17:18 pts/3    00:00:00 /bin/bash ./pf.sh init
```

Abbildung 12: Ausgabe von `./pf.sh init`

Betrachten wir die Werte auf der Abbildung 12 etwas genauer. Die einzelnen Spalten sind nun die Zeilen, die in der nachfolgenden Tabelle dargestellt sind.

UID	root	abr561
PID	1	14254
PPID	0	14246
C	in beiden Fällen 0	
STIME	Apr12	17:18
TTY	?	pts/3
TIME	00:00:29	00:00:00
CMD	/sbin/init maybe-ubiquity	/bin/bash ./pf.sh init

b) `frename.sh`

```
1  #!/bin/bash
2
3  inputdir=$1
4  inputstring=$2
5
6  if (( $# == 2 ))
7  then
8      for file in $inputdir/*
9      do
10         if [[ "$file" == *.* ]]
11         then
12             filename=$(echo "$file" | cut -d '.' -f1)
13             fileextension=$(echo "$file" | cut -d '.' -f2)
14             mv "$file" "$filename""$inputstring"".""$fileextension
                   "
15         else
16             mv "$file" "$file""$inputstring"
17         fi
18     done
19 else
20     echo "Usage: frename UNTERVERZEICHNIS STRING"
21 fi
```

Zeile 1 bis 7, 18 bis 21 Hintergründe bereits bekannt aus dem vorherigen Shell-Skript (vgl. Zeile 1 bis 6 und 8 bis 10 der `pf.sh` auf der vorherigen Seite).

Zeile 8 bis 18 Ab Zeile 8 beginnt eine `for`-Schleife, die in Zeile 18 beendet wird. Sie bewirkt, dass jede Datei im angegebenen Verzeichnis `$inputdir` *betrachtet* und *manipuliert* wird (in dem Fall des Skripts `frename.sh` also *umbenannt*). Die Verwendung von `*` in `$inputdir/*` vervollständigt somit jeden Dateipfad in dem angegebenen Verzeichnis (wie `~/frename.sh` für das Homeverzeichnis).

Zeile 10 bis 17 Bevor etwas mit den Dateien passiert, findet in Zeile 10 eine `if`-Abfrage statt, die mit dem regulären Ausdruck `*.*` überprüft, ob der Dateiname einen Punkt enthält. Ist dies *true*, handelt es sich um eine Dateierweiterung, wie `*.sh`, `*.txt` oder Ähnliches. Ist dies der Fall, springt die Ausführung in den `then`, ansonsten in den `else` Abschnitt.

Zeile 12 bis 14 Befindet sich ein Punkt im Namen der Datei, dann muss der neue Name *gebaut* werden. Dies geschieht in den Zeilen 12 bis 14. Zunächst wird der Name der Datei – also der Teil, der sich vor dem Punkt befindet – mit `cut -d '.' -f1` abgeschnitten und in eine Variable `filename` zwischengespeichert. Dasselbe geschieht mit der Dateierweiterung. Das `-f1` meint also den Teil vor dem Punkt und das `-f2` den Teil danach. Das `-d` steht für *delimiter*, also *Trennzeichen*. In Zeile 14 wird der neue Name zusammengesetzt und der alte mit `$filename` + `$inputstring` + `.` + `$fileextension` überschrieben.

Zeile 15 bis 17 Befindet sich kein Punkt im Namen der Datei, handelt es sich um ein Verzeichnis. In diesem Fall wird lediglich der `$inputstring` an den alten Namen angehängt. Da hier mit Strings gearbeitet wird, spielt es keine Rolle, ob ein Leerzeichen im Dateinamen vorhanden ist oder nicht. Der Inputstring wird immer an das Ende des Namens angehängt – auch wenn keine Dateierweiterung vorhanden ist.

c) Um den Status eines Skripts auf *»ausführbar«* zu ändern, wird der *»**ch**ange **mod**e«* Befehl benötigt. Dadurch kann jedes Skript als Programm gestartet werden, sodass der Aufruf von `bash` nicht mehr voran gestellt werden muss. Anstelle von `bash` tritt ein »`./`«, sofern sich das Skript im aktuellen Arbeitsverzeichnis befindet. Andernfalls muss anstelle von »`./`« der gesamte Pfad zum Skript angegeben werden.

```
1    chmod a+x pf.sh
2    chmod a+x frename.sh
```

Der Aufruf von `chmod` bewirkt, dass sich die Rechte einer Datei verändern. Diese Veränderung hat auch nach einem Neustart der Eingabeaufforderung bestand. Das `a` von `a+x` steht für *»**a**ll«* und das `x` für *»e**x**ecute«*. Mit diesem Befehl wird das jeweilige Skript also für alle Benutzer (Eigentümer, Gruppe, Rest) ausführbar gemacht.

Ob die Datei erfolgreich ausführbar gemacht wurde, kann mit dem aus der Teilaufgabe 1.a.7. bekannten Befehl (siehe Seite 10) getestet werden. Das Muster in der ersten Spalte für die Rechte hat sich von `-rw-r--r--` in `-rwxr-xr-x` geändert.

```
abr561@shell-18:~$ ls -lh
total 32K
drwx------ 3 abr561 students  512 Mär 27 14:29 Desktop
-rwxr-xr-x 1 abr561 students  336 Apr 17 14:14 frename.sh
-rw-r--r-- 1 abr561 students   30 Mär 27 13:12 heute.txt
-rw-r--r-- 1 abr561 students  913 Apr  3 21:58 pb.txt
-rwxr-xr-x 1 abr561 students   73 Apr 15 18:26 pf.sh
-rw-r--r-- 1 abr561 students 1,7K Mär 27 13:35 platz.txt
-rw-r--r-- 1 abr561 students   84 Mär 24 15:18 processes.txt
-rw-r--r-- 1 abr561 students 3,7K Apr 10 22:06 prozessbaum.txt
-rw-r--r-- 1 abr561 students  445 Mär 27 12:48 showme.sh
```

Abbildung 13: Ausgabe von `ls -lh` nach `chmod a+x`

d) Die Umgebungsvariable `$PATH` kann mit einem kleinen Befehl erweitert werden. Wie dieser aussieht, damit immer das momentan aktuelle Verzeichnis in der Umgebungsvariablen `$PATH` gehalten wird, soll das nachfolgende Beispiel zeigen.

```
1     PATH=$PATH:"." # alternativ auch nur . statt "."
```

Beim Aufruf von `echo $PATH` wird nun eben dieser Punkt an letzter Stelle angezeigt.

Achtung:

Diese Erweiterung geht nach einem Neustart der Eingabeaufforderung verloren.

Literatur und Quellen

Literatur

[Gün14] Günther, Karsten: *Bash kurz & gut*. O'Reilly Verlag, Köln. 3. Auflage, 2014, ISBN: 978-3-95561-764-6.

[MR19] Mandl, Prof. Dr. Peter und Rottmüller, M. Sc. Björn: *Grundlagen der Bash-Programmierung – Begleitmaterial zu den Übungen im Kurs Wirtschaftsinformatik II*, Version 4.0.0 online unter

`https://www.wirtschaftsinformatik-muenchen.de/mitarbeiter/peter-mandl/lehrveranstaltungen/wirtschaftsinformatik/`

(letzter Download am 19. März 2019)

[WK13] Wolf, Jürgen und Kania, Stefan: *Shell-Programmierung – Das umfassende Handbuch*. Galileo Press Computing, Bonn. 4., aktualisierte und erweiterte Auflage, 2013, ISBN: 978-3-8362-2310-2.

Das eBook ist in zweiter Auflage kostenfrei erhältlich unter:

`http://linuxint.com/DOCS/Linux_Docs/openbook_shell/shell_001_000.htm#Xxx999207`

Webseiten

[UbuUs] Ubuntuusers, online unter `https://wiki.ubuntuusers.de/Startseite/` (letzter Abruf am **. April 2019).

[BaWi] Bash, Wikipedia unter `https://de.wikipedia.org/wiki/Bash_(Shell)` (abgerufen am 26. März 2019).

Sonstige

[Sch16] Schürmann, Tim: *Shell-Programmierung lernen – Erste Schritte bei der Automatisierung unter Linux/Unix.*. Video2Brain – LinkedIn company, Graz, Österreich, 2016. Mehr Infos unter

`https://de.linkedin.com/learning/shell-programmierung-lernen`

(letzter Abruf am 30. März 2019).