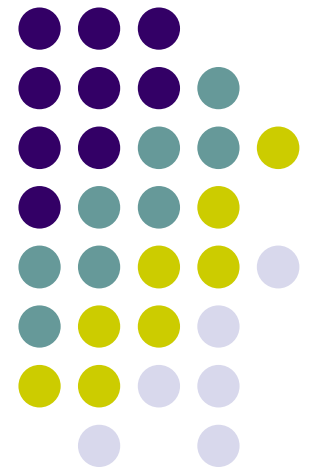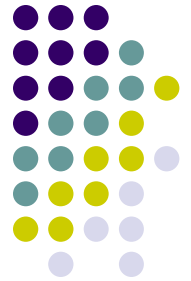# Operating Systems and Using Linux

Curtosey of John Y. Park

Operating Systems and Using Linux

Topics

- What is an Operating System?
- Linux Overview
- Frequently Used Linux Commands

What is an Operating System?

- A computer program that:
  - Controls how the CPU, memory and I/O devices work together to execute programs
  - Performs many operations, such as:
    - Allows you to communicate with the computer (tell it what to do)
    - Controls access (login) to the computer
    - Keeps track of all processes currently running
- Often referred to as simply OS

What is an Operating System?

- Provides a uniform interface for users and programs to access changing, evolving hardware (H/W)

- Very different H/W platforms can support a common OS (partially custom-written, of course) (standard "PC", Sony PSP can both run Linux)

- One H/W platform can support multiple OSs
  - E.g.: Latest Macs can run MacOS *or* Windows

How Do I Communicate With the Computer Using the OS?

- You communicate using the particular OS's **user interface.**

  - **Graphical User Interface (GUI)** – Windows, Linux

  - **Command-driven interface** - DOS, UNIX, Linux

- We will be using the **Linux** operating system, which is very similar to UNIX. Notice that it is listed as both GUI and Command-driven.

GUI vs. Command-driven

- We will be using both the GUI version of Linux and the Command-driven Interface.

- When you connect to GL through TeraTerm, you are using only the Command-driven Interface.

- When you reboot the computer into Linux, you will use both the GUI and the Command-driven Interface.

6

Example of Command-driven

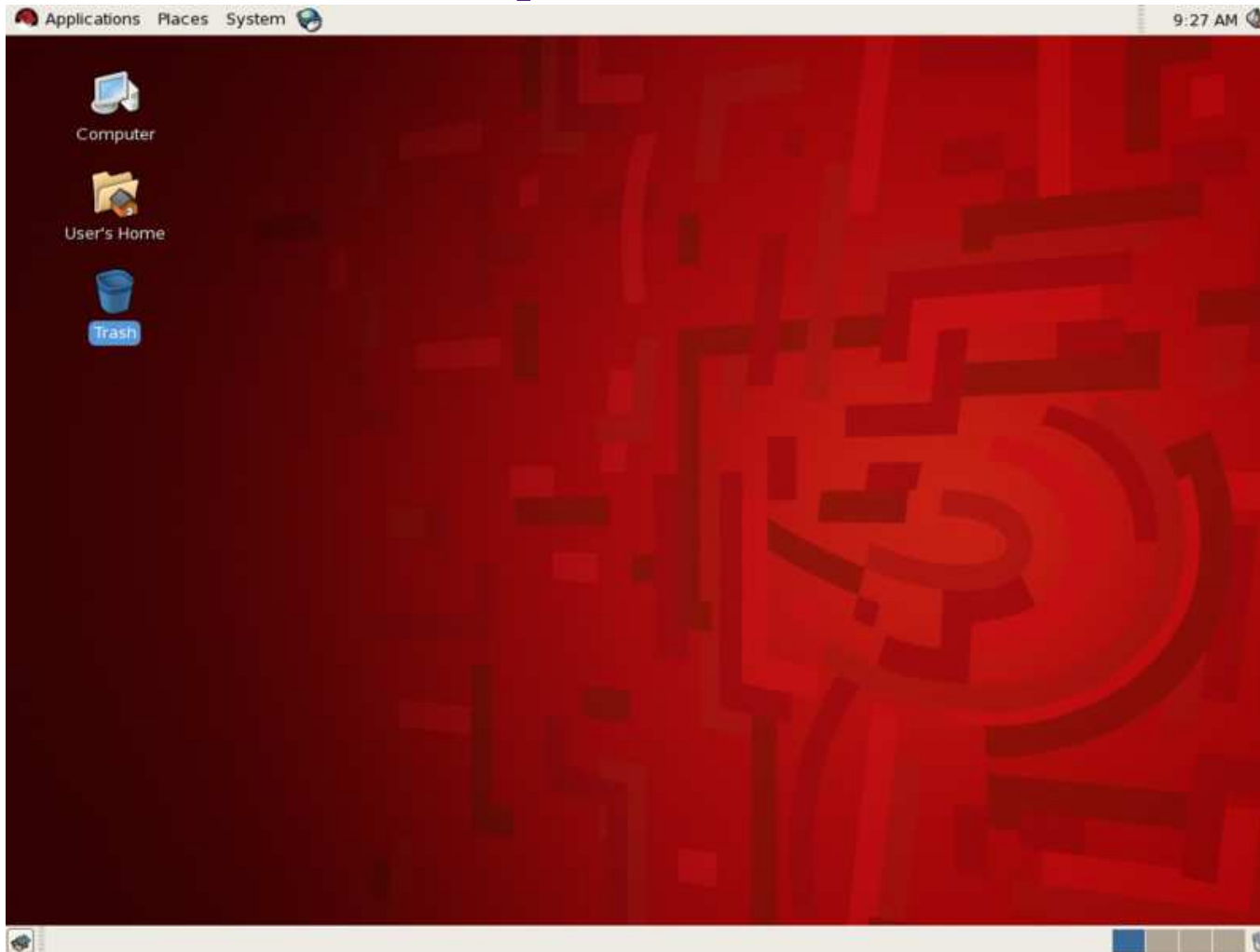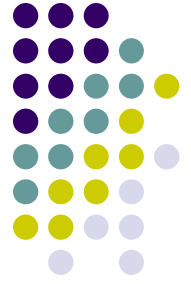*Screenshot of connection to linux3.gl.umbc.edu*

Example of GUI

Screenshot of Fedora 7

8

Another Example of GUI

Screenshot of Red Hat Enterprise Linux (RHEL) 5

Why a GUI?

- GUIs are sometimes better, because:
  - Give a good sense of "where I am"
  - Succinct visual summary of small sets
  - Easier to find "forgotten" target, then act on it
  - Simple to execute default behavior
    - Otherwise, often resort to complex "environments"

Why a Command Line?

- Command lines are sometimes better, because:
    - Easier to operate on large sets
    - Convenient if you remember filenames (and you should)
    - Can act on multiple objects in disparate locations
    - Easier if no simple default behavior

How Do I Communicate With the Computer Using the OS? (con't)

- When you **log in** to the Linux system here, a **user prompt** will be displayed:

```
linux#[1]% _
```

  where # is the number of the Linux server to which you have connected. You may use any of the Linux servers: linux1, linux2 or linux3.

- The number in the brackets will change as you work. It is the "number" of the command that you are about to type.
- If this prompt is not on the screen at any time, you are <u>not</u> communicating with the OS.
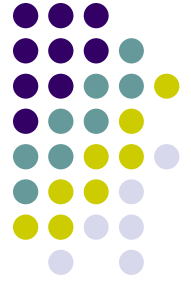
Linux Overview

- Files and Filenames

- Directories and Subdirectories

- Frequently Used Commands

Files

- A file is a sequence of bytes.
- It can be created by
  - a text editor (XEmacs or Notepad)
  - a computer program (such as a C program)
- It may contain a program, data, a document, or other information .
- Files that contain other files are called directories (sometimes called folders).

Linux Filenames

- Restrictions
  - Typically do not have spaces or other reserved characters
  - Have a maximum length (typically 255 characters but who wants to type that much!)
  - Are case sensitive

- For this class, you should stick with filenames that contain only letters (uppercase or lowercase), numbers, and the underscore (_) or hypen (-). No spaces!

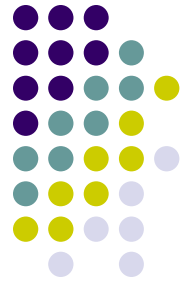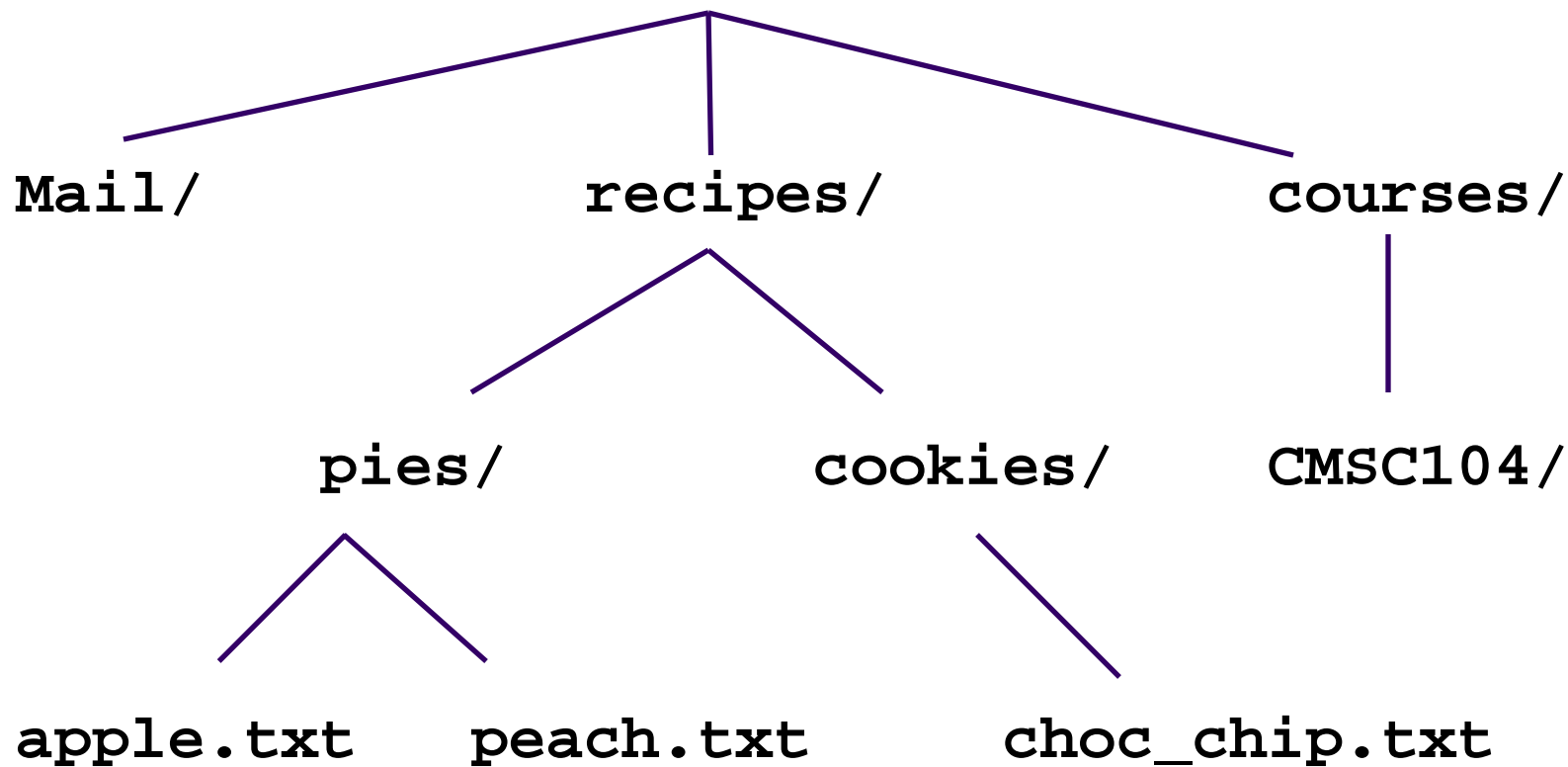- Some examples: `firefox.exe`, `things2do.txt`, `dinner_menu.pdf`
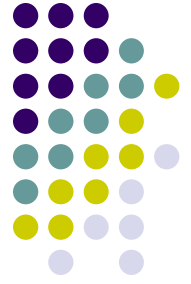
Directories

- Directories contain files or other directories called subdirectories. They may also be empty.

- Directories are organized in a hierarchical fashion.

- They help us to keep our files organized.

Example Directory Tree

`/afs/umbc.edu/users/j/d/jdoe28/home/`

```
Mail/              recipes/              courses/

              pies/        cookies/      CMSC104/

apple.txt   peach.txt     choc_chip.txt
```
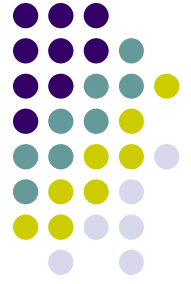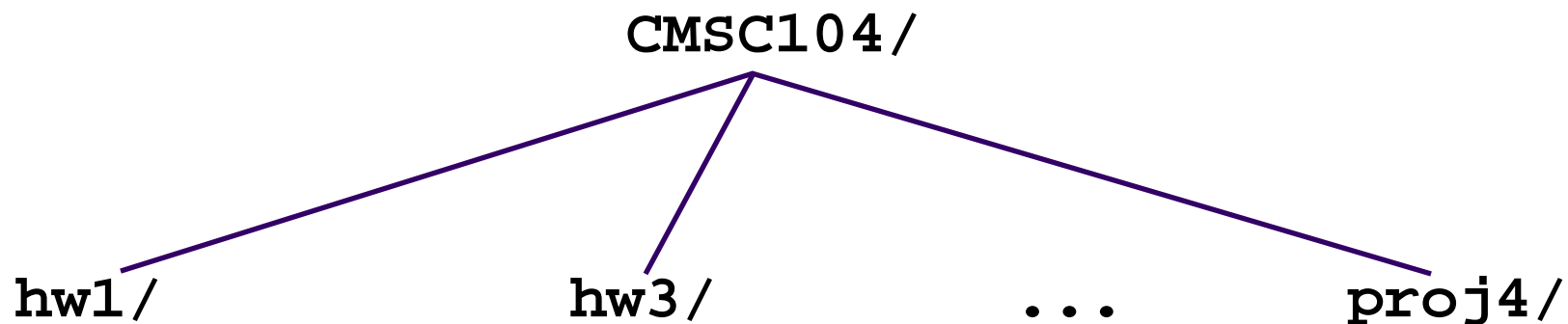
Subdirectories

- Are used for organizing your files
- For example,
  - make a subdirectory for CMSC104
  - make subdirectories for each project

```
                    CMSC104/


  hw1/              hw3/           ...        proj4/
```

More Directories

- Your **home directory** is where you are located when you log in

  (e.g., /afs/umbc.edu/users/j/d/jdoe28/home/).

- The **current directory** is where you are located at any time while you are using the system.

- The **/** (pronounced "slash") is the root directory in Linux.

- Files within the same directory must be given unique names.

- **Paths** allow us to give the same name to different files located in different directories.

- Each running program has a current directory and all filenames are implicitly assumed to start with the name of that directory unless they begin with a slash.

Moving in the Directory Tree

- **.** (**dot**) is the current directory.
- **..** (**dot-dot**) is the parent directory.
- Use the Linux command **cd** to change directories.
- Use dot-dot to move up the tree.
  - **cd ..**
- Use the directory name to move down.
  - **cd recipes**
- Use the complete directory name (path name) to move anywhere.
  - **cd /afs/umbc.edu/users/j/d/jdoe28/home/recipes/**

Absolute Path

- The absolute path is a path that contains the root directory and all other subdirectories you need to access the file
- It points to the same location in the directory tree regardless of the current working directory
- An example of an absolute path

`/afs/umbc.edu/users/j/d/jdoe28/home/recipes/`

Starts with
/

Relative Path

- The relative path is a partial path to a file in relation to the current working directory

- If inside of the home directory in the previous directory example, a relative path would be

`recipes/cookies/`

Does not start with /

Wildcard Characters

- You will find wildcard characters useful when manipulating files (e.g., listing or moving them).

- The wildcard characters are * and ?

- ? is used to represent any single character.

  - For example, `ls hw?.txt` would match the files `hw1.txt` and `hw2.txt` but not `hw123.txt`

- * is used to represent 0 or more characters.

  - For example, `ls hw*.txt` would match the files `hw1.txt` and `hw2.txt`, as well as `hw.txt`, `hw123.txt` and `hw_assignment.txt`
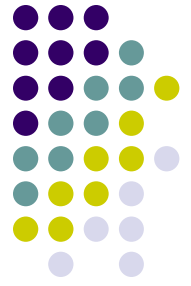
What is a "Shell"?

- The "most important program in the OS" ☺

- Your primary means of controlling the OS

- On Linux, just another program!
  - Can use other shells: sh, csh, bash, tcsh

- Can be programmed to do complex tasks

- Every command (almost) is just running another program

- Main differences are in syntax, ease of use
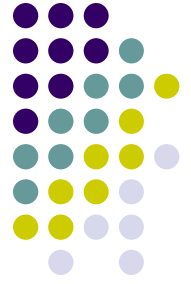
Common Commands

- Directory operations:
  - pwd, cd, mkdir, rmdir
- File manipulation:
  - ls, rm, cp, mv, cat
- File examination
  - cat, more, less, head, tail, file
- File editing
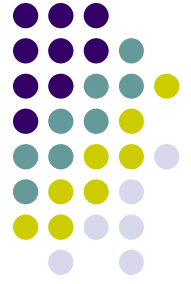 - ed, emacs, sed
- Misc (pine, find, etc.)

I/O Redirection

- All programs read from standard input "channel", write to standard output "channel"
  - Called "file descriptors"
- Shell can manipulate these file descriptors before executing command (i.e., program)
- Devices and files treated similarly
- "<": redirect input
- ">": redirect output

I/O Redirection

- Examples:
  - % ls > my-files.txt
  - % wc < my-files.txt

Pipes

- Communications channel *between* two programs
  - Can think of as a temporary file that first program writes to, second program then reads from
- Syntax:

  % program1 | program2

- Example:

  % ls | wc

  will give you the number of files you have

Command Line Editing

- Allows command to be edited before being executed

- Uses subset of emacs commands:
  - Ctl-B, Ctl-F, Ctl-A, Ctl-E, <Backspace>, Ctl-D

- Allows previous commands to be recalled, then optionally edited

- Very convenient for:
  - Typos
  - Repetitive commands