datto

Bash: New tricks for an old Shell

Fred Mora - System Engineering, Datto

The Bash Shell

- The old Bourne Shell (/bin/sh) was released in 1977
- It was the Unix v. 7 standard shell
- The Bourne Again shell (bash) was released in 1989. Mostly compatible with sh.
- Bash is now a POSIX standard
- Bash runs on Windows 10 in the new Linux subsystem
- So it's here to stay!

- Bash is GPL'd

New features: better job control with coproc

- Bash 3: Start jobs in the background with &, use %n to kill them.

```
$ sleep 120 &
$ kill %1
[1]+  Terminated              sleep 120
```

- Bash4: Use coproc to communicate with background jobs.

- Env var COPROC[0] and COPROC[1] contains stdout and stderr for the job

```
$ coproc { echo -e "line 1\nline 2"; sleep 120 ; }
[1] 10676
$ while read -u ${COPROC[0]} line ; do echo $line; done
line 1
line 2
^C
$ kill $COPROC_PID
[1]+  Terminated              coproc COPROC { echo -e "line 1\nline 2"; sleep 120; }
$
```

New feature: The declare keyword

- Bash 3 could set the variable type using the much abused set keyword
- Bash 4 has the declare keyword to declare variable types
- Numeric values:

```
$ declare -i myint
$ myint=90
$ echo $myint
90
$ myint=ninety
$ echo $myint
0
```

- Case conversion:

```
$ declare -l lowcase
$ lowcase=CaMeL
$ echo $lowcase
camel
```

New feature: Associative array

- Bash 3 had arrays with numerical index
- Bash 4 allows strings as indices

```
$ declare -A capital
$ capital[USA]="Washington, D.C."
$ capital[France]=Paris
$ echo ${capital[USA]}
Washington, D.C.
$ echo ${capital[France]}
Paris
```

New feature: Ranges with braces

- In Bash 3, we could do ranges with the seq command. Example: Create one dir for each month.

```
$ year=2016
$ for month in $(seq 1 12); do
> mkdir -p budget/$year/$month
> done
```

- More fun with brace expansion:

- The same with Bash 4:

```
$ year=2016
$ for month in {1..12}; do
> mkdir -p budget/$year/$month
> done
```

```
$ echo {10..50..5}
10 15 20 25 30 35 40 45 50
$ echo {100..10..10}
100 90 80 70 60 50 40 30 20 10
$ echo {a..g}
a b c d e f g
$
```

New feature: Recursive globs with **

- In Bash 3, * matches any file or dir, but not recursively:

```
$ cd /home/fmora/music/howard_shore/
$ echo *
2001-the_lord_of_the_rings__the_fellowship_of_the_ring
2002-the_lord_of_the_rings__the_two_towers
2003-the_lord_of_the_rings_the_return_of_the_king
2007-the_return_of_the_king_the_complete_recordings
2012-the_hobbit_1
```

- In Bash 4, * matches any file or dir recursively:

```
$ cd /home/fmora/music
$ shopt -s globstar
$ ls -1 **/*obbit*ogg
howard_shore/2001-the_lord_of_the_rings__the_fellowship_of_the_ring/02_concerning_hobbits.ogg
howard_shore/2012-the_hobbit_1/an_unexpected_journet_disk_1/14-A_Very_Respectable_Hobbit.ogg
```

New feature: Negative indices count from the end

- Bash 3 has array indices from 0
- To count from the end, it needs expressions with the array length

```
$ array=( zero one two three four five )
$ n=${#array[*]}
$ last=$((n-1))
$ echo $last
5
$ echo ${array[$last]}
five
```

- Bash 4 has negative indices

```
$ array=( zero one two three four five )
$ echo ${array[-1]}
five
$ echo ${array[-2]}
four
```

"Old" feature: Line editing

- Classic problem: editing a typo in the previous command that failed.
- Up arrow retrieves the previous command, then:

•Ctl-A : Go to start of line
•Ctl-E: Go to end of line
•Esc+Backspace: delete previous word
•Esc+D: delete word under cursor
•Ctl-K: Kill to end of line
•Clt-U: Kill the whole line

•Ctl-Y: Yank the kill buffer
•Esc+.: Copy last word from previous command
•Tab: Complete a file name
•Ctl-T: Transpose two chars
•Ctl-_: Undo last change
•Esc+r: Revert line as it was in history.

Retrieving commands from history

- Ctl-R*foo* will retrieve the most recent command containing foo
- Keep hitting Ctl-R for more hits
- Esc+< or Esc+> to go to start or end of history
- Went too far? Ctl-S to search forward
- History file is in ~/.history
- Command history lists it
- Not very good when you have multiple shells, alas.

Questions?