Shell Scripting 2

Joseph Hallett

January 12, 2023

University of
BRISTOL

Last time

We introduced shell scripting as a tool for automating stuff

- ▶ Gave a basic overview of syntax
- ▶ Mentioned `env` and `shellcheck`

This time

- ▶ More syntax and control flow
- ▶ Variables and techniques
  As before I'll try and keep to POSIX shell and mark where things are Bashisms...
  - ▶ but some Bash-isms are useful to know

Variables

All programs have variables... Shell languages are no different:

To create a variable:

```
GREETING="Hello World!"
```

(No spaces around the =)

To use a variable

```
echo "${GREETING}"
```

If you want your variable to exist in the programs you start as an *environment variable*:

```
export GREETING
```

To get rid of a variable

```
unset GREETING
```

Well...

Variables in shell languages *tend* to act more like macro variables.

► There's no penalty for using one thats not defined.

```
NAME='Joe'
unset NAME
echo "Hello, '${NAME}'"
Hello, ''
```

If this bothers you:

```
set -o nounset
echo "${NAME:? variable 1 passed to program}"
```

(There are a *bunch* of these shell parameter expansion tricks beyond `:?` which can do search and replace, string length and various magic...)

Standard variables

${0} Name of the script

${1}, ${2}, ${3}... Arguments passed to your script

${#} The number of arguments passed to your script

${@} and ${*} All the arguments

Control flow

If statements and *for* loops, with *globbing*, are available:

```
# Or [ -x myscript.sh ];
# Or [[ -x myscript.sh ]]; if using Bash
if test -x myscript.sh; then
    ./myscript.sh
fi

for file in *.py; do
    python "${file}"
done
```

Other loops

Well...okay you only have `for` really... but you can do other things with it:

```
for n in 1 2 3 4 5; do
    echo -n "${n} "
done
1 2 3 4 5
```

```
seq 5
1 2 3 4 5
for n in $(seq 5); do
    echo -n "${n} "
done
1 2 3 4 5
```

```
seq -s, 5
1,2,3,4,5
# IFS = In Field Separator
IFS=','
for n in $(seq -s, 5); do
    echo -n "${n} "
done
1 2 3 4 5
```

Case statements too!

```
3  # Remove everything upto the last / from ${SHELL}
   case "${SHELL##*/}" in
       bash) echo "I'm using bash!" ;;
       zsh)  echo "Ooh fancy a zsh user!" ;;
       fish) echo "Something's fishy!" ;;
       *)    echo "Ooh something else!" ;;
   esac
```

Basename and Dirname

In the previous example I used the "${VAR##*/}" trick to remove everything up to the last /...
Which gives you the name of the file neatly...
...but I have to look this up everytime I use it.
Instead we can use $(basename "${shell}") to get the same info.

```
echo "${SHELL}"
echo "${SHELL##*/}"
echo "$(basename "${SHELL}")"
echo "$(dirname "${SHELL}")"
```

You can even use it to remove *file extensions*:

```
for f in *.jpg; do
  convert "${f}" "$(basename "${f}" .jpg).png"
done
```

Pipelines

As part of shell scripting, its often useful to build commands out of chains of other commands. For example I can use ps to list all the processes on my computer and `grep` to search.

▶ How many processes is *Firefox* using?

```
ps -A | grep -i firefox
```

43172	??	SpU	0:10.69	/usr/local/bin/firefox		
59551	??	Sp	0:00.06	/usr/local/lib/firefox/firefox	-contentproc	-appDir
7023	??	SpU	0:06.10	/usr/local/lib/firefox/firefox	-contentproc	{a032331
59478	??	SpU	0:00.21	/usr/local/lib/firefox/firefox	-contentproc	{3cd651d
47320	??	SpU	0:00.60	/usr/local/lib/firefox/firefox	-contentproc	{50d5261
26734	??	SpU	0:00.18	/usr/local/lib/firefox/firefox	-contentproc	{68aa722
308	??	SpU	0:00.16	/usr/local/lib/firefox/firefox	-contentproc	{bd6ff5f
42479	??	SpU	0:00.14	/usr/local/lib/firefox/firefox	-contentproc	{d874750
45572	??	Rp/2	0:00.00	grep	-i	firefox

Too much info!

Lets use the awk command to cut it to just the first and fifth columns!

```
ps -A | grep -i firefox | awk '{print $1, $5}'
```

```
43172    /usr/local/bin/firefox
59551    /usr/local/lib/firefox/firefox
 7023    /usr/local/lib/firefox/firefox
59478    /usr/local/lib/firefox/firefox
47320    /usr/local/lib/firefox/firefox
26734    /usr/local/lib/firefox/firefox
  308    /usr/local/lib/firefox/firefox
42479    /usr/local/lib/firefox/firefox
 5634    grep
```

Why is grep in there?

Oh yes... when we search for *firefox* we create a new process with *firefox* in its commandline. Lets drop the last line

```
ps -A | grep -i firefox | awk '{print $1, $5}' | ghead -n -1
```

43172	/usr/local/bin/firefox
59551	/usr/local/lib/firefox/firefox
7023	/usr/local/lib/firefox/firefox
59478	/usr/local/lib/firefox/firefox
47320	/usr/local/lib/firefox/firefox
26734	/usr/local/lib/firefox/firefox
308	/usr/local/lib/firefox/firefox
42479	/usr/local/lib/firefox/firefox

And really I'd just like a count of the number of processes

```
ps -A | grep -i firefox | awk '{print $1, $5}' | ghead -n -1 | wc -l
8
```

Other piping techniques

- The | pipe copies *standard output* to *standard input...*
- The > pipe copies *standard output* to a named file... (e.g. `ps -A >processes.txt`, see also the `tee` command)
- The >> pipe appends *standard output* to a *named file...*
- The < pipe reads a file *into* standard input... (e.g. `grep firefox <processes.txt`)
- The `<<` pipe takes a string and *places it on standard input*
- You can even copy and merge streams if you know their file descriptors (e.g. appending `2>&1` to a command will run it with *standard error* merged *into standard output*)

Wrap up

Go forth and shell script!

What we covered

- ▶ Variable expansions
- ▶ Common control flow statements
- ▶ Different pipe tricks