Objectives

- UNIX File Management Commands
- UNIX Process Management Commands

Review

- What is are some components of the UNIX philosophy?
- What is a shell?
- What is the syntax of a UNIX command?
- What is the main security mechanism in UNIX?

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

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

Example of Simple Command

```
$ ls -l /bin
-rwxr-xr-x  3 root   root   63216 Sep 7  2006 zcat
$
```

prompt *command* *arguments*

- Execute a basic command
- Parsing into command and arguments is called splitting

Popular Success

Assignment 1 Feedback?

MORE FILE COMMANDS

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

Try Out These Examples

- `echo $HISTFILE`
- `file $HISTFILE`
- `dirname $HISTFILE`
- `basename $HISTFILE`
- `wc $HISTFILE`
- `wc –l $HISTFILE`

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@pascal ~]$ du -s ~/public_html/
785220 /home/faculty/sprenkle/public_html/

[sprenkle@pascal ~]$ du -sh ~/public_html/
767M  /home/faculty/sprenkle/public_html/
```

- Try out on your cs112 directory

Managing Disk Space

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

```
[sprenkle@hopper ~]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              48G  5.7G   40G  13% /
/dev/sda4             163G  109G   46G  71% /hopper1
/dev/sdb2             113G   58G   49G  55% /hopper3
/dev/sdb1             114G   68G   41G  63% /hopper2
/dev/sda1              99M   17M   77M  18% /boot
tmpfs                 1.5G     0  1.5G   0% /dev/shm
pascal:/exports/home  193G   72G  111G  40% /csdept/home
pascal:/exports/local  49G  3.0G   43G   7% /csdept/local
```


Timing Commands

- Often, you want to record when something happened or how long something takes
- `date`
  - Prints out system's current time
  - Lots of formatting options
    - Example: `date +'%A, %B %d, %Y'`
- `time <simple command>`
  - Measures command's resource use

USEFUL SHORTCUTS

Useful Shortcuts

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

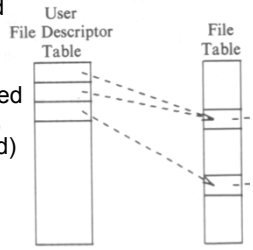
Useful Shortcuts: {}

- 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`

FILE SYSTEM INTERNALS

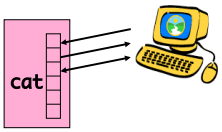
The File Descriptor Table

- Each process contains a table of files it has opened
- Inherits open files from parent
- Each open file is associated with a **number** or **handle**, called a **file descriptor** (fd)
- Each entry of this table points to an entry in the *open file table*
- Always starts at 0



Standard in/out/err

- The first three entries in the *file descriptor table* are special by convention:

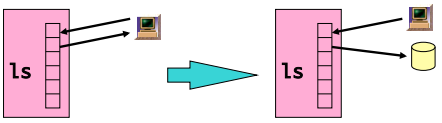


  - Entry 0 is for *input*
  - Entry 1 is for *output*
  - Entry 2 is for *error* messages

- What about reading/writing to the screen?

Redirection

- Before a command is executed, the input and output can be changed from the default (terminal) to a file
  - Shell modifies file descriptors in child process
  - The child program knows nothing about this



Redirection of input/ouput

- Redirection of output: >
  - Example:`$ ls > my_files`
  - Can save output from one of your programs
- Redirection of input: <
  - Example: `$ wc < input.data`
- Append output: >>
  - Example: `$ date >> logfile`
- Bourne Shell derivatives: `fd>`
  - Example: `$ ls 2> error_log`

Redirecting Output

- Save output from a program
  - `> java OlympicScore > score.out`
  - Redirected `stdout` to `score.out`
  - `stderr` would still go to terminal
- To redirect `stderr` to file as well
  - `> java OlympicScore >& score.out`

UNIX Command & Redirection Practice

- My research: analyze www access logs
- When an access log file gets too long or it's been a week, copied to access_log.1
  - Other files "bumped up" or deleted

```
[root@servo httpd]# ls -l access_log*
-rw-r--r-- 1 root root  213415 Apr 21 15:23 access_log
-rw-r--r-- 1 root root  679283 Apr 19 03:59 access_log.1
-rw-r--r-- 1 root root 1127828 Apr 12 04:01 access_log.2
-rw-r--r-- 1 root root  977639 Apr  5 03:43 access_log.3
-rw-r--r-- 1 root root  713767 Mar 29 04:01 access_log.4
```

- How can I put all the access logs in one file?
  - Is there anything else you need to know about these files?

UNIX Command & Redirection Practice

```
[root@servo httpd]# ls -l access_log*
-rw-r--r-- 1 root root  213415 Apr 21 15:23 access_log
-rw-r--r-- 1 root root  679283 Apr 19 03:59 access_log.1
-rw-r--r-- 1 root root 1127828 Apr 12 04:01 access_log.2
-rw-r--r-- 1 root root  977639 Apr  5 03:43 access_log.3
-rw-r--r-- 1 root root  713767 Mar 29 04:01 access_log.4
```

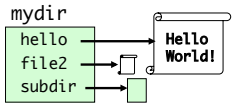
- One solution:
  - `cat access_log* > all_access.log`
- Better solution to preserve order:
  - `cat access_log.4 access_log.3 … access_log > inorder_access.log`

Want an easier way …

LINKS

Links

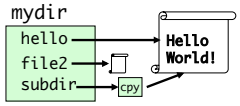
- Directories are lists of files and directories
- Each directory entry *links* to a file on the disk



- Hard links: Two different directory entries can link to the same file
  - Essentially gives same file another name
  - In same directory or across different directories
  - Cannot make a hard link to a directory

Links

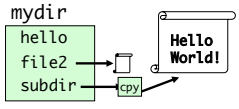
- Directories are lists of files and directories
- Each directory entry *links* to a file on the disk



- Two different directory entries can link to the same file
  - In same directory or across different directories
- Moving a file does not actually move any data around
  - Creates link in new location
  - Deletes link in old location
- **`ln`** command: `ln <target> <dest>`

Links

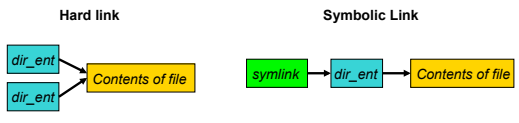
- Directories are lists of files and directories
- Each directory entry *links* to a file on the disk



- Two different directory entries can link to the same file
  - In same directory or across different directories
- Moving a file does not actually move any data around
  - Creates link in new location
  - Deletes link in old location
- **`ln`** command: `ln <target> <dest>`

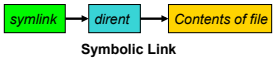
Symbolic links

- **Symbolic** links are different than regular links (often called **hard links**)
  - Created with `ln -s`
- Can be thought of as a directory entry that points to the ***name*** of another file



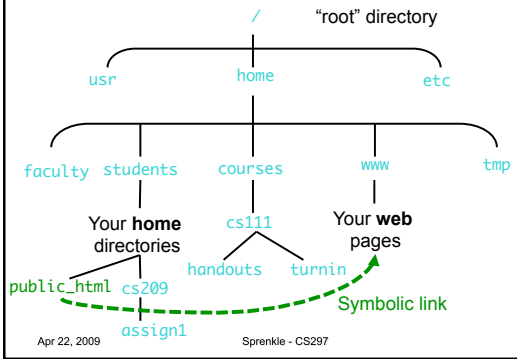
Symbolic links

- **Symbolic** links are different than regular links (often called **hard links**)
  - Created with `ln -s`
- Can be thought of as a directory entry that points to the ***name*** of another file
- Does not change link count for file
  - When original deleted, symbolic link remains
- They exist because
  - Hard links don't work across file systems
  - Hard links only work for regular files, not directories



Symbolic Link

(Partial) Linux File System



Practice

- Create a symbolic link to your `turnin` directory in your home directory

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, 2009?
  - 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 '*.c'`**
    - **`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 . 755`**
- **`-type`** *ch*
  - Find files of type *ch* (*c*=character, *b*=block, *f* for plain file, *d* = directory, etc.) Ex: **`find /home -type f`**
- **`-user`** *userid*/*username*
  - Find by owner *userid* or *username*
- **`-group`** *groupid*/*groupname*
  - Find by group *groupid* or *groupname*
- **`-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 rm "{}" ";"`

What does this command do?

find Examples

- Find all files beneath home directory beginning with f
  - `find ~ -name 'f*' -print`
- Find all files beneath home directory modified within last 24 hours
  - `find ~ -mtime 0 -print`

`-print` happens by default

- Find all files beneath home directory larger than 10K
  - `find ~ -size 10k -print`
- Count words in files under home directory
  - `find ~ -exec wc -w {} \; -print`
- Remove core files
  - `find / -name core –exec rm {} \;`

Practical Example

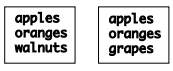
- Problem opening Firefox "another session is already running"
- Solution: need to remove the "lock" files in your `~/.mozilla` directory
- But where are those files?
- And how do you delete them?

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




```
$ diff test1 test2
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])

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 and initialize other data structures (file tables, I/O table, etc.)

Background Jobs

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

```
[sprenkle@hopper ~]$ /bin/sleep 10 &
[1] 7001
```

pid

Ending a process

- When a process ends, there is a return code associated with the process
- This is a integer
  - 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

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

Terminal Commands

- Ctrl-h : Erase or backspace character
- Ctrl-c : Interrupt or break character; stops printing and returns to UNIX
- Ctrl-z : Suspend current job
- Ctrl-s : Freezes screen
- Ctrl-q : Unfreezes screen
- Ctrl-u : Erase everything before this
- Ctrl-w : Erase previous word
- Ctrl-k : Erase remainder of line

Terminal Commands

Control +	Function
c	Interrupt or break job
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

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
  - Meaning, 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=.:/bin
$ ls
foo
$ cd /tmp/
$ ls
Congratulations, your files have been removed
and you have just sent email to Prof. Korn
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
- Create a new variable:
  - `CS297=/home/courses/cs297`
- Export the variable
  - `export CS297`
- In terminal, run the `source` command to load your new profile
  - `source ~/.bash_profile`
- Check that your new variable was created:
  - `echo $CS297`
- Use the variable
  - `cd $CS297`

Assignment 2

- Due Friday
- Play with commands learned more