Today

- More on Shell Scripting

Login
Open your favorite text editor
Open a terminal

Review

- What is Unix?
  - What are its goals?
- What is a shell?
- What are some of your favorite commands?
- What is the difference between an absolute path and a relative path?

Our Heroes: UNIX Developers

Ken Thompson Dennis Ritchie

Review: Why Unix?

- Open source = easier to study
  - Windows is proprietary & closed
  - OSX is proprietary and is built on top of Unix
- Historic: developed in the 60s & 70s
  - One of the oldest OS's in use today
- Most serious programmers & hackers know their way around Unix/Linux
- Linux is a Unix-like OS

Review: Unix Philosophy

- Make each program do one thing well
  - More complex functionality by combining programs
  - Make every program a filter
  - More efficient
  - Better for reuse
- Portability
- No GUIs
- Only error feedback

Quotes

- "Unix is simple. It just takes a genius to understand its simplicity." – Dennis Ritchie
- "UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things." – Doug Gwyn
- "Unix never says 'please'." – Rob Pike
- "Unix is user-friendly. It just isn't promiscuous about which users it's friendly with." – Steven King
- "Those who don't understand UNIX are condemned to reinvent it, poorly." – Henry Spencer

Review: What is a Shell?

- User interface to the operating system
- Command-line ***interpreter***
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- A program like any other
- Basic form of shell:

```
while <read command>:
  parse command
  execute command
```

hides details of underlying operating system

Directory Management

- How do you know what directory you're in?
- How do you make a new directory?
- How do you delete an empty directory?

Directory Management

- How do you know what directory you're in?
  - pwd
- How do you make a new directory?
  - mkdir
- How do you delete an empty directory?
  - rmdir

File Management Review

- How do you copy a file?
  - A directory and its contents?
- How do you move/rename a file?
- What is the short cut for the current directory?
- How do you delete a file?
- How do you delete a whole directory?

File Management Review

- How do you copy a file?
  - cp
  - A directory and its contents?
    - cp -r
- How do you move/rename a file?
  - mv
- What is the short cut for the current directory?
  - .
- How do you delete a file?
  - rm
- How do you delete a whole directory?
  - rm -r

Some useful file commands

- **grep** *pattern* [*file*]
  - select lines in the input that match *pattern*
- **head** *–n* [*file*]
  - show the first *n* lines of the input
- **tail** *–n* [*file*]
  - show the last *n* lines of the input
- **cp** $file_1$ $file_2$
  - copy $file_1$ to $file_2$
- **mv** $file_1$ $file_2$
  - move $file_1$ to $file_2$

Other File-Related Commands

Command	Purpose
`file`	Determine file type
`basename`	Strip directory and suffix from file names
`dirname`	Strip non-directory suffix from file name
`wc`	Print number of newlines, words, and bytes in files `-l` : lines `-m` : chars `-w` : words

Finding out about commands

Figuring out how to use a command

man *command*

"displays the on-line manual pages"

- Provides information about command options, arguments, return values, bugs, etc.

Managing Disk Space

Command	Purpose	Options
`du`	estimate file space usage	`-h` human readable `-s` summarize
`df`	report filesystem disk space usage	`-h` human readable

Many more options…
See man page

Managing Disk Space

- `du` Estimate file space usage (disk usage)
  - `-h` human readable format (e.g., MB, GB rather than KB)
  - `-s` summarize results for a directory

```
[sprenkle@perlman ~]$ du -s ~/public_html/
5680872 /home/faculty/sprenkle/public_html/

[sprenkle@perlman ~]$ du -sh ~/public_html/
5.5G    /home/faculty/sprenkle/public_html/
```


Managing Disk Space

- `df` File system disk usage
  - `-h` human readable format (e.g., MB, GB rather than KB)

```
[sprenkle@perlman ~]$ df -h
Filesystem             Size  Used Avail Use% Mounted on
devtmpfs               3.9G     0  3.9G   0% /dev
tmpfs                  3.9G   96K  3.9G   1% /dev/shm
tmpfs                  3.9G  900K  3.9G   1% /run
tmpfs                  3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda3              110G   15G   90G  14% /
tmpfs                  3.9G   48K  3.9G   1% /tmp
/dev/sda1              477M  153M  296M  35% /boot
tmpfs                  785M   16K  785M   1% /run/user/42
tmpfs                  785M     0  785M   0% /run/user/205
terras:/exports/home   248G  170G   66G  73% /csdept/home
terras:/exports/local   38G  8.9G   28G  25% /csdept/local
```


USEFUL SHORTCUTS

Useful Shortcuts

- Up arrow
- !command-prefix
  - ! = bang
  - Repeat most recent command that begins with prefix

Useful Shortcuts: {}

- Examples:
  - `mv file{,.bak}`
    - Expands to
      `mv file file.bak`
  - `tar cfz myDir{.tar.gz,}`
    - Expands to
      `tar cfz myDir.tar.gz myDir`
  - `cp index.{html,php}`
    - Expands to
      `cp index.html index.php`

ENVIRONMENT

The PATH environment variable

- Colon-separated list of directories
- Non-absolute pathnames of executables are only executed if found in the list
  - Searched left to right
- Example:

```
$ example.sh
-bash: example.sh not found
$ PATH=$PATH:.
$ example.sh
hello!
```


Shell Variables

- Shells have several mechanisms for creating variables. A variable is a name representing a string value. Example: PATH
  - Shell variables can save time and reduce typing errors
- Allow you to store and manipulate information
  - Ex: `ls $DIR > $FILE`
- Two types: local and environmental
  - Local are set by the user or by the shell itself
  - Environmental come from the operating system and are passed to children

Shell Variables

- Syntax varies by shell
  - `varname=value # sh, ksh, bash`
  - `set varname = value # csh`
- To access the value: **`$varname`**
- Turn local variable into environment:
  - All child processes from this terminal
  - **`export varname # sh, ksh, bash`**
  - **`setenv varname value # csh`**

Environmental Variables

Name	Meaning
$HOME	Absolute pathname of your home directory
$PATH	A list of directories to search for
$MAIL	Absolute pathname to mailbox
$USER	Your user name
$SHELL	Absolute pathname of login shell
$TERM	Type of terminal
$PS1	Prompt

To view all shell variables, `set`

My PATH Variable

- In my `.bash_profile`:
  - `PATH=$PATH:$HOME/bin`
- What does the above line mean?
- What is the result?

Setting Environment Variables

- You can set environment variables in your `~/.bash_profile` file
- Open `~/.bash_profile` using jedit or emacs
- Create a new variable:
  - `CS330=/csdept/local/courses/cs330`
- Export the variable
  - `export CS330`
- In terminal, run the `source` command to load your new profile
  - `source ~/.bash_profile`
- Check that your new variable was created:
  - `echo $CS330`
- Use the variable
  - `cd $CS330`

SHELL SCRIPTING

What is a shell script?

- A *shell script* is a list of commands to be run by a shell
  - basically a program
  - uses shell commands instead of C or Java statements
- Why?
  - automate repetitious tasks
    - e.g.: testing a program on a large set of test inputs
  - package up commonly executed command sequences
  - create our own commands

Shell Scripting vs. [C/Python/Java] Programming

Advantages
Easy to work with/use other programs
Easy to work with directories, files
Easy to work with strings (easier than C, at least)
Good for prototyping

Shell Scripting vs. [C/Python/Java] Programming

Advantages	Disadvantages
Easy to work with/use other programs	Slower
Easy to work with directories, files	Not well-suited for algorithms and data structures
Easy to work with strings (easier than C, at least)	
Good for prototyping	

Scripts won't be long

In some ways, we'll love it; in some ways, we'll hate it.

Most Commonly Used Shells

- `/bin/sh`
  - Points to bash on our system
- `/bin/bash` Bourne Again Shell
- `/bin/csh` C shell
- `/bin/tcsh` Enhanced C Shell
- `/bin/ksh` Korn shell

Invoking a Script

- A script can be invoked as:
  - `sh scr_name [ arg … ]`
  - `sh < scr_name [ args … ]`
  - `path/scr_name [ arg …]`
    - Before running it, it must have execute permission:
      - `chmod +x scr_name`

Where `sh` is whatever shell you want

We'll typically use either the 1st or 3rd execution option and we'll use the `bash` shell

Creating and executing a shell script

1. Create a file, say "foo", containing the commands to be executed by the script
2. Execute the script:
   - invoke the appropriate shell with foo as argument, e.g.:
     - bash foo
     - csh foo
   - `chmod a+x foo`
     `./foo`

Writing Your First Bash Script

- **Bash**: **B**ourne-**a**gain **sh**ell
  - Unix shell and command language
- Open your favorite text editor
- Write a simple bash script:
  - In your favorite UNIX text editor, type

```
echo "Hello World"
```

  - and save as `hello.sh`
- Type `bash hello.sh` to run

Specifying the shell interpreter

```
#!/bin/csh
```

- `#!` at the beginning of a file indicates that it is a script
- The rest of the first line specifies (absolute) path to the interpreter
- What if the interpreter is at a different location on a different system?
  - use **/usr/bin/env** to improve portability:

```
#!/usr/bin/env csh
```

  - Finds the first csh on a user's path, $PATH

Example Bash Script

- With little background info, tell me what these scripts do

Comments

- Comments begin with an #
- Comments end at the end of the line
- Comments can begin whenever a token begins
- Our text editors should help you with syntax highlighting
- Examples:

```
# This is a comment
# and so is this
grep foo bar # this is a comment
grep foo bar# this is not a comment
```


Variables

- To set:
  `name=value` ← Notice no spaces around =
  - Variables are *untyped*
- Read: `$var` or `${var}`
- Variables can be local or environment
  - Environment variables are part of UNIX and can be accessed by child processes
- Turn local variable into environment var:
  `export variable`

Variable Example

```
#!/bin/sh

MESSAGE="Hello World"
echo $MESSAGE
echo '$MESSAGE'
echo "$MESSAGE"
```

Prints variable
Prints literally
Prints variable

variable.sh

Using Environment Variables

```
#!/bin/bash

echo I am $USER
echo "I live at $HOME"
```

Both statements would work either with or without quotes

env_var.sh

Shell Scripting Practice

- Problem:
  - Iterate over the files in a given directory (as CL arg):
  - for each file, express its size in human-readable form (i.e., using KB and MB where appropriate)
    - functionality similar to ls –lh
- Requirements:
  - list files in a directory
    - [valid directory? ⇒ if-then-else]
  - iterate over these files
  - obtain the size of each file & display
    - Do not display the full ls –lh listing, just the size

Assignment 0a

- Write a bash script that takes one or more file names on the command line and prints out:
  - The name of the file
  - The number of lines & words in the file (NOT characters)
  - A list of all the words in the file and how many times they occur (ideally, the words should all be lower cased)
- Hint: you you will probably need to use/learn about pipe, loops, variables, echo, wc, cat, tr, sort, and uniq.

Due Friday, before class