Overview of Basic UNIX/Linux Commands and Terminology

Commands Covered

- This tutorial is interactive
- Following all steps will help to understand the commands:
  - ***pwd*** shows current directory
  - ***ls*** shows directory contents
  - ***mkdir*** creates a directory
  - ***touch*** creates a file
  - ***cd*** changes directories
  - ***mv*** moves a file from one directory to another
  - ***mv*** also renames a file
  - ***cp*** copies a file
  - ***rm*** deletes a file
  - ***rmdi***r deletes a directory
  - ***g++*** used to compile programs on UNIX/Linux servers

Organizing Your Work

- With Microsoft Windows, files are organized using nested ***folders***
- In **UNIX** and **Linux**, a ***folder*** is known as a ***directory***
- A ***directory*** may contain files or nested directories (**subdirectories**)
- When you first open a terminal window, you will be in your ***home directory***, the location where your files will be stored

Name Selection - 1

- Filenames and directory names in **UNIX/Linux** are ***Case Sensitive***
  - Examples:

    data.txt

    Data.txt

    data.TXT

    are treated as **three different file names**

Name Selection - 2

- **UNIX/Linux** systems prefer ***No Spaces*** within a filename or directory name
  - Use letters, digits, hyphens, and/or underscores
  - Example: To name a directory, use
    Project10 or ***Project_10*** instead of ***Project 10***
  - If you add spaces to a name, you must enclose the entire name within quotes ***"Project 10"***
    - ***For simplicity, I recommend omitting spaces!!***
  - A period and an extension may be appended to indicate the type of file (**.txt** for text file, **.cpp** for C++)
  - Examples:
    Project10.cpp ← a C++ source code file
    P10_input1.txt ← a text input file for Project10

Tab Complete

- Type the first few letters of a command or file name followed by pressing tab
  - Tab Complete will finish the name
    - Completely if it is unique or
    - Stopping once the name is no longer unique
    - Use with completing file names after commands
  - Example:
    mars $ ls
    Project1.cpp Project2.cpp Project3.cpp
    Typing a <command> followed by P<tab> at the prompt gives:
    mars $ <command> Project
    Typing a number and then <tab> again completes the name.
    Pressing <tab> twice will show all possibilities for completing the name

Print Working Directory - pwd

- Prints the ***name*** of the current working directory
- If you have just opened a terminal window, you will be in your home directory

  For example, assuming your login is hjs0001

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$
```


List Directory Contents - ls

- When used alone, ls lists the contents of the current working directory
  - Example for an empty directory:

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  ls   (nothing is shown)
-bash-3.2$
```

  - Example for a directory with files:

```
-bash-3.2$  ls
somefile   anotherfile    project1   project2
-bash-3.2$
```

More on ls - 1

- When used with the name of a directory, ls lists the contents of that directory
  - Example (project2 is a directory):

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  ls
somefile   anotherfile    project1   project2
-bash-3.2$  ls  project2
project2.cpp  p2_input.txt
-bash-3.2$
```


More on ls - 2

- To see additional information about the contents of a directory, type ls -l
  - Lines that start with a **d** indicate a directory
  - Lines that start with a **-** indicate a file
  - Lines that start with a **l** indicate a link
  - You will also see the file permissions, the file owner, file size, and last date of modification along with the name of the file or directory

Make Directory - mkdir

- Creates a new directory within the current working directory

- For example, to create a directory named example in the current working directory, type

```
-bash-3.2$ cd
-bash-3.2$ mkdir example
-bash-3.2$ ls
example
```


Change Directory - cd

- When used with a valid directory name, cd switches the current working directory to the specified directory

  – Example:

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  ls
example
-bash-3.2$  cd  example
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  ls –a  (this shows all files)
.     ..          (every directory has 2
-bash-3.2$        directories when created)
```

More on cd - 1

- When used alone, cd switches the current working directory back to your home directory
  - Example:

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  cd
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  ls
example
```


More on cd - 2

- To switch back to the parent directory of a current working directory, type cd ..
- Make sure that you are in the example directory – if pwd gives your home directory, then perform the following:

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  cd example
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  ls –a
.   ..
-bash-3.2$
```

hjs0001 is the parent directory of example

  - **Now to move up to the *parent directory of example*:**

```
-bash-3.2$  cd ..  (the directory .. Represents the parent
-bash-3.2$  pwd    directory of the current directory)
/home/student/hjs0001
```

More on cd - 3

- To switch back to the parent directory of a current working directory, type cd ..

  For example :

```
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  cd ..
-bash-3.2$  pwd
/home/student
-bash-3.2$  cd
-bash-3.2$  pwd
/home/student/hjs0001
```

student is the parent directory of hjs0001

cd moves you back to your home directory

Create an empty file - touch

- The touch command creates a new empty file if the file does not exist
- Make your current working directory the example directory by using the following commands:

```
-bash-3.2$  cd
-bash-3.2$  cd example
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$ ls
-bash-3.2$
```

- Next create a file to move by typing the command: -bash-3.2$ touch project1.cpp

```
-bash-3.2$ ls
project1.cpp
-bash-3.2$
```

Move/Rename Files - mv

- Moves or renames the source file to the specified destination (directory or another file)
- To rename the file from project1.cpp to project1.cpp.bk in the current working directory

```
-bash-3.2$  ls
project1.cpp
-bash-3.2$ mv project1.cpp  project1.cpp.bk
```

source file — destination file

```
-bash-3.2$  ls
project1.cpp.bk
```


More on mv -1

- To move the file project1.cpp.bk to directory Projects keeping the same filename.
- First create the directory Projects

```
-bash-3.2$  mkdir Projects
-bash-3.2$  ls
 Projects    project1.cpp.bk

-bash-3.2$ mv project1.cpp.bk  Projects
```

source file — destination Directory

```
-bash-3.2$  ls
Projects
-bash-3.2$  ls Projects
project1.cpp.bk
```

More on mv -2

- To move the file project1.cpp.bk from the Projects directory to the parent directory and change the name to project2.cpp :

```
-bash-3.2$ cd Projects
-bash-3.2$ pwd
/home/student/hjs0001/example/Projects
-bash-3.2$  ls
project1.cpp.bk

-bash-3.2$ mv project1.cpp.bk  ../project2.cpp
```

source file — destination file location

```
-bash-3.2$  ls
-bash-3.2$  cd ..          (move up to the example directory)
-bash-3.2$  ls
      Projects   project2.cpp
```


Copy Files - cp

- Copies the source file to the specified destination file
- For example, to make a backup copy of project2.cpp in the current working directory

```
-bash-3.2$  ls
 Projects   project2.cpp
-bash-3.2$ cp project2.cpp  project2_backup.cpp
```

source file — destination file

```
-bash-3.2$  ls
 Projects   project2.cpp   project2_backup.cpp
```

More on cp - 1

- A file can be copied from one directory (Programs) to another directory (Projects)

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  ls
Projects   project2.cpp   project2_backup.cp
-bash-3.2$  mkdir  Programs
-bash-3.2$  cp project2.cpp Programs
-bash-3.2$  ls
Programs  Projects   project2.cpp
  project2_backup.cp
-bash-3.2$  cd Programs
-bash-3.2$  ls
project2.cpp
```

(example continued on next slide)

More on cp - 2

- A file can be copied from one directory (Programs) to another directory (Projects) and renamed in the process

```
-bash-3.2$  pwd
/home/student/hjs0001/example/Programs
-bash-3.2$  cp project2.cpp  ../Projects/project3.cpp
-bash-3.2$  ls
project2.cpp
-bash-3.2$
-bash-3.2$  cd ../Projects
-bash-3.2$  pwd
/home/student/hjs0001/example/Projects
-bash-3.2$  ls
project3.cpp
-bash-3.2$
```

Remove File - rm

- Removes specified file

```
-bash-3.2$  cd
-bash-3.2$  pwd
/home/student/hjs0001
-bash-3.2$  cd example
-bash-3.2$  ls
Programs  Projects   project2.cpp   project2_backup.cp
-bash-3.2$  rm project2_backup.cpp
-bash-3.2$  ls
Programs  Projects   project2.cpp
-bash-3.2$
```


Remove Directory - rmdir

- Warning displayed if directory is not empty

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  ls
Programs  Projects   project2.cpp
-bash-3.2$  rmdir Projects
rmdir: directory “Projects/": Directory not empty
-bash-3.2$  ls
Programs  Projects   project2.cpp
-bash-3.2$
```

More on rmdir -1

- Removes specified directory – if empty

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  cd Projects
-bash-3.2$  ls
project3.cpp
-bash-3.2$  rm project3.cpp
-bash-3.2$  cd ..
-bash-3.2$  ls
Programs  Projects   project2.cpp
-bash-3.2$  rmdir Projects
-bash-3.2$  ls
Programs    project2.cpp
-bash-3.2$
```


More on rmdir -2

- Removes specified directory – if empty

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  cd Programs
-bash-3.2$  ls
project2.cpp
-bash-3.2$  rm project2.cpp
-bash-3.2$  cd ..
-bash-3.2$  rmdir Programs
-bash-3.2$  ls
project2.cpp
-bash-3.2$ rm project2.cpp
```

More on rmdir -3

- Removes specified directory – if empty

```
-bash-3.2$  pwd
/home/student/hjs0001/example
-bash-3.2$  ls
-bash-3.2$
-bash-3.2$  cd ..
-bash-3.2$  rmdir example
-bash-3.2$  ls
-bash-3.2$
-bash-3.2$  pwd
/home/student/hjs0001
```


Viewing File Contents

- To view the contents of a file use
  - more – displays one screen page at a time
  - cat – displays entire contents without pausing
  - less – displays one screen page at a time
- The following keys navigate the file
  - return – go down one line at a time
  - Space bar – go down one screen page
  - q – quit (exit out of the view)
  - b – go back one page

Proceed with Caution

- There is ***no undo command*** when using UNIX or Linux from the command line
  - ***It is possible to delete or overwrite your work without receiving a confirmation prompt***

```
-bash-3.2$  rm project10.cpp
-bash-3.2$
```

- We strongly suggest that you enable automatic backups (at a frequent interval) within your text editing program to facilitate recovery in case of a command line error or in case of a text editor crash

History

- The UNIX command **history** provides a listing of the previous commands run
  - Up and down arrow keys move through the list on the command line
  - !# (where # represents a number from the history list) runs the command from the history list with that number (i.e. !232)
  - !<letter(s)> runs the most recent command starting with the letter(s) given (i.e. !g++)

Editing Your Programs

- Sun Solaris
  - Use ***nedit***

    ```
    -bash-3.2$ nedit project10.cpp &
    ```
  - ***gedit*** is available but tends to crash often

- blackhawk/eagle Linux
  - Use ***gedit***

    ```
    -bash-3.2$ gedit project10.cpp &
    ```
- Regardless of the editor you select, enable frequent automatic backups to provide a means of recovery in the event that the text editor crashes

Basic Compile Commands

- Sun Solaris: C++ compiler named CC

  -bash-3.2$ CC $\underbrace{\text{project10.cpp}}_{\text{source file}}$ -o $\underbrace{\text{project10}}_{\text{executable file}}$
- eagle/blackhawk Linux: C++ compiler g++

  -bash-3.2$ g++ $\underbrace{\text{project10.cpp}}_{\text{source file}}$ -o $\underbrace{\text{project10}}_{\text{executable file}}$
- For compiling and linking multi-file programs, see the ***make Tutorial***

Manual Page - man

- Displays the manual page for the specified UNIX command
- For example, to display the manual page for ls

```
-bash-3.2$ man ls
```

  - To advance line-by-line through manual page, hit **RETURN**
  - To advance page-by-page through manual page, hit **SPACEBAR**
  - To quit viewing the manual page, hit **Q**

Input/Output Redirection - 1

- The Standard Input Device (***stdin***) is normally the keyboard
- The Standard Output Device (***stdout***) is normally the monitor
  - Example:

```
// The code below attempts to input an
// integer from stdin and write it to stdout
int  someInt;
cin >> someInt;
cout << someInt;
```

- With UNIX/Linux, one can
  - redirect ***stdin*** inputs to come from a specified file or
  - redirect ***stdout*** outputs to be written to specified file

Input/Output Redirection - 2

- Assume program1 inputs from *stdin* and outputs to *stdout*
- Execution of the command below causes program1 to execute normally

```
bash-3.2$  ./program1
```

- Execution of the commands below will cause program1 to input from infile1.txt and then execute again with input from infile2.txt without having to modify program1

```
bash-3.2$  ./program1 < infile1.txt
bash-3.2$  ./program1 < infile2.txt
```


Input/Output Redirection - 3

- Assume program1 inputs from *stdin* and outputs to *stdout*
- Execution of the command below will cause program1 to output to outfile1.txt without having to modify program1

```
bash-3.2$  ./program1 > outfile1.txt
```

- Execution of the command below redirects input and output

```
bash-3.2$  ./program1 < infile1.txt >
  outfile1.txt
```

Input/Output Redirection - 3

File input/output redirection is not the same as reading from files or writing to files using user defined input and output streams.

Input redirection causes input expected from the standard input stream (the keyboard) to instead be read from the specified input file.

Output redirection causes output destined for the standard output stream (the terminal) to instead be sent to the specified output file.

Input/output redirection affects stream flow <u>for the standard input/output streams only</u>. It does not affect user declared input and output file streams.