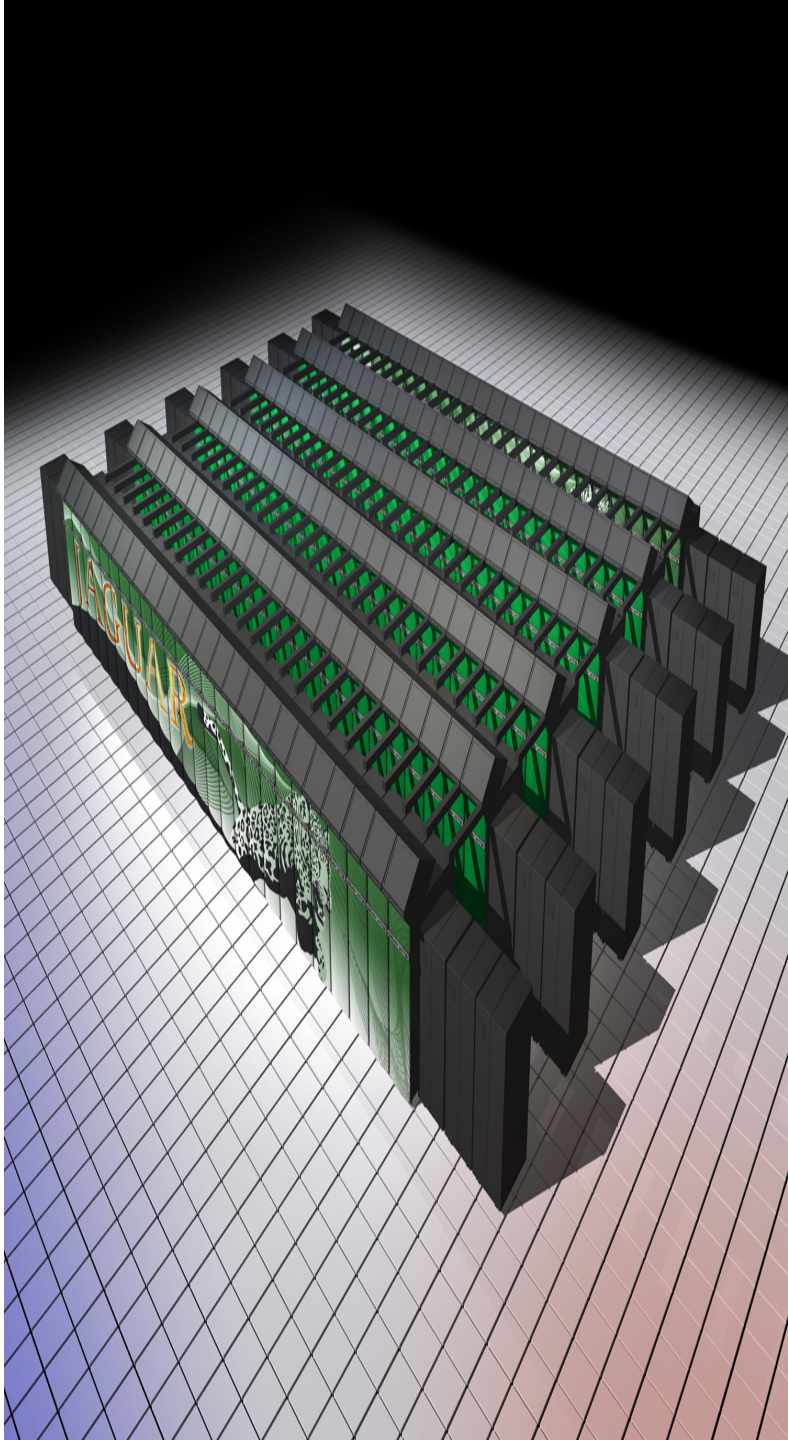
Introduction to Unix

Mitch Griffith

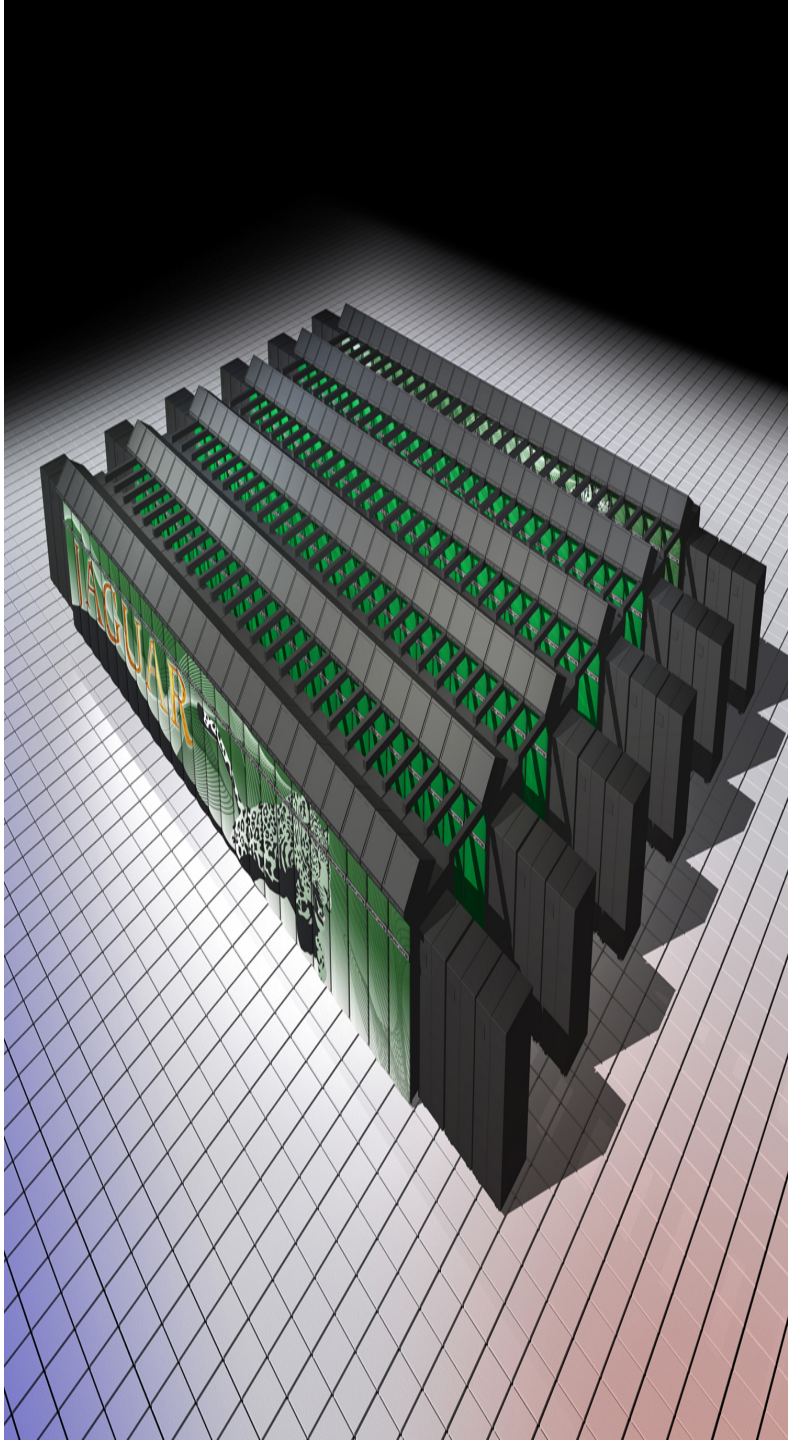
Welcome!

- Today's Agenda:
  - The basics
  - What is Unix?
  - Unix commands you can't live without
  - The vi editor
  - Compiling and make
  - C Programming Language



The basics

- Hardware
- Software
- Application software
- System software



Hardware

- Everything that can be touched in a computer
- Microprocessor
- Primary memory
- Secondary memory
- Network cables
- Printers
- Keyboards, mice, etc

Software

- Programming used to be difficult
  - Rewire the whole machine each time
- Makes the hardware usable
- System software (operating systems)
  - Controls hardware
  - Applications do not need to know how to use hardware
  - Kernel vs. utilities
  - GUI vs. CLI
- Application software
  - Word processing
  - Spreadsheets
  - Games

Unix

- Operating system
- Developed in early ‘70s by AT&T at Bell Labs
- Multi user system
- Unix has come to mean any Unix-like operating system
- Andrew Tanenbaum created Minix
  - Textbook demonstration
- Linus Torvalds created Unix-like kernel
  - Linux was born
  - Technically linux is the kernel only

Unix characteristics

- Multi-user operation
  - Accounts for users
  - Permissions based
- Command line interface (CLI)
  - No GUI (sort of)
    - X11 is a standard for doing GUI on unix systems
- Utility programs
  - Navigate system
  - Execute programs
- Device management through files

Unix characteristics

- Data security
  - Permissions based
    - Read / write / execute
  - File system based
- Data processing through filters
  - Text manipulating programs used heavily

Unix – accounts

- All users have a distinct account name
  - i.e. bob, mary, userx11, superdude, etc
- All accounts have passwords
  - Don't use common passwords (name, birthday, 'password', etc)
  - Authentication vs. authorization
- All accounts have a default home directory
  - More on file systems and directories in a bit
- All accounts have configuration files
  - For individual preferences
- All accounts have a command interpreter
  - Program that accepts and executes commands

Unix - permissions

- All data is stored in files
  - Files are collections of data lumped together
    - Addresses, recipes, raw data, etc
- All files have permissions
  - Read / write / execute
- Permissions based on:
  - Who you are & what group you are in
- Permissions are divided into three categories

Unix - permissions

- Example:

-rw-r--r-- 1 csep100 ccsstaff 303 Aug 28 14:21 staff 222 Jun 4 mpi.pbs

- Close up:

-rw-r--r--

Unix commands you can't live without

- man – read system manual pages
- pwd – identify the working directory
- cd – change the working directory
- echo – display a string
- ls – display contents of directory
- cat – display text of file
- more – display text of file
- cp – copy a file
- mv – move a file
- rm – remove a file
- mkdir – create a directory
- rmdir – remove a directory
- exit – end session

Unix commands you can live without, but who'd want to

- ssh – initiate remote connection
- scp – copy a remote file
- tar – archive files
- find – find files or directories
- less – less is more (with benefits)
- vi – edit files
- env – environment variables
- hostname – list name of current machine
- ln – create links
- history – show command history
- ps – show running processes
- kill – kill a running process
- chmod – change permissions

Unix file system

- Hierarchical file structure
  - Directories
  - Files
- Directories
  - Contain files and/or other directories (subdirectories)
- Files
  - Contain data (text, binary, etc)

Hierarchical (tree) file system

- There is one parent directory per file system
  - Root aka '/'
  - Everything else is contained in this directory
    - Subdirectories
    - Files
- Pathnames – name (address) of the file/directory
  - /ccs/home/csep100
  - Absolute vs. relative pathnames
- Navigating directories
- Creating files, directories
  - Redirection, pipes

Shell scripts

- Programs to help automate recurring task
- Text files that are interpreted by shell program
  - Interpreted vs compiled languages
- Example

Shell script basics

- Comments
- Values
- Variables
- Arrays
- Selection
- Loops

Shell script - comments

- Comments are not executed
- Useful in documenting you scripts
- # = comment everything to right
- Exception is very first line of script
  - #!/bin/bash

Shell script - values

- Values are object used in your script
  - String values – ‘bobby’
  - Numerical values – 94
- String values are quoted
  - ‘bobby’ – just a string, no processing
  - “bobby” – a string, but with processing
  - `bobby` - execute this string and get output back
- Numerical values can have math performed on them using the special form $(( ))
  - x=$((2+2))
  - echo $x
  - 4

Shell script - variables

- Use variables when the values may change
- Example
  - x=hello
  - echo $x
  - hello
  - x=5
  - echo $x
  - 5
- If you want to treat a string as a variable
  - let x=5*5

Shell scripts - arrays

- Arrays are variable with 1 or more elements
  - x=(a b c d)
  - echo ${x[0]}
  - a
  - echo $x
  - a
  - Where's everything else?
  - echo ${x[@]}
  - a b c d

Shell scripts - selection

- If this, then that, else something else
  - if … then … elif … fi

```
#!/bin/bash
if [ $1 = 'hello' ];
  then
        echo hello yourself
  elif [ $1 = 'howdy' ];
    then
        echo howdy to you too
  else
        echo nobody home
fi
```

Shell script - loops

- For this number of times, do something

```
for name in bobby doug frank; do
        echo $name
done

let x=1
until [ $x -eq 10 ]; do
        echo $x
        let x=$x+1
done

let x=1
while [ $x -ne 10 ]; do
        echo $x
        let x=$x+1
done
```

Exercise – Part 1

- Create a bash script that performs the following actions:
  - Create a text file called ex1.txt
  - Overwrite the ex1.txt and add to it your: name, email
  - Create a directory called exercise1
  - Create four subdirectories within the exercise1 directory: sub0/, sub1/, sub2/, sub3/
  - Create 1 subdirectory is sub0, 2 in sub1, 3 in sub2, 4 in sub3

Exercise – Part 2

- Modify your shell script to do the following:
  - Copy ex1.txt using relative path names into the following subdirectories:
    - exercise1/sub0/sub0
    - exercise1/sub2/sub1
    - exercise1/sub3/sub0
    - exercise1/sub3/sub1
  - Rename the files just copied to <your name>.txt
  - For each file just renamed, copy it back up one directory:
    - Example: exercise1/sub3/sub1/bobby.txt -> homework1/sub3/bobby.txt

Questions?

http://www.olcf.ornl.gov

HPC Fundamentals
Programming Languages

Mitch Griffith

Programming Languages

- Computers execute instructions, i.e.:
  - Move data to a memory location
  - Add data in one memory location to another
  - Jump to next instruction to execute
- Programming languages allow the order of instructions to be controlled
  - Each type of processor has different instructions
- Low-level vs. high-level languages
  - Machine languages
  - Assembly languages
  - High-level languages
- Interpreted vs. compiled

Machine languages

- Its what computers understand
  - Low level language
  - Specific to a particular architecture
    - x86, IA-32, x86-64, AMD64, Motorola's 6800 and 68000
- Binary instructions
  - General form

6 5 5 5 5 6 bits

[op | rs | rt | rd |shamt| funct] R-type

[op | rs | rt | address/immediate] I-type

[op | target address] J-type

- Example

000010 00000 00000 00000 10000 000000

Assembly language

- One-to-one mapping to machine instruction
- Specific to each architecture
  - Usually provided by manufacturer or processor
- Low-level language
- Uses mnemonics representation of instructions
  - mv ax, es
  - add ip, ax
- Example:

Machine code= 000010 00000 00000 00000 10000 000000

Assembly = jmp 1024

High-level languages

- Abstract representation of program
  - Details of architecture are hidden
  - More portability
  - Easier to program
- Structured for humans not machines
- Examples
  - C++
  - Fortran
  - Java
  - Python
  - Perl
  - Hundreds exist

Interpreted vs. compiled

- Interpreted languages
  - Require special programs that execute other programs
  - Typically 1 line at a time
  - Shell script, Python, Perl
- Compiled languages
  - Do not require a separate program to execute
  - Does require a program called a compiler
  - Typically faster than interpreted languages (not always)
  - C/C++, Fortran

Interpreted languages

- Start with a text file (call source file)
  - 1st line is typically #!<path to interpreter program>
    - #!/bin/bash or #!/bin/perl, etc.
- Interpreter reads and executes one line at a time
  - Syntax errors are caught immediately
  - Does not look forward to see what's next
- Programmers can usually test singe statements in interpreter
  - echo "hello"

Compiled languages

- Starts with text file (source code)
- A compiler program:
  - Coverts source into assembly language
  - Assembly is converted into machine code (object code)
  - Object code is linked with other object code to make executable by linker program (often part of compiler
- Final code does not need an interpreter to execute
  - Can execute on own
    - ./programName
- Typically faster than interpreted languages
  - Doesn't require overhead of running another program

C basics

- Comments
- Variables
- Constants
- Selection
- Loops
- Functions

C program example – Hello World

```
#include <stdio.h>


void main(void)
{
    printf(“Hello World\n”);
}
```

Hello World +

```
#include <stdio.h>
void main(void)
{
    int x;
    x=10;
    printf(“Hello World\n”);
    printf(“x=%d”,x);
}
```

Hello World ++

```
#include <stdio.h>


int main(int argc, char** argv)
{
    printf("Hello World, %s\n",
 argv[1]);
    return (0);
}
```

Variables

- Typical variables and their size

Type	Size	Range
unsigned char	8 bits	0 to 255
char	8 bits	-128 to 127
unsigned int	16 bits	0 to 65,535
short int	16 bits	-32,768 to 32,767
int	16 bits	-32,768 to 32,767
unsigned long	32 bits	0 to 4,294,967,295
long	32 bits	-2,147,483,648 to 2,147,483,647
float	32 bits	1.17549435 * (10^-38) to 3.40282347 * (10^+38)
double	64 bits	2.2250738585072014 * (10^-308) to 1.7976931348623157 * (10^+308
long double	80 bits	3.4 * (10^-4932) to 1.1 * (10^4932)

Variable declaration

- int I;
- char c;
- double dbl;
- float f;
- int i=0;
- const pi=3.14159265;

Arrays

- One dimensional
  - int i [10];
  - char str [25];
- Two dimensional
  - int d [2][2];

Selection

- if condition then something else something else.

```
if (x < 10)
{
  printf("low\n");
}
```

Loops

- While condition do

```
while (x < 10)
{
  printf("low\n");
  x++;
}
```

- Do until condition

```
do
{
  printf("low\n");
  x++;
} while (x<10);
```

Loops

- For expression do

```
for (i=0; i<10; i++)
{
  printf("low\n");
}
```

Functions

- Recurring pieces of code

```
<return> func (parameters)
{
 body
}
```

Questions?

http://www.olcf.ornl.gov