C

__attribute__((constructor)) run before main, dlopen call
__attribute__((destructor)) run after main, dlclose() call
Compilation: Preprocessor > Assembly Code > Object Code > Link Objects
./configure: check dependencies, may generate makefile
Data Type: 1 Byte char, 4 Byte int/float, no boolean type
Dynamic Linking: Link a reference so it can be access when it is needed, ".so" file
Dynamic Memory: malloc, realloc, free
fopen("file.txt","r"); Open a file with read permissions
-fPIC: position independent code works if code move
fstat: Gives information on a file
Functors: Function pointer, pass functions to functions
double (*func_ptr) (double, double);
func_ptr = &pow; // func_ptr now points to pow()
GCC Flags:
  -rpath=. // At runtime, find ".so" from this path
  -c: Generate object code out of C code
  -shared: Make object which can be shared with others
  -l: Name another library to link with
include <dlfcn.h>
  void* dlopen(const char* filename, int flag)
    Make object file accessible to program (7)
    flag is either RTLD_LAZY or RTLD_NOW
  void* dlsym(void* handle, const char* symbol)
    Take handle and return memory address
  int dlclose(void) // Decrement handle count
int getopt(int argc, char* const* argv, const char* options); ":" means that option has an argument
int main(int argc, char **argv): argument count, argument array of strings
Linker: collect procedures and link object modules into executable program
Macro: #define MAC v1 //replace MAC with v1 in code
make: requires a makefile to build the file
makefile: controls a recompile based on what changed, and keeps track of dependencies of files
qsort: Standard C sorting, given a comparison function*
  qsort(words, numwords, sizeof(char*), cmpFunct);
Static Library: ar rcs my_library.a file1.o file2.o
Static Linking: Gets all needed modules and copies it for use, denoted with a ".a"
<stdio.h>: functions for IO
Struct: No functions, No constructor, No privates
void free(void* ptr); Free up memory that was allocated
void* malloc(size_t size); Allocate space in bytes
  char *word = (char*)malloc(sizeof(char));
void* realloc(void* ptr, size_t size); Assign new bytes
  words = (char**) realloc(words, (wordnum + 1) * sizeof(char*));
getchar()/putchar(): stdIO optimized read and write

Definitions

Buffered: Collect bytes into buffer and use 1 syscall
CLI: Command line interface, more control than GUI
Compilation (C++): Preprocessor > Assembly Code > Object Code > Link Objects
Coreutils: Package of core GNU tools, ls, cat, rm, etc.
Debugger: Help find segmentation fault & logical errors
Detached Digital Signature: Signature separate, instead of stuck on the end, good for sending with tarballs
ELF: Executable Linking Format file, none, .bin, .s, .so
Entropy: randomness from the OS, slowly builds up
Git: Distributed version control software
GUI: Graphical user interface, easier to use
Interpreter (Python): Goes line by line and portable but is a lot slower and has more overhead
Kernel: Manages operations of computer and hardware
Kernel Mode: Unrestricted access, assume trust software
Linux: Open source unix operating system
Locale: Defines user's cultural preferences
Makefile: Manage compilation for updating -o files
Memory: Global Data > Code > Heap > Stack
Multiprocessing: Use different cores for different tasks
Multitasking: Several separate processes running
Multithreading: Seperating a task into independent pieces to run simultaneously (independent stacks)
Open Source: Source code available to the public
Parallelism: Run many computations simultaneously
Patching: Created with diff command, applied by patch
Processes: Address global data, code, heap, stack
Protection (OS): IO, memory, CPU protected vs user
Python: Scripting language, 2 different from 3
Diff: State different lines between two input files
Race Condition: Parallel threads execute differently based on which finishes first (BAD)
RegEx: Allows searching for a pattern
Segmentation Fault: Trying to access illegal memory
Shared Library: Dynamic linking, smaller memory footprint, don't need to recompile source
Shell: UI for computer, either command line or GUI
Shell Script: A program that shell runs, run by the OS
SSH: Secure shell (Asymmetric Key Encryption better)
SSH Agent: Stores your login, don't need password
System Calls: User-level software can attempt to use kernel space software, lot of overhead
Tar: Compiled file (.tar/.tar.gz) tar -xzvf file.tar.gz, z if .gz, c instead of x if creating a zip file
Thread: Flow of instructions in a process
.tgz: tar -zxvf yourfile.tgz
Trusted Software: Software in kernel space, in the OS
Unbuffered: Each byte written using system call
User Mode (OS): Have to use system call for access
Version Control: Track history of changes for software
x: Program runs on one PC and displays on another

POSIX Threads

pthread_create: create new thread, 0 if successful

create(name, attributes, function, 1 arg for function)

// creates a new stack each time it's called

pthread_join: waits for child threads to terminate

int pthread_join(pthread_t tid, void** status);

// NULL if no status, returns 0 if successful

pthread_equal: check if they're the same thread

pthread_self: return ID of calling thread

pthread_exit: terminate the current thread

Critical Section: Sections of code that must be run by 1 thread at a time or it might break

GDB

backtrace: The path leading to your bug

break: Break at a specific line or function, can also add conditionals, ex: break Class<int>::function if val==0

c: continue to run until next breakpoint, error, or end

delete [bp_number]: Deletes the breakpoint, default all

f: Finish the function you are in

format: decimal d, hexadecimal x, octal o, binary t

gdb [src]: Start gdb with the source code

help [command]: Get more info on the command

info [args]: Gives argument values of function call

info breakpoints: Gives info about your breakpoints

list: List source code lines around the current line

n: Go to next line, step over functions

print[/type] expression: decimal d, hex x, binary t

quit: Get out of gdb

run: Run, you can also list arguments afterwards

s: Go to next line, step into functions

watch [expression]: Stop when expression value change

x [memory address]: Gives the value at the address

Git

blob: holds the data for a file

branch: a line of development, current branch HEAD

conflict: incompatible commits, resolve before push

git add: move your changes to the staging area

git branch -av: list all existing branches

git checkout: get a copy of files from main repository

git clone: create a copy of an existing repository

git commit: move staging area changes to repository

git diff: compares changes made since staging area
  git diff b1..b2 difference, b1...b2 diff common ancestor

git init: create an empty git repo with branch master

git log: show the history of commits

git merge: merge branch into HEAD (or into each other)

git status: gives information on modified files

head: the most recent commit

merge: bring branches together, update master changes

tag: name for a commit

tree: holds all of the relationships between blobs

Python

Arg: Passed in, options can also have args, sys.argv[1:]

Dictionaries: Hash tables with key-value pairs

For: for element in list, do something

Interpreter: Goes line by line and portable but is a lot slower and has more overhead

List []: Dynamic array that can hold all types
  append() to the end, merge lists by adding
  List[a:b]: select items a-b, also works for strings

OptParse: Parse command line options
  Action, Destination, Store variable, Default, Help Msg

Option: -n, -i, optional attachments to scripts, arguments
  -o file.txt or --opt file.txt or --opt=file.txt

Try/Except: Attempt an action in try, execute except if the action failed

Tuple: Immutable type with set number of entries

```
tuple = ("apple", "banana", "cherry")          //packing
a, b, c = tuple                                //unpacking
```

Regular Expressions (regexp)

^ match following regexp with beginning of line/string

$ match preceding regexp with end of line/string

. match single char, * 0+ preceding char, + 1+ preceding char, ? 0 or 1 preceding char, {n} preceding char, {n,} n or more preceding char, {m.m} n to m preceding char

[...] match any one of enclosed char, - allow range in [...]

[^abc] matches anything not enclosed

[:alnum:][:lower:][:space:][:punct:][:alpha:]

(capture/quantifier) stored in 9 capture groups \1-\9

Basic regexp: standard, more literal, special with \

Extended regexp: use special meanings, unspecial \

grep [options] pattern [file…]: Search file/stdin for regexp pattern, return matching lines

-E extended regexps, -F match fixed strings

Sed script [file]: 's/regexp/replacement/flags' g - global I - case insensitive

Find any line beginning with 'th', case insensitive

grep -E '^[Tt][Hh]'

Find line with 2 digits separated by 1+ spaces

grep -E [[:digit:]][]+[[:digit:]]

Find lines with 1+ 2 or more d[0+ o's]t patterns

grep -E ((do*t) \2)+

sed 's/(\([0-9]\{3\}\))-\([0-9]\{3\}\)-\([0-9]\{4\}\)[[:space:]]/\1\2\3/g' < contact.html

Replace formatted phone numbers with int only form

Terminal

cat: combine input files in order and output it
cd : change directory
chmod: give read/write/execute to user/group/other/all
comm: compare two sorted files line by line
cp: create a copy of a file
Directory: home ~ current . parent .. root /
echo: print the value of $key
find: -name, -type, --perm, -user, -maxdepth
ln: create a hard link, -s for a soft link
ls: list contents, -a list all, -d directories only, -l long list info, -s show size in blocks
kill: terminate certain processes based on PID
man: manual for linux commands
mkdir: create a new directory
mv: move or rename a file
patch: patch -p[num] < patchfile.txt (the diff file)
ps: list processes currently running
pwd: print path directory you're currently in
rm: remove a file, -r removes files/dir recursively
rmdir: remove an empty directory
scp: secure copy file from source to destination
sort: sorts lines of text
time [options] command [args]: outputs amount of time program runs overall, and in user or system mode
touch: create a new file, -t to modify time
tr: translate or delete characters
wget: download from a url
which: show full path to a command
xargs: build and execute commands from standard input

Shell Scripting

Shebang (#!): Change the shell, #!/bin/bash
Then just use terminal commands to program actions
Includes conditional statements & variables (var="varr")
Reference vars: echo $varr, echo "${varr}_string"
Given var: PATH, # (current num arguments), ? (exit previous command), IFS (internal field separator)
Arguments: Autosaved vars echo "${1}", "${2}", etc.
If: if [${1} -ge 0] then echo ">=0" else echo "<0" fi
While: cnt=1 while[${cnt} -le 10] do echo "${cnt}" let cnt=cnt+1 done
For: var="string" for c in $var do echo "${c}" done
getopts OPTSTRING VARNAME[ARGS…]
stored in VARNAME, invalid options set to '?'
Quotes: '' literal meaning no expand, "" expand only backtick and $, `` expand as shell command
stdin(0) data in, stdout(1) output, stderr(2) error codes
stdin< stdout> stderr2> append stdout>> redirect|
Exit status code: 0 (success), 1-125 (fail), 127 (unfound)
Give permissions with chmod u+x
set -x, set +x: turn tracing on/off respectively
Run with, ./script.sh

```
find . -type f | xargs grep "poop"
```
search (.) directory for all (f) file types and opens them to grep for the term "poop"

```
awk 'BEGIN {sum=0; count=0; OFS="\t"}
{sum+=$2; count++} END {print "Average:",
sum/count}' file.txt
```
Get average of second column in file.txt using awk

```
tr -cs 'A-Za-z' '[\n*]' | sort -u | comm -23 - words
```
Translate (tr) non (-c) alphabetic characters into newlines and suppress (-s) duplicate newlines, sort the result for unique (-u) lines and compare (comm) them to a file called words, printing out only lines unique to the first input of the comparison (-23)

```
od -An -tfF -N $((4*(2**24))) < /dev/urandom | tr -s '
' '\n' | sed '/^$/d' > flts.txt
```
od and -tfF generate floating points from urandom, -An remove address, -N limit bytes to 4*2^24, tr SP to \n and suppress dups, remove empty lines, put in file flts.txt

```
#!/bin/bash
grep -E '<td>.+<\/td>' |          # get non-empty html tags
sed -n '1~2!p' |                  # !print every other line
tr [:upper:] [:lower:] |          # make all lowercase
tr '\`' "\'" |                    # replace ` with '
sed 's/<td>//g' |                 # remove opening tags
sed 's/<\/td>//g' |               # remove closing tags
sed 's/<u>//g' |                  # remove opening tag
sed 's/<\/u>//g' |                # remove closing tag
tr ' ,' '\n' |                    # replace , and SP with \n
sed "/[^pk\'mnwlhaeiou]/d" |      # remove lines non-H char
sed '/^$/d' |                     # remove empty lines
sort -u                           # sort unique results
```

```
#!/bin/bash
LC_ALL=C
for file in "$(ls /usr/bin | awk          # for each
'NR%101==304936424%101')";                # matching file
do
   loc=`which $file`   # locate the file
   ldd $loc            # print out the ldd of the file
done |
grep so |
sed '/^\/usr\/bin/d' |  # remove lines starting /usr/bin
sort -u > slist.txt     # sort unique into slist.txt
```

```
#!/bin/bash
export LC_ALL='C'   # change locale
dir=$1 # put input directory into var called dir
declare -a list  # create list to hold all files
let n=0          # create variable to count
h_files=`ls -a $dir | grep '^\.' | sort`   # find hidden files
for file in $h_files      #loop through hidden files
do      #if they are symbolic links or directories, ignore
   if [ -L "$dir/$file" ] || [ -d "dir/$file" ]
   then
                continue
   elif [ ! -r "$dir/$file" ]       # unable to read
   then
                echo "$file unreadable"
   elif [ -f "$dir/$file" ]         # are file type
   then
                list[$n]="$dir/$file"
                let n=$n+1
   fi    #if they are not files, then just don't do anything
done   #do the same with non-hidden files
for ((i=0; i<$n; i++))  #loop through entire array
do
   for ((j=i+1; j<$n; j++)) #compare to all files after
   do
         cmp -s "${list[$i]}" "${list[$j]}"       #compare
         if [ $? -eq 0 ]  #evaluate the result
         then    #make second link of first if same
                ln -f "${list[$i]}" "${list[$j]}"
         fi
   done
done
```

```
#!/bin/bash
mkdir newdir            #create new directory
files=`find . -maxdepth 1 -type f`
for file in $files        #for each file in current directory
do                        #add to new directory
      cp $file newdir/$file
done                      #create new file with 2 lines
touch newdir/newfile.txt
echo "line 1" >> newdir/newfile.txt
echo `ls newdir | sort | head -n 1` >> newdir/newfile.txt
```

```
#!/bin/bash
cd task1                          #change into this directory
files=`ls . | grep '\.'`          #find all files with '.'
size1_list=`ls -l | awk '/[0-9]/{print $5}'`
size1=0                           #total for sizes in 5th col
for num in $size1_list; do        #loop through all num
   let size1="$size1+$num"        #add them up
done
for file in $files                #for each file
do
   for i in $(seq 1 ${#file})     #loop through each file
   do                             #once you get to a period
      if [[ ${file:i-1:1} = '.' ]]  #get substring of extension
      then
         echo "Ext. of $file is ${line:i-1}"
         if [[ ${file:i-1} = '.tar.gz' ]]
         then                     #untar file if ends in .tar.gz
            `tar xvzf $file`
         fi
         break
      fi
   done
done
size2_list=`ls -l | awk '/[0-9]/{print $5}'`
size2=0
for num in $size2_list; do        #loop file sizes after untar
   let size2="$size2+$num"
done
echo "Size before unzip: $size1"
echo "Size after unzip: $size2"
```