Aufgabenblatt 1

Shell / Shell-Skripte / C-Programm

1 Erste Erfahrungen mit der Bash-Shell (Befehlsinterpreter)

1.1 Befehle ausprobieren

Machen Sie sich mit der **bash-Shell** vertraut, indem Sie in einem Linux-System die **fett** gedruckten Befehle in einem „Terminal"-Fenster ausprobieren. Protokollieren Sie alle Ihre Tests und halten Sie die Protokolldatei zur Abgabe bereit.

Syntax in der folgenden Befehlsbeschreibung:

- Großbuchstaben + Unterstreichung: Platzhalter für einen beliebigen String (Beispiel: NAME)
- Eckige Klammern: In den Klammern steht eine optionale Ergänzung, die weggelassen werden kann (Beispiel: [-r])

strg-C	**Laufendes Programm abbrechen**
alias NAME=VALUE	Zeichenersetzung („Alias") definieren (ohne Leerzeichen!)
bg	Programm im Hintergrund laufen lassen (ohne Benutzereingaben)
cat FILE	**Textdatei** FILE **auf Standardausgabe (stdout) ausgeben**
cd DIR	**Verzeichnis wechseln**
chmod [ugoa][+/-][rwx] FILE	**Zugriffsrechte bzgl. Datei FILE ändern (x: ausführbar)**
cp [-i] FILE1 FILE2	**Kopiere Datei** FILE1 **nach** FILE2
date	**Datum und Zeit anzeigen**
df	Information über Dateisysteme anzeigen
diff FILE1 FILE2	Unterschiede zwischen Datei FILE1 und Datei FILE2 anzeigen
echo STRING	**Zeichenkette** STRING **auf Standardausgabe (stdout) ausgeben**
env	**Umgebungsvariablen anzeigen**
exit	**Shell oder Skript beenden**
export VAR	**Shell-Variable** VAR **an alle Kindprozesse vererben**

fg	Programm in den Vordergrund holen
find DIR **-name** FILE **-print**	**Finde eine Datei namens** FILE **beginnend im Verzeichnis** DIR
grep [-r] STRING FILE	**Suche in der Datei** FILE **nach der Zeichnkette** STRING
jobs	Information über Hintergrund-Programme der aktuellen Shell
kill [-9] PID	Prozess (laufendes Programm) mit der Prozessnr. PID beenden
ln [-s] FILE1 FILE2	[symbolischen] Link FILE1 –> FILE2 erzeugen
lpq	Drucke Warteschlangen-Status
lpr [-P QUEUE] FILE	Drucke FILE auf Drucker-Queue queue
ls [-l] [FILE**]**	**Aktuellen Verzeichnis-Inhalt als Liste von Dateinamen ausgeben**
man PROG	**Beschreibung des Programms** PROG
mkdir DIR	**Verzeichnis erzeugen**
more FILE	**Textdatei** FILE **seitenweise anzeigen**
mv QUELLE ZIEL	**Datei** QUELLE **umbenennen bzw. verschieben**
passwd	Ändert das Passwort des aktuellen Benutzers
PROG	**Ausführbares Programm PROG starten (wird in den in $PATH angegebenen Verzeichnissen gesucht)**
PROG&	Programm PROG direkt als Hintergrundprozess starten (ohne Benutzereingaben)
ps [-efa]	**Prozess-Informationen anzeigen**
pstree	**Prozess-Informationen als Baumstruktur anzeigen (Eltern/Kinder)**
pwd	Name des aktuellen Verzeichnisses ausgeben
rm [-i] FILE	Datei FILE löschen
rmdir DIR	Verzeichnis DIR löschen
sleep SEC	Hält die aktuelle Shell-Ausführung um SEC Sekunden an
time PROG	Programm PROG starten und verbrauchte CPU-Zeit ausgeben
touch FILE	Legt eine neue Datei FILE an oder aktualisiert den Zeitstempel
VAR=VALUE	**Shell-Variable** VAR **den Wert** VALUE **zuweisen**

who	Aktuelle Benutzer dieses Systems anzeigen
> FILE	**Standardausgabe (stdout) auf file umlenken, file ggf. neu erzeugen oder überschreiben**
>> FILE	Standardausgabe (stdout) auf file umlenken, file ggf. neu erzeugen oder Ausgaben an file anhängen
cat << EOF	**Erzeugen eines here-Files. (Beenden mit "EOF")**
$VAR	**Die Zeichenkette $VAR durch den aktuellen Wert der Variablen VAR ersetzen**
ls -l | sort -rnk5	**Standardausgabe eines anderen Programms verbinden (siehe auch Info unten)**
$1 $2 $3 …	Zeichenkette $1, $2, $3, .. durch jeweils 1., 2., 3. .. Parameter der Befehlszeile ersetzen ($0: Programmname)
$#	Zeichenkette $# durch Anzahl der Parameter der Befehlszeile ersetzen (Dezimalzahl)
$?	Zeichenkette $? durch return value des zuletzt aufgerufenen Programms (Vordergrundprozesses) ersetzen
$(PROG)	Das Programm PROG starten und die Zeichenkette $(PROG) durch die Ausgaben des Programms ersetzen
.	Zeiger auf das aktuelle Verzeichnis
..	Zeiger auf das direkt übergeordnete Verzeichnis
*	Metazeichen: Platzhalter für beliebig viele Zeichen

Tipp: Informationen zur bash und UNIX/Linux-Befehlen finden Sie auf einem Linux-Rechner (Befehl „man"), in Büchern (Bibliothek) und im Internet (z.B. https://de.wikibooks.org/wiki/Linux-Praxisbuch/_Shellprogrammierung). Unten finden Sie außerdem Tipps, wie Sie auf ein Linux-System außerhalb des Labors zugreifen können.

Info: Zur Kommunikation zwischen verschiedenen Prozessen können in aktuellen Betriebssystemen sogenannte „Pipes" verwendet werden (siehe Vorlesung zu „Message Passing"). Auf der Kommandozeile (bzw. in der Shell) kann dazu der senkrechte Strich verwendet werden:

1.2 Umgebungsvariablen und Besonderheiten

Beantworten Sie die folgenden Fragen:

a) Was enthalten die folgenden Umgebungsvariablen (Environment Variables)?

$HOME, $PATH, $UID, $USER

b) Was bewirkt der Befehl "cd $HOME" ? Gibt es eine einfachere Alternative?

c) Was für eine Funktion haben die folgenden Eingaben?
↑ ↓ strg-d (in leerer Zeile)

d) Was ist die Funktion der .bashrc Datei im Verzeichnis $HOME?

1.3 Shell-Skripte

Die bash-Shell ist in der Lage, mehrere Befehle aus einer Textdatei („Shell-Skript“) zu lesen und auszuführen. In der Dokumentation finden Sie dazu auch die Beschreibung einfacher Kontrollstrukturen (if, for, ...) sowie von Funktionsaufrufen. Ein Beispiel-Skript `example.sh` finden Sie in moodle.

Erstellen Sie folgende Shell-Skripte (bash) und testen Sie diese. Für das Protokoll reicht der **kommentierte** Source Code des Skripts. Während der Abnahme sollten Sie den Code erklären können.

a) `frename.sh <string>`

Hängt für alle Dateien im aktuellen Verzeichnis die Zeichenkette *string* an den aktuellen Dateinamen an (Umbenennung).

b) `try_host.sh [-h|-s <sec>] <hostname>|<IP-Address>`

Der in der Befehlszeile angegebene Rechner (Hostname oder IP-Adresse) soll auf Erreichbarkeit hin überwacht werden. Dazu sendet das Skript in regelmäßigen Zeitabständen ein "ping" an den angegebenen Rechner (nur ein ping Paket) und wertet den return value aus (siehe `man ping`). War der ping Befehl erfolgreich, wird der Rechnername mit einem OK-Vermerk ausgegeben, andernfalls wird er mit einem FAILED-Vermerk ausgegeben.

Das Skript unterstützt folgende Optionen (es darf aber nur eine Option gleichzeitig angegeben werden):

`-h` : Nur Ausgabe der „Usage Message“

`-s <sec>` : Der ping wird zyklisch alle `<sec>` Sekunden ausgeführt.

Fehlt die –s Option, wird der ping alle 10 Sekunden ausgeführt.

Beispiel: Der Aufruf

```
bash try_host.sh -s 5 google.de
```

erzeugt alle 5 Sekunden eine Ausgabe der Art:

`google.de OK`

falls der Host google.de erreichbar ist, anderenfalls

`google.de FAILED`

Haben Sie eine Erklärung dafür, wenn ein Host z.B. von zu Hause aus erreichbar ist, von der HAW aus aber nicht? Versuchen Sie alternativ einmal

`bash try_host.sh -s 1 localhost`

Sie können mittels `nslookup` oder `route` weitere mögliche Gegenstellen herausfinden.

c) Ändern Sie den Status jedes Skripts auf „ausführbar“ und starten sie beide Skripte jeweils als Programm (ohne `bash`-Aufruf, aber mit Angabe des aktuellen Verzeichnisses, z.B. durch Voranstellen von `./`)

d) Erweitern Sie den Inhalt der Umgebungsvariablen PATH so, dass immer das momentan aktuelle Verzeichnis enthalten ist.

2 C-Programm mit Systemaufrufen

Laden Sie das Programm `hello.c` aus moodle herunter. Es wird mit dem Befehl `gcc -o hello hello.c` übersetzt, so dass anschließend das ausführbare Programm `hello` vorhanden ist.

Das zu erstellende C-Programm `mkfile` soll eine Eingabeaufforderung anzeigen, danach maximal 30 Zeichen von der Tastatur lesen (einen Dateinamen), daraufhin eine leere Datei mit diesem Namen und den Zugriffsrechten 0700 (Zugriff nur für Besitzer) erzeugen sowie eine entsprechende Meldung auf dem Bildschirm ausgeben. Falls ein Fehler aufgetreten ist, soll eine allgemeine Fehlermeldung ausgegeben werden.

Bei der Abgabe sollte das Programm **getestet** und **sinnvoll kommentiert** sein. Auch sollen die üblichen Empfehlungen für einen **guten Programmierstil** eingehalten werden (z.B. Definition und Verwendung von Konstanten).

Beispiel (Ausgaben sind *kursiv* dargestellt):

```
$ mkfile
Name der neuen Datei: bsp1
Die Datei bsp1 wurde erfolgreich angelegt!
```

Benutzen Sie zur Realisierung der Systemaufrufe folgende C-Bibliotheksfunktionen (Dokumentation auch über „man“-Befehl erhältlich):

- `fgets` : Liest eine Zeichenkette von stdin ein (Achtung: inkl. Newline!)
- `creat` : Legt eine Datei an und öffnet sie

- `close` : Schließt eine Datei
- `printf` : Erzeugt eine Ausgabe

Hinweise:

- Ggf. müssen entsprechende C-Bibliotheksfunktionen explizit in Ihren Programmcode eingebunden werden (`#include`)
- Bei Deklaration eines char-Arrays (z.B. `char myName[20]`), auch „String“ genannt, wird der Name des Arrays (`myName`) in einem Ausdruck durch die Adresse des ersten Elements ersetzt, hat also den Typ `char*` .
- Die Funktion `int strlen(char* string)` liefert als Rückgabe die Länge eines Strings.
- Ein String wird in C durch den ASCII-Wert 0 (char-Konstante: `'\0'`) nach dem letzten Zeichen abgeschlossen (zusätzlich zum Speicherplatz, der durch das char-Array belegt wird). Daher wird z.B. durch eine Zuweisung `name[5] = '\0'` der String `name` auf die ersten fünf Zeichen verkürzt (0 – 4).
- Beschreibung von `creat`:
  Prototyp: `int creat(char* pathname, int mode);`

  Effekt: Erzeugt eine neue Datei. Falls die Datei bereits existiert, wird ihr Inhalt gelöscht.

  Parameter:

`char* pathname`	Name oder Pfad der neuen Datei.
`int mode`	Bitmuster, das die Zugriffsrechte für die neue Datei festlegt. Die Positionen und Bedeutungen der Bits sind dieselben wie in der Ausgabe des Kommandos `ls -l` (Rechte für Besitzer, Gruppe und andere Benutzer).

  Rückgabe: Dateideskriptor (`int`) für folgende Dateizugriffe oder `-1` bei Fehler.

Tipp: Für den kostenlosen Zugriff auf ein Linux-System gibt es u.a. folgende Möglichkeiten:

- PC-Pool der Informatik nutzen (Linux booten)
- Z.B. Ubuntu (www.ubuntu.com), Open SuSE oder Linux MINT (http://www.linux-mint.com) downloaden, auf bootfähigen USB-Stick kopieren
  Anschließend von CD/DVD oder USB-Stick Linux booten (ohne Installation auf Platte)
- Linux downloaden und auf Partition des eigenen Rechners installieren
  (z.B. http://de.opensuse.org)
- Virtuelle-Maschine-Monitor (Typ2-Hypervisor) unter Windows installieren und Linux in der virtuellen Maschine installieren/starten
  - Z.B. http://www.virtualbox.org/ (SUN / Oracle – OpenSource)
- Zugriff auf Informatik-Server unter Windows mittels putty (ssh/telnet-Client)
  - Putty.exe downloaden http://www.putty.org und starten
  - SSH-Verbindung aufbauen mit ssh.informatik.haw-hamburg.de unter Port 22 (Benutzername und Passwort: HAW-Account)
- (Ubuntu in Windows 10 ist mit Vorsicht zu verwenden: Es gibt Unterschiede zu den oben genannten Möglichkeiten und kann zu Fehlern bei der Aufgabenbearbeitung führen!)