Übungsblatt 1

Übung zur Betriebssystemarchitektur
SS 2005

Dipl.-Inf. Bernhard Rabe
Betriebssysteme & Middleware

Kontrolle der Übungsaufgaben

- Mündliches Testat mit der **gesamten** Gruppe zum Tutoriumstermin
- Quelltexte werden entsprechend den Anweisungen auf dem Übungsblatt an bs@hpi.uni-potsdam.de und dem Betreff AUFGABE=x.x GRUPPE=X geschickt
- Abgabezeiten beachten!

Hinweise zur Email-Einsendung

- Vor den Absenden nochmals Auspacken ausprobieren
- Keine Umlaute (ä, ö, ü, Ä, Ö, Ü,...), Leerzeichen oder Sonderzeichen (ß, @, € ..) in Dateinamen

Tutoriumstermine

- Liste der Zeiten hängt am Aushang Haus C-1
- Termin gilt **nur** für das **1. Testat**
- Weitere Termine in individuelle Absprache mit dem Tutor
- Einschreibung bis **Montag 18 Uhr !**

Compiler

- Windows
  - Visual Studio .NET 2003

  cl.exe

  Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86
- Linux
  - gcc (GCC) 3.3.5

Make

- GNU Make unter Linux
- nmake unter Windows (→vsvars32.bat)
- **Makefile** oder **makefile**
- **$(CC)** Variable für den C-Compiler

Bourne Shell

- Stephen Bourne, 1977/1978 zusammen mit Unix V7
  - Ein-/Ausgabeumlenkung
  - Pipes
  - Hintergrundprozesse
  - Kontrollstrukturen
- Weiterentwicklung **bash** (Bourne again Shell)

Shell Skript

- Ausführbare Textdatei
  - Direkte Ausführung: `/bin/sh Skript`
  - `/bin/sh` Standard Bourne Kompatibel
- Ausführungsumgebung in der 1. Zeile
  - #!<Programm zum Ausführen> +x Bit auf Datei
  - #!/bin/sh
  - Aufruf wie jedes Binärprogramm
- Andere Ausführungsprogramme möglich
  - #!/usr/bin/perl
  - #!/usr/bin/python

Bourne Again Shell

- Bourne kompatibel
- POSIX-konform $(cmd)
- Variablen
  - Zuweisung: `Name=[Wert]`
  - Inhalt: `$Name`
  - Überargumente `$0…$n, n= $# -1`
    - Sichtbarkeit wird in Funktionen überschrieben
    - $0 Name des Skripts oder Shell

Basics

- Kommando ausführen
  - command
- Kommando nebenläufing ausführen
  - command &
- Einausgabeumlenkung
  - < Eingabe
  - > Ausgabe
  - 2>&1 Leitet Stderr auf Stdout

Pipes

- Verbindung der Ein- und Ausgaben von Prozessen
  - cmd1 | cmd2 [| cmd]
  - cmd2 liest von der Standardeingabe die Standardausgabe von cmd1
  - für jedes Kommando wird ein neuer Prozess erzeugt

Listen

- Trennzeichen ; && || & newline
- Sequenz von Kommandos mit newline oder ; getrennt
- command1 && command2
  - command2 wird nur ausgeführt wenn command1 Rückkehrcode 0 lieferte
- command1 || command2
  - command2 wird nur ausgeführt wenn command1 einen Rückkehrcode ungleich 0 lieferte

Variablen

- $? Rückkehrcode des letzten Programmaufrufs
- $# Anzahl der Übergabeparameter
- Parametererweiterung
  - ${name}
  - Notwendig für $n für n>9
    - `${12}` 12. Übergabeargument
  - Und noch vieles mehr: `man bash`

Arrays

- 1-dimensionale Arrays
- name[expr]=value automatische Definition
  - expr muss einer Zahl >=0 entsprechen
- Zugriff ${name[expr]}

Kommandosubstitution

- C-Shell Syntax $(Kommando)
- `Kommando`
  - Führt das Kommando aus und ersetzt es durch die Standardausgabe
  - CURPWD=`pwd` das aktuelle Verzeichnis
- Quiz: Was ist pwd ?

Reservierte Bezeichner

- ! case do done elif else esac fi for function if in select then until while { } time [[ ]]
- Strings "" oder ‘ ‘
  - Ausdrücke in "" werden vorher ersetzt
  - IFS Internal Field Separator, Menge von Trennzeichen für Worte
  - ‘ ‘ keine Ersetzung von Ausdrücken
- # Kommentar bis zum Ende der Zeile

- `$ echo “S $PATH E“`
  `S /bin:/usr/bin... E`
- `echo ‘S $PATH E‘`
  `S $PATH E`

test

- Arithmetischer Vergleich
  - -eq, ne, -lt, -le, -gt, -ge
- String Vergleich
  - ==, !=, <, >
- Dateieingenschaften: liefern true
  - -d Verzeichnis, -f normale Datei, -e Datei existiert, -L Datei ist ein Link
- Schlüsselwort test kann weggelassen werden

Kontrollstrukturen

- for name [in word] ; do list ; done
- führt Kommandos in list für alle Worte in word aus

```
for ff in "$@"; do
  echo $ff
done
```

```
for ff in "$@"
do
  echo $ff
done
```

Kontrollstrukturen II

- if list; then list; [elif list; then list;] ... [else list;] fi
- wenn die Auswertung der if list 0 liefert wird die then Liste ausgeführt, sonst die else list

```
if [ -f /etc/hosts ]; then
   cat /etc/hosts
else
   echo “no /etc/hosts“
fi
```

- wenn `/etc/hosts` existiert so wird diese mit cat ausgegeben sonst der Text ausgegeben

Kontrollstrukturen III

- case word in [[(] pattern [| pattern] ...) list ;;] ... esac
- der ausgewerte Ausdruck word wird mit pattern verglichen und die folgende list ausgeführt

```
case $1 in
   start)
       echo “Start“
   ;;
   stop)
       echo “Stop“
   ;;
      *)
       echo “Unbekannt“
   ;;
esac
```

Kontrollstrukturen IV

- while list; do list; done

```
NUM=$#
while test "${!NUM}"  do
   echo "argv[$NUM] = ${!NUM}"
   NUM=`expr $NUM - 1`
done
```

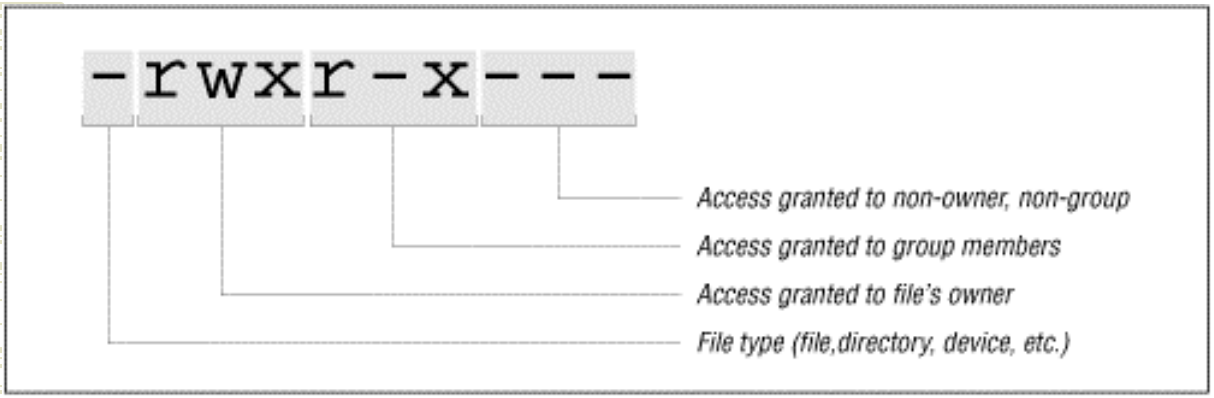
Funktionen

- [function] name () { list; }
- definiert eine Funktion mit dem Namen name welche list ausführt

```
function hello ()
{
    echo “Hello, $1!“
}
hello „World!“
```

(Bourne-Again) Shell Skript

- shellscript.sh
  - Unix Filesystem Rechte
- `-rw-rw-r-x 1 rabe hpistud 162 Oct 18 11:40 shellscript.sh`



- - Plain file, d Directory, c Character device (tty or printer)
  b Block device (usually disk or CD-ROM)
  l Symbolic link (BSD or V.4)

Aufgabe 1.4

- HTML Index
  - , <li> <a >
- Verzeichnisse rekursiv bearbeiten
  - Operationen von test
- pwd vs. /bin/pwd
- Wie erhält man Verzeichnisinhalt ?

Aufgabe 1.5

- Zahlenstatistik von Argumenten
- <ctype.h>
- Was sind Argumente ?
- Was sind Zahlen ?
  - `char '0'-'9'`

Literatur

- Wikipedia.de
- Cameron Newham, Bill Rosenblatt: *Learning the Bash Shell, 2nd Edition* O'Reilly & Associates, Inc., 1998, ISBN 1565923472