The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific
and Cultural Organization

International Atomic
Energy Agency

310/1779-4

Fourth Workshop on Distributed Laboratory Instrumentation Systems
(30 October - 24 November 2006)

Linux Refresher (1)

Razaq Babalola IJADUOLA

Strada Costiera 11, 34014 Trieste, Italy - Tel. +39 040 2240 111; Fax +39 040 224 163 - sci_info@ictp.it, www.ictp.it

Linux Refresher

Fourth Workshop on
Distributed Laboratory
Instrumentation Systems
(Trieste – Italy)

Razaq Ijaduola
30 October to 24 November 2006

Introduction

- Linux is a 'UNIX'-like operating system
- It was launched in 1991 by Linus Torvalds, a Finnish student.
- Hundreds of people contributed to it and thousands tested and reported bugs.
- It is therefore extremely stable.
- Millions of people, institutes and firms are using it now.
- It is distributed under the 'GNU General License' and it is practically cost-free.
- You can download it at no cost at all.

At the risk of boring those who know Linux already, I will present some of its features and show you a few commands which hopefully will turn out to be useful in your work during the next weeks (and later!).

Practically all present day operating systems are sort of descendants of UNIX: even MSDOS and Windows have taken features of it.

Linux is the interface between the hardware of the machine and programs running on it.

A program requests services from the system via a system call.

Such a system call could request to read a given sector from disk, or to allocate a block of memory, or to send out a character on the serial port, etc.

A user, sitting in front of his terminal, interacts with the system via a command interpreter or shell.

The shell translates from some 'jargon' which is not too difficult to understand into sequences of mysterious system calls.

There are a number of shells in circulation
('sh', 'ash', 'ksh', 'csh', 'tcsh', etc.).
You are supposed to use bash, the 'Bourne Again Shell'.

Features of Linux

- Here are some of the features of Linux:

- Files are stored in a hierarchical tree structure of directories and subdirectories (/ = top level).
  Shorthand for directories:
  - . stands for 'present directory'
  - .. stand for 'parent directory'
  - ~ stands for user's home directory.

  Preference files (hidden files) starts with “.”
  .bashrc, .mozilla/
- Files have access permissions (drwxr-xr-x).
  file type | owner | group | others (everyone else)

Features of Linux

- Files are a collection of bytes. No distinction between file types.
- Filenames are not limited to eight characters, there no suffixes.
- Linux is a multi-tasking, multi-user, time-sharing system.
- Programs of a user are protected against incursions by other users.
- Memory can be allocated dynamically.
- You must also strongly resist pushing 'reset' or
- hitting 'Ctrl-Alt-Delete'.

- Linux, with the help of the 'shell' is very flexible:
  - Most commands have (lots of) options.
  - You may change the name of the commands, or define (a set of) standard options for each of them: aliases.

    Examples:

    alias ts "cd /usr/local/micros/m6809/src/tests"

    alias cp "cp -p"
  - You can also access files with a sort of 'pseudonym', using 'symbolic links'.

    ln -s targetname linkname
  - Command 'completion':

    You type:

    less longf'tab' (longf, followed by a 'tab')

    and the shell translates into:

    less longfilename.longpostfix

    if that is an unambiguous, existing filename.

- User environment – a collection of specially named variables that have specific values.
- They are called environment variables:

to view the environment use:

- env or printenv (depending on the type of shell)

EX=/usr/local/micros/m6809/bin

- If you now type: $EX/cc09 -v -o myprog myprog.c the shell will launch execution of:

/usr/local/micros/m6809/bin/cc09 -v -o myprog myprog.c

- Use the echo command to list specific environment variable:
- echo $HOME or echo $EX

- An important environment variable is PATH.

PATH contains the list of directories where shell will look for the executable specified (by you) on the command line. You can set the PATH:

PATH=$PATH:/home/razaq/workarea

- Getting help
- You can get help on practically all commands from the man pages (manual pages).

  man command
- It is wise to invoke xman & after logging in.
- Other way to find help is to use -h for the command option.

Using the commands

- You must get accustomed to command names:
- 'passwd' for changing the password
- 'cp' for copy,
- 'rm' for del,
- 'ls' for directory list.
- 'mv' for moving or renaming a file
- 'mkdir' for creating a directory
- 'chmod' for changing the permission on a file or dir.
- 'less' or 'more' or 'cat' for listing the contents of a file
- 'ps' for reporting a snapshot of the current processes
- 'pwd' for showing the current location or current working directory

- When you have logged in, you will find yourself working in your home directory.
- When login for the first time, you may want to change your password using passwd. You will be asked for your current password and a new one.

- The contents of the any directory can be listed:

  ls

- You move to another directory with:
  cd dir-path, where 'dir-path' is either a full or a relative pathname.

- If I am in /home/razaq and I want to go to /home/jim/ I do:

cd /home/jim/ or

cd ../jim/

cd $EX takes me straight to:

/usr/local/micros/m6809/bin

- If you are lost and don't know where you are, using pwd will show your current location.
- creating a new directory:
- mkdir newdirname

- moving a file from one directory to another:

  mv source existing_dir.
- can also use mv to rename a file or directory

  mv source newname
- To delete a directory:

  rmdir dir_to_delete
- To delete a file(s) or directory:

  rm filename
- THINK TWICE BEFORE USING rm and rmdir !!!!
- You can change the permission mode for file and dir.

  chmod [ugoa] [+-=] [rwx]
- To display information about the active processes one can use ps:

  ps [options]

- Pipes:

command1 | command2 means:
Output from 'command1' is fed into 'command2' as its input. 'command1' and 'command2' are so-called filters.

- Normally when shell launches a command, it waits for its completion. Appending an '&' to a command changes this behaviour:

the shell takes back control immediately
after having launched the command.

- Consecutive commands:
- ; separates two commands; they will be executed one after the other.

Redirection:

- > file --Output destined for the screen (stdout) is written to 'file'. The file is overwritten.
- >> file --As above, but output is appended to existing file.
- 2> file and 2>> file: same as above, but now concerning error output (stderr).
- < file --Input is taken from 'file', instead of keyboard (stdin).

- Grouping commands:
- (comm1 ; comm2) any

comm1 and then comm2 will be executed before 'any' thing following after the ')' will be done

So, the following is perfectly legitimate (and even very useful!):

```
dump /usr/local f - | (cd /mnt/usr/local ; restore f - )
```

- Shell scripts:

You may collect a series of shell commands into a file.
After having set the execute permission for this file,
you can execute its contents with a single command:
the name of the shell script.

The first executable line in the script must be:

#!/bin/sh or #!/bin/bash or similar.

- Utility Programs

There are a number of utility programs you should be aware of. They can be very useful and you may find yourself using them repeatedly when you are developing software.

- The programs we want to talk about very briefly are:

cat, more, less, whereis, find, grep, tar, and sed.

- We will also dwell on:

make, gcc, gdb , to conclude with the development cycle:

--> edit --> compile --> debug -->

| |

<-- <-- <-- <--

cat, more or less

- cat filename lists a text file to your screen, you can read the last few lines.

- more filename does the same, but shows one screenful at a time.

(SPC) scrolls one entire page, (RET) one line.

- less filename does it better: you can go back also, one screenful at a time, typing a 'b'.

- cat file1 file2 ..... filen > newfile

concatenates the files and writes result to newfile.

- You may also try head and tail.

whereis, find

- whereis name-of-executable will show where the executable file is stored.

- find is of much more substance. It will find files of a given name, date, or length or older than date, longer than length, etc.

What is more, once a satisfactory file is found, you can make 'find' to do something with it.

whereis, find cont...

- One can explain 'find' for hours. For now, four "simple" examples:

```
find /usr/bin -name unzip

find . -cmin -60

find .. -size +110k

find .. -size +110k -printf “\t %s \t %p \n”

find .. -size +110k -exec tar zcvf { };
```

whereis, find cont...

Reading the 10 or so man pages is a MUST , if you wish to make reasonable use of 'find'.

tar

'tar' stands for 'tape archiving' which was the original aim of this useful program. It is now often used to make a nice compressed package of a (large) number of files:

tar zcvf parcel.tar.gz file1 file2 .... filen or
tar zcvf parcel2.tgz ./*

The latter will create (option 'c') verbosely ('v')
a compressed ('z') tar' file ('f') parcel2.tgz
from all files in the current directory.

Once created, you can inspect the contents:

tar ztvf parcel2.tgz | less

tar continue...

You can unpack your 'parcel2.tgz' with:

tar zxvf parcel2.tgz

sed

'sed' is a "stream editor" or "filter". It works on the contents of an entire file to delete, insert, append lines or replace a string and more. It works non-interactively.

For this reason you will see sometimes in a Makefile things looking like:

```
itmp=”echo $$itmp | sed 's;\.\./;;'”
```

sed continue...

More down-to-earth use are:

```
sed -e '1,10d' sample.txt
```

-e option to sed tells it to use the next item as command.
d command tells sed to delete lines 1 through 10 of the input stream sample.txt.

sed cont...

sed s/Jim/Pablo/g infile > outfile;

s means substitute Jim with Pablo.

Note that you can use 'sed' in pipes:

cat infile | sed -e s/Jim/Pablo/g >outfile

grep

'grep' is a most useful program:

grep -n string *.c *.h
will print for you all lines in all '.c' and
'.h' files in the current directory where the
string has been found, preceded by the
filename and the linenumber.
ls -l ~ | grep \.ps
will show all your '.ps' files and only those.

grep continue...

Similarly for creation dates:
ls -l ~ | grep May or even:
ls -l ~ | grep \.ps | grep -v May

The '-v' option will show all lines that do NOT contain the string.