



# Linux - introduction

Mag Selwa

ICTS Leuven



3 October 2019

# Overview

- Introduction – Linux philosophy
- Command line basics – getting help
- Navigating the file system
- The shell revisited: some features
- File manipulation
- Text editing
- Groups, users, security
- Process control

# Remark

- Commands will be displayed with different font and will start always with prompt sign (\$) – which is already included in the command line

# Introduction



# Purpose

- Get a grip on the very basics: an introduction to the (basic) Linux commands as normally used for computational research (e.g. HPC cluster)
- Focus on using the command line
- Short hands-on examples

# Information

- Most of the presentation is based on the information found in:  
<http://dontfearthecommandline.blogspot.com/>

Nicholas Marsh

## **Introduction to the Command Line (Second Edition)**

- <http://linuxcommand.org/tlcl.php>

William Shotts

## **The Linux Command Line**

- <http://www.oreilly.com/programming/free/ten-steps-to-linux-survival.csp?download=yes&order=1274753>

James Lehmer

## **Ten Steps to Linux Survival**

- [https://bootlin.com/doc/legacy/command-line/unix\\_linux\\_introduction.pdf](https://bootlin.com/doc/legacy/command-line/unix_linux_introduction.pdf)

Michael Opdenacker & Thomas Petazzoni

## **The Unix and GNU/Linux command line**

# Websites?

- <http://www.howtogeek.com/tag/linux/>
  - Includes help, tutorials, tips and how-to guides for Linux.
- <http://www.tldp.org>
  - The Linux Documentation Project
- <http://www.lwn.net>
  - Linux Weekly News: Covering the Linux and free software communities since 1998.
- <http://www.linuxjournal.com>
  - The monthly magazine of the Linux community, promoting the use of Linux worldwide.

# Try it?

- Install virtual machine software
  - Vmware player
    - <http://www.vmware.com/products/player/>
  - Virtualbox
    - <https://www.virtualbox.org/>
- Install a linux distribution
  - Ubuntu <http://www.ubuntu.com/>
  - CentOS <https://www.centos.org/>
  - Opensuse <http://www.opensuse.org/>
  - ...



# Knoppix can be a good solution

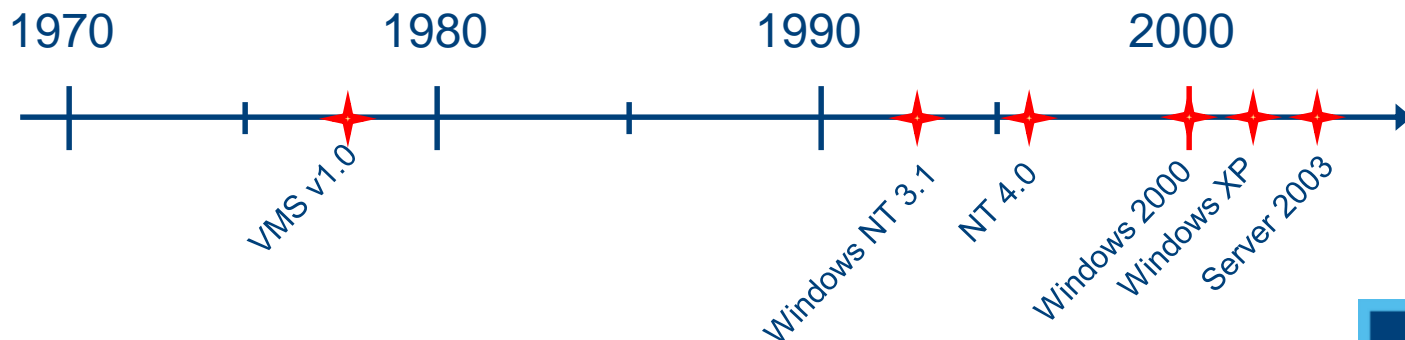
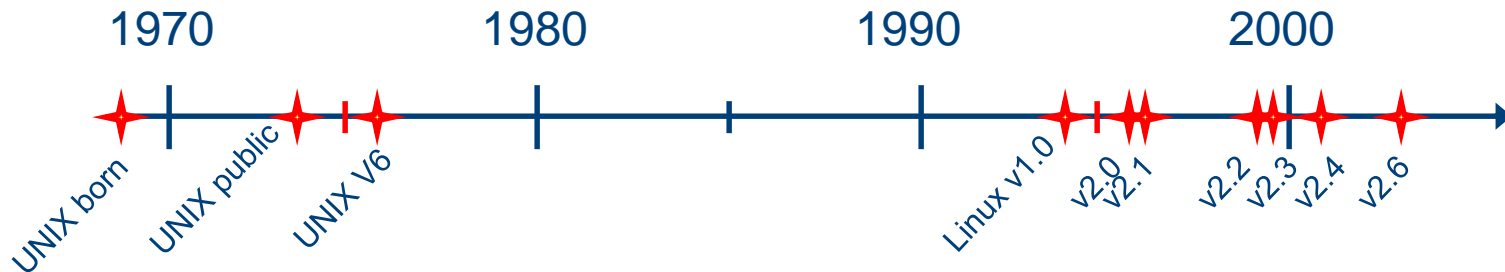
You **do not need to install** Knoppix on harddisk

- So it can be used in Demo of linux or software on Linux,
- So you need extra Linux machine lab in a few minutes,
- No extra space on harddisk on old PC's, just use Knoppix,
- Got a new laptop, just boot Linux on it.



# Windows and Linux

- Both Linux and Windows are based on foundations developed in the mid-1970s
- Both DOS, MAC and UNIX are proprietary, i.e., the source code of their kernel is protected
- No modification is possible without paying high license fees



# Some history

## GNU project:

- Established in 1984 by Richard Stallman (goal: software should be free from restrictions against copying or modification in order to make better and efficient computer programs),
- GNU is a recursive acronym for “**G**NU's **N**ot **U**nix”,
- Aim at developing a complete Unix-like operating system which is free for copying and modification,
- Companies make their money by maintaining and distributing the software, e.g. optimally packaging the software with different tools,
- Stallman built the first free GNU C Compiler in 1991. But still, an OS was yet to be developed

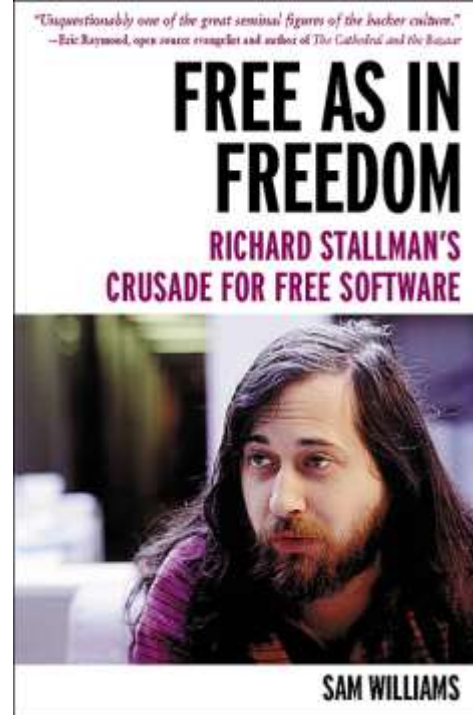


# Open Source Software

- **Open Source Software (OSS)** generally refers to software for which the source code is available and which the licensing scheme permits the user to modify it and redistribute it in modified or unmodified form.

## GNU copyleft ([www.gnu.org](http://www.gnu.org))

- Nobody should be restricted by the software they use. There are four freedoms that every user should have:
- the freedom to use the software for any purpose,
- the freedom to change the software to suit your needs,
- the freedom to share the software with your friends and neighbors, and
- the freedom to share the changes you make.
- When a program offers users all of these freedoms, we call it free software.



# Linux: some history

- UNIX: roots in Bell Labs (AT&T)
- 1985 Free Software Foundation (FSF) founded by Richard Stallman. Along with other programmers creates the tools needed to make a UNIX compatible OS
- 1985 Professor Andy Tannenbaum creates a UNIX like operating system based on System V Unix for the IBM PC & PC/AT computers. It is called Minix.
- 1989 Richard Stallman releases GPL and GNU software but lacks a free kernel.
- 1991 Building on the concepts in Minix, Linus Torvalds (Finnish college student) develops Linux along with help from other users on the web.



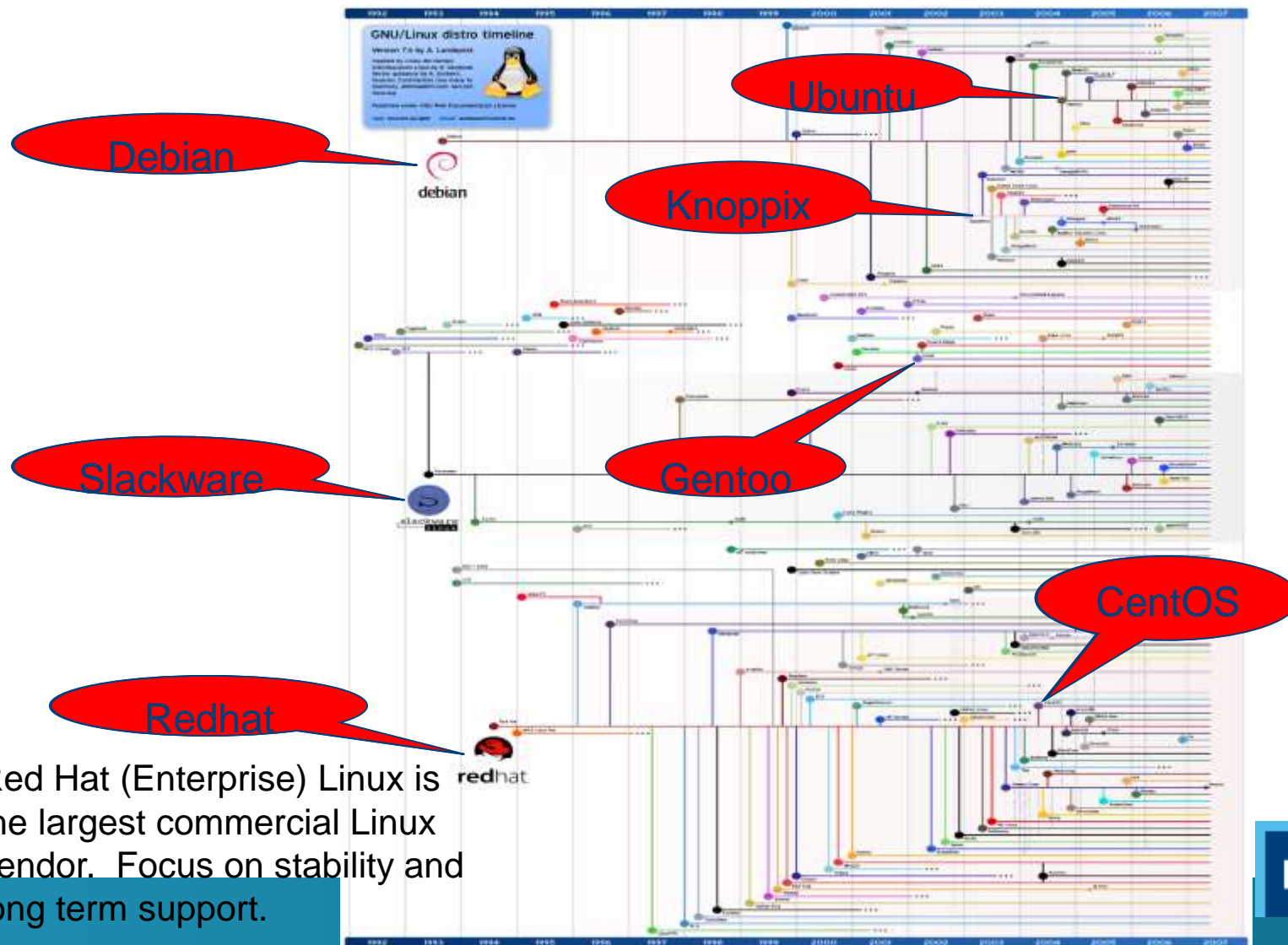
# Operating system?

- Strictly speaking Linux refers to the kernel
- GNU/Linux more accurately describes the Operating System. Linux Kernel combined with GNU utilities and libraries
- Distribution – GNU/Linux bundled with other applications. Examples Red Hat Linux, Debian, Ubuntu, Suse, Knoppix, etc.
- Distributions can be compiled and maintained by an individual or corporation. Can be small (single floppy disk) or span several CD/DVDs.

[www.distrowatch.com](http://www.distrowatch.com) for more information

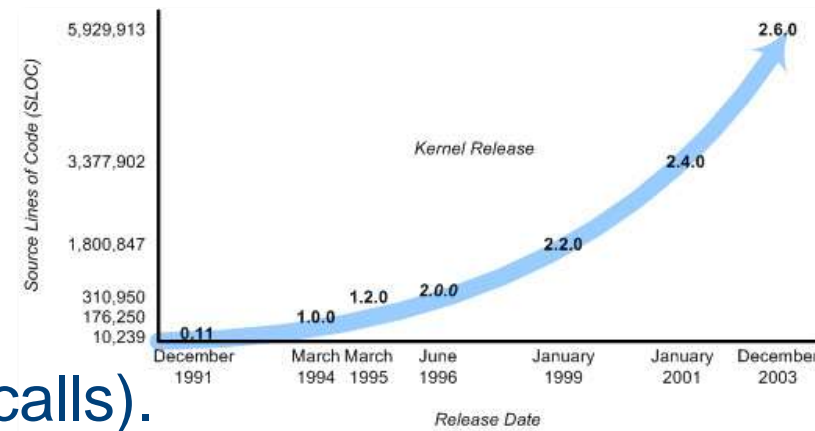
# Which Linux Distribution is better?

Source: <http://futurist.se/gldt/>



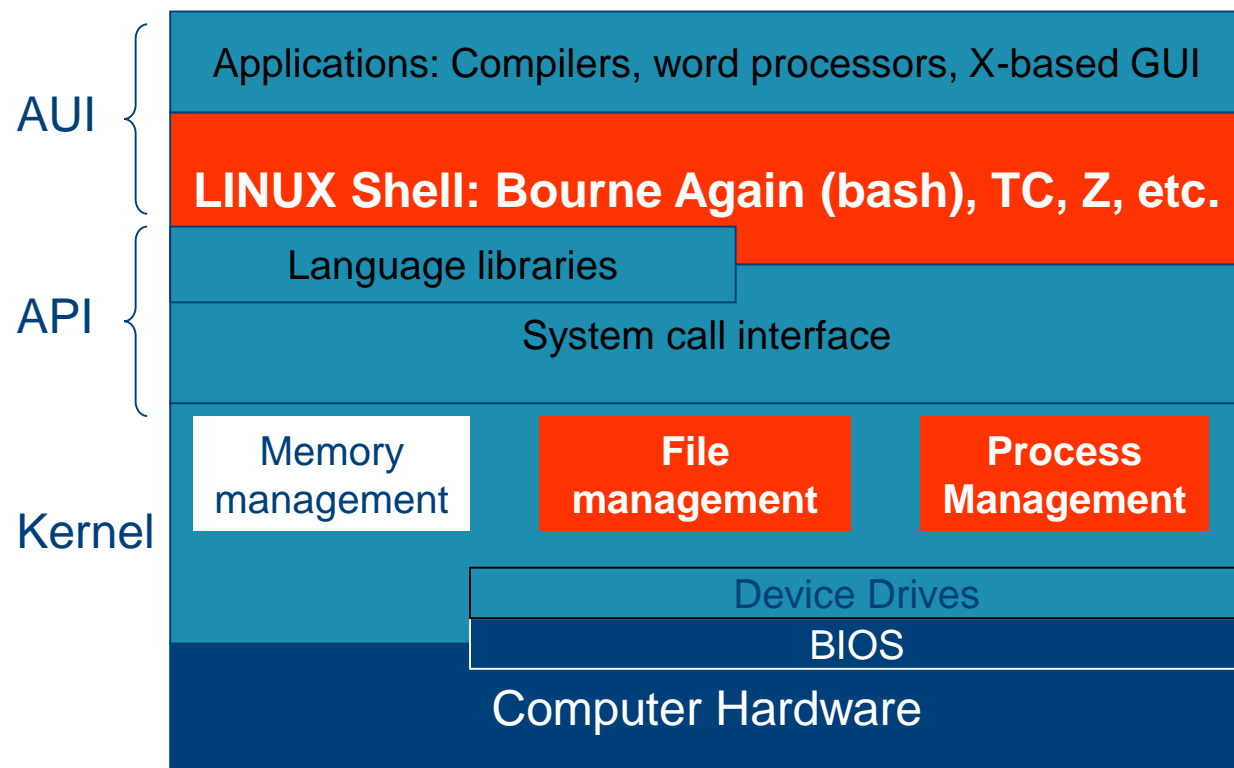
# Linux Kernel

- aka: executive, system monitor.
- The kernel provides a layer between the computer hardware and user applications.
- Provides an interface for software to use hardware (no direct access)
- Planning and assigning:
  - memory, CPU, disk, etc.
  - security aspects
  - Fulfill user requests (system calls).
  - Filesystem, networking, ...





# Linux OS



- Kernel
  - The part of an OS where the real work is done
- System call interface
  - Comprises a set of functions (often known as Application Programmer's Interface API) that can be used by the applications and library routines to use the services provided by the kernel
- Application User's Interface
  - Interface between the kernel and user
  - Allows user to make commands to the system
  - Divided into text based and graphical based

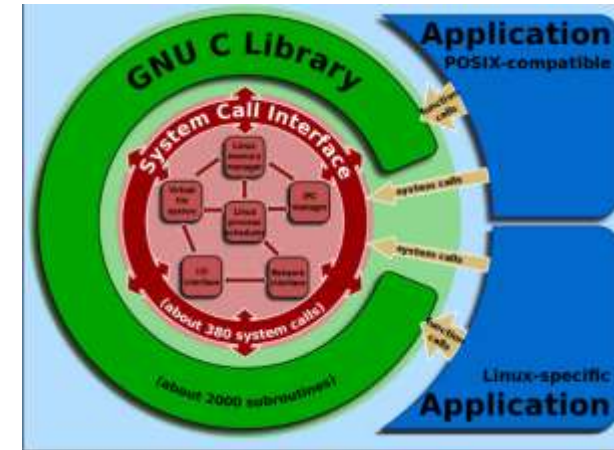
# Linux Kernel

- File Management
  - Controls the creation, removal of files and provide directory maintenance
  - For a multiuser system, every user should have its own right to access files and directories
- Process Management
  - For a multitask system, multiple programs can be executed simultaneously in the system
  - When a program starts to execute, it becomes a process
  - The same program executing at two different times will become two different processes
  - Kernel manages processes in terms of creating, suspending, and terminating them
  - A process is protected from other processes and can communicate with the others

# Linux Kernel

- Memory management
  - Memory in a computer is divided into main memory (RAM) and secondary storage (usually refer to hard disk)
  - Memory is small in capacity but fast in speed, and hard disk is vice versa
  - Data that are not currently used should be saved to hard disk first, while data that are urgently needed should be retrieved and stored in RAM
- Device drivers
  - Interfaces between the kernel and the BIOS
  - Different device has different driver

# Compatibility Statement



- **Definition of the OS version**

- Kernel version (i.e. kernel 2.2.12)
- glibc version (i.e. glibc 2.1.2)

- **Determining the OS Version**

- Kernel version (`$ uname -r -> 2.2.14`)
- System distribution (`$ uname -a`)
- glibc version (`$ ls -l /lib(64)/libc.so* -> libc-2.1.2.so` (Red Hat))
- glibc version (`$ ls -l /lib/i386-linux-gnu/libc.so* -> libc-2.1.2.so` (Debian 32-bit))
- glibc version (`$ ls -l /lib/x86_64-linux-gnu/libc.so* -> libc-2.1.2.so` (Debian 64-bit))
- `$ rpm -qa | grep libc -> glibc-2.1.2-11`
- `$ ldd -version`

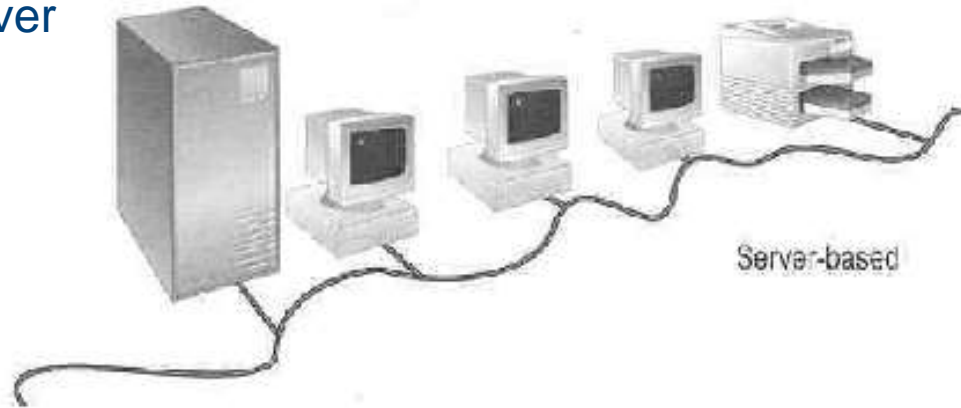
- **Determining the OS Flavor/release**

- `$ cat /etc/*-release`

# User login

- Linux is a multiuser OS ,
- Allows multiple users to use the resource of a computer at the same time
- Every user needs to login the system with the password provided to identify their right in using the resource
- Requires for both client-server based system or desktop

Linux  
Server



# Linux User Interface

Traditional Linux (Unix also) uses command-driven interface (or text-based interface)

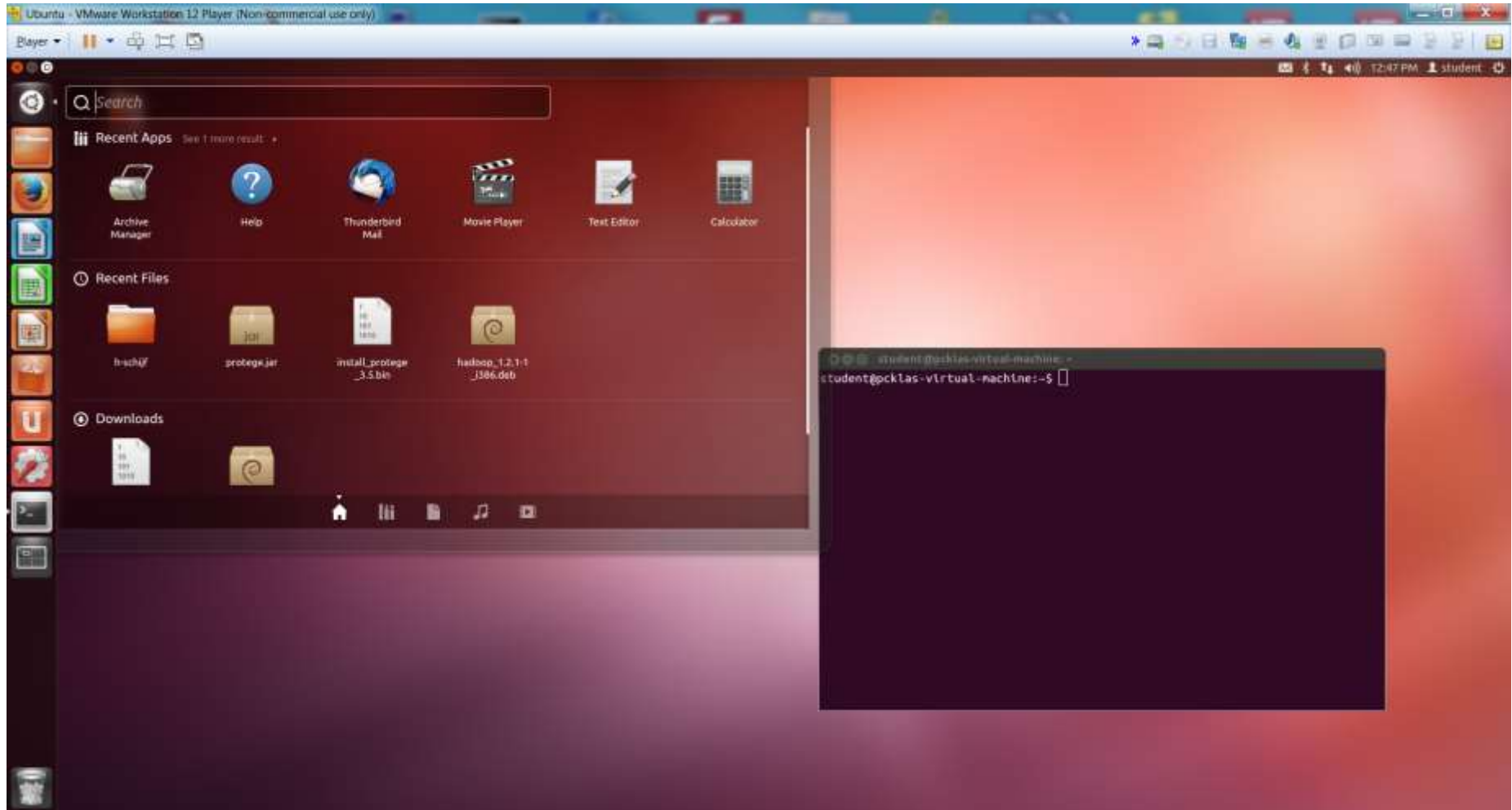
- User needs to type lines of command to instruct the computer to work, similar to DOS
- Advantage: fast in speed. Very few resource is required for its implementation
- Disadvantages: user needs to type, hence can easily make error. Besides, user needs to memorize all commands
- Suitable for expert users and for the systems that interaction with user is not frequent, such as servers

# Linux User Interface

By adopting the X-Window technology, graphical user interface (GUI) is available for Linux:

- Uses pointing devices (e.g. mouse) to control the system, similar to Microsoft's Windows
- Provide menu-driven and/or icon-driven interfaces
  - menu-driven: user is provided with a menu of choices. Each choice refers to a particular task
  - icon-driven: tasks are represented by pictures (icon) and shown to user. Click on an icon invokes one task
- Advantages: No need to memorize commands. Always select task from menus or icons
- Disadvantages: Slow and require certain resource for its implementation
- Suitable for general users and systems, such as PC

# Linux User Interface



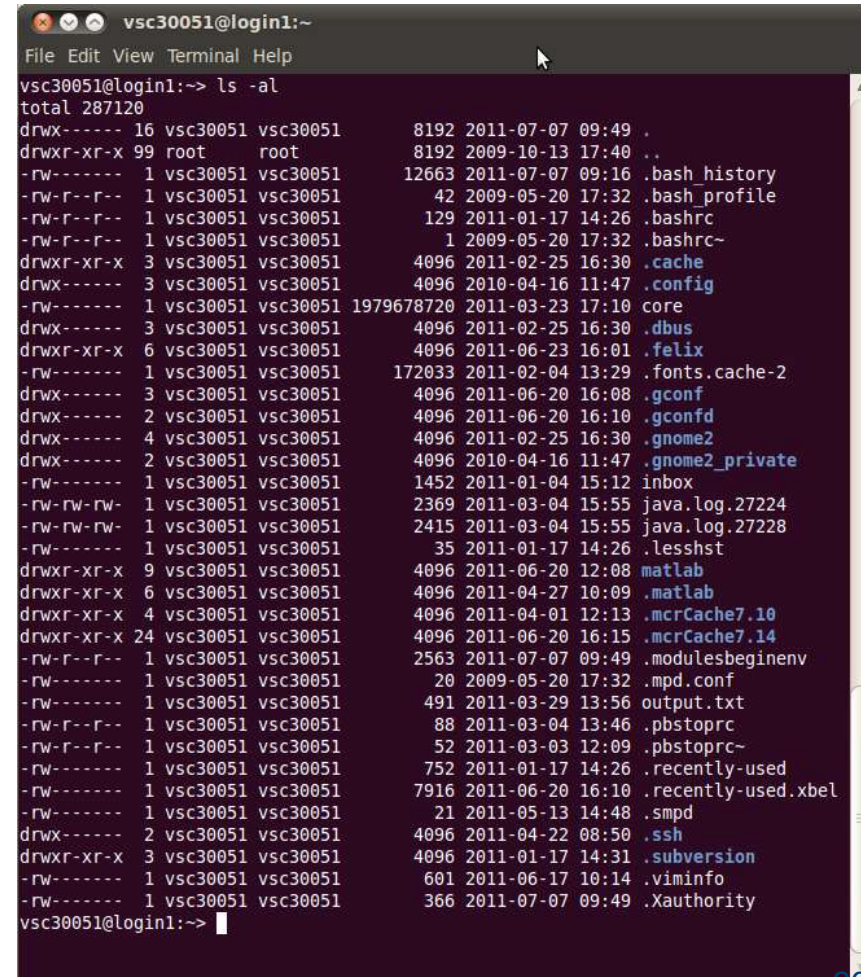


The screenshot shows a Linux desktop environment. On the left is a sidebar with 'Places' (Applications, Accessories, Documentation, Graphics, Internet, Office, Programming, Sound & Video, Sundry, System Tools, Utilities) and 'Activities Overview'. The main window area shows a file manager with a sidebar listing 'Home', 'Documents', 'Downloads', 'Music', 'Pictures', 'Public', 'Templates', 'Videos', 'Devices', and 'Network'. A terminal window is open in the foreground, displaying a list of installed packages for the 'lib64' architecture. The packages listed include:

- libc6-paranoid-10.2+8.90-11.e17.x86\_64
- libc6-8.3.1-18.e17.x86\_64
- libc6nfig-1.4.9-5.e17.x86\_64
- libc6roco-0.6.8-5.e17.x86\_64
- libc6lo-0.52-1.e17.x86\_64
- libc6hampain-0.12.4-4.e17.x86\_64
- libc6om\_err-1.42.9-4.e17.x86\_64
- libc6apng-0.7.3-5.e17.x86\_64
- libc6anberra-gtk2-0.39-5.e17.x86\_64
- libc6anberra-0.39-5.e17.x86\_64
- libc6hampain-gtk-0.12.4-4.e17.x86\_64
- libc6acard-1.5.3-60.e17.x86\_64
- glibc-2.17-55.e17.x86\_64
- libc6url-7.29.0-19.e17.x86\_64
- glibc-headers-2.17-55.e17.x86\_64
- libc6dr-0.8.14-3.e17.x86\_64
- libc6ap-2.22-0.e17.x86\_64
- libc6hewing-0.3.4-6.e17.x86\_64
- [mag@localhost lib64]\$ rpm -qa | grep glibc
- glibc-common-2.17-55.e17.x86\_64
- glibc-devel-2.17-55.e17.x86\_64
- glibc-2.17-55.e17.x86\_64
- glibc-headers-2.17-55.e17.x86\_64
- [mag@localhost lib64]\$

# Command Line

- Text mode
- Provides a way to control the computer via the keyboard.
- Is common use on HPC
- Is somewhat archaic, but very powerful.
- Can be much quicker than a GUI.
- Implicitly assumes that you know what your are doing
- Don't be scared!
- If anything goes wrong, you can stop the command with **Ctrl+C**



```
vsc30051@login1:~  
File Edit View Terminal Help  
vsc30051@login1:~> ls -al  
total 287120  
drwx----- 16 vsc30051 vsc30051      8192 2011-07-07 09:49 .  
drwxr-xr-x 99 root      root      8192 2009-10-13 17:40 ..  
-rw----- 1 vsc30051 vsc30051    12663 2011-07-07 09:16 .bash_history  
-rw-r--r-- 1 vsc30051 vsc30051      42 2009-05-20 17:32 .bash_profile  
-rw-r--r-- 1 vsc30051 vsc30051     129 2011-01-17 14:26 .bashrc  
-rw-r--r-- 1 vsc30051 vsc30051      1 2009-05-20 17:32 .bashrc~  
drwxr-xr-x 3 vsc30051 vsc30051     4096 2011-02-25 16:30 .cache  
drwx----- 3 vsc30051 vsc30051     4096 2010-04-16 11:47 .config  
-rw----- 1 vsc30051 vsc30051 1979678720 2011-03-23 17:10 core  
drwx----- 3 vsc30051 vsc30051     4096 2011-02-25 16:30 .dbus  
drwxr-xr-x 6 vsc30051 vsc30051     4096 2011-06-23 16:01 .felix  
-rw----- 1 vsc30051 vsc30051 172033 2011-02-04 13:29 .fonts.cache-2  
drwx----- 3 vsc30051 vsc30051     4096 2011-06-20 16:08 .gconf  
drwx----- 2 vsc30051 vsc30051     4096 2011-06-20 16:10 .gconfd  
drwx----- 4 vsc30051 vsc30051     4096 2011-02-25 16:30 .gnome2  
drwx----- 2 vsc30051 vsc30051     4096 2010-04-16 11:47 .gnome2_private  
-rw----- 1 vsc30051 vsc30051    1452 2011-01-04 15:12 inbox  
-rw-rw-rw- 1 vsc30051 vsc30051   2369 2011-03-04 15:55 java.log.27224  
-rw-rw-rw- 1 vsc30051 vsc30051   2415 2011-03-04 15:55 java.log.27228  
-rw----- 1 vsc30051 vsc30051      35 2011-01-17 14:26 .lessht  
drwxr-xr-x 9 vsc30051 vsc30051     4096 2011-06-20 12:08 matlab  
drwxr-xr-x 6 vsc30051 vsc30051     4096 2011-04-27 10:09 .matlab  
drwxr-xr-x 4 vsc30051 vsc30051     4096 2011-04-01 12:13 .mcrCache7.10  
drwxr-xr-x 24 vsc30051 vsc30051     4096 2011-06-20 16:15 .mcrCache7.14  
-rw-r--r-- 1 vsc30051 vsc30051   2563 2011-07-07 09:49 .modulesbeginenv  
-rw----- 1 vsc30051 vsc30051      20 2009-05-20 17:32 .mpd.conf  
-rw----- 1 vsc30051 vsc30051     491 2011-03-29 13:56 output.txt  
-rw-r--r-- 1 vsc30051 vsc30051      88 2011-03-04 13:46 .pbstoprc  
-rw-r--r-- 1 vsc30051 vsc30051      52 2011-03-03 12:09 .pbstoprc~  
-rw----- 1 vsc30051 vsc30051     752 2011-01-17 14:26 .recently-used  
-rw----- 1 vsc30051 vsc30051   7916 2011-06-20 16:10 .recently-used.xbel  
-rw----- 1 vsc30051 vsc30051      21 2011-05-13 14:48 .smpd  
drwx----- 2 vsc30051 vsc30051     4096 2011-04-22 08:50 .ssh  
drwxr-xr-x 3 vsc30051 vsc30051     4096 2011-01-17 14:31 .subversion  
-rw----- 1 vsc30051 vsc30051     601 2011-06-17 10:14 .viminfo  
-rw----- 1 vsc30051 vsc30051     366 2011-07-07 09:49 .Xauthority  
vsc30051@login1:~> |
```

# Linux text based interface: The shell

- Is the command line interpreter: a program that accepts input from a user (e.g. a command) and performs the requested task.
- The shell's prompt identifies the type of shell being used. There are two basic types of shell prompts:
  - **\$** Normal user shell (may also be **%** or **>** on some systems)
  - **#** Root user shell (login as root or `$ su`)  
*!!! Warning: The root user can do anything on Linux systems. It is important to know when you are working as root to prevent accidental damage to your system.*
  - The way to identify the user: `$ whoami`

# Using the root account

- If you have the root password:  
`$ su - (switch user)`
- In modern distributions, the `sudo` command gives you access to some root privileges with your own user password.  
Example: `$ sudo mount /dev/hda4 /home`
- root user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting networking, changing file ownership, package upgrades...
- Check the prompt **#**

# Most popular shells

- There are several types of shells for Linux.
- Check it with  

```
$ echo $SHELL
```

Shell	Prompt	Name	Note
sh	\$	Bourne Shell	Default on some Unix systems
bash	\$	Bourne Again Shell	Enhanced replacement for the Bourne shell Default on most Linux and Mac OS X systems
csh	%	C Shell	Default on many BSD systems
tcsh	>	TC Shell	Enhanced replacement for the C shell
ksh	\$	Korn Shell	Default on AIX systems

# Hands-on 1



# Vmware in pc-klas



# Command line basics

## Getting Help





# Important rules on the command line

1. Linux systems are case (and space) sensitive.
  - `MyFile` is not same as `myfile`
2. There is no "recycle bin" or "trash can" when working in the command line environment. There might be one for GUI.

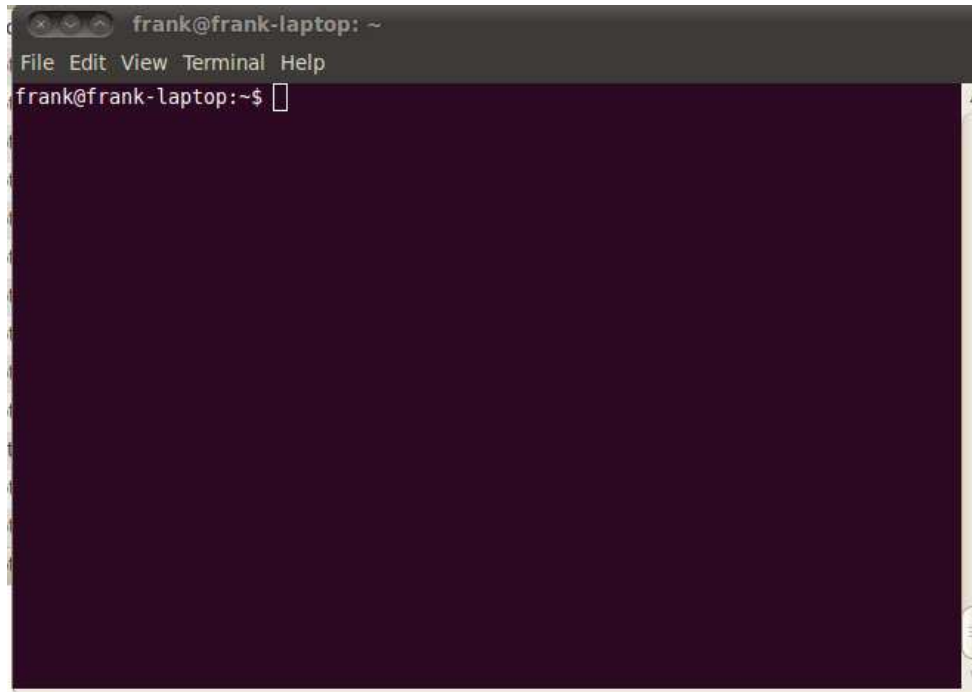
When files are deleted on the command line, they instantly disappear forever.
3. You should always practice new commands on a test system that is not used in a production environment. This minimizes the chances of an accident that can take down an important system

# Important rules on the command line

- “\” vs. “/”:
  - In Linux, the “/” is the directory separator, and the “\” is an escape character.
  - In Windows, the forward-slash “/” is the command argument delimiter, while the backslash “\” is a directory separator
- Filenames:
  - In Linux, there is no such thing as a file extension. Periods can be placed at any part of the filename, and “extensions” may be interpreted differently by all programs, or not at all.
  - Windows uses the “.extension” filename convention, (e.g. FILENAME.TXT).

# Getting help: command built-in

- Help on most Linux commands is typically built into the command themselves
- These flags usually look like “-h” or “--help”.
- `$ ls --help`

A screenshot of a terminal window titled 'frank@frank-laptop: ~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The prompt 'frank@frank-laptop:~\$' is visible, followed by a cursor. The terminal background is dark purple.

```
frank@frank-laptop: ~  
File Edit View Terminal Help  
frank@frank-laptop:~$
```

# Getting help: man pages

- Best source of information can be found in the online manual pages, “**man pages**” for short.

type “man command”.

```
$ man grep
```

- Tips:
  - To search for a particular word (e.g. file) within a man page, type “/word”.
  - To quit from a man page, type the “q” key.
  - If you do not remember the name of Linux command and you know a keyword relating to the command, search the man pages with the -k

```
$ man -k control
```

# Getting help: info pages

- Info pages are similar to man page, but instead of being displayed on one long scrolling screen, they are presented in shorter segments with links to other pieces of information.
- Access with the “`info`” command  

```
$ info ls
```
- Tips:
  - To quit from a info page, type the “`q`” key.
  - Type “`h`” to get more help on the info

# Getting help

- bash has a built-in help facility available for each of the shell *builtins*.  
\$ help cd  
\$ help pwd
- `whatis` displays a very brief description of a command  
\$ `whatis` pwd

# Useful commands

- Clear the contents of the current screen

```
$ clear
```

- `$ logout`

- The `logout` command logs your account out of the system (in a text mode).
- This will end your terminal session and return to the login screen.
- Some systems may have a file called `.logout` or `.bash_logout` in each user's home directory.

- `$ exit`

- Exit the current shell. The `exit` command is similar to the `logout` command with the exception that it does not run the `logout` script located in the user's home directory.

# Linux filesystem

## Navigating the filesystem

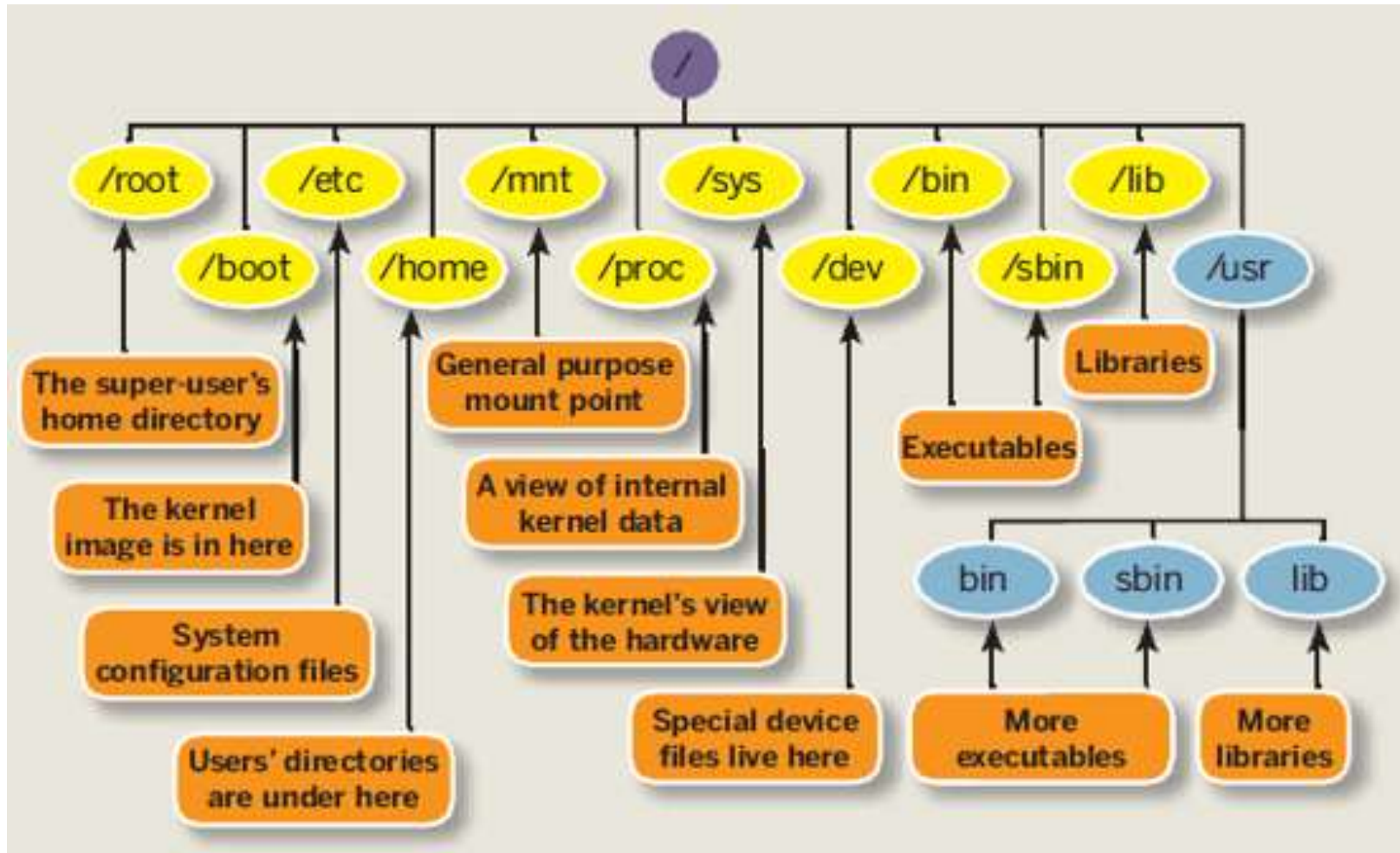




# Linux File System

- *hierarchical directory structure*: files are organized in a tree-like pattern of directories (folders), which may contain files and other directories, etc.
- Everything is a file:
  - Regular files
  - Directories: files listing a set of files
  - Symbolic links: files referring to the name of another file
- *root /*: the first directory in the file system.
- Note: comparison with Windows,
  - Windows has a separate file system tree for each storage device (e.g. C-drive, D-drive, I-drive, ...)
  - Linux has a single file system tree, regardless of how many drives or storage devices are attached to the computer.  
Storage devices are attached (or *mounted*) at various points on the tree.

# Linux File System



Source: <http://linuxsuperuser07.blogspot.be/2011/09/rhel-6-file-system.html>

# Linux File System

Not imposed by the system. Can vary from one system to the other, even between two GNU/Linux installations!

/	Root directory
/bin/	Basic, essential system commands
/boot/	Kernel images, initrd, configuration files
/dev/	Files representing devices
/etc/	System configuration files
/home/	User directories
/lib/	Basic system shared libraries
/media/	Mount points for removable media

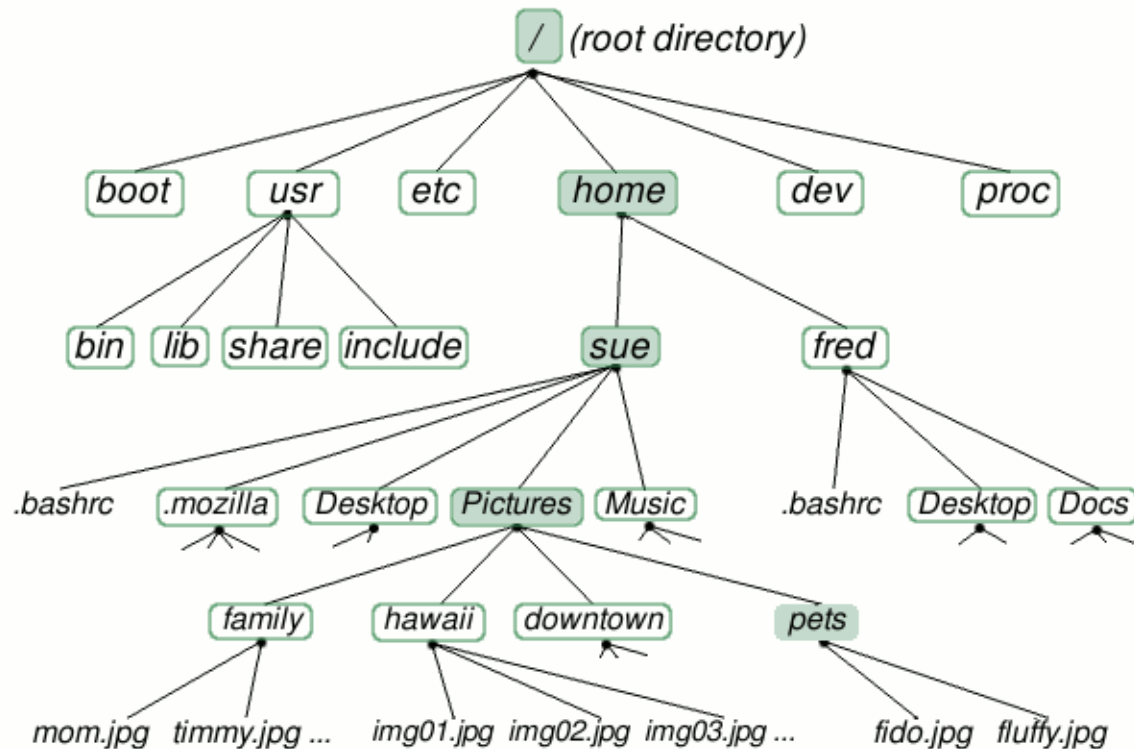
# Linux File System

<code>/lost+found/</code>	Corrupt files the system tried to recover
<code>/mnt/</code>	Mount points for temporarily mounted filesystems
<code>/opt/</code>	Specific tools installed by the sysadmin <code>/usr/local/</code> often used instead
<code>/proc/</code>	Access to system information
<code>/sbin/</code>	Administrator-only commands
<code>/sys/</code>	System and device controls
<code>/tmp/</code>	Temporary files

The Unix filesystem structure is defined by the Filesystem Hierarchy Standard (FHS):

<http://www.pathname.com/fhs/pub/fhs-2.3.html>

# Linux File System-home directory



Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the “.” character.

- `$ ls -a` (all)  
Lists all the files (including .\* files)
- `$ ls -l` (long)  
Long listing (type, date, size, owner, permissions)
- `$ ls -t` (time)  
Lists the most recent files first
- `$ ls -S` (size)  
Lists the biggest files first
- `$ ls -r` (reverse)  
Reverses the sort order
- `$ ls -ltr` (options can be combined)  
Long listing, most recent files at the end

# ls command

- `$ ls *txt`

The shell first replaces `*txt` by all the file and directory names ending by `txt` (including `.txt`), except those starting with `.`, and then executes the `ls` command line.

- `$ ls -d .*`

Lists all the files and directories starting with `.`  
`-d` tells `ls` not to display the contents of directories.

`$ ls -d */`

- `$ ls ?.log`

Lists all the files which names start by 1 character and end by `.log`

- <http://www.thegeekstuff.com/2009/07/linux-ls-command-examples/>

# ls command

```
vsc30051@login1:~/matlab/test> ls -al
total 1156
drwxr-xr-x  9 vsc30051 vsc30051    8192 2011-03-29 16:10 .
drwxr-xr-x  9 vsc30051 vsc30051    4096 2011-06-20 12:08 ..
drwxr-xr-x  3 vsc30051 vsc30051    4096 2010-01-08 10:54 courseAndreas
-rwxr--r--  1 vsc30051 vsc30051 149326 2011-03-10 15:14 fibonacci
-rw-r--r--  1 vsc30051 vsc30051    681 2011-03-10 15:09 fibonacci.m
-rwxr--r--  1 vsc30051 vsc30051 149068 2011-03-10 12:16 ftest
-rw-r--r--  1 vsc30051 vsc30051    443 2011-03-10 12:17 ftest_compiled.sh
-rw-r--r--  1 vsc30051 vsc30051    442 2011-03-10 11:28 ftest_compiled.sh~
-rw-r--r--  1 vsc30051 vsc30051     70 2011-03-10 12:15 ftest.m
```

- In Linux, file is defined as simply the thing that deals with a sequence of bytes
- Hence everything are files: an ordinary file is a file; a directory is also file; a network card, a hard disk, any device are also files since they deal with a sequence of bytes



# Moving around

- **Symbolic (soft) link**
  - Not a real file, just a link to another file
  - Allows giving another name to a file without actually duplicating it – hence saves memory space
- **Special file (device)**
  - Each hardware device, e.g. keyboard, hard disk, CD-ROM, etc. is associated with at least one file
  - Usually store in /dev directory
  - Applications can read and write any devices by reading and writing their associate file – hence the access method is known as device independent
  - Divide into two types: character special files, e.g. keyboard, and block special files, e.g. disk

# Is command

dlun@enpklun.polyu.edu.hk: /home/dlun/Desktop

File Edit Settings Help

```
[dlun@enpklun Desktop]$ ln -s ../CUI anotherCUI
[dlun@enpklun Desktop]$ ls -al
```

total 44

Permissions	Size	User	Group	Size	Month	Day	Time	Name
drwxr-xr-x	5	dlun	dlun	4096	Jan	4	18:36	.
drwx-----	16	dlun	dlun	4096	Jan	4	18:26	..
drwxr-xr-x	2	dlun	dlun	4096	May	17	2001	Autostart
-rw-r--r--	1	dlun	dlun	230	May	17	2001	Printer.kdeInk
-rw-r--r--	1	dlun	dlun	159	May	17	2001	Red Hat Errata.kdeInk
-rw-r--r--	1	dlun	dlun	153	May	17	2001	Red Hat Support.kdeInk
drwxr-xr-x	2	dlun	dlun	4096	May	17	2001	Templates
drwxr-xr-x	2	dlun	dlun	4096	May	17	2001	Trash
lrwxrwxrwx	1	dlun	dlun	6	Jan	4	18:36	anotherCUI -> ../CUI
-rw-r--r--	1	dlun	dlun	388	May	17	2001	cdrom.kdeInk
-rw-r--r--	1	dlun	dlun	395	May	17	2001	floppy.kdeInk
-rw-r--r--	1	dlun	dlun	144	May	17	2001	www.redhat.com.kdeInk

[dlun@enpklun Desktop]\$

Command that sets a symbolic link to a file called CUI to anotherCUI

A symbolic link begins with a letter /

Names in blue are directories, indicated by a letter d at the beginning of the line

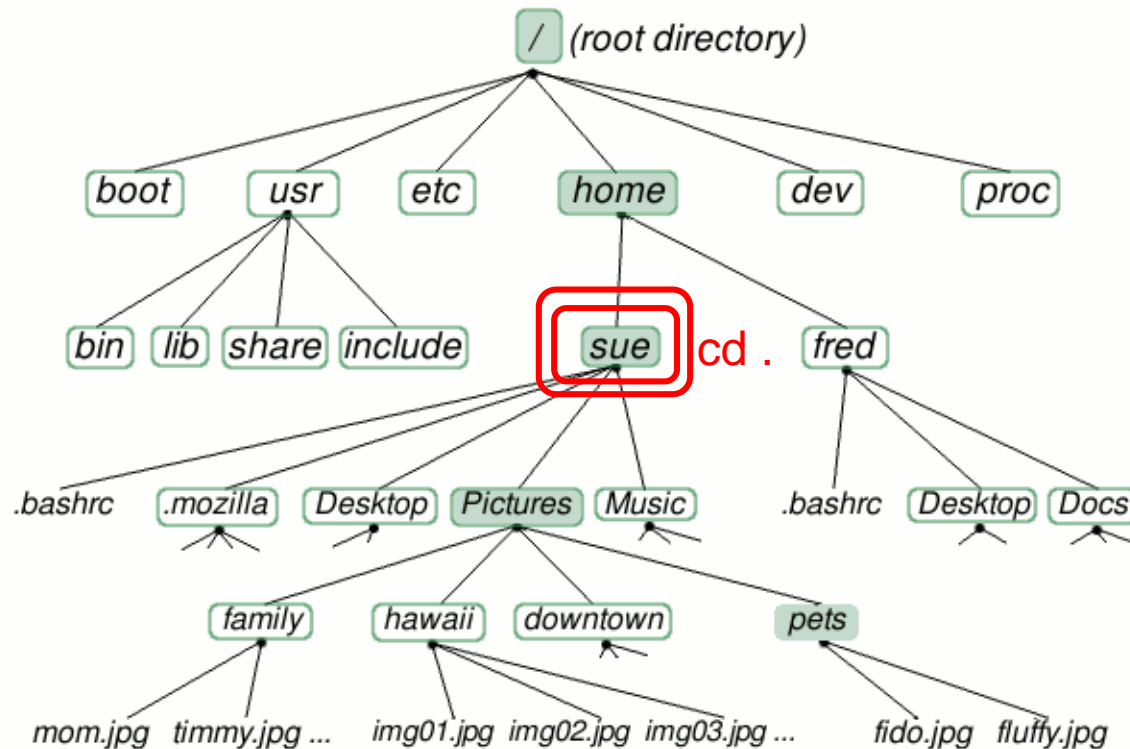
# Moving around

- Display the current/working directory
  - `$ pwd`
  - **Print Working Directory**
  - displays your current location within the file system.
- Change (navigate) directories.
  - `$ cd dir_name`
  - **Change Directories**
  - changes the position to the specific directory
- You can specify directory names in two ways:
  - Absolute pathname (starts from the root of the tree)  
`$ cd /u/home/hpc/test/bin`
  - Relative pathname (relative to your current directory)  
`$ cd`  
`$ cd .`  
`$ cd ..`  
`$ cd test/bin`

# Special directories

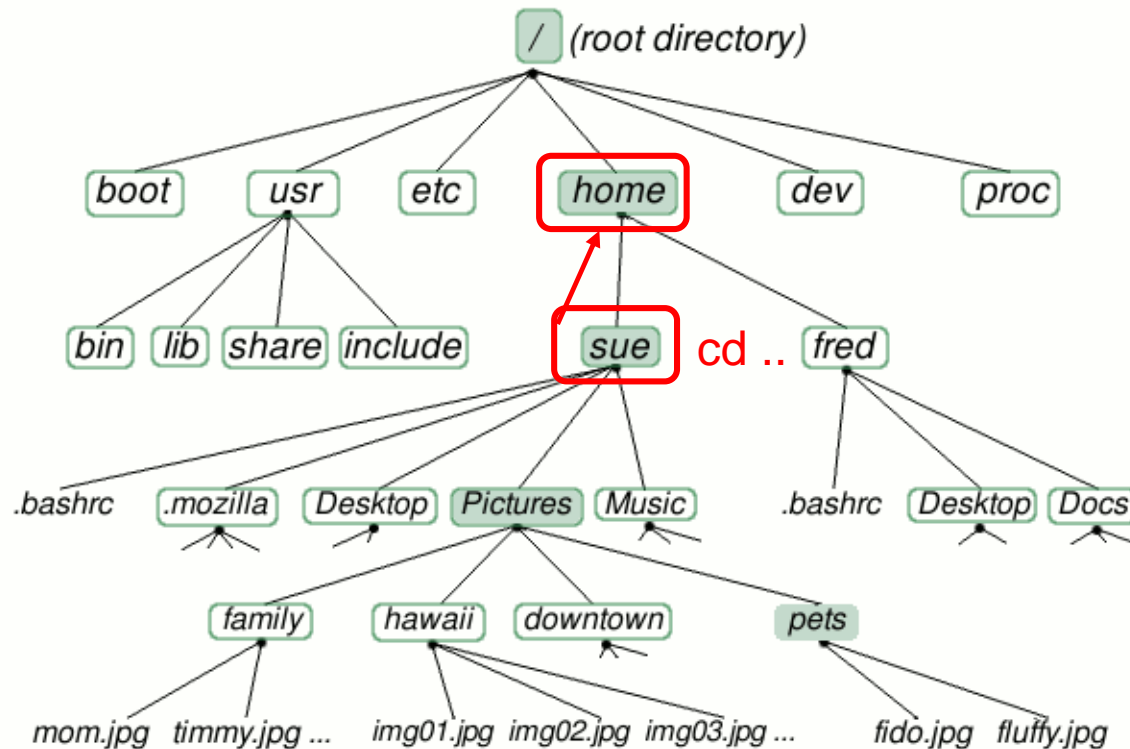
- `.`
  - The current directory.
  - Useful for commands taking a directory argument.
  - Useful to run commands in the current directory
  - `./readme.txt` and `readme.txt` are equivalent.
- `..`
  - The parent (enclosing) directory. Always belongs to the `.` Directory
  - Typical usage:  
`cd ..`
- `~`
  - Shells just substitute it by the home directory of the current user.
- `-`
  - `cd -` – jump back to the previous directory

# Linux File System - directories



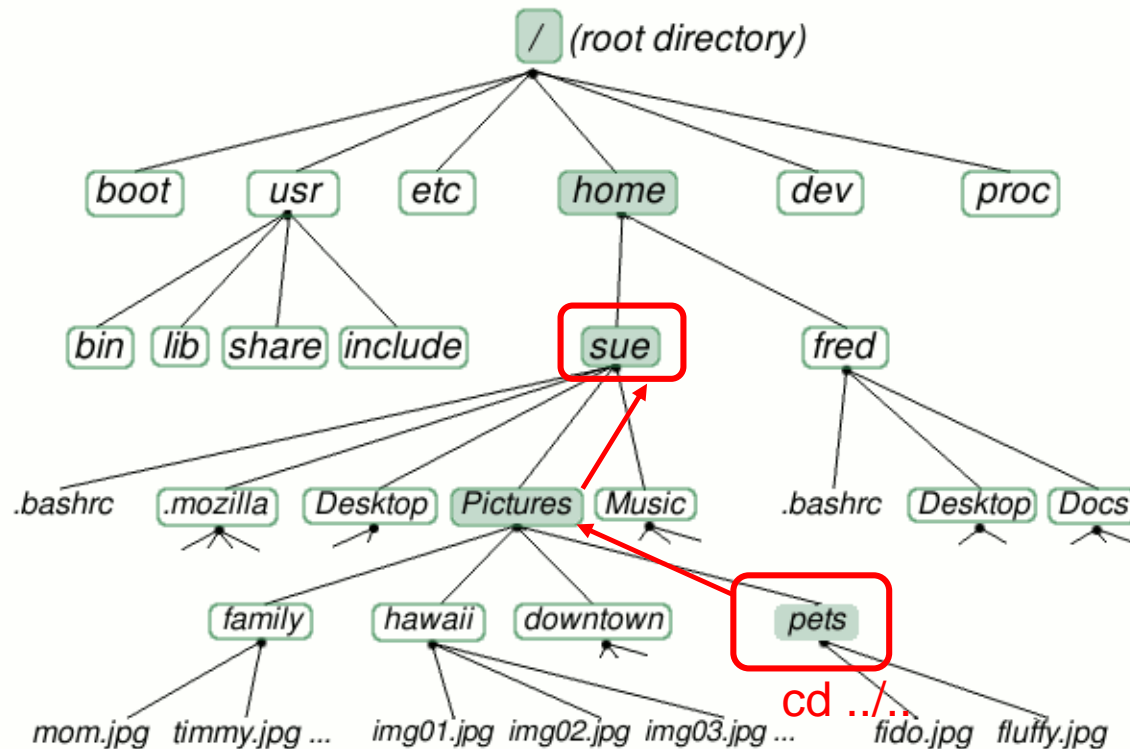
Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System - directories



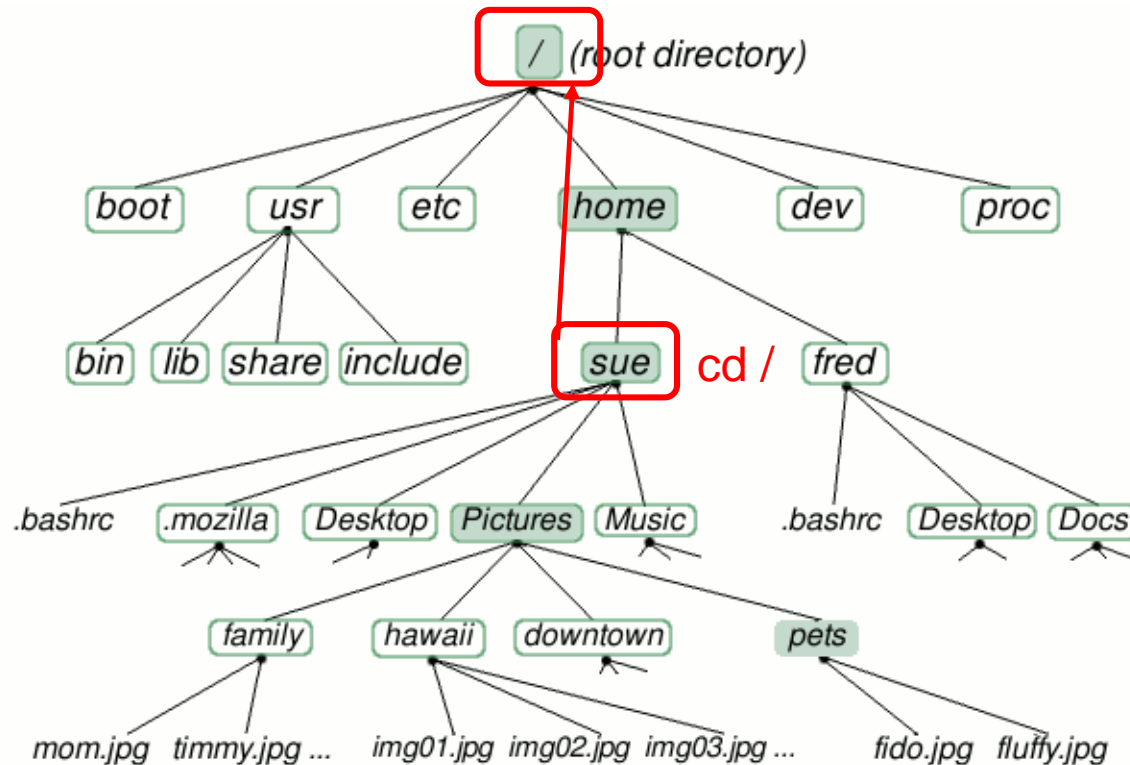
Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System - directories



Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

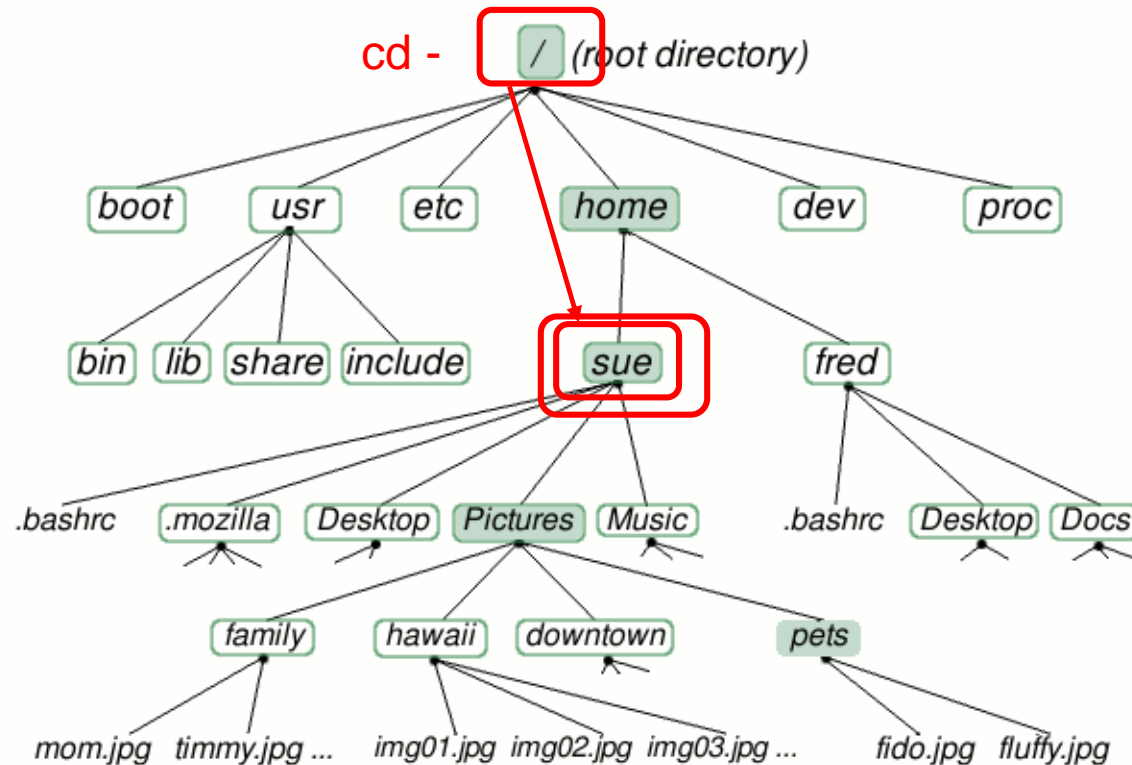
# Linux File System - directories



Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

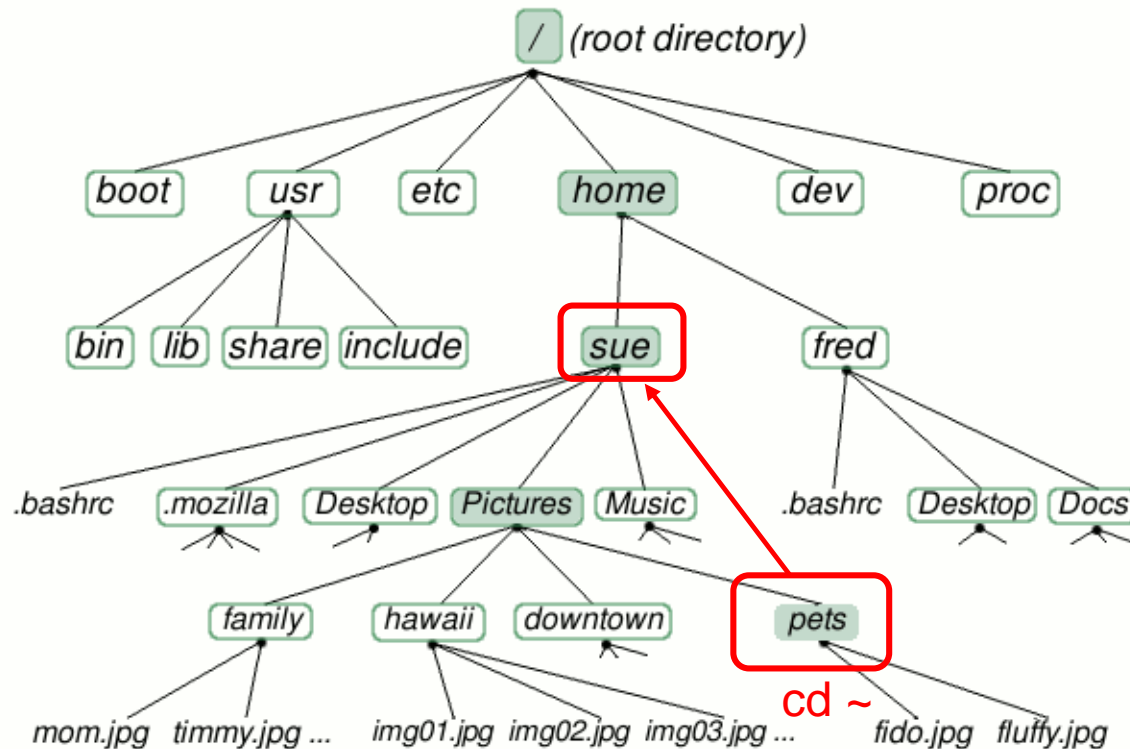


# Linux File System - directories



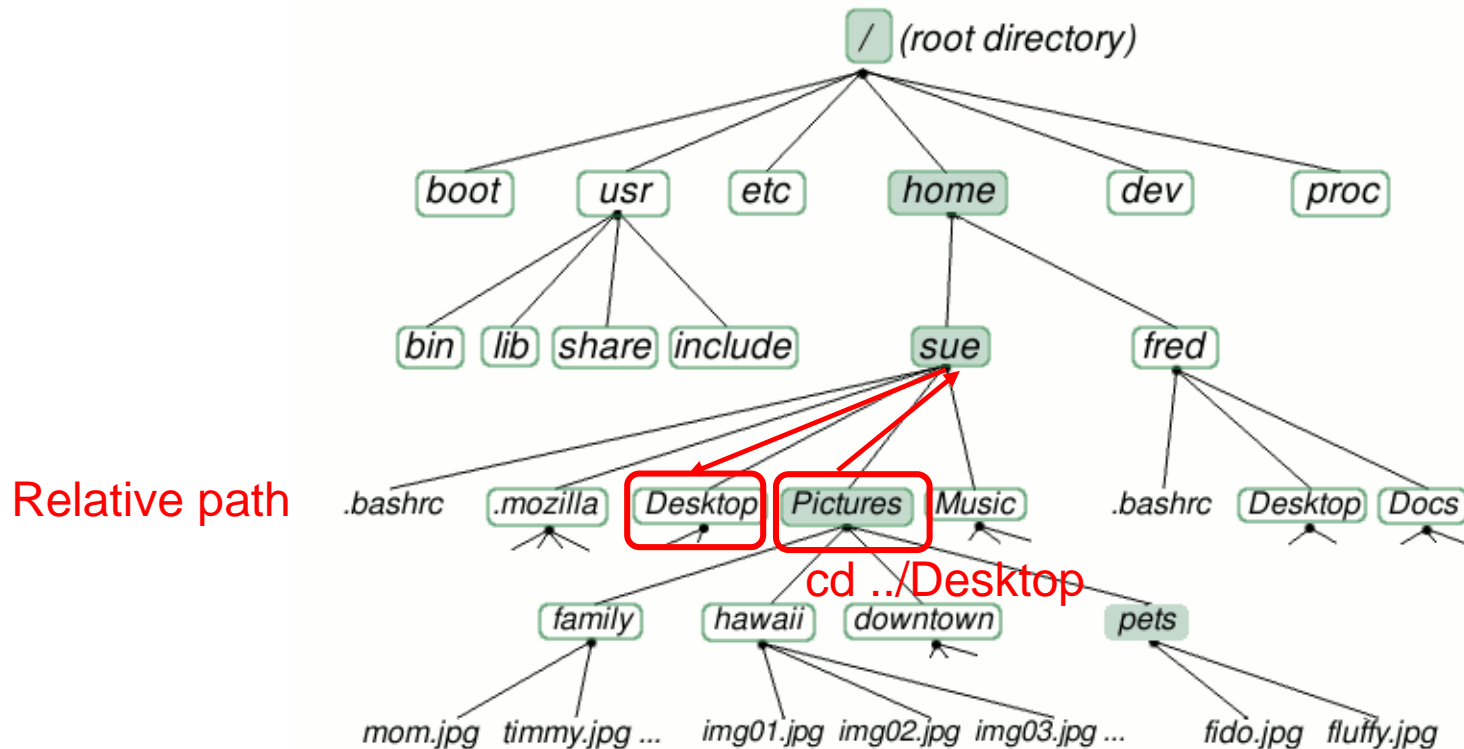
Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System - directories



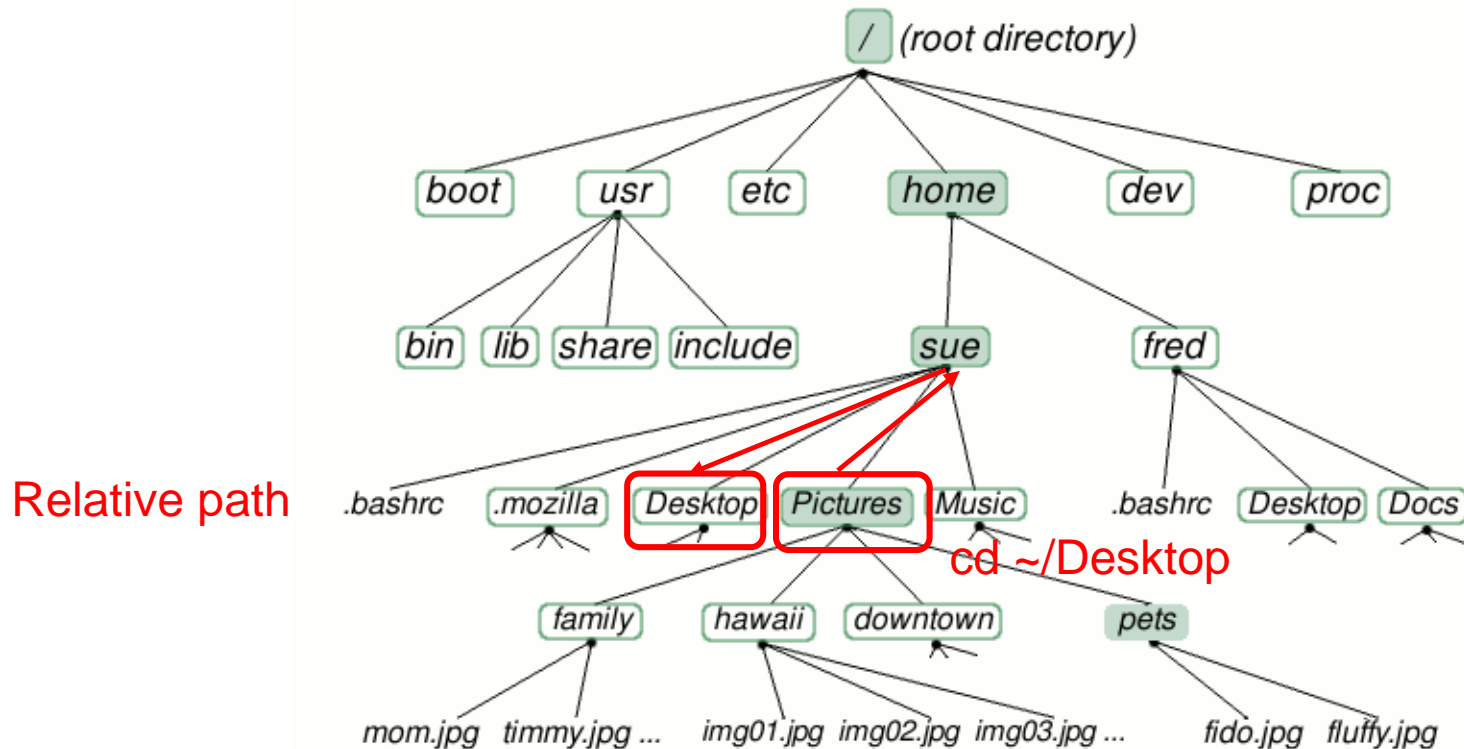
Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System-home directory



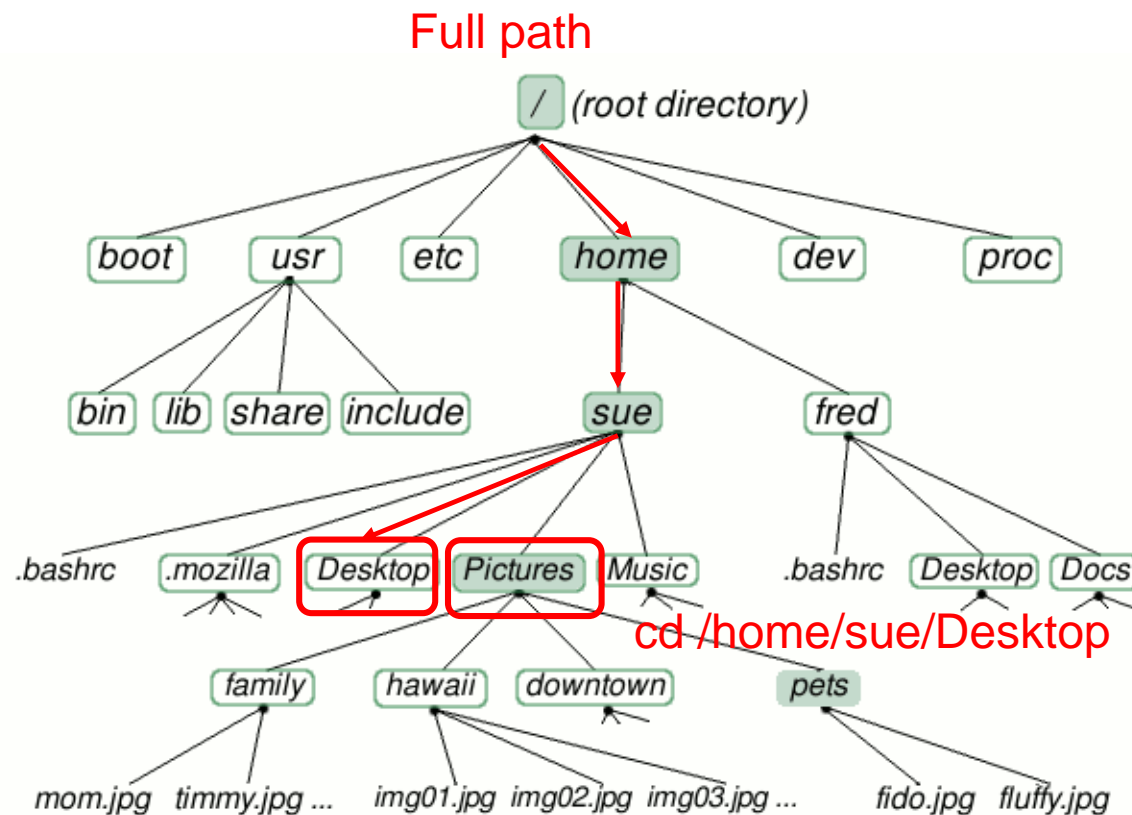
Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System - directories



Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# Linux File System - directories



Source: <http://www.linuxplanet.com/linuxplanet/tutorials/6666/1/screenshot3894/>

# File paths

- A path is a sequence of nested directories with a file or directory at the end, separated by the / character
- **Relative path:** `documents/fun/file1`  
Relative to the current directory
- To work with relative paths they need to have connection on the linux tree
- **Absolute path:** `/home/user/leuven/file2`
- / : root directory.  
Start of absolute paths for all files on the system  
Not a superuser's home directory (-> /root)

# More on files

- What does a file contain?
  - Determine a file's type: **file**
  - will print a brief description of the file's contents
  - `$ file filename`
- Search for files and directories
  - The **find** command performs a raw search on a file system to locate the specified items.
  - `$ find location -name some-name`  
`($ find /home/student -name *desktop)`
  - Be careful: results can sometimes be slow
- Search the locate database for files and directories
  - The **locate** command displays the location of files that match the specified name.
  - Faster than find because it searches a database of indexed filenames.
  - Disadvantage: lacks the ability to search for advanced characteristics such as file owner, size, and modification time.

# More on files

- Display extended information about a file system, file, or directory
- The `stat` command displays extended information about files. It includes helpful information not available when using the `ls` command



# File names

## File name features:

- Case sensitive
- No obvious length limit (255 characters)
- Can contain any character (including whitespace, except `/`).
- File name extensions not needed and not interpreted. Just used for user convenience. (File types stored in the file)
- a hidden file is any file that begins with a “.” (not seen with the bare `ls`)
- File name examples:

```
README                .bashrc                index.htm  
index.html            index.html.old
```

# The Shell revisited: features



# Auto-Completion

- Have the shell automatically complete commands or file paths.
- Activated using the **<TAB>** key on most systems
- examples
  - `$ whe<TAB>`
  - `$ whereis`
  - `$ ls -l /etc/en<TAB>`
  - `$ ls -l /etc/environment`
- When more than one match is found, the shell will display all matching results (use **<TAB>** twice)
  - `$ ls -l /etc/host<TAB>`

# Command history: Arrow Up

- Previously executed commands can be recalled by using the **Up Arrow** key on the keyboard.
- Most Linux distributions remember the last five hundred commands by default.
- Display commands that have recently been executed
  - The `history` command displays a user's command line history.
  - You can execute a previous command using `! [NUM]` where NUM is the line number in history you want to recall.

# Globbering: use wildcard

Wildcard	Function
*	Matches 0 or more characters
?	Matches 1 character
[abc]	Matches one of the characters listed
[a-c]	Matches one character in the range
[!abc]	Matches any character not listed
[!a-c]	Matches any character not listed in the range
{tacos,nachos}	Matches one word in the list

```
$ ls -l /etc/host*
$ ls -l /etc/hosts.{allow,deny}
$ ls -l /etc/hosts.[!a]*
$ ls -l /etc/host?
```

# Pipes

- Pipes (also referred to as pipelines) can be used to direct the output of one command to the input of another. The Shell arranges it so that the standard output of one command is fed to another command
- use the `|` key on the keyboard
- `$ ls -l | less`  
(compare with `$ ls -l > less`)

# Standard output

- All the commands outputting text on your terminal do it by writing to the **standard output**.  
This is where the normal output from a program goes.
- Standard output can be written (**redirected**) to a file using the **>** symbol

```
$ ls > /tmp/list.txt
```

```
$ date > date.txt
```

- Standard output can be **appended** to an existing file using the **>>** symbol

```
$ ls >> /tmp/list.txt
```

# Standard output

- There are two different types of output:
  - 1. Standard output (STDOUT)
  - 2. Error output (STDERR)

```
$ ls -l /NonExistFile 1>ls.txt 2>lserr.txt
```

```
$ ls -l ls*
```

```
$ cat ls.txt
```

```
$ cat lserr.txt
```

```
ls: cannot access /NonExistFile: No such  
file or directory
```



# Standard input

- In addition to *STDOUT* and *STDERR* you can also redirect input from another location (such as a file) to a command. This is known as standard input or *STDIN*.
- the source of input data for command line programs
- *STDIN*: usually keyboard
- redirect also possible
- `$ sort < tabel.dat`

# Hands-on 2



# File manipulation



# File Manipulation

- For all file manipulation commands relative or absolute paths can be used (or just files in the current directory when no extra path specified)
- Most of the commands are intuitive – shortcuts of English names

# Directories

- Create directories
  - The `mkdir` command is used to create directories
  - `$ mkdir dir1 dir2 dir3`
- Remove directories
  - The `rmdir` command removes directories
  - `$ rmdir dir1`
  - `rmdir` will only remove empty directories.  
To remove a non-empty directory, use  
`$ rm -r [DIRECTORY] instead.`

# Copy a file

- The `cp` command copies files and directories
- The default behavior will overwrite any existing file(s). The `-i` option overrides this behavior and prompts the user before overwriting the destination file.
- **syntax:** `cp [OPTIONS] [SOURCE] [DESTINATION]`
  - `$ cp <source_file> <target_file>`  
Copies the source file to the target.
  - `$ cp file1 file2 file3 ... dir`  
Copies the files to the target directory (last argument).
  - `$ cp -i (interactive)`  
Asks for user confirmation if the target file already exists
  - `$ cp -r <source_dir> <target_dir> (recursive)`  
Copies the whole directory.
  - `$ cp -v (verbose)`  
Displays what has been copied

# Move or rename files

- Move or rename files and directories: `mv`
- The default behavior will overwrite any existing file(s).
- **syntax:** `mv [OPTIONS] [SOURCE] [DESTINATION]`
  - `$ mv old_name new_name`  
Renames the given file or directory.
  - `$ mv -i (interactive)`  
If the new file already exists, asks for user confirm
- The `mv` command can also be used to move or rename directories
  - `$ mv NewFiles/ OldFiles/`
  - `-r` option is not necessary



# Remove files

- The `rm` command removes files.

- `$ rm file1 file2 file3 ...`

Removes the given files.

- `$ rm -i` (interactive)

Always ask for user confirmation.

- `$ rm -r dir1 dir2 dir3` (recursive)

Removes the given directories with all their contents.

- Tip:

Whenever you use wildcards with `rm` (besides carefully checking your typing!), test the wildcard first with `ls`. This will let you see the files that will be deleted. Then press the up arrow key to recall the command and replace the `ls` with `rm`.



# Create links

- Create links (shortcuts) to files or directories: `ln`
- Symbolic links are created when using the `-s` option with the `ln` command.
- Allows to jump to other files or locations on the file system
- Editing a symbolic link file is the same as editing the source file, but deleting the symbolic link does not delete the source file.
  - `$ ln -s file_v5.doc file_final.doc`
  - creates a symbolic link called `file_final.doc` that points to `file_v5.doc`
  - `$ ln -s /home/demo/dir1/dir2/dir3 /home/demo/jump2dir`
  - creates a symbolic link called `jump2dir` that points to a deep directory - allows for quicker access)

# Text editing



# Text editors

- Text-only text editors
  - Often needed for sysadmins and great for power users
  - vi, vim
  - nano
- Graphical text editors
  - Fine for most needs
  - Gedit,
  - Kate, Nedit
  - Emacs, Xemacs
- <http://www.thegeekstuff.com/2009/07/top-5-best-linux-text-editors/>

# Text editors

## How to start:

- Create a file
  - `$ touch filename` (creates an empty file)
  - start editing (non)-existing file:
    - `$ vi filename`
    - `$ nano filename`
    - `$ gedit filename`

# vi

- Text-mode text editor available in all Linux systems.
- Created before computers with mice appeared.
- Difficult to learn for beginners used to graphical text editors.
- Very productive for power users.
- Check the web for tutorials:
  - [https://upload.wikimedia.org/wikipedia/commons/d/d2/Learning\\_the\\_vi\\_Editor.pdf](https://upload.wikimedia.org/wikipedia/commons/d/d2/Learning_the_vi_Editor.pdf)
  - <ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf>

# vi

- 2 basic modes of operation:
  - *command mode* and *editing mode*.
  - Within Command Mode, signals from the terminal are interpreted as editing commands.
  - Editing mode: letters typed at the keyboard are inserted into the editing buffer.
- Pressing **Esc** on the keyboard activates command mode.

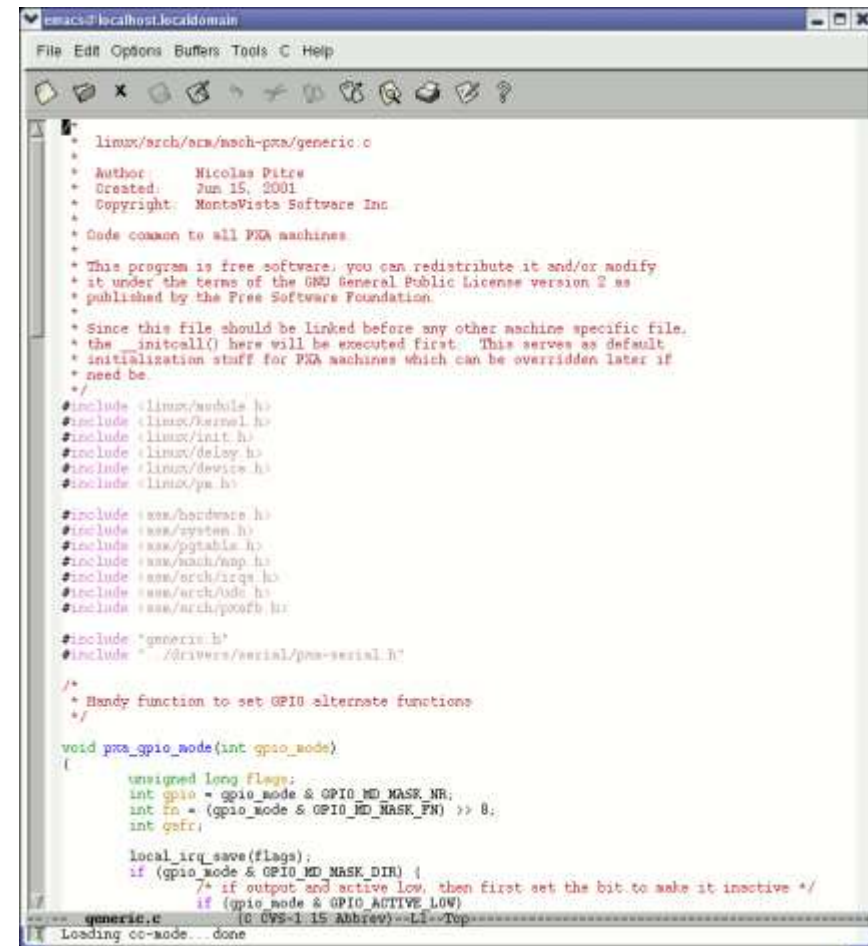
Key(s)	Function	Key(s)	Function
:w	Save	A	Append text after
:x	Save and exit	r	Replace text before cursor
:q	Quit	R	Replace text after cursor
I	Insert text after	i	Insert text before
P	Paste copied text	yy	Copy current line
a	Append text before	/[TEXT]	Search for the specified text

# Vi

- 2 modes
  - Input mode
    - ESC to back to cmd mode
  - Command mode
    - Cursor movement
      - h (left), j (down), k (up), l (right)
      - ^f (page down)
      - ^b (page up)
      - ^ (first char.)
      - \$ (last char.)
      - G (bottom page)
      - :1 (goto first line)
    - Switch to input mode
      - a (append)
      - i (insert)
      - o (insert line after)
      - O (insert line before)
- Delete
  - dd (delete a line)
  - d10d (delete 10 lines)
  - d\$ (delete till end of line)
  - dG (delete till end of file)
  - x (current char.)
- Paste
  - p (paste after)
  - P (paste before)
- Undo
  - u
- Search
  - /
- Save/Quit
  - :w (write)
  - :q (quit)
  - :wq (write and quit)
  - :q! (give up changes)

# Emacs

- Extremely powerful text editor features
- Great for power users
- Non standard shortcuts
- Much more than a text editor (games, e-mail, shell, browser).
- Some power commands have to be learnt.



```
emacs@localhost.localdomain
File Edit Options Buffers Tools C Help

linux/arch/arm/mach-pxa/generic.c
* Author: Nicolas Pitre
* Created: Jun 15, 2001
* Copyright: MontaVista Software Inc
* Code common to all PXA machines
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
* Since this file should be linked before any other machine specific file,
* the __initcall() here will be executed first. This serves as default
* initialization stuff for PXA machines which can be overridden later if
* need be.
*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/pa.h>

#include <asm/hardware.h>
#include <asm/system.h>
#include <asm/pgtable.h>
#include <asm/arch/omap.h>
#include <asm/arch/irqs.h>
#include <asm/arch/uda.h>
#include <asm/arch/pccofb.h>

#include "generic.h"
#include "drivers/serial/pxa-serial.h"

/*
 * Handy function to set GPIO alternate functions
 */

void pxa_gpio_mode(int gpio_mode)
{
    unsigned long flags;
    int gpio = gpio_mode & GPIO_MD_MASK_NR;
    int fn = (gpio_mode & GPIO_MD_MASK_FN) >> 8;
    int gqfr;

    local_irq_save(flags);
    if (gpio_mode & GPIO_MD_MASK_DIR) {
        /* if output and active low, then first set the bit to make it inactive */
        if (gpio_mode & GPIO_ACTIVE_LOW)
            (C CVS-1.15 Abbrev) <= Top
Loading cc-mode... done
```



# Emacs

- \$ emacs
- Cursor movement
  - ^f (forward one char.)
  - ^b (backward one char.)
  - ^a (begin of line)
  - ^e (end of line)
  - ^n (next line)
  - ^p (prev. line)
  - ^v (page up)
  - alt-v (page down)
- Deletion
  - ^d (delete one char)
  - alt-d (delete one word)
  - ^k (delete line)
- Paste
  - ^y (yank)
- Undo
  - ^/
- Load file
  - ^x^f
- Cancel
  - ^g
- Save/Quit
  - ^x^c (quit w/out saving)
  - ^x^s (save)
  - ^x^w (write to a new file)

# Nano

u0076109@hpc-p-svcs-10:~

GNU nano 2.2.6 File: testfile

Exit=Ctrl+X

[ New File ]

^G Get Help	^O WriteOut	^R Read File	^Y Prev Page	^K Cut Text	^C Cur Pos
^X Exit	^J Justify	^W Where Is	^U Next Page	^U UnCut Text	^T To Spell

# Displaying file contents

Several ways of displaying the contents of files.

- `$ cat file1`  
displays the contents of the given file.
- `$ cat file1 file2 file3 ...` (concatenate)  
Concatenates and outputs the contents of the given files.
- `$ more file1`  
Display the output of a command or text file one page at a time.
- Can also jump to the first occurrence of a keyword (/ command).

# Displaying file contents

- `$ less file1`

Does more than more.

Doesn't read the whole file before starting.

Supports backward movement in the file (? command).

Press `q` to exit

- `$ display file1`

Displays graphical file (simple image)

# The head and tail commands

- `$ head [-<n>] <file>`

Displays the first <n> lines (or 10 by default) of the given file.  
Doesn't have to open the whole file to do this!

- `$ tail [-<n>] <file>`

Displays the last <n> lines (or 10 by default) of the given file.  
No need to load the whole file in RAM! Very useful for huge files.

- `$ tail -f <file> (follow)`

Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.  
Very useful to follow the changes in a log file, for example.

# The grep command

- `$ grep <pattern> <files>`  
Scans the given files and displays the lines which match the given pattern.
- `$ grep error *.log`  
Displays all the lines containing error in the \*.log files
- `$ grep -i error *.log`  
Same, but case insensitive
- `$ grep -ri error .`  
Same, but recursively in all the files in the current directory and its subdirectories
- `$ grep -v info *.log`  
Outputs all the lines in the files except those containing info.
- <http://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples/>

# wget

- Instead of downloading files from your browser, just copy and paste their URL and download them with wget
- main features
  - http and ftp support
  - Can resume interrupted downloads
  - Can download entire sites or at least check for bad links
  - Very useful in scripts or when no graphics are available (system administration, embedded systems)

•\$ `wget -c http://microsoft.com/customers/dogs/winxp4dogs.zip`

Continues an interrupted download.

•\$ `wget -r -np http://www.xml.com/ldd/chapter/book/`

Recursively downloads an on-line book for off-line access.

-np: "no-parent". Only follows links in the current directory.



## Hands-on 3





# Various commands



# time

- time a helpful command for doing simple benchmarking
- run your scripts along with **time** command, and compare their's execution time.

- Example:

```
$ time ls
```

```
real    0m2.304s (actual elapsed time)
```

```
user    0m0.449s (CPU time running program code)
```

```
sys     0m0.106s (CPU time running system calls)
```

real = user + sys + waiting

waiting = I/O waiting time + idle time (running other tasks)

# information about users

- `$ who`  
Lists all the users logged on the system.
- `$ whoami`  
Tells what user I am logged as.
- `$ groups`  
Tells which groups I belong to.

# More commands

- `$ sleep 60`  
Waits for 60 seconds  
(doesn't consume system resources).
- `$ wc report.txt`  
word count  
Counts the number of lines, words and characters in a file  
or in standard input.
- `$ date`  
Returns the current date. Useful in scripts to record when  
commands started or completed.
- Before you run a command, `which` tells you where it is  
found  
`$ which ls`

# Measuring disk usage

- `$ du -h <file>`  
-h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes),  
Without -h, du returns the raw number of disk blocks used by the file (hard to read).  
Note that the -h option only exists in GNU du.
- `$ du -sh <dir>`  
-s: returns the sum of disk usage of all the files in the given directory.

# Measuring disk space

- `$ df -h <dir>`

Returns disk usage and free space for the filesystem containing the given directory. Similarly, the `-h` option only exists in GNU `df`.

- **Example:**

```
$ df -h .
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda5	9.2G	7.1G	1.8G	81%	/

- `$ df -h`

Returns disk space information for all filesystems available in the system. When errors happen, useful to look for full filesystems.

# Comparing files and directories

- `$ diff file1 file2`

Reports the differences between 2 files, or nothing if the files are identical.

- `$ diff -r dir1/ dir2/`

Reports all the differences between files with the same name in the 2 directories.

- These differences can be saved in a file using the redirection, and then later re-applied.
- <https://linuxacademy.com/blog/linux/introduction-using-diff-and-patch/>

# Other commands

- `cut` – cuts columns of text from a file
- `echo` – displays a line of text
- `paste` – will paste columns of text into a file
- `sort` – sorts a file of lines alphabetically
- `tr` – translates between characters (e.g. `tr a-z A-Z`)
- `rpm -q` – query about installed software
- `whereis` - locate the binary, source, and manual page files for a command
- `which` - shows the full path of (shell) commands
- `bc` – command line calculator (scale=2 to see 2 digits)



archiving



# File Archiving: tar

- Saves and restores multiple files to/from a single file. Directories followed recursively.
- Format:
  - `$ tar [options] [options_values] [files]`
  - `c` – create a new archive
  - `v` – verbosely list files which are processed.
  - `f` – following is the archive file name
  - `z` – filter the archive through gzip (compress)
  - `x` – extract files from archive
  - `j` - filter the archive through bzip (compress)

# File Archiving: tar

- Examples:

- `$ tar -cvf [FILE] [ITEM]` Backup the specified item(s)

- `$ tar -cvf /tmp/backup.tar ~/data ~/test`

- `$ tar -czvf [FILE] [ITEM]` Compress the archive to save space

- `$ tar -xvf [FILE] [ITEM]` Restore the specified item(s) **`$tar -xvf backup.tar`**

- `$ tar -tf [FILE]` List all files in the specified archive

- `e.g. $ tar -tf backup.tar`

- <http://www.thegeekstuff.com/2010/04/unix-tar-command-examples/>

# File Compression: gzip

- **Compressing files:** `gzip filename` or `bzip2 filename`

- `$ gzip backup.tar`
- `$ bzip2 backup.tar`

The resulted file is `backup.tar.gz/ backup.tar.bz2`

- **Uncompressing files:** `gzip -d filename.gz` or `bzip2 -d filename.bz2`

- `$ gzip -d backup.tar.gz`
- `$ bzip2 -d backup.tar.bz2`

The uncompressed file is `backup.tar`

# Users, groups, security



# File access rights

## Linux File Access Privilege

- Linux is a multiuser system, the files of all users are stored in a single file structure
- Mechanism is required to restrict one user to access the files of another user, if he is not supposed to
- User can impose access permission to each file to restrict its access
- The term “access permission” refers to
  - read permission
  - write permission
  - execute permission

# File access rights

## 3 types of **access rights**

- Read access (r)
  - reading, opening, viewing, and copying the file is allowed
- Write access (w)
  - writing, changing, deleting, and saving the file is allowed
- Execute rights (x)
  - executing and invoking the file is allowed. This is required for directories to allow searching and access.

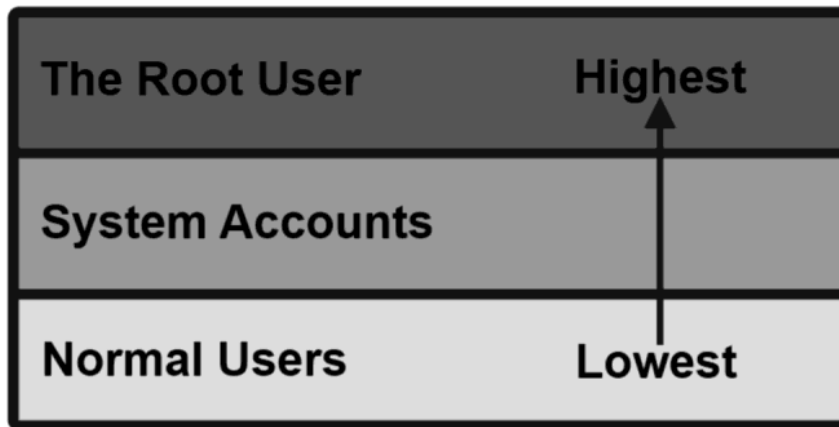
*Use `ls -l` to check file access rights*

# File access rights

## 3 types of **access levels**

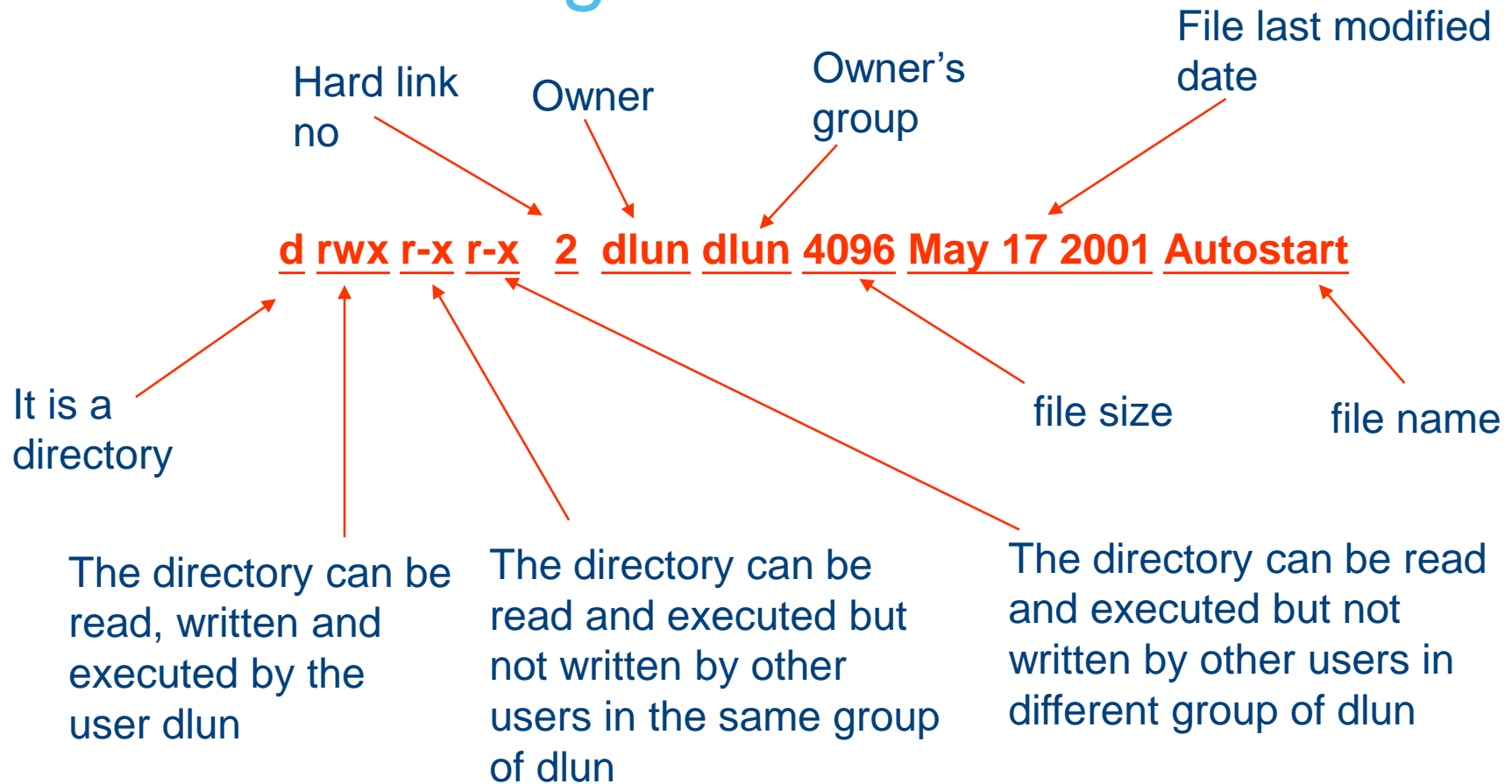
- User (u): for the owner of the file
- Group (g): each file also has a “group” attribute, corresponding to a given list of users
- Others (o): for all other users

### **Privileges**





# File access rights



The group of a user is assigned by the administrator when a user is added to the system

# File access rights

- Access permission can also be assigned to a directory
- Directory is also a file that contains the attributes of the files inside it
- If read permission is not given to a directory
  - cannot show the structure of this directory
  - e.g. cannot use ls
- If write permission is not given to a directory
  - cannot modify anything of the directory structure
  - e.g. cannot copy a file into this directory since it will modify the directory structure by adding one more file
- If execute permission is not given to a directory
  - nearly nothing can be done with this directory, even cd

# Access rights examples

- `-rw-r--r--`  
Readable and writable for file owner, only readable for others
- `-rw-r-----`  
Readable and writable for file owner, only readable for users belonging to the file group.
- `drwx-----`  
Directory only accessible by its owner
- `-----r-x`  
File executable by others but neither by your friends nor by yourself. Nice protections for a trap...

# Access rights examples

dlun@enpklun.polyu.edu.hk: /home/dlun/Desktop/test/temp

File Edit Settings Help

temp does not have execution right

```
[dlun@enpklun test]$ ls -l
total 12
-rw-r--r--  1 dlun    dlun      395 Jan  7 16:36 floppy.kdeInk
drw-----  2 dlun    dlun    4096 Jan  9 11:06 temp
-rw-rw-r--  1 dlun    dlun      16 Jan  7 16:05 test1.txt
```

even cd is not workable

```
[dlun@enpklun test]$ cd temp
bash: cd: temp: Permission denied
```

execution right is added

```
[dlun@enpklun test]$ chmod 700 temp
[dlun@enpklun test]$ ls -l
total 12
-rw-r--r--  1 dlun    dlun      395 Jan  7 16:36 floppy.kdeInk
drwx-----  2 dlun    dlun    4096 Jan  9 11:06 temp
-rw-rw-r--  1 dlun    dlun      16 Jan  7 16:05 test1.txt
[dlun@enpklun test]$ cd temp
[dlun@enpklun temp]$
```

now we can change the directory to temp

# chmod: changing permissions

- Permissions allow you to share files or directories or to lock them down to be private.
- `$ chmod (change mode)`
- `$ chmod <permissions> <files>`  
2 formats for permissions:
  - octal format (3 digit octal form)
  - symbolic format

# chmod: changing permissions

- octal format (abc):

$a, b, c = r*4 + w*2 + x*1$  (r, w, x: booleans)

- 0 none ---
- 1 execute-only --x
- 2 write -w-
- 3 execute and write -wx
- 4 read-only r-
- 5 read and execute r-x
- 6 read and write rw-
- 7 read, write, and execute rwx

- `$ chmod 644 <file>`  
(rw for u, r for g and o)

660 : 110 110 000

⇒ rw- rw- ---

545 : 101 100 101

⇒ r-x r-- r-x

# chmod: changing permissions

- **symbolic format:**

\$ `chmod go+r`: add read permissions to group and others.  
\$ `chmod u-w`: remove write permissions from user.  
\$ `chmod a-x`: (a: all) remove execute permission from all.

# Access right constraints

- x is sufficient to execute binaries  
Both x and r are required for shell scripts.
- Both r and x permissions needed in practice for directories:  
r to list the contents, x to access the contents.
- You can't rename, remove, copy files in a directory if you don't have w access to this directory.
- If you have w access to a directory, you can remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without w access to it.



# process control

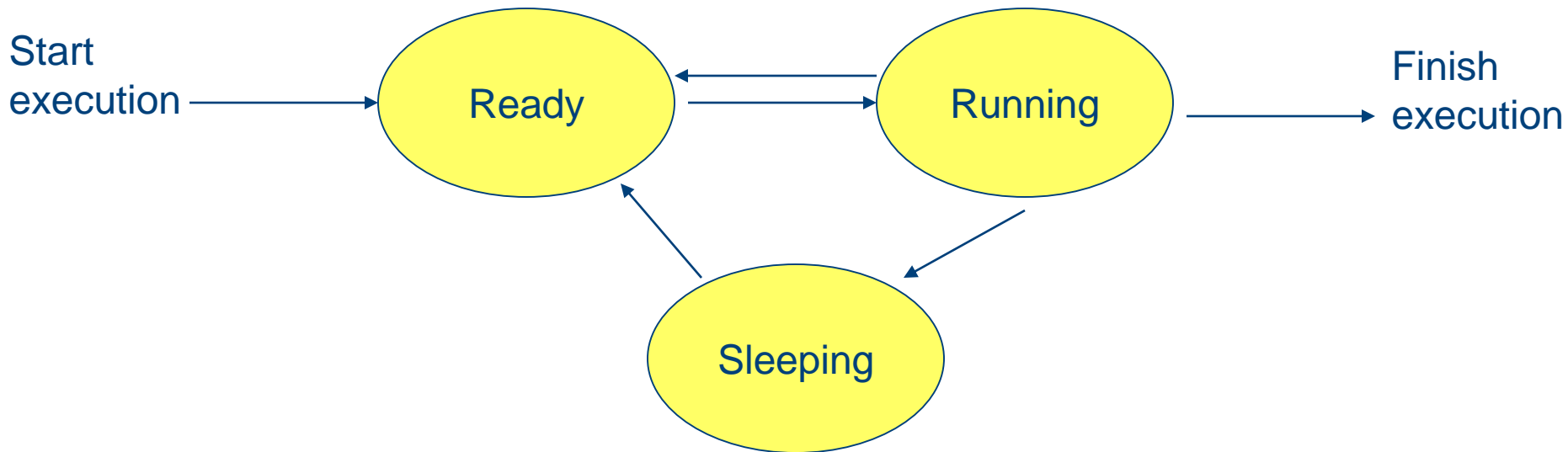


# Process

- “Everything in Unix is a file. Everything in Unix that is not a file is a process”
- Processes
  - Instances of a running programs
  - Several instances of the same program can run at the same time
  - Processes are assigned a unique identifier which is used to monitor and control the process (PID)

# Process

- A program that is claimed to be executing is called a process
- For a multitasking system, a process has at least the following three states:



# Process

- Ready state
  - All processes that are ready to execute but without the CPU are at the ready state
  - If there is only 1 CPU in the system, all processes except one are at the ready state
- Running state
  - The process that actually possesses the CPU is at the running state
  - If there is only 1 CPU in the system, at most there is only one process is at the running state
- Sleeping state
  - The process that is waiting for other resources, e.g. I/O, is at the sleeping state

# ps

- Display running processes  
(cfr. Windows Task Manager ctrl-shift-esc)
- `$ ps` Display the current user's processes
- `$ ps -e` Display all processes running on the system
- `$ ps -ef` Display detailed information about running processes
- `$ ps -u [USER]` Display processes owned by the specified user
- `$ ps a` Display extra info (running state)

# Process

PID	TTY	STAT	TIME	COMMAND
14748	pts/1	S	0:00	-bash
14795	pts/0	S	0:00	-bash
14974	pts/0	S	0:00	vi test1.txt
14876	pts/1	R	0:00	ps ...

Process ID

Terminal  
name

State:  
S – Sleeping  
(waiting for input)  
R – Running

How much time the process  
is continuously executing

- For the example above, both bash processes, which are the shell of both terminals, are waiting for the input of user. They must be in the sleeping state
- The vi process, which is an editor, is also waiting for the input of user. Hence it is also in sleeping state
- When ps reporting the processes in the system, it is the only process that is running. Hence it is in running state

# kill

- Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first.

- `$ kill <pid>`

**Example:**

```
$ kill 3039 3134 3190 3416
```

- `$ kill -9 <pid>`

Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck.

# killall

- The `killall` command terminates all processes that match the specified name
- `$ killall [-<signal>] <command>`

**Example:**

```
$ killall bash
```



# xkill

`xkill`

Lets you kill a graphical application by clicking on it!  
Very quick! Convenient when you don't know the application command name.

# &

- `&` is a command line operator that instructs the shell to start the specified program in the background.
- This allows you to have more than one program running at the same time without having to start multiple terminal sessions.
- Starting a process in background: add `&` at the end of your line:  
`$ gedit &`  
check with `ps`

# top

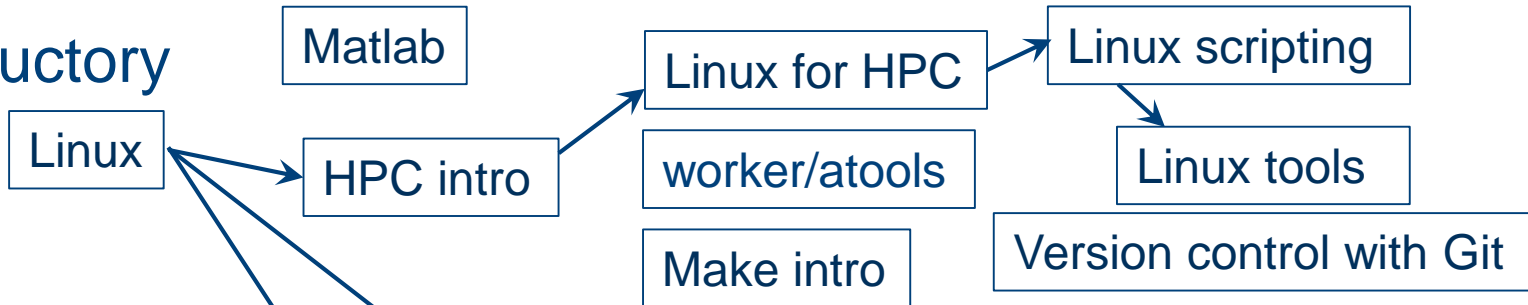
- Displays most important processes, sorted by cpu percentage (use > or < to change the order)

<http://www.thegeekstuff.com/2010/01/15-practical-unix-linux-top-command-examples/>

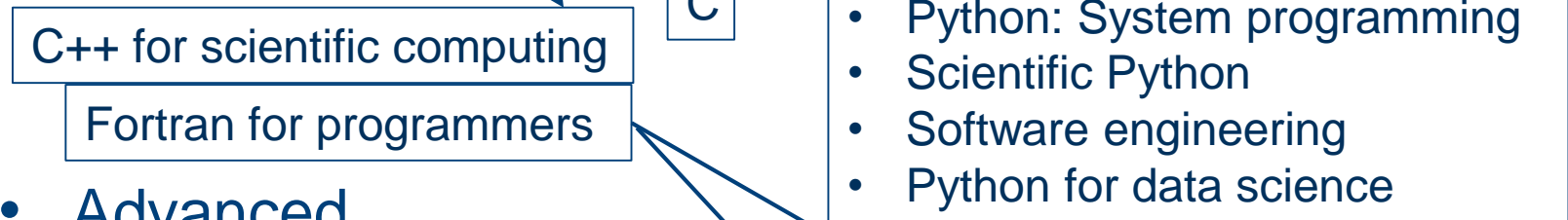
```
frank@frank-laptop: ~  
File Edit View Terminal Help  
  
top - 15:58:03 up 7:42, 3 users, load average: 0.33, 0.27, 0.20  
Tasks: 136 total, 1 running, 135 sleeping, 0 stopped, 0 zombie  
Cpu(s): 3.5%us, 5.0%sy, 0.0%ni, 91.2%id, 0.0%wa, 0.3%hi, 0.0%si, 0.0%st  
Mem: 1002232k total, 961444k used, 40788k free, 211248k buffers  
Swap: 916472k total, 0k used, 916472k free, 420348k cached  
  
  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND  
  791 root        20   0 54572  27m 8032 S   3.9   2.8   1:06.91 Xorg  
 1371 frank      20   0 46276  13m 9924 S   2.7   1.4   0:30.67 gnome-terminal  
 1762 frank      20   0 389m 136m 28m S   2.6  14.0   7:27.26 firefox-bin  
    6 root         0   0     0   0   0 S   0.2   0.0   0:00.80 events/0  
   18 root        20   0     0   0   0 S   0.2   0.0   0:16.18 ata/0  
 1086 root        20   0 3616 1320 1144 S   0.2   0.1   0:30.94 hald-addon-stor  
 1811 frank      20   0 66520  17m 12m S   0.2   1.8   0:00.32 plugin-containe  
 1989 frank      20   0 2548 1220  924 R   0.2   0.1   0:00.05 top  
    1 root        20   0 2808 1652 1204 S   0.0   0.2   0:00.86 init  
    2 root        20   0     0   0   0 S   0.0   0.0   0:00.00 kthreadd  
    3 root        RT   0     0   0   0 S   0.0   0.0   0:00.00 migration/0  
    4 root        20   0     0   0   0 S   0.0   0.0   0:00.57 ksoftirqd/0  
    5 root        RT   0     0   0   0 S   0.0   0.0   0:00.00 watchdog/0  
    7 root        20   0     0   0   0 S   0.0   0.0   0:00.00 cpuset  
    8 root        20   0     0   0   0 S   0.0   0.0   0:00.00 khelper  
    9 root        20   0     0   0   0 S   0.0   0.0   0:00.00 async/mgr  
   10 root        20   0     0   0   0 S   0.0   0.0   0:00.00 pm  
   11 root        20   0     0   0   0 S   0.0   0.0   0:00.02 sync_supers  
   12 root        20   0     0   0   0 S   0.0   0.0   0:00.03 bdi-default
```

# VSC training 2019/2020

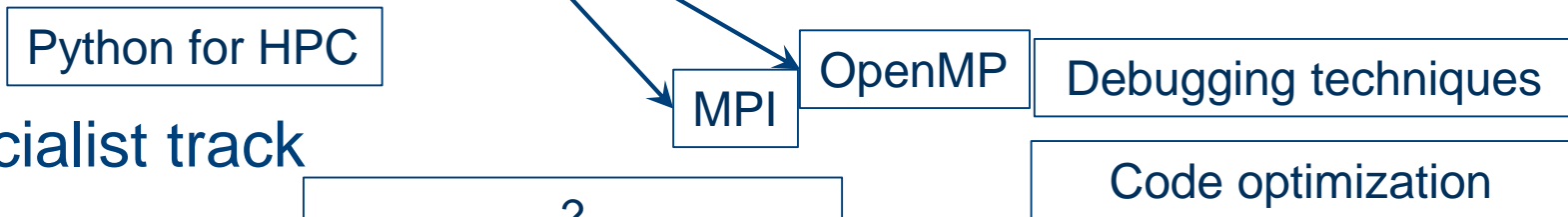
- Introductory



- Intermediate



- Advanced



- Specialist track



Info sessions:

- Containers
- Notebooks

Stay up-to-date <https://www.vscentrum.be/training>

# List of basic commands

- Managing files and directories:  
`cp, mv, rm, mkdir, rmdir, touch`
  - Displaying content of files  
`cat, more, less, head, tail`
  - Moving around  
`pwd, cd, ls`
  - Asking for help  
`help, man, info, whatis`
- + understanding of Linux directory tree (relative and absolute paths)  
+ understanding of file permissions

# Hands-on 4

