Unix Tutorial Slides

CSE 326 Quiz Section

April 4, 2002

With much thanks to the UW ACM

IWS (Instructional Work Servers)

- There are 4 instructional Unix servers:
  - `ceylon`, `fiji`, `sumatra`, and `tahiti`
- Accessing the servers:
  - Terminal Programs:
    - telnet (insecure; cannot be used)
    - ssh (via the TeraTerm or Putty programs from Windows)
      - Start `->` Program Files `->` Desktop Tools `->` TeraTerm
  - File Transfer Programs
    - ftp (insecure; cannot be used)
    - `\\<server name>\<username>` from Start -> Run menu
      - e.g -- `\\fiji\zanfur`
    - Secure file transfer (from C&C)
  - Xwindows
    - Run `xgo` from the command prompt
    - Come to the ACM tutorial!

This tutorial provided by UW ACM
http://www.cs.washington.edu/orgs/acm/
Questions to jdeibel@cs, hctang@cs, zanfur@cs, awong@cs

Logging In

- Which server you use (almost) doesn't matter – all four allow access to your files
- Although your Windows and Unix usernames (and passwords) are the same, *they are two separate accounts*
  - Your z: drive is not your Unix account
- Connecting:
  - We'll connect to the Unix machines via ssh
  - After connection, you are presented with a login prompt
  - After logging in, you're placed in your home directory (where your personal files are located)

Setting Up Your Environment

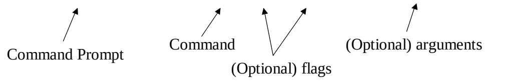
- To set up your Unix environment, follow the setup instructions on the first programming project
- To get the full benefit of `/uns`, you can run the `/uns/examples/setup-tutorial` script
- It's a good idea to look at what's in `/uns/bin` – there are many useful tools there:
  - `xemacs`
  - `ddd`
  - And much, much more …

The Command Prompt

- Commands are the way to "do things" in Unix
- A command consists of a command name and options called "flags"
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

```
[prompt]$ <command> <flags> <args>
```

```
fiji:/u15/awong$ ls –l -a unix-tutorial
```



Note: In Unix, you're expected to know what you're doing. Many commands will print a message only if something went wrong.

Two Essential Commands

- The most useful commands you'll ever learn:
  - `man` (short for "*man*ual")
  - `info`
- They help you find information about other commands
  - `man <cmd>` or `info <cmd>` retrieves detailed information about `<cmd>`
  - `man –k <keyword>` searches the man page summaries (faster, and will probably give better results)
  - `man –K <keyword>` searches the full text of the man pages

```
fiji:/u15/awong$ man –k password
passwd     (5)  - password file
xlock      (1)  - Locks the local X display
                  until a password is entered
fiji:/u15/awong$ passwd
```

Directories

- In Unix, files are grouped together in other files called *directories*, which are analogous to *folders* in Windows
- Directory paths are separated by a forward slash: /
  - Example: `/u10/jdeibel/classes/cse326`
- The hierarchical structure of directories (the directory tree) begins at a special directory called the *root*, or /
  - *Absolute paths* start at /
    - Example: `/u10/jdeibel/classes/cse326`
  - *Relative paths* start in the current directory
    - Example: `classes/cse326` (if you're currently in `/u10/jdeibel`)
- Your home directory is where your personal files are located, and where you start when you log in.
  - Example: `/u10/jdeibel`

Directories (cont'd)

- Handy directories to know
  - ~ Your home directory
  - .. The parent directory
  - . The current directory
- `ls`
  - *LiS*ts the contents of a specified files or directories (or the current directory if no files are specified)
  - Syntax: `ls [<file> … ]`
  - Example: `ls backups`
- `pwd`
  - *P*rint *W*orking *D*irectory

Directories (cont'd further)

- `cd`
  - *C*hange *D*irectory (or your home directory if unspecified)
  - Syntax: `cd <directory>`
  - Examples:
    - `cd backups/unix-tutorial`
    - `cd ../class-notes`
- `mkdir`
  - *MaK*e *DIR*ectory
  - Syntax: `mkdir <directories>`
  - Example: `mkdir backups class-notes`
- `rmdir`
  - *R*e*M*ove *DIR*ectory, which *must be empty*
  - Syntax: `rmdir <directories>`
  - Example: `rmdir backups class-notes`

Files

- Unlike Windows, in Unix file types (e.g. "executable files, " "data files," "text files") are *not* determined by file extension (e.g. "foo.exe", "foo.dat", "foo.txt")
- Thus, the file-manipulation commands are few and simple …

- `rm`
  - *R*e*M*oves a file, ***without a possibility of "undelete!"***
  - Syntax: `rm <file(s)>`
  - Example: `rm tutorial.txt backups/old.txt`

Files (cont'd)

- `cp`
  - *C*o*P*ies a file, preserving the original
  - Syntax: `cp <sources> <destination>`
  - Example: `cp tutorial.txt tutorial.txt.bak`
- `mv`
  - *M*o*V*es or renames a file, destroying the original
  - Syntax: `mv <sources> <destination>`
  - Examples:
    - `mv tutorial.txt tutorial.txt.bak`
    - `mv tutorial.txt tutorial-slides.ppt backups/`

Note: Both of these commands will over-write existing files without warning you!

Shell Shortcuts

- Tab completion
  - Type part of a file/directory name, hit `<tab>`, and the shell will finish as much of the name as it can
  - Works if you're running `tcsh` or `bash`
- Command history
  - Don't re-type previous commands – use the up-arrow to access them
- Wildcards
  - Special character(s) which can be expanded to match other file/directory names
    - `*` Zero or more characters
    - `?` Zero or one character
  - Examples:
    - `ls *.txt`
    - `rm may-?-notes.txt`

Text - editing

- Which text editor is “the best” is a holy war. Pick one and get comfortable with it.
- Three text editors you should be aware of:
  - `pico` – Comes with `pine` (`Dante`’s email program)
  - `emacs/xemacs` – A heavily-featured editor commonly used in programming (*326 staff recommends this one)*
  - `vim/vi` – A lighter editor, also used in programming
- Get familiar with one *as soon as possible!*

Text - printing

- Printing:
  - Use `lpr` to print
    - Use `–h` (no header) and `–Zduplex` (double-sided) to save paper
  - Check the print queue (including Windows print jobs!) with `lpq`
  - `lprm` to remove print jobs (including Windows print jobs)
  - For the above commands, you’ll need to specify the printer with `–P<printer name>`
- Check out `enscript` (quizlet: how do you find information about commands?) to print text files nicely!
  - **WARNING:** Do *NOT* use `enscript` with postscript files!

Unix I/O

- Input:
  - stdin: usually inputted through the keyboard, it is equivalent to `cin` in C++
- Output:
  - stdout: usually sent to the monitor, it is equivalent to `cout` in C++
  - stderr: usually sent to the monitor, it is equivalent to `cerr` in C++.

NOTE: It is ***good*** programming practice to use `cerr` for error messages instead of `cout`.

Redirecting I/O

- Redirecting input: a.out < file
  - a.out will read from the file using stdin (cin).
  - This is as if the user was typing the contents of the file as input.
- Redirecting output: a.out > file
  - a.out will write any output from stdout to file.
  - The file will be created if it does not already exist and overwritten otherwise.
  - Messages from stderr will not be written to the file.
- Piping: cmd1 | cmd2
  - cause the stdout output of cmd1 to be sent as stdin input to cmd2

The Unix Philosophy

- A large set of primitive tools, which can be put together in an infinite number of powerful ways
- An example:
  - Three *separate* tools are necessary to develop software:
    - Text editor
    - Compiler
    - Debugger (You *will* need this, unless “j00 R l33t”)
  - MSVC is an “IDE” (“Integrated Development Environment”)
    - All three tools are found in one shrink-wrapped box
  - Although there are IDE’s for Unix, for this course, you will most likely use (mostly) separate tools:
    - Text editor: `emacs/xemacs` or `vi/vim`
    - Compiler: `g++`
    - Debugger: `gdb`

This tutorial provided by UW ACM
http://www.cs.washington.edu/orgs/acm/
Questions to jdeibel@cs

Compilation with `g++` 3.0

- There are actually *three* `g++`s installed on the instructional machines
  - Version 3.0.4 is the one we’ll be using for 326
  - Version 2.96 is the default
- To use the most current version, you need to call `uns-g++`
- `uns-g++` is located in /uns/bin, which is not part of your standard Unix environment
- After running the course-setup script, `g++` will default to `uns-g++`

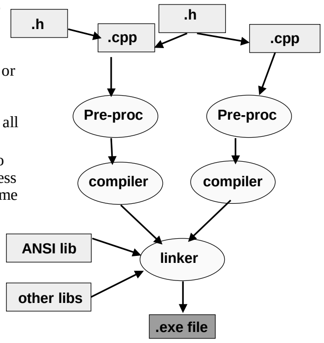
Compilation

- To compile a program:
  - `g++ <options> <source files>`
  - Recommended: `g++ -Wall –ansi -g –o <executable_name> *.cpp`
    - `-Wall` – *Warnings: ALL*
    - `-ansi` – Strict ANSI compliance
    - `-g` – Add debugging symbols to the executable (ie, make it debuggable!)
    - Quizlet: what does `*.cpp` mean?
- What's an "executable"?
  - In Windows, double-clicking on an icon runs a program
    - E.g. double-click on `C:\Windows\notepad.exe`
  - In Unix, you can run your executable from the command line!
    - Type the executable name at the prompt, just like a command
      - In fact, commands are actually executables
    - However, you may need to specify the path to your executables
      - `./<program>` runs `<program>` in the current directory
    - Example:

```
fiji:ehsu% g++ -Wall –ansi -g –o hello hello.cpp
fiji:ehsu% ./hello
```

"Compilation" or "The Big Lie"

- Does this picture look familiar?
- These are the *discrete* steps to program "compilation"
- Hitting the '!' button in MSVC or typing a "`g++ *.cpp`" to *build* (not "compile") your program hides all these separate steps.
- Question: would you want to do this entire process (ie, pre-process and compile *every* file) every time you wanted to generate a new executable?



Selective Recompilation and Makefiles

- Answer:
  - No. You only want to compile those files which were changed (or were affected by a change in another file [quizlet: when might this happen?]). We can reuse the .o/.obj files for files which weren't modified.
- You could do this yourself…
  - `g++ <options> <changed files>`
  - `g++ *.o`
- But you could also use the `make` command and a `Makefile`!
  - Create a `Makefile` to keep track of file dependancies and build options
  - The make command will read the `Makefile` and compile (not build) those files which have *dependancies on modified files!*

Makefile Syntax

- Makefiles consists of *variables* and *rules.*
- Rule Syntax:

```
<target>: <requirements>
        <command>
```

  - The `<requirements>` may be files and/or other targets
  - There ***must*** be a tab (not spaces) before `<command>`
  - The first rule in a `Makefile` is the default `<target>` for `make`
- Variable Syntax:

```
<variable>=<string value>
```

  - All variable values default to the shell variable values
  - Example:
    - BUILD_FLAGS = -Wall -g -ansi

Example Makefile

```
# Example Makefile
CXX=/uns/bin/uns-g++
CXXOPTS=-g -Wall -ansi -DDEBUG

foobar: foo.o bar.o
        $(CXX) $(CXXOPTS) –o foobar foo.o bar.o

foo.o: foo.cc foo.hh
        $(CXX) $(CXXOPTS) –c foo.cc

bar.o: bar.cc bar.hh
        $(CXX) $(CXXOPTS) –c bar.cc

clean:
        rm -f core foobar *.o *~
```

Writing Code

- What causes a bug?
  - What you meant != what you wrote
- Coding right the first time is making "what you meant" align with "what you write"
  - Invariants – `assert()` invariants to discover when your program's state has changed unexpectedly
  - Error handling and notification – Fix or report errors. Your program should never be in a bad state
  - Code review
  - Use a debugger!
    - See next slide …

Debugging

- How do you remove a bug?
  - Read the code. If you don't understand it, the bug will happen again
  - Examine the state of the program at key points in the code
    - Print statements in the code (suggestion: wrap debug output with `#ifdef DEBUG`)
    - Use a debugger to view the state of your program with greater flexibility/control
- Debugger advantages
  - Compile your code only once
  - Monitor all the values in the code
  - Make changes while executing the code
  - Examine core files that are produced when a program crashes
- In other words, debuggers are tools which allow you to examine the state of a program in detail!
  - In fact, debuggers can (and should) be used to understand and improve your code

Debugging Techniques

- Goal: Isolate the problem, then fix it
  - Don't try random things, looking for a solution
    - If you don't understand it, it'll be back
    - This method takes a *long* time
    - You don't learn anything from it
  - Look for the problem, not the solution
    - Figure out two points in code that the problem is between, and close the gap from there.

GDB - The GNU DeBugger

- To run `gdb` (a text-based debugger):
  - `gdb [<program file> [<core file>]]`

`<program file>`	Executable program file
`<core file>`	Crashed program's core dump

  - You must compile with `-g` for debug information!
- Within gdb:
  - Running gdb:

`run [<args>]`	Run program with arguments `<args>`
`quit`	Quit the gdb debugger
`help [<topic>]`	Access gdb's internal help

  - Examining program state:

`info [locals|args]`	Prints out info on [local variables|args]
`backtrace[<n>]`	Prints the top `<n>` frames on the stack
`p[rint] <expr>`	Print out `<expr>`

GDB continued

- Controlling program flow

`s[tep]`	Step one line, entering called functions
`n[ext]`	Step one line, skipping called functions
`finish`	Finish the current function and print the return value

- Controlling program flow with breakpoints

`c[ontinue]`	Continue execution (after a stop)
`b[reak][<where>]`	Set a breakpoint
`d[elete] [<nums>]`	Deletes breakpoints by number
`[r]watch <expr>`	Sets a watchpoint, which will break when `<expr>` is written to [or read]

- Modifying program state

`set <name> <expr>`	Set a variable to `<expr>`
`jump <where>`	Resume program execution at `<where>`

DDD – A Graphical Debugger

- Built-over GDB
- Easier-to-use point and click interface
- To run DDD:
  - `ddd [<program file> [<core file>]]`
- DDD is not standard, but is accessible through uns and through the course-setup.
- Nifty Tutorial available at:

  http://heather.cs.ucdavis.edu/~matloff/Debug/Debug.pdf

Other Tools for CSE 326

- Shell scripts
  - A series of shell commands which are read and executed by the shell (like a DOS batch script).
  - "Shell commands" may be:
    - Executables such as `emacs` and `time`
    - Built-in primitives such as `ls` and for-loops
  - Search the internet for tutorials or sample shell scripts
    - "tcsh builtin commands" worked well at Google …

Other Tools for CSE 326 (part 2)

- Awk
  - A pattern scanning and processing utility. It searches file(s) for specified patterns and perform associated actions.
  - Search the internet for tutorials or samples
    - “awk tutorial” worked well at Google …
- Gnuplot
  - A command-driven function and data plotting program
  - Try emailing the course alias with websites you found; your classmates will thank you!

More Information - In the Dept

- In the department
  - *Your neighbors!*
  - `info` and `man`
  - `uw-cs.lab-help` newsgroup
  - `.login`, `.cshrc`, and `/uns/examples` to see how other people have things set up
  - Course staff - office hours, email
    - Why do you think we get paid the big bucks? =)

More Information - On the Web

- On the web:
  - `http://www.faqs.org` (`comp.unix` questions FAQ)
  - `http://www.google.com`
  - `http://www.refcards.com`
  - ACM Tutorials:
    `http://www.cs.washington.edu/orgs/acm/tutorials/`
  - CSE326 webpage
    `http://www.cs.washington.edu/education/courses/cse326/02wi/computing/class_links.html`
- If you’re curious, check out these topics:
  - Source control (try searching the web for “cvs”)
    - Multiple people working on a file concurrently
    - Easily revert file changes
  - Profiling (try searching the web for “gprof”)
    - Find and eliminate inefficiencies in code