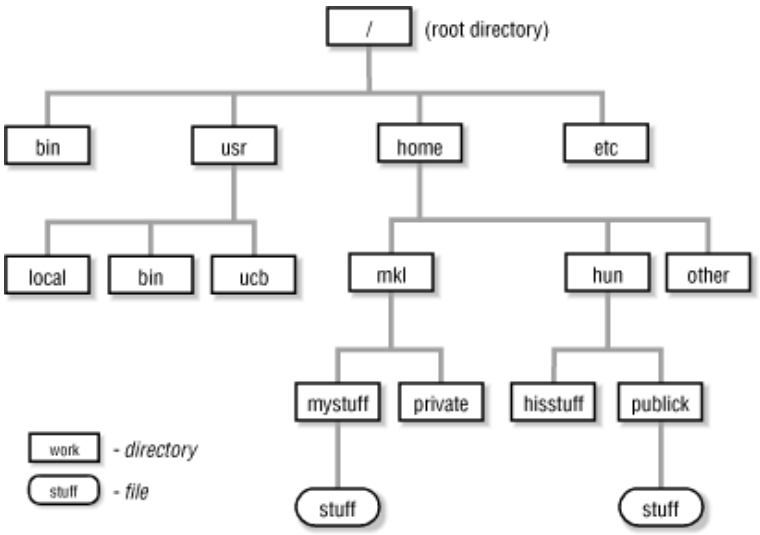
Using the Unix system

Navigating the Unix file system
Editing with emacs
Compiling with gcc

UNIX Introduction

- The UNIX operating system is made up of three parts:
  - the kernel, the shell and the programs
- The UNIX kernel manages program access to system hardware and software resources
- The shell acts as an interface between the user and the kernel.

Tree Directory Structure



Concepts

- Root directory
- Current directory
- Home directory
- Absolute path
- Relative path

Unix file security

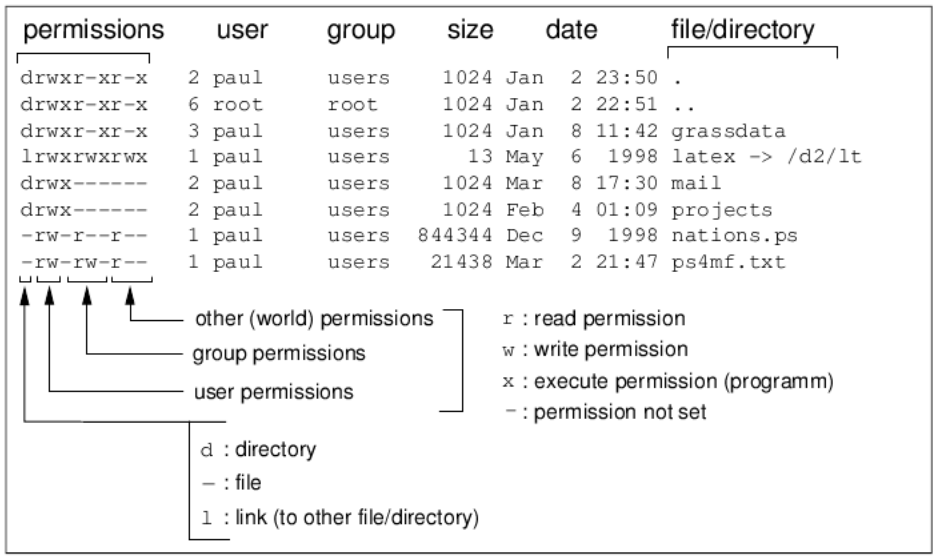
- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
- Only owner, root can change permissions
  - This privilege cannot be delegated or shared

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	1 1 0
c) **public access**	1	⇒	0 0 1

A Sample UNIX Directory Listing



Basic Unix Commands (1)

- ls — list files and directories
- ls -a — list all files and directories
- mkdir *name* — make a directory
- cd *name* — change to named directory
- cd — change to home-directory
- cd ~ — change to home-directory
- cd .. — change to parent directory
- pwd — display current directory path

Basic Unix Commands (2)

- cp *file1 file2* copy file1 and call it file2
- mv *file1 file2* move or rename file1 to file2
- rm *file* remove a file
- rmdir *directory* remove a directory
- cat *file* display a file
- more *file* display a file a page at a time
- who list users currently logged in
- * match any number of characters
- ? match one character
- man read online manual for a command

Basic Unix Commands (3)

- command > *file* redirect standard output to a file
- command >> *file* append standard output to a file
- command < *file* redirect standard input from a file
- grep *keyword file* search a file for keywords

  grep science science.txt
- wc file count number of lines/words/characters in file

  wc -w science.txt
- sort sort data (numerically or alphabetically)

  sort < biglist

  (the sorted list will be output to the screen)

Basic Unix Commands (4)

- chmod [options] *file* change access rights for named file

  For example, to remove read write and execute permissions on the file *biglist* for the group and others, type

  chmod go-rwx biglist

  This will leave the other permissions unaffected.

  To give read and write permissions on the file biglist to all,

  chmod a+rw biglist

Text editor EMACS

Text Editor emacs

- Configurable, extensible text editor
- To start emacs just "call it" typing

  emacs

- Basic editing in emacs is very intuitive
  - use arrows, "PG UP"and "PG DOWN"to move cursor
  - use DEL key to delete
  - BACK key to delete backwards
  - typing insert text at the cursor position

- To edit an existing file type

  emacs *filename*

Using emacs: keyboard commands

- We use the following abreviations

  "C" is the "Control" key

  "-" between two letters mean both have to be pressed simultaneously

- Basic commands

  C-x, C-s - save the file

  C-x, C-c - exit Emacs

Basic emacs Commands

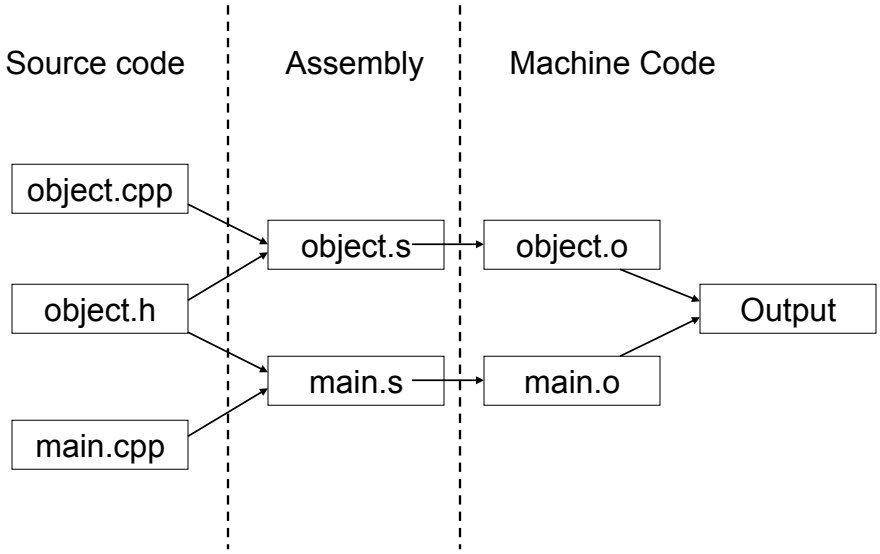
- Cursor movement
  - C-f (forward one char.)
  - C-b (backward one char.)
  - C-a (begin of line)
  - C-e (end of line)
  - C-n (next line)
  - C-p (prev. line)
  - C-v (page up)
  - alt-v (page down)
  - C-x C-j (jump to line)
- Deletion
  - C-d (delete one char)
  - alt-d (delete one word)
  - C-k (delete line)
- Paste
  - C-y (yank)
- Undo
  - C-/
- Load file
  - C-x C-f
- Cancel
  - C-g
- Save/Quit
  - C-xC-c (quit w/out saving)
  - C-xC_s (save)
  - C-xC-w (write to a new file)

Searching in Emacs

- C-s : search for a string
  - this search is incremental and goes as you search
  - typing C-s again will search for the next occurrence of the same string
  - to go back to the editing, just press any arrow key
  - after you go back, typing C-s twice resumes the search

GCC Compiler

Compilation Review



What is gcc?

- Stands for GNU C/C++ Compiler
- Popular console-based compiler for Unix platforms and others
- gcc to compile C programs; g++ for C++
  - *gcc file1.c* compile and link a C program
  - *g++ file1.c* compile and link a C++ program
  - output is an executable called a.out
  - *gcc file1.c –o file1* produce executable file1
- As always: there is man gcc

Options

- There are zillions of them, but there are some the most often used ones:
  - To compile: -c
  - Specify output filename: -o <filename>
  - Include debugging symbols: -g
  - Show all (most) warnings: -Wall
  - Be stubborn about standards: -ansi and -pedantic
  - Optimizations: -O, -O*

Options: -c

- gcc performs compilation and assembly of the source file without linking
- Output is object code files, .o; they can later be linked and form the desired executables
- Generates one object file per source file keeping the same prefix (before .) of the filename

```
gcc –c  file1.c
```

Output is file1.o

Options: -o <filename>

- Places resulting file into the filename specified instead of the default one.
- Can be used with any generated files (object, executables, assembly, etc.)
- If you have the file called source.c; the defaults are:
  - source.o if -c was specified
  - a.out if executable
- These can be overridden with the -o option.

Options: -g

- Includes debugging info in the generated object code. This info can later be used in gdb
- gcc allows to use -g with the optimization turned on (-O) in case there is a need to debug or trace the optimized code.

Options: -Wall

- Shows most of the warnings related to possibly incorrect code.
- -Wall is a combination of a large common set of the -W options together. These typically include:
  - unused variables
  - possibly uninitialized variables when in use for the first time
  - defaulting return types
  - missing braces and parentheses in certain context that make it ambiguous
- Always a recommended option to save your bacon from some “hidden” bugs.
- Try always using it and avoid having those warnings.

Options: -ansi and -pedantic

- For those who are picky about standard compliance
- -ansi ensures the code compiled complies with the ANSI C standard
- -pedantic makes it even more strict
- These options can be quite annoying for those who don't know C well

Options: -O, -O1, -O2, -O3, -O0, -Os

- Various levels of optimization of the code
- -O1 to -O3 are degrees of optimization targeted for speed
- If -O is added, then the code size is considered
- -O0 means "no optimization"
- -Os targets generated code size (forces not to use optimizations resulting in bigger code)

Options: -I

- Tells gcc where to look for include files (.h)
- Can be any number of these
- Usually needed when including headers from various-depth directories in non-standard places, eg.

  #include “myheader.h”
  vs.
  #include “../foo/bar/myheader.h”

For Your Assignments

- For your assignments, I strongly suggest to always include -Wall and –g
- Optionally, you can try to use -ansi and –pedantic, which is a bonus thing towards your grade
- Do not use any optimization options
- You won’t probably need the rest as well

Example

- For example, if you have the following source files in some project of yours:
  - fileops.h
  - fileops.c
  - process.h
  - process.c
  - parser.h
  - parser.c
- You could compile every C file and then link the objet files generated, or use a single command for the entire thing.
  - This becomes unfriendly when the number of files increases; hence, use Makefiles!
- NOTE: you don't NEED to compile .h files explicitly.

Compiling Example

- One by one:
  - gcc -g -Wall -ansi -pedantic -c parser.c
  - gcc -g -Wall -ansi -pedantic -c fileops.c
  - gcc -g -Wall -ansi -pedantic -c process.c
- This will give you four object files that you need to link and produce an executable:
  - gcc parser.o fileops.o process.o –o parser

Makefile Example

```
# Simple Makefile with use of gcc could look like this
CC=gcc
CFLAGS=-g -Wall -ansi -pedantic
OBJ:= parser.o process.o fileops.o
EXE=parser

all: $(EXE)

$(EXE): $(OBJ)
	$(CC) $(OBJ) -o $(EXE)

parser.o: parser.h parser.c
	$(CC) $(CFLAGS) -c parser.c
...
```