Lecture 23 - Programming and Scripting (Part 2)

DSE 511

Drew Schmidt
2022-11-15

Announcements

- Schedule:
  - Nov 15 - shell wrapup
  - Nov 17 and 22 - databases
  - Nov 24 - No class for US Thanksgiving
  - Nov 29 and Dec 1 - more databases
  - Dec 6 - course wrapup
- Old homework
  - Graded by next class period
- New homework (last one!)
  - Coming soon
  - Due Mon Dec 5? (fairly hard last date)
  - No homework on last modeule (databases)
- Questions?

Content

- Loops
- Functions and Arguments

Loops

Loops

- `for`
- `while`
- Some others not worth mentioning

Loop Syntax

```
for loopvar in $loopvars; do
  # ...
done
```

```
while [ condition ]; do
  # ...
done
```

Example

```
for i in 1 2 3; do
  echo $i
done
```

```
## 1
## 2
## 3
```

```
for i in `seq 1 3`; do
  echo $i
done
```

```
## 1
## 2
## 3
```

Example

```
seq 1 3
```

```
## 1
## 2
## 3
```

```
seq 0 2 6
```

```
## 0
## 2
## 4
## 6
```

```
seq 0 6 2
```

```
## 0
```

Example

```
root="/tmp/example"
mkdir -p $root
files="a b c"
for file in $files; do
  echo "Writing to $root/$file"
  touch $root/$file
done
```

```
## Writing to /tmp/example/a
## Writing to /tmp/example/b
## Writing to /tmp/example/c
```

```
ls /tmp/example
```

```
## a
## b
## c
```

Example

```
ls
```

```
data1.txt  data2.txt  data3.txt  data4.txt  data5.txt
```

```
for f in `ls *.txt`; do
  newf=`echo $f | sed -e 's/txt$/csv/'`
  mv $f $newf
  # or:
  # f_noext=${f%.*}
  # mv $f ${f_noext}.csv
done

ls
```

```
data1.csv  data2.csv  data3.csv  data4.csv  data5.csv
```

While Loops

- Not that useful in shell most of the time
- One exception is this pattern:

```
while true; do
if [ condition_is_met ]; then
do_whatever()
break
fi
done
```

Example

```
counter=1
while [ $counter -le 5 ]; do
  echo "counter=$counter"
  counter=$(( $counter + 1 ))
done
echo "Final value: $counter"
```

```
## counter=1
## counter=2
## counter=3
## counter=4
## counter=5
## Final value: 6
```

Example

```
RANDOM=1234
while true; do
  f="/tmp/myfile"
  if [ -e $f ]; then
    echo "Your file is ready at $f"
    break
  fi
  if [ $(( $RANDOM % 2 + 1 )) -eq 2 ]; then
    echo "Operation Succeeded!"
    touch $f
  else
    echo "Operation FAILED: trying again..."
    sleep 0.5
  fi
done
```

```
Operation FAILED: trying again...
Operation FAILED: trying again...
Operation Succeeded!
Your file is ready at /tmp/myfile
```

Functions and Arguments

Functions and Arguments

- Repeated operations within a script can be placed in a function
- Idea the same as every other language
- The implementation is *very* shell...

Syntax

Functions

```
function_name() {
  # commands go here
}

# call like this
function_name
```

Arguments

- Enumerated `$1`, `$2`, `$3`, ...
- "All" args is special: `$@`

Example

```
hello() {
  echo "Hello $1!"
}
hello world
```

```
Hello world!
```

```
hello universe
```

```
Hello universe!
```

```
hello to everyone out there
```

```
Hello to!
```

Example

```
hello() {
  echo "Hello $@!"
}
hello to everyone out there
```

```
Hello to everyone out there!
```

Example

```
hello() {
  echo "Hello to $1, $2, and $3"
}
hello you you you
```

```
Hello to you, you, and you
```

```
hello a b c d e
```

```
Hello to a, b, and c
```

```
hello x
```

```
Hello to x, , and
```

Example

```
only_one_arg() {
  if [ "$#" -ne 1 ]; then
    echo ERROR: only one argument should be provided
    exit 1
  fi
  echo You requested: $1
}
only_one_arg x
```

```
You requested: x
```

```
only_one_arg x y z w
```

```
ERROR: only one argument should be provided
```

Example

```
list_three() {
  ls $@ | sort | tail -n3
}
list_three /tmp
```

```
systemd-private-f79f97409b59440ca7d2f74fa98dc2ef-vnstat.service-DVWaXZ
tracker-extract-3-files.1000
tracker-extract-3-files.109
```

```
list_three /proc
```

```
vmallocinfo
vmstat
zoneinfo
```

Scoping

- Mostly works like you would expect
- Actually pretty similar to R and Python
- You can specify a local variable with `local`

Example

```
printer() {
  echo "$1 function: x=\"$x\" y=\"$y\""
}

scope_example() {
  local x="local x"
  printer During
  x="modified x"
  y="modified y"
}

x="global x"
y="global y"
printer Before
scope_example
printer After
```

```
## Before function: x="global x" y="global y"
## During function: x="local x" y="global y"
## After function: x="global x" y="modified y"
```

Redefining Commands

- Can (re-)define commands
- Usually not a good idea
- Opinion: bash aliases are usually the better approach

```
alias R='R --no-save --quiet'
```

Example

```
ls() {
  command ls -al $1 | head -n3
}
ls /proc
```

```
## total 4
## dr-xr-xr-x 559 root                root                      0 Nov 15 08:58 .
## drwxr-xr-x  20 root                root                   4096 Nov 15 09:03 ..
```

Wrapup

Wrapup

- Loops, functions, and scoping basically work like you would expect
- There are some peculiarities particular to the shell though...
- That's it for the shell!
- Next time: Introduction to Databases

Questions?