Work Effectively on the Command Line

Information

These notes were originally written in the year 2000 as part of a set of LPI Exam 101 training materials. The LPI training course at Bromley College was subsequently discontinued and some of the sections of the notes modified and incorporated into our one-day System Administration Courses. The remainder of the notes have now been made publicly available on the linuxtraining.org.uk website.

If you are a beginner please do not be put off of training courses by these notes, as they are rather technical. On the other hand if you are a more experienced Linux user we hope you find the coverage of this topic refreshingly clear.

For full details of our current Linux training please visit the site:
http://ce.bromley.ac.uk/linux

If you have reached this page from a search engine and wish to see the full contents list for the published notes please visit the site:
http://www.linuxtraining.org.uk

We hope you find these notes useful, but please remember that they apply to the 2.2 kernel. I will update them when I have the time.

Clive Gould - 21st December 2004

Work Effectively on the Command Line

Objective 1

Objective 1: Work Effectively on the Unix Command Line. Interact with shells and commands using the command line. Includes typing valid commands and command sequences, defining, referencing and exporting environment variables, using command history and editing facilities, invoking commands in the path and outside the path, using command substitution, and applying commands recursively through a directory tree.

The Shell

The shell acts as an intermediary between the user and the operating system, and acts a a command interpreter, interpreting what you type at the command line into a form the operating system can understand.

A shell script is a command file where commands are read from a file on disk, not from the keyboard. Shell programming is very powerful and it is quite possible to write complete programs using shell scripts.

The default command line interpreter for Linux is BASH (Bourne Again Shell). The default login shell, for each user, is defined in the **/etc/passwd** file. An entry from this file for user clive is illustrated below:

```
clive:x:500:500::/home/clive:/bin/bash
```

The name of the default shell is also defined in the environment variable **$SHELL**. To display the contents of the environment variable you can echo it to the screen as shown below:

```
[clive@redhat clive]$ echo $SHELL
/bin/bash
```

You can choose from a number of shells other than the default such as **ash**, **csh**, **ksh, tcsh** and **zsh**.

Work Effectively on the Command Line

A selection of common command shells is listed below:

Name	*Description*	*Command*
ash	Ash (a small shell) is a version of sh with features similar to those of the System Vshell.	/bin/ash
ash.static	This is a version of ash not dependent on software libraries	/bin/ash.static
bash	bash - GNU Bourne Again Shell. Bash is an sh compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh).	/bin/bash
bsh	A symbolic link to ash.	/bin/ash
csh	The C shell, a symbolic link to tcsh.	/bin/csh
ksh	Public domain Korn shell. ksh is a command interpreter that is intended for both interactive and shell script use. Its command language is a superset of the sh shell language.	/bin/ksh /usr/bin/ksh
pdksh	A symbolic link to ksh	/usr/bin/pdksh
rsh	A restricted shell intended for network operation.	/usr/bin/rsh
sh	A symbolic link to bash	/bin/sh
tcsh	A C shell with file name completion and command line editing, tcsh is an enhanced but completely compatible version of the Berkeley UNIX C shell, csh. It is a command language interpreter usable both as an interactive login shell and a shell script command processor.	/bin/tcsh
zsh	Zsh is a UNIX command interpreter (shell) usable as an interactive login shell and as a shell script command processor. Of the standard shells, zsh most closely resembles ksh but includes many enhancements.	/bin/zsh

With all but rsh you can try out a new shell just by typing its absolute command as given in the table above. This is illustrated below:

```
[clive@redhat clive]$ /bin/zsh
redhat% exit
[clive@redhat clive]$
```

In the above example we have tried out the zsh (note the change in command prompt) and then used **exit** to return to bash.

If you wish to change you login shell permanently you can use the chsh command.

Change your Login Shell - chsh

The command chsh is used to change your login shell as defined in /etc/passwd- until you you change it again using chsh. If a shell is not given on the command line, chsh prompts for one. The root can change any users shell, but a user can only change their own shell.

The use of chsh to change a users shell is illustrated below:

```
[clive@redhat clive]$ chsh
Changing shell for clive.
Password:
New shell [/bin/bash]: /bin/zsh
Shell changed.
[clive@redhat clive]$
```

After changing your shell it is necessary to log out and log back in again before the change will take effect.

Finding out Information about a Shell

The man pages on a particular shell are very useful. For instance, the man pages on bash give full information on the command line and shell programming, including the usage and settings of both user and environment variables.

Work Effectively on the Command Line

The Command Line

Once you have logged in the command prompt is displayed. With bash the default prompt shows the name of the logged on user, the machine name and the current working directory. This is illustrated below:

```
[clive@redhat /home]$
```

Where **clive** is the name of the logged in user, **redhat** is the name of the computer and **/home** is the current working directory. The **$** prompt appears when you are logged on as a user and is replaced by a **#** when you are logged in as root as shown below:

```
[root@ext7144 /root]#
```

The Linux operating system monitors what you type and it is only interpreted by the shell after you press the enter key.

You can enter multiple commands on the same line by separating them by semicolons. The syntax for this shown below:

```
[clive@redhat clive]$ ls ; ls -i
```

The shell reads the command line as a whole, and then breaks it down into words. A word is a series of non-blank characters separated by either spaces or tabs. The shell considers the first word to be a command, regardless of whether it has a relative or absolute name. The shell does not interpret any options associated with a command, it simply passes these options on to the command, which is itself responsible for any error messages.

Editing the Command Line

The command line editor is the **Readline Library** developed by the FSF (Free Software Foundation). By default the emacs mode is used. For full information on emacs command line editing you are referred to the **info emacs** page.
You can change from the Readline Library to another command line editor if you wish to using the **set** command.

Work Effectively on the Command Line

The use of the set command to change to the vi command line editor and then back to emacs is illustrated below:

```
[clive@redhat clive]$ set -o vi
[clive@redhat clive]$ set -o emacs
```

By default the file **/etc/inputrc** contains the configuration settings for the command line editor. You can create your own custom settings, including custom key usage (bindings), by creating and editing the **.inputrc** file in your home directory and setting the **INPUTRC** environment variable to point to this directory.

Using the **bind** command you can view your key bindings. A few lines of the resulting output are illustrated below:

```
[clive@redhat /etc]$ bind -v | less
abort can be found on "\C-g", "\C-x\C-g", "\e\C-g".
accept-line can be found on "\C-j", "\C-m".
```

Command Line Completion

The bash shell incorporates command line completion. If, when part way through typing the first word on the command line, you press the tab (or Esc) key, the command will be automatically completed for you, providing there is only one match. If there are multiple matches the PC beeps and pressing tab (or Esc) again will display a list of possible options to choose from. This is illustrated below:

```
[clive@redhat /etc]$ ca
cachegen  cal  callbootd  captoinfo  case  cat  catchsegv
```

Filenames in the working directory are also automatically completed on the same basis:

```
clive@redhat /etc]$ cat c
charsets         conf.modules    cron.daily      cron.weekly
codepages        core            cron.hourly     crontab
conf.linuxconf   cron.d          cron.monthly    csh.cshrc
```

Work Effectively on the Command Line

Environment Variables

Every shell stores pieces of information that it needs to use in what are called environment variables. These parameters are stored in various configuration files located in either **/etc** or in your home directory. The default environment variables for bash are located in **/etc/profile**.

You can display a selection of your environment variables using either the **printenv** or the **env** commands as illustrated below:

```
clive@redhat clive]$ env
USERNAME=
COLORTERM=gnome-terminal
BROWSER=/usr/bin/netscape
HISTSIZE=1000
HOSTNAME=redhat.bromley.ac.uk
LOGNAME=clive
HISTFILESIZE=1000
MAIL=/var/spool/mail/clive
TERM=xterm
HOSTTYPE=i386
PATH=/usr/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/hom
e/clive/bin
KDEDIR=/usr
HOME=/home/clive
INPUTRC=/etc/inputrc
SHELL=/bin/bash
USER=clive
BASH_ENV=/home/clive/.bashrc
DISPLAY=:0
SESSION_MANAGER=local/redhat.bromley.ac.uk:/tmp/.ICE-
unix/7601,tcp/redhat.bromley.ac.uk:1118
OSTYPE=Linux
WINDOWID=58720261
SHLVL=4 _=/usr/bin/env
```

When a shell like bash starts up, it examines its process environment and creates an internal local variable for every environment variable found.

The **set** command without a parameter will show you a list of all the shell internal variables, including any user variable definitions.

Work Effectively on the Command Line

The use of the set command is illustrated below:

```
clive@redhat clive]$ set
BASH=/bin/bash
BASH_ENV=/home/clive/.bashrc
BASH_VERSION=1.14.7(1)
BROWSER=/usr/bin/netscape
COLORTERM=gnome-terminal
COLUMNS=80
DISPLAY=:0
EUID=500
HISTFILE=/home/clive/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/clive
HOSTNAME=redhat.bromley.ac.uk
HOSTTYPE=i386
IFS=
INPUTRC=/etc/inputrc
KDEDIR=/usr
LINES=25
LOGNAME=clive
MAIL=/var/spool/mail/clive
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=Linux
PATH=/usr/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/hom
e/clive/bin
PPID=16787
PS1=[\u@\h \W]\$
PS2=>
PS4=+
PWD=/home/clive
SESSION_MANAGER=local/redhat.bromley.ac.uk:/tmp/.ICE-
unix/16628,tcp/redhat.bromley.ac.uk:1193
SHELL=/bin/bash
SHLVL=4
TERM=xterm
UID=500
USER=clive
USERNAME=
WINDOWID=50331653
```

Work Effectively on the Command Line

To see the current value of a variable you can echo its value to the screen. This is illustrated below:

```
[clive@redhat clive]$ echo $PATH
/usr/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/cli
ve/bin
```

PATH

The PATH variable is one of the most important environment variables. When you type in the name of an executable file such as a command, application, or shell-script, the operating system looks at the directories specified in the path to try and find the file. If the file is found it is then executed.

If the file is not found a message to that effect is output. This is illustrated below:

```
clive@redhat clive]$ lpc
bash: lpc: command not found
[clive@redhat clive]$ whereis lpc
lpc: /usr/sbin/lpc /usr/man/man8/lpc.8
[clive@redhat clive]$ echo $PATH
/usr/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/cli
ve/bin
```

In the example above the lpc command is not located in a directory which is part of the user's path and is therefore not found.

To add another directory to your path you can change the value assigned to the environment variable as illustrated below:

```
[clive@redhat clive]$ PATH=$PATH:/usr/sbin;export PATH
[clive@redhat clive]$ echo $PATH
/usr/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/cli
ve/bin:/usr/sbin
[clive@redhat clive]$ lpc
lpc> q
[clive@redhat clive]$
```

In the above example the directory **/usr/sbin** has been added to the user's path and the command lpc is now found.

These changes to the $PATH variable are temporary. As soon as you close a terminal window, or exit the current shell, any changes you have made to the variables are lost.

To make these changes permanent you need to edit the file **.bash_profile** in your home directory. This is illustrated below:

```
[clive@redhat clive]$ less .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/usr/sbin
BASH_ENV=$HOME/.bashrc
USERNAME=""

export USERNAME BASH_ENV PATH
```

In the above example the directory **/usr/sbin** has been added to the path.

To make a change in path available to all users it is necessary to edit the **/etc/profile** file as root and change the line where PATH is defined to add the extra directory. This is illustrated below:

```
# System wide environment and startup programs
# Functions and aliases go in /etc/bashrc

PATH="$PATH:/usr/X11R6/bin:/usr/sbin"
PS1="[\u@\h \W]\\$"
```

In the above example the directory **/usr/sbin** has been added to the path for all users.

Work Effectively on the Command Line

To run an executable which is not in your path you need to provide the full relative or absolute path to the command.

This is illustrated below for the lpc command:

```
P [clive@redhat clive]$ ../../usr/sbin/lpc
lpc> q
[clive@redhat clive]$ /usr/sbin/lpc
lpc> q
```

If the command is in the working directory you can type:

```
[clive@redhat clive]$ ./install
```

In the above example the **.** is expanded to the absolute path of the working directory.

Alternatively, if the command is in your home directory you can type:

```
[clive@redhat clive]$ ~/install
```

In the above example the ~ is expanded to the absolute path of your home directory.

Note: It is generally more secure to move a command you frequently use to a directory which is in your path such as **$home/bin** than to add additional directories to your path.

Prompt - PS1

The prompt is determined by the value of the environment variable PS1. The default value for this variable for bash is shown below:

```
[clive@redhat clive]$ echo $PS1
[\u@\h \W]\$
[clive@redhat clive]$
```

PS1 has a different default value for different shells. For example in the zsh PS1 has the following value:

```
[clive@redhat clive]$ /bin/zsh
redhat% echo $PS1
%m%#
redhat% exit
```

The table overleaf explains a selection of the special characters used when setting the prompt variable.

A selection of BASH prompt characters is shown in the table below:

Character	*Resulting Output*
\d	The full date
\h	The hostname up to the first .
\H	The entire hostname
\s	The name of the shell
\t	The time in 24 hour format
\T	The time in 12 hour format
\@	The time in a.m. / p.m. format
\u	The current user
\W	The present working directory
\$	Displays # if root, otherwise $

You can easily customise the prompt as is illustrated below:

```
[clive@redhat clive]$ PS1="\d: \s " ; export PS1
Fri Jul 28: bash
```

Obviously these changes are lost when you exit the shell. To make the changes permanent you can either edit your **.bash_profile** file and add the line **PS1="\d: \s"**.

If you wish to change the prompt for all users you will need to edit **/etc/profile** as root. (see page 9 above).

Work Effectively on the Command Line

History

Recently used commands are remembered for each user by the shell.

You can scroll through the command history by using the up and down arrow keys on the keyboard. Using your history well can save a lot of typing and speed up your use of the system.

Three environment variables determine how much history is kept and where it is kept. These are:

HISTSIZE -The number of commands to remember in the command history. The default value is 500.

HISTFILE -The name of the file in which command history is saved. The default value is .bash_history in the users home directory. If unset, the command history is not saved when an interactive shell exits.

HISTFILESIZE -The maximum number of lines contained in the history file. When this variable is assigned a value, the history file is truncated, if necessary, at logout, to contain no more than that number of lines. The default value is 500. Thus this variable determines the number of history events saved for the next login session.

Typical values for history variables as found in the file /etc/profile are illustrated below:

```
HISTSIZE=1000
HISTFILESIZE=1000
```

In this case the history is 1000 events and all these events would be available at the next login.

Each event in the history is given a reference number. You can review your history by using the **history** command.

The use of this command is illustrated below:

```
[clive@redhat clive]$ history | less
   11  joe crun
   12  crun ctrail.c
   13  joe crun
   11  joe crun
   12  crun ctrail.c
   13  joe crun
   14  crun ctrail.c
```

To save piping the history through less you can also tell the history command how many events you wish to view as illustrated below:

```
[clive@redhat clive]$ history 3
 1009  less /etc/profile
 1010  history | less
 1011  history 3
```

In the above example just the 3 most recent items of history have been displayed.

Editing the History File - fc

The fc command is used to display or edit selected events or ranges of events in your history file and rerun commands. The value of the environmental variable **FCEDIT** determines the editor used. If no value is set then the command defaults to the **EDITOR** environmental variable and if that is not set the editor defaults to vi.

The use of fc to list ten lines from the history file is illustrated below:

```
[clive@redhat clive]$ fc -l 20 30
20         history | less
21         history 3
22         man fc
23         man bash
24         echo $HISTFILESIZE
25         HISTFILESIZE=10
26         echo $HISTFILESIZE
```

```
27          history
28          exit
29          exit
30          history
```

Without the **-l** switch in the above example lines 20 to 30 in the history file would be opened into the editor.

To re-execute a specific command in the history list fc can be used with the -s switch as illustrated below:

```
[clive@redhat clive]$ fc -s  22
```

User Variables

Environment variables are already defined. You can define your own variables, and these are called user variables. Although user variables are often used in shell scripts they can also be used from the command prompt. The variable exists as long as the shell in which it was created exists. As soon as you close a terminal window, or exit the current shell, these temporary user variable definitions are lost.

A couple of examples of using user variables are illustrated below:

```
[clive@redhat clive]$ fred="Hello"
[clive@redhat clive]$ echo $fred
Hello

[clive@redhat clive]$ r="Recent"
[clive@redhat clive]$ ls $r
8051 Instruction Set Summary.doc.lnk
C&N1 WRITTEN FINAL 2000 marking scheme.doc.lnk
```

In the first example a user variable **fred** is created and assigned a string value **Hello**. The echo **$fred** command then echoes the contents of the variable to the screen.

In the second example a user variable **r** is created and assigned a string value **Recent**. The **ls $r** command then produces a listing of the contents of the Recent subdirectory.

Work Effectively on the Command Line

You can export user variables if you wish them to be available to child processes. This is illustrated below:

```
[clive@redhat clive]$ FRED="BLOGGS"
[clive@redhat clive]$ env | grep FRED
[clive@redhat clive]$ set | grep FRED
FRED=BLOGGS
_=FRED
[clive@redhat clive]$ export FRED
[clive@redhat clive]$ env | grep FRED
FRED=BLOGGS
[clive@redhat clive]$ set | grep FRED
FRED=BLOGGS
_=FRED
```

Aliases

Aliases are user variables that can be used to create custom commands. You can also include arguments in an alias.

An example of using an alias is illustrated below:

```
[clive@redhat clive]$ alias d="ls -la"
[clive@redhat clive]$ d
total 915
-rw-------   1 root     root           17 Oct 27  1999 #h#
drwx------  22 clive    clive        1024 Jul 27 09:49 .
drwxr-xr-x  29 root     root         2048 Jul 26 09:18 ..
-rw-------   1 clive    clive        1020 Jul 27 09:16 .
ICEauthority
```

In the above example an alias **d** created with the value **ls -la** assigned to it. When d is entered on the next command line the full directory listing of the current working directory is displayed. Note that this custom command will be lost when the current shell is closed.

Many Linux applications, including the command shell itself, read a configuration file at login. It is traditional for these start up files to be stored in the users home directory. When a user logs on, the bash command shell looks for a file called **.bashrc** in the users home directory. (The adduser command automatically copies this script into the users home directory).

Work Effectively on the Command Line

You can easily edit this file using your favourite editor to create custom commands which are available every time you log on. In the example below a custom command **d** has been set up to save typing **ls -la ¦ less** every time:

```
# .bashrc
# User specific aliases and functions
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
alias d="ls -la ¦ less"
```

Command Substitution

With command substitution you can have the output of a command interpreted by the shell instead of by the command itself. This is very useful in scripting as it allows you to use the output of one command as in input to another.

Command substitution is illustrated below:

```
[clive@redhat clive]$ echo The current user is $(whoami)
The current user is clive
```

In the above example the string "The current user is" is echoed to the screen followed by the output of the whoami command.

A further example is illustrated below:

```
[clive@redhat clive]$ echo The current date and time is
$(date)
The current date and time is Fri Jul 28 14:23:59 BST 2000
```