Der Umgang mit Linux[1]

1 Häufig verwendete Befehle der Kommandozeile

In dieser Aufgabe werden Befehle der *Kommandozeile* ausprobiert. Häufig gibt es alternative grafische Werkzeuge, die dasselbe tun.

Hinweis: Diese Aufgabe können Sie problemlos auf Ihrem Linux Rechner oder unter macOS bearbeiten. Erst ab Aufgabe 3 benötigen Sie die VM, die unter Emil bereitgestellt ist. Trotzdem ist es sicherlich hilfreich, wenn Sie jetzt schon die VM installieren und sich bei Problemen an Malte Nogalski wenden.

Aufgabe: Im Netz sind eine Reihe von Anleitungen zur Shell-Programmierung verfügbar. Arbeiten Sie eine einführende Anleitung zur bash Shell durch, bevor Sie diese Aufgabe bearbeiten. (`https://de.wikibooks.org/wiki/Linux-Praxisbuch:_Shellprogrammierung` ist gut geeignet.)

In diesen Boxen finden Sie kleine Aufgaben. Dokumentieren Sie die Ergebnisse dieser Aufgaben so, dass Sie die Aufgaben auf dieser Basis problemlos erneut ausführen können.

Hinweis: Falls Sie mit den VMs auf den USB Platten im Labor arbeiten, speichern Sie relevante Daten auf dem Shared Folder, so dass Sie die Daten auf Ihrem Z Laufwerk finden. Die Daten der USB Festplatten werden nicht gesichert.

1. 1 Hilfe

Die Programme `info, man, apropos` bieten Hilfe zu (fast) allen Konsolenprogrammen. Noch mehr Information steht im Verzeichnis `/usr/share/doc`.

Eine Linux Befehlssammlung finden Sie hier: `http://www.pc-erfahrung.de/linux/linux-befehle.html`

1. 2 script: Aktionen aufzeichnen

Mit diesem Befehl werden die Ein- und Ausgaben des Terminals in einer Datei protokolliert.

[1] Diese Aufgabe basiert auf Unterlagen von Prof. Dr. W. Fohl, HAW Hamburg

1. 3 mkdir: Erzeuge Verzeichnis

`mkdir` erzeugt im aktuellen Verzeichnis ein Unterverzeichnis. Somit erzeugt der folgende Befehl im Home-Verzeichnis das Unterverzeichnis `Bs_Prakt`.

```
franzkorf@linux-146z:~> mkdir ~/Bs_Prakt
franzkorf@linux-146z:~>
```

Das Zeichen `~` steht für das Home-Verzeichnis. In diesem Beispiel hätte man auch tippen können: `mkdir /home/bs/Bs_Prakt`.

Anmerkung: Unix-Filesysteme unterscheiden Groß- und Kleinschreibung.

Aufgabe:

- Zeichnen Sie alles auf, was Sie ab jetzt tun. Schauen Sie sich die Aufzeichnungen nach einigen Eingaben probeweise an.
- Was ist *Tab-Expansion* und was nützt Ihnen das bei der Arbeit mit der Kommandozeile?
- Was erhalten Sie beim Drücken der Tastenkombination <Alt><.>?

1. 4 cd: Verzeichnis wechseln

```
franzkorf@linux-146z:~> cd ~/Bs_Prakt
franzkorf@linux-146z:~/Bs_Prakt>
```

1. 5 pwd: Position im Verzeichnisbaum?

```
franzkorf@linux-146z:~> pwd
/home/franzkorf
franzkorf@linux-146z:~>
```

1. 6 whoami: Wer bin ich?

```
franzkorf@linux-146z:~> whoami
franzkorf
franzkorf@linux-146z:~>
```

1. 7 ls: Verzeichnisinhalt anzeigen

Bitte machen Sie sich mit den Optionen von diesem Befehl vertraut. Führen Sie folgenden Befehl aus:

```
franzkorf@linux-146z:~>
franzkorf@linux-146z:~> ls -l /etc
total 2296
```

```
-rw-r--r--  1 root root     15161 2010-07-05 22:01 a2ps.cfg
-rw-r--r--  1 root root      2565 2010-07-05 22:01 a2ps-site.cfg
-rw-r--r--  1 root root        25 2010-07-01 17:27 aclocal_dirlist
drwxr-xr-x  3 root root      4096 2010-07-06 08:13 acpi
-rw-r--r--  1 root root        44 2011-01-20 19:47 adjtime
... usw.
franzkorf@linux-146z:~>
```

Aufgabe:

- Geben Sie das Verzeichnis nach *Erweiterung* sortiert aus.
- Geben Sie das Verzeichnis nach *Modifikationszeit* sortiert aus.
- Kehren Sie für beide Sortiervarianten die *Reihenfolge* um.
- Geben Sie das Verzeichnis *rekursiv*, (d.h. mit allen Unterverzeichnissen) aus.

1. 8 Unix-Verzeichnisstruktur

/bin	Ausführbare Systemprogramme
/boot	Kernel
/dev	Geräte-Pseudodateien
/etc	Konfigurationsdaten
/home	Benutzerdaten
/lib	System-Bibliotheken und Kernelmodule
/lost+found	Fundsachen nach Dateisystemcheck
/media	Externe Datenträger
/mnt	Temporär eingehängte Datenträger
/proc	Pseudo-Dateisystem mit Prozess- (und zur Zeit noch Kernel-informationen – siehe sys)
/root	Heimatverzeichnis des Benutzers root
/sbin	Administrative Systemprogramme
/srv	Dateien für Server-Dienste
/sys	Kernel- und sonstige Systeminformationen
/tmp	Temporäre Dateien (werden beim Herunterfahren des Systems gelöscht)
/usr	Unix System Resources
/usr/bin	Anwendungsprogramme
/usr/lib	Anwendungsbibliotheken
/usr/share	Daten für Anwendungsprogramme
/usr/share/doc	Dokumentation
/usr/local	Dieselbe Unterverzeichnisstruktur noch mal für selbst installierte Programme
/var	Veränderliche Daten
/var/log	System-Protokolldateien

Aufgabe: Schauen Sie sich insbesondere den Inhalt des `/proc` Verzeichnisses an. Informieren Sie sich, wie die im `/proc` Verzeichnis dargestellten Daten im Prinzip erzeugt werden.

1. 9 less, more, cat: Textdateien anzeigen

```
franzkorf@linux-146z:~/Bs_Prakt> cat ~/.bashrc
# Sample .bashrc for SuSE Linux
# Copyright (c) SuSE GmbH Nuernberg

# There are 3 different types of shells in bash: the login shell, normal shell
# and interactive shell. Login shells read ~/.profile and interactive shells
# read ~/.bashrc; in our setup, /etc/profile sources ~/.bashrc - thus all
# settings made here will also take effect in a login shell.
#
# NOTE: It is recommended to make language settings in ~/.profile rather than
# here, since multilingual X sessions would not work properly if LANG is over-
# ridden in every subshell.

... usw.

franzkorf@linux-146z:~/Bs_Prakt>
```

Das ist OK für kurze Dateien. Für längere Dateien nimmt man den Pager `less` oder den einfachen Standard-Pager `more`.

1. 10 E/A-Umleitung und Pipes

Man kann die Standardausgabe eines Programms von der Konsole in eine Datei umleiten:

```
ls /etc > my_listing.txt
```

Wenn Sie die Ausgabe an eine Datei *anhängen* wollen, verwenden Sie den Operator `>>`:

```
ls /bin >> my_listing.txt
```

Auf dieselbe Weise kann man die Standard*eingabe* umleiten:

```
sort -r < my_listing.txt
```

Mit dem Operator `<<` kann man sogenannte *Here-files* erzeugen:

```
cat <<EOF
> Saemtlicher Text, der hier steht,
> wird ausgegeben, bis eine Zeile kommt,
> in der EOF (oder was auch immer oben angegeben wurde)
> am Beginn der Zeile steht
> EOF
Saemtlicher Text, der hier steht,
wird ausgegeben, bis eine Zeile kommt,
in der EOF (oder was auch immer oben angegeben wurde)
am Beginn der Zeile steht
```

Das verwendet man gerne in Skript-Dateien um längere Texte auszugeben.

Man kann die Standardeingabe eines Programms mit Hilfe des *Pipe-Symbols* `|` mit der Standardausgabe eines anderen Programms verbinden:

```
ls -l | sort -rnk5
```

Aufgabe: Erläutern Sie die in diesem Beispiel verwendeten Optionen von sort.

1. 11 cp: Kopieren, mv: Verschieben, ln: Verlinken, rm: Löschen

Erzeugen Sie eine kleine Textdatei:

```
cat << EOF > text1.txt
Hallo, dies
ist etwas
Text!
EOF
```

Kopieren Sie die Datei wie folgt:

```
cp text1.txt text2.txt
cp text1.txt text3.txt
cp text1.txt text4.txt
```

Mit dem Befehl `mv` werden Dateien verschoben bzw. umbenannt:

```
mv text4.txt text04.txt
```

Mit dem Befehl `ln` erzeugen Sie einen *Link* auf eine Datei. Mit dem Befehl `ln -s` erzeugen Sie einen *symbolischen Link* auf eine Datei:

```
ln -s text2.txt ltext2.txt
ln text3.txt ltext3.txt
```

Mit dem Befehl `rm` löschen Sie eine Datei

```
rm text04.txt
```

Aufgabe:

- Dokumentieren Sie mit `ls -l` das Resultat Ihrer Aktionen
- Editieren Sie `ltext2.txt`: Wie verändert sich `text2.txt`?
- Editieren Sie `ltext3.txt`: Wie verändert sich `text3.txt`?
- Was passiert, wenn Sie `text2.txt` löschen?
- Was passiert, wenn Sie `text3.txt` löschen?

1. 12 Shell-Sonderzeichen

Bei der Angabe von Pfadnamen (Dateinamen) können Sie Sonderzeichen verwenden:

Zeichen	**Bedeutung**	Beispiel
`*`	beliebiger Text („*“ entspricht dem Zeichen *)	`ls ~/Vorlesg/*/Skript`
`?`	Ein beliebiges Zeichen („\?“ entspricht dem Zeichen ?)	`ls text?.txt`
`[aei]`	Auswahl von Zeichen (hier: a, e, i)	`ls text[23].txt`
`[a-z]`	Bereich von Zeichen (hier: Kleinbuchstaben)	`ls version[1-4].c`

Aufgabe: Demonstrieren Sie die Platzhalterzeichen mit eigenen Beispielen.

1. 13 grep: Nach Text suchen

`grep` ist ein mächtiges Werkzeug zur Suche in Texten:

```
grep EDITOR .bashrc
# Some applications read the EDITOR variable to determine your favourite text
#export EDITOR=/usr/bin/vim
#export EDITOR=/usr/bin/mcedit
```

grep wird gerne in Kombination mit Pipes verwendet:

```
wolfo@waas:shared$ ls /etc/ | grep fs
ffserver.conf
fstab
fstab~
fstab-2009-02-16
fstab-2009-02-24
gnome-vfs-2.0
gnome-vfs-mime-magic
initramfs-tools
login.defs
mke2fs.conf
wolfo@waas:shared$ ls /etc/ | grep fs$
login.defs
wolfo@waas:shared$ ls /etc/ | grep ^fs
fstab
fstab~
fstab-2009-02-16
fstab-2009-02-24
wolfo@waas:shared$ ls -l /etc/ | grep ^fs
wolfo@waas:shared$ ls -l /etc/ | grep "\<fs"
-rw-r--r--  1 root root     1999 16. Sep 19:46 fstab
-rw-r--r--  1 root root     1996 16. Sep 19:43 fstab~
-rw-r--r--  1 root root      521 16. Feb 2009  fstab-2009-02-16
-rw-r--r--  1 root root      687 24. Feb 2009  fstab-2009-02-24
```

In dem oberen Listing sehen Sie ein paar einfache Beispiele von *regulären Ausdrücken*, die grep so mächtig machen.

Aufgabe: Finden Sie heraus, was die Wirkung der Zeichen `\$`, `^` und `\<` ist. Warum musste der Suchausdruck im letzten Beispiel in Anführungszeichen (*quotes*) gesetzt werden?

1. 14 ps, pstree: Laufende Prozesse anzeigen

```
franzkorf@linux-146z:~/Bs_Prakt> ps
  PID TTY          TIME CMD
12339 pts/0    00:00:00 bash
24317 pts/0    00:00:00 ps
franzkorf@linux-146z:~/Bs_Prakt> pstree
  ...
franzkorf@linux-146z:~/Bs_Prakt>
```

Probieren Sie die Optionen `ps a` und `ps aux`.

Aufgabe: Geben Sie alle Prozesse aus, deren Kommandozeile mit k *beginnt*.

1. 15 htop, top: Interaktive Prozessanzeige

Mit `top` oder `htop` können Sie interaktiv die Prozessliste durchstöbern. `info htop` liefert weitere Informationen.

Hinweis: Das Programm htop müssen Sie erst installieren:

```
sudo zypper install htop
```

1. 16 Ausführen eigener Programme

Bitte verwenden Sie das folgende Programm mit den Namen mockup.c

```
fohl@FOHL-PC:tmp$ cat << EOF > hello.c
> #include <stdio.h>
> #include <unistd.h>
>
> int main(int argc, char *argv[]) {
>    // Ausgabe des Aufrufs des Programms
>    for (int i = 0; i < argc; i++) {
>       fprintf(stdout, "%s ", argv[i]);
>    }
```

```
>     fprintf(stdout, "\n");
>     fprintf(stdout, "VMEM_PAGESIZE=%d\n", VMEM_PAGESIZE);
> #ifdef STOP_BEFORE_END
>     pause();
> #endif
>     return 0;
> }
> EOF
```

Geben Sie das Programm via `cat` oder mit Ihrem Lieblingseditor ein. Machen Sie sich mit den einzelnen Funktionen, die in dem Programm verwendet werden, vertraut. Dazu können Sie `man` verwenden.

Im Programm werden die `#define` Variablen `STOP_BEFORE_END` und `VMEM_PAGESIZE` verwendet. Sie sind im Programm selbst nicht definiert. `#define` Variablen können beim Aufruf des C Compilers mit `-D` übergeben werden. Schlagen Sie mit Hilfe von `man` dies in der Dokumentation nach.

Mit `-o` übergeben Sie dem Compiler den Namen der Ausgabedatei. Übersetzen Sie das Programm so, dass `STOP_BEFORE_END` definiert und `VMEM_PAGESIZE` den Wert 32 hat. Die ausführbare Datei des Programms soll `p1` heißen.

Überprüfen Sie wie folgt, ob das Programm `p1` erzeugt wurde.

```
fohl@FOHL-PC:tmp$ ls -l p1
-rwxr-xr-x 1 fohl fohl 6598 29. Sep 15:27 p1
```

Das Listing zeigt durch die Buchstaben `x`, dass `p1` ausführbar ist.

Starten Sie das Programm mit `./p1`. Was beobachten Sie?

Hinweis: In der Unix-Welt ist aus Sicherheitsgründen das aktuelle Verzeichnis *nicht* im Pfad der ausführbaren Programme enthalten!

1. 17 Diagnosetool ldd: Welche Bibliotheken benötigt ein Programm?

```
bs@linux-je7b:~/Schreibtisch> ldd p1
        linux-vdso.so.1 (0x00007ffef7d89000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f8846cb5000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f884706f000)
bs@linux-je7b:~/Schreibtisch>
```

1. 18 Diagnosetool strace: Systemaufrufe protokollieren

Das Zauberwerkzeug zum Lokalisieren von Problemen:

```
bs@linux-je7b:~/Schreibtisch> strace ./p1
execve("./p1", ["./p1"], 0x7ffe279d44e0 /* 95 vars */) = 0

... Jede Menge Text ...
```

```
write(1, "./p1 \n", 6./p1
)                   = 6
write(1, "VMEM_PAGESIZE=32\n", 17VMEM_PAGESIZE=32
)       = 17
...
```

1. 19 Diagnosetool dmesg: Kernel-Meldungen der aktuellen Sitzung ansehen

`dmesg` gibt aktuelle Meldungen des Kernels aus. Diese Meldungen schreibt der Kernel auf einem normal konfigurierten System in die Datei `/var/log/messages`. Ein direktes Auslesen z.B. mit `less /var/log/messages` scheitert, weil nur root Leserechte hat. Normaluser verwenden den Befehl `dmesg`, der die Kernelmeldungen der aktuellen Sitzung ausgibt (Tipp: Ausgabe durch `less` pipen). Will man nur die neuesten Meldungen haben, *diese aber fortlaufend*, gibt man ein:

```
dmesg | tail -f
```

Beenden der Anzeige mit <Strg><C>

Aufgabe: Beantworten Sie folgende Fragen:

- Was ist der Unterschied zwischen einer Shell- und einer Umgebungsvariablen (environment variable)?
- Welche Information enthalten die Umgebungsvariablen $HOME, $PATH, $UID und $USER?
- Was bewirkt der Befehl `cd \$HOME`? Gibt es eine einfachere Alternative?
- Welche Funktion hat die TAB-Rechts Taste bei der Eingabe eines nicht vollsttändigen Dateinamens oder eines nicht vollständigen Programmnamens?
- Welche Funktionen haben die Tasten <Pfeil-oben> und <Pfeil-unter>, wenn noch kein Befehl eingegeben wurde?
- Welche Funktion hat der history Befehl?
- Was ist die Funktion der `.bashrc` Datei im Verzeichnis`\$HOME`?
- Modifizieren Sie die Umgebungsvariable `PATH` so, dass ein Programm zuerst im aktuellen Verzeichnis gesucht wird.

2 Vorgänge automatisieren: Shellskripte

Hier ist das Gerüst eines Shellskripts:

```
#!/bin/bash
# The third shell script
# <Your name>
# <Date>

usage()
{
cat <<EOF
  $0  [OPTIONS]
  Asking the user for her or his name and display a greeting
  OPTIONS:
  -h    --help          Display this help
EOF
}
# ----------------------------------------------------------------
ask_for_name()
{
    echo "Please enter your name:"
    read user_name
}
# ################################################################
#                    main

# check for options
## note the blanks after '[' and before ']'
if [ $# -lt 1 ]; then
    # No option, so ask for name
    ask_for_name
    # display greeting
cat <<EOF

####################
  Hello $user_name,
  nice to meet you!
####################

EOF
else
    # at least 1 arg, let's check it
    case $1 in
        "-h" | "--help")     # the only valid arg
        usage
        ;;
        *)                   # anything else is not valid
        echo "Invalid option"
    esac
fi

exit 0
```

Hinweis: Ein gesittetes Programm oder Shellskript gibt bei Aufruf mit den Optionen `-h` oder `--help` einen *Hilfetext* aus.

Es gibt (zumindest in der GNU-Welt) *Kurzoptionen*, die aus *einem Strich* und *einem Buchstaben* bestehen, und *Langoptionen*, die aus *zwei Strichen* und *einem ganzen Wort* bestehen. Die Kurzoptionen können zusammengefasst werden: Z.B. sind die Befehle `tar -x -v -f ~/archiv.tar` und `tar -xvf ~/archiv.tar` identisch. Ein ganz besonders gesittetes Programm reagiert auf die Option `--version` mit der Ausgabe von Versionsinformationen.

Aufgabe:

- Was tut das oben angegebene Shellskript?
- Wie bekommen Sie heraus, welche Version des C-Compilers `gcc` auf Ihrer virtuellen Maschine installiert ist?

In der folgenden Teilaufgabe wird ein Shellskript erstellt. Da das Script in Aufgabe 3 wiederverwendet wird, ist die Beschreibung leider restriktiver als üblich.

Aufgabe: Schreiben Sie ein Shellskript `run_all` mit folgender Funktion

- Dieses Skript durchläuft vier Parameterlisten, die in den vier SHELL Variablen `seed_value_list`,`page_size_list`, `page_rep_algo_list` und `search_algo_list` gespeichert sind.
- Diese Variablen haben folgende Werte:
  - `seed_value_list="2806 225 353"`
  - `page_size_list="8 16 32 64"`
  - `page_rep_algo_list="FIFO CLOCK AGING"`
  - `search_algo_list="quicksort bubblesort"`
- Die Ergebnisse werden in Dateien im Unterverzeichnis `results` abgelegt. Beim Start des Skripts werden alte Ergebnisse aus diesem Unterverzeichnis gelöscht.
- Mit dem aktuellen Parameter aus der Liste `page_size_list` werden die beiden Programme `p1` und `p2` aus der C Datei `mockup.c` erzeugt.
  Bei der Compilation von `p1` und `p2` wird der aktuelle Wert aus der Liste `page_size_list` als Wert der `#define` Variablen `VMEM_PAGESIZE` übergeben. Bei der Compilation von `p1` wird des weiteren die `#define` Variable `STOP_BEFORE_END` gesetzt.
- Für jede Kombination der drei Parameter aus den Listen `seed_value_list`, `page_rep_algo_list` und `search_algo_list` werden die beiden Programme `p1` und `p2` wie folgt zusammen ausgeführt.
  - Die Parameterkombination des aktuellen Durchlaufs wird ausgegeben.
  - Zuerst wird Programm `p1` im Hintergrund mit folgendem Parameter gestartet `p1 -<akt. Param. aus page_repo_algo_list>`
  - Anschließend wird die Ausführung des Skripts mit `sleep` für 1 Sekunde unterbrochen. Hier wird unsynchronisiert gewartet, bis `p1` die Initialisierung durchgeführt hat.

- Anschließend wird Programm `p2` im Vordergrund wie folgt gestartet `p2 -seed=<aktuelle Parameter aus seed_value_list> -<aktuelle Parameter aus search_algo_list>`.
  Desweiteren wird die Ausgabe von `p2` in die Datei `output_<akt. Param. aus seed_value_list>_<akt. Param. aus seach_algo_list>_<akt. Param. aus page_rep_algo_list>_<akt. Param. aus page_size_list>.txt` im Unterverzeichnis `results` umgelenkt.
- Wenn `p2` fertig abgearbeitet ist, wird Programm `p1` mit `kill -s KILL <process id von p1>` terminiert.

Hinweis: Zur Beendung von `p1` schickt der Befehl `kill` ein Signal an `p1`. `kill` wartet nicht, bis `p1` terminiert ist – das Shellskript wird weiter ausgeführt.
Wenn `p1` schließlich terminiert, erzeugt die Shell eine entsprechende Ausgabe auf `stderr`. Wenn diese unterbunden werden soll, muss mit dem Befehl `wait` auf die Termination von `p1` gewartet werden und die Ausgabe auf `stderr` verworfen werden, indem sie nach `/dev/null` umgeleitet wird.
Bitte setzen Sie dies um.

3 make

Unter Emil finden Sie ein `tar` File mit dem Quellcode für drei kleine C Programme. Die drei Programme greifen auf das Module `print` zu. Der Quellcode liegt im Unterverzeichnis `src`.

Aufgabe:

Erstellen Sie ein `makefile`, das die drei Programme wie folgt übersetzt, bindet und installiert.

- Für jedes C Modul wird ein Objekt Modul im Verzeichnis `obj` erzeugt.
- Die Abhängigkeiten im Quellcode werden analysiert, in entsprechenden `*.d` Dateien im Verzeichnis `obj` gespeichert und in das `makefile` inkludiert.
- Über das Target `all` werden die drei Programme gebunden und im Verzeichnis `test` gespeichert. Dort können Sie getestet werden.
- Über das Target `install` werden die Programme installiert, indem Sie aus dem Verzeichnis `test` in das Verzeichnis `bin` kopiert werden.
- Über das Target `clean` werden alle temporären Dateien, jedoch nicht die installierten Programme aus dem Verzeichnis `bin`, gelöscht.

Hinweise:

- `http://www.ijon.de/comp/tutorials/makefile.html` gibt eine kleine Einführung in `make`.
- `https://www.gnu.org/software/make/manual/html_node/index.html` eignet sich als Nachschlagewerk für `make`.
- In einem C-Programm muss die Funktion `main` genau einmal existieren. Beachten Sie dies beim Binden der C Programme.

Viel Erfolg und viel Spaß!