

# CISC 3130

# Unix System Programming

Xiaolan Zhang  
Spring 2013

# Outline

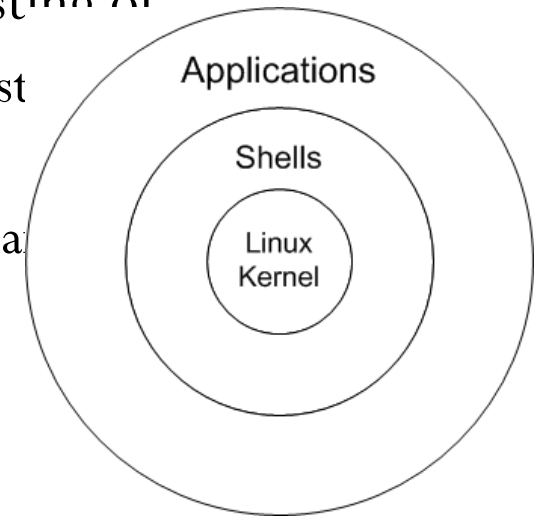
- Important course information
  - Objective
  - Roadmap
  - Requirement & policy
- Brief history of GNU/Linux
- GNU/Linux Architecture
- Getting started
  - Log in, simple command, shell

# You've written programs...

- Lot of times, one can use existing tools to implement new functionalities
- Real world applications are complicated:
  - generate input & output, or have GUI
  - communicate with other program (local or remote)
  - use multiple processes or threads for improved interactivities
  - Needs to be profiled/tested to improve performance

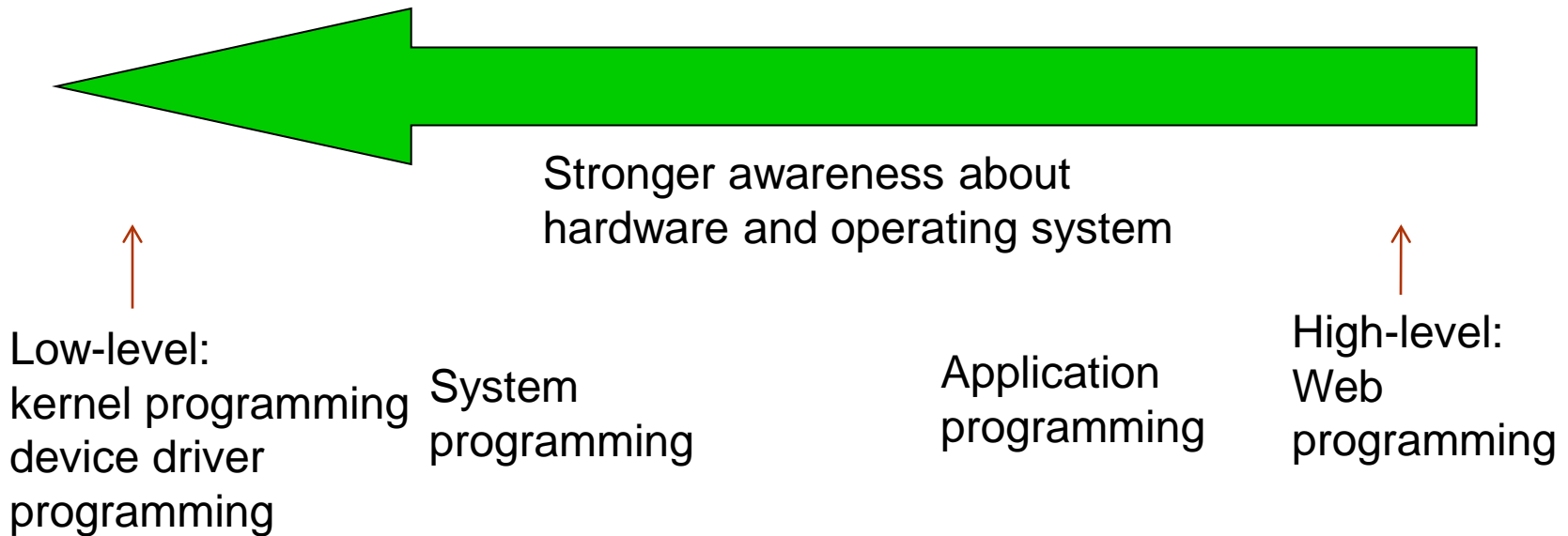
# About this course

- UNIX: time-sharing operating system, consisting of
  - kernel (program that controls and allocates system resources)
  - Essential programs: compilers, editors, command utilities
- Linux is a variation of Unix
  - programming environment is very similar



# About this course

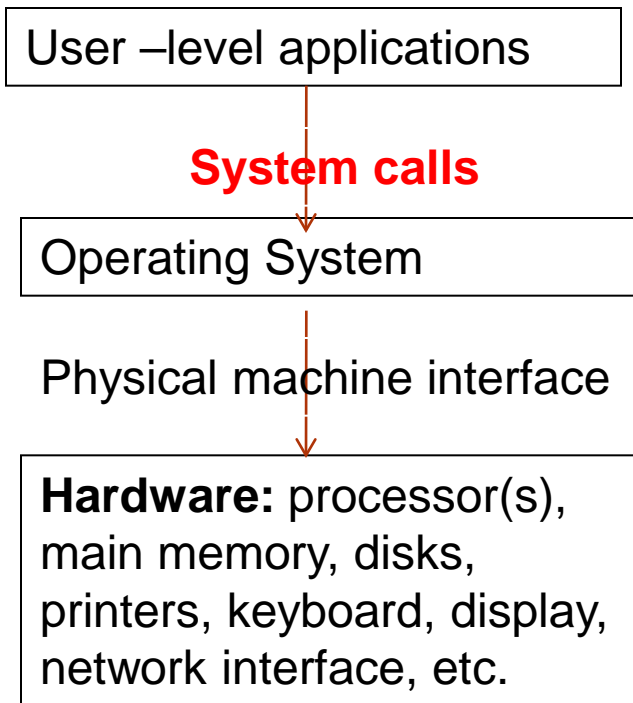
- Many levels of programming:



# Operating System, Kernel

- **operating system**: two different meanings
  - the entire package consisting of **central software managing a computer's resources** and all of **accompanying standard software tools**, such as command-line interpreters, graphical user interfaces, file utilities, and editors.
  - central software that manages and allocates computer resources (i.e., CPU, RAM, and devices).
- **kernel** is often used as a synonym for second meaning

# What is Operating System?



- From app. programmer's point of view:
  - O.S. manages hardware resources
  - O.S. provides user programs with a simpler interface, i.e. system calls
    - `cnt=read(fd, buffer,nbytes)`
    - `getc()` etc.
- We will encounter OS concepts, inevitably.

# Kernel Functionalities: Process scheduling

- Managing one or more central processing units (CPUs),
- Unix: a preemptive **multitasking operating system**
  - multiple processes (i.e., running programs) can simultaneously reside in memory and each may receive use of the CPU(s).
  - **Preemptive**: scheduler can preempt (or interrupt) a process, and resume its execution later => to support interactive responses
    - the processors are allowed to spend finite chunks of time (*quanta, or timeslices*) per process



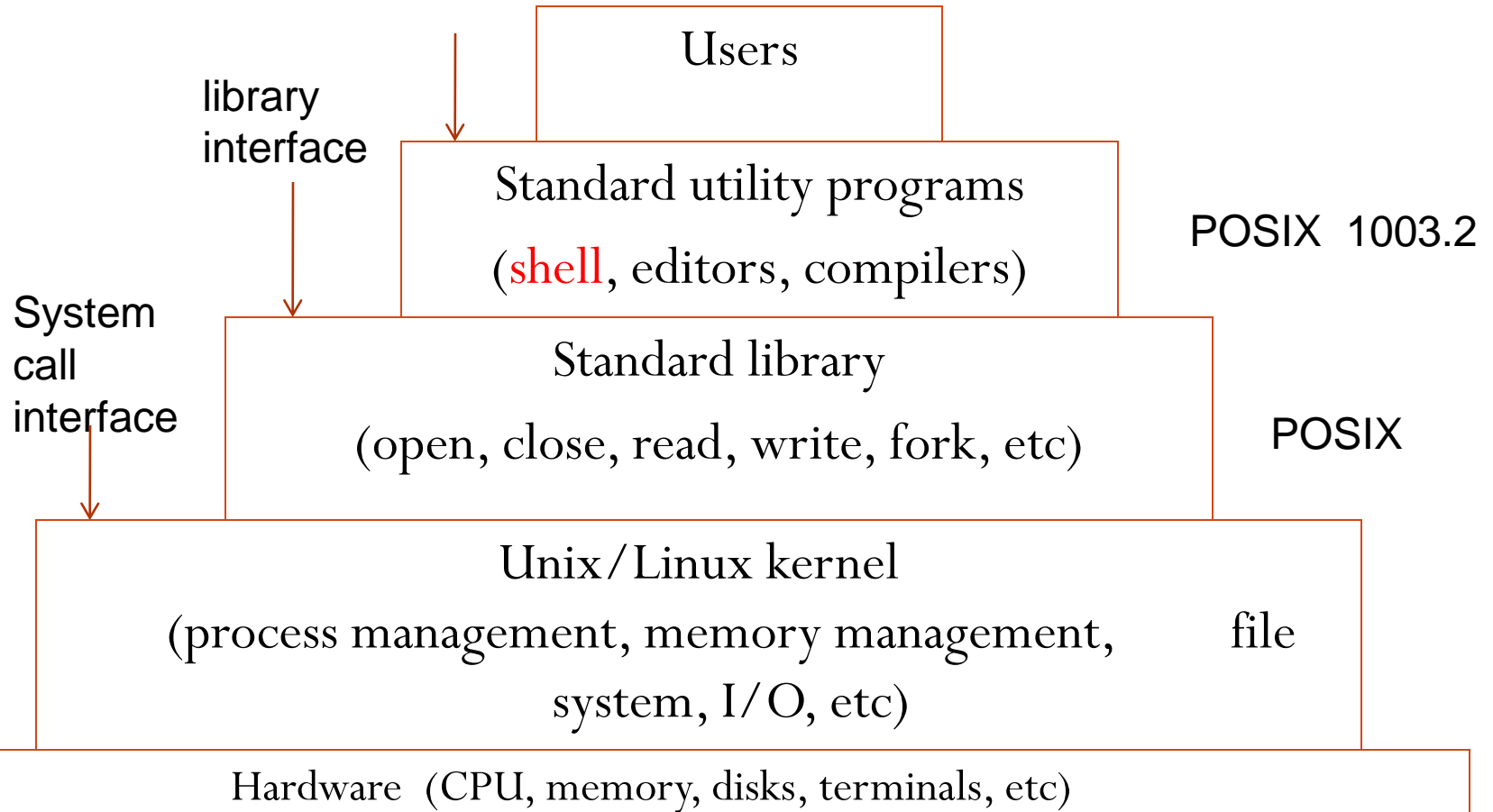
# Kernel Functionalities: Memory management

- Manage **physical memory (RAM) to be** shared among processes in an equitable and efficient fashion
- **Virtual memory management:**
  - Processes are isolated from one another and from the kernel, so that one process can't read or modify the memory of another process or the kernel.
  - Only part of a process needs to be kept in memory, thereby lowering the memory requirements of each process and allowing more processes to be held in RAM simultaneously.
  - better CPU utilization, since it increases the likelihood that, at any moment in time, there is at least one process that the CPU(s) can execute.

# Other OS functionalities ...

- The kernel provides a file system on disk, allowing files to be created, retrieved, updated, deleted, and so on.
- Creation and termination of processes
- Peripheral device: standardizes and simplifies access to devices, arbitrates access by multiple processes to each device
- Networking: transmits and receives network packets on behalf of user processes.
- Support system call interfaces: processes can request the kernel to perform various tasks using kernel entry points known as system calls.
  - Second part of this course: Unix system call API

# Layers in UNIX/Linux System



# Goal of this course

- Learn tools needed for develop application in GNU/Linux
  - A working understanding about UNIX
  - Basic commands, shell scripting
  - GNU tools for app. development
    - compiler, debugger, make, version control, library, testing/profiling tools
  - System calls provided in Unix:
    - to request services from operating system

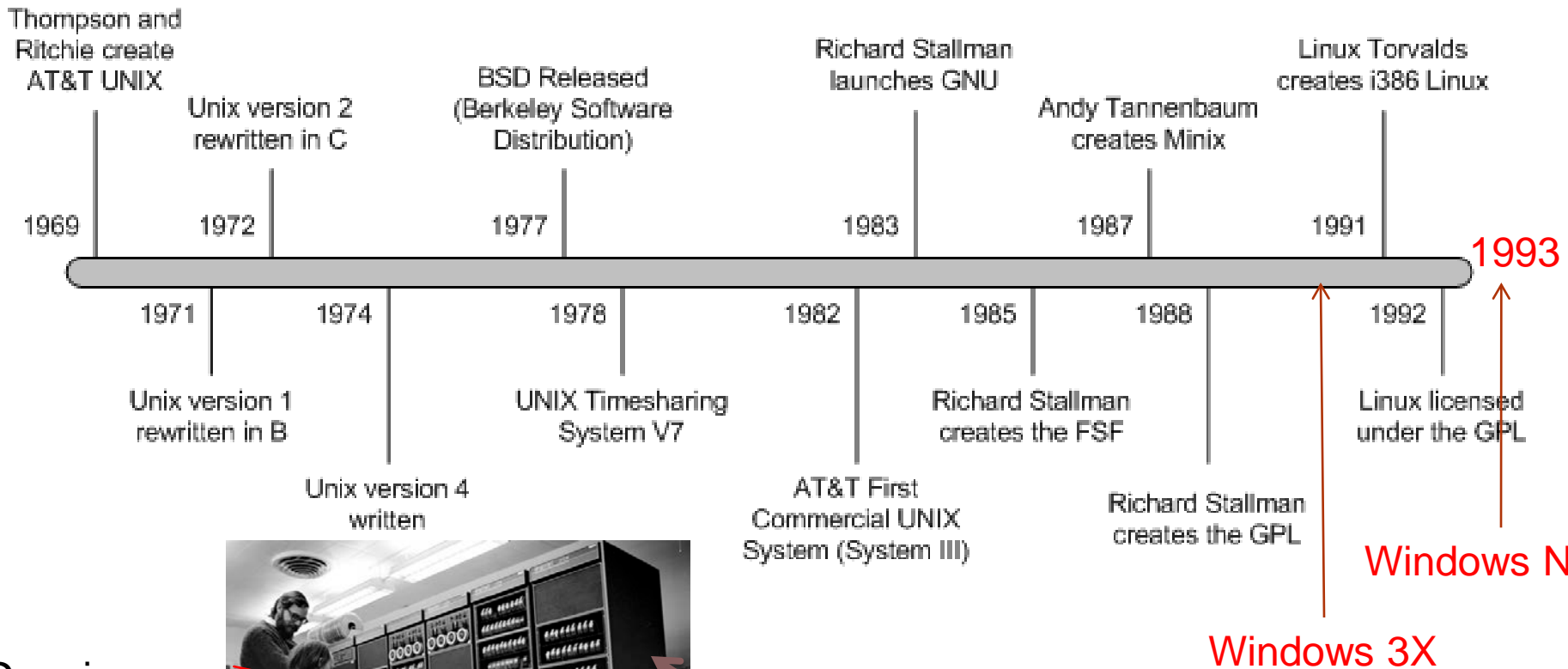
# Roadmap: a top-down approach

- Get started topics
  - Basic concepts & useful commands
- vi, emacs, sed, awk
- Bash programming
- Basic GNU Tools
  - Compiler chain, make, debugger (gdb)
- Unix system calls
- Advanced GNU Tools
  - Library, gcov, gprof, version control tools

Now let's get started with some background information.

---

# Timeline of Unix/Linux, GNU



Dennis Ritchie

Ken Thompson



PDP-11

Windows 3X

Windows NT

# GNU history

- **GNU**: GNU is Not Unix
- **Richard Stallman** (author of Emacs, and many other utilities, ls, cat, ..., on linux)
  - 1983: development of a free UNIX-like operating system
  - Free Software Foundation (100s of Programmers)
- Free software:
  - freedom to run the program, for any purpose.
  - freedom to study how the program works and adapt it to your needs.
  - freedom to redistribute copies so you can help others.
  - freedom to improve the program and release your improvements to the public, so that everyone benefits.





# GPL License

- GNU General Public License is a free, copyleft license for software and other kinds of works...
  - “The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.”
- Manual pages for commands include copyright info:

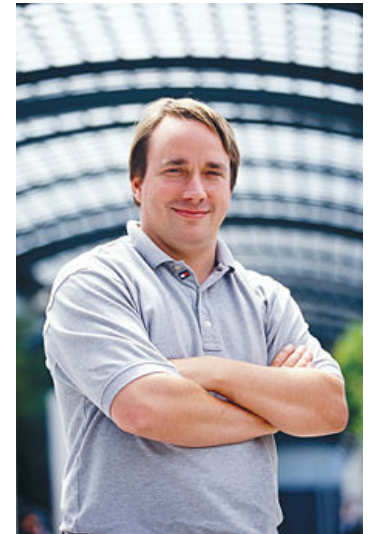
## COPYRIGHT

Copyright © 2011 Free Software Foundation, Inc. License GPLv3+: GNU GPL version 3 or later <<http://gnu.org/licenses/gpl.html>>.

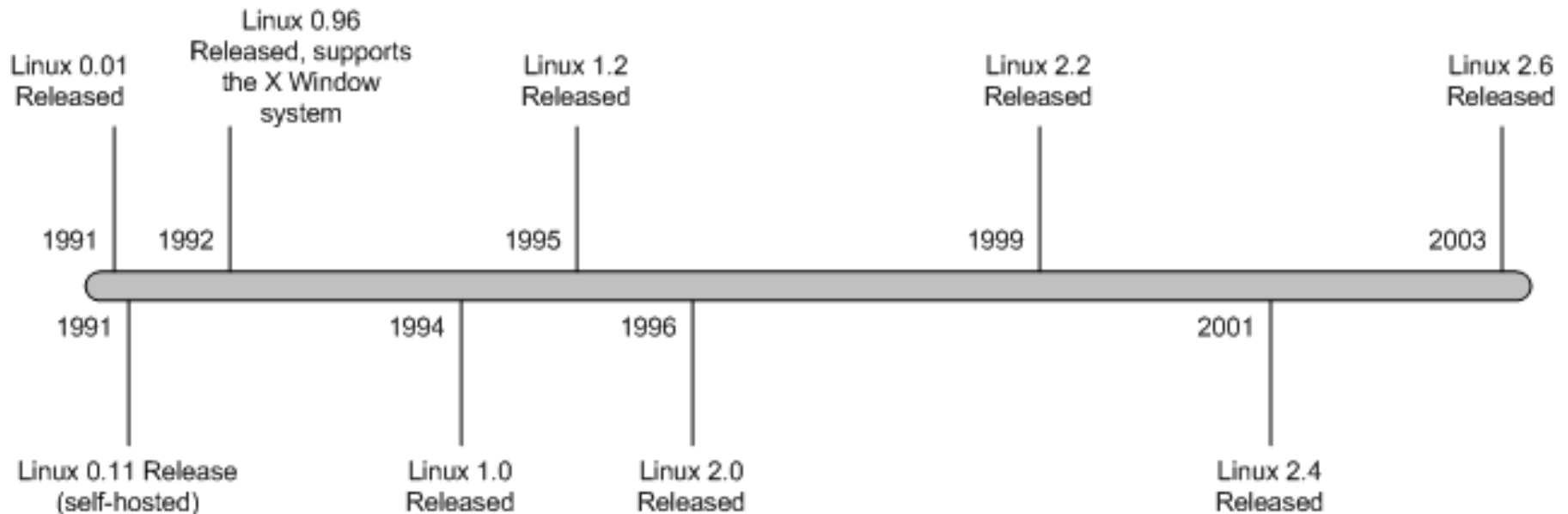
This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

# Linux history

- Linus Torvalds
  - 1991: “hobby” operating system for i386-based computer, while study in Univ. of Helsinki
- 1996: Linux becomes a GNU software component
- GNU/Linux: A fairer name than Linux?
  - “Most operating system distributions based on Linux as kernel are basically modified versions of GNU operating system. We began developing GNU in 1984, years before Linus Torvalds started to write his kernel. Our goal was to develop a complete free operating system. Of course, we did not develop all the parts ourselves—but we led the way. We developed most of the central components, forming the largest single contribution to the whole system. The basic vision was ours too.” --- RMS



# Linux kernel versions



Use “uname -a” to check system information (including kernel version).

# Linux version history

- **1.0**: only supported single-processor **i386**-based computer
- **1.2** support for computers using processors based on the Alpha, SPARC, and MIPS architectures.
- **2.0**: **SMP** support (symmetric multiple processors) and support for more types of processors
- **2.2**: removed global spinlock, improved SMP support, support m68k and PowerPC architectures, new file systems (including read-only support for Microsoft's NTFS)
- **2.4.0**: support for ISA Plug and Play, USB, and PC Cards, PA-RISC processor, Bluetooth, Logical Volume Manager (LVM) version 1, RAID support, InterMezzo and ext3 file systems.

# Linux version history

- **2.6.0** : integration of  $\mu$ Clinux , [PAE](#) support, support for several new lines of [CPUs](#), integration of ALSA , support for up to  $2^{32}$  users , up to  $2^{29}$  process IDs, increased the number of device types and the number of devices of each type, improved 64-bit support, support for file systems of up to 16 terabytes, in-kernel preemption, support for the Native POSIX Thread Library (NPTL), [User-mode Linux](#) integration into the mainline kernel sources, [SELinux](#) integration into the mainline kernel sources, ...
- **3.0** : 21 July 2011. the big change was, "NOTHING. Absolutely nothing." "...let's make sure we really make the next release not just an all new shiny number, but a good kernel too." , released near the 20th anniversary of Linux

# Understanding uname

```
$ uname -a
```

```
Linux storm.cis.fordham.edu 3.6.11-1.fc16.x86_64 #1 SMP Mon Dec 17  
21:29:15 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
```

- **Kernel name:** Linux:
- **Hostname**
- **Kernel release:** 3.6.11-1.fc16.x86\_64
- **Kernel version:** #1 SMP Mon Dec 17 21:29:15 UTC 2012
- **Machine hardware name:** x86\_64 (AMD64 instruction set)
- **Processor:**x86\_64
- **Operating system:** GNU/Linux

# Unix Standardization

- Different implementations of Unix diverged:
  - Different meaning for command options
  - System calls syntax and semantics
- **POSIX**, "**P**ortable **O**perating **S**ystem **I**nterface", is a family of standards specified by **IEEE** for maintaining compatibility between Unix systems.
  - C library level, shell language, system utilities and options, thread library
  - currently IEEE Std. 1003.1-2004
- **POSIX for Windows:**
  - Cygwin provides a largely POSIX-compliant development and run-time environment for Microsoft Windows.

# Single Unix Specification

- **1990:** X/Open launches XPG3 Brand. OSF/1 debuts.
- **1993:** Novell transfers rights to "UNIX" trademark and Single UNIX Specification to **X/Open**.
- **1994:** X/Open introduces **Single UNIX Specification**, separating the UNIX trademark from any actual code stream
- **1995:** X/Open introduces UNIX 95 branding program for implementations of Single UNIX Specification.
- **1996 :** **Open Group** forms as a merger of OSF and X/Open.



# X/Open => Open Group

- **1997:** Open Group introduces Version 2 of Single UNIX Specification, including support for realtime, threads and 64-bit and larger processors.
- **1998:** Open Group introduces UNIX 98 family of brands, including Base, Workstation and Server. First UNIX 98 registered products shipped by Sun, IBM and NCR.
- **1999:** Open Group and IEEE commence joint development of a revision to POSIX and the Single UNIX Specification.
- **2001: Version 3 of the Single UNIX Specification unites IEEE POSIX, The Open Group and the industry efforts.**

# Today's Unix Systems

- To be an officially Unix system, need to go through certification based on the Single Unix Specification
- **Registered Unix systems**: AIX, HP/UX, OS X, Reliant Unix, ....
- **Linux** and **FreeBSD** do not typically certify their distributions, as the cost of certification and the rapidly changing nature of such distributions make the process too expensive to sustain.

# Outline

- Important course information
  - Objective
  - Roadmap
  - Requirement & policy
- Brief history of GNU/Linux
- GNU/Linux Architecture
- Getting started
  - Shell
  - File system, file

# A Quick Start

- Terminal
- PuTTY, a telnet/ssh client
  - a free and open source terminal emulator application
    - a window in your desktop that works like old time terminal
- Enter your ID and password

A screenshot of a terminal window showing a login process on a Linux system. The window title is "james@volcano -". The text displayed is:

```
login as: james
james@shell's password:
Linux volcano 2.6.22-14-server #1 SMP Sun Oct 14 23:34:23 GMT 2007 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

james@volcano:~>
```

# Your first encounter: shell

- **Shell**: a special-purpose program, **command line interpreter**, read commands typed by a user and execute programs in response to entered commands
- Many different shells:
  - **Bourne Shell (sh)**: oldest,
    - I/O redirection, pipelines, filename generation (globbing), variables, environment variables, command substitution, background command execution, function
  - **C Shell (csh)**: syntax of flow-control similar to C, command history, command-line editing, job control, aliases
  - **Korn Shell (ksh)**: “csh”, compatible with sh
  - **Bourne again Shell (bash)**: GNU’s reimplementation of Bourne shell, supports features added in C shell, and Korn shell

# Check/Change Login Shell

- To check the shell you are using
  - `echo $SHELL`
  - `echo $0`
- `login shell`: default shell for a user, specified in `/etc/passwd`
- To change your login shell, use command
  - `chsh`

# Shell: interactive mode

- A shell session (a dialog between user and shell)
  1. Displays a **prompt** character, and waits for user to type in a **command line**
    - Prompt depends on shell: sh, ksh, bash: \$ csh: % tcsh: >
    - May be customized (with current directory, host, ...)
  2. On input of a **command line**, shell extracts **command name and arguments**, searches for the program, and runs it.
  3. When program finishes, shell continues to step 1
  4. The loop continues until user types “exit” or “ctrl-d” to end

# UNIX command line

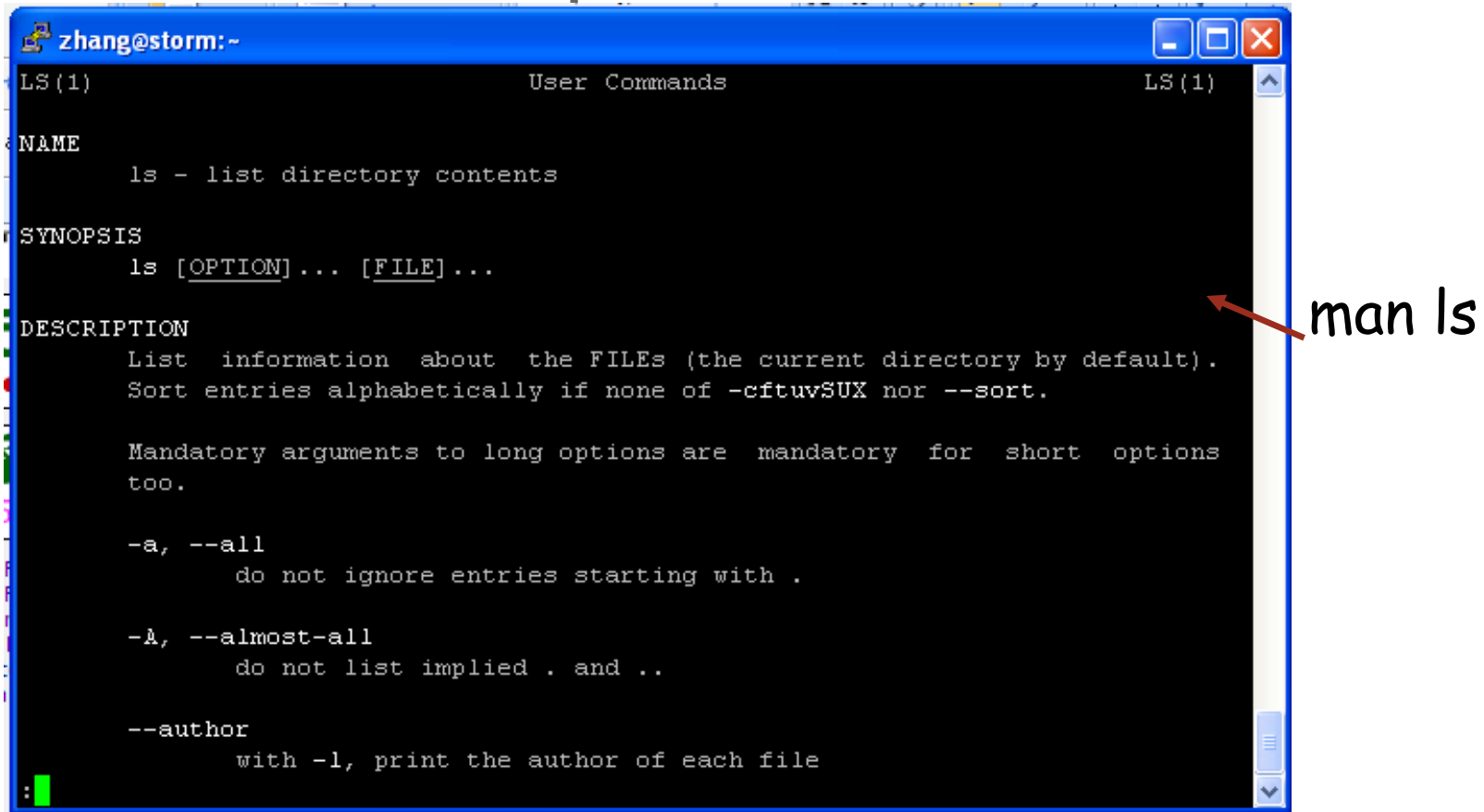
- Command name and arguments:

command [ [ - ] option (s) ] [ option argument (s) ] [ command argument (s) ]

- **Command arguments** are mostly file or directory names
  - `cp prog1.cpp prog1.cpp.bak`
- **Options**: used to control behavior of the command
  - `head -20 lab1.cpp`
  - `wc -w lab2.cpp` // count how many words
  - Some options come with **option argument**
    - `sort -k 1 data.txt`
    - // use the first column of data.txt as the key to sort



# The most important command !!!



```
zhang@storm:~
LS(1)                                User Commands                                LS(1)
NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort.

    Mandatory arguments to long options are mandatory for short options
    too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    --author
        with -l, print the author of each file
```

- man: displaying online manuals
  - Press **q** to quit, **space** to scroll down, **arrow** keys to roll up/down

# Correcting type mistakes

- Shell starts to parse command line only when **Enter** key is pressed
- Delete the whole line (line-kill): C-u
- Erase a character: C-h or backspace key
- Many more fancy functionalities:
  - **Auto-completion**: press **Tab** key to ask shell to auto-complete command, or path name
  - **History (repeat command)**: use arrow (up and down) keys to navigate past commands
  - ...

# Shell: batch/scripting mode

- In batch mode, shell can interpret and execute shell scripts

```
#!/bin/bash
```

```
# count number of files/directories in curr. directory
```

```
ls -l | wc -l
```

- Shell constructs:
  - variables,
  - Loop and conditional statements
  - I/O commands (read from keyboard, write to terminal)
  - Function, arrays ...

# Outline

- Important course information
  - Objective
  - Roadmap
  - Requirement & policy
- Brief history of GNU/Linux
- GNU/Linux Architecture
- Getting started
  - Shell
  - File systems, file,

# Unix File

- Files: store information
  - a sequence of 0 or more bytes containing arbitrary information
- What's in a filename?
  - Case matters, no longer limited to 14 chars
  - Special characters such as -, spaces are allowed, but you shouldn't use them in filename
    - Can you think of the reason ?
  - Dot files are hidden, i.e., normally not listed by command *ls*
    - To display all files, including hidden files, use **`ls -a`**

# What's in a file ?

- So far, we learnt that files are organized in a hierarchical directory structure
  - Each file has a name, resides under a directory, is associated with some admin info (permission, owner)
- Contents of file:
  - Text (ASCII) file (such as your C/C++ source code)
  - Executable file (commands)
  - A link to other files, ...
  - Virtual file:
    - /proc: a pseudo-filesystem, contains user-accessible objects on runtime state of kernel and executing processes
- To check the type of file: “file <filename>”

# File Viewing Commands

- cat: concatenate files and display on standard output (i.e., the terminal window)
  - cat [option] ... [file] ...
  - cat proj1.cpp
  - cat proj1.cpp proj2.cpp
  - cat -n proj1.cpp // display the file with line #
- More: file perusal filter (i.e., displaying file one screen at a time)
  - more proj1.cpp
- head, tail: display the beginning or ending lines of a file
  - **tail -f output // display the file, append more lines as the file grows**

*[ ] means the argument is optional  
... means there can be multiple arguments of this type*

# File manipulation commands

- `rm`: remove one or multiple files or directories
  - `rm [option] ... FILE ...`
  - `rm temp`
  - `rm temp1 temp2`
- **Wildcards (metacharacter)** can be used in command line
  - Letter `*` matches with any string
    - `rm *.o`: remove all `.o` files
  - `?`: match any one character
  - `[abc]`: match with letter a or b or c
- `rm -r`: remove directories and their sub-dirs recursively
- `rm -i` : confirm with user before removing files



# File manipulation commands (2)

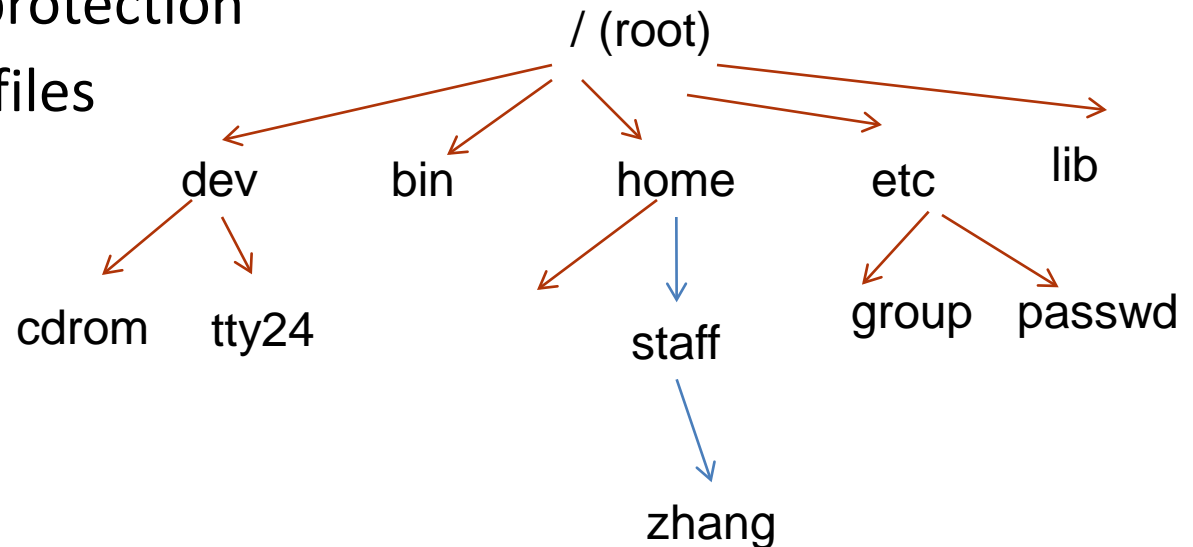
- cp: copy file or directory
  - cp [OPTION] SOURCE DESTINATION
- To make a backup copy of your program before dramatic change
  - `cp proj1.cpp proj1.cpp.bak`
- To make a backup copy of a whole directory
  - `cp -r lab1_dir lab1_dir_backup`
  - `-R, -r, --recursive`: copy directories recursively

# File manipulation commands (3)

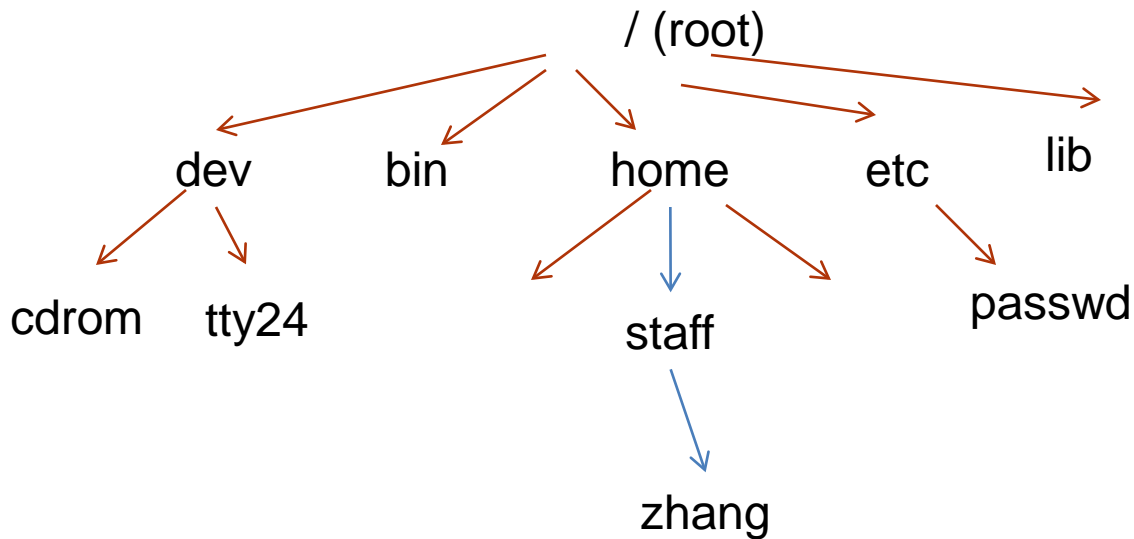
- mv: move (rename) files/directories
  - mv [OPTION] SOURCE DEST
    - Rename SOURCE to DEST
    - mv proj1.cpp lab1.cpp
  - mv [OPTION]... SOURCE... DIRECTORY
    - Move SOURCE to DIRECTORY
    - mv lab1.cpp lab2.cpp CISC3130

# Hierarchical file system

- Directory: a file that can hold other files
- Advantages of hierarchical file system:
  - Files can have same names, as long as they are under different directories
  - Easier for protection
  - Organized files



# Absolute pathname, path



- **Pathname** of a file/directory: location of file/directory in the file system
  - How do you tell other where your prog. is located ?
- **Absolute pathname**: path name specified relative to root, i.e., starting with the root (/)
  - e.g., /home/staff/zhang
  - What's the absolute pathname for the "passwd" file?

# Home directory

- Every user has a **home directory** created for him/her
  - When you log in, you are in your home directory
  - In home directory, a user usually has permission to create files/directories, remove files ..
  - `~` to refer to current user's home directory
  - `~username` to refer to username's home directory

# Current directory & Relative Pathname

- Tiring to specify **absolute pathname** each time
- To make life easier: **working directory**
  - User can move around the file system, shell remembers where the user is (i.e., current directory)
- To check your current directory, use command:
  - **pwd**

# Getting around in the file system

- To create a subdirectory:
  - `mkdir [option]... directory...`
  - `cd`
  - `mkdir CISC3130`
  - `cd CISC3130`
  - `mkdir lab1`
- To remove a directory:
  - `rmdir [option]... directory...`
  - Report failure if directory is not empty
    - Can use `rm -rf` to remove non-empty directory

# Command for change current directory (move around)

- cd [directory]

```
[zhang@storm Work]$ cd
```

```
[zhang@storm ~]$ pwd
```

```
/home/staff/zhang
```

```
[zhang@storm ~]$ cd Work
```

```
[zhang@storm Work]$ pwd
```

```
/home/staff/zhang/Work
```

```
[zhang@storm Work]$ cd ..
```

```
[zhang@storm ~]$ pwd
```

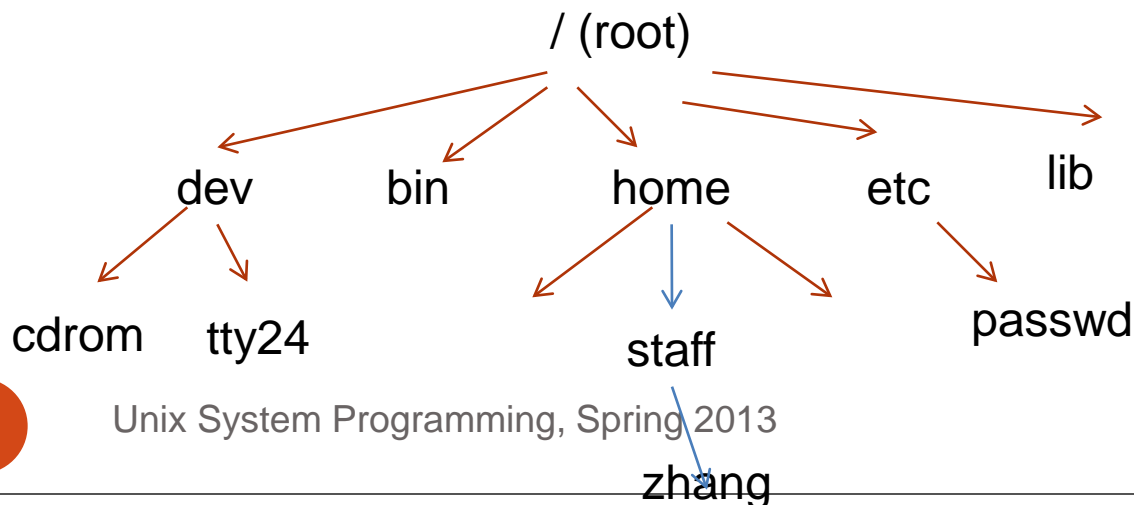
```
/home/staff/zhang
```

```
[zhang@storm ~]$
```



# Relative pathname

- **Absolute pathname:** specified relative to root
- **Relative pathname:** specified relative to current directory
  - . (current directory), .. (parent directory, one level up)
  - If current directory is at /home/staff/zhang, what is the relative pathname of the file passwd?
    - ../../../etc/passwd: go one level up, go one level up, go one level up, go to etc, passwd is there



# Relative pathname

- For all commands that take file/directory name as arguments, you can use pathnames of the file/directory
- Example:
  - `cd /home/staff/zhang/public_html`
  - `pico CISC3130/index.html`
  - `cd ..` (go up one level to parent directory)
  - `cp ../prog2.cpp prog2.cpp`

# Getting around in the file system

- ls: list directory contents
  - ls [OPTION] ... [FILE]

ls: list files/directories under current directory

ls -l: long listing,

```
[zhang@storm CISC1600]$ ls -l
```

```
total 59180
```

```
-rw-r--r-- 1 zhang staff 509952 Sep 7 13:02 3_types.ppt
```

```
-rw-r--r-- 1 zhang staff 593408 Sep 14 23:38 4_computation.ppt
```

```
-rw-r--r-- 1 zhang staff 1297 Sep 2 12:18 account.html
```

```
-rw-r--r-- 1 zhang staff 3304448 Nov 7 18:24 ArrayVector1.ppt
```

```
drwxr-xr-x 2 zhang staff 4096 Dec 8 22:36 Codes
```

# Long listing

- To get more information about each file

```
[zhang@storm Demo]$ ls -al
```

```
total 32
```

Total disc space taken in blocks (1024 Byte)

```
drwxr-xr-x  5 zhang staff 4096 2008-01-16 16:01 .
```

```
drwxr-xr-x 41 zhang staff 4096 2008-01-16 16:01 ..
```

```
drwxr-xr-x  2 zhang staff 4096 2008-01-16 15:55 CCodes
```

```
-rw-r--r--  1 zhang staff   38 2008-01-16 16:01 .HiddenFile
```

```
-rw-r--r--  1 zhang staff   53 2008-01-16 15:57 README
```

```
drwxr-xr-x  2 zhang staff 4096 2008-01-16 15:55 SampleCodes
```

```
drwxr-xr-x  4 zhang staff 4096 2008-01-16 15:56 ShellScriptes
```



d means directory  
Who has permission to read/write the file

User name of the owner and its group

# Long listing explained

`drwxr-xr-x 4 zhang staff 4096 2008-01-16 15:56 ShellScripts`

- field 1
  - 1st Character: specifies the type of the file.
    - - normal file, d directory, s socket file, l link file
  - next 9 characters – File Permissions
- field 2: specifies the number of links for that file
- field 3 : specifies owner of the file
- field 4: specifies the group of the file
- field 5 : specifies the size of file.
- field 6: date and time of last modification of the file
- field 7: File name

# File permissions

```
drwxr-xr-x 4 zhang staff 4096 2008-01-16 15:56 ShellScripts
```

- Each file is associated with permission info.
  - Differentiate **three type of users**: **owner user**, **users from same group as owner**, **others**
  - **Three type of access**: **- in the field means no permission**
    - **Read (r)**: use “cat” to open a file to read, use “ls” to list files/directories under a directory
    - **Write (w)**: modify the contents of the file, create/remove files from the directory
    - **Execute (x)**: execute the file, or “cd” or “search” the directory for file
- Trying to list other’s directory

```
[zhang@storm ~]$ ls ../roche/
```

```
ls: cannot open directory ../roche/: Permission denied
```

# User and Group

- Each user has unique login name (user name), and corresponding numeric ID
  - Group id, home directory, login shell:
  - Stored in file `/etc/passwd`
- Groups: each user can belong to multiple groups
  - For collaboration ...
  - Group name, group id, user list info stored in `/etc/group`
- Superuser
  - ID: 0, username: root
  - Bypass all permission checks ...

# More to play with

- who: show who is logged on
  - who am I
- write: write to another user's terminal (IMS?)
- which: show the full path name of a command
  - \$ which bash
  - /bin/bash
- How does shell find a command ?
  - Environment variable PATH stores a list of paths to search for programs: “set | grep PATH” or “echo \$PATH”, “set” to show all variable settings
    - PATH=\$PATH:\$HOME/bin:.
  - Built-in commands: history, set, echo, etc.
- mail: send email from command line
  - mail -s “graded project” zhang < proj1.cpp



# Summary

- Unix, a time-sharing operating system
- Operating system
  - a program that sits between hardware and user
  - Manages resources (CPU, memory, disk, network connection) and present user interface to user
- Unix programming environment, i.e., platform
  - the layered view

# Summary (cont'd)

- PuTTY emulates a terminal
- Shell: a command line interface
  - Engage in dialog (a **session** with user)
- What's in a command
  - Name, and arguments
- Hierarchical file system: file and directory
- Commands for working with file system