Sisteme de Operare

Curs 3-4

Learn VIM or Other command line editor!

## Standard Input/Output/Error and I/O redirections

1. 0 = standard input – where you read from when you use “scanf” or “gets” in C, “cin” in C++, or “input” in Python.
2. 1 = standard output – where you write when you use “printf” or “puts” in C, “cout” in C++, or “print” in Python.
3. 2 = standard error – similar to the standard output, but conventionally used to display errors, in order to avoid mixing results with errors.
4. I/O redirections
   1. What if I want the output of a command to be stored in a file?
      1. ls -l --color=never /etc > output.txt
   2. What if I want to add the output of another command to the same file?
      1. ps -ef >> output.txt
   3. What if I want the standard output of a command to be sent to the standard input of another command?
      1. ls| sort
   4. What if I want the standard input to be taken from a file?
      1. sort < a.txt

## Command Truth Values

1. The truth value of a command execution is determined by its exit code. The rule is the opposite of the C convention, with 0 being true, and anything else being false. Basically, there is only one way a command can be executed successfully, but many ways in which it can fail. The exit code is not the output of the command.
2. There two standard commands true and false, that simply return 0 or 1.
3. Command test offers a lot of options for comparing integers, strings and verifying file and directory attributes
4. Commands can be chained using logical operators && and ||. Lazy logical evaluation can be used to nice effects. The negation operator ! reverses the truth value of a command.
   1. true || echo This should not be displayed
   2. false || echo This should definitely by displayed
   3. true && echo This should also be displayed
   4. false && echo Should never be displayed as well
   5. grep –q “=” /etc/passwd || echo There are no equal signs in file /etc/passwd
   6. test –f /etc/abc || echo File /etc/abc does not exist
   7. test 1 –eq 2 || echo Not equal
   8. test “asdf” == “qwer” || echo Not equal
   9. ! test –z “abc” || echo Empty string
5. Test command conditional operators
   1. String: ==, !=, -n, -z
   2. Integers: -lt, -le, -eq, -ne, -ge, -gt
   3. File system: -f, -d, -r, -w, -x

## Shell Variables and Embedded COmmands

1. Defined as A=”Tom” or B=5
2. Embedded commands
   1. Delimited by ` (back-quote). Are replaced by the output of the command. Store a command output in a variable: N=`grep “/gr211/” /etc/passwd | wc –l`
3. Referred as $A or ${A}
   1. echo $A is a human
   2. echo $Acat is a feline or an application server – doesn’t work
   3. echo ${A}cat is a feline or an application server
4. When used in strings delimited by “, variables and embedded commands will be replaced by their value. Strings delimited by ‘ do not allow any substitutions in their content.
   1. echo “$A$A is a GPS navigator”
   2. echo “There are `grep “/gr211/” /etc/passwd | wc –l` students in group 211”

## Shell Scripts

1. Any text file with execution permissions can be a script, as long as it contains commands interpretable by the current shell. Comments start with #
2. Hello World example
   1. Create file a.sh with the content below

echo Hello World

* 1. Give the script execution permissions using chmod 700 a.sh
  2. Execute the script using ./a.sh

1. Permissions
   1. Run ls –l and see the first 10 characters on each line
      1. The first character tells the file type: - is a regular file, d is a directory
      2. Characters 2-4 show the permissions for the owner of the file (field 3 displayed by ls –l)
      3. Characters 5-7 show the permissions for the group of the file (field 4 displayed by ls –l)
      4. Characters 8-10 show the permissions for everybody else
   2. Each permission triplet describes the read, write and execution permissions
   3. Can be described as a number, by considering each of the 3 positions to be a binary digit.
      1. 7 = 111 = rwx
      2. 6 = 110 = rw-
      3. 5 = 101 = r-w
   4. Command chmod is used to assign permissions to files
      1. chmod 700 a.sh gives the owner of the file full permissions, and nothing to the group or the others
2. Hello World example with shell specification
   1. Create file hello.sh with the content below

#!/bin/bash

echo Hello World

* 1. Give the script execution permissions using chmod 700 hello.sh **or** chmod +x hello.sh
  2. Execute the script using ./hello.sh

1. Special variables
   1. $0 – The name of the command
   2. $1 - $9 – Command line arguments; $n the n-th argument; shift n to shift cmd line arguments by n
   3. $\* or $@ - All the arguments together as string or as array
   4. $# - Number of command line arguments
   5. $? – Exit code of the previous command
2. Special variables example, special-vars.sh

#!/bin/bash

echo Command: $0

echo First four args: $1 $2 $3 $4

echo All args: $@

echo Arg count: $#

true

echo Command true exited with code $?

false

echo Command false exited with code $?

* 1. chmod 700 special-vars.sh
  2. ./special-vars.sh a b c d e f g h i j k l m

1. Accessing arguments using shift. Script using-shift.sh

#!/bin/bash

echo Command: $0

echo First four args: $1 $2 $3 $4

echo All args: $@

echo Arg count: $#

shift

echo Some args: $1 $2 $3 $4

echo All args: $@

echo Arg count: $#

shift 3

echo Some args: $1 $2 $3 $4

echo All args: $@

echo Arg count: $#

* 1. chmod 700 using-shift.sh
  2. ./using-shift.sh a b c d e f g h i j k l m

## UNIX Shell FOR Loop

1. Similar to the Python foreach. The variable cycles through a list of space separated values. Basic example using-for.sh, showing the do on the same line or on the next line. Semicolon is the command separator.

#!/bin/bash

for A in a b c d; do

echo Here is $A

done

for A in a b c d

do

echo Here is $A

done

* 1. chmod 700 using-for.sh
  2. ./using-for.sh

1. Iterating over the command line arguments, for-args.sh, showing the short and not very intuitive second possibility

#!/bin/bash

for A in $@; do

echo Arg A: $A

done

for A; do

echo Arg B: $A

done

* 1. chmod 700 for-args.sh
  2. ./for-args.sh a b c d e f g h i j k l m

1. Count all the lines of code in the C files in the directory given as command line argument **and its subdirectories**, excluding lines that are empty or contain only blank spaces

#!/bin/bash

S=0

for F in `find $1 –type f –name “\*.c”`; do

N=`grep -E -v -c "^$" $F`

S=`expr $S + $N`

done

echo $S

* + - * + Cand dorim **outputul** unei comenzi (adica ce afiseaza la STDOUT )folosim

aprostreafe intoarse: ` grep -E -v -c "^$" $F `

sau operatorul $() vu acelasi effect : $( grep -E -v -c "^$" $F)

* + - * + Cand dorim sa **evaluam apresii artimetice** folosim

Expr : S=`expr $S + $N` sau S=$(expr $S + $N)

Sau pperatorul $(()) fara $ in interior : S=$((S+N))

* + - * + Cand dorim **codul de retur** (true sau false) nu folosim evaluatori `` sau $()

if grep -q -E $username /etc/passwd ; then

echo $username exista in system

fi

sau

if ! test -d $1 ; then

echo $1 nu este un director

exit 1

fi

1. Să se scrie un script shell care va tipări numele de utilizator (username), numele complet al utilizatorului (full user name) și directorul personal al tuturor utilizatorilor din sistem. Utilizatorii unui sistem unix sunt reținuți în fișierul /etc/passwd (sau /etc/pseudopasswd). Tiparirea se va face la iesirea standard.

#!/bin/bash

cat /etc/passwd | cut -d: -f1,5,6"

#sau

awk -F: ‘{print $1, $5, $6}’ /etc/passwd

1. Să se scrie un script shell care numără procesele unui anumit utilizator dat ca parametru în linia de comandă. Rezultatul va fi afisat la iesirea standard.

#!/bin/bash

p=`ps -au $1 | wc -l`

np=`expr $p - 1` # deoarece e linia de head in plus

echo "numar de procese pentru: "$1" este: "$np

|  |
| --- |
| #!/bin/bash  for user in $@; do  if grep -E -q $user /etc/passwd ; then  np=`ps -ef | grep -E -c $user`  np=`expr $np - 2`  echo $user ruleaza $np procese  else  echo $user nu exista in sistem  fi  done |

1. Scrieti un program care primeste ca prim argument un nume de fisier apoi oricate cuvinte. Se cere sa se stearga toate aparitiile cuvintelor date ca parametrii din fisierul care este dat ca prim parametru al programului.

#!/bin/bash

fisier=$1

if [ ! -f $fisier ] # verificam ca primul parametru este fisier

then

echo $1” trebuie sa fie fisier”

exit 1

fi

shift # mutam argumentele cu o pozitie spre stanga, pentru a sari peste numele fisierului

cp $fisier copie # facem o copie la fisier numita copie, pentru ca vom modifica continutul lui

for cuv in "$@";

do

sed -i "s/$cuv//g" $fisier #optiunea -i modifica permanent fisierul, stergand cuvintele

done

1. Sa se calculeze numarul de vocale dintr-un fisier text dat ca parametru in linia de comanda.

#!/bin/bash

if [ ! -f $1 ] # verificam ca primul parametru este fisier

then

echo $1” trebuie sa fie fisier”

exit 1

fi

# cautam cu grep o vocala (-i case insensitive, deci si litere mari si litere mici)

# cu -o afisam fiecare potrivire separate pe o linie (deci fiecare vocala gasita pe o linie)

# cu pipe continuam procesarea cu wc -l care Numara liniile gasite, pe care avem cate o vocala

nv=`grep -E -i -o “[aeiou]” $1 | wc -l` # in nv vom avea numarul rezultat

echo Numarul de vocale este $nv

## UNIX Shell IF/ELIF/ELSE/FI Statement

1. Every command is a condition. Commands can be grouped with parentheses and logical operators
   1. Check whether a file does not exist or if it exists whether it is not readable
   2. ! test -f a.txt || ( test -f a.txt && ! test -r a.txt )
2. Present the basic IF syntax, using script basic-if.sh which checks each argument and announces whether it is a file, or a directory, or a number, otherwise it states that it does not know what it is. Just like do, then can be either on the same line or on the next line. Do not introduce the [ ... ] syntax.

#!/bin/bash

for A in $@; do

if test -f $A; then

echo $A is a file

elif test -d $A

then

echo $A is a dir

elif echo $A | grep -q "^[0-9]\+$"; then

echo $A is a number

else

echo We do not know what $A is

fi

done

* 1. chmod 700 basic-if.sh
  2. ./basic-if.sh /etc /etc/passwd . 1234 a2b rr

1. **To make the condition look a bit more natural, there is a second syntax, in which [ is an alias of command test and ] marks the end of the command test**. Pay attention to leaving spaces around these square brackets or there will be syntax errors. The basic IF example from the lecture, can be re-written as follows

#!/bin/bash

for A in $@; do

if [ -f $A ]; then

echo $A is a file

elif [ -d $A ]

then

echo $A is a dir

elif echo $A | grep -q "^[0-9]\+$"; then

echo $A is a number

else

echo We do not know what $A is

fi

done

|  |
| --- |
| #!/bin/bash  # tipuri de validare numere  for n in $@  do  if echo $n | grep -E -q "^-?[0-9]+\.?[0-9]\*$"  then  echo $n este numar  fi  if [ $n -eq $n ] 2>/dev/null  then  echo $n este un numar intreg  fi  if [[ $n =~ ^-?[0-9]+\.?[0-9]\*$ ]]  then  echo $n verificat cu [[ ]]  fi  done |

## UNIX Shell WHILE Statement

1. Read user input until the input is stop
2. The user input is read with command read which stores the input in the variable given as argument
3. Script basic-while.sh

#!/bin/bash

while true; do

read X

if test "$X" == "stop"; then

break

fi

done

* 1. chmod 700 basic-while.sh
  2. ./basic-while.sh

1. Read the console input until the user provides a filename the exists and can be read

#!/bin/bash

F=””

while [ -z "$F" ] || [ ! -f "$F" ] || [ ! -r "$F" ]; do

read -p "Provide an existing and readable file path:" F

done

or

#!/bin/bash

F=””

while test -z "$F" || ! test -f "$F" || ! test -r "$F"; do

read -p "Provide an existing and readable file path:" F

done

1. Analyse the code sequences:

|  |  |  |
| --- | --- | --- |
| echo “expr 1 + 2”  echo `expr 1 + 2`  echo `expr 1+2` | Foo=sun  echo $Fooshine  echo ${Foo}shine | count=$((count+1))  count=$((count + 1))  count=count+1 |

1. Verify if a variable is a number

a.

|  |
| --- |
| if echo “$var”| grep –q “^-?[0-9]\*\.?[0-9]\+$”; then  echo “$var is a number”  else  echo “$var is not a number”  fi |

* 1. Redirectam eroarea standard 2 cu operatorul >

|  |
| --- |
| #!/bin/bash  var=a  if [ “$var” –eq “$var” ] 2>/dev/null; then  echo Is a number  else  echo Is not a number  fi |
|  |

1. Sort files given as cmd line arguments in ascending order according to file size

|  |
| --- |
| #!/bin/sh  for i in $\* ;do  if [ -f $i ]  then  du –b $i  fi  done | sort -n |

Sau utilizand fisier temporar si operatorul append >>

|  |
| --- |
| #!/bin/sh  Touch temp.txt  for i in $\* ;do  if [ -f $i ]  then  du –b $i >> temp.txt  fi  done  sort -n temp.txt  rm temp.txt |

Salvam rezultatul intr-un fisier folosind operatorul de redirectare >

./script fis1.txt fis2.txt > rezultat.txt

1. Citim din fisier cu operatorul redirectare < . Scriptul primeste ca prim parametru un nume de fisier.

#!/bin/bash

if [ ! -f $1 ]; then

echo Primul parametru nu e fisier

exit 1

fi

l=0

while read line ; do

l=`expr $l + 1`

echo $l: $line # afisam numarul liniei si linia citita din fisierul dat ca prim param

done < $1 # $1 este input pentru read din while

|  |
| --- |
| #citeste de la tastatura cu read pana cand se da 0.  s=0  while true; do  read n  if [ $n == "0" ]; then  break  fi  s=`expr $s + $n`  done |

9. Scriptul primeste perechi de cuvant fisier si afiseaza de cate ori apare cuvantul in fisierul corespunzator

./perechi.sh cuvan1 fisier1 cuvant2 fisier2 cuvant3 fisier3

|  |
| --- |
| #!/bin/bash  if [ `expr $# % 2` -eq 1 ]; then  echo Ne trebuie prechi vuvant fisier  exit 1  fi  touch temp.txt  while [ $# -ne 0 ]; do    if [ -f $2 ]; then  n=`grep -E -o -i "\<$1\>" $2 | wc -l`  echo Cuvantul $1 apare de $n ori in fisierul $2.  echo $n $2 >> temp.txt  fi  shift 2  done  sort -n -r temp.txt  rm temp.txt |

\*\*\*. Construieste o comanda proprie:

In .bashrc (a hidden file in your personal home folder ~ ) add at the end these lines

if [ -r /etc/bashrc ] ; then

. /etc/bashrc

fi

. /home/alina/mylibraries/**myfunc** # path to your function

File **myfunct** contains:

#!/bin/bash

function alina {

clear

echo -e "Hi Alina!"

sleep 1

echo -e "Here is a motivational quote for you `whoami`!\n"

sleep 1

motivate # external package installed

echo " "

echo " "

if [ "$1" == "b" ] ; then

echo "Flowers for you" | boxes -d spring # external package:

fi

}