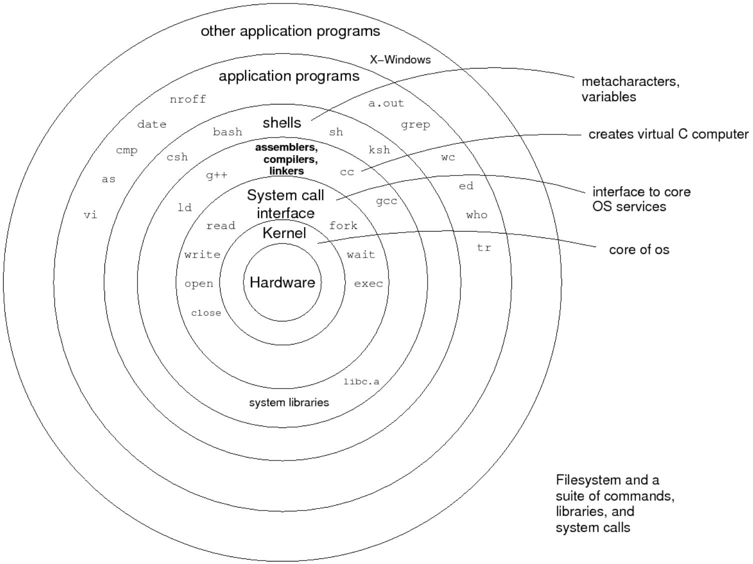
Lab 2: Linux/Unix shell

Comp Sci 1585
Data Structures Lab:
Tools for Computer Scientists

MISSOURI S&T | Computer Science

Outline

What is a shell?


other application programs
X–Windows
application programs
nroff
a.out
date
grep
cmp
wc
as
ed
vi
who
tr
shells
bash
sh
csh
ksh
assemblers, compilers, linkers
g++
cc
gcc
ld
System call interface
read
fork
write
wait
open
exec
close
Kernel
Hardware
libc.a
system libraries
metacharacters, variables
creates virtual C computer
interface to core OS services
core of os
Filesystem and a suite of commands, libraries, and system calls

Conceptual Architecture of UNIX SYSTEMS

What is a shell?

- `login` is a program that logs users in to a computer.
- When it logs you in, `login` checks `/etc/passwd` for your shell.
- After it authenticates you, it runs whatever your shell happens to be.
- Shells give you a way to run programs and view their output.
- They also usually include some built-in commands.
- Shells use variables to track information about commands and the system environment.
- The standard interactive shell is `bash`:

BASH
THE BOURNE-AGAIN SHELL

- There are others, though, e.g., `zsh` and `fish` .

Outline

Outline

Navigating the filesystem

- `$ ls` **L**i**s**t files. You can give it a directory to list.
  - `$ ls -l` Display the output in a detailed **l**ist, one line per file.
  - `$ ls -h` Display file sizes in a **h**uman-readable format.
  - `$ ls -a` Display **a**ll files, including hidden ones.
- `$ pwd` **P**rint **w**orking **d**irectory.
- `$ cd DIRECTORY` **C**hange **d**irectory.
  - `$ cd` without a directory takes you `$HOME`.
  - `$ cd -` takes you to the previous directory you were in.

Outline

File and Directory Shortcuts

- `.` always refers to the directory you are currently in.
- `..` always refers to the parent of the current directory.
- `~` refers to your home directory.
- `/` refers to the root directory. Everything lives under root.
- Globs:
  - `*` matches 0 or more characters in a file or directory name
  - `?` matches exactly one character in a file or directory name
  - For example, `$ ls *.cpp` lists all your cpp files.

Outline

Rearranging files

- `$ mv SOURCE DESTINATION` **M**o**v**e (or rename) files.
  - `$ mv -i` **I**nteractively ask you before overwriting files.
  - `$ mv -n` **N**ever overwrite files.
- `$ cp SOURCE DESTINATION` **C**o**p**y files.
  - `$ cp -r` **R**ecursively copy directories, which is what you want to do.
- `$ rm FILE` **R**e**m**ove one or more files.
  - `$ rm -f` **F**orcibly remove nonexistent files.
- `$ mkdir DIRECTORY` **M**a**k**es a **dir**ectory.
  - `$ mkdir -p DIRECTORY/SUBDIRECTORY` Makes every missing directory in the given **p**ath

Outline

Looking at files

- `$ cat [FILE]` Print out file contents.
- `$ less [FILE]` Paginate files or STDIN.
- `$ head [FILE]` Print lines from the top of a file or STDIN.
- `$ tail [FILE]` Print lines from the end of a file or STDIN.
  - `$ tail -n LINES` Print LINES lines instead of 10.
  - `$ tail -f` Print new lines as they are appended (`$ tail` only).
- `$ sort [FILE]` **Sort**s files or STDIN.
  - `$ sort -u` Only prints one of each matching line (**u**nique).
  - Often paired with `$ uniq` for similar effect.
- `$ diff FILE1 FILE2` Shows **diff**erences between files.
  - **a/d/c** Added/Deleted/Changed.

Outline

Outline

Redirecting I/O

- Each program has three default I/O streams:
  - STDIN: input, by default from the keyboard (cin).
  - STDOUT: output, by default to the screen (cout).
  - STDERR: output, by default to the screen (cerr).
- We can redirect IO to or from files or other programs.
- `$ cmd1 | cmd2` Pipe STDOUT from `cmd1` into STDIN for `cmd2`.
- `$ cmd <input.txt` Funnel data from `input.txt` to STDIN for `cmd`.
- `$ cmd >output.txt` Funnel STDOUT from `cmd` into `output.txt`.
- Question: what do you think the following does?

```
$ cmd <input.txt >output.txt
```

Outline

STDERR redirection tricks

- `bash` uses `1` and `2` to refer to STDOUT and STDERR.
- `$ cmd 2> err.txt` Funnel STDERR from `cmd` into `err.txt`.
- `$ cmd 2>&1` Funnel STDERR from `cmd` into STDOUT.
- `$ cmd &> all-output.txt` Funnel all output from `cmd` into `all-output.txt`
- Common usage: `$ cmd &> /dev/null` dumps all output to the bit bucket.

Outline

Environment Variables

- Shells keep track of a lot of information in variables.
- `$ printenv` shows all the environment variables set in your shell
- `$ env` shows **exported** environment variables (variables that are also set in the environment of programs launched from this shell).
- `$ set` lets you set them
- `$ VAR="value"` sets the value of `$VAR`. (No spaces around the `=`!)
- `$ echo $VAR` prints the value of a variable in the shell.
- You can get environment variable values in C++ with getenv()

Useful variables

- `$PATH` Colon-delimited list of directories to look for programs in.
- `$EDITOR` Tells which editor you would prefer programs to launch for you.
- `$ ~/.bashrc` runs every time you start `bash`, so you can `export` customizations there.

Outline

Neat bash tricks

- Tab completion works for files and commands!
- History:
  - ↑/↓ scroll through history.
  - Ctrl+r searches backwards through history.
- `$ !!` holds the last command executed.
- `$ !$` holds the last argument to the last command.
- `$ alias sl=ls` runs `ls` when you type `sl`.

Outline

Processes

- `$ ps` **Process list.**
    - `$ ps aux` / `$ ps -ef` show lots of information about all processes.
    - `$ ps` has crazy whack options.
- `$ top` and `$ htop` give an interactive process listing.
- Job Control:
    - Start processes in the background: `$ command &`
    - If you have a command running in the foreground, you can stop it with Ctrl + z.
    - `$ fg` starts the last process in the foreground.
    - `$ bg` starts the last process in the background.
    - `$ jobs` shows your running jobs.
    - `$ fg %2` starts job 2 in the foreground.
    - `$ kill PID` Kills a process. (You can do `$ kill %1`!)
    - `$ killall command` Kills every process running `command`.

Outline

Last but not least: –help, -h, and man

- `$ COMMAND --help` or `$ COMMAND -h` often provide concise help
- `$ man COMMAND` opens a full manual listing for that command.
- q quits the manual.
- j/k scroll up and down a line.
- Space scrolls down one page.
- `/thing` within a `man` page, less, more, and Vim searches for things.
- n/N go to next/previous search result.
- `$ man man` gives you the manual for the manual!