Review: Unix Commands

- We talked a lot about security and file-related commands
  - Tell me about them!
- How do we redirect output from the terminal to a file?

Today

- More file system commands
- Process commands

Tree Walking

- How can do we find a set of files?
- One possibility:
  - `ls –lR /`
- What about
  - All files below a given directory in the hierarchy?
  - All files since Jan 1, 2017?
  - All files larger than 10K?

`find` utility

- **`find`** *<pathlist> <expression>*
- **`find`** recursively descends through *pathlist* and applies *expression* to every file
- *expression* can be:
  - **`-name`** *pattern*
    - *true* if file name matches pattern. Pattern may include shell patterns such as *, must be in quotes to suppress shell interpretation
      - `find / -name '*.java'`
      - `find ~ -name "*.py"`
  - …

What do these commands do?

find utility (continued)

- **-perm** [+-]*mode*
  - Find files with given access mode, mode must be in octal. Eg: **find . -perm 755**
- **-user** *userid/username*
  - Find by owner *userid* or *username*
- **-atime n**
  - File was last accessed n*24 hours ago. When find figures out how many 24-hour periods ago the file was last accessed, any fractional part is ignored
  - To match -atime +1, a file has to have been accessed at least two days ago.
- **-size** *size*
  - File size is at least *size*
- *many more…*

find: logical operations

Logical Operation	Functionality
! *expression*	returns the logical negation of expression
op1 -a *op2*	matches both patterns *op1* and *op2*
op1 -o *op2*	matches either *op1* or *op2*
( )	group expressions together

find: actions

- **-print** prints out the name of the current file (default)
- **-exec** *cmd*
  - Executes *cmd*, where *cmd* must be terminated by an escaped semicolon (**\;** or **';'**)
  - If you specify **{}** as a command line argument, it is replaced by the name of the current file just found
  - **exec** executes *cmd* once per file
  - Example:
    - **find . -name "*~" -exec ls -l {} \;**

What does this command do?

find Examples

- Find all files beneath home directory beginning with f
  - **find ~ -name 'f*'**
- Find all files beneath home directory modified within last 24 hours
  - **find ~ -mtime 0**
- Find all files beneath home directory larger than 10K
  - **find ~ -size 10k**
- Count words in files under home directory
  - **find ~ -exec wc -w {} \;**
- Remove core files
  - **find / -name core -exec rm {} \;**

Practical Example

- Problem opening Firefox "another session is already running"
- Solution: need to remove the "lock" files in your `~/.mozilla` directory
- But where are those files?
- And how do you delete them?

Practical Example

- Problem opening Firefox "another session is already running"
- Solution: need to remove the "lock" files in your `~/.mozilla` directory
- But where are those files?
  - Try: `find ~/.mozilla –name "*lock*"`
- And how do you delete them?
  - `find ~/.mozilla –name "*lock" –exec rm {} \;`

`diff`: comparing two files

- **`diff`**: compares two files and outputs a description of their differences
  - Usage: **`diff`** [*options*] *file1 file2*
  - **`-i`** : ignore case
  - `-u` : human readable
  - `-bB` : ignore white space

apples oranges walnuts	apples oranges grapes

```
$ diff list1 list2
3c3
< walnuts
---
> grapes
```


Other file comparison utilities

- cmp
  - Tests two files for equality
  - If equal, nothing returned. If different, location of first differing byte returned
  - Faster than **`diff`** for checking equality
- comm
  - Reads two files and outputs three columns:
    - Lines in first file only
    - Lines in second file only
    - Lines in both files
  - Must be sorted
  - Options: fields to suppress ([-123])

CONTROL-COMMANDS

Control-Commands

Control +	Function
c	Interrupt or break job; stops printing and returns to UNIX
z	Suspend current job bg to run in background
h	Erase or backspace character
s	Freezes screen
q	Unfreezes screen
u	Erase everything on line before this
w	Erase previous word
k	Erase remainder of line

PROCESSES

Unix Processes

- **Process**: An entity of execution
- UNIX can execute many processes simultaneously
- Creation of a process
  - A unique process id (pid) is assigned to the new process
  - Inherit, create, or initialize other data structures (e.g., file tables, I/O table, etc.)

Background Jobs

- By default, when executing a command in the shell, the shell will wait for the command to exit before printing out the next prompt
- Trailing a command with & allows the shell and command to run simultaneously

```
[sprenkle@fred ~]$ jedit &
[1] 7001
```

pid

Ending a process

- When a process ends, there is a return code (an integer) associated with the process
  - 0 means success
  - >0 represent various kinds of failure, up to process

Process Information Maintained

- Working directory
- File descriptor table
- Process id
  - number used to identify process
- Process group id
  - number used to identify set of processes
- Parent process id
  - process id of the process that created the process
- Umask
  - Default file permissions for new file

Process Information Maintained

We haven't talked about these yet:

- Effective user and group id
  - The user and group this process is running with permissions as
- Real user and group id
  - The user and group that invoked the process
- Environment variables

ps

- Report a snapshot of the current processes
- By default, just displays processes in the current terminal
  - Columns by default: PID, TTY, TIME, and CMD
- Accepted options:
  - UNIX options, which may be grouped and must be preceded by a dash
  - BSD options, which may be grouped and must not be used with a dash
  - GNU long options, which are preceded by two dashes

ps Examples

Command	Meaning
ps -e	See every process on the system
ps -ef	See every process on the system, in full listing
ps ax	See every process on the system
ps -ejH	See a process tree

Pipe through `more`

Process Subsystem Utilities

Utility	Functionality
`top`	Monitors tasks
`kill <pid>`	Terminate a process Use `-9` if bugger won't die
`nohup <cmd>`	Makes a command immune to hangup and terminal signal
`sleep <#>`	Sleep in seconds
`nice <cmd>`	Run processes at a low priority

PROCESS ENVIRONMENT

Environment of a Process

- A set of key-value pairs associated with a process
- Keys and values are strings
- Passed to children processes
- Cannot be passed back up
  - i.e., what you do in the child doesn't affect parent
- Common examples:
  - **PATH**: Where to search for programs
  - **TERM**: Terminal type

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!
```


Having . In Your Path

```
$ ls
foo
$ foo
sh: foo: not found
```

```
$ ./foo
Hello, foo.
```

- What **not** to do:

```
$ PATH=.:$PATH
$ ls
foo
$ cd /tmp/
$ ls
Congratulations! Your files have been removed
and you have just sent email to Steve
challenging him to a fight.
```


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`

Setting Environment Variables

- You can set environment variables in your `~/.bash_profile` file
- Open `~/.bash_profile` using jedit or emacs or some text editor
- Create a new variable:
  - `CS397=/csdept/courses/cs397`
- Export the variable
  - `export CS397`
- In terminal, run the `source` command to load your new profile
  - `source ~/.bash_profile`
- Check that your new variable was created:
  - `echo $CS397`
- Use the variable
  - `cd $CS397`

Bash's Configuration Files

File Name	Purpose
.bash_profile	Read and executed by Bash every time you log into the system
.bashrc	Read and executed by Bash every time you start a subshell
.bash_logout	Read and executed every time a login shell exits

Open your .bash* files in jedit
Notice what each file contains

ALIAS

- Allow you to rename commands or type something simple instead of a list of options
- Can be defined on the command line, in `.bash_profile`, or in `.bashrc`
- To see all defined aliases
  - `alias`
- To see the definition for an alias
  - `alias name`
- To create an alias
  - `alias name=command`

Create a new ALIAS

- Open `~/.bashrc` and `.bash_profile`
- Move your definition of `CS397` and its export from `.bash_profile` to `.bashrc`
- Add an alias called `cd397` (or something easy to remember) that cds to the CS397 directory

Deleting an ALIAS

- `unalias name`
- Just for the current shell/session

Assignment 1

- More practice, due Friday