
# Using Glenn—the IBM Opteron 1350 Introduction to UNIX

October 19, 2010

# Functions - Scope of Activity

**Supercomputing.** Computation, software, storage, and support services empower Ohio's scientists, engineers, faculty, students, businesses and other clients.

**Networking.** Ohio's universities, colleges, K-12 and state government connect to the network. OSC also provides engineering services, video conferencing, and support through a 24x7 service desk.

**Research.** Lead science and engineering projects, assist researchers with custom needs, partner with regional, national, and international researchers in groundbreaking initiatives, and develop new tools.

**Education.** The Ralph Regula School of Computational Science delivers computational science training to students and companies across Ohio.

# Contents

- Introduction
- Useful Concepts
- Commands
- Files and Directories, Part I
- Screen Editors
- Files and Directories, Part II
- UNIX Shell
- Filters and Information Processing
- Selected Sources for Further Enlightenment

# Introduction

- Acknowledgments
- Purpose of Course
- What is UNIX?
- Software Uses
- Structure of UNIX

# Introduction—Acknowledgments

Jim Giuliani—Lead, Science and Technology Support Group

(see end of notes for selected references)

# Introduction—Purpose of Course

- Introduce the basics of UNIX
- Introduce the basics of UNIX screen editors
- Provide hands-on practice

# Introduction—What Is UNIX?

- Operating system **and** attendant application programs
- Available on virtually all machines in one form or another
- Long history
- Adapted to new platforms
- Based on C programming language

# Introduction—Software Uses

- Operating
  - For the computer
  - Liaison between computer and user
- Applications
  - Basic functions
    - Electronic filing
    - Word processing
    - Database maintenance (not in this workshop)
    - Electronic mail and networking access (not in this workshop)
  - Additional function
    - Programming

# Introduction—Structure of UNIX

- Kernel—not for average users
  - CPU scheduling
  - Memory management
  - Process management
  - Other duties
- Shell
  - Interacts between kernel and user
  - User invokes commands through shell
  - Choices
    - Bourne
    - Korn
    - Cshell
    - TCshell
    - BASH

# Useful Concepts

- Getting Out of Trouble
- Terms and Concepts

# Useful Concepts—Getting Out of Trouble

- **^** often means press the **Control** key simultaneously with another key
- **`kill`**: **kill** a job in the works
- **`^U`**: **delete** a full command line to the prompt
- **`^u`**: undo last command
- **`^s`**: stop scrolling
- **`^q`**: resume scrolling

What happens if you type a word that is not a valid UNIX command?

# Useful Concepts—Terms and Concepts

- Standard input stdin
- Standard output stdout
- Case sensitive
  - Note whether you use upper case or lower case
  - UNIX commands—usually lower case
- **^** control key
  - If **^** is part of a command, press the control key and the second key simultaneously
- **<Return>** key
  - Almost always used to tell system end of command line

# Useful Concepts—Terms and Concepts

- Prompt
  - A symbol (usually **%** or **$**)
  - User can change prompt in dot-file
  - When cursor is at prompt, user can enter a command
- Permission
  - Ability to read, write, or execute a directory or a file by user, group member, other
  - Default permissions when directory/file created
  - Can change permissions manually

# Commands

- Structure
- Special Features
- First (and Last) Commands
- Exercise 1
- Easy Commands
- Exercise 2

# Commands—Structure

**`command -option argument`**

- **`command`**
  - Usually lower case
  - What you want to do
- ***`-option`***
  - Sometimes not required
  - Enhances output of command; tailors output
  - Often can be combined with one or more other options
- ***`argument`***
  - What command will act upon
  - Often have more than one argument
  - Sometimes not required

# Commands—Special Features

- Can combine several commands on one line—separate with semicolons
- Can create complex commands with redirection signs (**|**, **>**, **>>**, **<**, **<<**)
- Can combine frequently used sequence of commands in a file and run that file like a command
- Can find information about every available command by typing at prompt (finish with **<Return>**)
  - **man *command***
  - Press **q** to leave on-line manual

# Commands—First (and Last) Commands

- Logging on
  - **`userid`**
    - Assigned by systems administrator
    - Probably won't change
    - At OSC, different userid for each **project**
  - Password
    - Assigned by systems administrator
    - User should change it often for security
    - At OSC, password should be between **6** and **8** characters long
  - To change password
    - Go to URL in account letter you receive
    - Keep account letter—if you ever need your password reset, it will be reset to the default password in the account letter

# Commands—First (and Last) Commands

- Connecting to an OSC machine remotely
  - Use Secure Shell protocol: at prompt, enter

    **`ssh userid@machine.osc.edu`**
  - Enter password
- Logging off
  - At prompt, enter

    **`exit`**
  - May differ from system to system, but usually works

# Commands—Exercise 1

- Use your OSC userid/password or a workshop userid/password to log on to the Bale Cluster node at which you are sitting
- Bring up an **`ssh`** client window (usually under Systems Tools)
- Log on to Glenn
  - Machine address is: **`glenn.osc.edu`**
- Quick reminder—are you ending every command with a **<Return>**? Good job!

# Commands—Easy Commands

- **`date`**
- **`cal`** *`year`* or *`month year`*
- **`finger -m, -l, -s`**
- **`man`** **`command`**
- **`man -k`** *`keyword`*
- **`who`**
- **`who am i`**

# Commands—Exercise 2

- At the prompt type: **`man man`**
- What do you type to exit an online manual page? (Pop quiz from a passing reference earlier; do you remember?)
- Type the easy commands on the previous page, one at a time (with a **<Return>** after each command), and look at the output.
- Type **`man`** ***`command`*** for some of the commands on the previous slide to accustom yourself to using the online manual and to understand how the various options change the output.

# Commands—Exercise 2

- Type some of the commands with the options from the online manual pages. Can you combine some of the options under one hyphen? (The options can't be combined if they contradict each other.)
- To answer a question you may have: you can use the command **`echo`** and one of the commands, such as **`date`**, in your shell programs so that you have an automatic dating system for the resulting output. This procedure is convenient for multiple runs of a shell script. Check the command **`echo`** in the online manual pages.

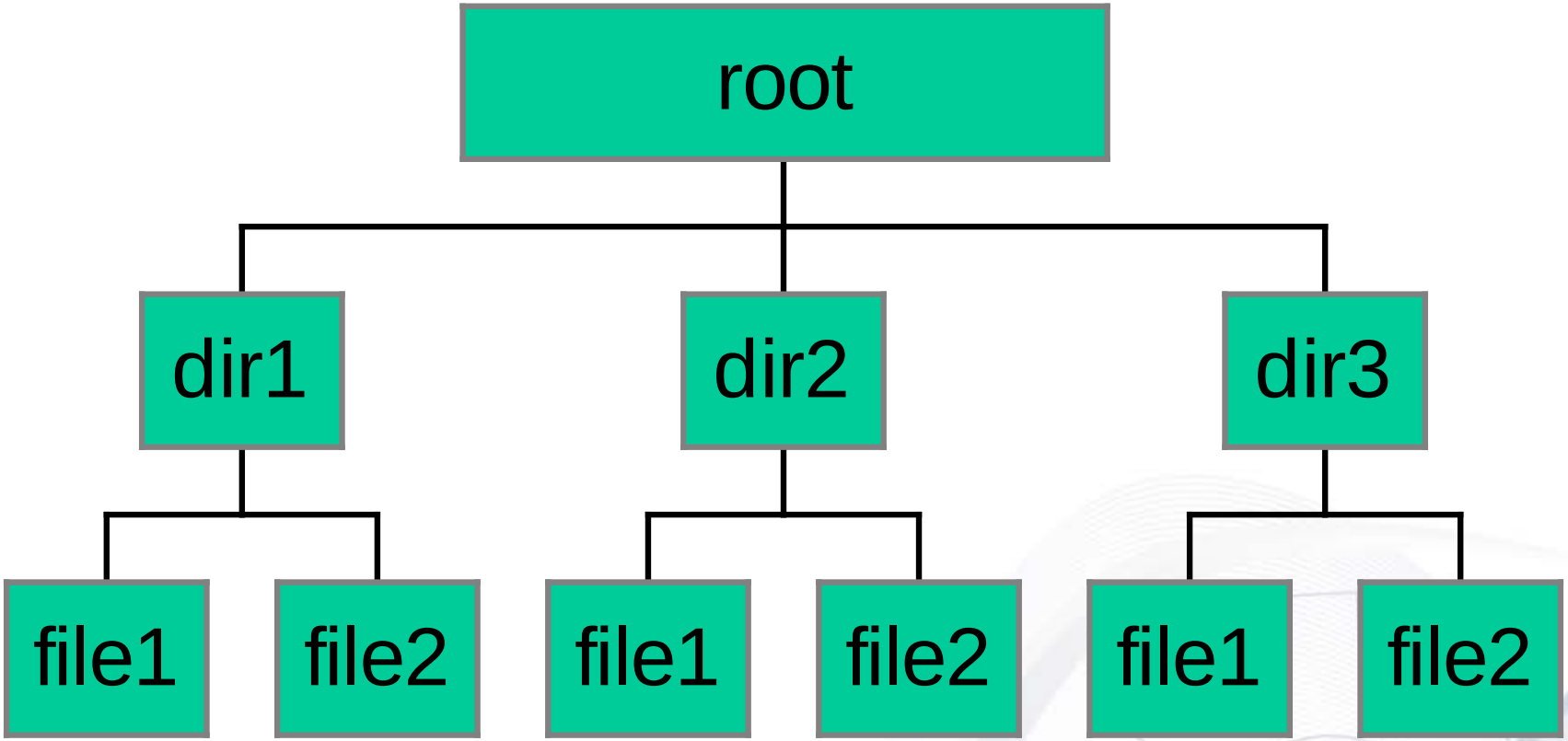
# Commands—Exercise 2

- At the prompt, type **`date`**
  - Study the output
- At the prompt, type **`cal` *`year_you_were_born`***
- At the prompt, type **`who`**
- At the prompt, choose a userid from the output of **`who`**, and type **`finger` *`userid`***
- At the prompt, type **`man who`**
  - Study the options
- At the prompt, type **`man finger`**
  - Study the options

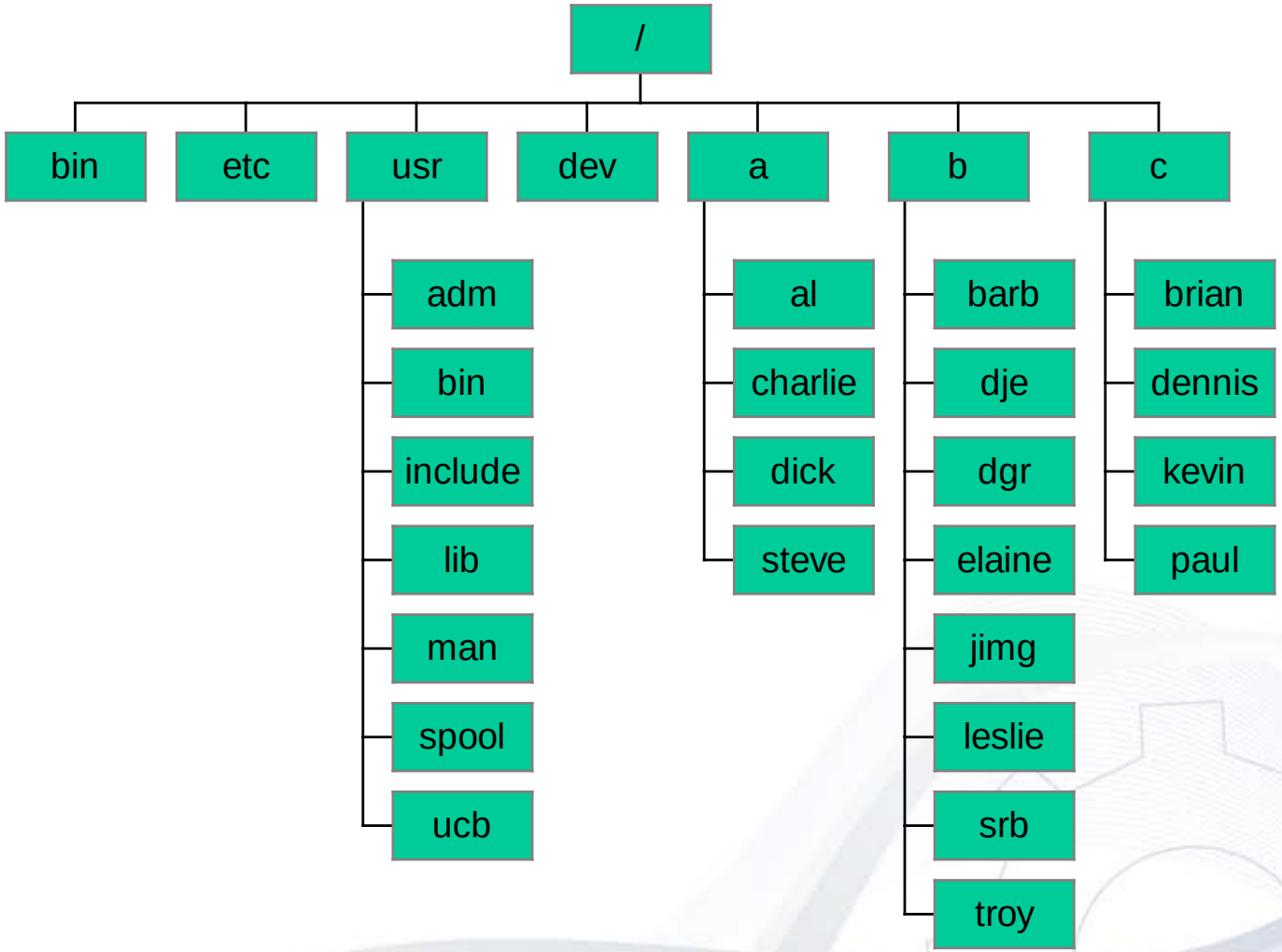
# Files and Directories, Part I

- Illustrations
- Where Am I?
- First Commands
- Name Hints
- Creating Files

# Files and Directories, Part I—Illustrations



# Files and Directories, Part I—Illustrations



# Files and Directories, Part I—Where Am I?

- **pwd**
  - Tells you where you are in the file hierarchy
  - Important in increasingly complex hierarchy
  - Important for moving files
  - Important in moving to and from directories of others
- At the prompt, type **pwd**
  - Study the output

# Files and Directories, Part I—First Commands

- **`ls -a -c -l -f -r -s -t`** ***`directory`***
  - Don't forget to check **`man`** page
- **`cat`** ***`[several options] file`***
  - The command **`cat`** has several uses
- **`more`** ***`[several] file`***
- **`less`** ***`[several] file`***
- **`lpr -l`** ***`[other options] file`***
  - Printing can be indicated in a couple of ways; system-dependent

# Files and Directories, Part I—Name Hints

- Avoid spaces; separate words with dots or underscores
  - ***my.file***
- Begin directory names with upper-case letters
  - ***My.directory***
- Avoid special characters
  - **/ \ ‘ “ ; - ? [ ] ( ) ~ ! $ { } < >**
- Make names descriptive

# Files and Directories, Part I—Creating Files

- **`cat`** command (remember—several uses)
  - At prompt type **`cat > file.name`**
  - Note, after **<Return>**, no prompt
  - Type two or three lines of text, each followed by **<Return>**
  - How do I get prompt back? Type **`^d`**
- At the prompt, type **`ls`**
  - Do you see your new file?
- How do we see contents?
  - At the prompt, type **`cat file.name`**
  - At the prompt, type **`more file.name`**
  - At the prompt, type **`less file.name`**

# Files and Directories, Part I—Creating Files

- Use redirection symbols to redirect output from command to file—can create file this way
  - Example of redirection **>**
    - Usage ***command* > *file***
    - Verify creation **ls**
    - Read contents **cat *file*** **more *file*** **less *file***
  - Practice
    - **cal > calendar**
      - What is the command? What is the ***file.name***?
    - **ls calendar**
      - Does the file **calendar** exist?
    - **cat calendar** **more calendar** **less calendar**

# Files and Directories, Part I—Creating Files

- Practice/object lesson
  - At the prompt, type **`ls`**
    - Do you have a file called **`calendar`**?
  - At the prompt, type **`cat calendar`** (or **`more`** or **`less`**)
    - What are the contents?
  - Now type **`who > calendar`**
  - At the prompt, type **`ls`**
    - Do you have a file called **`calendar`**?
  - Now type **`cat calendar`** (or **`more`** or **`less`**)
    - What are the contents?
    - What has happened?

# Files and Directories, Part I—Creating Files

```
$ cat > workshop
  first line
  second line
  third line
  ^d
$ more workshop
$ who > people
$ ls
$ more calendar (or cat or less)
$ more workshop (or cat or less)
$ who > workshop
$ ls
$ more workshop (or cat or less)
$ more people (or cat or less)
```

# Screen Editors

- Cheat Sheet
- **`emacs`**
- **`vi`**
- **`pico`**

# Screen Editors—Cheat Sheet

- **cat**
  - Create a file: **cat > *file.name***
  - Read an existing file: **cat *file.name***
  - Cannot change any line other than current line
  - Quit: **^d**
- **vi**
  - Create a file/open existing file: **vi *file.name***
  - Insert mode: **i**
  - Change to command mode: **<ESC>**
  - Quit/save: **ZZ** (upper case; make sure you are in command mode)
- **emacs**
  - Create a file/open existing file: **emacs *file.name***
  - Save: **^x^s**
  - Quit: **^x^c**
- **pico**
  - Create a file/open existing file: **pico *file.name***
  - Quit: **^x**
  - You will be prompted to save
  - Not always on the system

# Screen Editors—`emacs`

- Can use **`emacs`** to open a file created with any other text editor
- Especially useful for writing long code—many helpful features
  - High overhead, though
  - Color highlighting for certain codes—advanced usage
- Free to download
- Widely available
- Document has prompts for user—relatively intuitive

# Screen Editors—`emacs`

- Create/open a file
  - **`emacs` *`file.name`***
    - If ***`file.name`*** exists, will open file
    - If ***`file.name`*** doesn't exist, will create a new file
- Look at the screen after invoking **`emacs`**
- Information areas
  - **Echo area**
    - Displays certain commands
    - Prompts input to a command
    - Cursor in **echo area**—can use any **`emacs`** editing tools that work on one line to change what has been typed
      - To abort a command started on **echo area**: **^g**
  - **Mode line**
    - Can be ignored for simple text editing on single files

# Screen Editors—emacs

- Position cursor
  - Use arrow keys if configured
  - **^p** (up) **^n** (down) **^b** (left) **^f** (right)
  - **^a** (beginning of line) **^e** (end of line)
- Position cursor with numerical arguments
  - Precede numerical argument with **<ESC>**
  - Examples
    - **<ESC>4^p** (move up 4 lines)
    - **<ESC>5^f** (move forward to 5th character)
    - **<ESC>8^n** (move down 8 lines

# Screen Editors—emacs

- Add text
  - Move cursor to correct position, start typing
- Delete text
  - **^d** (**delete** character at cursor)
  - **delete** key (**delete** character before cursor)
  - **<ESC>d** (**kill** word after cursor)
  - **^k** (**kill** from cursor to end of line)
  - **^y** (**yank** back a previous kill)
- Concepts
  - **delete**—does not save deleted characters to buffer
  - **kill**—killed characters go to buffer, limited capacity, deletes oldest **kill**s when new ones added

# Screen Editors—emacs

- Undo changes
  - **^x  u** (undo last change)
  - **^x^c** (leave without saving—prompt appears)
  - **<ESC>x** (restore buffer to original contents)
- Conserve CPU time
  - In documents with text already entered, insert a couple of blank lines and type there
    - Prevent screen from having to redraw with addition of every character

# Screen Editors—`emacs`

- Try this
  - At the prompt, type **`ls`**
    - Does file **`workshop`** exist? (It should.)
  - At the prompt, type **`emacs workshop`**
    - Identify the **echo line**
    - Type some lines
    - Note changes and information in **echo line** as you type
    - Identify **mode line** (hint: in reverse type)
    - Move cursor
      - Use arrow keys
      - Use key combinations
      - Use numerical arguments
    - Delete characters, undo typing

# Screen Editors—emacs

- Manage line length
  - Press **<return>** at end of line
  - Use **auto-fill mode** for current editing session only
    - **<ESC>x auto-fill-mode**
      - Toggle auto-fill-mode on or off
    - **<ESC>64<ESC>x set-fill-mode**
      - Set line length to 64 characters

# Screen Editors—emacs

- File management
  - Create new/open existing file — **emacs *file.name***
  - Save changes — **^x^s**
    - Does not exit file
  - Insert file into buffer at cursor — **^x i *file.name***
  - Read a file, no editing permitted — **^x^r *file.name***
  - Exit a file — **^x^c**

# Screen Editors—emacs

- Commands
  - More than 400
  - Have long names, but often have abbreviations
  - Abbreviations are bound to command
  - To refer to abbreviations, look at **command dispatch** table
- Long command name use
  - **<ESC>x command-name**
- Long name use—workarounds
  - Keystroke abbreviation (e.g., **^n**)
  - Typing assistance
  - List of possible commands in **echo line**
  - Prompts in **echo line**

# Screen Editors—emacs

- Assistance
  - Internet
    - Do NOT print **`emacs`** manual—it's about a million pages
  - Written manual
  - Reference card from manual
  - Online tutorial
  - Online **help** system
    - **^h** = **help** options
    - **^h^h** (possible **help** options, prompts you to type desired one)
    - Third **^h** displays what option means

# Screen Editors—`vi`

- Early form of word processing
- Not especially easy to use
- Good for quick edits
- Exists on all UNIX systems, so good to know
- Approximately 100 commands
  - Can do a lot with just a few commands

# Screen Editors—`vi`

- Create new/open existing file **`vi` *`file.name`***
- Basic **`vi`** concept
  - **Modes**
    - **Input**
      - Different ways to get into input mode
      - Only one way to get out of input mode **<ESC>**
    - **Command**
      - Always in **command** mode when you create a new or open an existing file in **`vi`**

# Screen Editors—`vi`

- Create a new/open an existing file
  - **`vi`** ***`file.name`***
  - **`vedit`** ***`file.name`***
- Move cursor
  - If system is so configured, use arrow keys
  - **`h`** (left) **`j`** (down) **`k`** (up) **`l`** (right)
- Enter insert mode (check bottom of screen)
  - Insert at cursor **`i`**
  - Append after cursor **`a`**
  - Append at end of line **`A`**
  - Insert at beginning of line **`I`**
  - Others

# Screen Editors—`vi`

- Exit insert mode
  - **<ESC>**
- Exiting file
  - Save changes, exit file **<ESC>ZZ**
  - Save changes, do not exit file **<ESC>:w**
  - Exit file, do not save changes **<ESC>:q!**
  - Save changes, exit file **<ESC>:wq**
  - (Note—I added **<ESC>** as a gentle reminder; it may not be necessary, as you may be in **command mode**, but just in case…)

# Screen Editors—`vi`

- Erase
  - **Delete** key
  - Sometimes use **^h**
- Deleting (make sure in command mode)
  - Delete character **x**
  - Delete line **dd**
  - Replace character **r**

# Screen Editors—vi

- Line management
  - Press <R**eturn>** at end of each line
  - Set line wrap at 15 characters from right margin
    - **<ESC>:set wm=15**
- Searching long file
  - Search forward for pattern **<ESC>/*pattern***
    - Search for next instance of pattern **n**
  - Search backward for pattern **<ESC>/*pattern***
    - Search for next instance of pattern **n**

# Screen Editors—`vi`

- Create a new file
  - **`vi`** ***`file.name`*** (***`file.name`*** doesn't exist)
  - Note a new file has a row of **~**
- Set margin wrap 16 characters from right margin
- Type some lines
  - Type **`i`** (note bottom of screen—**insert** mode)
- Do a search for a particular character, pattern
  - Be sure to press **<ESC>** to enter command mode
- Exit the file, saving the changes

# Screen Editors—`vi`

- Many more commands, features
  - This is a brief introduction
  - Again, this is enough to allow you to create files
  - Again, good editor for quick but powerful edits

# Screen Editors—`pico`

- **`pico`**
  - Comes with some UNIX mail systems
  - Simple
  - Menu-based
  - Limited
  - Use for very quick edits
  - Not always on system you are using
  - Can ask systems administrator for access
  - Check Cheat Sheet (earlier) for basics

# Files and Directories, Part II

- Concepts
- Manipulating
- Wildcards

# Files and Directories, Part II—Concepts

- **pathname**
  - Path through directory system to file
  - Example
    - **`/usr/Workshop1/Subdirectory/file.name`**
  - **`tail`** (basename)
    - Last part of **pathname**
  - **head**
    - Everything except the **`tail`**
  - **Absolute** (full) **pathname**
    - Shown when you type **`pwd`**

# Files and Directories, Part II—Concepts

- / (forward slash)—two meanings
  - Very first / in **absolute pathname** = root or top of file system
  - Every other / in absolute or **relative pathname** = end of directory or file name

# Files and Directories, Part II—Manipulating

- **`rm` *`-i -r file.name`***
  - Irreversible
- **`cp` *`-i file.name1 file.name2`*** or ***`file.name pathname`***
  - Irreversible; make sure you don't use existing ***`file.name`***
- **`mv` *`-i file.name1 file.name2`*** or ***`file.name pathname`***
  - Irreversible
- **`ln` *`file.name1 name.2`*** or ***`file.name pathname`***

# Files and Directories, Part II—Manipulating

- **`mkdir` *`directory.name`***
  - Create ***`directory.name`***
  - Must be unique in current directory
- **`rmdir` *`directory.name`***
  - ***`directory.name`*** must be empty (no files)
  - Delete files in ***`directory.name`***, then invoke **`rmdir`**
- **`cd` *`directory.name`***
  - Move to ***`directory.name`***
  - Invoke **`ls`** to see if ***directory.name*** in current directory
- **`pwd`**
  - Determine what current ***`directory.name`*** is

# Files and Directories, Part II—Manipulating

- Create some empty directories (may want first letter in upper case)
- Invoke **`ls`** command
- Invoke **`ls -F`** command—what is the difference?
- Invoke **`ls -a`** command—we'll discuss output later
- Move to one of new directories
- Create another new directory
- Invoke **ls** command (with or without options)
- Move to that new directory
- Create another new directory
- Move to new directory
- Invoke **`pwd`** command
- Type **`cd`** with no argument
- Invoke **`pwd`** command—where are you now?

# Files and Directories, Part II—Manipulating

- Invoke **`cd`** command to make sure you are in your **home directory**
- Invoke **`pwd`** command to ascertain where you are
- Create some empty files (do not use existing ***`file.name`***)
  - Easy way: **`touch`** ***`file.name`***
    - Verify creation of ***`file.name`*** with **`ls`**
    - Verify empty contents with **`cat`** ***`file.name`*** **`more`** ***`file.name`*** **`less`** ***`file.name`***
- Copy one of new ***`file.names`*** into ***`directory.name`***
- Move one of new ***`file.name`***s into ***`directory.name`***
- Invoke **`ls`** to see **home directory** contents
- **`cd`** into ***`directory.name`***
- Invoke **`ls`** to see directory contents
- Invoke **`cd`** with no arguments
- Invoke **`pwd`** to verify where you are

# Files and Directories, Part II—Wildcards

- Used to save time, typing
- Used for pattern searching
- Actually two types
  - **Wildcards**
    - For many simple UNIX tasks
  - **For general expressions**
    - For many more complex tasks

# Files and Directories, Part II—Wildcards

- **?**
  - Match one character and one character only
  - Example: **`ls arks?`**
    - Results, if any: a 5-letter ***`file.name`*** that begins with **`arks`** and ends with one character only
- ***
  - Match zero or more characters
  - Example: **`ls arks*`**
    - Results, if any: a 4-letter to n-letter ***`file.name`*** that begins with **`arks`**
- **[ ]**
  - Match one character of the ones (individually noted or in a range) in **[ ]**
  - Example: **`ls arks[a-d]`**
    - Results, if any: 5-letter ***`file.name`*** that begins with **`arks`** and ends with **`a`**, **`b`**, **`c`**, **`d`**

# Files and Directories, Part II—Wildcards

- **`cd`**
- **`pwd`**
- Create new directory **`States`**
- Move to **`States`**
- Create new files with **`touch`**
  - **`Touch alabama alaska florida massachusetts ohio`**
- **`ls`** with different wildcards (think what output will be)
  - **`ls al*`**
  - **`ls ?la`**
  - **`ls [a-f]*`**
  - **`ls [a-f]labama`**
  - **`ls m*`**
  - **`ls *a`**
  - **`ls ?la*`**
  - **`ls [b-z]labama`**
  - **`ls *[i]*`**
  - **`ls O*`**

# Files and Directories, Part II—Abbreviations

- `.` (dot)
  - Current working directory
  - Note: dot in UNIX overall has several uses
- `..` (two dots)
  - Directory above the one in which you are working (**parent directory**)
- `~` (tilde)
  - **Home directory**
- **`cd` *`[no arguments]`***
  - Takes you to your own **home directory**

# Files and Directories, Part II—Permissions

- Allow/restrict access to ***file.name***s, ***directory.name***s
- Command and structure
  - **chmod *ugo+rwx file.name***
  - **chmod *ugo+rwx directory.name***
- To see permissions
  - **ls -l** (print contents of directory in long form)
  - **ls -l *directory.name*** or **ls -l *file.name*** (print long form of ***directory.name*** or ***file.name***)

# Files and Directories, Part II—Permissions

- Examples of result of **`ls -l`**
  - **`drwxr-xr-x …`**
  - **`-rw-r--r-- …`**
  - Directory = **d**
  - Columns 2, 3, 4: **permission**s for **u**ser (**r**ead, **w**rite, **e**xecute)
  - Columns 5, 6, 7: **permission**s for **g**roup (**r**ead, **w**rite, **e**xecute)
  - Columns 8, 9, 10: **permission**s for **o**thers (**r**ead, **w**rite, **e**xecute)
- Example use
  - **`chmod g+rwx .`** (change **home directory permission** for **g**roup)
  - **`chmod u+rwx file.name`** (change **permissions** for ***`file.name`*** so **u**ser—owner—can **r**ead, **w**rite, **e**xecute )
  - **`chmod a+r directory.name`** (change **permission**s for ***`directory.name`*** so all can read it; **a**ll = **u**ser, **g**roup, **o**ther)

# Files and Directories, Part II—Permissions

- Create a file called **`filez`** (**`touch`**, **`cat`**, **`vi`**, **`emacs`**)
  - **`ls –l filez`**
  - **`chmod g_rwx filez`**
  - **`ls –l filez`**
- Create a directory called **`Dir.z`**
  - **`ls –l Dir.z`**
  - **`chmod a-rwx Dir.z`**
  - **`ls –l Dir.z`**

# UNIX Shell

- Concepts
- Job Control
- Redirection
- History
- **`alias`** (csh)
- File Name Completion (csh)
- Shell Scripts
- Variables
- Dot Files

# UNIX Shell—Concepts

- Two functions of shell
  - Command interpreter between machine and user
  - Programming language
    - Can string together basic commands to perform larger task
      - One way: put several commands on one line, separate commands with semicolon
    - Basic command structure
      - **`command -option argument`** **<Return>**
    - Can use **pipe** (**|**) to combine commands
      - **Pipe** takes output of one command, uses it as input to next command
      - Other redirections can be used, too

# UNIX Shell—Concepts

- **Pipe** example
  - **`ls -l | more`**
- Other redirect examples
  - **`lp -d oscps5 < file.name`**
  - **`mail address < file.name`**
  - **`who > who.file`**
  - **`date >> who.file`**
    - Look at **`who.file`** for results of **`>>`**

# UNIX Shell—Concepts

- Redirection
  - Redirect output to a file **>**
  - Override file protection, if set (csh option) **>!**
  - Redirect/append to existing file **>>**
  - Redirect input from file; get data out of a file; use with commands/programs that normally take input from terminal **<**
  - Combined redirects
    - Take data from file, process through initial command, redirect to a file **<>**
    - Act on information derived from last named file **><**
  - Take input, route it to two places (terminal and file.name)
    - **`tee [-a] file.name`** (**`-a`**—append, do not overwrite)

# UNIX Shell—Job Control

- **^z**
  - Stop current job
- **fg**
  - Resume stopped job
- **jobs**
  - List, label jobs you've begun

```
[2]    Stopped        vi file.name1
[3]    Stopped        vi file.name2
[4]    Stopped        vi file.name3
[5]    Running        command
```

- **fg *%n***
  - Brings job number ***n*** to foreground
- **bg *%n***
  - Sends job number ***n*** *to background*

# UNIX Shell—Job Control

- Can use **`ps`** (report on processes currently on computer) to find **PID**

```
PID     TT   STAT TIME Command
431     A5   R    0:05 sort file.name1
432     A5   R    0:00 ps
```

- **Kill**ing a job that is out of control—shell command
  - **kill** ***%ID*** or **`kill`** ***`pid`***
    - Even more sure **kill**: **`kill -9 %n`**
- On Glenn, can use **`qdel`** to kill job—PBS command
  - **`qdel`** ***`jobid`***

# UNIX Shell—History

- csh
  - Command: **`history`**
  - Default: 20 commands (number can be changed)
  - Repeat earlier commands: **`!!`** (repeat last command) or **`history`** ***`n`***
  - Able to modify earlier commands before repeating
  - Able to set **`history`** number for current session or permanently
    - **`set history=30`**
    - Modify **`.cshrc`** file with same **`set`** command
- Other shells
  - Command: **`history`**
  - Repeat last command: **`r`** (ksh) **`fc –s`** (bash)
  - Repeat ***n***th command: **`r`** ***`n`*** (ksh) **`fc –s`** ***`n`*** (bash)
  - ksh often can press **<ESC>k** to go through history (varies with system), edit with **`vi`**

# UNIX Shell—`alias` (csh)

- Purpose: rename, redefine, rearrange commands
- Can be temporary (for present session) or permanent (put in **`.cshrc`** or **`.profile`**)
- Can use arguments
- Can be used for compound (more than one) commands, complex commands, inside other **`alias`**es

# UNIX Shell—`alias` (csh)

- Summary
  - Print list of **`alias`**es: **`alias`** ***`[none] [none]`***
  - Create **`alias`**: **`alias`** ***`[none] abb command`***
  - Remove **`alias`**: **`unalias`** ***`[none] abb`***

# UNIX Shell—File Name Completion (csh)

- Purpose: speed up typing
- Format
  - ***command/file.name/dir.name chars*<ESC>**
  - If more than one command/file.name/dir.name has common characters, result will show the next common letters; then type another character followed by **<ESC>**
- Temporary use
  - At command line, type: **`set filec`**
- Permanent use
  - Put **`set filec`** in **`.cshrc`** file

# UNIX Shell—Shell Scripts

- Purpose
  - Construct new commands by creating file with desired commands
  - When you run ***shell.script***, new shell opens for duration of command, then closes
- Method
  - Create ***shell.script*** file with command(s) you want
- Run ***shell.script***
  - **csh *shell.script*** or **sh *shell.script*** OR
  - **chmod u+x *shell.script*** (what does this do?)
    - At prompt, type ***shell.script***

# UNIX Shell—Shell Scripts

- Flexible
  - Script can contain more than one command: separate with semicolons
  - Command can use command-line arguments
  - Can create loops and other programming features

# UNIX Shell—Shell Script

- OSC has software package called COMSOL—only one license
  - Command to check to see if someone is using COMSOL license:

```
/usr/local/sbin/multil lmstat -c 6660@license2.osc.edu -f COMSOL
```

  - Create a file called **`comlic`** with just the above command in it (use **`cat`**, **`emacs`**, or **`vi`** to create file)
  - Type **`ls -l comlic`** to check the permissions
  - Make the file executable by you: at the prompt, type **`chmod u+x comlic`**
  - Type **`ls -l comlic`** to check permissions again
  - To run the script, at the prompt, type **`./comlic`**

# UNIX Shell—Shell Scripts

- Sample
  - **vi i.remember**

```
echo This is the $0 command
echo My first argument is $1
echo My third argument is $3
echo Here are all my arguments: $*
```

  - **chmod u+x i.remember**
  - **i.remember three cats each wearing hats**
  - Output:

```
This is the i.remember command
My first argument is three
My third argument is each
Here are all my arguments: three cats each wearing hats
```

# UNIX Shell—Variables

- Purpose
  - Names that can have different values assigned
  - Some are built-in
  - Some user can define

# UNIX Shell—Variables

- Built-in variables
  - csh
    - Two kinds
      - Shell variable
        - » Known just to shell in which it's created
      - Environment shell variable
        - » Known to shell in which defined AND descendant shells
    - Determine shell variables user has
      - **`set`**
      - Variables—lower case
    - Determine environment variables user has
      - **`setenv`**
      - Variables—upper case

# UNIX Shell—Variables

- Obtaining value of variables
  - **echo $*variable*** or **echo $*VARIABLE***
  - Can use **$** construction to make shell script more general, more versatile, usable by other users in their own directories
    - Example: **cp $1 $HOME**
      - Anyone can use this, and file will be copied into that person's **$HOME**
- Note difference between following
  - **echo *TERM*** output ***TERM***
  - **echo $*TERM*** output value of variable ***TERM***

# UNIX Shell—Variables

- To set variables
  - Ordinary variables
    - **`set` *`var.name = value`***
      - Check to see if variable has been created by typing **`set`**
      - Get rid of variable by typing **`unset` *`var.name`***
  - Environment variables
    - **`setenv` *`VAR.NAME value`***
      - No equal sign
      - Variable name—upper case

# UNIX Shell—Dot Files

- **`.login`**
- **`.cshrc`**
- **`.profile`**
- Ordinary **`txt`** files
- Can be modified
- Can contain programs
- Can see them with **`ls -a`** command
- Environment variables—usually in **`.login`** file
- Ordinary shell variables, **`alias`**es—usually in **`.cshrc`** file

# Filters and Information Processing

- Concepts
- `spell`
- `grep`
- `find`

# Filters and Information Processing

- Several; can be extremely useful
- Include
  - **`pr`**
  - **`wc`**
  - **`sort`**
  - **`join`**
  - **`sed`** stream editor
  - **`nroff`**, **`troff`**

# Filters and Information Processing—`spell`

- **`spell`**
  - Primitive manner of spelling checking
  - Format
    - **`spell` *`file.name`***
      - Simple **`spell`** command
    - **`spell` *`file.name`* `|` `more`**
      - **`spell`** command when you have more than one screen of output
    - **`spell` *`file.name`* `>` *`spell.file`***
      - When you want to capture output in a file called ***`spell.file`***
    - **`spell` *`file.name`* `|` `lp -d` *`printer.name`***
      - When you don't want to create a file, just print the output to a particular printer

# Filters and Information Processing—`grep`

- Find which files contain a particular word, etc.
- Examples:
  - **`grep` *`userid`* `/etc/passwd`**
    - Search for a particular ***`userid`*** in a systems file called **`/etc/passwd`**
  - **`grep` *`expression file.name1 file.name2`***
    - Search for a particular ***`expression`*** in each of two named files
  - **`grep` *`expression *.c`***
    - Search for ***`expression`*** in all files in current directory that end in **`.c`**
  - **`grep` *`expression */*`**
    - Search for ***`expression`*** in all subdirectories of current directory

# Filters and Information Processing—`grep`

- Options
  - **`-n`** = precede each found line with line number in file
  - **`-v`** = print lines that don't match ***`expression`***
  - **`-c`** = count of the number of lines that match, don't print lines themselves
  - **`-i`** = all lines that match without being case-sensitive

# Filters and Information Processing—`grep`

- Forms of regular expression
  - Differ from UNIX wildcards but have similar purpose
- Some regular expressions
  - `.` (dot) = any one character (= ? Of UNIX wildcards)
  - `[`***`string`***`]` = matches any character in string (same as UNIX wildcards)
  - `^`***`expression`*** = match lines that begin with expression
  - `\`***`character`*** = turn off special meaning of character

# Filters and Information Processing—`find`

- Search for files that meet some criterion
  - Name
  - Size
  - Files not accessed for certain number of days
  - Files having certain number of links
  - More
- General format
  - **`find` *`directory.pathname search.criterion action`***
    - ***`directory.pathname`*** = **pathname** of directory to be recursively searched
    - ***`search.criterion`*** = files that are sought
    - ***`action`*** = what to do with found files

# Filters and Information Processing—`find`

- Arguments
  - ***directory.pathname(s)***
  - ***search.criteria***
  - ***action(s)***
- ***search.criteria***
  - **-name *file.name*** = files named ***file.name***
  - **-size *n*** = files of size ***n*** blocks
  - **-links *n*** = files with ***n*** links
  - **-atime *n*** = files accessed ***n*** days ago
  - **-mtime *n*** = files modified ***n*** days ago
  - **-newer *file.name*** = files modified more recently than ***file.name***

# Filters and Information Processing—`find`

- ***`actions`***
  - **`-print`** = print **pathnames** of found files
  - **`-exec`** ***`command`*** **`{ } \;`** = executes given command on finding file ( **`{ }`** = found file)
  - **`-ok`** command \; = **`-exec`**, except prompts user to say **`y`** or **`n`** to executing command

# Selected Sources for Further Enlightenment

- Internet (of course)
- Arthur, Lowell Jay, et al. UNIX Shell Programming, ISBN 0-471-16894-7
- Cameron, Debra. Learning GNU Emacs, ISBN 0-596-0064-9
- Dougherty, Dale. sed & awk, ISBN 1=56592-225-5
- Dyson, et al. UNIX Complete, ISBN 0-7821-2528-X
- Kernighan, Brian W., et al. The UNIX Programming Environment, ISBN 0-13-937681-X
- Kochan, Stephen G., et al. UNIX Shell Programming, ISBN 0-672-48448-X
- Lamb, Linda, et al. Learning the vi Editor, ISBN1-56592-426-6
- Lasser, John. Think Unix, ISBN0-7897-2376-X

# Selected Sources for Further Enlightenment


- Martin, Don, et al. UNIX Primer Plus, ISBN 1-57169-165-0
- Michael, Randall K. Mastering Unix Shell Scripting, ISBN 0-471-21821-9
- Muster, John. UNIX Made Easy, ISBN 0-07-219314-X (**my favorite**)
- Ray, Deborah S., et al. Visual Quickstart Guide: UNIX, ISBN 0-201-35395-4
- Raymond, Eric S. The Art of UNIX Programming, ISBN 0-13-142901-9
- Reichard, Kevin, et al. UNIX in Plain English, ISBN 0-7645-7011-0
- Robbins, Arnold, et al. Classic Shell Scripting, ISBN 0-596-00595-4
- Robbins, Arnold. UNIX in a Nutshell, ISBN1-56592-427-4
- Robbins, Arnold. vi Editor, Pocket Reference, ISBN 1-56592-497-5
- Rosen, Kenneth, et al. The Complete Reference: UNIX, ISBN-13: 978-0-07-226336-7; ISBN-10: 0-07-226336-9
- Taylor, Dave. SAMS Teach Yourself Unix, ISBN 0-672-32127-0