The Shell

Joshua Kwan
System Administration DeCal
February 20, 2008

What is the shell?

- A program; your gateway into any Unix system
- Allows you to inspect, manipulate, add/remove files and parts of the system
- Allows you to run other programs and control how they run
- A compact programming environment to allow automation of many tasks (viz. DOS batch files)

Diving in: Basic Navigation

Demonstration of:

cd – change directory
ls – list directory
pwd – show current directory
(Print Working Directory)
mkdir – create an empty directory
rmdir – remove an empty directory

Review

- First checked to see where we were (pwd)
- Created a 'people-i-like' directory (mkdir)
- Entered that directory (cd), created *subdirectories* of 'people-i-like'
- Deleted the 'calvin' subdirectory (rmdir)
- Went back to our home directory (cd)

Looking for Things: find, grep, locate

- **grep** – used to search within files for certain patterns (*regular expressions*)
  `grep Josh santas-naughty-list.txt`
- **find** – used to find files within a directory structure
  `find . -name ‘present.txt’`
- **locate** – quick but not necessarily up-to-date file search (indexed every 24 hours)
  `locate my-brain.dat`

Moving Things Around: cp, mv, rm

- **cp** – Used to copy one file to another location, or just a different name.
  `cp people.txt persons.txt`
- **mv** – Used to move a file to another location or to rename it.
  `mv old-records.txt new-records.txt`
- **rm** – Used to delete a file.
  `rm incriminating.txt`

The UNIX Paradigm

- Similar to the RISC vs. CISC ideology
- Write small programs with small purpose and chain them together, vs. huge programs that do just one thing
- The shell makes this chaining possible with its most powerful feature: the **pipe** (‘|’)
- … Put that in your pipe and smoke it!

Pipes (a.k.a. Cool Stuff)

- Pipes are a way to chain the **output** of one command into the **input** of another
- For example, you can *grep* the output of *ls* or *find* or anything you want!
- Or.. You can *ls* the output of *grep*! Anything goes.
- If you have a Mac, this is the idea upon which Automator is based.

Impressive Pipe Examples

Convert all WAV files in a directory to OGG

```
find | grep ‘.wav$’| xargs oggenc
```

Count how many lines a text file has

```
cat jonathan.txt | wc -l
```

Get the file size of every file in a directory by using ls verbose options

```
ls -l | awk '{print $5 $8}'
```

Output Redirection

- Running programs can relay output to the screen via two channels: standard output and standard error
- The shell lets you control the *flow* of these using > and 2>
- Log the output of a program to prog.log and errors to error.log:
  `someprogram >prog.log 2>error.log`

Input Redirection

- Many UNIX-type programs wait for input; they read from “standard input”, by default user input.
- So you can pipe things *to* them, or redirect their input *from* something.

```
frobnicate < foobar.txt
```

/dev/null

- A special file on the system that contains nothing and ignores what you write to it!
- Thus: to discard the output of a program, redirect it to /dev/null!
- Or, to explicitly pass no input to a program that waits for input, redirect its standard input FROM /dev/null!

Shell Programming

- The bash shell supports some basic programming constructs, e.g. for, while, if-then
- Use in tandem with shell commands to do really cool stuff!

```
ls people-i-like | while read person; do
  if [ “$person” = Iris ]; then
     echo “$person is unusually cool!”
  else
     echo “I really like $person! <3”
  fi
done
```

Substitution

- Sometimes, you want to substitute the output of one command into the command line of another
- Or, insert the contents of a file into a command line. This can be done with **substitution**

```
rm $(locate .avi)
rm $(<files-to-delete.txt)
```