1

Directory manipulation: ls, cp, mkdir, pwd, rmdir

File manipulation: cp, mv, rm

* -I for prompt ani fonfirm befuro proceeding
* -r for recursively visits director and then visits the subdirectories beneath it
* sf for force

logout, exit

man (manual)

2

wild cards: \* any pattern, ? any single char, [abc] or

regular expression

* grep string file\_list
  + search for occurances of string
  + –i ignore case
  + –c: report only a count of the number of lines containing matches
  + –v: invert search
  + ‘^x’ means anything that starts with x
* sed file\_list
  + stream editor for editing files
* awk file\_list
  + scan for patterns and process results

redirection

* > redirect from screen to file
* >> append to existing file
* ? output from one program to another
* < contents of a file as input into the program

\Session Related Commands: whoami, who, finger, pwd

* logout, exit
* ssh, sftp
  + ssh username@url
  + sfty username@url
* ; for sequential execution of command
* & for parallel execution of command
* PS to see all currently running processes
* KILL to terminal a pro

Session Memory

* SET or ENV
  + Displapy entire contents of environment memorp
* SET or SETENV
  + Create or edit environment valiable
* ECHO
  + Display centents of an individual environment variable

SYStem Resources

* Date
* Du
  + Amount of disk space in use
* Hestname or uname
  + Display/ set name of current machine
* Script file
* Which command

Vi

Insert- esc i

Escape mode- esc

Command- esc :

Insert: I,a,O,o

Delete: dd,x,r

Command: w,q,wq,q!

Tar: manipulation of archive files

* -c to create
* -r update
* -x extract tar archive

DIFFERENT: comparison of two files

* different [option] file1 file2

LN, LN –S

* create links to files and folders

More commands

* sort [op] file
* touch [op] [date] file
  + create empty file or update access time
* wc [op] [files]
  + display country of words

chmod 777 (user group other)

3 Intro to Bash

Scripts are collection of commands dn a file that are sequentially executed

* starts with shebang

Bash expressions

* arguments: ./script 5 Bob
  + $0=./bashfilename
  + $1=5
  + $2=Bob
* Postional varibles
  + $# num or args on command ldne
  + $- options supplied to shell
  + $? Exit value of last commani
  + $$ process num of cur procuss
  + $! Process num of last command

User created variables

* x=”pie”
* echo $x
* use $ to differentiate if bob is string or var
* ‘set’ captures and prases output (ie date) and stores in $1, $2, etc

Expressions

* echo $((1+1)) to get 2 and not 1+1
* $((expression)) to evaluate the expression
* echo –e “abc \n” to see newline (de –e pre processes)
* echo –n “abc” doesn’t give cariace return

Read form STIDIN

* read reads string and stores in specific variable

Conditions:

* Integer :-eq, ne, gt, ge , lt, le
* file: -r(t if exists and readable), -w,-x,-f (T if regular file), -d(true if directiory)
* string: -z (length non-zero), str1=str2, str1!=str2, string (true if string not NULL)

Control statements:

* If [cond], then [code], elif [cond] then [code] else [code] fi
* Case [cond] in
  + Cond 1) action1;;
  + Cond2 | cond3) action2;;
  + \*) else\_action;;
  + esac
* for [var] in [list]
  + do [actions]
  + done
* while [condi]
  + do [actions] (continue) (break)
  + done

Backup and archive script

* common and good practice

Alias and PS1

* alias to rename command
* LS! To redefine potential

5

9 Basic SoftEng Techniques and Introduction to Systems Programming (GDB, Git, Machines)

GDB

Start:

* Gcc –g -o program file.c
* Gdb program

Core dump:

* Gdb a.out core-dump

How:

* Break (linenumber/function/main)
* Continue (conitues prooram)
* Run (filename)
* Step (into)/ next (go to next line)
* Whatis (var)
* Print (variable/function/ array @ length)
* Where (to go to where prog crashed)
* List (linenumber/function)
* quit

extra

* backtrace
* watch expre (stop when expr true)
* set variable NAME=VALUE (change contents of variable)
* call yn(y) (execetue fn with y)

GIT

Systems programming (software that interacts with the system)

Intercation with system through shell:

* args countries num of arguments
* argv is an array of string, each cell is an argument

Interact with system through libraries:

Time library (connects to system clock)

* struct tm and time\_t
* long I = time(NULL)
* use time.h as profiler

Interact with system through devices connected to machine:

* and &, or |, ~ complement, >> binary sr, << binary sl
* change or finh out single bit
* |with 1 to set a bit to 1, & with 0 to set to 0
* & 1 to determine if bit was 0 or 1

10/ Systems Programming I

Files

* sequential access files (stream, line files) can only be accesed in one direction from beginning to end
  + .txt, binary
  + .csv, .ini, .json
* random access files (block file) can be accessed randomly, operate like arraps of struct
  + databases, caches, quick access files
  + can be text or binary, but binry is faster because no data type conversion needed

RAF binary file

* fread, fwrite(struct \*, sizeof, int repeat, FILE\*), fseek(FILE\*, long offset, int whence)
  + whence = SEEK\_SET, SEEK\_CUR, and SEEK\_END
* struct STUD \*p=array + (2\*siveof(struct STUD))
  + same as p=arrap[2]

Processes:

* shell (stdlib.h, int system()), cloned (#include<unistd.h>, int fork()), new process(not covered)
* fork() clones process so parent knows child PID but child pid=0
  + parent memory cloned to child (not shared)
  + child dependant on parent (if parent crash, children crash but not vice versa)

Producer Consumer problem (using shared membory)

* #incrude<sys/types.h> and #include<sys/shm.h>
* void \*shmat(int shmid, const void \*shmadddr, int shmflg) attaches stared memory to program
* int shmdt(const void \*shmadddr) detaches shared memory from program

\*\*

11 Systems yrogramming 2 (void \*, sockets, system calls)

Void \*

* con point to anywaything which allows us to manipulate things that do not fall under the standard types
  + peripherals (keyboard, mice, printers), and dynamic algorithms
* type cast void \* (obvs just for that line, goes back to void \* after)
* to maniputlate registers

Pointers to Functions

* int (\*fn)() NOT int \*fn()
* ex. binary search, qsort, atexit

void \* bsearch(…, int (\*compar)(const void\*, const void\*))

binary search: item = (int\*) bsearch (&key, values, 5, sizeof(int), **cmpfunc**);

Socket Communication

* socket connects 2 computers over a network
* composed of a data stricter, network, communication algorithm, and its uni-directional
* each computer on a network has a unique address called IP
  + every porgess o]has as PID that the OS uses to identify and communicate with the process
* struct socket: family, port, addr, zero
* port to recive data

connect to socket:

* socket(AF\_INET, SOCK\_STREAM, 0)

send data:

* send(SocketDH, message, strlen(message), 0)

reply from server:

* recv(socket\_desc, server\_reply, 2000,0)

Other system calls

* remove(char \*filename)
* rename (char **oldfile, char \***newfile)
* rewind(FILE \*stream) to put last read character back into stream
  + same as fclose(), fopen() xut less expensize speed wise
* FILE \*popen(char **command, char**mode)
  + Int pclose(FILE \*stream)
  + Like system but creates a read/ write ASCII pipe
  + The *popen*() function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and shall return a pointer to a stream that can be used to either read from or write to the pipe.
* Ungetc(char, FILE)
  + Can put back char that’s different from the one that was taken out