Bash Scripting & Shortcuts

Command Piping & Chaining

Mostly a refresher, but we'll demo these out

- Can send output using ">"
- Can receive input using "<"
- Can send error output using "2>"
- Pipe one command to another using "|"
- Run a 2nd command only if the 1st succeeds with "&&"
- Run a 2nd command only if the 1st fails with "||"

Common Keyboard Shortcuts

- C-C: Kill running program
- C-D: Send EOF indicator
- C-Z: Send running program to background
- C-L: Clear the terminal
- C-R: Search prior commands

Exclamation Marks!

Exclamation marks can be used for a variety of things:

- !n Repeat n^{th} command from history
- !! Repeat previous command
- !word Repeat previous command starting with “word”
- !:n Repeat the nth argument from the prior command
- !:* Repeat all arguments from the prior command

Similar:

- ^this^that^: Re-execute the previous command, but replace “this” with “that”

Variables

```
x=10

echo x    # Prints “x”

echo $x   # Prints “10”

y=5

x=$y

echo $x   # Prints “5”
```

Expressions

Various kinds of expressions in bash are represented by their enclosing paren type

- Double parens: integer math
  - Ex: `(( 5 + 5 ))`
  - When using as args, put a dollar sign before expression
    - Ex: `echo $(( 5 + 5 )) # Prints 10`
- Single square brackets: condition testing (usually used in control flow)
  - Ex: `[ $x -eq 5 ]` checks if the variable “x” is equal to 5
  - There are also double square brackets, which are very similar
  - More choices than just “equal”, more on next slides
- Braces:
  - Used for expansion (more on next slides)
  - Used with dollar sign to reference variables
  - ex:

```
location=RIT
echo ${location}lug # Prints RITlug
```

Bash Boolean “test” Operators

Bash doesn’t use “<”, “>”, etc. signs since those are reserved for IO, instead it uses:

- -lt <
- -gt >
- -le <=
- -ge >=
- -eq ==
- -ne !=
- -n (string is not empty)
- -z (string is empty)
- -f (file exists)

Braces Expansion

Braces can expand to make copies of a string:

```
echo “Hello “{pete,repeat} # Prints “Hello pete Hello repeat”
```

They can also enumerate values:

```
echo {1..10} # Prints “1 2 3 4 5 6 7 8 9 10”
```

Put these together and you get…

```
echo "file"{1..10}".txt"
file1.txt file2.txt … file10.txt
```

If statements

```
if condition
then
  cmd1
else
  cmd2
fi
```

While loops

```
while condition
do
  cmd1
  cmd2
  ...
end
```

For Loops (Bash Style)

```
for var in list
do
  cmd1
  cmd2
  ...
done
```

For Loops (C Style)

```
for ((initialize ; condition ; increment))
do
   cmd1
   cmd2
   ...
done
```

These have a weird syntax, see demo

Aliases

If you run the same command often, you can alias it to something easier

Ex:

```
alias push='git push'
alias la='ls -a'
alias rit='ssh myname@glados.cs.rit.edu' # There are better ways to do this
alias clean='rm *.class *.out *.o &> /dev/null'
```

Functions Syntax

Functions can do even more complex things

```
function() {
  cmd1
  cmd2
  ...
}
```

Notes about functions

- Parens are just syntax, no args go in there
- When calling the function, use its name only, no parens
- There is no automatic scoping, vars declared in functions are global
- Functions “return” the exit code of their last command
- Arguments
  - $1, $2, $3 are the 1st, 2nd, 3rd, etc. arguments
  - $# is the number of arguments
  - $* or $@ is all arguments (there are slight differences when interpreting these as strings)
- If trying to refer to a builtin command within a function, use the builtin keyword

Example Functions

```
cd () {
  builtin cd "$@"
  ls -A --color=auto
}

mkcd () {
  mkdir "$@" && cd "$@"
}
```

Writing a .sh file (bash script)

- First line is `#! /bin/bash` or `#! /usr/bin/bash`
  - Generally whatever comes after the shebang is used to interpret the file
- After saving the file, run `chmod +x filename`to make it executable
- Run it with `./filename.sh`

That's it, it's actually quite straightforward once you have the hang of the scripting language

Miscellaneous Stuff

Sourcing:

```
. file_to_run # Commonly used for dotfiles
```

PS1 Special Variable: The string printed when prompting

You can use semicolons to put multiple commands on the same line

Thank You!

Lots of resources for learning:

- The obvious: `man bash` and `help`
- Bracket types reference: https://www.assertnotmagic.com/2018/06/20/bash-brackets-quick-reference/
- Bash scripting operators: https://linuxconfig.org/bash-scripting-operators
- If statements: https://www.geeksforgeeks.org/bash-scripting-if-statement/
- Loops: https://linuxhandbook.com/bash-loops/
- Functions: https://linuxize.com/post/bash-functions/
- My personal dotfiles: https://github.com/jzaia18/dotfiles