1 Der Umgang mit Linux

1. 1 Die Macht der Kommandozeile

Ich werde im Folgenden die meisten Dinge mit Befehlen der *Kommandozeile* beschreiben. Häufig gibt es alternativ auch grafische Werkzeuge, die dasselbe tun, aber in diesem Skript ist die Beschreibung und Dokumentation *viel* einfacher mit den Kommandozeilenbefehlen, frei nach dem Motto: *Ein Wort sagt mehr als tausend Bilder*.

1. 1.1 Hilfe!

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

Hier noch ein paar Links zu Linux *cheat sheets*:
`https://www.cheatography.com/davechild/cheat-sheets/linux-command-line/`
`https://www.linuxtrainingacademy.com/linux-commands-cheat-sheet/`
`https://learncodethehardway.org/unix/bash_cheat_sheet.pdf`
`https://files.fosswire.com/wpu/2007/08/fwunixref.pdf`
`https://www.pc-erfahrung.de/linux/linux-befehle.html`

1. 1.2 `script`: Aktionen aufzeichnen

1. 1.3 Erzeuge Arbeitsverzeichnis

```
mkdir ~/Bs_Prakt
```

Hinweis: Das Zeichen ~ steht für das eigene Heimatverzeichnis. In diesem Beispiel hätte ich auch tippen können:
`mkdir /home/bs/Bs_Prakt`

Beachte: Unix-Filesysteme unterscheiden Groß- und Kleinschreibung

Aufgabe:

- Zeichnen Sie alles auf, was Sie ab jetzt tun.
- Was ist *Tab-Expansion*, und was nützt Ihnen das bei der Arbeit mit der Kommandozeile?
- Was erhalten Sie beim Drücken der Tastenkombination <Alt><.>?

  Hinweis: Die Antwort „Der letzte Befehl wird ausgegeben“ ist *falsch*!

1. 1.4 `cd`: Verzeichnis wechseln

```
cd ~/Bs_Prakt
```

1. 1.5 `pwd`: Wo bin ich?

```
pwd
```

```
/home/bs/Bs_Prakt
```

1. 1.6 `whoami`: Wer bin ich?

```
whoami
```

```
bs
```

1. 1.7 `ls`: Verzeichnisinhalt anzeigen

Tonnen von Optionen!

```
ls /etc
```

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. 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

1. 1.9 `less` / `cat`: Textdateien anzeigen

```
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.
```

Das ist OK für kürzere Dateien (Hinweis: Blättern mit <Shift><PageUp> und <Shift><PageDown>)
Für längere Dateien nimmt man den Pager `less` (Merkwürdiger Unix-Hacker Humor: der

karge Unix-Standard-Pager heißt *more* – und: *less is more*!)

1. 1.10 E/A-Umleitung, Pipes

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

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

Wenn ich die Ausgabe an eine Datei *anhängen* will, verwende ich 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, in Abschnitt 1. 2 finden Sie diese Technik wieder.

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. 1.11 `cp`: Kopieren, `mv`: Verschieben, `ln`: Verlinken, `rm`: Löschen

Wir erzeugen eine kleine Textdatei:

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

Mit dem Befehl `cp` erzeugen wir Kopien:

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

Mit dem Befehl `mv` verschieben wir Dateien bzw. benennen sie um:

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

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

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

Mit dem Befehl `rm` löschen Sie eine Datei

```
rm text01.txt
```

Aufgabe:

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

1. 1.12 Shell-Sonderzeichen

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

Zeichen	**Bedeutung**	Beispiel
`*`	beliebiger Text (außer „\“)	`ls ~/Vorlesg/*/Skript`
`?`	Ein beliebiges Zeichen (außer „\“)	`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. 1.13 `grep`: Nach Text suchen

Ein wirklich mächtiges Werkzeug in allen Lebenslagen ist *grep* 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. 1.14 `ps`, `pstree`: Laufende Prozesse anzeigen

`ps`

```
...
pstree
...
```

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

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

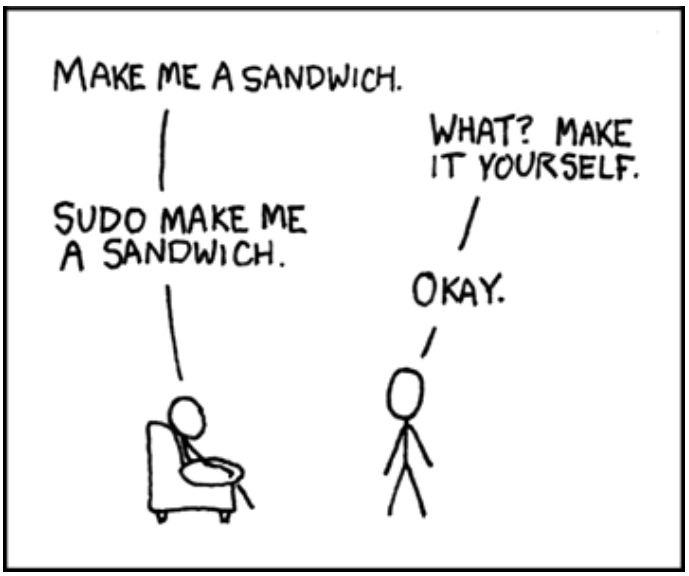
1. 1.15 `(h)top`: Interaktive Prozessanzeige

Mit `top` oder (mein Favorit) `htop` können Sie interaktiv die Prozessliste durchstöbern. `info htop` erzählt Ihnen mehr.

Hinweis: Das Programm htop müssen Sie erst installieren:

```
sudo zypper install htop
```

Der Befehl `sudo` verschafft Ihnen die dazu nötigen Administrator-Rechte



1. 1.16 Ausführen eigener Programme

Dazu müssen wir erst mal ein eigenes Programm haben. Hier die minimalistische Version:

```
fohl@FOHL-PC:tmp$ cat << EOF > hello.c
> #include<stdio.h>
> int main (void)
> {
>   printf("Hallo BS-Praktikum!\n");
>
>   return (0);
> }
> EOF
```

Wir prüfen, ob die Programmdatei richtig erzeugt wurde:

```
fohl@FOHL-PC:tmp$ cat hello.c
#include<stdio.h>
int main (void)
{
  printf("Hallo BS-Praktikum!\n");

  return (0);
}
```

Nun kompilieren wir das Programm `hello`:

```
fohl@FOHL-PC:tmp$ make hello
cc     hello.c   -o hello
fohl@FOHL-PC:tmp$ ls -l hell*
-rwxr-xr-x 1 fohl fohl 6598 29. Sep 15:27 hello
-rw-r--r-- 1 fohl fohl   88 29. Sep 15:26 hello.c
fohl@FOHL-PC:tmp$ ./hello
Hallo BS-Praktikum!
```

Das Listing zeigte uns durch die Buchstaben `x`, dass `hello` ausführbar ist.

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

1. 1.17 Diagnosetools – `ldd`: Welche Bibliotheken benötigt ein Programm?

```
fohl@FOHL-PC:tmp$ ldd hello
        linux-gate.so.1 =>  (0xb8085000)
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7f03000)
        /lib/ld-linux.so.2 (0xb8086000)
```

1. 1.18 Diagnosetools – `strace`: Systemaufrufe protokollieren

Das Zauberwerkzeug zum Lokalisieren von Problemen:

```
fohl@FOHL-PC:tmp$ strace ./hello
execve("./hello", ["./hello"], [/* 54 vars */]) = 0
brk(0)                                  = 0x83b4000

... Jede Menge Text ...

write(1, "Hallo BS-Praktikum!\n"..., 20Hallo BS-Praktikum!
) = 20
exit_group(0)                           = ?
```

Doll, was unser bescheidenes Programm so alles an Systemaufrufen macht!

1. 1.19 Diagnosetools – journalctl

Mitunter ist es interessant, die aktuellen Meldungen des Kernels oder anderer Systemprogramme anzusehen. Dazu gibt es den Befehl
`sudo journalctl -xe`.

Die Option `-e` bewirkt, dass die Anzeige ans *Ende* des Journals springt, die Option `-x` führt zu etwas ausführlicheren Erläuterungen zu den Meldungen.

1. 1.20 Exkurs: Datenaustausch mit dem Hostrechner

Wahrscheinlich möchten Sie die Resultate Ihrer Arbeit irgendwo außerhalb der virtuellen Maschine speichern. Dazu müssen Sie einen *shared folder* anlegen:

1. Fahren Sie die virtuelle Maschine herunter und erzeugen Sie ein Verzeichnis *auf dem Hostrechner*, das vom virtuellen Rechner aus erreichbar sein soll, z.B.
   ```
   mkdir ~/vm_share
   ```
2. Richten Sie mit dem Befehl
   ```
   VBoxManage sharedfolder add "VM name"
         --name "<sharename>" --hostpath "~/vm_share"
   ```
   ein Share-Objekt ein.

   Angenommen, Ihre virtuelle Maschine heißt (wie auf meinem Entwicklungsrechner) *BSLab*, und der gemeinsame Ordner soll auf dem virtuellen Rechner als `bs_share` eingebunden werden, dann lautet der Befehl:
   ```
   VBoxManage sharedfolder add "BSLab"
         --name "vm_share" --hostpath "$HOME/vm_share"
   ```
3. Starten Sie nun die virtuelle Maschine und öffnen Sie eine Konsole. Sie müssen das gemeinsame Verzeichnis in das Dateisystem *einhängen*. Das geschieht mit dem Befehl `mount`. Die eingehängten Verzeichnisse bringt man traditionell unterhalb von `/mnt` unter, (oder unterhalb von `/media`, wenn es sich um Speichermedien handelt):
   ```
   sudo mkdir /mnt/vm_share
   sudo mount -t vboxsf vm_share /mnt/vm_share
   ```

   Hinweis: Es ist nicht nötig, dass der Share-Name, der Name des Hostverzeichnisses, und der Name des Verzeichnisses auf der virtuellen Maschine gleich sind, aber es erleichtert den Durchblick.

   Hinweis: Sie ahnen schon: es gibt auch einen Menüeintrag im Fenster der virtuellen Maschine, mit dem Sie ebenfalls ein Share-Objekt erzeugen können.

1. 2 Vorgänge automatisieren: Shellskripte

Hier ist das Gerüst eines Shellskripts:

```
#!/bin/bash
# A shell script with in- and output
# <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:
`tar -x -z -v -f ~/archiv.tgz` ist dasselbe wie `tar -xzvf ~/archiv.tgz`

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?
- Schreiben Sie ein Shellskript `typesort.sh`, das die als Argument übergebenen Dateien nach dem *Dateityp* sortiert, so wie er von dem Befehl `file` ausgegeben wird.
  - Sehen Sie sich dazu die Ausgabe des file-Befehls genau an.
  - Sortieren wollen Sie nach dem Teil der Ausgabe, der nach dem Doppelpunkt und ggf. folgenden Leerzeichen folgt.
  - Schauen Sie sich die Dokumentation von `sort` an:
  - Wie steuern Sie, nach welchem Teil der Zeile sortiert wird?
  - Wie sorgen Sie dafür, das führende Leerzeichen nicht bei der Sortierung berücksichtigt werden?
  - Wie kehren Sie die Sortierreihenfolge um?

  Mit der Option `-h` oder `--help` soll es diesen Hilfetext ausgeben, und damit gleichzeitig erklären, was genau Sie implementieren sollen.

```
  $thiscommand  [ OPTIONS ] FILE [ FILE . . . ]
            Sort filenames by file type as given by
            the ''file'' command
  $thiscommand --version       Print Version number
OPTIONS :
-t              sort text files only
-n              sort non-text files only
-r              reverse sort order
-v --verbose    print debugging messages
```

  Die Aufgabe im Einzelnen:

  - Die Grundidee ist, die Ausgabe von `file` von `grep` filtern zu lassen, und dann mit `sort` zu sortieren.

- Um die Optionsflags eins nach dem anderen abzuarbeiten, verwenden Sie eine while-Schleife, an deren Ende Sie mit dem Befehl **`shift`** die Argumente um eine Position aufrücken lassen. Dadurch müssen Sie immer nur das erste Argument `$1` bearbeiten.

Hinweis:

- Denken Sie daran, dass bei Shellskripten UNIX-Zeilenenden zwingend erforderlich sind (ggf. mit `fromdos` umformatieren)
- Die eckigen Klammern in den Bedingungen von **`if`** und **`while`** müssen innen ein Leerzeichen haben.
- Wenn Sie sich gar nicht mit der Bash anfreunden wollen, dürfen Sie die Aufgabe auch in Perl lösen.
- Die Arbeit Ihres Sortierprogramms demonstrieren Sie am Inhalt des Archivs `Bsp1files.zip`, das in meinem Pub-Verzeichnis steht. Entpacken Sie das Archiv irgendwohin und führen dann Ihr Programm auf den Daten des Verzeichnisses aus.

Viel Erfolg und viel Spaß!