Bash Shell

Programming Course

Michel Bisson

Table of Contents

Bash Shell Programming Course

- **Introduction**

bash - **B**ourne **S**hell **A**gain **S**hell (other Shells: `csh, tcsh, zsh`, etc...)
Command allowing a user to change permanently his shell in /etc/passswd:
eg. `chsh -s /bin/ash`
Shell must be in the list of valid shells`/etc/shells` .
Exercise:
- Start konsole and try:
`alias` (Bash's aliases)
`csh`
`alias` (Cshell's aliases)

- **Shell scripts programming purpose**
- Automatization, Flexibility, Exploit the power of each command

- **Requirements and how-to of running a shell script**
- Make the script executable (access rights 755)
- Interpreters name at start of script (`#! /bin/sh` or `#! /bin/bash`)
- Use the **./** before the script file name to run the script located in the path where we are.
- Reading external scripts using **'**. *`/script/path/name`*' .
eg. Use of the `. /etc/rc.status` in SuSE scripts(library of functions)

- **To check the exit status of a the last run script:**
echo $?
Scripts exit status can be controlled by the command `exit` *x* (x = exit status)

- **To Prevent overwriting files via '>' operator:**
`set -o noclobber` or `set -C`
Exception: operator '>|' can override this with for a single file.

- **Meta-characters in bash:**
`$ & ; ( ) { } [ ] * ? ! < >`

- **Debugging shell scripts:**
  Display all the executed lines as it executes them.
  `/bin/sh -xv` *`Scritpname`* [*`ScriptParameters`*]
  or install the program `bashdb` (bash debug) and runtit with the command:
  `bashdb -t` *`debug_terminal_device scriptname`*
  eg. `bashdb -t /dev/pts/3 /home/michel/bin/script1`
  or set the mode (-x) afterwards with the command
  `set -x`

- Set variables display and stops in the script
  `echo $variable ; read dummy`

- **Scripts Headers**

```
#! /bin/bash
# Name:       Scriptname
# Purpose:    Purpose of script....
# Syntax:     scriptname [options] parameter1 parameter2 ...
# Output:     Output of script if any
# Author:     Author Name
# History:    12.05.2003     First inplementation of script
#             24.06.2003     Added option -e for erase
#-------------------------------------------------------------
```

- **GLOBAL, Local, Special Variables and Special Parameters.**
  - GLOBAL(exported from the shells that started it) are often named in capitals.
  eg. `EDITOR, DISPLAY` etc

  - Local (created and valid only in current script) are often named in small letters.
  eg. `program1, content1, result` etc.

  - Special keywords (reserved Environment variables names)
  eg. `CDPATH,HOME,IFS,LANG,MAIL,MAILCHECK,MAILPATH,PATH, PS1,PS2,SHACCT,SHELL,TERM`

  - Special Parameters (reserved short variable names having a specific meaning)
  eg. *`$n, $#, $@, $*, $?, $$, $!, $is`*

- **Suggestions concerning Variables of programs:**
  - Set most of the used programs (incl. paths) in variables
  eg. `iptables="/usr/sbin/iptables"`
  `ifconfig="/sbin/ifconfig"`

  - Test the presence of programs used in script before using them
  eg. `test -x $iptables || exit 5`

  - Loading of external functions.
  eg. (using SuSE) `. /etc/rc.status`

  `dirname` Extracts the path from a filename that includes the path.
  eg. `dirname /usr/local/httpd/htdocs/index.html`
  result: `/usr/local/httpd/htdocs`

  `basename` Extracts the name from a filename that includes the path.
  eg. `basename /usr/local/httpd/htdocs/index.html`
  result: `index.html`

Practical Bash programming exercises:

- **First minimal script : Bash special parameters '$*x:* bashtest'**

(Learning: $*n*, $#, $@, $*, $?, $$, $!, $is, special variables)

```
#!/bin/sh
# Name:         bashtest
# Author:       Michel Bisson
# Syntax:       bashtest param1 param2 param3
# Date:         31 January 2005
# Purpose:      Test script for bash programming course
#-------------------------------------------------------------
#------------ SPECIAL COMBINATIONS WITH '$'-------------------
echo "Values of positional parameters'\$1 \$2 \$3': $1 $2 $3"

echo "The number of positional parameters'(\$#)' is : $#"

# $@=Same as echo $1 $2 $3
echo "All positional parameters'(\$@)' are : $@"

# $*=Same as echo "$1 $2 $3". Understood as one parameter
echo 'All positional parameters(\$*):' $*

ls / &>/dev/null
echo "Exit status'(\$?)' of last command(ls /): $?"

ls /ggg &>/dev/null
echo "Exit status'(\$?)' of last command(ls /ggg): $?"

echo "PID of current shell'(\$\$)': $$"
xterm &

echo "PID of last background command'(\$\!)': $!"
ps
kill $!
ps

#----------------- BASH SPECIAL VARIABLES --------------------
echo "Variable MAILPATH is: $MAILPATH"       (Not exported)
echo "Variable MAIL is: $MAIL"               (Not exported)
echo "Variable MAILCHECK is: $MAILCHECK"     (Not exported)

echo "Variable HOME is: $HOME"
echo "Variable LANG is: $LANG"
echo "Variable SHELL is: $SHELL"
echo "Variable TERM is: $TERM"
echo "Variable PATH is: $PATH"
```

- **Create a `userinfo` script:** Shows oll sorts of information on users (Learning: `for, test,` Variable increment with `for,${var}`)

```
#! /bin/bash
# Name:         userinfo
# Purpose:      Shows info about users from passwd,shadow
#               and smbpasswd
# Syntax:       userinfo user1 user2 user3 .....
# History:      12.09.2003     First inplementation of script
#-------------------------------------------------------------
# Taken from /etc/passwd, /etc/shadow and 'id' command
# for $Benutzer in $@; do
#    echo "Benutzer ist jetz : $Benutzer"
#  done
```

Example 1: `for` using positional parameters

```
#for user in $@; do
for user ; do
     echo "------Info for User: $user ----------------------"
     grep “^$user” /etc/passwd
     grep “^$user” /etc/shadow
     id $user

     echo "------------------------------------------------"
done
```

Example 2: `for` using parameter list

```
for file in $(ls /boot); do
     echo "Extended Status for:"
     stat /boot/$file
     echo "-----------------------------------------"
done
```

Exapmle 3: `for` using incremented variable

```
#!/bin/sh
let lines=$(wc -l /etc/passwd) &>/dev/null
echo $lines
start=5
for ((i=$start; $i<=$lines; i++)); do
        sed -n ${i}p /etc/passwd
done
```

Example 4: (German)

```
#!/bin/bash
# Name:    userinfo
# Zweck:   Information über Bernuter
# Syntax:  userinfo Benutzer1 Benutzer2 Benutzer3 ...
# Datum:   01.02.05
# Lernziel:     'for' Schleife, $#, $@, test, &&, grep, id, if, '!'
#-----------------------------------------------------------------
#if [ $# -ge 1 ]; then
if ! [ $# -ge 1 ]
then
     echo "FEHLER: Mindesten eine Parameter soll gegeben."
     echo "Syntax: userinfo Benutzer1 Benutzer2 Benutzer3 ..."
fi

#------------- Benutzer info Ausdruck -------------
for user in $@
do
     echo "---- Info für Benutzer '$user' --------------------"
     echo "/etc/passwd Zeile"
     grep "^$user:" /etc/passwd
     echo "Info von befehl 'id'"
     id $user
#    if (su -c "grep "^$user" /etc/samba/smbpasswd &>/dev/null")
#    then
#         echo "Benutzer $user ist ein SAMBA Teilnehmer"
#    else
#         echo "Benutzer $user ist kein SAMBA Teilnehmer"
#    fi

done
#------------------ 'for' mit eine Liste von dateien --------
for file in $(ls /boot); do
     echo "-------- Erweitert info von /boot Dateien -----"
     stat /boot/$file
     echo "-----------------------------------------------"
done

#---------------- Inkrementale Integer mit 'for' Schleife

#lines=$(wc -l /etc/passwd) &>/dev/null
let lines=$(wc -l /etc/passwd) &>/dev/null
start=5
#------------ Zeile 5 bis letzte Zeile anzeigen
for ((i=$start; $i<=$lines; i++)); do
     sed -n ${i}p /etc/passwd
done
#------------ Letzte Zeile bis Zeile 5 anzeigen
for ((i=$lines; $i>=$start; i--)); do
     sed -n ${i}p /etc/passwd
done

for ((i=$start; $i<=$lines; i=i+2)); do
     sed -n ${i}p /etc/passwd
done
```

- **Bash Shell Functions Declarations:**

Functions are more useful than alias when complex commands are needed especially when arguments are given to the function.
To declare a function use the following format:

```
fname () { command1;command2;...; }
```

or

```
fname ()
     {
     command1;
     command2;
     ......;
     return 0 (0-255)
     }
```

to call the function, use the following format:

```
fname param1 param2 ...
```

The `param1 param2` will be recognized as `$1 $2` etc inside the function

Important:

- Make there is at least one space or a line break between the { or } and the command.
- *fname* is the name of the function.
- The functions should be saved in `~/.bashrc` or `~/.profile` if wanted permanently.
- Positional parameters passed to the functions are available through the same method (`$1 $2 $3` etc.) as the script's positional parameters. They do not interfere with each other.
- All variables are global to the script including the functions except the positional parameters(`$# $1 $2 $3`) which are local to each individual function.
  Exception: The `$0` stays global.
- The exit status of a function is the exit status of the last command executed in the body of the function.
- The command `return` exits the function and resumes after the function call.
- The function can be exported using the command: `export -f` *Functioname*

- **Function exercise** Graphic Display of a message(using xmessage) as needed.

```
#!/bin/bash
# Name:         flashtime
# Purpose:      Flash for 4 sec a message and then kills it
#               waits 6 sec. then flash the time again......
# Syntax:       flashtime
#-------------------------------------------------------
# Declare the function
flash ()  {
    xmessage -center -fn 10x20 "$1" &
}
# Create an infinite loop
while [ "1" ] ; do
     flash "Es ist: $(date +%H.%M.%S)"
     sleep 6
     kill $!
     sleep 10
done
```

- **System admin Script: `newclient`**

(Learning: Script header, standard system commands)

FULL EXERCISE SCRIPT(english)

```
#! /bin/bash
# Name:         newclient
# Purpose:      Adds a new ftp and possibly a samba client
#               (no login, empty home directory, group ftp)
# Syntax:       newclient /homebase username
# Author:       Author Name
# History:      12.09.2003     First inplementation of script
# Exit codes:    1 = Number of parameters is not 2
#                2 = Username already existing
#                3 = Homebase is not an absolute path
#                4 = Home directory already existing
#-------------------------------------------------------------
#----------- Defining the system dependant variables---
smbpasswd="/etc/samba/smbpasswd"

# Creating the ftp group
groupadd ftp 2>/dev/null
```

- **Checking number of parameters:**

  (Learning: if [...-ne...])

  See Bash Reference Test Integer Operators page 8

```
# Checking for the number of parameters
if [ "$#" -ne 2 ]
then
     echo "False number of parameters:"
     echo "Syntax: newclient /homebase username"
     exit 1
fi
# Declaring variables for better use of positional parameters
homebase=$1
clientname=$2
```

- **Checking of existing user**

  (Learning: `if (command)`and silent checking... `&>/dev/null`)

```
# Checking for existing user
if (grep "^$clientname:" /etc/passwd &>/dev/null)
then
     echo "ERROR: User '$clientname' already exist."
     exit 2
fi
```

- **Checking for homebase directory is an absolute path**

```
# Checking for homebase if it is an absolute path
if ! (echo $homebase | grep "^/" &>/dev/null)
then
     echo "ERROR: Homebase parameter must be an absolute path."
     exit 3
fi
```

- **Checking of existing home directory**

```
# Checking for existing home directory.
if [ -d $homebase/$clientname ]
then
     echo “ERROR: Homebase $homebase/$clientname already exist.”
     exit 4
fi
#if test -d $1/$2 ; then
#    echo "FEHLER: Heimatverzeichnis $1/$2 schon angelegt"
#    exit 3
#fi

#if test -d $1/$2 ; then echo "ERROR: Home directory $1/$2\
#     already exist." ; exit 3 ; fi

#---------- START to create the user ----------
# Creating the empty user home directory
mkdir -p $1/$2

# Creating the new ftp user
useradd -s /bin/false -d $1/$2 -g ftp $2

# Giving the home directory to the new user
chown $2. $1/$2

# Asking 2 times for the password
echo “Enter Password 2 times for $2”
passwd $2
```

- **Asking if the client should be also a Samba client** , add to samba if so. (Learning:[.... `-a` ....], `test, while, read, $?, case, ${var:="n"} [ "$s1" != "ok" ]`)

```
#------------- SAMBA ----------------------------------
#-- Check if smbpasswd samba config file is present and not a runnable file.
if [ -e $smbpasswd -a ! -x $smbpasswd ] ; then
s1=""
    while [ "$s1" != "ok" ] ; do
          # Asking if client is also a Samba client
          echo -n "Should $2 also be a Samba client?(y/n)[n]":
          read samba
          case ${samba:="n"} in
               # ------- The answer is YES('y' or 'Y') ------
               y|Y)
                    echo "Enter Samba password 2 times for $2"
                    smbpasswd -a $2
                    if [ $? -eq 0 ] ; then
                         s1="ok"
                         echo "Samba password successfully\
                              entered"
                    else
                         echo "ERROR: Samba password failed!"
                    fi
               ;;

               # ------- The answer is NO('n' or 'N') ------
```

```
                n|N)
                     s1="ok"
                ;;

                #-------- Not acceptable answer------------
                *)
                     echo "ERROR: Please answer with 'y' or 'n':"
                ;;

          esac
     done
fi
```

Creating a report

(Learning: for, echo -n, id, grep, egrep, cut, print empty line(echo))

```
# Listing all users, their home directories and their INFO
# and if he is a samba client insert '(SAMBA)'

# ---- Get the list of users having UID= 500-999 ----
list=$(egrep ":[5-9][0-9]{2}:" /etc/passwd | cut -d: -f1)
echo "--------- List of normal users from /etc/passwd--------"
echo $list
echo "------------------------------------------------------"

# For each user in list get some passwd info, (samba)*, user ID
for user in $list ; do
     echo -n "INFO for $user: $(grep ^$user: /etc/passwd \
          | cut -d: -f6) :"
     if grep "^$user:" /etc/samba/smbpasswd &>/dev/null
     then
          echo -n "(SAMBA) : "
     fi
     id $user
done
echo ; echo "Client account for $2 is added successfully"
```

FULL EXERCISE SCRIPT(Deutsch)New Version

```
#!/bin/bash
# Name:          newclient
# Zweck:         Neue FTP/Samba Client anlegen
# Syntax:        newclient homepfad benutzer
# Exit Codes:    1 = Falsche Parameter Anzahl
#                2 = Benutzer schon existiert
#                3 = homepfad ist nicht Absolute
#                4 = homeverzeichnis schon existiert
#-----------------------------------------------------------------
#--------------- Farbe definieren ---------------
ROT="\\033[31m"
GRUEN="\\033[32m"
GELB="\\033[33m"
FETT="\\033[1m"
BLAU="\\033[34m"
NORMAL="\\033[m"
#----------- Pfad von Samba Kennwort Datei
smbpasswd="/etc/samba/smbpasswd"
#----------- FTP Gruppe anlegen
groupadd ftp &>/dev/null
#----------- Parameter ersetzen
homepfad=$1 ; benutzer=$2
#----------- Anzahl von Parameter testen. Soll 2 sein.
if [ $# -ne 2 ] ; then
     echo -e "${ROT}FEHLER: Falsche Anzahl von Parameter."
     echo -e "Syntax:  newclient homepfad benutzer.$NORMAL"
     exit 1
fi
#---------- Benutzer Ueberpruefen -----------------
#if (grep "^$benutzer:" /etc/passwd &>/dev/null); then
if (id $benutzer &>/dev/null); then
     echo -e "${ROT}FEHLER: Benutzer '$benutzer' existiert
schon.$NORMAL"
     exit 2
fi
# --------- homepfad als absolute Pfad Ueberpruefen -------------
if ! (echo $homepfad | grep "^/" &>/dev/null)
then
  echo -e "${ROT}FEHLER: homepfad Parameter ist nicht ein absolute
  Pfad.$NORMAL"
  exit 3
fi
# --------- Heimatverzeichnis ueberpruefen -------------------
if test -e $homepfad/$benutzer ; then
     echo -e "${ROT}FEHLER: Heimatverzeichnis '$homepfad/$benutzer'
is present in System.\nEs soll NICHT existieren.$NORMAL"
     exit 4
fi
#--------- START von neue Benutzer Anlegung -------------
echo "Bitte das root Password eingeben"
# --------- Als root HeimatVerzeichnis und Benutzer anlegen
su -c "mkdir -p $homepfad/$benutzer ; useradd -s /bin/false
      -d $homepfad/$benutzer -g ftp $benutzer ; passwd $benutzer;
       chown $benutzer: $homepfad/$benutzer"
if  [ $? -ne 0 ] ; then
```

```
        echo -e "${ROT}FEHLER: Falsche root Password.$NORMAL"
    else
        su -c "mkdir -p $homepfad/$benutzer"
        # Schleife für SAMBA Benutzer
        while [ "$s1" != "ok" ] ; do
            echo -ne "${GRUEN}Soll der Benutzer as Samba Benutzer auch
                      angelegt? (j/n) [n]$NORMAL"
            read antwort
            #echo $antwort
            # Antwort bearbeiten
            case ${antwort:="n"} in
              j|J)
                 echo -e   "${GELB}Geben Sie bitte das 'root'
                           passwort ein Mal."
                 echo -e   "Dann zwei Mal der Samba Passwort von
                           Benutzer.$NORMAL"
                # ------- als root Samba benutzer anlegen ----
                 su -c "smbpasswd -a $benutzer && echo ja >/tmp/tm
                           || echo no >/tmp/tm"
                #------ Falsche root Passwort?
                if  [ $? -ne 0 ] ; then
                     echo -e "${ROT}FEHLER: Falsche root
                               Password.$NORMAL"
                else
                     #------mbpasswd behehl ervolgreich ?
                     if (cat /tmp/tm | grep ja &>/dev/null); then
                          s1="ok"
                          echo -e "${GRUEN}Samba Benutzer
                                   '$benutzer' ist ervolgreich
                                    angelegt.$NORMAL"
                     else
                          echo -e "${ROT}FEHLER: Falsches
                                   Passwort."
                          echo -e "Bitter zwei Mal das selbe
                                   Passwort von Benutzer
                                   eingeben.$NORMAL"
                     fi
                fi
              ;;
              n|N)
                     s1="ok"
              ;;
              *)
                     echo -e "${ROT}FEHLER: Bitte mit 'j' oder 'n'
                               antworten$NORMAL"
              ;;
            esac
        done
    fi
    #-------------- Bericht Ausgabe --------------------
    list=$(grep ":[1-9][0-9][0-9][0-9]:" /etc/passwd | cut -d: -f1)
    for user in $list ; do
       echo "------------------------------------------------------------"
       echo  "INFO fuer $user: $(grep $user: /etc/passwd | cut -d: -f6)"
       id $user
    done
```

FULL EXERCISE SCRIPT(Deutsch)Old Version

```
#! /bin/bash
# Name :   newclient-de
# Zweck:   Eine FTP und möglich SAMBA client anlegen
# Syntax:  newclient-de /homebase username
# Variable:     $0        $1       $2
# Author:  elop klasse
# History: Ver. 0.01.01 29.09.2003  Erzeugung von Script
# Ergebnisse:   exit code:
#                    1 = Falsche anzahl vom Parameter
#                    2 = Benutzer schon angelegt
#                    3 = Heimatverzeichnis schon angelegt
#                    4 = Homebase nich absolute Pfad
#                    5 = Samba is nicht installiert
#-------------------------------------------------------------
#--------- Variable definieren ----------
smbpasswd="/etc/samba/smbpasswd"

# Gruppe für FTP cliente erzeugen
groupadd ftp 2>/dev/null

# -------------- Parameter Anzahl ist 2 ?
if [ "$#" -ne 2 ]
    then
    echo "FEHLER: Falsche Parameter Anzahl"
    echo "Syntax: newclient-de /homebase username"
    exit 1
fi
# -------------- Variable fuer besser lesebarkeit von script----
homebase=$1
clientname=$2

# --------- Ist die Benutzer schon angelgt?
if (grep "^$clientname:" /etc/passwd &>/dev/null)
    then
    echo "FEHLER: Der Benutzer is schon angelgt."
    exit 2
fi

#--------------- Homebase parameter ist Absolute Pfad?
if ! (echo $homebase | grep "^/" &>/dev/null)
     then
     echo "FEHLER: homebase Parameter ist nicht eine Absolute Pfad"
     exit 3
fi

# -------- Ist der Heimatverzeichnis schon angelegt?
if test -d $homebase/$clientname ; then
     echo "FEHLER: Heimatverzeichnis $homebase/$clientname schon angelegt"
     exit 4
fi
#if [ -d $homebase/$clientname ] ; then
#    echo "FEHLER: Heimatverzeichnis $homebase/$clientname schon angelegt"
#    exit 4
#fi
```

```
# --------- Benuzer anlegen-----------------
mkdir -p $homebase/$clientname
useradd -s /bin/false -g ftp -d $homebase/$clientname $clientname
chown $clientname. $homebase/$clientname
echo "Bitte Passwort zwei Mal für $clientname eingeben:"
passwd $clientname

#--------- Client soll fuer Samba angelegt werden?
if [ -e $smbpasswd -a ! -x $smbpasswd ] ; then
  s1=""
  while [ "$s1" != "ok" ] ; do
   echo -n "Wollen Sie diesen Benutzer fuer Samba anlegen?(j/n) [n]:"
   read samba
    case ${samba:="n"} in
      #------- JA Ich will ----------
      j|J) echo "Samba passwort fuer $clientname 2 Mal eintragen:"
        smbpasswd -a $clientname
        if [ "$?" -eq 0 ] ; then
            s1="ok"
         echo "Benutzer $clientname fuer Samba erfolgreich angelegt"
        else
         echo "FEHLER: Sie mussen 2 Mal das Passwort eingeben"
        fi
       ;;
       #--------- NEIN Ich will nicht ---------
       n|N)
           s1="ok"
       ;;
       #---------- Bloedsinn eingetragen-------
       *)
           echo "FEHLER: Bitter mit 'j' oder 'n' antworten."
       ;;
     esac
  done
fi
#----------------- Bericht von script ausgeben --------
echo "Inhalt von /etc/passwd"
# Liste von normale Benutzer in Variable 'list' einfuehlen
list=$(egrep ":[5-9][0-9]{2}:" /etc/passwd|cut -d: -f1)
#-------------- Schleife: schreibt eine Zeile von
#-------------- Benutzer Info pro Gueltige Benutzer
for user in $list ; do
   # Benutzername und Heimatverzeichnis ausgeben
   echo -n "INFO fuer $user: $(grep ^$user: /etc/passwd|\
            cut -d: -f6) : "
   if grep "^$user:" $smbpasswd &>/dev/null ; then
      echo -n "(SAMBA):"
   fi
   # Benutzer Zeile mit Benutzer info von 'id' Befehl erganzen.
   id $user
done
#--------------- Letzte meldung und exit code 0 -------
echo
echo "Klientkonto fuer Benutzer $clientname erfolgreich ausgefuehrt"
exit 0
```

- **Methods of Testing** (exercise in interactive bash)

Brackets `[ .... ]`

```
[ -r /etc/motd ] && echo "All ok" || echo "NOT ok"
[ -r /etc/mot  ] && echo "All ok" || echo "NOT ok"
```

Test command: `test` *`option parameter`*

```
test -r /etc/motd && echo "All ok" || echo "NOT ok"
test -r /etc/mot  && echo "All ok" || echo "NOT ok"

test "aa" = "tt"  && echo "All ok" || echo "NOT ok"
test "aa" = "aa"  && echo "All ok" || echo "NOT ok"
```

Commands

```
ifconfig eth0 &>/dev/null && echo "All ok" || echo "NOT ok"
ifconfig ppp0 &>/dev/null && echo "All ok" || echo "NOT ok"
```

Adding testings: `AND, OR`

```
test "t" = "t" -a "r" = "r"  && echo "All ok" || echo "NOT ok"
[ "a" = "a" -a "b" = "c" ]   && echo "All ok" || echo "NOT ok"
[ "a" = "a" -o "b" = "c" ]   && echo "All ok" || echo "NOT ok"
```

`AND` for commands exit codes in `if` condition

```
if (cat /etc/motd &>/dev/null && cat /etc/fstab &>/dev/null);
```

`OR` for commands exit codes in `if` condition

```
if (cat /etc/motd &>/dev/null || cat /etc/fstab &>/dev/null);
```

- **Exercise 1: Creating a client delete script**

To do:
Script Header
Client system (ftp) account
- Check if client account exist
- Delete it if so.
-

Client Samba account
- Check if client is a Samba client
- Delete it if so.
-

Client home directory
- Check if client home directory has files in it.
- Tells that its content will be shown (press Enter to show content)
- Use `ls -la` *`homedir`* `| less` to show content of home directory
- Ask if it can be erased and erase if yes

- Exercise 2: Service Ports monitor (See file 68_Services_Monitor_Exercise.sxw)

- **Incrementing loops:** sping script: Pings all IPs of a 'C' class subnet
  (Learning:
  Functions, $IFS, [..-a..] [..-o..], echo, &>/dev/null , if [! ..] ,until)

check combinations [*check* **-a** *check*], [*check* **-o** *check*],
egrep of variable (echo *variable* | egrep)
number increment (let "*variable*+=1")
no output on commands results check (*command* &>/dev/null)
negative check (if ! (*command*) ; then)

```
#! /bin/sh
#------------------- Super Ping : sping --------------------------
# File:         sping
# Purpose:      ping a full subnet (netmask 255.255.255.0)
# Date:         07.09.2003
# Author:       Michel Bisson
# Syntax:       sping 192.168.70.  (the last number is omitted!)
#-----------------------------------------------------------------
# Declare error function using the Variable $errortext:
# Displays error message and exits with error code 1
error () {
      echo "$errortext"
      echo "Syntax: eg. $0 192.168.70. "
      exit 1
}
#-----------------------------------------------------------------
# --------Check the validity of the given parameter (Partial IP)
# Only one parameter
if [ "$#" -ne 1 ] ; then
      errortext="ERROR: Incorrect number of parameters."
      error
fi
#-----------------------------------------------------------------
# ----------Parameter is a correct Partial IP? Correct it if possible
# Accept if xxx.yyy.zzz. or xxx.yyy.zzz
# if ! (echo $1 | egrep "^[0-9]*\.[0-9]*\.[0-9]*\.?$" &>/dev/null) ;
if !(echo $1 | egrep "^([0-9]{1,3}\.){2}[0-9]\.?$" &>/dev/null) ;
then
      errortext="ERROR: Bad subnet Partial IP Syntax"
      error
fi
#-----------------------------------------------------------------
# Verify validity if all numbers in IP (0-255)
IFS="."
len=0
for num in $1 ; do
      #let "len+=1"
      let len++
      # Do not accept 192.168.71.XXX (fourth number)
      if [ "$len" -eq 4 -a "$num" != "" ] ; then
            errortext="ERROR: NO Full IP...Just give a partial IP."
            error
      # Do not accept numbers higher than 255
      elif [ "$num" -gt 255 ] ; then
            errortext="ERROR: Wrong values in partial IP Syntax."
```

```
            error
      # Do not accept empty fields eg. 192..168.30
      elif [ "$num" = "" -a "$len" -le 3 ] ; then
            errortext="ERROR: Wrong format of partial network IP."
            error
      fi
done
unset IFS

# Check if subnet IP is valid for local machine in network
if ! (ifconfig | grep "inet " | grep $1 &>/dev/null) ; then
      errortext="ERROR: Partial Subnet IP is not in our local subnet"
      error
fi

#-----------------------------------------------------------------
# Correct by adding a '.' if missing at the end
if (echo $1 | egrep "\.$" &>/dev/null) ; then
      netip="$1"
else
      netip="$1."
fi
#echo $netip

#=================================================================
# Generate all the IPs from xx.xx.xx.1 to xx.xx.xx.254--
netnum=1

#---- ping them all almost at the same time (sent as separate jobs)
until [ $netnum -eq 255  ]; do
    /bin/ping -w3 -c1 $netip$netnum 1>>/tmp/sping &
    let "netnum++"
done

#=================================================================
#----------- wait a bit to let some answers back ----------
sleep 4

#----------- Kill all the ping that are still waiting ---
killall ping &> /dev/null

#----------- Show only the pings that received an answer --
iplist=$(grep '64 bytes from' /tmp/sping | cut -d: -f1 |\
          cut -d" " -f4 | sort -t. -k4n)
for ip in $iplist ; do
      name=$(host $ip | grep pointer | head -n1 | cut -d" " -f 5)
      echo -n "Active Host: $ip"
      if [ $name ] ; then
            echo " ---> $name"
      else
            echo
      fi
done
#-----------------------------------------------------------------
exit 0
```

■ Exercise 2 for the class:

Create a script that displays a message for 4 sec and disappears when a new mail comes or a mail is read from the mailbox:

```
#!/bin/bash
# Name:         newmail
# Purpose:      Flash a message of newmail for 4 sec and go if
#               new mail arrives or if mail has been read
# Syntax:       newmail
#-------------------------------------------------------------
# Declare the display function
flash ()  {
        xmessage -center -fn 9x15 "$1" &
}
#
# Create an endless loop
while [ "1" ] ; do

        # Any mail at all is present?
        if (mail -H &>/dev/null) ; then
                mailnow=$(mail -H | wc -l)

                # New mail has arrived?
                if [ "$mailnow" -gt "$lastmail" ] ; then
                        # How many new mails?
                        newmails=$[$mailnow-$lastmail]
                        # Show the message
                        flash "You have $newmails New Mail"
                        sleep 4
                        kill $!

                # Some Mail is been read?
                elif [ "$mailnow" -lt "$lastmail" ] ; then
                        # How many mail Read?
                        mailread=$[$lastmail-$newmail]
                        flash "You have $mailread less mails"
                        sleep 4
                        kill $!
                fi
                 # Actualize the watcher counter
                 lastmail=$mailnow
        fi
        sleep 1
done
```

● **Text pop-up menus with 'dialog'**
Creating a simple script to set the Hardware and System clock.
(Learning: `read`, `dialog`, `&&`, `date` formats, `hwclock`)

```
#! /bin/bash
# Name:         setwatch
# Purpose:      Change the Hardware and System date and time
#               Just like  'setclock 09/18/2003 21:13:00'
# Syntax:       setwatch
# Author:       Michel Bisson
# History:      12.09.2003      First inplementation of script
#-------------------------------------------------------------
# Ask for today's Day
today_d=$(date +%d)
echo -n "Enter today's Day (01-31) [$today_d]:";read day
# [ "$day" ] &&  day=$today_d
[ "$day" = "" ] &&  day=$today_d

# Ask for today's Month
today_m=$(date +%m)
echo -n "Enter today's Month (01-12) [$today_m]:";read month
[ "$month" = "" ] &&  month=$today_m

# Ask for today's Year
today_y=$(date +%Y)
echo -n "Enter today's Year (2000-2099) [$today_y]:";read year
[ "$year" = "" ] &&  year=$today_y

#echo "Today's date is: $month/$day/$year"
#exit 0

# Ask for today's time via dialog pop-up window
time=$(dialog --stdout --timebox "Hardware/System time setting:\
          \nDate: $month/$day/$year" 0 0)

if [ "$time" ] ; then
     # Set the hardware clock
     #format:   hwclock --set --date="9/22/2002 16:45:05"
     hwclock --set --date "$month/$day/$year $time"

     # Sets the System clock to current hardware clock
     hwclock --hctosys
fi

# Show the new Hardware and System date and time
echo "Present Hardware Clock setting is: $(hwclock)"
echo "Present System   Clock setting is: $(date)"

exit 0
```

- **Creating pop-up Data entry field, menus, yes/no question, and message box.**

(Learning: read, dialog, password entry, shift, eval, $* , $?)
Creating a script (menu1) that will bring a pop-up menu and send the result to a pop-up message.

```
#! /bin/bash
# Name:    menu1
# Purpose: Displays a text menu in terminal
#          - asks for a password
#          - asks for a Presentation text
#          - shows presentation text and choice in a pop-up message
# Syntax:  menu1 MenuTitle choice1 choice2 choice3 .....
# Output:  The text of one of the  chosen
# Author:  Michel Bisson
# History: 12.09.2003 First inplementation of script
#-------------------------------------------------------------
#Preparation:
# echo 'mypasswd' > ~/.scrpasswd
# chmod 600 ~/.scrpasswd
#-------------------------------------------------------------
# Allow minimum 3 parameters (title and 2 choices)
if [ "$#" -lt 3 ] ; then
     echo "ERROR: Wrong number of parameters"
     echo "Syntax: $0 MenuTitle choice1 choice2 choice3 ....."
     exit 1
fi
# Ask for a password (now : mypasswd)
echo -n "Please enter password: "
read passwd

# Not completely tested dialog password box
#passw=$(dialog --stdout --passwordbox \
#    "Please enter password for using this script" 0 0)
#echo $passwd
#exit 0

# Check the entered password against the file ~/scrpasswd
if [ "$passwd" != $(cat ~/scrpasswd) ] ; then
     echo "ERROR: Wrong password."
     exit 1
fi

# Save the menu title before deleting it(shift) from parameter list
title=$1
shift

# Build-up the beginning of the dialog command
command="dialog --stdout --menu $title 0 0 $#"

# Build-up the rest of the dialog command parameters
# Here we generate a tag (item number) in front of each
# choice item in command parameter list
counter=1
for param in $* ; do
     command=$command" "$counter" "$param
```

```
      let "counter++"
done


# Execute the dialog menu command
# The result (item Nr.) is stored in variable '$result'
result=$($command)

# Ask if user want to display the result
# (result is exit code 0 for YES and 1 for NO)
dialog --yesno "Do you want to display the result?" 0 0

# Display the result of the choice if yes selected
if [ "$?" -eq 0 ] ; then
      # Ask for a title for the display of the choice
      mytext=$(dialog --stdout --inputbox \
            "Enter the display title of the result:" 10 60)


      # If the the OK button was presssed then
      # print the choice in a pop-up message
      # otherwise notify of cancellation in pop-up message
      if [ "$result" != "" ] ; then
            eval "choice=""\$""$result"
            dialog --msgbox "$mytext$choice" 0 0
      else
            dialog --msgbox "Choice was cancelled" 0 0
      fi
fi
```

• kdialog

KDialog can be used to show nice graphic dialog boxes from shell scripts
Syntax: `kdialog [Qt-options] [KDE-options] [options] [arg]`

Generic options:

`--help`	Show help about options
`--help-qt`	Show Qt specific options
`--help-kde`	Show KDE specific options
`--help-all`	Show all options
`--author`	Show author information
`-v, --version`	Show version information
`--license`	Show license information
`--`	End of options

Options:

`--yesno <text>` Question message box with yes/no buttons
Result is in exit code: 0=yes 1=no

eg.
```
if { kdialog --yesno "Do you want to proceed?" ; } ; then
```

`--yesnocancel <text>` Question message box with yes/no/cancel buttons
Result is in exit code: 0=yes 1=no 2=cancel

`--warningyesno <text>` Warning message box with yes/no buttons
Result is in exit code: 0=yes 1=no

`--warningcontinuecancel <text>`
Warning message box with continue/cancel buttons.
Result is in exit code: 0=continue 1=cancel

`--warningyesnocancel <text>`
Warning message box with yes/no/cancel buttons
Result is in exit code: 0=yes 1=no 2=cancel

`--sorry <text>` 'Sorry' message box

`--error <text>` 'Error' message box

`--msgbox <text>` Message Box dialog

`--inputbox <text> <init>` Input Box dialog
Output is in STDOUT. Result is in exit code
0=<Ok> 1=<Cancel>(with STDOUT empty)

eg.
```
answer=$(kdialog --inputbox "Enter your name" "Jane Mason")
```

`--password <text>` Password dialog. Replaces typed chars. with '*'
Output is in STDOUT. 0=<Ok> 1=<Cancel>

`--textbox <file> [width] [height]`
Text Box dialog.
eg. `kdialog --textbox /etc/motd`

`--menu <text> [tag item] [tag item]`
Menu dialog. Output is the tag value of the chosen menu text in STDOUT. Result is in exit code.
0=<Ok> 1=<Cancel>
`tag`= 1,2,3..a,b,c ...
`Item`="text to select"

eg. `ans=$(kdialog --menu "Choose item" 1 Item1 2 Item2 3 Item3)`

`--checklist <text> [tag item status] [tag item status] ...`
Check List dialog. Allows to select multiple items. Output is the tag values of the chosen items. Result is in exit code.
0=<Ok> 1=<Cancel>
`tag`= 1,2,3..a,b,c ...
`Item`="text to select"
`Status`=on/off

eg.
```
ans=$(kdialog --checklist "Select desired items" 1 "bold" on \
2 "italic" off)
```

`--radiolist <text> [tag item status]`
Radio List dialog. Selects only one Item form list. Output is the tag value of the chosen item. Result is in exit code.
0=<Ok> 1=<Cancel>
`tag`= 1,2,3..a,b,c ...
`Item`="text to select"
`Status`=on/off

`--title <text>` Dialog title. Display text with Ok Button

`--separate-output` Return list items on separate lines (for checklist option)

`--passivepopup <text> <timeout>` Passive Popup

`--getopenfilename [startDir] [filter]` File dialog to open an existing file

`--getsavefilename [startDir] [filter]` File dialog to save a file

`--getexistingdirectory [startDir]` File dialog to select an existing directory

`--getopenurl [startDir] [filter]` File dialog to open an existing URL

`--getsaveurl [startDir] [filter]` File dialog to save a URL

`--geticon [group] [context]` Icon chooser dialog

`--progressbar <text> [totalsteps]` Progress bar dialog, returns a DCOP reference for communication

`--print-winid` Outputs the `winid` of each dialog

`--embed <winid>` Makes the dialog transient for an X app specified by `winid`

`--dontagain <file:entry>` Config file and option name for saving the "dont-show/ask-again" state

Arguments:

`arg` Arguments - depending on main option

Exercise1:

```
#!/bin/bash
# Name:   menu
# Zweck:  menu demo mit kdialog
# Syntax: menu
# ------------------------------------------------
# Kdialog abfragen
antwort=$(kdialog --menu "Bitte Operation whaehlen" \
                  1 "Home List"\
                  2 "FSTAB Inhalt"
                  3 "Verbindungen")
case $antwort in
     1)
     ls -la /home
     ;;

     2)
     cat /etc/fstab
     ;;

     3)
     netstat -tu
     ;;
esac
```

Exercise2: Delete an FTP user using kdialog

```
#!/bin/bash
# Name:          del_user
# Purpose:       Demonstrate some Functions of kdialog
#                - It makes a list of users of the ftp group in a menu
#                - It shows the content of the user's home directory
#                  and asks if it should be deleted as well
#                - It asks for confirmation of the delete operation
#                - If confirmed then it asks for the root password
#                - if password ok then it executes the operation
#                - It then confirms the operation
# Syntax:        del_ftp_user
#-------------------------------------------------
# Preparation:
#    users to delete must be from the group 'ftp' as primary group
#    create users with the command: useradd -m -g ftp username
#-------------------------------------------------
# Get all the ftp users into a menu
ftpgid=$(grep "^ftp:" /etc/group | cut -d: -f3)
ftpusers=$(grep ":$ftpgid:" /etc/passwd | grep -v "^ftp:"\
 | cut -d: -f1)
# Put the users in a kdialog menu
index=1
for user in $ftpusers ; do
     menu="$menu $index $user"
     let index++
done
#echo $menu

#--------------- call the kdialog menu
answer=$(/opt/kde3/bin/kdialog --menu "Select user to delete" $menu)
#----------- exit if 'cancel' button is clicked
[ $answer ] || exit 1

#echo $answer

#----- Retrieve the user from the menu list. Learning && and break
index=1
for user in $ftpusers ; do
      [ $answer -eq $index ] && break
     let index++
done
#echo $user

#------------ show the content of the home dir of the user
# get the home dir of the user
homedir=$(grep "^$user:" /etc/passwd | cut -d: -f6)
xterm -e sh -c "ls -la $homedir | less" &
#------------ get the PID of the xterm
pid=$!
#------------ ask if home directory should also be deleted
/opt/kde3/bin/kdialog --yesno "The content of the home directory
'$homedir' of user '$user' is \
shown in a terminal.\nShould it also be deleted?"
#echo $?
```

```
#----------- ask for confirmation of the delete action
if [ $? -eq 0 ]; then
     # Sure to delete the user AND directory?
     /opt/kde3/bin/kdialog --yesno "Are you sure that you want to\
     delete the user '$user'\nAND his home directory :$homedir: ?"
     [ $? -eq 0 ] && erase=dir
else
     # Sure to delete only the user but NOT directory?
     /opt/kde3/bin/kdialog --yesno "Are you sure that you want to\
     delete the user '$user'\nbut NOT his home directory '$homedir'?"
     [ $? -eq 0 ] && erase=user
fi
#echo $erase
#exit
#------------ kill the xterm
kill $pid

case $erase in
     dir)
          #------------ get the root password
          passwd=$(/opt/kde3/bin/kdialog --password "Please enter
root password")
          #------------- Erase the user AND his directory
          if ! (echo $passwd | su - -c "userdel -r $user") ; then
             /opt/kde3/bin/kdialog --error "ERROR: root password is\
             incorrect.\nUser '$user' and his directory '$homedir'\
             will NOT be erased!"
          else
             #------------- Annonce that it is done
             /opt/kde3/bin/kdialog --msgbox "User '$user' and his \
              home directory '$homedir' has been erased."
          fi
          ;;
     user)
          #------------ get the root password
          passwd=$(/opt/kde3/bin/kdialog --password "Please enter\
                 root password")
          #------------- Erase the user but NOT his directory
          if ! (echo $passwd | su - -c "userdel $user") ; then
             /opt/kde3/bin/kdialog --error "ERROR: root password is\
               incorrect.\nUser $user will NOT be erased!"
          else
             #------------- Annonce that it is done
             /opt/kde3/bin/kdialog --msgbox "User '$user' has been\
erased."
          fi
          ;;

     *)
          #------------- Annonce that delete has NOT been done
          /opt/kde3/bin/kdialog --msgbox "User '$user' and his home\
directory '$homedir'\nwill NOT be deleted"
        ;;
esac
```

- **Trapping kill events**

(Learning: `trap`, `while true`, `while :`)
Kill events (messages sent to application via the kill program) can be trapped and a program can be called each time the kill event is received. The most common events to trap are:

- `kill 15 PID` or `kill PID`
- `kill 2 PID` (or CTRL-C sent from the terminal where the script runs)

See demo script(`killtrap`) below.

```
#! /bin/bash
# Name:         killtrap
# Purpose:      trap a kill(15) command and
#               display an xmessage to tell to 'fuck-off'
# Syntax:       killtrap

trap "xmessage -center process $$ say fuck off. & " 15
trap "xmessage -center CTRL-C has been pressed...Humm?" 2

#Look infinitely while showing the PID of the script process
# Only a kill -9 will kill him

while true
#while :
do
        sleep 5
        echo "Scrip with PID $$ still runs"
done
```

Extra scripts as examples:

```
dirmod
filemod
translate
uppercase
lowercase
urcp
colfmt
```