WWU Computer Science
UNIX Software Development

This document is for those who have done no software development under UNIX. It is intended to give you a quick jump on using UNIX and to make it easy to edit, compile and run programs.

Under UNIX, the "standard" development rule is "One job, one tool." That means that the C compiler does not do anything other than compile. Debugging is done by a different tool, editing (writing or changing your program) is done by yet another tool and so forth. There are recent programs that make it look different, but they usually have a program to do the GUI (graphical user interface) and it then calls all the other tools to get the jobs done. So in some way, the GUI tool is doing just one job, the GUI!

This document will give you a brief introduction to several things you will need to operate in the UNIX environment. First you need to be able to edit files. Many UNIX installations have many different editors. The My favorite is "emacs". Emacs in some flavor is available for almost every computer and operating system made! The lab computers also have other editors like "vi", 'vim (VIimproved)", "gedit", "pico", "nano", "kate" and many others. Vi is a standard editor and every UNIX person should know how to do basic editing in vi. "gedit" is a simple editor for the GNOME environment.

Next, you need to be able to interact with the system to "see" your files and do other things. This includes compiling your programs and getting them ready to run, printing your programs and so forth.

Finally, this document will give you an introduction to compiling a program from inside emacs and running gdb, the GNU debugger.

1 EMACS

Emacs is an editor that is available (in some "flavor") for almost every machine on campus. (It may not be there but is available for it if someone wanted to compile and install it there.) The "best" version is "GNU emacs". It is a very powerful editor that does much more than just editing files. When run on a X11 desktop, it includes many point and click features, including text copying, text deletion, point at the spot to edit and menus of many functions. The most recent version is 21.2. The source code for the editor is found at ftp.gnu.org in the directory /pub/gnu/emacs.

To start off, emacs uses "control characters" for many commands. The character "control-g" (the character "g" with the control key down) is written as "C-g". The "C-g" runs the "quit" command. This command is very useful if you end up in some strange mode. Hit "C-g" several times and you will often get out of trouble. Also, you should know how to exit emacs. To exit emacs, you need to type the multi-key sequence "C-x C-c".

If you are using "GNU emacs" you can get help using the "C-h" help command. It requires a second character to tell it which type of help you want. The most useful one to the beginner is "C-h t" which runs a tutorial. (If the C-h is the same as "delete" and you can not run the tutorial using "C-h t", try the following: "M-x help-with-tutorial" where the "M" is usually the ESC key. So "M-x" could be the same as "ESC x".) The tutorial is very good and makes you use emacs to read the tutorial. This is what I recommend for the new user. Run emacs (just type emacs to your prompt) and then type "C-h t" to get the tutorial.

Finally, if you are using GNU emacs, try the following: "C-u 7 M-x hanoi RET" where "RET" is the return key.

2 UNIX Commands

Most people interact with UNIX using a program called a "shell". It is a "command line" program where you type the name of a program and it then runs that program. Most people at WWU are given the shell "csh" (the C-shell) as their login shell. It has many features that I will not discuss. (If you want to learn about csh, read the "manual page.") Another popular shell at WWU is called "bash." (Bash has more features than csh.)

This section gives short descriptions of several commands that are available from csh (and bash) that you will need. These commands helps you manipulate files and directories. I am describing just the basic usage. For most commands, they have options that change the behavior of the command. The csh uses "*" as a "wild card" to represent any number of characters in a file name. When you login to a UNIX system, you are using a directory called your "home" directory. It is also your initial "working directory". Unless your command says different, commands use the working directory to find files.

ls This command used as "ls" lists the files in your "working directory". Use the command "ls -l" to get a long listing of your files. It can also be used to list any directory in the system.

cp This command copies one file to another. For example, "cp prog1.c prog2.c" makes a copy of "prog1.c" and makes a new file called "prog2.c".

mv This command just renames (moves) a file. For example, "mv prog1.c prog2.c" just changes the name of the file from "prog1.c" to "prog2.c".

rm This command removes files. BE CAREFUL with this command. "rm *" will remove ALL files in your working directory. Once they are removed, you *can not* get them back.

mkdir This commands makes a new directory. For example, "mkdir cs347" will make a new directory called "cs347" and put it in the working directory.

rmdir This command will remove an empty directory.

cd This changes your working directory. For example, "cd cs347" will make the directory cs347 your working directory. (This command is part of csh.) If you just say "cd" with no directory name, it will make your home directory the working directory.

pwd "pwd" will print the name of your working directory. For example, "pwd" may print "/home/phil" if "/home/phil" is the working directory. Notice, the full name of a directory starts with the "/" character. The full name of a file is the full name of the directory followed by "/" and the file name. For example, "/home/phil/cs347/Prog1.c" is a full file name.

lpr This command prints files on a printer. (Some versions of UNIX use the command "lp".) You should ask a consultant about printing on a particular UNIX machine.

more "more prog1.c" displays the file prog1.c one page at a time on your terminal. After displaying a page, it waits for you to respond to the "more" prompt and the end of the page. A carriage return makes more display the next line, a space character makes it display another page.

chmod This commands changes which users can have access to your files. For more information, read the manual page.

The next few commands are commands that do other things. Most of them have very short descriptions. They are included so you know some commands but if you want more information, you can get it.

man This command gets you HELP. "man ls" will give you the "manual page" on ls. It tells you almost anything you want to know about the command "ls". For more information on any of the commands in this document say "man command". If you want to find help on a particular topic say "man -k topic". For example, "man -k file" will try to tell you all commands which have the word "file" in their short description.

date This prints the current date and time.

finger This lets you see who is logged on.

ps This shows you programs that are running. "ps -aux" will show you all programs running on the machine.

Finally, you need to know how to compile and run C programs. First, edit your programs with an editor (emacs). Let us assume you are editing the program in the file "prog1.c". To compile the prog1.c give the command:

```
gcc prog1.c
```

That one command will compile and if there are no errors, link your program. The executable file will be named "a.out". To run it, it is best to type "./a.out".

To specify the name of the executable file use the command:

```
gcc -o prog1 prog1.c
```

The executable file will be named "prog1". This should be all you need to know about the C compiler for the first few weeks of the quarter. (There is a newer C compiler called "clang". You can use it if you like, but you should know how to use gcc.)

3 Compiling using Emacs

Emacs can be used to compile C programs on the UNIX systems. Also, when errors occur in the compilation, emacs can place your cursor at lines where the compiler detected errors.

First, before talking about how to do the commands, we need to review how to talk about key strokes in emacs. Control characters are denoted by the form "C-g" (control g). There is also the idea of a "meta" key that works similarly to the control key. The "meta" keys are denoted by the form "M-x" (meta x). On most terminals, there is no "meta" key. To allow for the "M-x" characters on most terminals, "M-x" maps to the key sequence "ESC x" where "ESC" is the escape key.

Assume you are editing a file named "prog1.c". To compile the file from emacs, give the command "M-x compile". The first time this command is given for each emacs session, emacs will respond to the command with

```
Compile command: make -k
```

in the bottom line of the display. (After you learn how to use make(1), you will not have to do the following.) Delete the "make -k" using the delete key or using the commands "C-a C-k". (C-a == move cursor to beginning of line. C-k == kill characters to end of line.) Then type your usual UNIX command for compiling prog1.c. That would be

```
gcc prog1.c
```

or the form

```
gcc -o prog1 prog1.c
```

A carriage return after typing the compile command will run the compiler. (emacs will remember this new command for future executions of the emacs compile command. For further executions, all that is required is the carriage return.)

To do the compile, emacs will split your screen into two windows. One window is the original program, the other window is the "compile log". You will see the "gcc ..." command given by emacs and the results. When the compilation is completed you will see the message

```
Compilation finished at ....
```

where the dots represent the date and time. (You may need to scroll the compile log window to see all the results. Consult your emacs documentation for dealing with two windows and scrolling windows.)

If you had errors in your compile, emacs will help you find your errors. With your cursor in the program window, give the command "C-x `". (That is the single back-quote (`), not the single quote (').) Emacs will then move your program window to the line of the first error message in the compile log. It will also move the compile log so that the first error is displayed at the top of the window. You then edit as usual the program to correct your error. When you have finished dealing with the first error, give the "C-x `" command to go to the next error. Again, emacs will place your program at the line of the error and show the error at the top of the window. You continue this edit and "goto next error" until you are ready to recompile. To recompile, just give the compile command again.

Finally, if you like this but don't like typing "M-x compile" all the time, you can configure emacs by creating a file called ".emacs" in your home directory. In this file, add the following line:

```
(global-set-key "\C-x\C-m" 'compile)
```

This allows the keys "C-x C-m" to run the compile command. Another favorite key stroke addition that can help is:

```
(global-set-key "\C-xg" 'goto-line)
```

4 GDB

"gdb" is the GNU debugger. It is the debugging tool. It allows you to do many useful things. To start out, it is the best way you have to find out where your program is crashing with a "segmentation violation" error. To use gdb, add the "-g" flag to your compile line. So your compile line would look something like:

```
gcc -g -o prog prog.c
```

Also, to get your environment ready for this, add the following line to your .cshrc file in your home directory. (Do this is you are using CSH or don't know what shell you are using. Bash users have a slightly different way to set this.)

```
 limit coredumpsize unlimited
```

After this change, logout and login again. (There are other ways of getting your CSH updated, but this is the easiest to tell you.)

Now, with the above two changes, you then develop your programs. If you get a "segmentation violation", you should see the message "core dumped" and see a new file called "core". (On NetBSD you will see a file called "prog.core".) Now run gdb as follows:

```
gdb prog core
```

It will tell you which file and what line of your program the problem occurred. You can even look at the values of the variables at the time of the the problem.

The following is an example session on a Ubuntu machine:

```
---->~/347/gdb
Linux-01[201]$ ls
p1.c

---->~/347/gdb
Linux-01[202]$ cat p1.c
#include <stdio.h>

void f1 (int *a)
{
   printf ("a is %d\n", *a);
}


int main()
{
  int *myptr=NULL;

  f1(myptr);

  return 0;
}

---->~/347/gdb
Linux-01[203]$ gcc -g -o p1 p1.c

---->~/347/gdb
Linux-01[204]$ ./p1
Segmentation fault (core dumped)

---->~/347/gdb
```

```
Linux-01[205]$ ls
core  p1  p1.c

---->~/347/gdb
Linux-01[206]$ gdb p1 core
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/phil/347/gdb/p1...done.
[New Thread 17303]

warning: Can't read pathname for load map: Input/output error.
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6...Reading symbols from /usr/lib/debug/l
done.
Loaded symbols for /lib/x86_64-linux-gnu/libc.so.6
Reading symbols from /lib64/ld-linux-x86-64.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib64/ld-linux-x86-64.so.2
Core was generated by `./p1'.
Program terminated with signal 11, Segmentation fault.
#0  0x0000000000400504 in f1 (a=0x0) at p1.c:5
5          printf ("a is %d\n", *a);
(gdb) print a
$1 = (int *) 0x0
(gdb) quit
```

The following is a list of commands that are useful in gdb, available at the "(gdb)" prompt:

quit Exit gdb.

run Run (or rerun) the program being debugged. You can add command line parameters like "run arg1 arg2 arg3".

step Run the next line of code. If it is a function call, stop in the function.

next Run the next line of code. Don't stop in any functions called.

print Prints values of variables.

help Gives simple help on how to use gdb.

5 Printing

Note: This section on printing is specific to the CS lab at WWU.

To print a file from Linux, you use the command "lpr". (Read the man page for all possible options.) You do need to know how to select which printer on which to print. This is done by use of the "-P" flag. For example:

```
lpr -PCF405 file1
```

prints "file1" on the printer in CF 405. (Printers for student use are only found in CF 405 and CF 162.) If you don't use the "-P" flag, it may defaults to the printer you don't want to use. To have a "standard" printer for you, define the "PRINTER" environment variable with the name of your preferred printer. The possible printer names valid with the "-P" flag and as values for the "PRINTER" environment variable are CF405 and CF162. You should only print postscript files this way. Use a2ps (see below) to print ASCII (e.g. source) files. If you use this command directly, you could waste a lot of paper.

To see the print jobs in the queues, you use the "lpq" command. For example:

```
lpq -PCF405
```

will show the list of jobs waiting to be printed on the cf416 printer. It should print something like:

```
Rank   Owner      Job  Files                                 Total Size
1st    phil       1    Makefile                              685 bytes
```

Notice the job number. With that job number and the program "lprm" you can delete a print job from the queue. For example:

```
lprm -PCF405 1
```

will delete job 1 from the CF405 print queue.

Also, there are other programs that may help you print. First, some editors can directly print a copy of your source code. "a2ps" is a program that takes text files and prints them in a nice format on a postscript printer. All our printers understand postscript. You may need to define the environment variable PRINTER for some of the editors. "a2ps" also has a "-P" flag just like the "lpr". "a2ps" also takes a "-o filename" flag if you want to generate a postscript file instead of print it. You can view the postscript files using either "gs" (ghostscript) or the KDE version called "okular". The second one produces nicer output. "okular" also allows you to view .pdf files.

Feedback on how to improve this document is requested.

Last modified: 8 Jan 2019