An Introduction to Unix Power Tools

Randolph Langley

Department of Computer Science
Florida State University

August 27, 2008

Introduction to COP 4342, Fall

- History of Unix
- Shells: what they are and how they work
- Commands: built-in, aliases, program invocations, structural
- Tree-structured resources: processes and files
- Finding more information: `man`, `info`, and `Google`.

Unix is now more than 30 years old. It first began in 1969.

Great Reference on Unix origins:

The Evolution of the Unix Time-sharing System, Ritchie at http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

Original Unix Goals

- Simplicity
- Multi-user support
- Portability
- Universities could get source code easily
- Users shared ideas, programs, bug fixes

Unix Is Based on Collaboration

Rather than a *product* from a manufacturer, Unix began as a collaborative effort designed to let a small group of people work closely together

The development of early Unix was user-driven rather than corporate-driven

- The first meeting of the **Unix User Group** was in May, 1974; this group would late become the **Usenix Association**

Unix, Linux, and the BSDs

Note that Linux and the BSDs (FreeBSD, OpenBSD, NetBSD) now flourish in similiar "open source" environments (http://www.freebsd.org, http://www.openbsd.org, http://www.netbsd.org)

- Started at AT&T's Bell Labs, originally derived from MULTICS.
- Original hardware was a DEC PDP-7.

Filesystem: Close but different

The filesystem was hierarchical but did not have path names (i.e., there was no equivalent to path names such as `/etc/hosts`, it would just be `hosts`; directory information was kept in a special file called `dd`)

Original structure for processes in Unix:

- Parent first closed all of its open files
- Then it linked to the executable and opened it
- Then the parent copied a bootstrap to the top of memory and jumped into the bootstrap

Things have changed a lot, more for processes and less for filesystems.

Original structure for processes in Unix, cont'd:

- The bootstrap copied the code for the new process over the parent's code and then jumped into it
- When the child did an exit, it first copied in the parent process code into its code area, and then jumped back into the parent code at the beginning

Today the parent process does:

- `fork(2)` (to create a new child process)
- `exec*(2)` (to have the child process start executing a new program)
- `wait*(2)` (to wait on the child (or at least check on its status if non-blocking))

Three stages of engineering refinement

1. Clumsy but basically functional
2. Complex but reasonably functional
3. Elegant and highly functional

Linux is a complete, Unix-compatible operating system:

- Based on Linux Torvalds' kernel (he is still in charge of kernel development, though now many people work on the kernel)
- The Linux distribution on the linprog machines is Centos 5.2; it includes a full development environment, X-Windows, Perl, C, C++, Fortran, and whole lot more (a full install is 5 gigabytes)
- Linux is mostly POSIX.1 compliant

Ubiquity of Linux

Linux runs on a huge array of hardware, from IBM's biggest machines down to commodity routers such as the Linksys WRT54G

There are good reasons to prefer a command line interface over a graphical interface:

- Typing is faster than mousing
- Graphics are computationally expensive, terminal handling is computationally inexpensive
- Easy to automate command lines, especially by utilizing histories
- Unix tools are designed to act as filters

Unix, like most operating systems, can be view in layers:

- Kernel → Provides access to system resources, both virtual and physical
- Shell → Provides a means to start other processes via keyboard input and screen output
- Tools → The vast array of programs that you can run to accomplish tasks

Some definitions

- "executable" → A file that can be "executed" in an existing process. There are two types of executables: binary executables, which natively run on hardware, and "script" executables which first invoke an interpreter. Script executables generally are human-readable (though, for instance, Zend PHP scripts can be pre-compiled into a crude intermediate representation.)
- process → An activation of a program. Creating a new process is done by making a new entry in the process table (however, in Linux, a thread, which retains the execution context of the caller, also goes into the process table.)
- daemon → Generally a persistent process (or at least the child of a persistent process) that is usually intended to provide some sort of service.

- user shell → Provides an environment that accepts keyboard input and provides screen output in order to allow a user to execute programs.
- "built-in" command → A "built-in" command does not cause the execution of a new process; often, it is used to change the state of a shell itself.
- alias → An alias expands to another command
- variable → A way to reference state in a shell
- flag → A way to specify options on the command line, generally via either a single dash or a double dash

Characteristics of Filters

- Should read from stdin and write to stdout by default (though some older utilities require explicit flags).
- Generally, filters should not read configuration files but should instead take their input from stdin and look at the command line for options via command line “flags”.
- The output from one filter ideally should be easily readable by another filter.

Well-known shells

- bash
- sh
- ksh
- csh
- tcsh
- zsh

Unix Files

- Unix files normally follow the paradigm of a “byte-stream”
- Filenames may consist of most characters except the NUL byte and “/”
- They are case sensitive
- Periods are generally used for any filename extensions
- Filenames that start with a period are treatly somewhat differently
- Unix does not generally make automatic backups of files

Some popular filename "extensions"

- `.c .h` → C files
- `.pl .pm` → Perl files
- `.py .pyc` → Python files
- `.cpp .c++ .CC` → C++ files
- `.s` → assembly files
- `.o` → object file
- `.gz` → gzipped file
- `.rpm` → rpm file
- `.tar` → tarfile

Shell wildcards and globbing

- `*` → matches any string
- `?` → matches any one character
- `[ ]` → lets you specify a character class

Note: often, you can use "`[ ] [ ]`" to specify a match for "`]`" or "`[`"

Filesystems

- Directories are tree-structured
- / is the root of a filesystem (Unix uses the model of a single filesystem)
- `CWD` or "Current Working Directory" is the default directory for a process

Directories are just special files that contain pointers to other files (including other directories)

You can see the `CWD` for a process PID by doing

```
ls -l /proc/PID/cwd
```

which shows a soft link to the current working directory for process PID.

Filesystem paths

- In Unix, we use / to distinguish elements in a path
- Absolute paths start with / which means start at the root
- Relative paths start with any other character and are interpreted as being relative to the current working directory

More on paths

- “.” is a special path (actually in the filesystem) that points at the current directory
- “..” is a special path (actually in the filesystem) that points at the parent directory
- “ /” is often understood by a shell as the home directory of the current user
- “ username/” is often understood by a shells as the home directory of “username”

Listing files

- `ls` → show all of the non-dot files as a simple multicolumn listing
- `ls -l` → show a detailed listing, one line per file
- `ls -a` → include the dot files
- `ls -d DIRNAME` → just show the information about the directory and not its contents
- `ls NAME NAME ...` → show the named files (if they exist)

File permissions, user classes

- owner → Each file in the filesystem has an uid associated with it called the owner
- group → Each file in the filesystem also a gid associated with it called the group
- others → Refers to all others users

File permissions, rwx

- `r` → permission to read a file
- `w` → permission to write to a file
- `x` → permission to execute a file

Changing permissions with `chmod`

- Octal notation : `chmod 4755 /bin/ls`
- Symbolic notation : `chmod og+w /etc/hosts`

Removing files

- `rm FILENAME` removes the named files
- `rm -r DIRNAME` removes a directory, even if it has some contents
- `rm -f NAME` removes a file (if possible) without complaining or query
- `rm -i NAME` queries any and all removals before they are committed
- `rmdir DIRNAME` removes directory iff it is empty

Recovering files after deletion is generally very hard (if not impossible); if the filesystem is not quiescent, it becomes increasingly difficult to do

Manipulating files with `cp` and `mv`

- `cp FILE1 FILE2` copies a file
- `cp -r DIR1 DIR2` copies a directory; creates DIR2 if it doesn't exist otherwise puts the new copy inside of DIR2
- `cp -a DIR1 DIR2` like `-r`, but also does a very good job of preserving ownership, permissions, soft links and so forth
- `mv NAME1 NAME2` moves a file directory

Standard i/o

- Each process that starts on a Unix system starts with three active file descriptors: 0, 1, and 2
- 0 → is standard input, and is where a process by default expects to read input
- 1 → is standard output, and is where a process by default will write output
- 2 → is standard error, and is where a process by default sends error messages

Redirection

- You can use > and < to provide simple redirection
- You can be explicit in bash and provide the actual file descriptor number
- For instance, in bash you can do

  ```
  ls whatever 2 > /dev/null
  ```

  which will make any error message disappear just like the `-f` option in `rm`.
- You can use >> to append to a file

Displaying files

- `cat` → Lets you see the contents with no paging.
- `more` → Pages output
- `less` → Also pages output, will let you go backwards even with piped input
- `head` → Just show the first lines of a file
- `tail` → Just show the end lines of a file

`cat` has many interesting options, including `-n` which automatically adds numbers to lines.

Piping

- A pipe lets you join the output of one program to the input of another
- The `tee` program lets you split the output of one program to go to the input of a program and to stdout

Finding more information

- The `man` program is a great place to start. You can use `man -k KEYWORD` to search for information on a particular KEYWORD.
- The `info` program puts you in an emacs session, and can be quite useful.
- `Google` is a very good resource.

Programs that report on other programs: `whereis`, `whatis`, `which`

- `whereis` – for instance:

```
COP4342\$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

- `whatis` – for instance:

```
COP4342\$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

- `which` – for instance:

```
COP4342\$ which ls
/bin/ls
```

Other useful information

- `who` – shows a list of users:

```
COP4342$ who
langley  tty7         2008-08-20 08:48 (:0)
langley  pts/0        2008-08-22 11:10 (:0.0)
```

- `w` – shows a list of users and more:

```
COP4342$ w
 09:50:15 up 7 days,  1:03,  4 users,  load average: 0.04, 0.16, 0.12
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
langley  tty7     :0                20Aug08  0.00s 32:42   0.09s gnome-session
langley  pts/0    :0.0              Fri11    0.00s 21.78s 21.78s emacs -nw
```

- `tty` – find your “terminal”

```
COP4342$ tty
/dev/pts/1
```