CSE 15L: Software Tools and Techniques Laboratory

Winter 2021 - http://ieng6.ucsd.edu/~cs15x

Instructors: Gary Gillespie Keith Muller

Class sessions will be recorded and made available to students asynchronously.

More Useful Commands

- **sort** accepts a list of data from either standard input or a single filename argument and, by default, sorts the list in alphabetical order
- **uniq** accepts a sorted list of data from either standard input or a single filename argument and, by default, removes any duplicates from the list
- **head/tail** print first/last part of a file (or standard input) default is 10 lines

Sort & Uniq

```
kmuller@keithm-pi4:~ $ cat >XX
one
two
one
kmuller@keithm-pi4:~ $ uniq XX
one
two
one
kmuller@keithm-pi4:~ $ cat > YY
one
one
two
kmuller@keithm-pi4:~ $ uniq YY
one
two
kmuller@keithm-pi4:~ $ sort XX
one
one
two
```

head/tail

```
kmuller@keithm-pi4:~ $ ls -l /usr/bin | head
total 128204
-rwxr-xr-x 1 root root     42744 Feb 28  2019 [
-rwxr-xr-x 1 root root        96 Oct 10  2019 2to3-2.7
-rwxr-xr-x 1 root root     13780 Mar 30  2019 aconnect
-rwxr-xr-x 1 root root     17880 Jan 10  2019 addpart
lrwxrwxrwx 1 root root        29 Feb  6  2020 addr2line -> arm-
linux-gnueabihf-addr2line
-rwxr-xr-x 1 root root     18252 Dec 13 04:16 agnostics
-rwxr-xr-x 1 root root      1046 Sep  1 03:56 alacarte
-rwxr-xr-x 1 root root     38404 Mar 30  2019 alsabat
-rwxr-xr-x 1 root root     62932 Mar 30  2019 alsaloop
kmuller@keithm-pi4:~ $ ls -l /usr/bin | tail -n 5
-rwxr-xr-x 1 root root      2953 Jul 30  2019 zipgrep
-rwxr-xr-x 2 root root    145240 Jul 30  2019 zipinfo
-rwxr-xr-x 1 root root      2205 Jan  5  2019 zless
-rwxr-xr-x 1 root root      1841 Jan  5  2019 zmore
-rwxr-xr-x 1 root root      4552 Jan  5  2019 znew
```

More Piping

- Question: find programs (if any) with the same name that appear in both the */bin* and */usr/bin* directories

```
$ ls /bin /usr/bin | sort | uniq | tail -n 5
zipgrep
zipinfo
zless
zmore
znew
```

Expansion

- Each time we type a command and press the enter key, bash performs several substitutions upon the text before it carries out our command

```
$ echo this is a test
this is a test
```

- pathname expansion

```
$ echo *
Desktop Documents output.txt Music Pictures Public
Templates Videos
```

- brace {} expansion

```
$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

More Brace Expansion

```
$ mkdir Photos
$ cd Photos
$ mkdir {2007..2009}-{01..12}
$ ls
2007-01 2007-07 2008-01 2008-07 2009-01
2009-07 2007-02 2007-08 2008-02 2008-08
2009-02 2009-08 2007-03 2007-09 2008-03
2008-09 2009-03 2009-09 2007-04 2007-10
2008-04 2008-10 2009-04 2009-10 2007-05
2007-11 2008-05 2008-11 2009-05 2009-11
2007-06 2007-12 2008-06 2008-12 2009-06
2009-12
```

Command Substitution

- $(cmd args) allows us to use the output of a command as an expansion

Example

- **which** returns the pathnames of the files (or links) which would be executed in the current environment, had its arguments were used as commands in a shell

```
$ which cp
/usr/bin/cp
$ ls -ls $(which cp)
112 -rwxr-xr-x 1 root root 112780 Feb 28  2019 /usr/bin/cp
```

Controlling Expansion: Quoting

- *word splitting* is where the shell removes extra whitespace from a command's list of arguments
  - word splitting looks for the presence of spaces, tabs, and newlines (line feed characters) and treats them as *delimiters* between words
  - unquoted spaces, tabs, and newlines are not considered to be part of the text. They serve only as separators

```
$ echo this is a          test
this is a test
```

- *double quotes* all the special characters used by the shell lose their special meaning and are treated as ordinary characters.
  - The exceptions are $ (dollar sign), \ (backslash), and ` (backtick).

```
$ echo "this is a          test"
this is a          test
```

Quote Characters

Three different quote characters with different behavior:

" : double quote, weak quote

If a string is enclosed in " " the references to variables (i.e *$variable. - later lecture*) are replaced by their values

- back-quote and escape \ characters are treated specially.

' : single quote, strong quote

Everything inside single quotes are taken literally; nothing is treated as special.

` : back quote (back tick) – alternative is $(cmd)

Enclosed string is treated as a command and the shell attempts to execute it. If the execution is successful, the primary output from the command replaces the string.

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Quoting

```
kmuller@keithm-pi4:~ $ cal
    January 2021
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

or echo `cal`

this is all on one line, line wrap is in pptx slide

```
kmuller@keithm-pi4:~ $ echo $(cal)
January 2021 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31
kmuller@keithm-pi4:~ $ echo "$(cal)"
    January 2021
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
kmuller@keithm-pi4:~ $ echo '$(cal)'
$(cal)
```

What is Shell Script?

- A **shell script** is an interprested script written for the shell

- Two key ingredients
  - Any UNIX/LINUX commands
  - Shell script

– Good and complete reference on the web

https://www.gnu.org/software/bash/manual/

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Steps to Write A Shell Script

1. **Write a script**. Shell scripts are ordinary text files, use a text editor to write them.
2. **Make the script executable**. set the script file's permissions to allow execution.
3. **Put the script somewhere the shell can find it**. The shell automatically searches certain directories for executable files **when no explicit pathname is specified**
   - the system searches a list of directories each time it needs to find an executable program, if no explicit path is specified

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
```

Getting started…

- Create a shell script file
  ```
  $ vi scriptfilename.sh
  ```

- First line contains #! (the #! if often referred to as the ***shebang***) and it tells the OS the absolute path to desired shell for interpreting the shell script
  ```
  #!/usr/bin/bash
  ```

- All other lines starting with # are comments
  - Make code readable by including comments!!

- Tell Unix that the script file is executable (only for first choice below)
  ```
  $ chmod u+x scriptfilename.sh
  $ chmod +x scriptfilename.sh
  ```

- Run the shell-script
  ```
  $ ./scriptfilename.sh   # needs execute permission, new process
  $ bash scriptfilename.sh # executes in new bash process

  $ . scriptfilename.sh    # executes in current shell process
  $ source scriptfilename.sh  # executes in current shell process
  ```


Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

A Simple Shell Script

```
$ vi myfirstscript.sh
$ cat myfirstscript.sh
#!/usr/bin/bash

# The first example of a shell script
directory=`pwd`
echo Hello World!
echo The date today is `date`
echo The current directory is $directory
$ chmod +x myfirstscript.sh
$ ./myfirstscript.sh
Hello World!
The date today is Wed Oct 21 12:20:09 PDT 2020
The current directory is /home/linux/ieng6/ieng6/cs15x
```


Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Access Rights to Files and Directories

- Access rights (by a user) to file system "file" (regular files, directories, etc.) are defined in terms of read access, write access, and execution access

```
$ ls -ls config.txt
4 -rw-r--r-- 1 kmuller kmuller 1895 Jan 14 21:06 config.txt
```

ls displays type & permission with 10 characters

File_type(1) owner_rights(3) group_rights(3) Other_rights(3)

Attribute	File type
-	A regular file.
d	A directory.
l	A symbolic link. Notice that with symbolic links, the remaining file attributes are always rwxrwxrwx and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A *character special file*. This file type refers to a device that handles data as a stream of bytes, such as a terminal or /dev/null.
b	A *block special file*. This file type refers to a device that handles data in blocks, such as a hard drive or DVD drive.

Access Rights to Files and Directories

- The remaining nine characters of the file attributes, called the *file mode*, describe the read, write, and execute permissions for the file's owner, the group owner (many users in a group), and everybody else (world). Groups lists groups you are a member of

```
$ groups
kmuller adm sudo audio video plugdev games users netdev lpadmin gpio i2c spi
```

Special	Owner	Group	World (other)
suid, segid, restricted_deletion	rwx	rwx	rwx

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered, e.g., `cd directory`.

Permission Examples

File Attributes	Meaning
`-rwx------`	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
`-rw-------`	A regular file that is readable and writable by the file's owner. No one else has any access.
`-rw-r--r--`	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
`-rwxr-xr-x`	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
`-rw-rw----`	A regular file that is readable and writable by the file's owner and members of the file's group owner only.

chmod: Changing File Permissions

- **chmod** (change mode) command sets a file's permissions (read, write, and execute) for all three categories of users (owner, group, and others)

command operation file

```
$ chmod u+x myfirstscript.sh
```

category permission

- **Category of user:** owner (**u**), group (**g**), others (**o**), or all (**a**)
- **Operation**: add (**+**), remove (**–**), or assign (**=**) a permission
- **Permission**: read (**r**), write (**w**), or execute (**x**)

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

chmod: Changing File Permissions

To add the executable permission to a file for the user:

```
$ ls –l myscript.sh
-rw-r--r-- 1 cs15lfa20 cs15lfa20 39 Apr 15 13:13 myscript.sh
$ chmod u+x myscript.sh
$ ls –l myscript.sh
-rwxr--r-- 1 cs15lfa20 cs15lfa20 39 Apr 15 13:13 myscript.sh
```

To remove all permissions from a file for the user:

```
$ ls –l notforme.txt
-rw-r--r-- 1 cs15lfa20 cs15lfa20 2223 Apr 15 13:16 notforme.txt
$ chmod u-rwx notforme.txt
$ ls –l notforme.txt
----r--r-- 1 cs15lfa20 cs15lfa20 2223 Apr 15 13:16 notforme.txt
```


Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

chmod: Changing File Permissions

To add the read permission to a file for all users and the write permission for the user:

```
$ ls –l ssh_key
-r-------- 1 cs15lfa20 cs15lfa20 39 Apr 15 13:18 ssh_key
$ chmod a+r,u+w ssh_key
$ ls –l ssh_key
-rw-r--r-- 1 cs15lfa20 cs15lfa20 39 Apr 15 13:18 ssh_key
```

You can assign the permission with the **=** operator:

```
$ ls –l forme.txt
-rw-r--r-- 1 cs15lfa20 cs15lfa20 2223 Apr 15 13:16 forme.txt
$ chmod u=rwx forme.txt
$ ls –l forme.txt
-rwxr--r-- 1 cs15lfa20 cs15lfa20 2223 Apr 15 13:16 forme.txt
```


Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

`chmod`: Changing File Permissions

Instead of incremental changes, you can assign the permission with the **=** operator!

Several different, equivalent methods for assigning read permissions for all users:

```
$ chmod ugo=r goodforreading.txt
$ chmod a=r goodforreading.txt
$ chmod =r goodforreading.txt
```


Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

`chmod`: Changing with Short Notation

File permissions represent a binary bitfield (viewable by `ls -l`)

`rwxrwxrwx`

User Group Others

`r`, **`w`**, **`x`** represent set and **`-`** represents unset permission
Each permission represented as a single bit (binary) value

describe (and change) the field using octal numbers

Each type of permission is assigned a weight as shown:

- Read permission: 4
- Write permission: 2
- Execute permission: 1

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

chmod: Changing with Short Notation

bit weight	421	421	421
binary	111	111	111
	rwx	**rwx**	**rwx**
	User	**Group**	**Others**

Read (4 = b100) Write (2 = b010) Execute (1 = b001)

In a category each permission is ether enabled (1) or disable (0)
The three bits in a category are described by a octal number

For instance, if user has read (**r**) and write (**w**) permissions, this category is represented by the number:

06 = 4 + 2 + 0 = b100 + b010 + b000 = b110 = **rw-**

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

chmod: Changing with Short Notation

-rwxrwxrwx

User Group Others

When repeated for each category, you have a three-character octal number representing: user, group, and others

$ chmod 640 target

user: r+w = 6, group: r = 4, other: none = 0

Apply the **chmod** command recursively to all files and subdirectories with **–R** (recursive) option.

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Directory Permissions

Slightly different meaning than file permissions!

- Read permission for a directory means that **`ls`** can read the list of filenames stored in that directory.
- Write permission for a directory implies that you are permitted to create or remove an entry in it.
- Execution privilege of a directory means that a user can pass through and access any entries in the directory (after which the entries permissions are then applied)

`drwxrwx---`	A directory. The owner and the members of the owner group may enter the directory and create, rename, and remove files within the directory.
`drwxr-x---`	A directory. The owner may enter the directory and create, rename, and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete, or rename files.

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

What are the permissions on **`script1.sh`** after the following command?

`chmod 562 script1.sh`

a. `r-xrw--w-`
b. `r-xrw-r--`
c. `r-xr---w-`
d. `r-xr--r--`
e. None of the above

Assuming the file ***note*** originally has "**`r-xr--r--`**" as access permissions, the

`chmod ugo+w note`

command can be represented in octal notation as:

a. `chmod 222 note`
b. `chmod 766 note`
c. `chmod 666 note`
d. `chmod 746 note`
e. None of the above

Commenting

- Lines starting with # are comments except the very first line where #! indicates the location of the shell that will be run to execute the script.
- On any line, characters following unquoted # are considered to be comments and ignored

- Comments are used to:
  - Identify who wrote it and when
  - Identify input variables
  - Make code easy to read
  - Explain complex code sections
  - Version control tracking
  - Record modifications

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

echo command

echo command is useful when trying to debug scripts!

Syntax : **echo [options] stringtoprint**

String can be **"weakly quoted"** or **'strongly quoted'**
In weakly quoted strings, references to variables are replaced by the value of those variables before the output.

Options: **-e** : expand **** (back-slash) special characters
-n : do not output a new-line at the end.

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

User Input During Shell Script Execution

User input from stdin (terminal or file) can be captured using the **read** command

```
$ cat exampleread.sh
#!/usr/bin/bash
echo "Please enter three filenames:"
read  filea fileb filec
echo "These files are used:$filea $fileb $filec"
```

Each **read** statement reads an entire line (terminated by \n). In the above example, if there are less than 3 items in the response the trailing variables will be set to blank `''`.

Three items are separated by one space.

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Debugging shell scripts

Generous use of the **`echo`** command will help.

Run script with the **`–x`** parameter.

e.g. **`$ bash –x ./myscript`**

or **`$ set –o xtrace`** # before running the script.

These options can be added to the first line of the script where the shell is defined.

e.g. **`#!/usr/bin/bash -xv`**

Slide ref: *its2.unc.edu/divisions/rc/training/scientific/short.../**Shell_Scripting.ppt*** byShubin Liu, Ph.D.

Next Lecture

1. Build Automation
2. Using **`make`** and Makefiles