CSCB09S18 MT Seminar

Albion, Mustafa
github.com/conanap

Common UNIX Commands

- ls, stat
- pwd
- cd (and special symbols .., ., ~, -)
- ps, jobs, fg, bg, kill, ctrl + C, ctrl + D
- rm, mv, cp
- wc, cat, head, tail
- chmod
- vi / editor of choice, touch
- grep, sed
- sort, cut, uniq

File System

- Uses inodes
- Hierarchical system from root
- EVERYTHING in the FS is a file; ie directories are a special type of file
- `ln [-s] target name` to link
- file only rm’d if no more ref
- Soft link: a shortcut
- Hard link: a file that has same inode entry
- file permissions - rwxrwxrwx
    - user, group, other users
    - directories use drwxrwxrwx to denote dir

/dev/null

- System’s black hole
- ditch any output you don’t want in here

Shell (sh)

I/O Redirection

- > for overwrite redir
- >> for append redir
- < for input redir
- 1 = stdout, 2 = stderr
- ls 2>&1

Pipe

- Pass output of a command to another
- |
- Think of “output becoming standard input”

File expansion / regex

- * = 0+ chars
- ? = exactly 1 char
- [x-y] = 1 char in range(x, y), x, y are int
- [^...] = not … chars
- ~ = home dir
- ~u = user home dir (often same as ~)

Quick Exercise

List all current directories that start with the char a, and redirect it into a file called out.txt, redirecting all errors to /dev/null

```
ls | grep a.* >out.txt 2>/dev/null
```

Shellscript

- A file with commands shell can understand
- start with #!/bin/bash // or sh or csh or whatever shell you prefer
- # = comment
- chmod before running with ./
- sh shellscripFileName won’t need chmod (why?)
- a=value => a is a variable.
  - NO SPACE (use “” or ‘’ appropriately for space)
- $a to access value of a
- `export` to export variables out of shellscript scope
- `read` to read stdin

Shellscript cont’

- $0 = programme name
- $1, $2… = argument
- $* = all arguments ONLY
- “ vs ‘:
  - ‘ = str literal
  - “ = will execute $, `, and \

If in shell

```
if <command>
then
    …
else
    …
fi
```

If in shell cont’

- if evaluates to true based on exit condition of a command, not stdout
  - 0 = true, 1 = false (because 0 = success in C)
- Use test or [ ]
- expr for math

while

while command

do

…

done

for

for varName in command and values

do

    …

done

Subroutines / funcs

```
funcName() { // no need to declare arguments

    shellscript here

    return value # stored in $?

}

funcName arg1 arg2
```

C

C

- Static weakly typed procedural language
  - Can cast almost anything into anything
  - Everything must be declared a type
- 2 parts
  - C preprocessor
    - #include header files
    - defines and undefs
  - C
- Procedural => no objects, has structs
- must have main()

C Preprocessor

#include <stdio.h> // and other required header files

#define i true // i is always true now; avoid using define when possible

#undef i // i no longer always true

Variable declaration

- ALL VARIABLES MUST BE DECLARED AND INITIALIZED
- int, bool, char, float, double, short, long
- Modifiers: unsigned, signed, long
- Special type: void, *

int a; // variable declaration; NOT INITIALIZED. (“Hey! This var exists.”)

a = 4; // variable initialized. (“Hey! Make a 4 please”)

char b = “1”; // important: double quotes (variable declared AND initialized

// (“Hey! I need a var called b and make it “1” please”)

Malloc

- Allocates memory so it persists outside of current scope
- Returns success or fail
- Can do if((a = malloc(...) == NULL) { error }

Strings and arrays

char b[24]; // an array of 24 chars = a string. Remember \0 for terminating the string

char *c; // a pointer! Therefore a string of undef length. Needs malloc

c = malloc(sizeof char * 24); // assume malloc success

// some init for char *c here to put chars in it

c[2]; // perfectly legal access

*(c + 2); // same as c[2]

if

```
if(evaluates to true) { // {} optional for 1 line
        do this
} else {
        do something else
}
```

Note: 0 is false, any non-zero value is true.

for

```
for( variable; terminating condition; step) { // {} optional for 1 line
        do this
}
```

while, do while

```
while(condition is true) { // {} optional for 1 line
        do this
}
do { // {} optional for 1 line
        do this at least once
} while(condition is true);
```

functions

```
returnType functionName(argumentType argumentName, …) {
    // function code
    return … // if applicable
}
```

printf

- equiv. Python print
- printf(string, extra args…)
- %s = string, %d = int, etc (look up)

Exercise

Create a function that takes in 2 integers, returns the sum after printing hi for 5 times.

```
int sum(int a, int b) {
    for(int i = 0; i<5; i++)
        printf(“Hi”);
    return a + b;
}
```

Memory model

Memory Model

- Blocks of memory with addresses
- Pointers are addresses

Pointers

- An address; just points to a place in memory
- * to declare; * to derefence and get the value at the memory
- can assign by taking addr of another var as well: use &
- int *a = &arrySth;

Structs and typedef

- C version of “Objects”

```
struct structName {

    // stuff in struct

};

struct structName *b;

typedef struct { … } abc;

abc *c;
```

Exercise

```
void helper( … ) {
        …
}

int main(int argc, char *argv[]) {
        char *string = “Hello”;
        helper( ... );
        printf(“%s \n”, string);
}
```

Complete the code so that only “Goodbye” is printed.
You may only edit areas with “...”