Lecture 16 - Basic Shell

DSE 511

Drew Schmidt
2022-10-20

Announcements

- New homework not yet ready
- Coming Soon (TM)
- Questions?

Content

- The File System
- Permissions
- Pipes and Redirection
- Some Useful Tricks

The File System

The File System

- We need to talk about "the file system"
- Not at the level of partition/format
- Structure, basic data operations, etc.

FS Hierarchy

- `/` - root (top level)
  - `/home` - user storage
    - `/home/user1`
    - `/home/user2`
  - `/tmp` - volatile temp space
  - `/bin` and `/lib` - "essential"/core programs/libraries
  - `/usr` - "UNIX Software Resources" (kind of like "Program Files")
  - `/opt` - "add-on" software (often non-free)
  - `/proc` - important kernel info
  - ...

Some FS Concepts

- List contents - `ls`
- Current directory - `pwd` (print working dir)
- Change directory - `cd`
- Create (empty) file - `touch`
- Create directory - `mkdir`
- Delete file/directory - `rm` (remove)
- Rename file/directory - `mv` (move)

Some FS Concepts

- Absolute paths
  - `cd /tmp`
  - `cd /home/my_username`
- Relative paths
  - `cd ..`
  - `mkdir ./example`
- Combining the two
  - `mv /tmp/x .`

You've Seen All This Before (Probably)

R

```
#
setwd("/tmp")
getwd()
```

```
[1] "/tmp"
```

```
dir.create("/tmp/example")
```

Python

```
import os
os.chdir('/tmp')
os.getcwd()
```

```
'/tmp'
```

```
os.mkdir("/tmp/example")
```

Some FS Concepts

```
cd /tmp
pwd
```

```
/tmp
```

```
mkdir example && cd example
touch x
ls -al
```

```
drwxrwxr-x   2 mschmid3  mschmid3 4096  Oct 16 15:49 .
drwxrwxrwt 107      root      root 36864 Oct 16 15:49 ..
-rw-rw-r--   1 mschmid3  mschmid3     0 Oct 16 15:49 x
```

```
rm /tmp/example
```

```
rm: cannot remove '/tmp/example': Is a directory
```

```
rm -rf /tmp/example
```

Working With Files: The Basics

- Basic examination tools
  - `cat`
  - `head` and `tail`
  - `less`
- Editors
  - `nano` / `pico`
  - `vim`
  - `emacs`

Example: Working With Proc Files

```
head -n 3 /proc/meminfo
```

```
MemTotal:        65787120 kB
MemFree:          2091012 kB
MemAvailable:    44710744 kB
Buffers:          2192132 kB
Cached:          40933800 kB
SwapCached:           216 kB
Active:          15906844 kB
Inactive:        45051672 kB
Active(anon):       62084 kB
Inactive(anon): 19295248 kB
```

```
head /proc/cpuinfo
```

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family       : 23
model           : 1
model name       : AMD Ryzen 7 1700X Eight-Core Processo
stepping       : 1
microcode       : 0x8001137
cpu MHz           : 2313.924
cache size       : 512 KB
physical id       : 0
```

Permissions

Permissions

- Different users have different file permissions
- Remember: *NIX is multi-user!
- User
    - reads/writes to user space and tmp
    - reads/executes various bin/lib paths
- root reads/writes everywhere!

Recall Our 'ls -al' Example

```
ls -al
```

```
drwxrwxr-x   2 mschmid3  mschmid3 4096  Oct 16 15:49 .
drwxrwxrwt 107     root      root 36864 Oct 16 15:49 ..
-rw-rw-r--   1 mschmid3  mschmid3     0 Oct 16 15:49 x
```

Permissions

```
---     ---     ---
rwx     rwx     rwx
user    group   other
```

- `r` - read access
- `w` - write access
- `x` - executable (permissions)
- `-` - permission is lacking

Some Notes on Permissions

- If you don't have permission to execute a program, you can't run it
- Even if it's a perfectly legitimate binary executable!
- "Wide open" permissions (`777`) are a bad idea, always
- root sees all!
- This stuff actually gets kind of complicated...

Changing Permissions

Use `chmod`

Shorthand

- Give read access: `chmod +r some_file`
- Give write access: `chmod +w some_file`
- Give execution access: `chmod +x some_file`
- Remove by `s/+/-/`

Octals

- A binary triple (number from 0 to 7)
- Defines a file mode `rwx`
- In the triple: 1 means "allowed" 0 means "not allowed"
- $4_{10} = 100_2$ - `r--`
- $6_{10} = 110_2$ - `rw-`

Permissions Example

```
mkdir test
cd test
touch x
chmod 600 x
ls -al x
```

```
-rw------- 1 mschmid3 mschmid3 0 Oct 16 15:43 x
```

```
chmod 640 x
ls -l x
```

```
-rw-r----- 1 mschmid3 mschmid3 0 Oct 16 15:43 x
```

Pipes and Redirection

Pipes and Redirection

- Output from programs can be "redirected"
  - To files
  - To other programs
- "Standard output" and "standard error" are different

Redirecting to Files

Standard Output

```
Rscript -e "1+1" > /tmp/output.txt
cat /tmp/output.txt
```

```
[1] 2
```

Standard Error

```
Rscript -e "stop()" > /tmp/output.txt
```

```
Error:
Execution halted
```

```
cat /tmp/output.txt
```

```
Rscript -e "stop()" 2> /tmp/output.txt
cat /tmp/output.txt
```

```
Error:
Execution halted
```

Redirecting to Files

```
Rscript -e "1+1; stop()" > /tmp/output.txt
```

```
Error:
Execution halted
```

```
cat /tmp/output.txt
```

```
[1] 2
```

```
Rscript -e "1+1; stop()" 2> /tmp/output.txt
```

```
[1] 2
```

```
cat /tmp/output.txt
```

```
Error:
Execution halted
```

Redirecting to Files

```
Rscript -e "1+1; stop()" &> /tmp/output.txt
cat /tmp/output.txt
```

```
[1] 2
Error:
Execution halted
```

Pipes

- Can "pass the output" to a file
- Can also pass it to a program
- Use the pipe `|` (`shift` + `\` on US keyboard)
- Sort of like R's native `|>` or magrittr's `%>%`

Pipes

```
Rscript -e "1+1; stop()" | head -n 1
```

```
[1] 2
Error:
Execution halted
```

```
Rscript -e "1+1; stop()" 2>&1 | head -n 1
```

```
[1] 2
```

Pipes

```
Rscript -e "rnorm(1)" | wc
```

```
      1       2      13
```

```
Rscript -e "rnorm(1)" | wc -c
```

```
13
```

An Observation

- The shell is a powerful interactive inspection tool
- Data science is an interactive inspection job!

Logical Operators

- There are logical operators for programs
- `&&` and `||`
- Don't confuse `|` with `||`!
- I use `&&` as a shorthand for "and then do this other thing"
- We'll return to these later

Some Useful Tricks

Tab Completion

- The shell supports tab completion
- This can be made case-insensitive (see TODO)
- ALWAYS MASH TAB

Control Characters

- `ctrl`+`c` - "breaks" running program; resets what you're typing
- `ctrl`+`d` - exits the current shell
- `ctrl`+`l` - clears screen
- `ctrl`+`r`
  - Searches history
  - I generally use `history | less`
- `ctrl`+`z`
  - "Stops" a program
  - Use `fg` to bring it back!

Checking Command History

```
history | tail
```

```
1992  man sh
1993  man ls
1994  ls --color=always
1995  ls
1996  ls ..
1997  ls --color=always ..
1998  ls --color=never ..
1999  cd bin
2000  cat buildrd
2001  history | tail
```

Foreground and Background

```
R
```

```
x = 1
# I press ctrl+z here
```

```
[1]+  Stopped                 R --no-save --quiet
```

```
ps aux | grep bin/exec/R | grep -v grep | tail -n 1
```

```
mschmid3 2248505  1.6  0.1 2313472 71412 pts/15  Tl   08:18   0:01 /usr/lib/R/bin/exec/R --no-save -
```

```
fg
```

```
x
```

```
[1] 1
```

pushd and popd

```
cd /tmp
pushd .
```

```
/tmp /tmp
```

```
cd /proc
pwd
```

```
/proc
```

```
popd
```

```
/tmp
```

```
pwd
```

```
/tmp
```

Wrapup

Wrapup

- A bit of a whirlwind…
- Working with files is where the shell really shines.
- Pipes and redirection are extremely powerful: more on this later!
- We've seen some helpful utilities: `head`, `cat`, …
- Next time: more shell utilities

Questions?