Introduction to Unix

Alark Joshi

Adapted from University Technology Services, Ohio State University

History of Unix

- 1960's – Multics Project (MIT, GE, AT&T)
- 1970's – AT&T Bell Labs
- 1970's/80's – UC Berkeley
- 1980's – DOS imitated most of Unix ideas
  - Commercial Unix fragmentation
  - GNU Project
- 1990's – Linux
- Since then Unix and Linux have become widespread and available from many sources

Unix Philosophy

- Multi-user and Multi-tasking
- Toolbox approach
- Flexibility/Freedom
- Conciseness
- **Everything** is a file
- File system has places, processes have life
- Designed by programmers for programmers

Structure of the UNIX OS

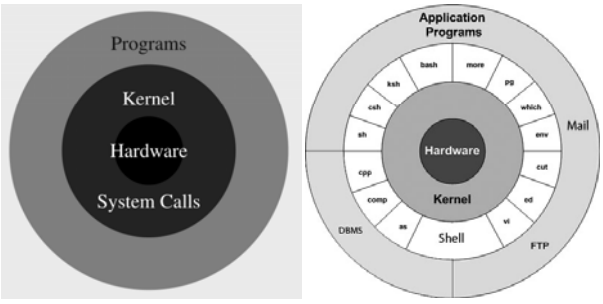
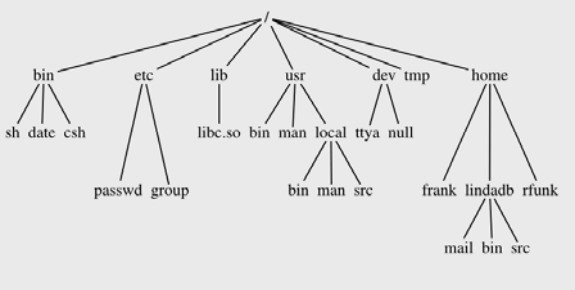


Image credits: http://www.tutorialspoint.com/images/unix_architecture.jpg

The File System



Unix Programs

- Shell is the command line interpreter
  - Just another program
- A program or command
  - Interacts with the kernel
  - Could be either:
    - Built-in shell command
    - Interpreted script
    - Compiled object code file

Unix Command Line Structure

- A command is a program that tells the Unix system to do something. It has the form

  `command options arguments`
- "whitespace" separates parts of the cmd line
- An *argument* indicates the input for the command to perform its action
- An *option* modifies the command, usually starts with a '-'

Getting Help in Unix

- man – On-line manual

```
% man command
% man –k keyword
```

Control Keys

- ^S – Pause display
- ^Q – Restart display
- ^C – Cancel operation
- ^U – Cancel / Clean line
- ^D – Signal End of File

List directory contents

```
ls [options] [arguments]
```

-a : list all files

-l : long listing (mode, link info, owner, size, last mod)

-g : Unix group (requires –l option)

List directory contents

- Each line (when using –`l` option of `ls`) includes the following:
  - Type field (first character)
  - Access permissions (characters 2-10)
    - First 3 – User/Owner
    - Second 3 – Assigned Unix group
    - Last 3 – Others/ Rest of the world

Permissions

- Designated as
  - r : read permission
  - w: write permission
  - x: execute permission
  - - : no permission

Commands to Change Permissions

- chmod – change file or directory access permissions
- chgrp – change the group of the file
- chown – change the owner of the file

Changing Permissions of a File

- `chmod [options] file`
- Using + and – with a single letter:
  - u: user owning file
  - g: those in assigned group
  - o: others
- `chmod u+w file`
  - gives the user (owner) write permission
- `chmod o-x file`
  - removes execute permission for others

Changing Permissions of a File

- `chmod [options] file`
- using numeric representations:
  - r = 4
  - w = 2
  - x = 1
  - Total: 7
- `chmod 777 filename`
  - gives user, group and others read, writer and exec permissions

Changing Permissions of a File

- `chmod 750 file`
  - gives the user read, write and execute
  - gives group members read, execute
  - gives other no permissions

Changing Permissions of a File

- `chmod 640 file`
  - gives the user read, write
  - gives group members read
  - gives other no permissions

Disk space commands

- `df [options] [directory]`
- Display free disk space

```
% df –k /
```

- `du [options] [directory]`

```
% du .
```

Status of Processes

- `ps [options]`

```
% ps
% ps –ef
% ps auxw
```

- Options vary with flavor of OS – see man pages

Terminate a process

- `kill [-signal] processID`
- `% kill –l`
  - displays the available kill signals
- `% kill -9 processID`
  - Last resort – “nuke” process

Program Locations

- `whereis [options] command`

-b : report binary files only

-m : report manual page files only

-s : report source files only

- Examples:
  - `% whereis mail`
  - `% whereis –b mail`
  - `% whereis –m mail`

Report the command found

- `which command`
- will report the name of the file that will be executed when the command is invoked
  - Full path name
  - Alias found first

Information about the Machine

- `hostname`
  - Reports the name of the machine that the user is logged into
- `uname [options]`
  - has additional options to print information about system hardware and software

Record your session

- `script [-a] filename`

-a : append content to a file

```
%script
(… commands)
% exit
% cat typescript
```

- typescript is the default name of the file used by script

Link to another file

- `ln [options] source target`

```
% ln –s chkit chkmag
```
  - symbolic link

```
% ln chkit chkmag2
```
  - hard link

- Hard links cannot link paths on different volumes or file systems.
- Symbolic links may point to any file or directory irrespective of the volumes on which the source and destination reside.
- Hard links always refer to an existing file, symbolic links may contain arbitrary text that doesn't point to anything.

Find Files

- `find directory [options] [actions] […]`

```
% find . –name ay –ls
% find . –newer empty –print
% find /usr/local –type d -print
```

Compression

- `compress [options] [file]`
- `zcat [file.Z]`
- `uncompress [options][file.Z]`

```
% compress logins.*
% zcat archive.Z | head
% uncompress logins. *.Z
```

- gzip/ gunzip often used too - .gz extension

Archive Files

- `tar [options] [directory/file]`
- Options:
  - c : create an archive
  - t : table of contents list
  - x : extract from archive
  - f file : Archive file is named file
  - v : verbose

Archive Files

```
% tar –cf logfile.tar logs.*
% tar –tf logfile.tar
% tar –xf logfile.tar
```

Find text in a File

- `fgrep [options] text [files]`
- Simplified version of grep
- fgrep is used to search of exact strings in text files
- Options:
  - i: ignore case
  - v: display online lines that **don't** match
  - n: display line number with matching line

Shells

- The shell sits between you and the operating system
  - Acts a command interpreter
  - Reads input
  - Translates commands into actions
- To see what your current login shell is:
  - echo $SHELL

Bourne Shell (sh)

- Great features for I/O Control – often used for scripts
- Not as well suited for interactive users
- Default prompt is $

C shell (csh)

- Uses C-like syntax for scripting
- I/O not as straightforward as *Bourne Shell*
- Nicer for interactive use
- Job control
- History
- Default prompt is %
- Uses a ~ symbol to indicate a home directory

Other Shells

- Based on Bourne Shells:
  - Korn shell (ksh)
  - Bourne-Again Shell (bash)
  - Z Shell (zsh) - Extended Bourne improvements, including some features of bash, ksh, and tcsh.
- Based on C-shell
  - T-C shell (tcsh)

Environment Variables

- DISPLAY
- EDITOR
- PAGER
- PATH
- TERM
- `csh setenv NAME value`
- `sh NAME = value; export NAME`

Shell Variables

- `csh set name = value`
- `sh name = value`

- These are used by the shell and shell scripts; not seen or used by external programs

Shell Startup

- The file .profile (sh) or .login (csh) is used at login to:
  - Set Paths
  - Define functions
  - Set terminal parameters (stty)
  - Set terminal type
  - Set default file permissions (umask)

.login and .cshrc

- .login runs only at login time
- Tells you whether you have mail
- Who else is online
- Configure terminal settings
- .cshrc runs whenever the shell starts
- set environment and shell variables
- set aliases

Changing your shell

- `chsh`
- `passwd –e /usr/local/bin/tcsh`
- The new shell must be the full path name for the shell on the system
- Shell paths in some cases:
  - Bourne: /bin/sh
  - Korn : /bin/ksh
  - C : /bin/csh
  - Zsh: /bin/zsh

Unix I/O

- I/O Redirection and piping
  - Output redirection to a file
  - Input redirection from a file
  - Piping
    - Output of one command becomes the input of a subsequent command

```
ls –l | wc
```

Standard File Descriptors

- stdin – Standard input to the program
- stdout – Standard output from the program
- stderr – Standard error output

- These are not called by name at shell prompt, but are often references by these names
- stdin – Normally from keyboard, can redirect
- stdout & stderr – Normally to the screen, can redirect

Redirection

- > Redirect Standard output to file
  command > outfile
- >> Append standard output to file
  command >> outfile
- < Input redirection from file
  command < infile
- | Pipe output to another command
  command1 | command 2

File Redirection

- To redirect stdout and stderr to separate files
- csh
```
% (command > outfile ) >& errfile
```
- sh
```
$ command > outfile   2 > errfile
```

Other Special Command Symbols

- ; command separator
- & run command in the background
- && run the following command only if previous command completes successfully
- || run the following command only if prev command does NOT complete successfully

Text Processing / Filters

- Filters perform some kind of a transformation on an input file to create the output

tr

- tr transliterates one set of characters into another
- Examples:
  - Convert lowercase letters to uppercase letters

```
tr a-z A-Z
```

  - Convert voxels to X

```
tr aeiou XXXXX
```

  - Caesar Cipher

```
tr A-Za-z D-ZA-Cd-za-c
```

uniq and sort

- `uniq` removes adjacent duplicate lines from its input

```
uniq file file.new
```

- `sort` orders the lines in a file; Default order is lexicographic
  - n option specifies numeric sort
  - kn sorts by the nth field (fields are space separated)
  - r sorts in reverse
    - Files/temp.c – sort temp.c and sort –n temp.c

Regular Expressions

- First described by Stephen Kleene
- Used for pattern matching in Unix utilities like grep and awk
- No standard notation

Elements of a RegEx

- Atoms – Characters that can be combined to make the pattern
- Concatenation – Sequence of atoms
- Alternation – choice between several patterns
- Kleene Closure (*) – 0 or more occurrences
- Positive Closure (+) – 1 or more occurrences
- Character classes
  - `e.g. tr a-z A-Z`

grep

- Find lines in a file that match a particular regular expression

```
grep [options] pattern [file/s]
```

- Options:
  - v : find lines that don't match the pattern
  - f : get the pattern from file that follows
  - i : ignore case
- egrep / fgrep have similar functionality with more power

Metacharacters used in RegEx

- . :any single character except newline
- ^ $: anchor to the beginning or end of line
- […] : match any character in the listed set; ranges like [a-z] are allowed
- [^…] : match any character not in listed set
- Any character that doesn't have a special meaning represents itself
- Backslash in front of a metacharacter represents that character

Combining regular expressions

- If r, r1 and r2 are regular expressions:
- r* : matches zero or more occurrences of r
- r+ : matches one of more occurrences of r (egrep only)
- r? : matches 0 or 1 occurrences of r (egrep only)
- r1r2 matches r1 followed by r2
- r1 | r2 matches either r1 or r1

grep Examples

- `grep "this" *.c`
  - Search for the keyword ***this*** in all .c files in that dir
- `grep -i "string" string.c`
  - Case insensitive searching
- `grep “boise.*state” testfile`
  - Display all the lines that contain *boise* with some text in between and end with *state*.
  - grep "fopen.*r" *.c

grep Examples

- `grep -iw “token" string.c`
  - Check for the complete word *token*
- `grep -A <N> “fopen" filename.c`
  - grep –A 3 “fopen” file.c (display matched line with 3 lines after match)
- `grep -B <N> “fopen" filename.c`
  - Display N lines before match
- `grep -C <N> “fopen" filename.c`
  - Display N lines before and after match

grep Examples

- `grep –r “fopen” *`
  - Recursive search in subdirectories
- grep –c “pattern” filename
  - Count the number of occurrences
- grep –n “pattern” filename
  - Show line numbers

grep Examples

- grep '\<c...h\>' /usr/share/dict/words
  - list of all five-character English dictionary words
- grep '\<c.*h\>' /usr/share/dict/words
  - selects all words starting with "c" & ending in "h“
- grep ‘*’ /etc/profile
  - To search for the asterisk character using ‘ ’

ed

- A line editor
- Can be run from a script
- Commands are a single letter
- Most commands can be preceded by a line number (a literal number, . for current line, $ for last line of the file), or a range of line numbers (n1, n2) or % for all lines in the file)
- In `vi`, you can use `ed` commands by typing a : followed by a command

sed

- sed is a filter that edits the lines of a file according to a set of one of more commands
- sed commands come from the ed editor
- Typical usage is:
  - sed 'commands' files
  - commands enclosed in single quotes
  - There can be any number of files. If files is empty, input comes from standard input (stdin)
  - Output goes to standard output (stdout)
- Commands applied to each line

sed

- sed [options] command [filenames ...]
- -f option allows the commands to come from a file. In the file, you can have multiple commands. Each goes on a separate line.
- -n suppress automatic output. Use p to print the lines you want.
- -i in-place modification (results go to stdout by default)
- -e allow you to put more than one command in a single call to sed

sed examples

- s/old/new/ f1 f2 replace old by new
- y/str1/str2 replace each character from str1 with corresponding character from str2
- d delete line (use with pattern or a count)
- p print current line (if output is suppressed)
- w file write line to a file
- q quit

sed examples

- echo "123 abc" | sed 's/[0-9]*/& &/‘
  - Displays 123 123 abc
- sed 's/\([a-z]*\).*/\1/‘
  - Keep the first word of the line and remove the rest
- sed 's/^\(.\)\(.\)\(.\)/\3\2\1/‘
  - Reverse the first three characters
- sed 's/\([a-zA-Z]*\) \([a-zA-Z]*\) /\1 /’
  - Delete the 2nd word in a string

Substitution

- Syntax
  `<restrictions> s/<pattern>/<replacement>/<flag>`
- `/` is the usual delimiter for the pattern and replacement string but you can use any character
- `<pattern>` is the text to be replaced. It is a regular expression so you can use the usual metacharacters. Using `()` in the pattern allows you to access parts of the matched text by position.
- `<replacement>` is what to replace the matched pattern with. A `&` in the replacement text puts the matched text into the replacement.
- `<flag>` can be one of the following
  - `g` says to replace all occurrences
  - a number allows you to specify the occurrence to be replaced by index
  - `w file` says to write the output to a file
- `<restrictions>` tells sed how to select lines for modification. (See next slide)

Adding lines to the output

- You can add extra lines to the file being processed using the following commands:
  - r Insert text from file
  - a Append lines to output
  - i Insert text before next output
  - c Replace lines by following text
- Lines to be added end at the closing single quote for the command
- You can specify which lines in the original file are affected in the usual way

Flow Control

- sed has a primitive form of flow control
- b label branch to label (goto)
- t label branch to label if substitution is made on current line
- : label set label for b and t commands
- { } create a compound statement
- !cmd do cmd if line is not selected
- Resource: *www.grymoire.com/Unix/Sed.html*

awk

- awk is a language that uses regular expressions to determine which lines of a file to process
- The form of an awk program is

```
pattern {actions}
```

- where pattern is a regular expression or a logical expression and action is commands to process the line that matches

awk

- Each line is broken up into words (whitespace-delimited); each word can be accessed by its position in the line
- $0 is the entire line
- $1, $2 … are the individual fields
- This is especially useful for files that contain many lines with basically the same structure

Using awk

- awk runs the program specified in the command line on each file listed

```
awk program file ..
```

- The program can be provided literally or the –f command line option can specify the file that contains it
- Variables can be assigned values from the command line
  - v var=val
- Use –F fs to use a field separator other than whitespace

Patterns in awk

- Patterns can be regular expressions with the usual metacharacters
- There are a number of pre-defined character classes
  - \w for letters, digits and the underscore character
  - \W for characters that are NOT letters, digits or _
- Patterns can be relational expressions with the usual comparison operators

Patterns in awk

- The ? : operator can be used to select a pattern
- Patterns can be negated with !
- Patterns can be combined with && and ||
- Multiple patterns can be separated by commas
- BEGIN and END are special patterns that match the start and end of the file respectively

Useful String functions

- gsub (pattern, replacement[, string])
  - gsub replaces globally
  - default string is $0
  - returns number of replacements
- index (string, toFind)
- length (string)
- split (string, array [, fieldSep])
- substr(string, start [, length])

awk Built-in variables

- ARGC, ARGV – command line arguments
- FILENAME – name of current file
- FNR – record number in current file
- NR – number of records read so far in all files
- FS – field separator (“ ” by default)
- RS – record separator (newline by default)
- RSTART – start of string matched

awk Examples

- awk '/fopen/ ' temp.c
  - Print line that matches the pattern
- awk '{print $2,$5;}' employees.txt
  - Print only specific words/fields in a file
-
```
awk 'BEGIN {print
"Name\tDesignation\tDepartment\tSalary";}
{print $2,"\t",$3,"\t",$4,"\t",$NF;}
END{print "Report Generated\n--------------";
}' employees.txt
```
  - Format text

awk Examples

- awk '$1 >200' employees.txt
  - Display employees whose id is greater than 200
- awk '$4 ~/Technology/' employees.txt
  - Display employees in the Technology department
- awk 'BEGIN { count=0;} $4 ~ /Technology/ { count++; } END { print "Number of employees in Technology Dept =",count;}' employees.txt
  - Print number of employees in the Technology department

References

- The UNIX Programming Environment – Kernighan and Pike
- The AWK Programming Language, Ajo, Kernighan and Weinberger
- sed & awk – 2nd edition, Dougherty and Robbins, O'Reilly
- GNU sed Manual
- Mastering Regular Expressions – Friedl