Prof. Dr. M. Jäger
FB MNI

23. Oktober 2015

Übung 2 - Betriebssysteme

Hinweise

- Für die praktischen Aufgaben benötigen Sie einen Rechner mit einer POSIX-Shell und einer POSIX-konformen C-Programmierumgebung, z.B. eine UNIX-/Linux- oder Mac OSX-Plattform. Für eine Windows-Plattform empfiehlt sich die Installation der Cygwin-Umgebung (https://www.cygwin.com). Alternativ können Sie sich mit einem ssh-Client-Programm auf einem THM-Rechner anmelden und dort arbeiten, z.B. auf *saturn.mni.thm.de*.
- Die Dokumentation aller UNIX-Kommandos und aller Systemaufrufe ist im System-Manual verfügbar, das z.B. mit dem Kommando *man* gelesen werden kann. Die Dokumentation der eingebauten BASH-Befehle kann mit dem BASH-Kommando *help* gelesen werden.
- Machen Sie sich mit einem Texteditor ihrer Wahl vertraut, z.B. *nano* oder *gedit*. Für Fortgeschrittene und Mutige ist *emacs* geeignet. Emacs ist zwar komplex, aber dafür Plattform-unabhängig, bietet umfangreiche Funktionen und ist durch LISP-Programme erweiterbar.

Aufgabe 1 (Shell-Grundlagen und einfache Kommandos)

Als *Shell* bezeichnet man einen Kommandointerpretierer. Wir verwenden *bash* („Bourne Again SHell“). Eine Shell kann einfache Kommandos, aber auch komplexe Programme ausführen. Dabei spielt es keine Rolle, ob die Kommandos interaktiv eingegeben werden oder in einer Datei („Shellscript“) gespeichert sind.

a) Lesen Sie ein Tutorial zur BASH, z.B. http://m-jaeger.de/lv/bs1/skripten/bash/bash.pdf

b) Machen Sie sich mit dem System-Manual vertraut:

- Lesen Sie den Eintrag zum Befehl *pwd* komplett.
- Lesen Sie, was in der Beschreibung zum Befehl *man* über die Einteilung des System-Manuals in verschiedene Sektionen steht. Wie wählt man eine Sektion?
- Lesen Sie die Kurzbeschreibung zum Befehl *apropos* und probieren Sie den Befehl aus.

c) Lesen Sie im System-Manual die grundlegende Verwendung (ohne Details) folgender Befehle: *echo*, *pwd*, *ls*, *ps*, *mkdir*, *rmdir*, *cat*, *rm*, *cp*, *mv*, *head*, *tail*, *grep*

d) Machen Sie sich mit der Shell durch interaktive Ausführung der nachfolgenden Befehle vertraut

```
cd                      # gehe in das Heimatverzeichnis
V=bs/aufgaben/a1        # Pfad des neuen Verzeichnisses in Variable V merken
mkdir -p $V             # erzeuge neues Verzeichnis
cd $V                   # gehe in das neue Verzeichnis
touch d                 # erzeuge leere Datei "d"
ls -l d                 # Dateiattribute von "d" anzeigen
ls -l                   # Dateiattribute aller Dateien im Verzeichnis anzeigen
```

e) Welches Ausgaben erwarten Sie am Bildschirm in der oben beschriebenen Situation bei den unten angegebenen Kommandos? Beachten Sie dabei: „ls -l d e“ soll alle Attribute der dateien „d“ und „e“ anzeigen, eine Datei „e“ existiert aber nicht!

Hinweis: Die spezielle Datei „/dev/null“ ist bei der Ausgabe eine Art „schwarzes Loch“, in der alle Ausgabedaten verschwinden. Beim Lesen erhält man sofort „End-Of-File“.

```
ls -l e
ls -l d e
ls -l d e > /dev/null
ls -l d e 2> /dev/null
ls -l d e > /dev/null  2> /dev/null
ls -l d e > /dev/null 2>&1
```

Aufgabe 2 (Statuscode)

Ein Kommando liefert einen ganzzahligen Statuscode zurück: 0 heißt immer „OK“, andere Werte stehen für bestimmte Fehler. Den Statuscode des letzten Kommandos zeigt man an mit

```
echo $?
```

Betrachten Sie noch einmal das Beispiel von oben: Welche Statucodes liefern die nachfolgenden Kommandos?

```
ls -l e; echo $?
ls -l d; echo $?
ls -l d e; echo $?
```

Aufgabe 3 (Filter und Pipelines)

Ein Filter ist ein Programm, das einen Strom von Eingabedaten transformiert und die Ergebnisse der Transformation als Ausgabestrom weiterleitet. Filter kann man in einer Pipeline nach dem Fließbandprinzip hintereinander anordnen.

a) Verwenden Sie folgende Filter einzeln interaktiv und beobachten Sie die Wirkung.

```
cat
wc
tr e x
tr -d e
grep [0-9]
grep ^[0-9]
awk '{print $2}'
```

Eine geeignete Testeingabe ist beispielsweise:

```
1 Hallo Welt
2 Ende
Gleichung: 12 + 24 = 36
```

Bei interaktiver Eingabe erzeugen Sie mit der Taste Strg-D das Eingabeende.

b) Betrachten Sie die nachfolgenden Pipelines. Probieren Sie die Befehle einzeln aus. Welche Funktion haben die Pipelines?

```
ls -l | grep ^d
ls -l | grep ^d | wc -l'
ls -l | grep ^d | awk '{print $9, $1}'
echo 4 + 3 | bc
echo "sqrt(2)" | bc -l
```

Aufgabe 4 (Kommandosubstitution)

a) Testen Sie Kommandosubstitution mit folgenden Beispielen:

```
$ date "+%Y-%m-%d"
$ DATE=‘date "+%Y-%m-%d"‘;  echo $DATE
$ DATE=$(date "+%Y-%m-%d"); echo $DATE
$ date
$ TAG=$(date|awk '{print $1}'); echo $TAG
```

b) Wie kann man an die Variable WURZEL2 die Quadratwurzel von 2 zuweisen?

Aufgabe 5 (Einfaches Shellscripting)

a) Ein neues Kommando *anzahlsubdirs* soll die Anzahl der Unterverzeichnisse eines beliebigen Verzeichnisses ausgeben. Die folgende Alias-Definition zählt nur die Unterverzeichnisse des aktuellen Verzeichnisses. Erweitern Sie die Definition zu einer Funktion, die das Verzeichnis als Parameter erhält. Testen Sie die Funktion interaktiv und als Shellscript.

```
alias anzahlsubdirs="ls -l | grep ^d | wc -l"
```

b) Schreiben Sie ein Shellscript „showargs“, das seine Argumente wie im folgenden Beispiel ausgibt:

```
$ showargs hallo Welt
"hallo"
"Welt"
```

c) Erweitern Sie das Shellscript so, dass es die Argumentnummern mit ausgibt:

```
$ showargs hallo Welt
1. "hallo"
2. "Welt"
```

Hinweis: Die Shell führt normalerweise keine Berechnungen mit Variablen durch. Beispiel:

```
$ i=1; i=$i+1; echo $i
1+1
```

Man kann hier Hilfsprogramme wie `expr` oder `bc` und Pipelines verwenden. Einfacher geht es mit dem BASH-internen Kommando `let` (s. „help let“): `let i=1; let i++`