LPIC-1 Study Group
1 Command Line Tools

R. Scott Granneman
scott@granneman.com
www.granneman.com

1

This presentation
is based on
Roderick W. Smith's
LPIC-1: Linux Professional Institute Certification Study Guide,
2nd edition

That said,
there are many
additions, subtractions, & changes

2

Introduction

3

Command Line Basics

4

What's a *shell*?

5

Common shells

csh (C shell)

tcsh

bsh (Bourne shell)

bash (GNU Bourne Again Shell)

ksh (Korn shell)

zsh (Z shell)

tcsh = TENEX csh

ksh = csh + bsh

6

You can always point to a shell & make sure one is called

```
$ ls -l /bin/sh
/bin/sh -> bash*
```

7

Ubuntu is different

```
$ ls -l /bin/sh
/bin/sh -> dash*
```

Why?

wiki.ubuntu.com/DashAsBinSh

8

Starting a shell

9

If you log in
using a text-mode login screen,
you're dropped into a shell

If you log in via a GUI,
you start a shell manually
(e.g., xterm, Konsole, Terminal)

10

login vs *non-login* shell

When you log in or open a shell,
it's a *login* shell

When you invoke a shell
from within a login shell,
or run a shell script,
a *sub-shell* or *non-login shell* opens

The type of shell determines
what files are sourced
for commands & info

11

2 environment variables

`$SHELL`

The login shell

`$0`

The current shell

12

```
$ echo $SHELL
/bin/bash
$ echo $0
/usr/local/bin/bash
$ ksh
$ echo $SHELL
/bin/bash
$ echo $0
ksh
$ exit
$ echo $0
/usr/local/bin/bash
```

13

Another way to invoke a sub-shell

`(foo)`

14

```
$ pwd
/home/scott
$ (cd /tmp && ls)
launch-H89RmI/  launch-UFdOL0/
[…]
$ pwd
/home/scott
```

15

Files are sourced
based on the shell type

16

Login shells

On login

1. If `/etc/profile` exists, source it
2. If `~/.bash_profile` exists, source it,
else if `~/.bash_login` exists, source it,
else if `~/.profile` exists, source it

On exit

If `~/.bash_logout` exists, source it

17

Non-login interactive shells

On startup

If `~/.bashrc` exists, source it

18

Internal & External Commands

19

Shell commands are
internal (built in)
or
external (not built in)

20

Common internal commands

`cd`

`pwd`

`echo`

`exec`

`time`

`set`

`exit/logout`

21

cd

Change working directory

```
cd /var/www/www.granneman.com
cd
cd ~
cd ~/bin
cd ..
```

22

pwd

Print working directory

```
$ pwd
/var/www
$ cd ..
$ pwd
/var
```

23

echo

Display entered text

```
$ echo Hello
Hello
$ echo "Hello World"
Hello World
```

24

```
$ echo $PATH
/home/scott/bin:/usr/local/
sbin:/usr/local/bin:/usr/sbin:/
usr/bin:/sbin:/bin

$ backup_dir="~/backup"
$ echo $backup_dir
~/backup
```

25

`exec`

Execute a program
& then exit the shell

```
$ exec /home/scott/bin/backup.sh
```

26

`time`

Time how long commands
take to execute

```
$ time find . -name config
./www.fvfpd.com/config
./www.foobar.com/config
[…]
real    0m2.519s
user    0m0.520s
sys     0m1.712s
```

27

Total execution time
User CPU time
System CPU time

`set`

Display & set options for bash

Not the same thing as environment variables

28

```
$ set
BASH=/usr/local/bin/bash
BASHOPTS=cdspell:checkwinsize:cm
dhist:expand_aliases:extglob:ext
quote:force_fignore:histappend:i
nteractive_comments:progcomp:pro
mptvars:sourcepath
BASH_COMPLETION=/usr/local/etc/
bash_completion
BASH_VERSION='4.2.37(2)-release'
COLUMNS=100
```

29

```
EDITOR=/usr/bin/vim
HISTFILE=/Users/
scott/.bash_history
HISTFILESIZE=10000
HISTIGNORE='&:l[als]:[bf]g:exit'
HOME=/Users/scott
IFS=$' \t\n'
LANG=en_US.UTF-8
PATH=/usr/local/bin:/usr/bin:/
bin:/usr/sbin:/sbin:/usr/local/
sbin:/Users/scott/bin
```

30

```
PS1='\n\[\033[01;32m\]\u@\h\
[\033[00m\]:\[\033[01;34m\]\w\
[\033[00m\]\$ '
PWD=/Users/scott
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hash
all:histexpand:history:interacti
ve-comments:monitor
TERM=linux
TERM_PROGRAM=iTerm.app
UID=501
USER=scott
```

31

`exit`

Terminates any shell

`logout`

Terminates login shells

32

Internal commands
may be duplicated by
external commands

Internal commands take precedence
unless you specify
the complete path
to the external command

33

Good example of the problems this can cause in the book, in "Confusion over Internal and External Commands"

When you type a command
that's not internal,
the shell checks it's *path*
to look for the program

PATH environment variable
defines the list of directories
in which to look

34

```
echo $PATH
/usr/local/bin:/usr/bin:/bin:/
usr/sbin:/sbin:/Users/scott/bin/
ec2-api-tools/bin:/usr/local/
sbin:/Users/scott/bin
```

```
echo $PATH
/root/bin:/root/perl5/bin:/usr/
local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/
games:/var/lib/gems/1.8/bin/
```

35

Run commands & programs
not in your path
by providing a complete path

```
$ pwd
/home/scott/temp
$ ls
foo.sh
$ ./foo.sh
OR
$ /home/scott/temp/foo.sh
```

36

Never place . in your PATH!

37

Shell Command Tricks

38

Command completion

```
$ pwd
$ /home/scott/bin
$ ls
backup_daily.sh  backup_mysql.sh
dotfiles.sh
$ ./d[tab]otfiles.sh
$ ./b[tab]ackup_[tab][tab]
backup_daily.sh  backup_mysql.sh
$ ./backup_|m[tab]ysql.sh
```

39

Command (& file name) completion is great

Saves time

Avoids typos

40

`history`

Record of every typed command in `~/.bash_history`

`$ history`

`$ ↑` (or Ctrl-P)

`$ ↓` (or Ctrl-N)

41

`$ history 20`

Shows last 20 lines

`$ history -c`

Clears history

42

Ctrl-R

Backward (reverse) search of history

Ctrl-S

Forward search of history

Ctrl-G

Terminate search

43

```
$ history | grep foo
```

44

Create or edit ~/.inputrc

```
# when pressing up or down
arrows, show only history
entries
# that match what was already
typed
"\e[A":history-search-backward
"\e[B":history-search-forward
```

45

After editing ~/.inputrc, reload with: bind -f ~/.inputrc

Editing

46

Ctrl-A
Move cursor to start of line

Ctrl-E
Move cursor to end of line

47

Ctrl-B or ←
Move back one character

Ctrl-F or →
Move forward one character

48

Ctrl-← or Esc-B

Move backward one word

Ctrl-→ or Esc-F

Move forward one word

49

Ctrl-D or Delete

Delete character under the cursor

Backspace

Delete key to the left of the cursor

50

Ctrl-K

Delete all text
from cursor to end of line

Ctrl-U

Delete all text
from cursor to beginning of line

Ctrl-E Ctrl-U

Delete all text on line

51

Ctrl-T

Transpose characters
before & on the cursor

Esc-T

Transpose two words
before or on the cursor

52

Esc-U

Convert text
from cursor to end of word
to ALLCAPS

Esc-L

Convert text to lowercase

Esc-C

Convert text to Uppercase

53

Ctrl-X Ctrl-E

Launch editor to edit command

Editor defined by

1. `$FCEDIT` environment variable
2. `$EDITOR` environment variable
3. emacs

54

Shell Configuration

55

User configuration files

`~/.bashrc`

`~/.profile`

Global configuration files

`/etc/bash.bashrc`

`/etc/profile`

56

Back them up before changing them!

57

Environment Variables

58

Like any variable:
they hold data to be referred to using the variable's name

Programs, including bash, rely on environment variables

59

To assign an environment variable

Edit `~/.bashrc`

```
export EC2_HOME=~/bin/ec2-api-tools
export PERL_LOCAL_LIB_ROOT="/root/perl5"
```

60

Getting Help

61

`man foo`

Display help for foo

62

Moving around in man pages

Spacebar or `f`
Go forward one screen

Esc-V or `b`
Go back one screen

63

↓

One line down

↑

One line up

64

`/foo`

Search for text

`n`

Search forward

`N`

Search backward

`q`

Exit man

65

Man pages categorized in sections

1–9

Most common is 1

Some commands are in more than one section

`man 5 passwd`

66

See Table 1–1 for list & meanings

`info foo`

Display help for foo
using `info` pages
instead of `man`

67

Streams, Redirection, Pipes

68

Streams

69

Input & output treated as *streams*:
data that can be manipulated

70

Standard input (*stdin*)
Normally keyboard input

Standard output (*stdout*)
Normally displayed on screen

Standard error (*stderr*)
2nd kind of output stream
intended for high priority errors
normally sent to same screen
as stdout

0 = stdin
1 = stdout
2 = stderr

71

Redirection

72

Send stdin, stdout, & stderr
to different places

```
$ echo "Hello world"
Hello world
$ echo "Hello world" > hello.txt
Hello world
$ cat hello.txt
Hello world
```

73

>

stdout goes to file,
overwriting if it already exists

>>

stdout appended to existing file,
creating it if it doesn't exist

74

2 is stderr, remember?

2>

stderr goes to file,
overwriting if it already exists

2>>

stderr appended to existing file,
creating it if it doesn't exist

75

&>

stdout & stderr goes to file,
overwriting if it already exists

76

<

stdin comes from file

<<

stdin comes from next several lines
(AKA a *here document*)

77

```
$ date +'%d %m %Y'
19 08 2012
$ read day month year <<< $(date
+'%d %m %Y')
$ echo $day
19
$ echo $month
08
$ echo $year
2012
```

78

<>

stdin & stdout
comes from, & goes to, a file

79

tee

stdout goes to stdout *and* to file,
so you can see output & store it

```
$ foo | tee output.txt
```

80

Pipes

81

Pipe redirects 1st program's output to 2nd program's input

```
$ foo | bar
$ history | grep foo
$ env | grep JAVA
$ ps aux | grep ikiwiki
$ foo | bar | abc | xyz
```

82

xargs

Used to
build & execute command lines
from stdin

83

```
$ rm /path/*
```

OR

```
$ find . -name foo -exec rm {} \;
```

might fail with
`Argument list too long`

Instead, use xargs

```
$ xargs rm /path/*
$ find . -name foo | xargs -0 rm
```

84

Command substitution

Run command first,
& then use results
as arguments to another command

Use `foo` or $(foo)

85

Last line of "Generating Command Lines" in book is wrong! Uses ' instead of `

```
zip -r /tmp/scripts_`date +%Y%m
%d`.zip /var/shared_assets/scripts
```

is the same thing as

```
zip -r /tmp/scripts_$(date +%Y%m
%d).zip /var/shared_assets/scripts
```

but,
it's better
to use `$(foo)` instead of `` `foo` ``

86

Better to use
`$(foo)` instead of `` `foo` ``

Why?

```
$ for directory in $(find $(pwd)
-type d -mindepth 1 -print)
do
  cd "$directory"
done
```

Nesting is a nightmare with ``

87

Filters

88

Program that processes data stream

89

Viewing

90

`head`
View the beginning of a file

`tail`
View the end of a file

`less`
Read a file a screen at a time

91

`head`
View the beginning of a file

92

`$ head foo.txt`
Display 1st 10 lines

`$ head -n 100` (or `--lines=100`)
Display 1st 100 lines

`$ head -c 10` (or `--bytes=10`)
Display 1st 10 bytes

93

`tail`
View the end of a file

94

`$ tail foo.txt`
Display last 10 lines

`$ tail -n 100` (or `--lines=100`)
Display last 100 lines

`$ tail -c 10` (or `--bytes=10`)
Display last 10 bytes

95

`$ tail -f foo.log` (or `--follow`)
Display last 10 lines,
but keeping file open
& displaying new lines
as they are added to the file

Very handy for log files

`$ tail -f foo.log --pid=10011`)
Display last 10 lines,
but stop when process terminates

96

PID = Process ID

`less`

Read a file a screen at a time;
AKA, *page* through a file

Joke because `more`
was an early pager,
& `less` is more than `more`!

97

`less foo.txt`

Can't really use `less` in a pipe,
except at the very end,
since it takes over the whole screen

98

`f` or spacebar	Forward a page
`b` or `Esc-v`	Back a page
`↓`	Down 1 line
`↑`	Up 1 line
`h`	Help
`/foo`	Search for foo
`n`	Repeat search forward
`N`	Repeat search backward
`g25`	Go to line 25
`q`	Quit

99

Formatting

100

`fmt`

Reformat text files

`nl`

Number lines

`pr`

Format file suitable for printing

101

`fmt`

Reformat text files
so that long lines
wrap at 75 characters

```
fmt foo.txt
```

102

Change default width

```
fmt -65 foo.txt
fmt -w 65 foo.txt
fmt --width=65 foo.txt
```

103

`nl`

Number lines

104

```
$ cat a.txt
Barsoom
Pellucidar
Venus

Caspak
The Moon
Torn
```

```
$ nl a.txt
1 Barsoom
2 Pellucidar
3 Venus

4 Caspak
5 The Moon
6 Torn
```

105

`nl` recognizes 3 parts of a page:

body (`-b`)
header (`-h`)
footer (`-f`)

Number all lines, including blanks, in the body

```
nl -b a foo.txt
```

106

`pr`
Format file suitable for printing

```
pr foo.txt
```

Remember!

`pr` does not actually print

It *prepares* files for printing

```
pr foo.txt | lpr
```

107

Creates:

Headers
Footers
Page breaks

108

Change header
from file name to *string*
`-h "lorem ipsum"`
`--header="lorem ipsum"`

Omit header
`-t` or `--omit-header`

109

Options include:

Multi-column output

Double-spaced output

Form-feed characters between pages

Set page length in lines

Set left margin

Set page width

110

Combining

111

`cat`

Combines files end-to-end

`join`

Combines files based on fields

`paste`

Combines files line-by-line

112

`cat`

Con*cat*enate (join) 1 or more files

113

```
$ cat a.txt
I'm the contents of a.txt
$ cat b.txt
I'm the contents of b.txt
$ cat a.txt b.txt
I'm the contents of a.txt
I'm the contents of b.txt
```

114

```
$ cat a.txt b.txt > ab.txt
$ cat ab.txt
I'm the contents of a.txt
I'm the contents of b.txt
```

115

`$ cat -E` (or `--show-ends`)
Show line endings

`$ cat -n` (or `--number`)
Number lines

`$ cat -b` (or `--number-nonblank`)
Number only lines with text

116

`$ cat -s` (or `--squeeze-blank`)
Combines multiple blank lines to 1

`$ cat -T (or --show-tabs)`
Display tabs as `^I`

`$ cat -v` (or `--show-nonprinting`)
Display special characters

117

`tac`

Reverse `cat`!

118

`join`

Combine 2 files
by matching key field

119

```
$ cat a.txt
1234 Scott
1235 Finny
$ cat b.txt
1234 780-0489
1235 555-1212
$ join b.txt a.txt
1234 780-0489 Scott
1235 555-1212 Finny
```

120

`paste`

Combine 2 files line-by-line, separating lines from each file by tabs

Use `join` if files have key fields; use `paste` if files lack key fields

Ideally, files must have same number of lines to work correctly

121

Kinda silly:

```
$ cat a.txt
1234 Scott
1235 Finny
$ cat b.txt
1234 780-0489
1235 555-1212
$ paste b.txt a.txt
1234 780-0489[tab]1234 Scott
1235 555-1212[tab]1235 Finny
```

122

Makes more sense:

```
$ cat a.txt
Scott
Finny
$ cat b.txt
1234 780-0489
1235 555-1212
$ paste b.txt a.txt
1234 780-0489[tab]Scott
1235 555-1212[tab]Finny
```

123

Summarizing

124

`wc`

Word count
(& lines, bytes, & characters)

`cut`

Extract portions of text to stdout

125

`wc`

*W*ord *c*ount
(& lines, words, bytes, & characters)

`-l` or `--lines`

`-w` or `--words`

`-c` or `--bytes`

`-m` or `--chars`

126

```
$ cd /var/log/apache2
$ ls -lh access.log
… 1.6G … access.log
$ wc -l access.log
5547689 access.log
$ wc -w access.log
132190171 access.log
$ wc -c access.log
1612896169 access.log
$ wc -m access.log
1612896169 access.log
$ wc access.log
5547689 132190171 1612896169 access.log
```

-l = lines, -w = words, -c = bytes, -m = characters

wc = lines words bytes

127

cut

Extract portions of text to stdout

128

Specify what to cut

`-b` *list* or `--bytes=`*list*

`-c` *list* or `--characters=`*list*

`-f` *list* or `--fields=`*list*

List

4 or 2-4 or -4 or 4-

--bytes & --characters usually produce the same results, but not always

129

`-f` assumes tabs as delimiters

Specify different delimiters

```
-d char
--delim=char
--delimiter=char
```

```
cut -f 3 -d ":" foo.txt
cut -f 5 -d " " bar.txt
```

130

```
$ head 2 access.log
www.d20srd.org:80 24.236.164.48 - -
[19/Aug/2012:06:54:06 -0500] "GET /
srd/spellLists/spellLists.htm HTTP/
1.1" …
www.stlzoo.org:80 97.87.98.113 - -
[19/Aug/2012:06:54:06 -0500] "GET /
themes/stlzoo/style/header_logo.png
HTTP/1.1" …
…
$ cut -f 1 -d ":" access.log
www.d20srd.org
www.stlzoo.org
…
```

131

Transforming

132

Change files & send results to stdout

133

`expand`
Convert tabs to spaces

`unexpand`
Convert spaces to tabs

`sort`
Sort contents

`uniq`
Remove duplicate lines

134

`split`
Break a file into pieces

`tr`
*Tr*anslate characters
from stdin to stdout

`od`
*O*ctal *d*ump of files
(actually, octal, decimal, hex, ASCII)

135

`expand`

Tabs → spaces

Assumes a tab stop every 8 characters

Change this spacing with `-t [num]` or `--tabs=[num]`

136

`unexpand`

Spaces → tabs

Assumes a tab stop every 8 characters

Change this spacing with `-t [num]` or `--tabs=[num]`

137

`sort`

Sort contents

138

```
$ cat a.txt     $ sort a.txt
A               1
2               11
20              2
B               20
11              A
b               B
1               a
a               b
```

139

sort	-r	-f	-n	-rn
	Reverse	Ignore case	Numeric	
1	b	1	A	20
11	a	11	B	11
2	B	2	a	2
20	A	20	b	1
A	20	A	1	b
B	2	a	2	a
a	11	B	11	B
b	1	b	20	A

140

cat	sort	-M
		Month sort
MAR 3	APR 8	APR 8
DEC 12	AUG 22	AUG 22
JAN 20	DEC 12	DEC 12
APR 8	FEB 1	FEB 1
JUN 10	JAN 20	JAN 20
NOV 29	JUN 10	JUN 10
FEB 1	MAR 3	MAR 3
AUG 22	NOV 29	NOV 29

141

cat	sort	-k 2	-k 2 -n
		Sort field	Sort field & numeric
MAR 3	APR 8	FEB 1	FEB 1
DEC 12	AUG 22	JUN 10	MAR 3
JAN 20	DEC 12	DEC 12	APR 8
APR 8	FEB 1	JAN 20	JUN 10
JUN 10	JAN 20	AUG 22	DEC 12
NOV 29	JUN 10	NOV 29	JAN 20
FEB 1	MAR 3	MAR 3	AUG 22
AUG 22	NOV 29	APR 8	NOV 29

142

`uniq`

Remove duplicate lines
(*unique,* get it?)

143

Often used in conjunction with `sort`

```
sort a.txt | uniq
```

Given a text file containing
“Ask not what your country
can do for you;
ask what you can do
for your country.”

No punctuation

Each word on a separate line

144

```
$ sort          $ sort a.txt | uniq
a.txt           ask
ask             can
ask             country
can             do
can             for
country         not
country         what
do              you
do              your
for
for
…
```

145

`split`

Break a file into pieces,
all with similar prefixes

146

Split by…

`-b 100` or `-b 100k` or `-b 100m`

Bytes or KB or MB

`-l 100`

Number of lines

`-p pattern`

Regular expression

147

`-a 2`

Suffix padding length

2 starts with `aa` & goes to `zz`

4 starts with `aaaa` & goes to `zzzz`

etc.

148

If you don't specify
a filename to use,
`split` uses `x`

149

```
$ cat tweets.txt
---
Tweet #1
---
Tweet #2
---
$ split -a 2 -p --- tweets.txt
$ ls
xaa xab xac … xaz xba xbb …
$ cat xaa
---
Tweet#1
```

x is the default, unspecified filename

150

You can specify a file name
for `split` to use
at the end of the command

151

```
$ cat tweets.txt
---
Tweet #1
---
Tweet #2
---
$ split -a 2 -p --- tweets.txt twit
$ ls
twitaa twitab twitac …
twitaz twitba twitbb …
$ cat twitaa
---
Tweet#1
```

You can specify a filename, in this case "new"

152

`tr`

*Tr*anslate characters
from stdin to stdout

`tr set1 set2`

153

```
$ echo test | tr t T
TesT
$ cat a.txt
the sun
the moon
$ cat a.txt | tr t T
The sun
The moon
$ cat a.txt | tr t T > b.txt
$ cat b.txt
The sun
The moon
```

154

```
$ cat a.txt
the sun
the moon
$ tr t T < a.txt
The sun
The moon
$ tr t T < a.txt > b.txt
$ cat b.txt
The sun
The moon
```

155

`-d` deletes the pattern

```
$ cat a.txt
* foo
* bar
* foobar
$ tr -d "* " < a.txt
foo
bar
foobar
```

156

Shortcuts

`[:alnum:]`
All numbers & letters

`[:upper:]`
All uppercase letters

`[:lower:]`
All lowercase letters

`[:digit:]`
All digits

157

Ranges

```
a-m
3-8
```

158

od

*O*ctal *d*ump of files
(actually, octal, decimal, hex, ASCII)

Good for viewing codes & strings
in binary or text files

159

```
$ cat a.txt
Call me Ishmael.
$ od a.txt
0000000 060503 066154 066440 020145↩
        071511 066550 062541 027154
0000020 000012
0000021
$ od -t a a.txt
0000000 C a l l sp m e sp I s h m a↩
        e l .
0000020 nl
0000021
```

od prints octal (base 8) by default
-t a prints ASCII

All Together Now

The problem:

I have an enormous log file listing all the hits received by 68 different websites

How can I find out which websites received the most hits?

```
$ head 2 access.log
www.d20srd.org:80 24.236.164.48 - -
[19/Aug/2012:06:54:06 -0500] "GET /
srd/spellLists/spellLists.htm HTTP/
1.1" …
www.stlzoo.org:80 97.87.98.113 - -
[19/Aug/2012:06:54:06 -0500] "GET /
themes/stlzoo/style/header_logo.png
HTTP/1.1" …
…
$ wc -l access.log
5547689 access.log
```

163

```
$ cut -f 1 -d ":" access.log > /
tmp/websites.txt
$ wc -l /tmp/websites.txt
5547689 /tmp/websites.txt
$ head 5 /tmp/websites.txt
www.stlzoo.org
www.d20srd.org
www.stlzoo.org
gilgamesh.websanity.com
gilgamesh.websanity.com
```

164

```
$ sort /tmp/websites.txt | uniq
-c | sort -n
      2 mockups.websanity.com
     17 staging.aclu-em.org
     39 training.websanity.com
      …
 141601 www.granneman.com
 240687 www.aclu-nj.org
 455348 neuro.wustl.edu
1525228 www.d20srd.org
2473495 www.stlzoo.org
```

165

I could also have done it this way:

```
$ cut -f 1 -d ":" access.log |
sort | uniq -c | sort -n
```

166

Regular Expressions

167

Understanding RegEx

168

grep

169

sed

170

Review

171

Thank you!

Email: scott@granneman.com
Web: www.granneman.com
Publications: www.granneman.com/pubs
Blog: ChainSawOnATireSwing.com
Twitter: scottgranneman

LPIC-1 Study Group
1 Command Line Tools

R. Scott Granneman
scott@granneman.com
www.granneman.com

Licensing of this work