Overview of Programming in the UNIX Environment

Robert Spier

September 12, 2000

Abstract
An overview of some tools for programming in the UNIX environment for the CSE380 and CSE381 projects.
This Presentation

- Overview of lots of things

- This is a whirlwind tour:
  We are going to go through some things very quickly

- For more information, first try the resources in the next section.

- Then, use the newsgroups.
  slrn/netscape/outlook express
Windows

Much of this applies to the Windows Environment as well.

• Most of these tools/techniques can also be used there.

• (Unix programmers who were forced to work on Windows ported the established toolset.)
The Unix Environment...

- is infinitely flexible

- contains lots of small programs used together

- but also some really big programs too
This talk...

- Getting Help
- Manipulating Files
- Compiling
- Debugging
- Controlling and Configuring
- Other Tools
Getting Help

There are many resources already on the systems:

- **man pages**
  - [http://fling.seas/~cse380/man.cgi](http://fling.seas/~cse380/man.cgi)

- **info documents**

- **other documentation**

- **CETS**
  - reference cards (gdb, emacs, etc.)
  - answers
Finding man pages

man pages are organized into sections and subsections

1. Commands
2. System services and error numbers
3. User-level library functions
4. Device drivers, protocols, and network interfaces
5. File formats used or read by various programs
6. Games and demos
7. Miscellaneous useful information pages
8. System maintenance and operation commands
Finding man pages (2)

A named page can exist in more than one section, so sometimes you need to specify the section. For example, printf has 3 different man pages.

printf (1) - write formatted output

printf (3b) - formatted output conversion

printf (3s) - print formatted output

use man -s section page to pick a specific one.

for example: man -s 3s printf
Finding man pages (3)

use man \(-k\) keyword to perform a keyword search

$ man -k conversion
atoi        strtol (3c)    - conversion routines
atol        strtol (3c)    - conversion routines
atoll       strtol (3c)    - conversion routines
chrtbl      chrtbl (1m)    - generate character classification and conversion tables
ddi_btop    ddi_btop (9f)  - page size conversions
ddi_btopr   ddi_btop (9f)  - page size conversions
ddi_ptob    ddi_btop (9f)  - page size conversions
man page confusion

Sometimes the man pages and documentation can be quite confusing. Don’t give up after the first reading.

Also, make sure you are reading the right page.
Other sources of information

There is a ton of information out there. Other people may have asked or answered your question already.

- **The Web**
  - Google [http://www.google.com](http://www.google.com)

- **Newsgroups**
  - use [deja.com](http://www.deja.com) to search archives
Manipulating Files

• Moving Files Around

• Editing Files
Moving Files Around

**cp**  copy files

**mv**  move/ rename files

**rm**  delete files

**cat**  display a file on your terminal

**more/less**  display a file page by page

**chmod, chgrp**  for changing permissions
Editing Files

You will spend most of your time editing files, generally source code or documentation. It is important to take the time to get to know what tools your editor provides. Being able to full utilize your editor will make you much more efficient.
pico

- don’t even *think* about using pico
- it is only designed for email
- it is not a source code editor
  - does not support indentation, colors, etc.
vi

• vi is related to ex, which is related to ed

• small, fast, and arcane

• use vim, Vi Improved, instead

• supports syntax highlighting, indentation, and other advanced things.

• but - I don’t use it, so I can’t tell you too much more.

/pkg/v/vim-5.5/bin/vim command line version
/pkg/v/vim-5.5/bin/gvim gui version

http://www.vim.org (for more info and a sample .vimrc)
my recommendation for a editor

it is an exception to the unix paradigm of small programs. emacs is a complete environment, that will let you do everything from edit code, to read your mail, chat with friends, or browse the web.

Towards the end of the lecture, we are going to go into great detail about how to use Emacs.
Compiling

The third most time consuming part of any programming project (behind #1 - planning, and #2 - coding) is compiling.

You should have learned the general concepts here in CSE240.

```
gcc -o hello hello.c
```
Make

- Often it is necessary to split programs across several source files
- If only one changes, you don’t need to recompile them all
- Just recompile the one that changed, and re-link

Solution: use make
Make (2)

Make and Makefiles are a topic large enough for an entire presentation by themselves. For more info, please see the “Getting Started with The Makefile” document on the 380 web page, and the make documentation.

make is a program which operates on files called Makefiles. These files need to be named 'Makefile'

A Simple Makefile: (incorrect indentation)

```plaintext
program:  main.o data.o io.o
gcc main.o data.o io.o -o program
main.o : main.c data.h io.h
gcc -c main.c
data.o : data.c data.h
gcc -c data.c
io.o : io.c io.h
gcc -c io.c
```
Parts of the Makefile

- Macro Lines
- Dependency Lines
- Shell Lines
Another Makefile

# Makefile for The OS Project
#
OBJS = main.o data.o io.o # all object files
CC = /pkg/bin/gcc
CFLAGS = -O2
LDFLAGS = -lm

program: $(OBJS)
    $(CC) $(LDFLAGS) $(OBJS) -o program

%.o : %.c
    $(CC) $(CFLAGS) -c $<

# Dependencies for object files
main.o : main.c data.h io.h
data.o : data.c data.h

clean:
    rm *.o
The C Pre-Processor

The C compilation process has several stages:

1. preprocessing

2. compilation

3. assembly

4. linking

The pre-processor (cpp) is responsible for including the include files, and dealing with other pre-processor directives. All pre-processor directives must be at the beginning of a line, and they start with # (pound, hash, sharp, or octothorpe).
Substitution with \#define

#define PI 3.14
#define HELLO_WORLD "Hello World!\n"
#define ABS(x) ( x < 0 ? (-x) : x )

void main() {
    printf(HEHELLO_WORLD);
    printf("Pi is \%f\n", PI);
    printf("ABS of 10 is \%d\n ABS of -22 is \%d\n",
           ABS(10), ABS(-22));
}
Substitution with \#define (2)

becomes...

```c
void main() {
    printf("Hello World!\n");
    printf("Pi is %f\n", 3.14);
    printf("ABS of 10 is %d\nABS of -22 is %d\n",
        ( 10 < 0 ? (-10) : 10 ),
        ( -22 < 0 ? (- -22) : -22 ));
}
```

That would have been a pain to type. This also makes it easier to change things. \#defines are often used for numerical constants (i.e. pi, or error codes.)
Conditional Compilation

```c
void main() {
    printf("Always!\n");
    #ifdef IAMDEBUGGING
        printf("only if debugging\n");
    #endif
}
```

gives us...

```bash
$ gcc cppmacros1.c
$ ./a.out
Always!
$ gcc -DIAMDEBUGGING cppmacros1.c
$ ./a.out
Always!
only if debugging
```
gcc flags

-W  enable common warnings

-Wall enable all warnings

-g  include debugging info

-O2  optimize

-E  only run preprocessor

-S  stop after compilation, do not assemble

-o  choose output file
gcc flags (2)

-c  compile

-I  add to include <> file path

-l  link in a specific library

-L  add to library search path
file extensions

It is important to use consistent and proper file extensions with gcc. It will by default do different things with a file depending on the extension. (Weird things can happen if you use the wrong extension.) See the gcc man page for more.

.c  C source; preprocess, compile, assemble

.C,.cc,.cxx  C++ source; preprocess, compile, assemble

.s  Assembler source; assemble

.h  Preprocessor file; not usually named on command line

.o  Object file; passed to linker
Debugging

You will spend lots of time debugging. It makes sense to get good at it.

PRACTICE IT!
gdb

gdb is the GNU debugger, it works with gcc , the GNU C compiler.

important keystrokes:

b - set breakpoint
b <function name>
b <line number>
delete <breakpoint number>
help <thingy>
n - next line, step over
s - next line, step in
ni/si - next instruction
p/FMT - print
x - examine memory
bt - show backtrace
gdb demonstration

It’s hard to show how to use gdb with pictures, so we’ll do a very short demonstration here.
ddd: The Data Display Debugger

ddd is a gui frontend to gdb. It will allow you to visually see your data structures and how they relate to each other. It is full featured, and more powerful than the Visual Studio debugger.

The only downside to it is that it is slow. But since it shows you the gdb commands it is using when you click something, you can learn gdb from it.
void main() {
    int r;
    int p[4] = {11, 22, 33, 44};
    char *q[3] = {"dog", "cat", "fish"};
    r = func( p[0] );
}
electric fence

Electric Fence helps you detect two common programming bugs: software that overruns the boundaries of a malloc() memory allocation, and software that touches a memory allocation that has been released by free().

It was originally developed at/for Pixar.

An obviously bad program, right?

```c
#include <malloc.h>
#include <stdio.h>
void main() {
    int *p = malloc(100);
    *(p+1000) = 7;
    puts("Didn't crash!\n");
}
```
electric fence (2)

$ gcc efence1.c
$ ./a.out
Didn’t crash!

That’s not right. We *know* that program was bad. This *will* come back to haunt us later.

**Fix it now!**
The magic of electric fence

$ gcc efence1.c -L/home4/c/cse381/lib -lefence
$ ./a.out

Electric Fence 2.0.5 Copyright (C) 1987-1998
Bruce Perens.
Segmentation Fault

If you run that in gdb, it will stop on the line causing the problem.

Electric Fence will be a lifesaver. Use it.
Controlling / Configuring your environment

This is another topic which could fill an entire presentation...

Your main interface to UNIX is the shell.

Eniac and the CIS machines have several different shell’s installed on them. Unless you have a specific reason not to, you should make sure that you are using either bash or tcsh. (Both have tab completion.) Use the chsh program to change your shell. I recommend bash.

Time permitting: show a customized .profile file here.
Job Control

You should know by now that UNIX is a multiprogramming / multitasking environment. You can run more than one program at once. Your shell handles the logistics of this for you, but it is called job control.

```
$ perl -e'while(1){}';
^Z
[2]+ Stopped /pkg/bin/perl5.00503 -e'while(1){}'
$ jobs
[1]- Running /pkg/e/emacs-20.4/bin/emacs /var/tmp/efence1.c
&
[2]+ Stopped /pkg/bin/perl5.00503 -e'while(1){}'
$ bg
[2]+ /pkg/bin/perl5.00503 -e'while(1){}'
$ jobs
[1]- Running /pkg/e/emacs-20.4/bin/emacs /var/tmp/efence1.c
&
[2]+ Running /pkg/bin/perl5.00503 -e'while(1){}' &
$ fg
/pkg/bin/perl5.00503 -e'while(1){}';
```
Tools: piping, grep, head, tail

Grep is a super-powerful super simple tool. It searches through files line by line, returning things that match. Here, we pipe the output of grep into head and tail.

```
$ grep cow /usr/dict/words | head -5
cow
coward
cowardice
cowbell
cowbird
$ grep cow /usr/dict/words | tail -5
cowry
cowslip
locoweed
Moscow
scowl
```
Tools: grep (2), wc, sort

Grep also supports regular expressions:

```
$ grep ’^zo’ /usr/dict/words
    zoo
    zoology
    zoom
```

wc counts words, lines, and characters. sort, sorts.

```
$ wc -l *[ch] | sort -n
  26 common.h
  91 mutex.c
  241 barber.c
  358 total
```
Source Control: rcs

The Revision Control System (RCS) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, and form letters.

It is good for keeping track of what you’ve done, and having a record of changes, so if something breaks, you can track down what code changed.

RCS files are named the same as the file you are putting under version control, but with .v appended to it.
rcs example

ci -l checks in the current version and “locks” it for your use. When you are done with it, you should use ci -u to check it in unlocked.

$ cat test.c
void main() { puts("Hi!\n"); }
$ ci -l test.c
test.c,v <-- test.c
enter description, terminated with single ‘.’ or end of file:
NOTE: This is NOT the log message!
>> a sample program
>> .
initial revision: 1.1
done
$ vim test.c

rcsdiff displays the difference between version of the file. You can tell it to show the differences between any two arbitrary versions. The -u specifies the format of the diff.
rcs example (continued)

$ rcsdiff -u test.c

RCS file:  test.c,v
retrieving revision 1.1
diff -u -r1.1 test.c
--- test.c  2000/09/11 02:41:19  1.1
+++ test.c  2000/09/11 02:42:14
@@ -1 +1 @@
-void main() { puts("Hi\n"); }
+void main() { printf("Hi\n"); }

$ ci -u test.c

test.c,v  <--  test.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> changed from puts to printf
>> .
done
Version Control: cvs

CVS is another version control system, which allows you to keep old versions of files (usually source code), keep a log of who, when, and why changes occurred, etc., like RCS or SCCS. Unlike the simpler systems, CVS does not just operate on one file at a time or one directory at a time, but operates on hierarchical collections of directories consisting of version controlled files. CVS helps to manage releases and to control the concurrent editing of source files among multiple authors. CVS allows triggers to enable/log/control various operations and works well over a wide area network.
**rCS vs. CVS**

RCS generally works “in place” (i.e. everyone is working on the same physical files at once.) RCS acts as the gatekeeper, allowing only one person to edit a file at a time.

CVS works remotely. There is a central repository where the files and version information are stored. Each user has their own local copy, which gets synchronized with the repository periodically.

If you have any questions about which one might be more appropriate for you or your team, please ask a TA.
More Emacs

Time permitting... we do the rest of the slides which go into more detail about Emacs.
Escape Meta Alt Control Shift

Emacs uses the following notation for keybindings:

\textbf{M-k} Meta (Alt or Esc) \textit{k}

\textbf{C-k} Control \textit{k}

\textbf{C-x C-c} Control X followed by Control \textit{c}

\textbf{C-x k} Control X followed by \textit{k}

If your keyboard doesn’t have a Meta key, use the Alt key. If that doesn’t work, Emacs also understands Esc followed by the key (Esc \textit{k}) to be the same as M-k.
Execute Extended Command

Emacs supports too many commands to have them all bound to keys. M-x means execute-extended-command. For example, if you wanted to call the replace-regexp function, type:

M-x replace-regexp

Tab completion is supported.
Emacs GUI

If you’re scared of Emacs already, don’t be. It is possible to just use a fraction of its features, and be quite happy. You don’t have to memorize all the key bindings.

If you are running Emacs in X or under Windows, there is a fully functional GUI with pulldown menus and dialog boxes.
Getting Help in Emacs

C-h ? help

C-h m help on this mode

C-h k help on a specific key

C-h f help on a function

C-h a apropos - search for a function
Loading and Saving Files

C-x f find file (and load it into a buffer)

C-x s save file

C-x w write file

C-x C-c prompt to save all files and then quit
Tab Completion and History

Tab completion allows you to type the first few characters of a filename, hit tab, and emacs will auto-complete it if it can, otherwise it will show you a list of possible matches.

Emacs also keeps track of your previous entries. You can use the up and down arrows (or M-p and M-n) to scroll up and down through them.
Buffers

- emacs can have multiple buffers open

- most buffers are directly associated with files

- some can be running programs (shell, make) or internal things

\textbf{C-x b} select buffer

\textbf{C-x k} kill buffer

\textbf{C-x C-b} list buffers
Emacs supports multiple windows inside a single frame.
Windows (2)

C-x 0 hide current buffer

C-x 1 hide other buffer

C-x 2 split frame top/bottom

C-x 3 split frame left/right
Frames (2)

Frames allow you to have multiple emacs windows on the screen, all sharing the same buffers. There is no reason to start multiple copies of emacs up. (Starting multiple copies is a bad thing because of potential conflicts if you edit the same file in multiple copies.)

C-x 5 2 duplicate this buffer in a new frame

C-x 5 b open buffer in new frame

C-x 5 f visit file in new frame

C-x 5 0 delete frame
Search and Replace

Emacs has amazingly powerful search and replace tools.

query-replace  M-x %

replace-string

replace-regexp

query-replace-regexp

isearch-forward  C-s

isearch-backwards  C-r
Modes

Emacs categorizes things based on modes. Each mode is a collection of elisp code and variables. Sample modes include:

- elisp-mode
- c-mode
- c++-mode
- text-mode

Emacs also knows about the filename extensions associated with each mode. So when you open a .c file, it automatically sets the buffer to c-mode.

You can usually manually switch modes by using M-x modename-mode.
Speedbar

```
~/381/barber/
[
+] Makefile
[-] barber.c #
  + main
  + mkcontext
  + customer
  + always
  + barber
  + barber_init
  + setup_signals
  + timer_interrupt
  + scheduler
[-] common.h
[-] mutex.c #
  + sema_trywait
  + sema_post
  + sema_wait
  + sema_init
  + mutex_destroy
  + mutex_unlock
  + mutex_init
  + mutex_lock
  + TestAndSet

<< SPEEDBAR 4 >>
```
Speedbar (2)

Speedbar is a nifty little utility (mode) that is part of Emacs. It works with most of the programming related modes (c-mode, java-mode).

It shows you a list of files in the current directory, and allows you to click on them and switch between them. It also scans them for function names, and allows you to jump directly to a function.
font-lock-mode

font-lock-mode provides syntax highlighting. syntax highlighting makes code easier to read and helps catch typos.

```c
#include <stdio.h>

void main() {
    printf("hello world!\n");
}
```

---

--:*** foo.c (C)--L7--All-------
Configuring Emacs

Emacs is infinitely configurable using elisp.

We have provided you a sample .emacs file which sets some sensible and useful defaults.

Configuring emacs is an art, but there is a shortcut.

M-x customize provides a clickable interface for changing settings, which then get saved to your .emacs file. One thing you may wish to do is change the colors used for font-lock-mode.

We have provided two extra keybindings for you:

f4  goto-line

f5  compile/recompile
vi

#include <basic_vi_key_commands.txt>
The End

Any questions?
RCS Version Page