Introduction to UNIX

Benjamin Brewster
Adapted from slides by Jon Herlocker, OSU

Topics

- History, etc.
- The Shell
  - How to run commands on the command line
  - Useful commands
- Shell Programming

UNIX

- **1965]** Bell Laboratories joins with MIT and General Electric in the development effort for the new operating system, Multics, which would provide a multi-user, multi-processor, and multi-level (hierarchical) file system, among its many forward-looking features.

From: http://wks.uts.ohio-state.edu/unix_course/intro-2.html#HEADING2-0

UNIX

- **1969]** AT&T was unhappy with the progress and drops out of the Multics project. Some of the Bell Labs programmers who had worked on this project, Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and implemented the first version of the UNIX File System on a PDP-7 along with a few utilities. It was given the name UNIX by Brian Kernighan as a pun on Multics.

From: http://wks.uts.ohio-state.edu/unix_course/intro-2.html#HEADING2-0

UNIX

- **1970]** Jan 1 becomes "time zero" for UNIX

- **1971]** The system now runs on a PDP-11, with 16 Kbytes of memory, including 8 Kbytes for user programs and a 512 Kbyte disk.

UNIX

- **1971]** First real use is as a text processing tool for the patent department at Bell Labs. That utilization justified further research and development by the programming group. UNIX caught on among programmers because it was designed with these features:
  - programmers environment
  - simple user interface
  - simple utilities that can be combined to perform powerful functions
  - hierarchical file system
  - simple interface to devices consistent with file format
  - multi-user, multi-process system
  - architecture independent and transparent to the user.

From: http://wks.uts.ohio-state.edu/unix_course/intro-2.html#HEADING2-0

UNIX

- **1973]** Unix is re-written mostly in C, a new language developed by Dennis Ritchie. Being written in this high-level language greatly decreased the effort needed to port it to new machines.

- Unix development and C development go hand in hand – they are symbiotic

From: http://wks.uts.ohio-state.edu/unix_course/intro-2.html#HEADING2-0

Linux

- Initially developed by Linux Torvalds
- Supports the POSIX UNIX specification
  - Code written for another POSIX-based UNIX (ie Solaris, HPUX, AIX, etc) shouldn't need many changes to run on Linux.
- Open source
  - Protected by the GNU Public License (GPL)
- Over the years, gained many developers
- Now a robust, stable, and free OS
- Linux knowledge will apply to most UNIX systems

You Can Run Linux at Home

- There are many Linux distributions
  - Red Hat, SuSE, Debian, Mandrake, etc.
  - All can be downloaded; some can be purchased
  - ENGR has one major Linux machine for CS311
    - flip.engr.orst.edu

- Programming assignments for this class
  - Can be done on your home Linux machine
  - But make sure they work on flip before submitting them!

The UNIX Shell

I'm going to call *it* UNIX for the rest of the class

- UI: A command line interpreter
  - Sometimes called a shell, prompt, or terminal(!)
- Basic Shells
  - Bourne Shell (/bin/sh)
  - C-Shell (/bin/csh)
- Enhanced Shells
  - BASH "Bourne-again shell" (/usr/local/bin/bash)
  - TCSH enhanced C-Shell (/usr/local/bin/tcsh)
  - Korn Shell (/bin/ksh)
- You can change your shell
  - Use the `chsh` command.

Learning the UNIX UI

- Man pages
  - man command
    - "man man" – retrieves the online manual for the manual command.
  - Web versions of man pages and more
    - Just search for them on google.
    - Make sure you have the LINUX man page.
      - There are slight differences between POSIX OS's
  - Keyword search from the shell
    - Apropos or "man –k"

Man pages?

- Man pages == PAIN for learning the UNIX UI
  - Really
  - Ask anyone
  - Documentation is *Linux's* biggest problem

- Man pages are not useful for learning to use UNIX, but they do describe each command in (painful) detail – if you know what you're looking for

- Man pages are sometimes broken on flip
  - Fix with this command:
    - `% unsetenv MANPATH`

Learning the UNIX UI

- Geeks
  - They're everywhere
  - How do I ___?
  - Best source

- Teh intarweb
  - Primary source – figure it out yourself via your own research!

- A Book
  - Many, many books, in fact
  - Careful: your textbook is a UNIX *programming* book – not a directory of UNIX commands

Man Pages Strike Back

- The `man` manual is usefully organized into sections based on the type of command:

  1. User Commands
  2. System Calls
  3. C Library Functions
  4. Devices and Network Interfaces
  5. File Formats
  6. …

A Critical Difference

- **User Commands**
  - Commands that can only be run from the shell, and shell scripts; `cd`, `ls`, `write`
- **System Calls**
  - A request for service that causes the normal CPU execution to be interrupted and control to be given to the OS; `read`, `write`
- **C Library Functions**
  - `printf`, `sqrt`

Most Common UNIX Commands

- Directory/file management
  - cd, pwd, ls, mkdir, rmdir, ==mv==, ==cp==, ==rm==, ==ln==, ==chmod==
- File viewing and selecting
  - ==cat==, ==more/less==, ==head, tail==, ==grep==, ==cut==
- Editors
  - vi, (x)emacs, pico, textedit
- Other useful commands
  - script, find, telnet, ==ssh==, and many more!
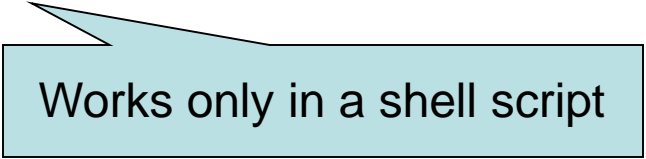
Shell Scripting

- All of the commands that are accessible from the shell can be placed into a shell "script"

  - Shell scripts are executed line by line, as if they were being typed in line by line

What Is The Shell?

- First UNIX "user interface"
- Text-based
- Provides access to all UNIX user-level programs
  - Start programs
  - Manage running programs (job control)
  - Connect together programs (pipes)
  - Manage I/O to and from programs (redirection)
  - Kill programs

Communicating between commands

- How can your commands (scripted or not) pass data?
  - Variables: set the var using it's name, $name to reference
    - home=1
    - echo $home
    - echo $SHELL

    Works only in a shell script

  - Command line parameters
    - Like "-size 1000000c"
    - In shell script, accessible as
      - $1, $2, $3, etc.
      - $# is the number of parameters
      - $*, $@ contain all parameters
  - Environment variables
    - HOME is defined as /home/brewsteb, for example
    - export HOME

Return Values Examined

- The `exit` and `return` shell commands returns its result to the ? variable

```
% echo hello world
hello world
% echo $?
0
% cd %=^2%21
=^2%21: No such file or directory.
% echo $?
1
```

Quick Shell demo

- Use semi-colon to put multiple commands on the same command prompt.

- Type the name of the shell to start a NEW shell.

- Echoing of variables
  - echo $HOME

- Return values of previously executed commands
  - echo $?

Communicating between commands

- Pipes
  - pass the standard output into the next command's standard input
    - `find $HOME -size +1000000c -ls | sort | head -5`

- Return values from programs
  - Exit value from the main function in C
    - exit
    - return
  - 0 indicates success, non-zero is an error code

Shell Script – High Level

- High level programming language
  - Variables, conditionals, loops, etc.
- Dynamic
  - No compiling, no variable declaration, no memory management == interpreted
- Powerful/efficient
  - Do lots with little code
  - Can interface with any UNIX program
- String oriented

When to use shell scripts?

- Automating frequent UNIX tasks
- Simplify complex commands
- For small programs that…
  - … you need to create quickly
  - … will change frequently
  - … need to be very portable
  - … you want others to see and understand easily
- As a glue to connect together other programs

simple shell script

```sh
#!/bin/sh
#          find_large.sh
# This shell script locates all files
# larger than a megabyte in my home directory
# and prints the top 5 largest in decreasing
# order.

find $HOME -size +1000000c -ls | sort -nrk 7 | head -5
```
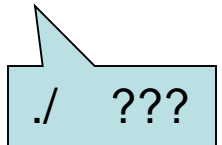
Using Shell Scripts

- Invoke the shell explicitly
  - /bin/sh find_large.sh

  Shell 'sh' chosen specifically

- The OS will invoke the shell specified in the file:
  - Add "`#!/bin/sh`" as the first line
  - `% chmod +x find_large.sh`
  - `% ./find_large.sh`

  ./ ???

Find_large.sh Output

```
: jasper{~/cs311}176% find_large.sh
2778522 6712 -rw-------  1 herlock  faculty   6861050
   Sep 28 08:39 /nfs/phantom/u10/herlock/mail/Sent
1254431 4896 -rwx------  1 herlock  faculty   4997632
   Sep 26 09:35
   /nfs/phantom/u10/herlock/thesis/thesis.doc
1254427 4856 -rwx------  1 herlock  faculty   4962816
   Sep 17 16:18
   /nfs/phantom/u10/herlock/thesis/old/thesis.doc
2778511 1752 -rw-------  1 herlock  faculty   1784763
   Sep 18 13:47 /nfs/phantom/u10/herlock/5102.tar.gz
1022355 1752 -rw-------  1 herlock  faculty   1784763
   Sep 25 15:04 /nfs/phantom/u10/herlock/tmp/5102.tar.gz
```

More Redirection

```
% cat > list
cookie
beefcake
apple
% cat list
cookie
beefcake
apple
% sort < list
apple
beefcake
cookie
% sort < list > sortedList
% cat sortedList
apple
beefcake
cookie
```

Here, input comes from stdin; you type each element in, hitting return each time, and terminate by typing ^d

sort takes as its input the file list

Give the file list to sort, and store it into the file sortedList

if

```
if command-list
then
    command-list
elif command-list
then
    command-list
else
    command-list
fi
```

while

```
while command-list1
do
    command-list2
done
```

for

- Doesn't use exit status

```
for i in a b c d
do
    printf "<%s>" $i
done
```

- Output

```
<a><b><c><d>
```

Quoting

- Quoting
  - Protect metacharacters
  - Groups into a single parameter
- Single quotes(' ')
  - Protects all metacharacters – no variable expansion
- Double quotes
  - Protects spaces
  - Variables are expanded
- Backslash
  - Protects any single metacharacter

Quoting Examples

- Somewhere, the variable '$' is defined as 10771

```
% printf '%s\n' "I have lots of $s"
s: undefined variable

% printf '%s\n' "I have lots of $$s"
I have lots of 10771s

% printf '%s\n' "I have lots of \$\$s"
I have lots of $$s

% printf '%s\n' 'I have lots of $$s'
I have lots of $$s
```

Efficient code creation

- Borrow and learn from other code
- Places to look:
  - /bin, /usr/bin, /sbin, lots of other places
  - cd /bin
    grep '/bin/sh' *

Error handling

- Correct syntax, but command fails
  - Shell will keep executing
- If you want the shell to exit if any commands have a problem:
  - Use -e with /bin/sh
- Most *signals* will kill script immediately
  - EX: control-C

Disastrous results?

```
#!bin/sh

cp thesis.doc thesis_current.doc
rm -f thesis.doc
```

One Fix

```
#!/bin/sh -e

cp thesis.doc thesis_current.doc
rm thesis.doc
```

Another Fix

```sh
#!/bin/sh

if cp thesis.doc thesis_current.doc
then
    rm thesis.doc
else
    echo "copy failed" 1>&2
    exit 1
fi
```
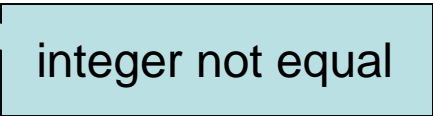
Yet Another Fix

- You can test for file existence, equality of strings, length, permissions, etc.

```
% test 1 -ne 2
% echo $?
0
% test 1 -ne 1
% echo $?
1
```

integer not equal

Yet Another Fix

```sh
#!/bin/sh

cp thesis.doc thesis_current.doc
if test $? -ne 0
then
    echo "copy failed" 1>&2
    exit 1
fi
rm thesis.doc
```

Trap command

- Use the trap command to catch signals and clean up

- Usage:

```
trap <code to execute> list of signals
```

Trap example

```
#!/bin/sh

TMP=ps$$

trap "rm -f $TMP; exit 1" INT HUP TERM

ps -u brewsteb | grep tcsh > $TMP
lpr $TMP
rm $TMP
```

Shell File Expansion

- Certain metacharacters are expanded and replaced with all files with matching names
- * - matches anything
- ? – matches any one character

Examples – file expansion

- List all files in the current directory whose names start with "cs311"
  - ls cs311*
- List all files in the current directory that have "cs311" somewhere in their name
  - ls *cs311*
- List all files in subdirectories that contain "cs311"
  - ls */*cs311*

End