Unix/Linux: History and Philosophy

History and Background

- Multics project
  - Multiplexed Information and Computing Service
  - Collaborative venture between General Electric, Bell Telephone Labs, and Project MAC of MIT
  - Design started in 1965 and completed in 1972
  - Goals of Multics
    * Simultaneous computer access to large community of users
    * Ample computation power and data storage
    * Easy data sharing between users, if desired
- Unix
  - Pun on the name Multics, was originally named UNICS
  - Originated as a research project in the AT&T Bell Labs in 1969
  - Designed by Ken Thompson to better play the space war game on a PDP-7 computer
  - Designed to be a small and simple operating system
  - Rewritten to run on a PDP-11 machine in 1970
  - In 1973, Dennis Ritchie and Thompson rewrote the kernel in C, creating the first portable operating system
- Linux
  - Created by Linux Torvalds to learn operating systems
  - Uses many applications developed by the Gnu project, hence the name Gnu/Linux

Why Unix/Linux?

- Conceptually simple
  - User is presented with many simple utilities, known as *tools*
  - A tool does only one thing, but does it well
- Consistent
  - Uniform interface for most of the commands (in the form of *pipe*-able commands or *filters*)
  - Similar functions on different objects handled in uniform manner
    * Files and I/O devices are treated in the same manner
- Extendible
  - Combine simple tools to build complex tools (again, filters)
  - Create custom environments through aliasing and window managers
    * Tailorability to individual preferences is there to make the machine work for the people and not vice versa
- Modular
  - Most of the applications are not part of the operating system, and hence, can be replaced at will
  - You have a choice of shells and other general purpose programs, and if you do not feel satisfied, you can easily write one of your own choice

  - A number of people have contributed to its development, both good and bad; this has also resulted into multiple ways to solve the same problem

Pros and cons of Unix/Linux

- Pros
  - Truly multitasking – Trivial to run multiple programs even from a dumb terminal
  - Truly multiuser – Even the PC-based Linux can handle multiple users simultaneously while running on a 386
  - Robust
    * It is rare to see system crashes caused by a user program
    * The memory allocated to a process is private to the process, that is, other programs cannot see the code and data for a given program unless special permission is granted
    * The files that belong to a certain user may not be visible to other users unless permission is granted
      · The superuser has permission to read all files
      · Superuser can be prevented from reading personal files by encrypting the files
  - Excellent programming environment
    * Designed by the hackers, for the hackers
    * A vast repository of development tools is standard on the system
    * Another large repository of tools is available from the Free Software Foundation under the Gnu project (`www.gnu.org`)
  - Portable – There is a version for almost every processor
  - Extensible
    * Easy to add new features
    * New commands can be added at the user level or system level, and integrate seamlessly with the existing commands
    * You can write shellscripts effortlessly to perform sets of commonly repeated commands
  - Easy to maintain – Older features can be easily replaced, or superseded
  - In existence for over 40 years
    * Has been constantly refined by a large community of users
    * Has been responsible for countless number of developments in the field of computing, including internet, email, and web browsers
    * Vast amount of literature available on every aspect of the system
    * Well understood to the point where you can get free technical help on the Internet for your problems
  - GUI vs. CUI – Working on the computer or working for the computer
- Cons
  - Designed by the hackers, for the hackers
  - Cryptic commands
    * Unix follows the philosophy that all commands should be two to three letters long
    * You may be able to correlate some commands with words, such as `ls` with 'list directory' but there are obscure commands such as `nm` which is used to see the contents of shared library
  - Too many commands, you never finish learning it
  - Too much material to learn
  - Constantly increasing number of additional utilities
  - Multiple commands to do the same thing

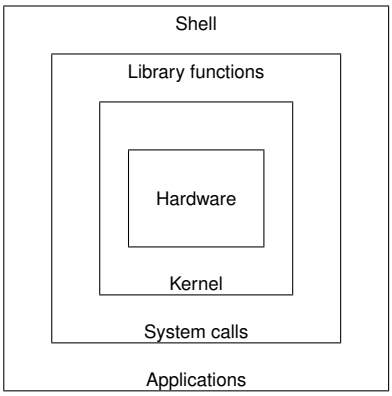
  - Too many different versions

"Unix was not designed to stop you from doing stupid things, because that would also stop you from doing clever things."

– Doug Gwyn

Structure of the system

- Structured as a nut



  - The innermost part of the operating systems is called the kernel
    * Kernel directly interfaces with the hardware of the system
    * Hardware can not be directly manipulated by the user
  - The outermost part of the operating system is called the shell
    * A user interfaces with the shell
    * User is free to select a shell of his/her own choice, from the variety of shells available on the system
- In between the shell and kernel, there is a large number of utilities that can be classified as development tools and applications
- The applications and development tools are invoked from the shell and may interact with shell (for user inputs) as well as kernel (for manipulating the hardware)

Shell and tools

- Shell is responsible for establishing all interaction with the user
  - Application programs communicate with the kernel under the control of shell
- Interprets and executes commands, as well as may provide job control
  - Parses the command line, breaking it into tokens separated by whitespace
  - Any metacharacters are evaluated by shell
- Treated as just another program

- Every user has a default shell assigned by the system administrator
  - Can be customized to user preferences by startup files
  - Enable history and aliases
- Major shells are classified as follows:

Shell name	Author	Command	Prompt
Bourne Shell	S.R. Bourne	`sh`	`$`
C Shell	Bill Joy	`csh`	`%`
tcsh	Bill Joy	`tcsh`	`%`
Korn Shell	David Korn	`ksh`	`$`
bash	Brian Fox	`bash`	`$`

- Built-in commands
  - An extremely small set of commands, compared to the number of utilities on the system
  - Some of the built-in commands provide a programming language like functionality and are useful in writing shell scripts
  - These commands provide the real functionality in the shell
- External commands
  - Act just like the built in commands
  - User may not be able to tell the difference
  - Newly added commands become part of the system

I/O redirection

- The I/O devices of concern at this point are the ones used for communication with the computer, e.g., monitor and keyboard
- Unix does not make a distinction between files and I/O devices
  - In effect, monitor and keyboard can be treated as files
  - Keyboard is an input file
  - Terminal monitor is an output file
- All devices classified to belong to one of the three standard devices, unless otherwise noted
  - `stdin` – Standard input device (keyboard)
  - `stdout` – Standard output device (monitor)
  - `stderr` – Standard error device (monitor)

  There is an important distinction between output and error devices that will become clear over time
- Redirection
  - Sending the output of one program to a file, instead of `stdout` or `stderr`
  - Receiving the input to a program from a file instead of `stdin`
  - Based on operators <, >, <<, >>, as well as pipes and `tee`

File system

- Hierarchy of stored user data files
  - Structured like an inverted tree, with the root at the top and branches growing downwards from it
  - Unix file system may look like that of Windows, except that it is much more sophisticated
- Manages the files and devices
  - All devices are part of the file system, in the directory `/dev`
- `root`
  - A special directory from which all the other directories branch out
  - Always at the top of the tree
  - Ancestor of every file
    - Can be used to uniquely identify a file, even if there is more than one file with the same name
- Directory
  - A branch in the tree
  - May be spawned off of another directory (parent-child relationship)
  - Current working directory
- Files
  - Leaves in the file system tree
  - Collection of bytes with no structure imposed by Unix
  - Files `.` and `..`
  - Regular files
- All directories and files belong to just a single file system
  - Disks are not special (as in Windows)
  - Unix user need not be concerned about which physical disk his/her files reside on
  - Each disk is *mounted* to a directory
    - Disks from one computer can be mounted to a directory on a different computer, without the user perceiving this
  - User can move across disks by simply changing directories
- Protecting files; access control

Basic shell commands

- Command structure
  - Each command in Unix is made of a single word, with no spaces
  - Virtually every command is in all lowercase letters (Unix is case-sensitive)
  - The command may be optionally followed by a set of parameters, called *tokens*
- Entering commands
  - Entered at the prompt
  - The format of each command is:

```
<command> [<option(s)>] [<argument(s)>]
```

where the items in square brackets are not always required

- Command may be
  - Name of an internal command (built into shell)
  - Name of a Unix utility
  - Name of a user-defined command
- Option part of the command
  - Exactly one character long
  - Each option preceded by the character −
  - May be followed by an argument for the option
  - Options with no argument can be grouped after a single − character
  - All options precede the command arguments
  - `--` may be used to indicate the end of options
- Arguments
  - Specific to the command
  - Could be file names
- Command may result into an error if it is not properly parsed
- Other command line features
  - The features described in the following table allow for special functions on the command line itself

Character	Example	Comment
`\`	`$ cp file1 \` `file2`	Continuation of a command on next line
`;`	`date; who am i`	Allows specification of more than one command on same line
whitespace	`cp file1 file2`	Delimiter to separate arguments
`|`	`cat file | less`	Combine two commands

  - The whitespace is any number of blanks (by pressing space bar) or any number of tabs (tab key)
- Special key sequences
  - Used to perform certain functions from the shell

Key	Name	Action
`^?` or `^H`	`erase`	Backup one space and delete character
`^U`	`kill`	Erase entire command from command line
`^C`	`intr`	interrupt/stop a currently executing command and return to prompt
`^Z`	`susp`	Suspend the currently running command It can later be restarted or killed

- Return value
  - Every command returns a value to indicate if it executed successfully
  - Typically, return value 0 indicates success
  - Available in bash shell variable `$?`

Getting help

"Acts oddly on nights with full moon."

`BUGS` section for `catman` from 4.2BSD Unix manual

- Unix provides an online reference manual
  - Same as the printed reference manual
  - Not tutorial in nature
  - Does not compare favorably with the help manuals on Windows but provides the most comprehensive information, especially as you move from one flavor of Unix to another
  - Divided into sections:

Section	Description
1	User commands
2	System calls
3	Subroutines
4	Devices (Special files and hardware)
5	File formats and configuration files
6	Games and demos
7	Miscellaneous: characters sets, filesystem types, etc
8	System administration and maintenance
l	Local
n	New

- The command to get help on a given section is `man` (for manual)
- You can search the manual pages by using the command `apropos` or using the option `-k` with the `man` command
- The following two are equivalent

```
man -k directory
apropos directory
```

- Common convention after man page entries
  - The number in parenthesis after the command refers to the section of the manual where the command is found
- You can get a one-line description of the command by typing the command `whatis` or by using the option `-f` with the `man` command