Test Lab
Linux Shell / GNU Make
Embedded Linux

Julia Desmazes
Michael Nissen

October 17, 2017

Command line, environment variables, and user permissions

1. In the bash prompt, what is the meaning of the character ~ ?

The tilde (~) character refers to the users home directory. The full path is `/home/user`.

2. Explain the behaviour of running in the order `VV=3`, `export VV`, `bash`, `unset VV`, `exit` and finally `echo $VV`?

Environment variables is a value, named at runtime, that can affect the way running processes behave. A running process will be able to query the value of an environment variable for example, to discover a location to store files. When creating an environment variable, it will persist in the shell in which it was initalised.
When exporting the creatied environment variable by using `export VARNAME`, it sets the environment variable not only for current shell, but all other processes started from that shell.

- Set local variable VV to 3
- Export local variable as environment variable
- Enter the bash shell
- Unest the local variable
- Exit the bash shell
- Display the environment variable

3. How to run `/home/user/ls` instead of `/usr/bin/ls` automatically by typing `ls` without changin the behaviour of the other commands?

An easy way to achive this kind of behaviour, is to create a shell alias. A shell alias is simply a shortcut to reference a command. It can be used to avoid typing long commands or as a mean to run increase efficiency. In this example, we want to run the `/home/user/ls` command instead of the buildin one located in `/usr/bin/ls`. As the `/usr/bin` folder is added to the path of the shell, one is able to run any script from that folder by simply entering the name of that script.
As we don't necessarily want to add the `home/user` folder to the path, we can create an alias to the specific file we need.

One can create an alias using echo and redirecting it to the shell configuration file:
`echo alias 'ls="/home/user/ls"' >> .zshrc` where the last file is your shell configuration file.

4. How to provide `file.txt` as input of the command `flex` and copy its `stderr` to `resu.txt`?

3 different streams. Use 2> to redirect the content on stderr. (-p writes the performance report to stdderr)
`flex -p file.txt 2> resu.txt`

5. Propose a command that displays the middle line of `/etc/passwd` ?

In this commande we are using `wc -l` to get the line count of file `/etc/passwd` and storing it in a bash variable named NUM. Then we are dividing NUM by two `expr $NUM / 2` and start grabbing the line starting from this new line count with `head -n $NUM /etc/passwd`. In order to only grab one line from this line, we pipe the output to `tail -1`, witch grab only the first line.

```
NUM=$(wc -l </etc/passwd)&&NUM=$(expr $NUM / 2 )&& head -n $NUM /etc/passwd | tail -1
```

6. How to change the owner of a first file to the owner of a second one by using a command substitution based on the command `ls` and `cut` ?

Easy way is to just use the build in functionality of `chown`. No need to reinvent the wheel:
`chown -reference=result.txt file.txt`

The `grep`/`sed` and Ragular Expressions

1. Propose a `grep` command that displays user names in `/etc/passwd` whom UID is multiple of 2?

We use `grep` line by line on `/etc/passwd` and select all the UID's that finish by either 0,2,4,6,8. Then we pipe the output to cut where we delimite the output with `:` and select only the first match `cut -d: -f1`.

```
grep '\^[\^:]*:[\^:]*:[0-9]*[02468]:.' /etc/passwd | cut -d: -f1
```

2. Propose a grep command that displays from the output of `ifconfig`, the WiFi IP address?

To only display the wifi IP adresse we first only printed out the content of wlan0[1], then used `grep 'inet '` to select the line where the IPv4 IP addresse is displayed. Notice we added a space behind `inet`. This was to avoid matching inet6. Afterwards we piped the output to `awk $2` in order to select only the second motif[2], here our IP.

```
sudo ifconfig wlan0 | grep 'inet '| awk '{print \$2}'
```

Shell Script

Write a Shell script that displays the entries of `/etc/passwd`, using `sed` and `for` loops

`Username: <username>, Password: encrypted, UID: <uid>, GID: <gid>, Home: <home_directory>` where i is the line number of the corresponding <username> in `/etc/passwd`.

The script for this can be found in the `script` folder under the name `script3.sh`.
In this script we are using the `for` loop to read the file `/etc/passwd` line by line. Also we have a local

[1] Refers to the wifi hardware.
[2] By default seperation of `awk` is the space, this can be changed with the `-F` argument.

value `i` to keep trake of the line number. For the `sed` commande we identifing substrings with our regex `(<match substring>)` and refrencing them latter with `<number>` pattern in the latter part of our commande where we re-define the output format.

```
#!/bin/bash

#init i
i=1
#read line by line
for line in $(cat /etc/passwd)
do
    #increment i
    i=$((i+1))

    #parse and modify output stream with sed
    echo $i":"$line | sed  's/^\([^:]*\):\([^:]*\):\([^:]*\):\([0-9]*\):\([0-9]*\)
    ::\([^:]*\).*/\1) Username: \2 ,Password: encrypted, UID: \4, GID: \5, Home: \6/'
done
```

Code Snippet 1: RegEx in Shell Script

GNU Make

In order to compile both nativly and cross-compile for out RaspberryPi we check at the start of our makefile for the variable `CROSS_COMPILE`. If set, we compile with the cross compilation toolchain that we have stored in tools from our linux kernel compilation lab located in `/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-r`
If not, this variable will not be set and we will be using the machines standard compiler: gcc or g++ , depending on the value given to `TMPGCC` at the start of the makefile.
To access `CROSS_COMPILE` in the makefile we have to make sure the variable is exported to the bash `export CROSS_COMPILE`.
The rest of the makefile is very standard; we generate an object files based on ther .c files and link them together to create our executable main. To clean we select all object file `*.o` and delete them as well as our generated executable.

```
# if we wanted to cross compile to rpi we would use the toolchain we have downloaded for
    the kernel compilation : arm-linux-gnueabi-gcc, but to use it we would have to
    specify a path. In our case we have exported the path to the folder in the vaiable
    CROSS_COMPILE

TMPGCC=gcc

ifeq (${CROSS_COMPILE},'')
    echo "CROSS_COMPILE is not set";
    GCC=${TMPGCC}
else
    # to avoid recursice refrencing to GCC variable
    GCC=${CROSS_COMPILE}${TMPGCC}
endif

ADD=minmax
DEPS=minmax${HEADER_SUFFIX}

all: main

main: main.o minmax.o
  ${GCC} -o main minmax.o main.o

main.o: main.c minmax.h
  ${GCC} -c main.c

minmax.o: minmax.c minmax.h
  ${GCC} -c minmax.c
clean:
  rm -f *.o main
```

Code Snippet 2: MinMax Makefile

Managing Process, System Call `fork()`

The system call `fork()` is used to create processes and returns a process ID. When calling `fork()`, it creates a new process which becomes a *child* process of the caller. After a child process has been created, both the *child* and *parent* process will execute the next instruction following the `fork()` system call. Therefore, one has to distingush the difference between the two processes.
This can be done by testing the process ID returned by the system call.

If `fork()` returns a negative value, the creation of a child process was unsuccessful. It will return a zero to the newly created *child* process, and it will return a possitive value, the `pid` of the *child* process, to the parent.

Therefore it is simple to check what which is the child process - this can be seen in Code Snippet 1 below.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {

    // The process ID
    pid_t pid;

    // Store the process ID
    pid = fork();

    // Check for the process ID
    if (pid == 0) {

        // Child process

        // Create a new fork
        pid = fork();

        // Check for the process ID
        if (pid == 0) {

            // Grand child process
            printf("Child of Child");
        } else {

            // Child process
            printf("Child");
        }
    } else {

        // This is the parent process
        printf("Parent");
    }
}
```

Code Snippet 3: Managing system calls