

Midterm Review

1. Midterm Review

1.1 Overview

CSCI 330 UNIX and Network Programming



Midterm Review



1.2 Topics covered so far

Topics covered so far

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

1.3 History of UNIX

History of UNIX

- Invented by Ken Thompson 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

1.4 Command Line Structure

Command Line Structure

% command [-options] [arguments]

Command prompt

Command name

Command modifier:
usually one character
preceded by + or - sign

Arguments can be:
1. names of files
2. more information

- **UNIX is case sensitive**
- Must be a space between command, options and arguments
- No space between the plus or minus sign and option letter
- Fields enclosed in [] are optional

1.5 RTFM: The man Command

RTFM: The man Command

- show pages from system manual

Syntax: man [options] [-S section] command-name

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

Caveats:

Some commands are aliases
Some commands are part of shell

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

1.6 Path

Path

- path: list of names separated 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 initial forward slash (/)
 - dot (.) refers to current directory
 - two dots (..) refers to one level up in directory hierarchy
 - Example: Desktop/assign1.txt

1.7 Long List Option

Long List Option

```
% ls -al
total 126
drwxr-xr-x 13 ege csci 1024 Apr 26 15:49 .
drwxr-xr-x 15 root root 512 Apr 24 15:18 ..
-rwxr--r-- 1 ege csci 885 Dec 2 13:07 .login
-rwx----- 1 ege csci 436 Apr 12 11:59 .profile
drwx----- 7 ege csci 512 May 17 14:11 330
drwx----- 3 ege csci 512 Mar 19 13:31 467
drwx----- 2 ege csci 512 Mar 31 10:16 Data
-rw-r--r-- 1 ege csci 80 Feb 27 12:23 quiz.txt
```

Annotations:

- List all files in current directory in long format
- . is current directory
.. is parent directory
- dot (.) names are hidden files
- directories
- plain file

Field Headers:

- File Permissions
- Links
- Owner
- Group
- File Size
- Last Mod
- Filename

1.8 Linking Files

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`

1.9 User's Disk Quota

User's Disk Quota

- quota is upper limit of
 - amount disk space
 - number of filesfor each user account
- command: `quota -v`
 - displays the user's disk usage and limits
- 2 kinds of limits:
 - Soft limit: ex. 100MB
 - Maybe exceeded for one week
 - System will nag
 - Hard limit: ex. 120MB
 - Cannot be exceeded

1.10 Unix Text editors

Unix Text editors

- vi
- pico, nano
- GUI editors
 - gedit, xedit
 - geany
 - mousepad
 - emacs

1.11 Networking Utilities

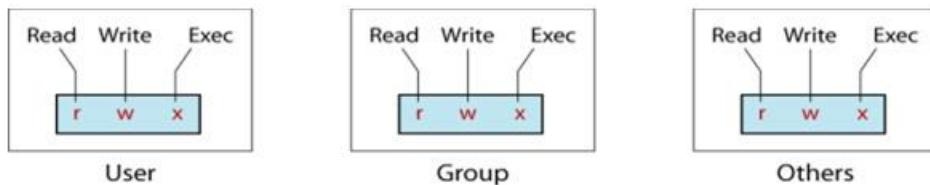
Networking Utilities

- most network protocols were developed on UNIX
- most (not all) UNIX systems provide them
- login to another computer
 - telnet, rlogin, rsh, ssh
- copy files to another computer
 - rcp, scp
 - ftp, sftp
- Linux desktop provides GUI-enabled tools
 - file manager

1.12 Permissions: Categories of Users

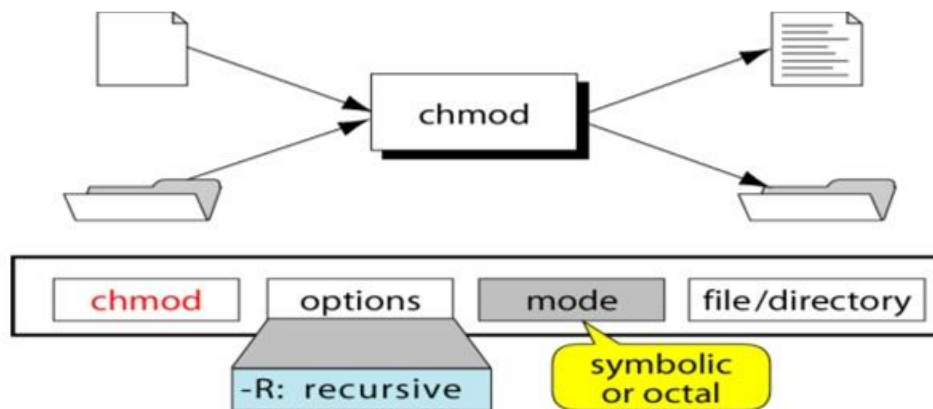
Permissions: Categories of Users

- 3 categories of users want access



1.13 Change Permissions with chmod

Change Permissions with chmod



1.14 Special Permissions

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)
 - Set Group ID (SGID)
 - Sticky bit

1.15 File mode creation mask

File mode creation mask

- umask (user mask)
 - governs default permission 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)
- permission for new
 - file: 110 100 100 (644)
 - directory: 111 101 101 (755)

1.16 UNIX Command Interpreters

UNIX Command Interpreters

common term: shell

- standard:
 - every UNIX system has a “Bourne shell compatible” shell
- history:
 - sh: original Bourne shell, written 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

1.17 bash shell basics

bash shell basics

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

1.18 Summary: Redirections and Pipe

Summary: Redirections and Pipe

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

1.19 Wildcards: * ? [] { }

Wildcards: * ? [] { }

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

***** zero or more characters **?** exactly one character

% `rm *`

% `ls *.txt`

% `wc -l assign1.*`

% `cp a*.txt docs`

% `ls assign?.cc`

% `wc assign?..??`

% `rm junk.???`

1.20 Regular Expression

Regular Expression

- 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

1.21 Metacharacters

Metacharacters

<code>.</code>	Any one character, except new line
<code>[a-z]</code>	Any one of the enclosed characters (e.g. a-z)
<code>*</code>	Zero or more of preceding character
<code>?</code> also: <code>\?</code>	Zero or one of the preceding characters
<code>+</code> also: <code>\+</code>	One or more of the preceding characters
<code>^</code> or <code>\$</code>	Beginning or end of line
<code>\<</code> or <code>\></code>	Beginning or end of word
<code>()</code> also: <code>\(\)</code>	Groups matched characters to be used later
<code> </code> also: <code>\ </code>	Alternate
<code>x{m,n}</code> also: <code>x \{m,n\}</code>	Repetition of character x between m and n times

1.22 Quoting & Escaping

Quoting & 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 (")

1.23 Shell Script: the basics

Shell Script: the basics

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

1.24 bash Shell Programming Features

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

1.25 Command line arguments

Command line arguments

	<u>Meaning</u>
\$1	first parameter
\$2	second parameter
\${10}	10th parameter
	{ } prevents "\$1" misunderstanding
\$0	name of the script
\$*	all positional parameters
\$#	the number of arguments

1.26 User input: read command

User input: read command

Syntax:

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

- words entered by user are assigned to `varname` and “`more vars`”
- last variable gets rest of input line

1.27 test command

test command

Syntax:

```
test expression  
[ expression ]
```

- evaluates ‘expression’ and returns true or false

Example:

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

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


1.28 Shell scripting features

Shell scripting features

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

1.29 What can you do with awk?

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:
 - transform data files
 - produce formatted reports
- Programming constructs:
 - format output lines
 - arithmetic and string operations
 - conditionals and loops

1.30 Basic awk script

Basic awk script

- consists of patterns & actions:

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

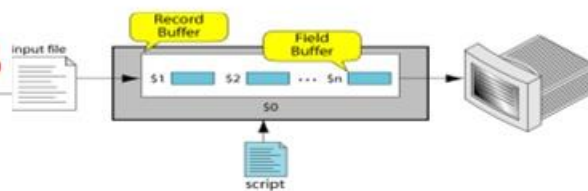
Example:

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

- prints all lines containing string "for" in testfile

1.31 awk variables

awk variables



- awk reads input line into buffers:
record and fields

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

- record buffer:
 - \$0 holds the entire record

Also:

NR Number of the current record
NF Number of fields in current record

also:

FS Field separator (default=whitespace)

1.32 *awk Patterns*

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/`

1.33 *awk actions*

awk actions

- basic expressions
- output: `print`, `printf`
- decisions: `if`
- loops: `for`, `while`

1.34 How Does sed Work?

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

1.35 sed instruction format

sed instruction format



- address determines which lines in the input file are to be processed by the command(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

1.36 Topics covered so far

Topics covered so far

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