LPIC-1 101-500 – Lesson 1

103.1 Work on the Command Line (CLI)

Terminology

- There are many different terms related to the CLI:
  - **Shell**: the command line interface that runs in a terminal to execute commands.
  - **Terminal**: Programs that emulate the behavior of an old school Unix terminal (e.g. VT100)
  - **Console**: Synonym to Terminal

Examples of Shells

- **bash**: the most popular shell in Linux, default for most systems.
- **bsh**: a simple shell upon which bash was based on
- **dash**: combines the speed of bsh with the functionality of bash
- **csh/tcsh**: inspired from C. Fundamentally different from bash
- **ksh**: merges elements of bsh and csh
- **zsh**: a feature-rich and powerful shell

Examples of Terminals

- GNOME Terminal
- Konsole (KDE)
- xterm
- Terminator
- TTYs (Ctrl-Alt-F2 … F6)
- MobaXterm (Windows)
- PuTTY (Windows)
- TeraTerm (Windows)
- Windows Subsystem for Linux - WSL (Windows)

The Shell Prompt

- **user@hostname: src$**

  The ”$” sign implies non privileged user

- **root@hostname:~#**

  The ”#” sign implies privileged user (root)

- **echo $PS1**

  The $PS1 variable (Prompt String 1) defines the shell form:

  [\u@\h \W]\$

  explanation: \u: username, \h:hostname \W:basename

- Additional information:

  $ man bash # Lookup PROMPTING

Basic Command Syntax

- **<command> <command options> <arguments>**
  e.g.:

```
$ ls -la src

total 24

drwxrwsr-x  6 root src  4096 2011-06-21 11:34 .

drwxr-xr-x 11 root root 4096 2011-05-29 14:34 .

drwxr-xr-x  4 root root 4096 2011-05-29 14:34 fglrx-8.840

drwxr-xr-x 24 root root 4096 2011-05-29 14:30 linux-headers-2.6.38-8

drwxr-xr-x  7 root root 4096 2011-05-29 14:30 linux-headers-2.6.38-8-generic

drwxr-xr-x 11 root root 4096 2011-06-21 11:34 virtualbox-ose-4.0.4
```

Builtins and external commands

- Builtin commands are commands provided by the shell itself, e.g. export, alias, cd etc
- more info: man builtins
- External commands are distinct executable files, e.g. **ls**, **man**, **which**

  more info: **man <command>**
- There are commands that are both external and builtin, like **echo** and **pwd**

  In this case priority goes to builtins

Basic Commands

- **cd**: change directory
- **pwd**: print working directory
- **echo**: print text/variables in stdout
- **export**: export variables
- **man**: manual pages for commands
- **uname**: system information
- **exec**: Execute a file
- **type**: Show type of command
- **which**: show path of external command
- **exit**: exit current session/shell
- **logout**: exit current session
- **time**: calculate execution time
- **history**: show command history
- **env**: show environment variables
- **set**: show/set variables
- **unset**: unset variables
- **history**: show list of past commands

Absolute – Relative Paths

- Absolute paths always start with “/”, e.g.:

  /home/user/bin

- Relative path start from the current direstory, e.g.:

  ./bin points to **/home/user/bin** if you are in **/home/user** already

- The dot and slash “**./**” can be omitted, e.g.:

  bin points to **/home/user/bin** if you are in **/home/user** already

- A double dot and slash **“../”** is interpreted as “Go back one directory” e.g.:

  ../user2/bin points to **/home/user2/bin** if you already in **/home/user**

- The tilde character “~” and the variable **$HOME** point to the current user’s home directory (homedir), e.g.

  if the user name is “**user**” then **~/bin** and **$HOME/bin** point to **/home/user/bin**

Command Execution

- First priority goes to builtins.
- Next priority goes to every executable file in the **$PATH**, e.g.:

  ```
  $ echo $PATH
  /home/theo/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/us
  r/bin: /sbin:/bin:/usr/games
  ```

- Directories in the left side are given higher priority than directories in the right, e.g. **/home/theo/bin/ls** has priority over **/bin/ls**
- `$ ls -l /bin/bash`

```
-rwxr-xr-x 1 root root 954896 2011-04-01 00:20 /bin/bash
```

The `x` character states that the file **/bin/bash** is an executable.

Command Execution

- For commands not included in the **$PATH** you should explicitly define the absolute or relative path, e.g: **/usr/lib/gettext/hostname** or **./commands/testing**
- The **exec** command can execute other executables
- To execute a command in the current directory we use “**./**” e.g.: **./testing**
- For successive command execution we can use “**;**” e.g.: **<cmd1> ; <cmd2> ; <cmd3>**

Command Substitution

- We can expand the output of a command to be used as an argument to another command.
- There are two ways to do this: **$(command)** or **`command`**. The former is recommended as it is safer when there are strange meta-characters in the command
- `$ echo $HISTFILE # show the file where the command history is saved`
- `$ ls -l $(echo $HISTFILE) # The echo $HISTFILE command is invoked first and it’s output is passed as an argument to the ls -l command.`

Command Completion

- Bash as well as other shells provide a “**command completion**” feature by invoking the “**Tab**” key
- A single **Tab** will auto-complete the following characters, provided they are unique:

```
$ pass<Tab> → $ passwd
```

- Two successive **Tabs** will display other possible commands, if the set already typed is not unique:

```
$ pas<Tab><Tab> →
passwd          paste               pasuspender
```

Command Completion

- The same logic applies to paths, e.g.:

```
$ cd /var/lo<Tab><Tab> →
local/ lock/  log/
```

- `$ cd /var/loca<Tab> →`

```
$ cd /var/local/
```

Note: some systems (like Ubuntu) have extended this concept to options/parameters completion or even the file type expected by the command.

Command History

- The **history** command will return a list with the most recent commands
- **$HISTSIZE**: this variable will display the size of the command history
  (default: **1000** commands)
- **$HISTFILE**: this variable will return the command history file
- (default: ~/.bash_history)

Command History Expansion

- **!!** Executes the most recent command
- **!n** Execute the nth command. We can use the **history** command to see the command numbers.
- **!-n** Execute the nth from the end of the command history.
- **!string** execute the most recent command starting with the characters "**string**".
- **!?string** execute the most recent command containing the characters "**string**".
- ^**string1**^**string2** repeat last command, replacing "**string1**" with "**string2**".
- **$ fc** edits the most recent command history

Shell Shortcuts

- **Ctrl-p** Go a command back (also ‘Up Arrow’)
- **Ctrl-n** Next command (also ‘Down Arrow’)
- **Ctrl-b** A character backwards (also ‘Left Arrow’)
- **Ctrl-f** A character forward (also ‘Right Arrow’)
- **Ctrl-a** Go to the beginning of a line (also ‘Home’)
- **Ctrl-e** Go to the end of line (also ‘End’)
- **Ctrl-t** Transpose the character left of the cursor with the character under the cursor
- **Ctrl-l** Clear screen but leave the current line to the top of the screen

Note: The Bash shell has the same shortcuts as the Emacs editor

Shell Shortcuts

- **Meta-<** Go to the top of the command history
- **Meta->** Go to the bottom of the command history
- **Ctrl-d** Delete character right of the cursor
- **Ctrl-k** Delete (kill) the text to the end of line
- **Ctrl-y** Paste (yank) the deleted text
- **Meta-d** Delete (kill) the current word
- **Ctrl-rtext** search text backwards
- **Ctrl-stext** search text forward
- **Ctrl-x Ctrl-e** invoke the default text editor

Note: the 'Meta' key is usually assigned to the 'Alt' key

Environment and Shell Variables

- **$ PROXY=http://proxy.domain.int** # set a Shell variable
- **$ export PROXY** # export a variable to child shells (Environment Variable)
- **$ export PROXY=http://proxy.domain.int** # combine the previous two commands in one

The `echo` command

- **$ echo $PROXY** # show the value of the PROXY variable

  ```
  http://proxy.domain.int
  ```

- **$ echo "Proxy = $PROXY"** # Double quotes expand variable

  ```
  Proxy = http://proxy.domain.int
  ```

- **$ echo 'Proxy = $PROXY'** # Single quotes show the exact string

  ```
  Proxy = $PROXY
  ```

Commands env, set and unset

- The **env** command will return the list of environment variables:
  $ env | more # (press q to exit **more**)
- The **set** command will return the list of shell variables:
  $ set | less # (press q to exit **less**)
- **$ unset PROXY** # unset the variable **$PROXY** from the Shell and Environment
- **$ set -o** # status of shell options
- **$ set -o/+o <option>** # set/unset shell options
- **$ set -o vi** # use **vi** shortcuts instead of **emacs** in the **bash** shell
- **$ set +o history** # disable the command history
- **$ set -o allexport** # export all variable to the Environment

The `uname` command

The **uname** command will return some useful information about our system

```
$ uname -a # display all available info
$ uname -r # kernel release
$ uname -n # machine hostname
$ uname -v # kernel version and info
$ uname -o # os name
$ uname -s # kernel name
$ uname -m # system architecture
```

The `which` and `type` commands

- **$ which set** # no external command named `set`
- **$ $ type set** # set is a builtin command
  set is a shell builtin
- **$ which echo** # path of echo external command
  /usr/bin/echo
- $ **type echo** # `echo` is builtin AND external
  echo is a shell builtin
- **$ type ls** # ls is in fact an alias
  ls is aliased to `ls –color=auto'
- **$ \ls** # run the unaliased version of `ls`

Getting Help with commands

- Most command support the **-h** or **--help** options (or both) for basic help, e.g.:
- `$ ls --help`
- `$ gzip -h`
- The **man** will give us a more detailed description of the command, e.g.:

```
$ man bash
```

- Some command make use of the **info** command for an even more detailed description. **info** supports hyperlinks. Example:

```
$ info date
```

”When all else fails, read the manual”
~ Ancient UNIX proverb ~

Manpages Sections

Section ID	Description
1	User Programs and Commands
2	Kernel System Calls
3	Library Calls
4	Devices Files in /dev
5	File Formats
6	Games
7	Various
8	System commands
9	Kernel Routines

Using the `man` command

- **$ man -wa passwd** # show all man files related to **passwd**
- **$ man passwd** # displays the first of the 3 pages based on the priority: **1:8:2:3:4:5:6:7:9**
- **$ man 1 passwd** # shows the man page related to **passwd** in section **1**
- **$ man 1ssl passwd** # shows the man page related to **passwd** in subsection **1ssl**
- **$ man 5 passwd** # shows the man page related to **passwd** in section **5**
- **$ man -a passwd** # shows successively all man pages named **passwd**
- **$ man -f passwd** # (identical to **whatis**) shows a brief description of all pages named **passwd**
- **$ man -k passwd** # (identical to **apropos**) shows a brief description of all pages containing **passwd**
- **$ man -K passwd** # shows successively all man pages named containing **passwd** in their content

License