EECS 470 Lab 5
Linux Shell Scripting

Department of Electrical Engineering and Computer Science
College of Engineering
University of Michigan

Friday, 7th February, 2014
Overview

Administrivia

UNIX
  Files
  Utilities
  Connecting Utilities

Bourne Again Shell
  Variables
  Flow Control
  Functions
  Globbing

Scripting

Assignment
Homework

- Homework 3 is due Wednesday, 12th February at the beginning of lecture

Projects

- Project 3 is due Sunday, 9th February at 9:00PM
- If you haven’t, you need to get started now

We are available to answer questions on anything here. Office hours can be found in the course google calendar.
UNIX

What is UNIX?

- Mainframe operating system
- Written by Dennis Ritchie, Ken Thompson and Rob Pike (among others) at Bell Labs
- The basis for many modern operating systems, e.g. Linux, BSD, Mac OSX

History of UNIX

- Written at Bell Labs in 1969
- First version of BSD is installed in 1974
- Last Bell Labs UNIX (Version 7) is published in 1979
- The GNU Project is started by Richard Stallman in 1984
- Linux Torvalds writes a monolithic kernel operating system in 1991
This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.”

– Douglas McIlroy

“Everything is a file descriptor or a process.”

– Linus Torvalds
What is a *file*?

- Anything referenced through a filesystem
- Anything with a file descriptor
Types of Files

- Regular – Anything not in one of the following categories
- Directory – Can contain other files and directories (read up on inodes sometime)
- Symbolic Link – A pointer to another file
- Pipe – Covered in a few slides
- Socket – Covered in EECS 482/489
Permissions

Table: UNIX permissions are represented with one bit for each permission in each category, e.g. 700.

<table>
<thead>
<tr>
<th>r</th>
<th>w</th>
<th>x</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Group</td>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Three Permissions

- Read
- Write
- Execute

Three Categories

- User
- Group
- Other
Utilities

What is a *utility*?

- A program used to process text streams/files
- Called from some command line.shell

Why do I care?

- Utilities form the basis of “Linux skills”
- Useful for automation
- Necessary for today’s lab
Navigation Utilities

pwd

- Description: print working directory – where you are
- Synopsis: pwd [OPTIONS]

ls

- Description: list directory contents
- Synopsis: ls [OPTIONS]... [FILE] ...

cd

- Description: change directory
- Synopsis: cd [OPTIONS] [PATH]
File Utilities

**cp**
- Description: copy files
- Synopsis: `cp [OPTIONS]... SOURCE DEST`

**mv**
- Description: move files
- Synopsis: `mv [OPTIONS]... SOURCE DEST`

**rm**
- Description: remove files
- Synopsis: `rm [OPTIONS]... FILE...`
diff

Description

- Shows the line-by-line differences between files
- Good for checking if your output is correct

Synopsys

- `diff [OPTIONS] FILES`

Examples

- `diff -uy ../project3_correct/writeback.out writeback.out`
- `vimdiff ../project3_correct/writeback.out writeback.out`
Alternatives to `diff`

Parsing output of `diff` is hard, so it might be useful to use some kind of “graphical diff” tool, like `vimdiff`, which opens up two files side-by-side in Vim showing their differences. Try it and you’ll see how much easier to parse this.
grep

Description

- Print lines matching a pattern

Synopsys

- grep [OPTIONS] PATTERN [FILE]

Examples

- grep '@@@' program.out
- ps -axfuw | grep "$USER" | grep "vcs"
Regular Expressions

- Really powerful/useful
- Complicated, and beyond the scope of this presentation
- Read up on them
  - at Wikipedia
  - in the grep manual
Description

- An interface to the on-line reference manuals (pager)

Synopsys

- `man PAGE`

Examples

- `man grep`
- `man diff`
Pager

Definition

- A program which allows browsing of large text files by breaking them into screen-sized chunks.

Examples

- less
- man
Other Utilities

These will be useful...

▶ cut
▶ touch
▶ tee
▶ xargs
▶ tail
▶ column
▶ find
▶ less

These are harder, but even more useful...

▶ sed
▶ awk
▶ patch
▶ vi(m)
▶ fmt
▶ tmux
Program Features

Methods of Communication

- Standard Text Streams
  - stdin
  - stdout
  - stderr

- Return value/code
Return Codes

What is a *return code*?

- The integer value a program returns (e.g. `return(0);`)
- Conventionally, returning zero indicates success, non-zero failure
- Specific values other than zero mean different things for different programs
Standard Text Streams

**stdin**
- The default input to a program
- From a keyboard by default

**stdout**
- The default output of a program
- To a display by default

**stderr**
- The default error output of a program
- To a display by default, requires additional effort to save
Connecting Utilities

Why do we want to connect utilities?

▶ Combination jobs without intermediate files
▶ e.g. take the diff of two different grep operations (what you need to do for today’s lab)

How can we connect utilities?

▶ Pipes
▶ Redirection
Pipe

What is a pipe?

- Connects the stdout of one program to the stdin of another
- Does not connect stderr

How do I use one?

- Call a program on one side of the | and then call another on the other side
- e.g. dmesg | less
Pipe: `xargs`

Problem
What if we want to pipe to utility that uses arguments instead of input?

Solution: `xargs`

`xargs` splits input into individual items and calls the program that is its argument once for each input.
Redirection

What is redirection?

- Allows for modification of the standard text streams
  - stdin is 0
  - stdout is 1
  - stderr is 2

- Several types:
  - 0< Use a file instead of the keyboard for stdin
  - i> Use a file instead of the terminal for the stream i
  - i>> Like i>, but append to a file instead of overwriting
  - i>&j Put stream i into the same place as stream j
Redirection by Example

Example

▶ ./vs-asm < test_progs/evens.s > program.mem

Example

▶ make | tee 2>&1 build.out

Example

▶ ./test 1>&2 2>&3
What is a *shell*?

- Before we had graphical environments, we had text shells
- Basically, an interpreter for commands, executing programs and saving information
- Possibly a Read-Execute-Print-Loop (REPL)
What is *BASH*?

▶ Stands for the Bourne Again Shell
▶ Created in 1989 by Brian Fox
▶ Default shell in most Linux distributions and Mac OS X

Why BASH?

▶ What I learned first
▶ Default in CAEN Redhat, finally
Warning

Everything after this slide will be specific to BASH. Other shells behave similarly, but not identically. If you want to use something else (e.g. ZSH, TCSH, etc.), please find other resources.
Variables

BASH Variables

- Store data
- Contain text, for the most part
- No type system

Syntax

- Assignment/Declaration
  - `variable=value`
- Referencing
  - `$variable`
  - `${variable}`
Variable Scope

Scope

- Variables exist inside the shell they’re in
- Unless exported
e.g. `export EDITOR=vim`
- This is very important in scripts, particularly the shell startup scripts (`.bashrc, .bash_profile`)
Special Variables

$#  The number of command line arguments
$0  The name of the script/function called
$1  The first argument to the script/function
$?  The return code of the last program run in this shell
$USER The current user
$HOME The user’s home directory
if/else

if [ var -eq "string" ];
then
    command1
elif [ var -eq "string2" ];
then
    command2
else
    command3
fi
case

```bash
case "$var" in
  val)
    command
  val2)
    command
  ( * )
    default
  ;;
esac
```
Flow Control

for

for file in ./__; do
    command $file
    command2
done

for (( a=1; a <= LIMIT; a++ )) do
    command
    command2
done
Conditionals

Testing

- Testing happens in []
- String tests are different than arithmetic tests
- Generally use \(-lt, -gt, -le, -ge, -eq, -ne\)
- Tests for files are special, e.g. \([ -x \ simv ]\) makes sure that the binary \texttt{simv} has execution permissions
Functions

- Packages of commands
- Arguments are referenced by position
- Useful for packaging up commonly reused bits

Syntax

```bash
function () {
    commands
}
```
File Globbing

What is *globbing*?

- File name wildcarding in the shell
- Expansion is done, and then passed to the command to be executed
- e.g. `test_progs/**.s`

What should I know globbing?

- Superior to parsing `ls` output in every way
- Can get more complicated, see this page of the BASH manual.
1. Write a series of shell commands into a text file
2. On the first line of the file, specify an interpreter
   e.g. #!/bin/bash
3. Name it something appropriate
   e.g. test.sh
4. Add execute permissions
   e.g. chmod +x test.sh
5. Run the script
   e.g. $ ./test.sh
Assignment is posted to the course website as Lab 5 Assignment.

If you get stuck...
- Ask a neighbor, quietly
- Put yourself in the help queue

When you finish the assignment, sign up in the help queue and mark that you would like to be checked off.

If you are unable to finish today, the assignment needs to be checked off by a GSI in office hours before the next lab session.