UNIX System Programming
Lecture 3: BASH Programming

- Outline
  - Filesystems
  - Redirection
  - Shell Programming

- Reference
  - BLP: Chapter 2
  - BFAQ: Bash FAQ
  - BMAN: Bash man page
  - BPRI: Bash Programming Introduction
  - BABS: Advanced Bash Scripting Guide

Lecture 3: BASH Programming
Filesystems

- Disk devices (/dev/hda1) contain raw disk data that is not normally accessed directly by the user.
  - A filesystem contains inode lists and data blocks.
  - An _____ structure (on disk) contains all file info except the file name (owner, group, perms, times, link count, indexes to the data blocks that make up the file).
  - A directory block has only filename and inode # pairs.
  - Use "ls -l" to display inode numbers. Use "find / -inum #" to find all filenames with inode number #.
  - Use mkfs (administrator) to create a filesystem (format):
    $ mkfs -t ext3 /dev/hdb1

Lecture 3: BASH Programming
Filesystems

- All files in a UNIX systems are arranged in a tree rooted at /.
  - No C:, D:, etc drives as in Windows.
  - Easily add disk space and allow files to maintain the same position in directory tree.
  - _______ is used to attach a filesystem to the tree

  $ mount -t ext3 /dev/hdb1 /home
  $ mount /dev/fd0 /mnt/floppy
  $ mount /dev/hdc /mnt/cdrom
  $ mount # Display mounted filesystems

Lecture 3: BASH Programming
Filesystems

- List of automatically mounted devices is in /etc/fstab
- Use df (diskfree) to display the amount of freespace on each filesystem.
- _______ (not unmount) detaches a filesystem
- Many systems have an automounter program running to automount removable media (floppies, CD-ROMs, ZIP drives, etc.)

Lecture 3: BASH Programming
Standard Directory Tree

- / root directory (don't confuse with /root)
- /bin essential utilities
- /lib essential libraries
- /sbin essential admin tools
- /etc configuration files
- /home contains user HOME directories
- /root root user HOME directory
- /dev device directory
- /var system files that change (log, spool files)
- /tmp for temp file usage (avail to all users)
- /usr/bin user applications
- /usr/lib user application libraries
- /usr/local contains applications added by local admin

Lecture 3: BASH Programming
Introduction to the Shell

- The shell is a command interpreter and a full-featured programming language. It is especially suited for system administration and file, directory and ______ management.
- Several different shells are available:
  sh, bash, csh, zsh, ksh
- Change your default shell to bash

  $ chsh -s /bin/bash
Lecture 3: BASH Programming
Our First Shell Script

A "Hello world" example:

```bash
$ cat > hello
#!/bin/bash
# Prompt user for name
echo -n "Enter your name: "
read name
echo Hello there $name!
$ chmod +x hello
$ ./hello
Enter your name: Elmer Fudd
Hello there Elmer Fudd!
```

Lecture 3: BASH Programming
Wildcards (globbering)

- Wildcard expansion is done by the _________ not the application. A * matches any character except a leading dot (.). A? matches a single character.

```bash
$ echo *
a.1 b.1 c.1 t2.sh test1.txt
$ ls *.sh
  t2.sh
$ ls [a-c]*
a.1 b.1 c.1
```

Lecture 3: BASH Programming
Redirection

- Every program automatically has three files/streams available: standard input, standard output, and standard error.
- In C, the FILE streams are stdin, stdout, and stderr.
- In C++, the IO streams are cin, cout, cerr.
- By default, they are connected to the keyboard, the display, and the display but they can be ___________.

This program uses the standard streams.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    string line;
    getline(cin, line);
    cout << "stdout: " << line << endl;
    cerr << "stderr: " << line << endl;
    return 0;
}
```

Lecture 3: BASH Programming
Redirection

- Now let's use redirection

```bash
$ ./redirect
Hi there
stdout: Hi there
stderr: Hi there
$ ./redirect < /etc/passwd
stdout: root:x:0:root:/root:/bin/bash
stderr: root:x:0:root:/root:/bin/bash
$ ./redirect < /etc/passwd > /dev/null
stdout: root:x:0:root:/root:/bin/bash
$ ./redirect < /etc/passwd 2> /dev/null
stdout: root:x:0:root:/root:/bin/bash
$ ./redirect < /etc/passwd 2> /dev/null > output.txt
stdout: root:x:0:root:/root:/bin/bash
```
Lecture 3: BASH Programming
Redirection

- To redirect both standard output and standard error:
  
  ```
  ./program &> output.txt
  ```

- Shell script input and output is redirected in a similar manner.

- To send a message to standard error from within a shell script:
  
  ```
  echo "usage: floof filename" >&2
  ```

Lecture 3: BASH Programming
Pipes

- A pipe (|) connects the standard output of the program on the left of the pipe to the standard input of the program on the right of the pipe.

- Here's an example that displays all usernames in alphabetical order:
  
  ```
  cut -d: -f1 /etc/passwd | sort | less
  ```

Lecture 3: BASH Programming
Built-in vs External

- Many commands are built into the shell: cd, for, while. Type “help” to see a list of built-in commands (or the bash man page.)

- Many other commands are programs: ls, vi, grep, perl.

- There are both built-in and external versions of many commands: echo, pwd, test. Use "type command" to see which is being used.

Lecture 3: BASH Programming
Shell Variables

- All shell variables are:
  
  ```
  file=/tmp/myfile
  list="apples oranges"
  echo $list > $file
  let a=3*4 # a will equal the string 12
  ```

- Variable substitution occurs within double quotes, but not within single quotes.

  ```
  echo "$a"     # displays 12
  echo '$a'     # displays $a
  echo \$a       # also displays $a
  ```

Lecture 3: BASH Programming
The Environment

- Each program owns an area of memory called the environment. Exported variables are copied from the parent's environment to the child's.

  ```
  $HOME  # User's home dir
  $PATH  # Colon separated directory list
  $TERM  # Terminal type
  $PS1   # Command prompt
  $0     # The name of the script
  $#     # Number of arguments
  $?     # Exit status of last program
  $$     # Process ID
  ```

Lecture 3: BASH Programming
Selection

- The commands if, while, until and case are available. if, while and until use program status to determine truth. 0 implies true/success, 1 implies false/error.

  ```
  if grep -qi fred /etc/passwd
  then
    # do something
  elif test -d /home/fred
  then
    # do something else
  else
    # do something
  fi
  ```
Lecture 3: BASH Programming Selection

- All programs return an exit status. The default exit status of a shell script is the exit status of the last command.
- The _____ command (built-in and external) can test file status and compare strings and numbers. The [ command is equivalent, but requires a final argument of ].

```bash
test -e ~/readme.txt  # does file exist?
test "$user" = fred   # string comparison
[ $count -le 10 ]    # test $count -le 10
```

Lecture 3: BASH Programming Selection

- Use “help test” to find out more about the test built-in command.
- The commands “true” and “false” have exit status of 0 and 1. They are useful in _____:

```bash
while true
    clear; ls -1; sleep 10
done
```

- Here is the C++ code for true and false:

```cpp
int main() { return 0; } // true
int main() { return 1; } // false
```

Lecture 3: BASH Programming Iteration

```bash
#!/bin/bash
# move html files to backup dir
for f in *
    do
        case $f in
            *
            .htm | *.html)
                cp $f ~/html_files
            ;;
            *)
                echo $f HAS NOT been backed up
            ;;
        esac
    done
```

Lecture 3: BASH Programming Iteration

```bash
#!/bin/bash
# loop through pos. parameters
for arg
    do
        lpr $arg
done
```

Lecture 3: BASH Programming Positional Parameters

- Shell variables $1, $2, ..., ${10}, etc. refer to script __________. $0 refers to script name.

```bash
if [ "$1" = -f ]    # Why "$1" and not $1?
    then
        # do something
    fi
```

- The set command can be used to assign to the positional parameters ($1, $2, etc).

```bash
set $(date)
today="$2 $3, $6"
echo $today
```

Lecture 3: BASH Programming Positional Parameters

- $* and $@ represent all of the positional parameters. $# is the number of parameters.

```bash
if [ $# -eq 2 ]
    then
        echo "usage: flowf arg1 arg2" >&2
        exit 1
    fi
```

- The shift command shifts all of the parameters left one position (2 -> 1, 3->2). ____ is useful for parsing parameters to distinguish between options and arguments.