Session 11:
Bash Shell Scripting

Contents

- Loops: for, while
- Script Arguments
- Functions

Loops: for and while

For samples

```
for x in 1 2 a
do
   echo $x
done
```

```
for x in *
do
   echo $x
done
```

```
for x in {1..4} {5..20..3}
do
   echo -n "$x "
done
1 2 3 4 5 8 11 14 17 20
```

For samples

```
for i in $(cat list.txt) ; do
  echo item: $i
done
```

```
for (( i=0; i<10; ++i )) ; do
  echo item: $i
done
```

While sample

```
COUNTER=0
while [  $COUNTER -lt 10 ] ; do
    echo The counter is $COUNTER
    let COUNTER=COUNTER+1
done
```

Iterate over Lines of a File

```
while read $Line
do
        # do sth with line
done <filename
```

Loop Control

- `break` terminates current loop
- `continue` causes a jump to the next iteration of the current loop

- `break n`: break n levels
- `continue n`: resume at the nth enclosing loop

دستور کار: 4 - shell scripting

- برای نوشتن اسکریپت‌های زیر از حلقه استفاده کنید.

اسکریپتی بنویسید که:

1. تعداد فایل‌های واقع در مسیر /usr/bin/ را که با هر یک از حروف شروع می‌شود در فایل /tmp/letter-no قرار دهد؛ یعنی به صورت زیر:

a: 50
b: 38
c: 32
...

2. تعداد فایل‌های واقع در مسیر /usr/bin/ که از اندازه‌ی آنها از 1MB بیشتر است را در خروجی چاپ کند.

- راهنمایی: دستور du با آپشن s– اندازه‌ی فایل داده شده برحسب کیلوبایت را به همراه نام فایل نمایش می‌دهد. برای استخراج اندازه از خروجی du از دستور awk '{print $1}' استفاده کنید.

Functions

- As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion.
- Declaring a function is just a matter of writing function my_func { my_code }.
- Calling a function is just like calling another program, you just write its name.

Local variables

```
#!/bin/bash
HELLO=Hello
function hello {
    local HELLO=World
    echo $HELLO
}

$ echo $HELLO
$ hello
$ echo $HELLO
```

Functions with parameters sample

```
#!/bin/bash

function quit {
    echo 'Goodbye!'
    exit
}
function hello {
  echo "Hello $1"
}


for name in Vera Kurt ;
do

  hello $name

done

quit
```

Output:

```
Hello Vera

Hello Kurt

Goodbye!
```

Scripts and Arguments

- Scripts can be started with parameters, just like commands

  aScript arg1 arg2 …

- The scripts can access these arguments through shell variables:
  - "$*n*" Is the value of the n^{th} paramter.
    - The command is parameter zero
  - "$#" Is the number of parameters entered.
  - "$*" Expands as a list of all the parameters

Some Special Variables

- `$#` the number of arguments
- `$*` all arguments
- `$@` all arguments (quoted individually)
- `$?` return value of last command executed
- `$$` process id of shell

Handle User Input

- Use `read` command:
  - `read VAR1`
  - `read VAR1 VAR2`

Debugging Tip

- If you want to watch the commands actually being executed in a script file, insert the line "`set -x`" in the script.

```
set -x
for n in *; do
  echo $n
done
```

- Will display the expanded command before executing it:

```
+ echo bin
bin
+ echo mail
mail
```

Parameter Expansion

- `${parameter:-word}`
  - Use Default Values.
- `${parameter:=word}`
  - Assign Default Values.
- `${parameter:?word}`
  - Display Error if Null or Unset.

More Parameter Expansion

- We can remove parts of a value:
  - `${param#pattern}`
  - `${param##pattern}`
    - removes shortest (#) or longest (##) leading pattern, if there's a match
  - `${param%pattern}`
  - `${param%%pattern}`
    - removes shortest(%) or longest (%) trailing pattern, if match
- *pattern* is expanded just as wildcards - *, ?, [] – (not regexes).

More Parameter Expansion

- Find the length of a string:
  - `echo ${#foo}`
- Extract substrings
  - `echo ${foo:2:3}`
- Regex search and replace
- There are more. See the Bash manpage

Example – Parameter Expansion

```
$ foo=j.i.c
$ echo ${foo#*.}
i.c
$ echo ${foo##*.}
c
$ echo ${foo%.*}
j.i
$ echo ${foo%%.*}
j
```

دستور کار: 4 - shell scripting

1. اسکریپتی بنویسید که دو عدد را به عنوان پارامتر گرفته، که اولی باید کوچکتر یا مساوی دومی باشد، و در خروجی فهرست اعداد بین این دو را (شامل دو عدد ورودی) چاپ کند.
2. یک دایرکتوری ایجاد کرده و به آن وارد شوید. چند فایل در آن ایجاد کنید. اسکریپتی بنویسید که دو رشته را از ورودی گرفته و همه ی فایل های واقع در دایرکتوری جاری را با استفاده از دو رشته ی ورودی تغییر نام دهد. رشته ای اول را پیشوند و رشته ی دوم را پسوند در نظر بگیرید؛ مثلاً:

```
./myrename.sh trip jpg
file1,file2, … becomes trip1.jpg, trip2.jpg,…
```

اسکریپت قبل از انجام کار باید به کاربر هشدار داده و از وی تایید بگیرد.

Last Note: Text Processing Tools

- awk, cut, tr, sed, grep, sort, wc, ...

Compile Kernel in Debian

- Why compile kernel?!

- `$ apt-get install fakeroot kernel-package linux-source-x.y libncurses5-dev bzip2`
- **fakeroot**: runs a command in an environment wherein it appears to have root privileges for file manipulation. Useful for allowing users to create archives (tar, ar, .deb etc.) with files in them with root permissions/ownership.
- **kernel-package**: used to create the kernel related Debian packages
- **linux-source**: kernel source code (vs. vanilla kernel?)
  - Need around 6 GB disk space at the end of compilation.
- **libncurses5-dev**: simple graphic menus for make menuconfig command

Compile Kernel in Debian

- `$ tar xjf /usr/src/linux-source-x.y.tar.bz2`
- `$ make menuconfig`
  - Apply your changes or directly change source code,
  - / for search, * for to select parameter …
- `$ fakeroot make-kpkg --initrd --revision=1.0 kernel_image`
  - Takes time depending on the machine power
  - My case: with Vbox VM, a Single 2.8 GHz Core, 2 GB RAM: took about 1.5 hour,
- Last command generates the deb package: linux-image-x.y.z_1.0_arch.deb
  - Contains mainly: kernel image (vmlinuz) and loadable modules,
- `dpkg -i linux-image-x.y.z_1.0_arch.deb`
  - Installs kernel files and configures GRUB properly; pretty easy!
- Reboot.