

# Introduction to Linux

## Session 3 – Shell Programming

Pete Ruprecht  
Research Computing Group  
University of Colorado Boulder  
[www.rc.colorado.edu](http://www.rc.colorado.edu)

# Outline

- Quoting
- Variables
- Tests and conditionals
- Decisions
- Arguments
- Loops
- Functions
- Alternative scripting languages

Slides available at

<https://github.com/ResearchComputing/Meetup-Fall-2013>

# Overview

- All Linux shells have built-in programming elements; bash is most feature rich
- Can program directly on the terminal command line or in script files
- Shell scripts must start with the definition of the shell used to interpret subsequent shell commands: `#!/path/to/shell`  
e.g., `#!/bin/bash`

# Quoting

- `'string'` – take string literally
- `"$MYVAR"` – allow variable interpolation
- ``cmd`` - string output from command
- `{ }` – delimits variable names

```
export NOW=`date +%Y%m%d`  
touch "data2.${NOW}.dat"
```

# Variables

- Shell variables are “local”
- Environment variables are “global”
- Use “.” to return a variable to the parent shell
- Variables can hold 1-D arrays:

```
city[0]=Juneau
```

```
city[1]=Wasilla
```

```
echo ${city[1]}
```

# Tests and Conditions

- Put test condition in [ ]
- String comparison

```
[ string1 = string2 ]
```

```
[ string1 != string2 ]
```

```
[ string1 =~ string2 ]
```

```
(string1 contains string2)
```

Spaces are important!!

# Tests (continued)

- Integer comparison

```
[ num1 -eq num2 ]
```

-ne (not equal), -gt (greater than),

-ge (greater or equal), -lt (less than),

-le (less or equal)

- Use bc to compare non-integers

```
[ `echo "$a>$b" | bc` = 1 ]
```

## Tests (still continued)

- Compound tests with && (AND) or | | (OR) go in [ [ ] ]

```
[ [ s1 != s2 && n1 -ge n2 ] ]
```

- Don't need [ ] if testing a return code:

```
if ! rm file.txt; then  
    echo "Remove Failed"  
fi
```



# Arguments

- It's often useful to pass arguments to a shell script
- `$1` denotes the first argument, `$2` the second, up to `{ 99 }`
- `$*` (all arguments, as a single word)
- `@` (all arguments, as individual words)
- `#` (total number of arguments)
- `$0` (name of script)

# Arguments (continued)

- Example:

```
#!/bin/bash
# calculate sine of argument (radians)
if [ $# -eq 1 ]; then
    sine=`echo "scale=4; s($1)" | bc -l`
    echo "The sine of $1 is $sine"
else
    echo "Usage: $0 <number in radians>"
fi
```

# Decisions

- if / then / else

```
if [ test ]; then
```

```
    command(s)
```

```
elif [ test2 ]; then
```

```
    command(s)
```

```
else
```

```
    command(s)
```

```
fi
```

## Decisions (continued)

```
#!/bin/bash
if [ $1 -gt 0 ]; then
    echo "$1 is positive"
elif [ $1 -eq 0 ]; then
    echo "$1 is zero"
else
    echo "$1 is negative"
fi
```

# Decisions (continued)

- case

```
case $variable in
value1)
    action1
;;
value2)
    action2
;;
value3|value4)
    action3
;;
*)
    default action
;;
esac
```

# Loops

“while”

```
while [ test ]; do  
    commands  
done
```

“for”

```
for variable in list; do  
    commands  
done
```

(when used in for or while loops, the `continue` and `break` commands will, respectively, immediately start the next iteration of the loop or exit the loop)

# “while” loop examples

```
c=-40
echo "Celsius Fahrenheit"
while [ $c -le 40 ]; do
  echo $c `echo "scale=3";(9/5)*$c+32" |bc`
  c=`expr $c + 1` #increment c by 1
  # c=$((c+1)) #alternate increment syntax
done
```

```
cat myfile | \
while read line; do
  if [ $line =~ data ]; then
    echo $line | awk '{print $3, $2*3.14}'
  fi
done
```

# “for” loop examples

```
for f in `ls -l *.txt`; do
  now=`date +%Y-%m-%d-%H-%M`
  cat $f | sed 's/UNIX/Unix/g' > ${f}_${now}
done
```

```
for i in {0..10..2}; do
  echo "$i is an even number"
done
```

```
for guy in Tom Dick Harry; do
  echo "$guy is my buddy"
done
```



# Functions

If a script needs to do the same task in several places, create a function.

```
function_name () {  
    commands;  
}
```

# Function example

```
send_email () {
  echo "Directory $dir is $stat" | \
  mail -s "size check" me@colorado.edu
return 0;
}
for dir in /data /home; do
  pct=`df $dir | grep $dir | \
  awk '{print $5}' | cut -d% -f1`
  if [ $pct -gt 90 ]; then
    stat="full"
    send_email
  else
    stat="ok"
    send_email
  fi
done
```

# HPC example

```
#!/bin/bash
# set up parameter and batch files for a set of cluster
# runs, then submit those jobs to the queue
xmax=30
ymax=20
x=10
while [ $x -le $xmax ] ; do
  y=10 # need to reinitialize y each time thru the x loop
  while [ $y -le $ymax ] ; do
    # use "here document" to create parameter file
    cat > param_${x}_${y} <<ENDofDOC
    $x
    $y
    3700
    output_${x}_${y}
  ENDofDOC

  # use "echo" commands to create batch scripts; compare with "here document" method
  echo "#!/bin/bash" > batch_${x}_${y}
  echo "#PBS -N ${x}_${y}" >> batch_${x}_${y}
  echo "#PBS -l nodes=3:ppn=8" >> batch_${x}_${y}
  echo 'cd $PBS_O_WORKDIR' >> batch_${x}_${y}
  echo "./my_prog.x < param_${x}_${y}" >> batch_${x}_${y}
  #submit job
  qsub -q queue batch_${x}_${y}
  y=$((y+5)) # increment y
done # repeat inner loop over y
x=$((x+10)) # increment x
done # repeat outer loop over x
```

# Alternatives

- **perl** – exceptional text manipulation and parsing
- **python**– designed for clarity rather than compactness; excellent scientific and numerical extensions
- **php** – used for preprocessing dynamic web pages
- **Tcl/Tk** – useful for creating windows via GUI library
- **make** – for building executable programs from source code

# Thank you!

More info about CU Research Computing:  
[www.rc.colorado.edu](http://www.rc.colorado.edu)