Class 2

↳ text file : all bytes are ASCII chars
↳ binary file : otherwise

Use nano or vi text editors to edit directly on the server
↑ simple ↑ more complex

- Type man cd for help on cd command
- Type apropos 'keyword' for list of commands containing it
- Add more to view pages

pwd : print working directory
ls : list items in current directory

Class 3

cat file : show contents of file
more file
less file
tail file : last 2 lines

rm file : deletes file
mkdir dir : creates directory dir
rmdir dir : removes directory dir, but only if it's empty

rm -r dir : removes directory dir even if it is not empty
rm * : delete all files without warning!
rm -r * : delete everything without warning!

cp old new : copies old to new
↳ no warning! new is replaced if it exists
cp -i old new:
↳ warning

mv old new : renames + relocates old to new
↳ new is replaced if it exists!
mv -i old new:
↳ warning

locate file: finds where file is located
which cmd: tells you where the command is

Wildcards

* : matches 0 or more characters
↳ ex. ls * will list all files

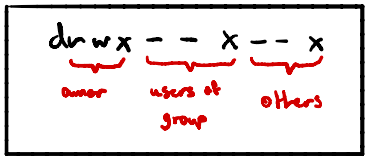
? : matches a single character
↳ ex ls file? will list file1, fileA, etc., not file11

[...] : matches any combination of characters in []
↳ ex. ls file[1-9] will list file1, file2, ..., file99

ls -la, good list view:

- = regular file
d = directory
r = read
w = write
x = execute

ex. Permissions:



left side of ls -la

ex. chmod g+w yyyy
↳ gives the group write permissions to yyyy

grep

grep regex file

grep string file
↳ searches for string in the file 'file'
↳ shows the lines containing the string

grep -v string file
↳ same as above, but shows lines that do not contain the string

Class 4

Sept. 17/20

grep main *.c
↳ searches all C programs in dir and displays all lines containing main

escape character
grep '2\|3' *
regular expression
↳ searches all files in dir looking for characters 2 or 3

Input / Output

< file(s)
↳ redirects standard input to the file or files

ex. tr 'a' 'b'
↳ reads what you type and translates every a to b and displays out line by line

tr 'a' 'b' < XXX
↳ reads from input XXX file and displays the same way

> file
↳ redirects standard output to the file, creating or overwriting

>> file
↳ redirects standard output to the file, appending

ex. tr 'a' 'b' < xxx > yyy
input output (overwritten)

echo "hello"
↳ displays hello to the std output

who | grep franek
↳ chains commands through piping, finds "franek" in list produced by who

Execution Controls

ps
↳ list all processes

ps -u franek
↳ list all processes of user franek

kill -sigl pid
↳ send signal sig to the process with id pid
↳ sig = 9 terminates process

&
↳ runs program in background

fg
↳ runs program in foreground

Bash

↳ bash shell default when logging into Moore

Class 5

Sept. 22 / 20

#!/bin/bash bash shell, beginning of the script

Shell commands

name = value (no spaces around =)

ex. name = /usr/lib/orb
echo $name will show "/usr/lib/orb"
↳ accesses the value

ex. x = `ls` (backticks) Stores directory as a string,
echo $x and prints using echo

Command-line args

$0 command name and path
$1 first argument
$2 second argument
$# number of arguments

For loops

```
files = `ls`
for i in $files
do
  echo "Echoing file name: " $i
done
```

Class 6

Sept. 24/20

Conditionals

```
if cmd
then
    # command(s)
fi
```

```
if cmd
then
    # command(s)
else
    # command(s)
fi
```

true returns 0 ex.
false returns ≠ 0 (1)

spaces around equals

```
if [ str1 = str2 ]
if [ str1 != str2 ]
if [ -r file ]   (does the file exist and is readable)
if [ expr1 -a expr2 ]   (logical and)
   [ expr1 -o expr2 ]   (logical or)
if [ ! -e $1 ]   (logical not)
   [ $x -eq 2 ]   (x is equal to 2)
   [ $x -ne 2 ]   (x is not equal to 2)
```

Class 7

Sept. 29/20

`.bash_profile`

`alias rm='rm -i'` When you type rm it executes rm -i (remove with confirmation)

`` `basename $0` `` returns just the name of command without path

`cd ..` return to prev. dir.

Class 8

Oct. 1/20

`$HOSTNAME` → ex. moore.cas.mcmaster.ca
`$SHELL` → ex. /bin/bash
`$PATH` → full path

`sed` = stream editor

ex. `sed -e 's/oldstring/newstring/g' infile > outfile`

- -e means expression (default)
- s means substitute
- replaces oldstring by newstring in infile
- g means global: replaces all occurrences
- > means output is redirected to outfile

ex. sed -e '1,8d' infile
↳ removes lines with numbers from 1 to 8

ex. sed -e '/include/d' infile
↳ removes the lines that contain include

ex. sed -e '/^FIRST/,/^LAST/d' infile
↳ removes lines between a line starting with FIRST and a line starting with LAST

^ = beginning of a line
$ = end of line

Class 9

Oct. 6 /20

Make files
↳ does not execute top-down, uses dependencies

Class 10

Oct. 8 /20

Make files cont.

ex.
```
program : main.o test.o lo.o
    gcc -o program main.o test.o lo.o
main.o : main.c
    gcc -c main.c
test.o : test.c
    gcc -c test.c
lo.o : lo.asm
    nasm -f elf lo.asm
clean :
    rm *.o program
```

→ Headers can be in any order
→ Refreshes files if dependencies change or file does not exist

make searches for file named makefile or Makefile

to use a file with any name, make -f <filename>

to see what would be executed, make -n

to execute without displaying, make -s
(silent mode)

cat -v -t -e <filename>
↳ shows tabs as ^I
↳ shows end of line as $

Class 11

Oct. 20/20

Macros (Scripts)

spaces around the "="

```
name = string
```

can use either one

To access the value, $(name) or ${name}

Internally defined macros:

- $(CC) the default C compiler
- $(CXX) the default C++ compiler
- $(LD) loader/linker

ex.
```
main.o : main.c
    $(CC) -c main.c
```

To see all internally defined macros, type `make -p`

$@ evaluates to current target

ex.
```
program : main.o test.o lo.o
    $(CC) -o $@ main.o test.o lo.o
```

$? evaluates to a list of preregs. newer than current target

ex.
```
program : main.o test.o lo.o
    $(CC) -o $@ $?
```

$<** is like **$? but for suffix rules

Class 12

Oct. 22/20

Class 13

Oct. 27/20

Hardware

Class 14

Oct. 29/20

Assembly / NASM intro

	Suffix	Prefix
Integer	d	0d
Hexadecimal	h	0x
Octal	q	0_q
Binary	b	0b

(underscores allowed)

NASM is case-sensitive for labels/variables

not case-sensitive for keywords, mnemonics, register names, etc. (instructions)

First character must be a letter or _

Syntax: [] = optional

```
[label[:]] [mnemonic] [operands] [; comment]
```

Label often on separate line

Class 15

Nov. 3 /20

Directives + Pseudo-Instructions

ex. limit EQU 100 ; limit becomes a constant with value 100

Data Definitions

ex. msg: db 'Welcome to Assembler!'

...

Instructions

ex. mov rax, rbx

ADD RCX, 10

Directives

global asm_main ; declares asm_main to be an entry point

%include "file10" ; same as copying file10 contents at this position

%include b(x) 2*x ; b(x) is replaced by value of 2*y

DB = 1 byte

DW = 1 word (2 bytes)

DD = D word (4 bytes)

DQ = Qword (8 bytes)

↳ used to declare data

ex. label1 db ABh

declares byte with value AB in hex with label "label1"

Class 16

Nov. 5 /20

used to declare uninitialized data:

RESB again B = 1 byte

RESW W = 2 bytes

RESD D = 4 bytes

RESQ Q = 8 bytes

times 10 db 0

↳ allows for loop to initialize array

↳ same as db 0,0,0,0,0,0,0,0,0,0

```
str1 DB 'This is a string'
slen EQU $-str1          ; const slen = 16
```

↳ now slen = 16