Midterm Review

Matt Warner

1 Topics covered so far

- Introduction
- File system
- Editors
- Net Ultilities
- Permissions
- Shell
- Shell scripts
- awk report generator
- sed stream editor

2 History of UNIX

- Invented by Ken Thomson at Bell Labs in 1969
  - First version written in assembly language
  - single user system, no network capability
- Thompson, Dennis Ritchie, Brian Kernighan
  - Rewrote Unix in C
- Unix evolution:
  - Bell Labs, USL, Novell, SCO
    - BSD, FreeBSD, Mach, OS X
    - AIX, Ultrix, Irix, Solaris, ...
  - Linux: Linus Torvalds
  - Newest:
    - Linux on portables: Android

3 Command Line Structure

Syntax: $ command [-options] [arguments]

- UNIX is case sensitive
- Must be a space between command, options and arguments
- Fields enclosed in [ ] are optional

Note:-

The brackets as used to show that they are optional

4 RTFM: The man Command

Shows pages from system manual

Syntax:

```
$ man date
$ man -k date
$ man crontab
$ man -S 5 crontab
```

Section	Description
1	User commands
2	System calls
3	C library functions
4	Special system files
5	File formats
6	Games
7	Misc. features
8	System admin utilities

Caveats:

Some commands are aliases

Some commands are part of the shell

5 Path

- path: list of names seperated by "/"
- **Absolute Path**
  - Traces a path from root to a file or a directory
  - Always begins with the root (/) directory

    Example: /home/student/Desktop/assign1.txt
- **Relative Path**
  - Traces a path from the current directory
  - No inital forward slash (/)
    - dot (.) refers to current directory
    - (..) referes to one level up in the directory hierarchy

      Example: Desktop/assign1.txt

6 Linking Files

- Allows one file to be known by different names
- Link is a reference to a file stored elsewhere

- 2 types:
  - Hard link (default)
  - Symbolic link (a.k.a. "soft link")

Syntax:

```
ln [-s] target local
```

7 User's Disk Quota

- Quota is upper limit of
  - amount of disk space
  - number of files

  For each user account
- 2 Kinds of limits:
  - Soft limit: ex. 100MB (system will only complain)
  - Hard limit ex. 120MB (Cannot be exceeded)

- **command:** quota -v
  - displays the user's disk usage and limits

8 Unix Text editors

- Vi or vim
- pico, nano

- GUI editors
  - gedit or "text Editor"
  - geany
  - emacs
  - VS code

9 Networking Utilities

- Most network protocols were developed on UNIX
- Most (not all) UNIX systems provided them:

- login to another computer
  - telnet, rlogin, rsh, ssh
- copy files to another computer
  - rcp, spc
  - ftp, sftp
- Linux desktop provides GUI-enabled tools
  - file manager

10 Special Permissions

- The regular file permissions (rwx) are used to assign security to files and directories
- 3 additional special permissions can be optionally used on files and directories
  - Set User Id (SUID) (allows programs to run with the permissions of the owner of the file)
  - Set Group ID (SGID) (allows programs to run with the permissions of the group of the file)
  - Sticky bit (makes sure you can only delete files that you actually own)

11 Permissions

12 File mode creation mask

- umask (user mask)
  - governs default permissions for files and directories
  - sequence of 9 bits: 3 times 3 bits of rwx
  - default 000 010 010 (022)
- in octal form its bits are removed from:
  - for a file: 110 110 110 (666)
  - for a directory: 111 111 111 (777)
- new permissions
  - file: 110 100 100 (644)
  - directory: 111 101 101 (755)

13 UNIX Command Interpreters

Common Term: Shell

- Standard:
  - every UNIX system has a "Bourne shell compatible" shell
- history:
  - sh: Original Bourne shell, written in 1978 by Steve Bourne
  - ash: Almquist shell, BSD-licensed replacement of sh
- Today:
  - bash: Bourne again shell, GNU replacement of sh
  - dash: Debian Almquist shell, small scripting shell

14 Bash shell basics

- Customization
  - startup and initilization
  - variables, prompt and aliases
- Command line behavior
  - history
  - sequence (;)
  - substitution (``)
  - redirections and pipe

15 Redirections and Pipe

Command Syntax	Meaning
`command < file`	redirect input from `file` to `command`
`command > file`	redirect output from `command` to `file`
`command >> file`	redirect output of `command` and appends it to `file`
`command 2> file`	add error output to standard output, redirect both into `file`
`command 2>&1`	redirect both standard output and standard error to `file`
`command1 | command2`	take/pipe output of `command1` as input to `command2`
`command <& label`	take input from current source until `LABEL` line

16 Wildcards: * ? [ ] { }

A pattern of special characters used to match file names on the command line

* zero or more characters
$ rm *
$ ls *.txt
$ wc -l assign.*
$ cp a*.txt docs

? exactly one character (can be chained together)
$ ls assign?.cc
$ wc assign?.??
$ rm junk.???

17 Regular Expressions

- A pattern of special characters to match strings in a search
- Typically made up from special characters called meta-characters: . * + ? [ ] { } ( )
- Regular expressions are used throughout UNIX:
  - utilities: grep, awk, sed, ...
- 2 types of regular expressions: basic vs. extended

18 Metacharacters

. Any one character, except new line

[a-z] Any one of the enclosed characters (e.g. a-z)

* Zero or more preceding characters

? also \? Zero or one of the preceding characters

+ also: \+ One or more of the preceding characters

^ or $ Beginning or end of line

\< or \> Beginning or end of word

( ) also: \(\) Groups matched characters to be used later

| also: \| Alternate

x{m,n} also: x\{m,n\} Repetition of character x between m and n times

19 Quoting and Escaping

Allows to distinguish between the literal value of a symbol and the symbols used as meta-characters

Done via the following symbols:

- Backslash (\)
- Single quote (')
- Double quote (")

20 Shell Script: the basics

- 1. line for shell script
  #! /bin/bash
  or: #! /bin/sh
- to run:
  $ bash script
- or:
  - make executable: $ chmod +x script
  - invoke via: $ ./script

21 bash shell Programming Features

- Variables
  - string, number, array
- Input/Output
  - echo, printf
  - command line arguments, read from user
- Decision
  - conditional execution, if-then else
- Repetition
  - while, until, for
- Functions

22 Command line arguments

	Meaning
$1	first parameter
$2	second parameter
$10	10th parameter
$0	Name of the script
$*	all positional parameters
$#	The number of arguments

23 User input: read command

Syntax:

```
read [-p "prompt"] varname [more vars]
```

Note:-

Once again the brackets are for stuff that is optional

- words entered by user are assigned to

  ```
  varname and "more vars"
  ```

- last variable gets rest of input line

24 test command

Syntax:

```
test expression
[ expression ]
```

- evaluates 'expression' and returns true or false

Example:

```
if test $name = "Joe"
then
  echo "Hello Joe"

if [ $name = "Joe" ]; then
  echo "Hello Joe"
```

25 Shell scripting features

- How to debug?
- Decision
  - case
- Repetition
  - while, until
  - for
- Functions

26 What can you do with awk?

- awk operation:
  - reads a file line by line
  - splits each input line into fields
  - compares input line/fields to pattern performs action(s) on matched lines
- Useful for:

- transforming data files
- producing formatted reports

- Programming constructs
  - format output lines
  - arithmetic and string operations
  - conditionals and loops

27 Basic awk script

- consists of patterns and actions

Syntax:

`pattern {action}`

- if pattern is missing, action is applied to all lines
- if action is missing, the matched line is printed
- must have either pattern or action

Example:

```
awk '/for/ { print }' testfile
```

- prints all lines containing string "for" in testfile

28 awk variables

awk reads input line into buffer: record and fields

- field buffer:
  - one for each field in the current record
  - variable names: $1, $2, ...
- record buffer
  - $0 holds the entire record

- NR Number of the current record
- NF Number of fields in the current record
- FS Field separator (default=whitespace)

29 awk Patterns

- simple patterns
  - BEGIN, END
  - expression patterns: whole line vs. explicit field match
    - whole line /regExp/
    - field match $2 ~/regExp
- range patterns
  - specified as from and to:
    Example: /regExp/,/regExp/

30 awk Patterns

- basic expressions

- output: print, printf
- decisions: if
- loops: for, while

31 How Does sed Work?

- sed reads file line by line
  - line of input is copied into a temporary buffer called pattern space
  - editing instructions are applied to line in the pattern space
  - line is sent to output (unless "-n" option was used)
  - line is removed from pattern space
- sed reads next line of input, until end of file

Note:-

input file is unchanged unless "-i" option is used

32 sed instruction format

address determines which lines in the input file are to be processed by the commands(s)if no address is given, then the command is applied to each input line

- address types:
  - Single-Line address
  - Set-of-Lines address
  - Range address