Software Construction Laboratory CS35L – Lab 1

Course Webpage: https://web.cs.ucla.edu/classes/fall18/cs35L/index.html
TA: Zhiyi Zhang
Email: zhiyi@cs.ucla.edu
Webpage: https://zhiyi-zhang.com

Session 10-1: Review

- Week 1: Linux and Shell
- Week 2: Shell Script and Regular Expression
- Week 3: Compilation, Makefile, Diff and Patch, Python
- Week 4: Debugging and C

Week 1: Linux and Shell

GNU/Linux

- Open-source operating system
  - **Kernel**: core of operating system
    - Allocates time and memory to programs
    - Handles file system and communication between software and hardware
  - **Shell**: interface between user and kernel
    - Interprets commands user types in
    - Takes necessary action to cause commands to be carried out
  - **Programs**

Files and Processes

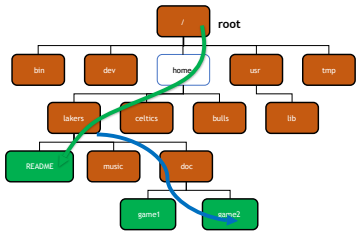
Everything is either a **process** or a **file**:

- **Process**: an executing program identified by PID
- **File**: collection of data
  - A document
  - Text of program written in high-level language
  - Executable
  - Directory
  - Devices

Absolute Path vs. Relative Path

Absolute Path (green)
- /home/laker/README

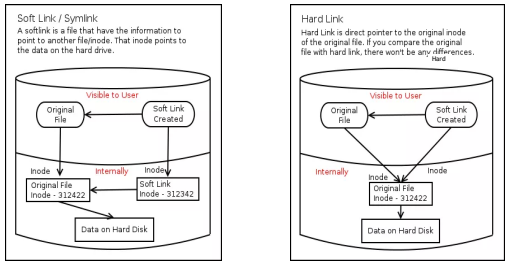
Relative Path (blue)
- You pwd: /home/laker
- doc/game2



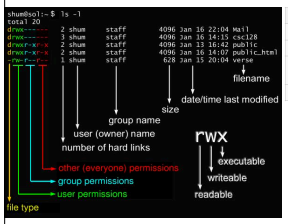
Shell commands you should be familiar with

- pwd
- cd
- mv
- cp
- rm
- mkdir
- ls
- ln
- touch
- find
- whatis
- whereis
- man
- ps
- kill
- diff
- wget

Soft Link v.s. Hard Link



chmod command



Reference	Class	Description
u	user	the owner of the file
g	group	users who are members of the file's group
o	others	users who are not the owner of the file or members of the group
a	all	all three of the above, is the same as *ugo*

chmod with symbolic and numeric operators

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

Mode	Name	Description
r	read	read a file or list a directory's contents
w	write	write to a file or directory
x	execute	execute a file or recurse a directory tree

#	Permission
7	full
6	read and write
5	read and execute
4	read only
3	write and execute
2	write only
1	execute only
0	none

Week 2: Shell Script and Regular Expression

Environment Variables

Definition
- Variables that can be accessed from any child process

Common ones:
- **HOME**: path to user's home directory
- **PATH**: list of directories to search in for command to execute

Change value:
- export VARIABLE=…

locale command

- Set of parameters that define a user's cultural preferences
  - Language
  - Country
  - Other area-specific things
- Prints information about the current locale environment to standard output

- **Locale Settings Can Affect Program Behavior**

sort, comm, and tr

sort: sorts lines of text files
- Usage: sort [options] [FILE]
- Commonly used flag: -u for unique keys
- Sort order depends on locale, e.g., C locale: ASCII sorting

comm: compare two sorted files line by line
- Usage: comm [options] FILE1 FILE2
- Comparison depends on locale

tr: translate or delete characters
- Usage: tr [options] set1 [set2]

Shell Scripts: First Line

A shell script file is just a file with shell commands
- When shell script is executed, a new child "shell" process is spawned to run it

The first line is used to state which child "shell" to use
- #! /bin/sh
- #! /bin/bash

Variables in Shell Scripts

Declared using [var_name]=[value]
- Example: var="hello"
- Notice there is no space in between

Referenced using $[var_name]
- Example: echo $var

Array in Shell Scripts

- **declare -a ARRAY** to declare an array called ARRAY
- **ARRAY[n]** to access the nth value
- **${ARRAY[@]}** to return all the values in the array called ARRAY
- **$[#ARRAY[@]]** to return the size of the array called ARRAY
- **$[#ARRAY[n]]** to return the length of the nth element in the array called ARRAY
- **unset ARRAY** to delete the array
- **unset ARRAY[n]** to delete the nth element

Accessing Arguments in Shell Script

- Positional parameters represent a shell script's command-line arguments
  - echo $1
- For historical reasons, enclose the number in braces if it's greater than 9
  - echo ${10}

Quotes in Shell Script: Three Types of Quotes

Single quotes ' '
- Do not expand at all, literal meaning
- echo ' $HOME' will output $HOME

Double quotes " "
- Weak quote. Expand backticks and $
- echo "$HOME" will output your HOME directory

Backticks ` ` or $()
- Expand as shell commands
- echo $(ls) will output the output from ls command
- echo `ls` will output the output from ls command

If Statement

- If statements use the test command or [ ]
  - In fact, [] will invoke test command
  - "man test" to check the syntax of expression
- Support boolean operations: AND: &&, OR: ||

```
if [ <test> ]
then
  <commands>
fi
```

```
if [ <test> ]
then
  <commands>
else
  <commands>
fi
```

```
if [ <test> ]
then
  <commands>
elif [ <test> ]
then
  <commands>
else
  <commands>
fi
```

Loop in Shell Script

- while loop

```
while [ <test> ]
do
  <commands>
done
```

```
#! /bin/bash
COUNT=6
while [ $COUNT -gt 0 ]
do
    echo "Value: $COUNT"
    let COUNT=COUNT-1
done
```

- for loop

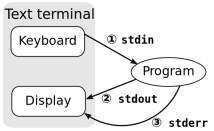
```
for var in <list>
do
  <commands>
done
```

```
#! /bin/bash

for var in `ls`
do
    echo $var
done
```

Standard Input and Output Streams

Every program has these 3 streams to interact with the world
- stdin (0): contains data going into a program
- stdout (1): where a program writes its output data
- stderr (2): where a program writes its error messages



Redirection and Pipelines

program < file redirects file to programs's stdin:
- E.g. cat <fileA

program > file redirects program's stdout to file:
- E.g., cat <fileA >fileB

program >> file appends program's stdout to file

program1 | program2 assigns stdout of program1 as the stdin of program2;
- E.g., cat <fileA | sort >fileB

Regular Expressions

Notation that lets you search for text with a particular pattern:
- Strings that starts with the letter a, ends with three uppercase letters
- Strings that are in the form of [letters numbers]@[letters numbers].[letters]

Learn Regular Expressions with simple, interactive exercises.
- Simple regex tutorial https://regexone.com/

Regular Expressions

Character	Meaning in a pattern
\	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	**Dot sign**. A wildcard. Match any single character except NULL. Individual programs may also disallow matching newline.
*	**(Kleene) Star sign**. Match any number (or none) of the single character that immediately precedes it.
^	**Hat sign**. This matches the following regular expression at the beginning of the line or string.
$	**Dollar sign.** Match the preceding regular expression at the end of the line or string.

Regular Expressions

Character	Meaning in a pattern
[...] [^...]	**Square brackets.** This matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. A hat (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. All other metacharacters are treated as members of the list (i.e., literally).
\{m, n\}, {m, n}	**Curly braces notation.** This matches a range of occurrences of the single character that immediately precedes it. \{m\} matches **exactly** m occurrences. \{n,\} matches **at least** n occurrences. \{n,m\} matches any number of occurrences **between** n and m. (n and m must be between 0 and RE_DUP_MAX (min: 255), inclusive)
\(\), ()	Save the pattern enclosed between \(\) in a special holding space. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later by the escape sequences \1 to \9. For example, \(ab\).*\1 matches two occurrences of ab, with any number of

Examples

Expression	Matches
regex	regex, anywhere on a line
^regex	regex, at the beginning of a line
regex$	regex, at the end of a line
^regex$	A line containing exactly regex, and nothing else
[Rr]egex	Regex, or regex, anywhere on a line
reg.ex	reg + any character + ex, anywhere on a line. E.g., regaex, regBex, and so on
reg.*ex	reg + any sequence of zero or more characters + ex, anywhere on a line. E.g., regex, reggex, regABC123ex, and so on
reg{2,5}ex	re + repetitive g for 2 to 5 times + ex, anywhere on a line. E.g., regaex, reggaex, reggggex, regggggex

Regular Expressions

Character	Meaning in a pattern
\n	**Backreference**. Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
+	Match **one or more** instances of the preceding regular expression.
?	Match **zero or one** instances of the preceding regular expression. E.g.,
|	Match the regular expression specified before or after. Denote different possible sets of characters. E.g., I love (cats|dogs) matches I love cats and I love dogs.
(())	Apply a match to the enclosed group of regular expressions. E.g. Use (IMG(d+))\.png to capture both IMG1.png and 1.

Matching Multiple Characters with One Expression

- ?: one or zero
- +: one or more
- *: zero or more
- {n}: n times
- {n,}: at least n times
- {m, n}: between m and n

4 Basic Concepts of Regular Expression

- Quantification
  - How many times of previous expression?
  - Most common quantifiers: ?(0 or 1), *(0 or more), +(1 or more)
- Grouping
  - Which subset of previous expression?
  - Grouping operator: ()
- Alternation
  - Which choices?
  - Operators: [] and |
- Anchors
  - Where?
  - Characters: ^ (beginning) and $ (end)

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	[:lower:]	**Lowercase characters**
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	**Space and tab characters**	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	[:upper:]	**Uppercase characters**
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Searching for Text

grep: Uses basic regular expressions (BRE)

- *"meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions" - `man grep`*

egrep (or grep -E): Uses extended regular expressions (ERE) – no backslashes needed

fgrep (or grep -F): Matches fixed strings instead of regular expressions.

sed command to replace text

- Now you can extract, but what if you want to replace parts of text? Use sed command!
- sed 's/**regExpr**/**replText**/[g]'

Example:

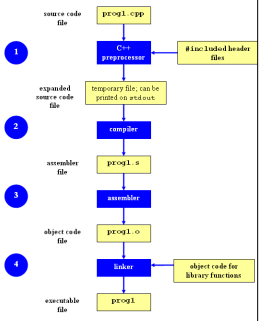
- $ echo "Let's try unix" | sed 's/unix/linux/g'
- $ echo "Welcome To CS35L" | sed 's/[0-9]/1/g'
- $ echo $PATH | sed 's/:.*//'

Review Your sameln Shell Script

Week 3: Compilation, Makefile, Diff and Patch, Python

Compilation Process

- Preprocessing (Preprocessed Code)
  - Include Headers
  - Replace symbolic constants
- Compilation (Assembly Code)
  - Preprocessed code is translated to assembly instructions specific to the targe processor architecture
- Assembly (Object Code)
  - To object code. The output consists of actual instructions to be run by the target processor.
- Linking (Executable File)
  - Link the missing pieces of the program, e.g., dynamic link libraries



Command-Line Compilation

- shop.cpp
  - #includes shoppingList.h and item.h
- shoppingList.cpp
  - #includes shoppingList.h
- item.cpp
  - #includes item.h
- How to compile?
  - g++ -Wall shoppingList.cpp item.cpp shop.cpp -o shop

Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
        g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.hpp
        g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.hpp
        g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.hpp shoppingList.hpp
        g++ -g -Wall -c shop.cpp
clean :
        rm -f item.o shoppingList.o shop.o shop
```

Rule

Comments
Targets
Prerequisites
Commands
Dependency Line

Diff command to generate a patch

- diff [original file] [updated file]
- Understand the output:
  - Line number, e.g., 12,15
  - Operations: c (replace), d (delete), a (append)
- Create a patch:
  - diff [original file] [updated file] > [blabla].patch

Unified Format in diff

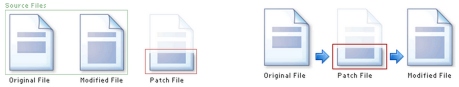
- diff -u [original file] [updated fil
  - --- path/to/original_file
  - +++ path/to/modified_file
  - @@ -l, s _l, s @@
    - @@ begin of a hunk
    - l: beginning of a hunk
    - S: number of lines the change hunk applies to for each file
    - A line with a
      - - sign was deleted from the original
      - + sign was added to the original
      - stayed the same

```
→ lab3-1 diff -u item.cpp item-new.cpp
--- item.cpp    2018-10-14 22:32:29.000000000 -0700
+++ item-new.cpp        2018-10-14 22:54:56.000000000 -0700
@@ -1,6 +1,8 @@
 #include "item.hpp"

 Item::Item()
-  : name("I am an item")
+{
+  name = "I am an item";
+  std::cout << "item constructor" << std::endl;
+  std::cout << name << std::endl;
 }
```

Applying the Patch to One File

- Syntax: patch [original file] -i [patch file] -o [updated file]
- patch command will apply the patch file to the original file and create an updated file



Applying the Patch to One Directory

- Syntax: patch -p[num] < [patch file]
- patch command will apply the patch file to the current directory
- -p is used to strip the smallest prefix containing num leading slashes from each file name found in the patch file.
  - Example: in the patch: ---/usr/tmp/project/file1~ +++/usr/tmp/project/file1
  - When use -p3, the file path becomes file1 (use th call patch command in /usr/tmp/project directory)

Python

- A scripting language
- An object-oriented language: classes, member functions, etc.
- Interpreted programming language
  - Execute instructions directly and freely by interpreter
  - No need of previous compilation to make your code machine-language instructions
- Super popular in both research and industry
  - One of the most popular language for machine learning and AI
  - Used by Google, Facebook, and other famous companies for their products

Review Your shuf.py

Week 4: Debugging and C

How to use GDB

Compile Program

- Without debugging info: gcc [flags] <source files> -o <output file>
  - Check our previous slides (week 3) if you forgot this :D
- With debugging info: gcc [other flags] **-g** <source files> -o <output file>
  - Produce debugging information that can be used by built-in debugger

Specify Program to Debug

- Use gdb command with argument: gdb <executable>
- Specify program in gdb: "gdb" and then (gdb) file <executable>

How to use GDB

Run your program

- (gdb) run
- (gdb) run [arguments]

When you are in GDB's interactive shell

- Tab to autocomplete, up-down arrow to recall history commands
- help [command] to get help information of a command

Exit

- (gdb) quit

How to use breakpoints

Setting breakpoints

- (gdb) break file1.c:6
- (gdb) break my_function
- (gdb) break [position] if [expression]

Check the existing breakpoints

- (gdb) info breakpoints
  - Or break or br or b

How to use breakpoints

Delete breakpoints
- (gdb) delete [breakpoint_id]

Enable/Disable breakpoints
- (gdb) disable/enable [breakpoint_id]

Ignore breakpoints for several times
- (gdb) ignore [break_point_id] [iterations]

More commands when using GDB

Printing data of interest at run time
- (gdb) print [format] [expression]
  - d, x, o, t for decimal, hexadecimal, octal, and binary notation

Watch changes to variables
- (gdb) watch [var_name]
- (gdb) rwatch [expression]

When a program stops at a breakpoint
- (gdb) continue | step | next | finish

c or continue: debugger will continue executing until next breakpoint.

s or step: debugger will continue to next source line.

n or next: debugger will continue to next source line in the current (innermost) stack frame.

f or finish: debugger will resume execution until the current function returns. Execution stops immediately after the program flow returns to the function's caller

More commands when using GDB

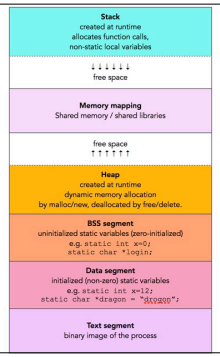
List all functions in the program
- (gdb) info functions

List the source code lines around the current line
- (gdb) list

Brief Introduction to Process Memory

Stack
- Push frame when function invoked
- Pop frame when function returned
- Stores local vars, return address, etc.

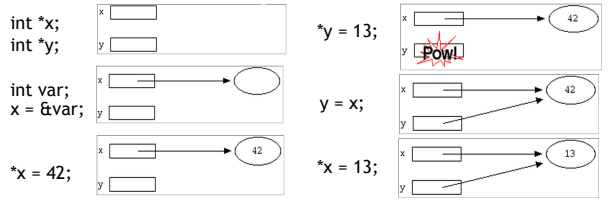
Heap
- Dynamic memory allocation



Data types, pointers, struct

- Basic data type
  - int
  - float
  - double
  - char
  - void
- Pointers
  - <variable_type> *<name>
- Struct
  - struct <name> { ... }

Pointer Example



Dynamic Memory Allocation

void *malloc (size_t size);

- Allocates *size* bytes and returns a pointer to the allocated memory

void *realloc (void *ptr, size_t size);

- Changes the size of the memory block pointed to by *ptr* to *size* bytes

void free (void *ptr);

- Frees the block of memory pointed to by *ptr*