Lecture 1:
Course Overview &
Introduction to Unix
Enrollment

• Who’s in the class?
• Who’s not?
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shell Prompts
  – Files
  – File Editors

• Project 1
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shell Prompts
  – Files
  – File Editors

• Project 1
Hi, I’m Hank...

• Associate Professor of CIS at UO
• Arrived at UO March 2013
• Research interests include visualization, high-performance computing, computer graphics
• Director of CDUX Research Group (8 PhD, 5 MS, 4 BS)
My Background

• Previously:
  – Lawrence Livermore, 1999-2009
  – Lawrence Berkeley, 2009-2016
  – UC Davis 2009-2013

• Education:
  – Ph.D., 2000-2006
VisIt: Application for Visualizing and Analyzing of Very Large Data

- Open source tool
- 1.5M lines of C++ code
  - also Python & Java bindings
- Downloaded >200K times, run on many different UNIX environments

Because of these experiences, I am pretty conservative in what features I make use of.

I also served as the CSWA of multiple teams, leading SW integration efforts.

Simulation of Rayleigh-Taylor Instability, run on LLNL BG/L supercomputer with MIRANDA code, visualized with VisIt
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shell Prompts
  – Files
  – File Editors

• Project 1
CIS 330 Goals

• Goals: **excellence** in C, C++, and Unix

• Why?
  – Many of our 400-level classes require strong knowledge in C, C++, and Unix
  – Critical for success with many development jobs
    • Development jobs are good jobs!!!

• Programming Languages Beacon: http://www.lextrait.com/vincent/implementations.html
How Will We Develop Excellence in C, C++, and Unix?

• Answer: many, many projects
• Very hard to learn this material from lecture
  – Really need to get your hands dirty and “do it”
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shell Prompts
  – Files
  – File Editors

• Project 1
Syllabus is Online

- http://ix.cs.uoregon.edu/~hank/330

CIS 330: C/C++ AND UNIX

Lecture Time: Mon/Weds 10:00-11:20
Lecture Location: 125 McKenzie Hall
Lab Time: Fri 10:00-11:20
Lab Location: 125 McKenzie Hall

Instructor: Hank Childs
Instructor Office Hours: TBD
Office Hours Location: 301 Deschutes Hall

Teaching Assistant: Andrew Hampton
Andrew’s Office Hours: TBD
TA OH Location: TBD

Propose Hank OH as Tues/Thurs 2-3
Andy’s OH TBD

Goal: OH all 5 days

Note: can only do OH this week on Friday (and can meet with folks after class)
Grading For This Course

• Final: 30%

• Projects (made up of multiple pieces)
  – Project 1: Learning Unix (5%)
    • 1A (1%), 1B (2%), 1C (2%)
  – Project 2: C/C++ Primer (15%)
  – Project 3: Large System + Object Oriented Programming (40%)
  – Project 4: Debugging/Profiling (10%)

• Extra Credit through:
  – Community participation

These percentages *might* be adapted as the quarter goes on. I will notify you all via email and via class lecture if this happens.
Prerequisites For This Course

• CIS 314 is the only prerequisite
# Introductory Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Assigned</th>
<th>Due</th>
<th>Task</th>
<th>% Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>4/3</td>
<td>4/5</td>
<td>Editing Files</td>
<td>1%</td>
</tr>
<tr>
<td>1B</td>
<td>4/5</td>
<td>4/10</td>
<td>Scripts/Permissions</td>
<td>2%</td>
</tr>
<tr>
<td>1C</td>
<td>4/10</td>
<td>4/17</td>
<td>Build</td>
<td>2%</td>
</tr>
<tr>
<td>2A</td>
<td>4/5</td>
<td>4/10</td>
<td>Memory</td>
<td>3%</td>
</tr>
<tr>
<td>2B</td>
<td>4/12</td>
<td>4/20</td>
<td>Unions + Function Pointers</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(This is notional ... here to give you an idea)
This is scary...

• From my end:
  – Huge amount of material to cover...

• From your end:
  – Being asked to learn so much...

• I really want to put you in a winning position
  – Let me know if you think I’m not

• Take this seriously
  – You’ve probably never had a class like this before
Expectations

• I ask you put a lot of time into this course, and I believe the payoff will be significant for each of you.

• The grading is designed to make sure you are keeping up with the assignments.
  – Staying on top of the projects will be critical to succeeding in this class.
Expectations

• The projects in this class will be hard work.

• It is difficult to quote exactly how much time, since there is variation in background and programming skill.

  – I expect those who have less developed programming skills will find this class to be a considerable effort, but also that they will have significant improvement by the end of the course.
This Class In a Nutshell...

• Learn the basics of Unix so that we can be functional
  – More Unix introduced as needed throughout course
• Learn more C, and gently ease into C++
• Do a large C++-based project that embraces object-oriented programming
• Learn basics of development: debugging and profiling

  Most of the learning will happen with projects. The lectures are designed to help you do the projects.
Who Is This Class For?

• I am assuming that you have never done Unix or C++ and only got minimal C in 314
  – The class is designed for this profile.

• If you do know a lot of Unix/C/C++, then please try to avoid asking super advanced questions during lecture.
  – Rather, try asking and accept that I may say “too advanced” in response.
Norms for this class

• Please ask questions
• Please ask me to slow down
• Please give feedback
Do You Need To Attend Class?

• The class is not part of your grade

• However, I **strongly** recommend you attend every class
  – It is a really bad situation if you miss class and then want me or the TA’s to “catch you up”
    • we won’t do it
      – unless you had a good reason for missing (in which case, of course you will be accommodated)
  – A lot of the material will come via lecture
20% of S16 class failed (typically 10%)

• How to fail
  – Get behind on projects
  – Stop attending class
    • Every class is basically on how to do the HW

• How to succeed
  – Come to class
  – Get every project completed on time
  – Have a network
My mental model about getting behind in this class

If you get too far behind, then it will not be possible to catch up.
When does this class get hard?

- If you have never done Unix, it might feel hard right away
- If you have done Unix/C, it might feel a bit easy at the beginning, but don’t be fooled.
  - Project 3 is challenging for just about everyone
Norms for interacting with me

<table>
<thead>
<tr>
<th>Time</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 - 9</td>
<td>take kids to class</td>
</tr>
<tr>
<td>7 - 9</td>
<td>take kids to class</td>
</tr>
<tr>
<td>7 - 9</td>
<td>take kids to class</td>
</tr>
<tr>
<td>7 - 9</td>
<td>take kids to class</td>
</tr>
<tr>
<td>9 - 10</td>
<td>Vis organizing committee</td>
</tr>
<tr>
<td>10 - 11</td>
<td>Cyrus/Eric</td>
</tr>
<tr>
<td>10 - 11</td>
<td>Nicole and Sudhanshu</td>
</tr>
<tr>
<td>11 - 12p</td>
<td>Visit developers</td>
</tr>
<tr>
<td>11 - 12p</td>
<td>ryan and patrick @ 1115</td>
</tr>
<tr>
<td>12p - 1p</td>
<td>Tori &amp; Tom</td>
</tr>
<tr>
<td>12p - 1p</td>
<td>Scott</td>
</tr>
<tr>
<td>1p - 4p</td>
<td>faculty meeting</td>
</tr>
<tr>
<td>4p - 6p</td>
<td>Charlotte at ortho</td>
</tr>
<tr>
<td>6p - 8p</td>
<td>Vincent gravit</td>
</tr>
<tr>
<td>8p - 10</td>
<td>Wayne</td>
</tr>
<tr>
<td>10 - 11</td>
<td>Earl and Brad</td>
</tr>
<tr>
<td>10 - 11</td>
<td>Xvis</td>
</tr>
<tr>
<td>11 - 12p</td>
<td>Kirsten &amp; Elliott</td>
</tr>
<tr>
<td>11 - 12p</td>
<td>Kirsten and Sudhanshu</td>
</tr>
<tr>
<td>11:30 - 12:30p</td>
<td>Wayne Ford</td>
</tr>
<tr>
<td>12:30p - 1p</td>
<td>James</td>
</tr>
<tr>
<td>1p - 4p</td>
<td>Pugmire</td>
</tr>
<tr>
<td>2p - 7p</td>
<td>pick up kids</td>
</tr>
<tr>
<td>3p - 5p</td>
<td>Colloquium</td>
</tr>
<tr>
<td>4p - 5p</td>
<td>Hank does pick up this week</td>
</tr>
<tr>
<td>6:30p - Wayne Ford</td>
<td></td>
</tr>
<tr>
<td>7p - 8p</td>
<td>CISL</td>
</tr>
<tr>
<td>7p - 8p</td>
<td>1d texture</td>
</tr>
</tbody>
</table>

- You are welcome to call me Hank
- You can email me with a bug
- More than once becomes problematic
- Getting in touch with me:
  - OH
  - I'm pretty good about email
  - If I don't respond within 24 hours, resend
  - It is almost never a good idea to drop by my office without an appointment
- If you want to meet personally, set up an appointment
- [hkp://cdux.cs.uoregon.edu/childs_calendar.html](hkp://cdux.cs.uoregon.edu/childs_calendar.html)
Course Materials

• Textbook: Thinking in C++ (free online book)

• PowerPoint lectures will be posted online.

• Other materials (e.g., webpages) will be occasionally used

• I will “live code” sometimes.
Working Together

• All projects are *individual* projects.
• Copying code from other students is cheating.
• However: I highly encourage you to discuss your roadblocks with each other and lean on each other to figure out solutions to your problems.
Working Together, part 2

• I have set up a forum on Piazza.
  – I will monitor (and respond) to the forum and encourage you all to do the same.
  – I may award extra credit to students who are particularly helpful on Piazza.
  – The amount of credit will vary based on involvement, with a maximum of 5%.
Late Passes

• You have 2 "late passes."
• Late passes allow you to turn in your project two days after the due date for full credit.
• If you run out of late passes, then you may continue to earn half credit on any project.
• Every unused late pass is worth 0% extra credit.
• Don’t need to tell me if you want to use it
  – Late pass assignment will be a question on the final
  – All late projects will be scored at 50% and then credit from late passes will be awarded at the end of the term
Class Summary

• This class will teach you about Unix, C, and C++
• This class will improve your programming skills
• This class will likely help you succeed at a job
• This class will require a lot of work
• The projects are the heart of the class
  – The lectures are designed to help you do the projects
IDEs

• IDEs are great
  – … but in this class, we will learn how to get by without them

• Many, many Unix-based projects don’t use IDEs
  – The skills you are using will be useful going forward in your careers
Accessing a Unix environment

- Rm 100, Deschutes
- Remote logins (ssh, scp)
- Windows options
  - Cygwin / MSYS
  - Virtual machines

Who has home access to a Unix environment?

Who has Windows only and is planning to pursue Cygwin/VM?
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shells
  – Files
  – File Editors

• Project 1
What is Unix?

• Operating system
  – Multi-tasking
  – Multi-user
• Started at AT&T Bell Labs in late ’60s, early ‘70s
• First release in 1973
What is Unix?

• 80s & 90s: many competing versions, all conforming to same standard
  – AIX (IBM), Solaris (Sun), HP-UX (Hewlett-Packard)

• 1990s: Linux takes off
  – Open source

• 2000s: commercial Unixes abandoned, companies use Linux, back Linux
  – Several variants of Linux

• OS X: used on Macs since 2002
  – Meets Unix standard
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shells
  – Files
  – File Editors

• Project 1
Shells

- Shells are accessed through a terminal program
  - Typically exposed on all Linux
  - Mac: Applications->Utilities->Terminal
    - (I always post this on the dock immediately upon getting a new Mac)
Shells

• Shells are interpreters
  – Like Python

• You type a command, it carries out the command
Shells

• There are many types of shells
• Two most popular:
  – sh (= bash & ksh)
  – csh (= tcsh)
• They differ in syntax, particularly for
  – Environment variables
  – Iteration / loops / conditionals

The examples in this course will use syntax for sh
Environment Variables

• Environment variables: variables stored by shell interpreter
• Some environment variables create side effects in the shell
• Other environment variables can be just for your own private purposes
Environment Variables

New commands: export, echo, env
Shells

• There is lots more to shells ... we will learn about them as we go through the quarter
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shells
  – Files
  – File Editors

• Project 1
Files

• Unix maintains a file system
  – File system controls how data is stored and retrieved

• Primary abstractions:
  – Directories
  – Files

• Files are contained within directories
Directories are hierarchical

• Directories can be placed within other directories

• “/” -- The root directory
  – Note “/”, where Windows uses “\”

• “/dir1/dir2/file1”
  – What does this mean?

File file1 is contained in directory dir2,
which is contained in directory dir1,
which is in the root directory
Home directory

• Unix supports multiple users
• Each user has their own directory that they control
• Location varies over Unix implementation, but typically something like “/home/username”
• Stored in environment variables

```
fawcett:~ childs$ echo $HOME
/Users/childs
```
Anatomy of shell formatting

- “~” (tilde) is shorthand for your home directory
  - You can use it when invoking commands

The shell formatting varies over Unix implementation and can be customized with environment variables. (PS1, PS2, etc)
File manipulation

New commands: mkdir, cd, touch, ls, rmdir, rm
cd: change directory

• The shell always has a “present working directory”
  – directory that commands are relative to
• “cd” changes the present working directory
• When you start a shell, the shell is in your “home” directory
Unix commands: mkdir

• mkdir: makes a directory
  – Two flavors
    • Relative to current directory
      – mkdir dirNew
    • Relative to absolute path
      – mkdir /dir1/dir2/dirNew
        » (dir1 and dir2 already exist)
Unix commands: rmdir

• rmdir: removes a directory
  – Two flavors
    • Relative to current directory
      – rmdir badDir
    • Relative to absolute path
      – rmdir /dir1/dir2/badDir
        » Removes badDir, leaves dir1, dir2 in place

• Only works on empty directories!
  – “Empty” directories are directories with no files

Most Unix commands can distinguish between absolute and relative path, via the “/” at beginning of filename.
(I’m not going to point this feature out for subsequent commands.)
Unix commands: touch

• touch: “touch” a file

• Behavior:
  – If the file doesn’t exist
    • → create it
  – If the file does exist
    • → update time stamp

Time stamps record the last modification to a file or directory

Why could time stamps be useful?
Unix commands: **ls**

- **ls**: list the contents of a directory
  - Note this is “LS”, not “is” with a capital ‘i’
- Many flags, which we will discuss later
  - A flag is a mechanism for modifying a Unix programs behavior.
  - Convention of using hyphens to signify special status
- “ls” is also useful with “wild cards”, which we will also discuss later
Important: “man”

• Get a man page:
• → “man rmdir” gives:

RMDIR(1) BSD General Commands Manual RMDIR(1)

NAME
rmdir -- remove directories

SYNOPSIS
rmdir [-p] directory ...

DESCRIPTION
The rmdir utility removes the directory entry specified by each directory argument, provided it is empty.

Arguments are processed in the order given. In order to remove both a parent directory and a subdirectory of that parent, the subdirectory must be specified first so the parent directory is empty when rmdir tries to remove it.

The following option is available:

-p Each directory argument is treated as a pathname of which all components will be removed, if they are empty, starting with the last most component. (See rm(1) for fully non-discriminant
Outline

• Class Overview
  – My Background
  – Goals
  – Syllabus

• Getting Started With Unix
  – Unix History
  – Shells
  – Files
  – File Editors

• Project 1
File Editors

• Existing file editors:
  – Vi
  – Emacs
  – Two or three hot new editors that everyone loves (and ultimately fade away and die)

• This has been the state of things for 25 years

I will teach “vi” in this course. You are welcome to use whatever editor you want.
Vi has two modes

• Command mode
  – When you type keystrokes, they are telling vi a command you want to perform, and the keystrokes don’t appear in the file

• Edit mode
  – When you type keystrokes, they appear in the file.
Transitioning between modes

• Command mode to edit mode
  – i: enter into edit mode at the current cursor position
  – a: enter into edit mode at the cursor position immediately to the right of the current position
  – I: enter into edit mode at the beginning of the current line
  – A: enter into edit mode at the end of the current line

There are other ways to enter edit mode as well
Transitioning between modes

• Edit mode to command mode
  – Press Escape
Useful commands

- `yy`: yank the current line and put it in a buffer
  - `2yy`: yank the current line and the line below it
- `p`: paste the contents of the buffer
- `x`: delete the character at the current cursor
- `":100"` go to line 100 in the file
- Arrows can be used to navigate the cursor position (while in command mode)
  - So do `h`, `j`, `k`, and `l`

We will discuss more tips for “vi” throughout the quarter. They will mostly be student-driven (Q&A time each class)
My first vi sequence

• At a shell, type: “vi cis330file”
• Press ‘i’ (to enter edit mode)
• Type “I am using vi and it is fun” (text appears on the screen)
• Press “Escape” (to enter command mode)
• Press “:wq” (command mode sequence for “write and quit”)
vi / vim graphical cheat sheet

Esc
normal mode

~ toggle case
.\ mark
# prev ident
% goto match
^ "soft" bol
&\ repeat :s
* next ident (begin sentence)
) end sentence
0 "hard" bol
-p prev line
+= next line
Q ex mode
W next WORD
E end WORD
R replace mode
T back 'till
Y yank line
U undo
I insert at bol
O open above
P paste before
f{ begin parag.
f} end parag.
A append at eol
S subst line
D delete to eol
f"back" find ch
G eof/ goto lin
H screen top
J join lines
K help
L screen bottom
F visual line
N prev (find)
M screen mid\1
< unindent
> indent
? find (rev.)
\ find

Q* record macro
q record macro
w next word
r replace char
t tuck
y yank
u undo
i insert mode
O open below
P paste after
f{ begin parag.
f} end parag.
A append
S subst char
D delete char
f change char
f extra cmds
h back/ f find char
G extra cmds
H screen top
J join lines
K help
L screen bottom
E visual line
N prev (find)
M screen mid\1
< unindent
> indent
? find (rev.)
\ find

Z: quit
Z* extra cmds
X x-space
C change to eol
V visual lines
B prev WORD
N prev (find)
M screen mid\1
< unindent
> indent
? find (rev.)
\ find

motion
moves the cursor, or defines the range for an operator

command
direct action command, if red, it enters insert mode

operator
requires a motion afterwards, operates between cursor & destination

extra
special functions, requires extra input

motion
moves the cursor, or defines the range for an operator

command
direct action command, if red, it enters insert mode

operator
requires a motion afterwards, operates between cursor & destination

The main command line commands ("ex"):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:
CTRL-R: redo (vim),
CTRL-E/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:
Move around and type operator to act on selected region (vim only)

Notes:
(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*)
    (e.g.: "ay$" to copy rest of line to reg 'a')
(2) type in a number before any action to repeat it that number of times
    (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line
    (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top,
    zb: bottom, zz: center
(6) gg: top of file (vim only),
    gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to  www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

http://www.viemu.com/vi-vim-cheat-sheet.gif
Project 1A

• Practice using an editor
• Must be written using editor on Unix platform
  – I realize this is unenforceable.
  – If you want to do it with another mechanism, I can’t stop you
    • But realize this project is simply to prepare you for later projects
• Due Saturday (→ 6am Sunday)
Project 1A

• Write $\geq300$ words using editor (vi, emacs, other)
• Topic: what you know about C programming language
• Can’t write 300 words?
  – Bonus topic: what you want from this course
• How will you know if it is 300 words?
  – Unix command: “wc” (word count)
Unix command: wc (word count)

```
fawcett:~ child$ vi hanks_essay
fawcett:~ child$ wc -w hanks_essay
  252 hanks_essay
fawcett:~ child$ wc hanks_essay
   63    252    1071 hanks_essay
fawcett:~ child$
```

(63 = lines, 252 = words, 1071 = character)
Project 1A

CIS 330: Project #1A
Assigned: April 3rd, 2017
Due April 8, 2017
(which means submitted by 6am on April 9th, 2017)
Worth 1% of your grade

Assignment:
1) On a Unix platform (including Mac), use an editor (vi, emacs, other) to write a 300 word "essay"
   a. The purpose of the essay is to practice using an editor.
      i. Grammar will not be graded
   b. I would like to learn more about what you know about C and want from this class ... I recommend you each write about that.
   c. If you run out of things to say, you don’t have to write original words (do a copy/paste using vi commands: yyp)

Do not write this in another editor and copy into vi.

Also, do not put more than 100 characters onto any given line. (I want you to practice having multiple lines and navigating.) If you have more than 100 characters per line, you will receive half credit.
Accessing remote machines

• Windows->Unix
  – ??? (Hummingbird Exceed was the answer last time I used Windows)

• Unix->Unix
  – ssh: secure shell
    ssh –l hank ix.cs.uoregon.edu
  – scp: secure copy
    scp hank@ix.cs.uoregon.edu:~/file1
  • Also, ftp: file transfer protocol
Note on Homeworks

• Project 1A due on Saturday
• Project 2A will be assigned on Wednesday
  – Due in class on Monday
• Project 1B will be assigned on Friday
  – Due on Weds, April 12
• I will lecture on Friday (instead of lab)
How to submit

• Canvas

• If you run into trouble:
  – Email me your solution
Don’t forget

• This lecture is available online
  – http://ix.cs.uoregon.edu/~hank/330

• All project prompts are available online