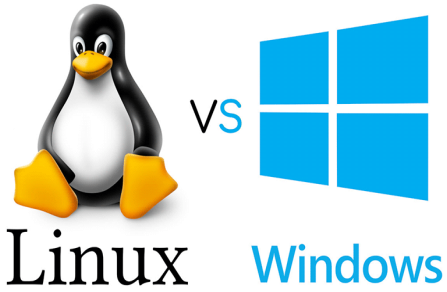
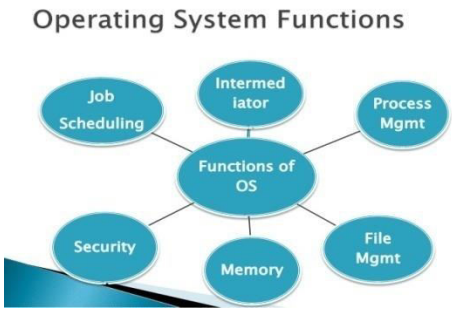
# CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

## Week2Day2 : Unix/Linux Operating Systems, Automating Linux Installation, Boot Terminology, Directory Structure, CLI, Shell Programming, Permissions





# Installing Linux

- **Linux provides the following methods for installing the Linux operating system:**
  - **Locally:**
    - From the DVD
    - From the hard disk
  - **Across the Network:**
    - Using the NFS
    - From the File Transfer Protocol (FTP) server
    - From the Hypertext Transfer Protocol (HTTP) server

# Automating the Linux Installation

- If you are installing linux on multiple computers, you can save considerable time by saving the answers to questions that are generally asked during installation.
- The method of automating the installation procedure of Linux is called kickstart installation.
- To perform a kickstart installation, perform the following steps:

  1. Create a kickstart file.
  2. Copy the kickstart file available on a floppy disk or on the network.
  3. Create the installation tree for network installation.
  4. Start the kickstart installation.

# Kickstart Configuration

- When you install Fedora Linux, a configuration file called *anaconda-ks.cfg* is automatically created in the /root directory.
- A Kickstart file to facilitate an automatic installation can be created by:
  - Editing this text file directly at the command -line
  - Creating the Kickstart file from scratch at the command-line
  - Editing an anaconda-ks.cfg file using the GUI-based Kickstart Configurator utility or by
  - Creating the Kickstart file from scratch using the Kickstart Configurator.

# DNF vs Yum

- The DNF package manager offers robust features for installing, updating, and removing packages.
- Dandified yum, better known as DNF, is a software package manager for RPM-based Linux distributions that installs, updates, and removes packages.
- It was first introduced in Fedora 18 in a testable state (i.e., tech preview), but it's been Fedora's default package manager since Fedora 22.

# DNF vs Yum

| S.No | DNF (Dandified YUM) | YUM (Yellowdog Updater, Modified) |
|---|---|---|
| 1 | DNF uses ‘libsolv’ for dependency resolution, developed and maintained by SUSE. | YUM uses the public API for dependency resolution |
| 2 | API is fully documented | API is not fully documented |
| 3 | It is written in C, C++, Python | It is written only in Python |
| 4 | DNF is currently used in Fedora, Red Hat Enterprise Linux 8 (RHEL), CentOS 8, OEL 8 and Mageia 6/7. | YUM is currently used in Red Hat Enterprise Linux 6/7 (RHEL), CentOS 8, OEL 6/7. |
| 5 | DNf supports various extensions | Yum supports only Python-based extension |

# DNF vs Yum

| | | |
|---|---|---|
| 6 | The API is well documented so it's easy to create new features | It is very difficult to create new features because the API is not properly documented. |
| 7 | The DNF uses less memory when synchronizing the metadata of the repositories. | The YUM uses excessive memory when synchronizing the metadata of the repositories. |
| 8 | DNF uses a satisfiability algorithm to solve dependency resolution (It's using a dictionary approach to store and retrieve package and dependency information). | Yum dependency resolution gets sluggish due to public API. |
| 9 | All performance is good in terms of memory usage and dependency resolution of repository metadata. | Overall performance is poor in terms of many factors. |
| 10 | DNF Update: If a package contains irrelevant dependencies during a DNF update process, the package will not be updated. | YUM will update a package without verifying this. |

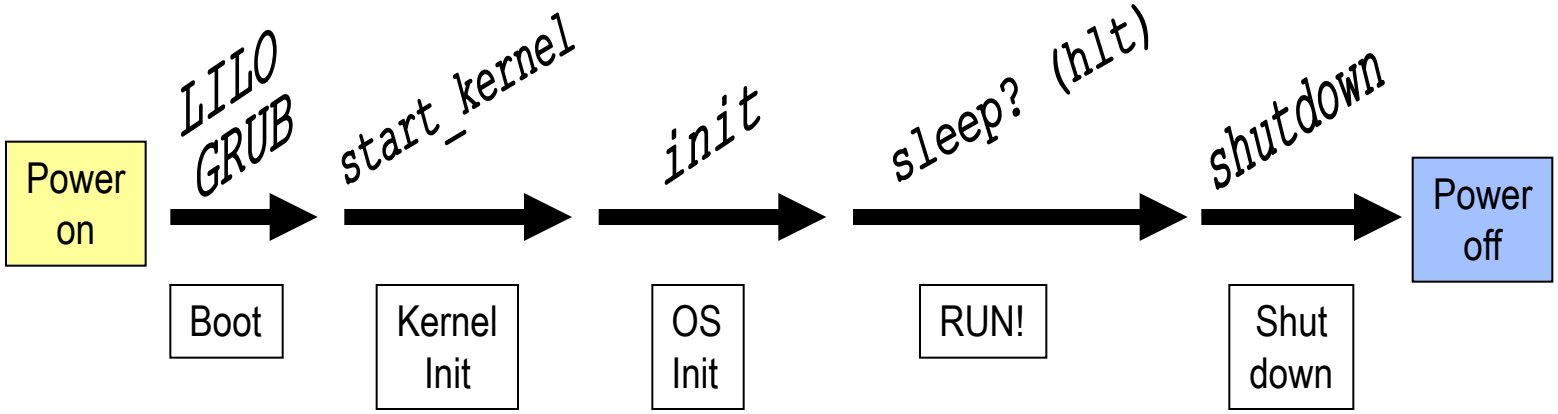
# DNF vs Yum

| S.No | DNF (Dandified YUM) | YUM (Yellowdog Updater, Modified) |
|---|---|---|
| 11 | If the enabled repository does not respond, dnf will skip it and continue the transaction with the available repositories. | If a repository is not available, YUM will stop immediately. |
| 12 | dnf update and dnf upgrade are equals. | It's different in yum |
| 13 | The dependencies on package installation are not updated | Yum offered an option for this behavior |
| 14 | Clean-up Package Removal: When removing a package, dnf automatically removes any dependency packages not explicitly installed by the user. | Yum didn't do this |
| 15 | Repo Cache Update Schedule: By default, ten minutes after the system boots, updates to configured repositories are checked by dnf hourly. This action is controlled by the system timer unit named "/usr/lib/systemd/system/dnf-makecache.timer". | Yum do this too. |

# DNF vs Yum

| | | |
|---|---|---|
| 16 | Kernel packages are not protected by dnf. Unlike Yum, you can delete all kernel packages, including one that runs. | Yum will not allow you to remove the running kernel |
| 17 | libsolv: for solving packages and reading repositories.<br><br>hawkey: hawkey, library providing simplified C and Python API to libsolv.<br><br>librepo: library providing C and Python (libcURL like) API for downloading linux repository metadata and packages.<br><br>libcomps: Libcomps is alternative for yum.comps library. It's written in pure C as library and there's bindings for python2 and python3 | Yum does not use separate libraries to perform this function. |
| 18 | DNF contains 29k lines of code | Yum contains 56k lines of code |
| 19 | DNF was developed by Ales Kozumplik | YUM was developed by Zdenek Pavlas, Jan Silhan and team members |

# System Lifecycle: Ups & Downs



# Boot Terminology

- **Loader:**
  - Program that moves bits from disk (usually) to memory and then transfers CPU control to the newly "loaded" bits (executable).
- **Bootloader / Bootstrap:**
  - Program that loads the "first program" (the kernel).
- **Boot PROM / PROM Monitor / BIOS:**
  - Persistent code that is "already loaded" on power-up.
- **Boot Manager:**
  - Program that lets you choose the "first program" to load.

# GRUB/LILO: LInux LOader

- A versatile boot manager that supports:
  - Choice of Linux kernels.
  - Boot time kernel parameters.
  - Booting non-Linux kernels.
  - A variety of configurations.
- Characteristics:
  - Lives in MBR or partition boot sector.
  - Has no knowledge of filesystem structure so…
  - Builds a sector “map file” (block map) to find kernel.
- /sbin/lilo – “map installer”.
  - /etc/lilo.conf is lilo configuration file.

# Example lilo.conf File

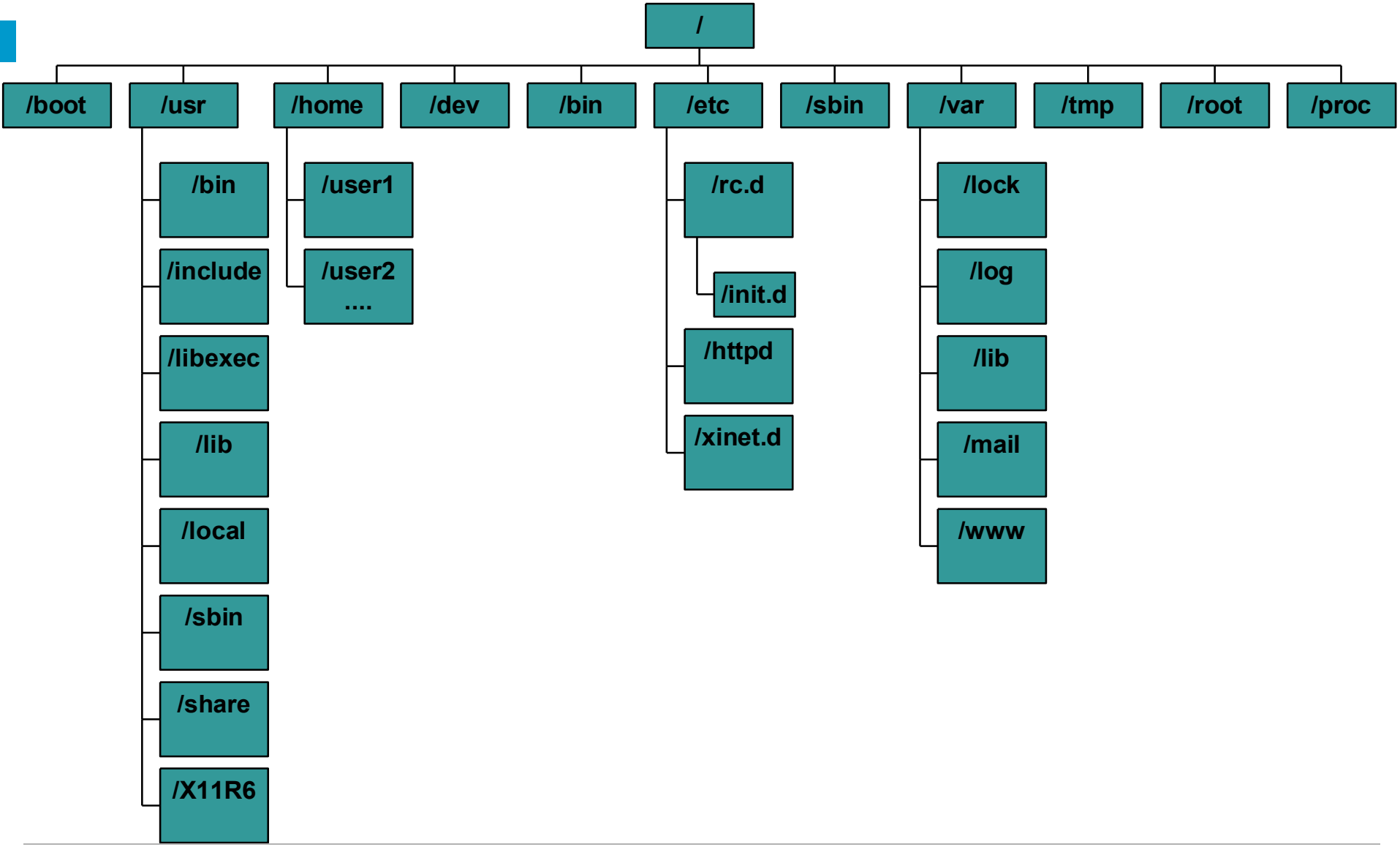
```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=linux

image=/boot/vmlinuz-2.2.12-20
        label=linux
        initrd=/boot/initrd-2.2.12-20.img
        read-only
        root=/dev/hda1
```

# /sbin/init

- Ancestor of all processes.
- Controls transitions between “runlevels”:
  - 0: shutdown
  - 1: single-user
  - 2: multi-user (no NFS)
  - 3: full multi-user
  - 5: X11
  - 6: reboot
- Executes startup/shutdown scripts for each runlevel.

# Linux Directory Tree



# Linux Directory Structure

## / (root directory)

- The root directory is the top of the filesystem hierarchy.
- The contents of the root filesystem must contain all the files needed
  - to boot,
  - restore,
  - recover,
  - and repair the system.

# Linux Directory Structure

## /bin

- bin directory contains several useful commands that are of use to both the system administrator as well as non-privileged users.
- These are required commands when no other file systems are mounted.
- It usually contains the shells like bash, csh, etc.... and commonly used commands like cp, mv, rm, cat, ls.
- contains essential system programs that must be available even if only the partition containing / is mounted.

# Linux Directory Structure

## /boot

- The /boot directory contains all files, except configuration files, needed to boot the operating system.
- The /boot stores data that is used before the kernel begins executing user-mode programs.
- This may include redundant (back-up) master boot records, sector/system map files, the kernel and other important boot files and data that is not directly edited by hand.

# Linux Directory Structure

## /dev

- The /dev directory contains file system entries which represent devices that are part of the system.
- Some common device files as well as their equivalent counterparts under Windows that you may wish to remember are:

- **/dev/ttyS0 (First communications port, COM1) First serial port (mice, modems).**
- **/dev/psaux ( PS/2) PS/2 mouse connection (mice, keyboards).**
- **/dev/lp0 (First printer port, LPT1) First parallel port (printers, scanners, etc).**

# Linux Directory Structure

## /etc

- This is the nerve center of your system,
- it contains all system related configuration files in here or in its sub-directories.
- A "configuration file" is defined as a local file used to control the operation of a program; it must be static and cannot be an executable binary.
- For this reason, it's a good idea to backup this directory regularly.
- It will definitely save you a lot of re-configuration later if you re-install or lose your current installation.
- Normally, no binaries should be or are located here.

# Linux Directory Structure

## /home

- /home directory contains a subdirectory for each user added to the system.
- you can write files, delete them, install programs, etc....
- Your home directory contains your personal configuration files, dot files (their name is preceded by a dot).
- Personal configuration files are usually 'hidden', if you want to see them, run ls with the -a switch.
- If there is a conflict between personal and system wide configuration files, the settings in the personal file will prevail.

# Linux Directory Structure

## /lib

- The /lib directory contains kernel modules and those shared library images (the C programming code library) needed to boot the system and run the commands in the root filesystem, ie. by binaries in /bin and /sbin.
- Libraries are readily identifiable through their filename extension of *.so. Windows equivalent to a shared library would be a DLL (dynamically linked library) file.
- They are essential for basic system functionality. Kernel modules (drivers) are in the subdirectory /lib/modules/'kernel-version'.

# Linux Directory Structure

## /mnt

- This is a generic mount point under which you mount your filesystems or devices.
- Mounting is the process by which you make a filesystem available to the system.
- After mounting your files will be accessible under the mount-point. This directory usually contains mount points or sub-directories where you mount your CD.

# Linux Directory Structure

## /opt

- This directory is reserved for all the software and add-on packages that are not part of the default installation.
- For example, StarOffice, Kylix, Netscape Communicator and WordPerfect packages are normally found here.
- All third party applications should be installed in this directory.
- Although most distributions neglect to create the directories /opt/bin, /opt/doc, /opt/include, /opt/info, /opt/lib, and /opt/man they are reserved for local system administrator use.

# Linux Directory Structure

## /proc-

- /proc is very special in that it is also a virtual filesystem.
- It's sometimes referred to as a process information pseudo-file system.
- It doesn't contain 'real' files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc).
- By altering files located in this directory you can even read/change kernel parameters (sysctl) while the system is running.
- The /proc filesystem is used to handle storage and retrieval of process information.

# Linux Directory Structure

## /usr

- /usr usually contains by far the largest share of data on a system.
- Contains all the user binaries, their documentation, libraries, header files, etc.... X and its supporting libraries can be found here. User programs like telnet, ftp, etc.... are also placed here

# Linux Directory Structure

## /var

- Contains variable data like system logging files, mail and printer spool directories, and transient and temporary files.
- Some portions of /var are not shareable between different systems.
- /var' contains variable data, i.e. files and directories the system must be able to write to during operation, whereas /usr should only contain static data.

# Linux Directory Structure

**/root**

- The /root directory is the home directory of the root user.

**•/sbin-**

- The /sbin directory contains utilities used for system administration.
- These utilities are executed after /usr is known to be mounted and there are no problems.

**•/tmp-**

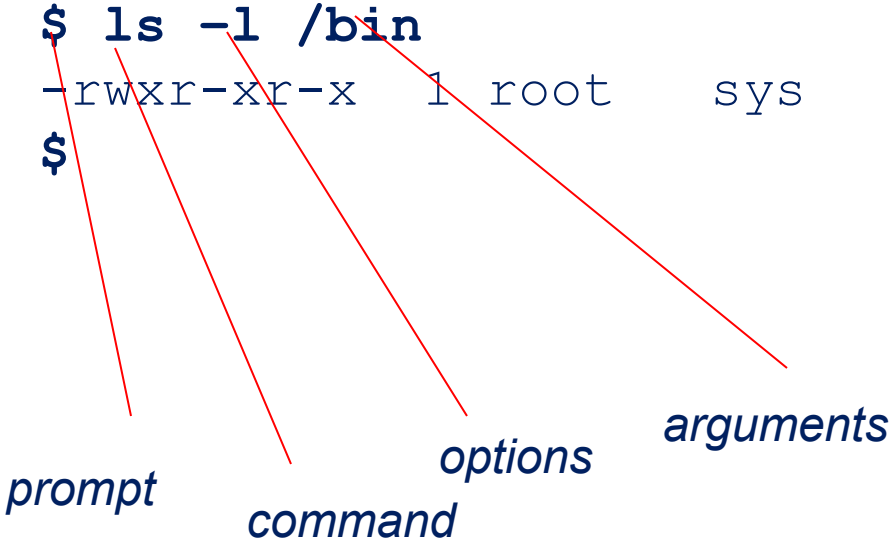
- This directory contains mostly files that are required temporarily.
- These files are lost if the system restarts.

# Simple Commands

- *simple command*: sequence of non blanks arguments separated by blanks or tabs.
- 1st argument (numbered zero) usually specifies the name of the command to be executed.
- Any remaining arguments:
  - Are passed as arguments to that command.
  - Arguments may be filenames, pathnames, directories or special options (up to command)
  - Special characters are interpreted by shell

# A simple example

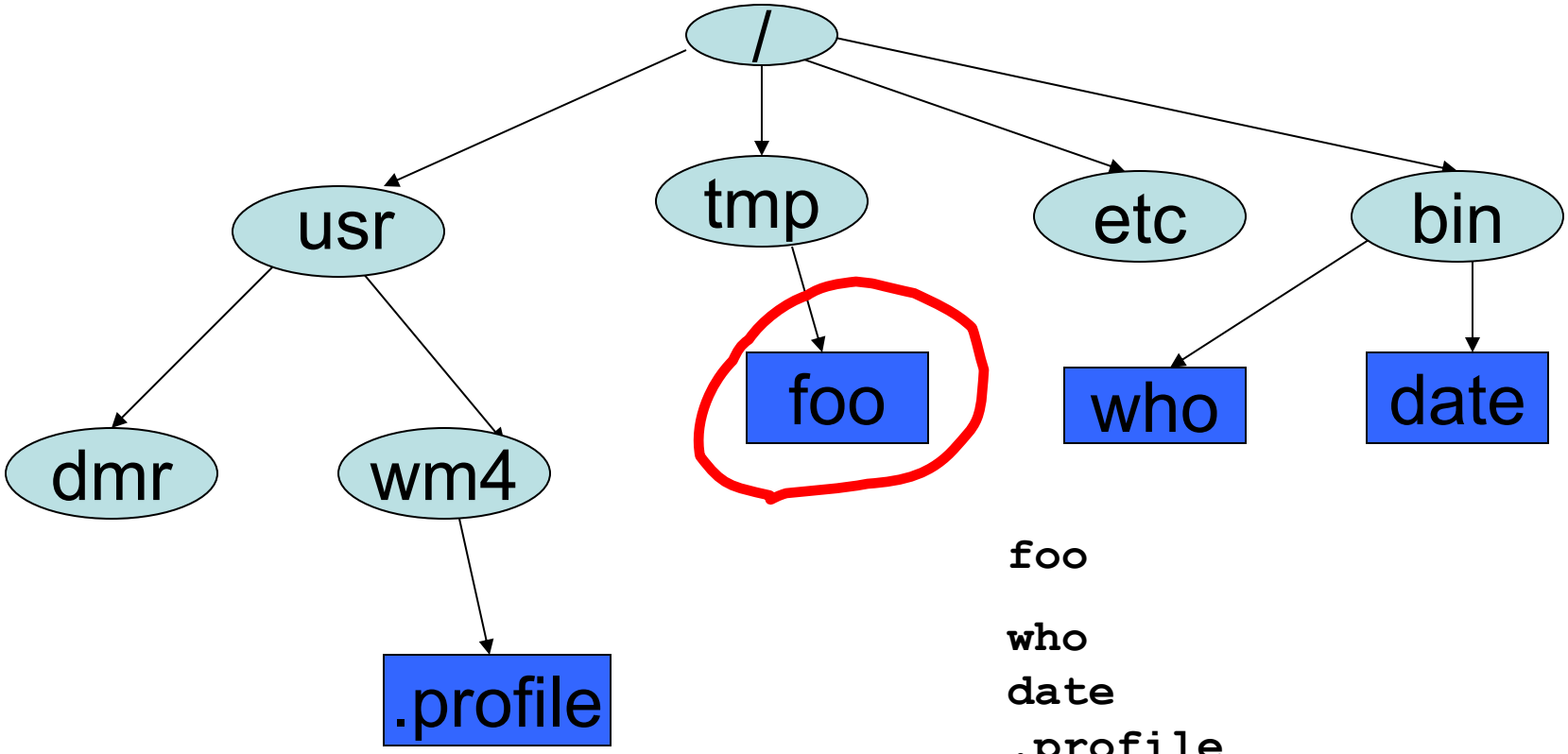
```
$ ls -l /bin
-rwxr-xr-x  1 root   sys    43234 jan 2  2023 date
$
```




- **Execute a basic command**
- **Parsing into command in arguments is called *splitting***

# Filenames

*A sequence of characters other than slash.* ***Case sensitive.***

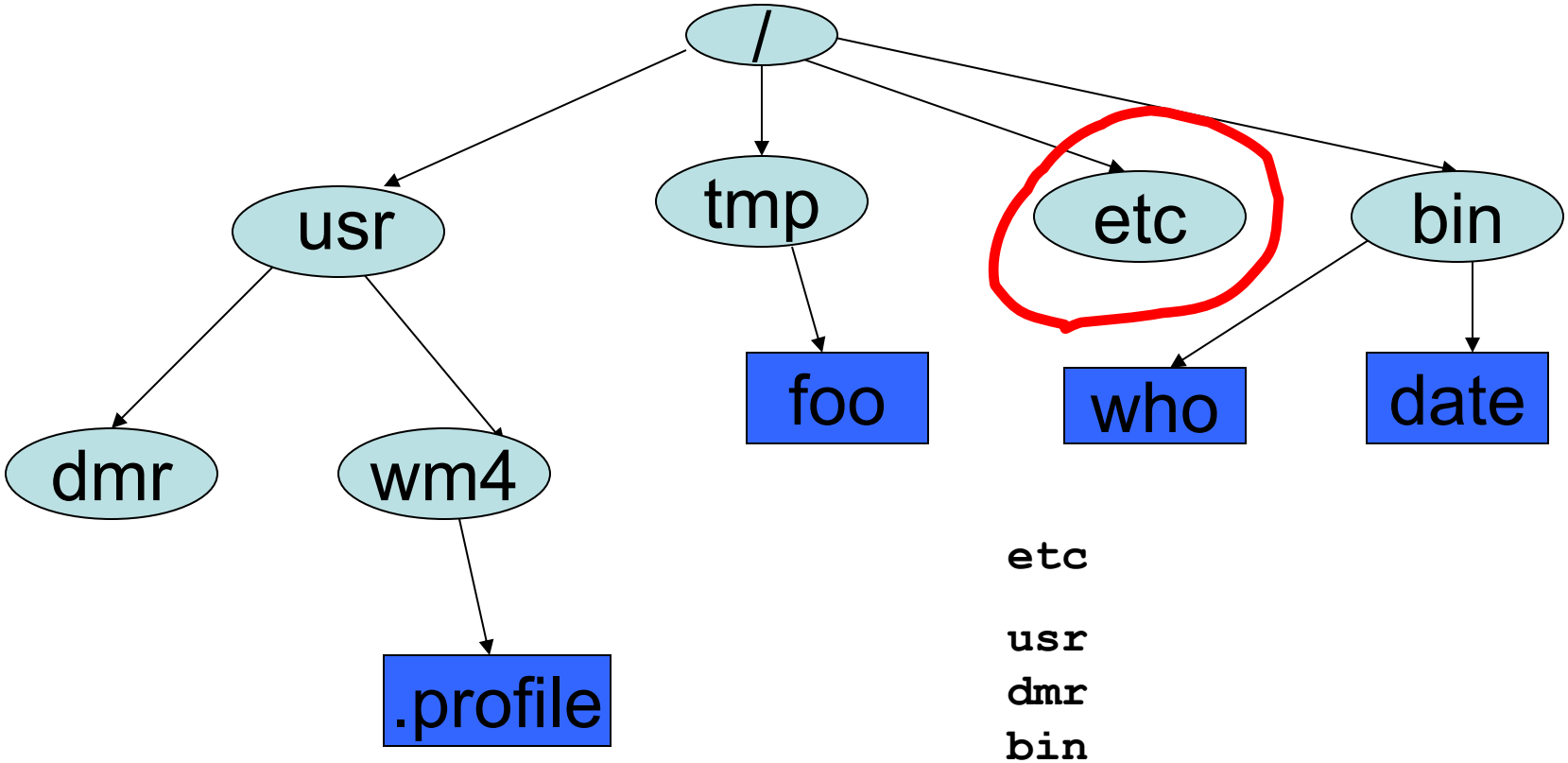



```
foo

who
date
.profile
```

# Directory

*Holds a set of files or other directories.* ***Case sensitive.***

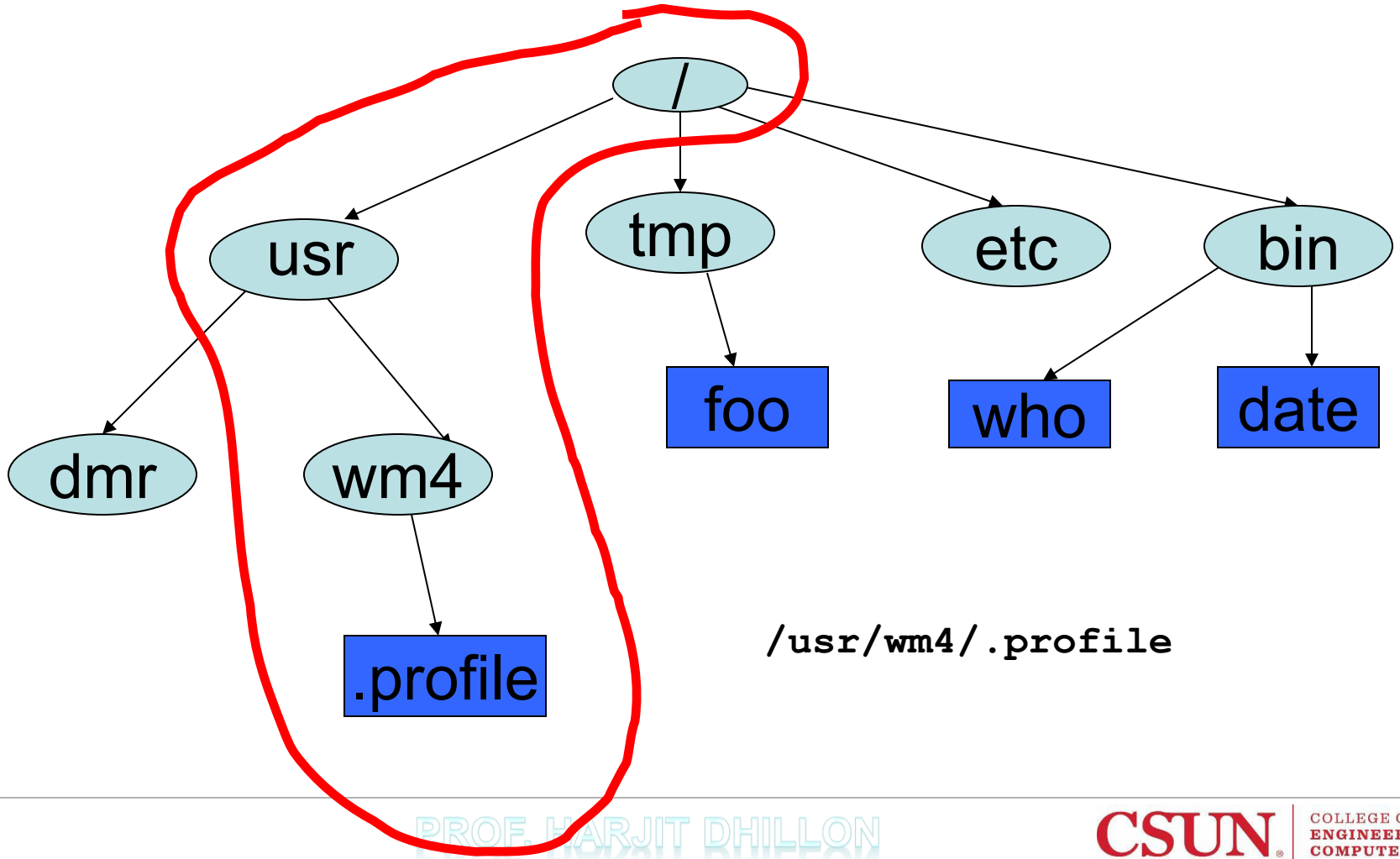



```
etc

usr
dmr
bin
```

# Pathname

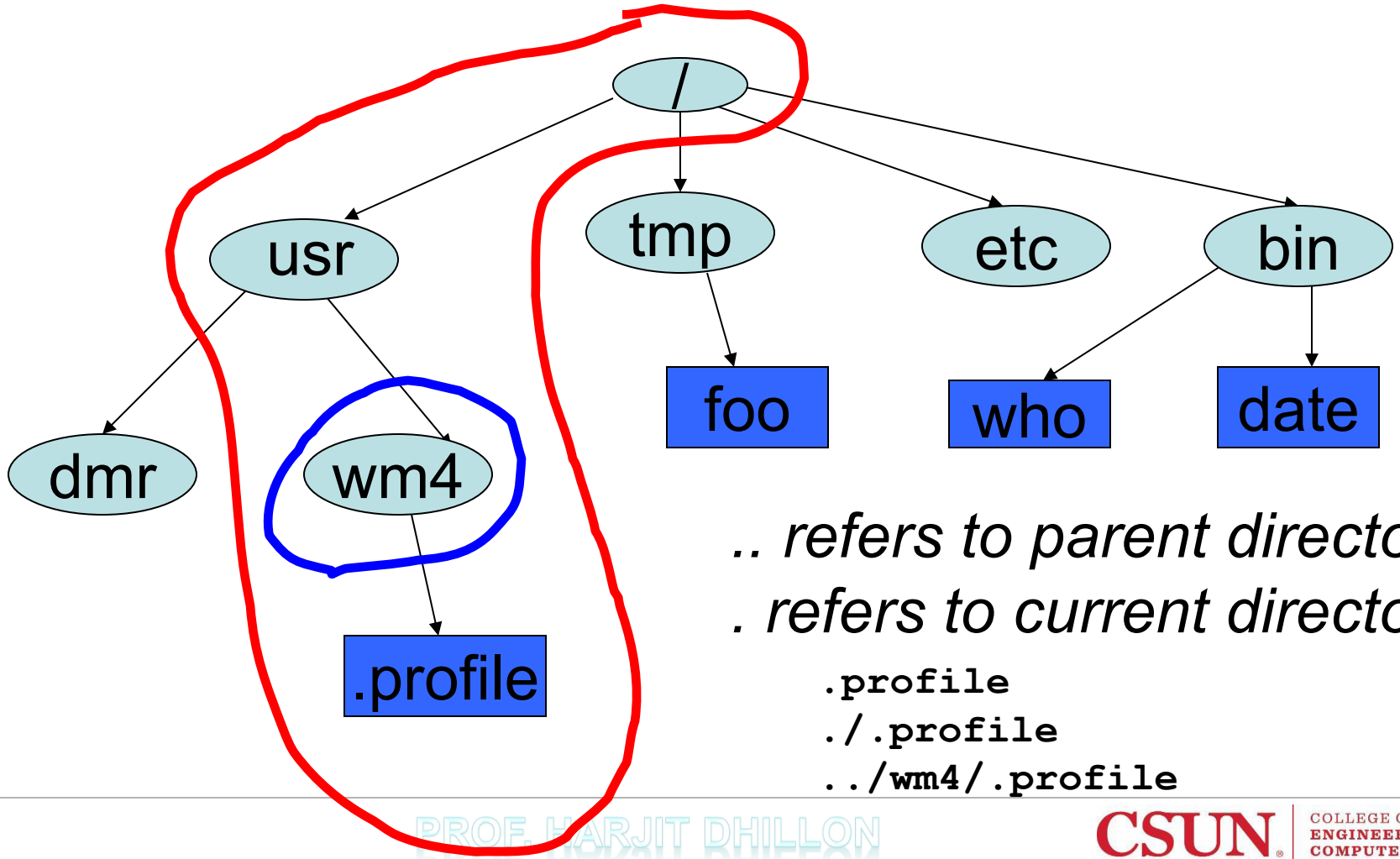
*A sequence of directory names followed by a simple filename, each separated from the previous one by a /*




`/usr/wm4/.profile`

# Relative Pathname

*A pathname relative to the working directory (as opposed to **absolute pathname**)*




*.. refers to parent directory*

*. refers to current directory*

```
.profile
./.profile
../wm4/.profile
```

# Files and Directories

- Files are just a sequence of bytes
  - No file types (data vs. executable)
  - No sections
  - Example of UNIX philosophy
- Directories are a list of files and status of the files:
  - Creation date
  - Attributes
  - etc.

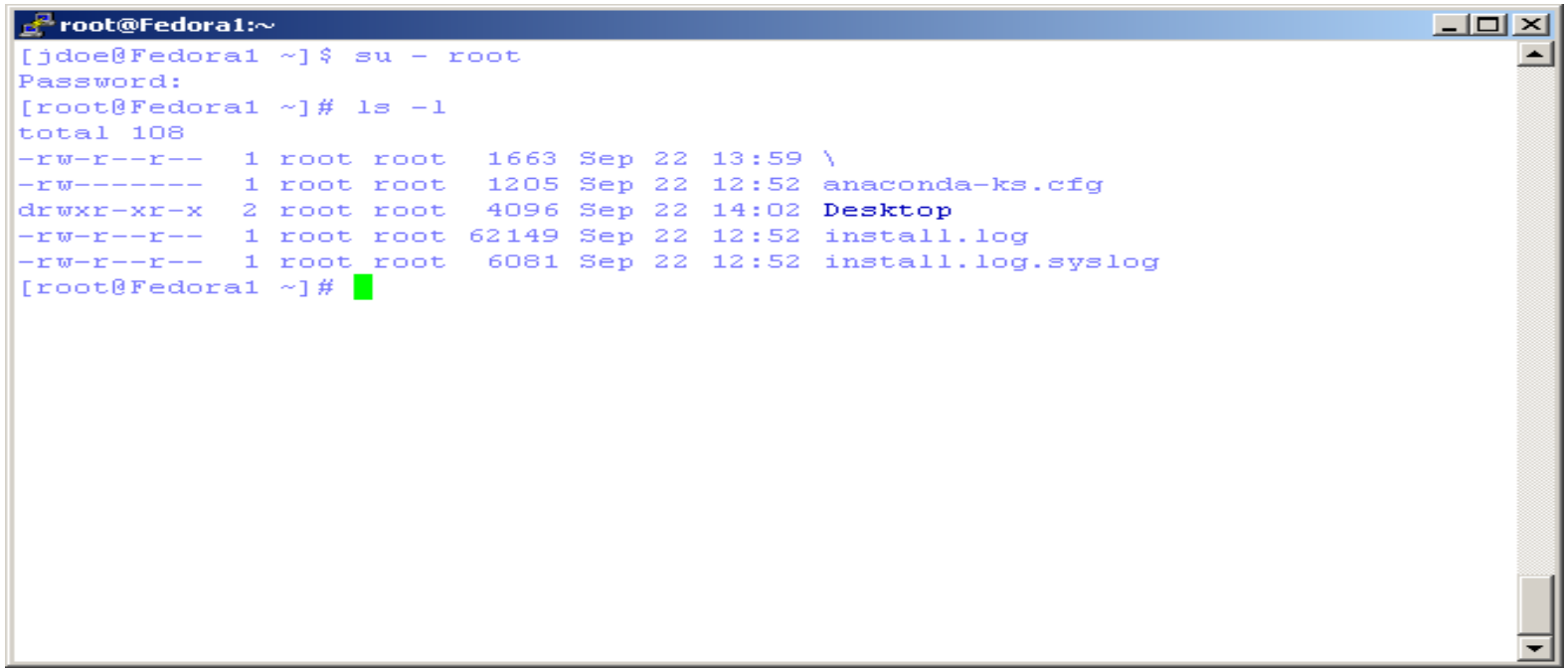
# Mounting File Systems

- When UNIX is started, the directory hierarchy corresponds to the file system located on a single disk called the *root device*.
- *Mounting* allows root to splice the root directory of a file system into the existing directory hierarchy.
- File systems created on other devices can be attached to the original directory hierarchy using the mount mechanism.
- Commands mount and umount manage

# Changing and Viewing Directories from the CLI

- When a user logs in to the Linux operating system the directory that they will start in is their home directory.
- Most users will have /home/*userid* as their home directory.
- The root user has /root as it's home directory.
- To view the contents of the current directory you can type:

  ls –l



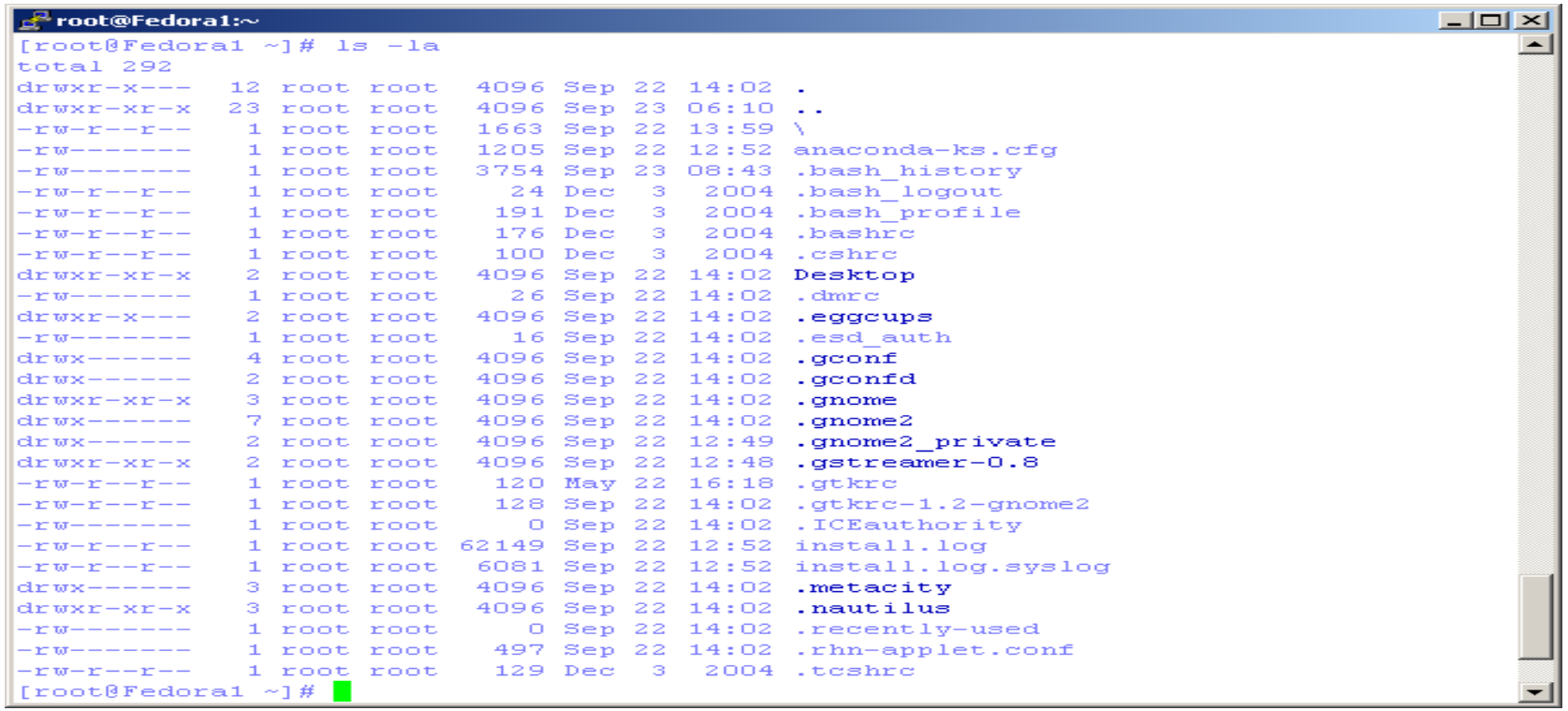
# Changing and Viewing Directories from the CLI

- The –l option says to give a long list which includes
  - file permissions,
  - owner, group,
  - size (in bytes),
  - date created,
  - and filename.

# Changing and Viewing Directories from the CLI

- You must have permission to read a directory before you can view it.
- You can view the hidden system files by including the –a option:

  ls –la



# Changing and Viewing Directories from the CLI

- You can view specific files by including the filename or a filter of a group of filenames. To view the file named install.log type:

```
ls –l install.log
```

- You can list all files that start with the letters install by using the wildcard character (*). Type:

```
ls –l install*
```

# Changing and Viewing Directories from the CLI

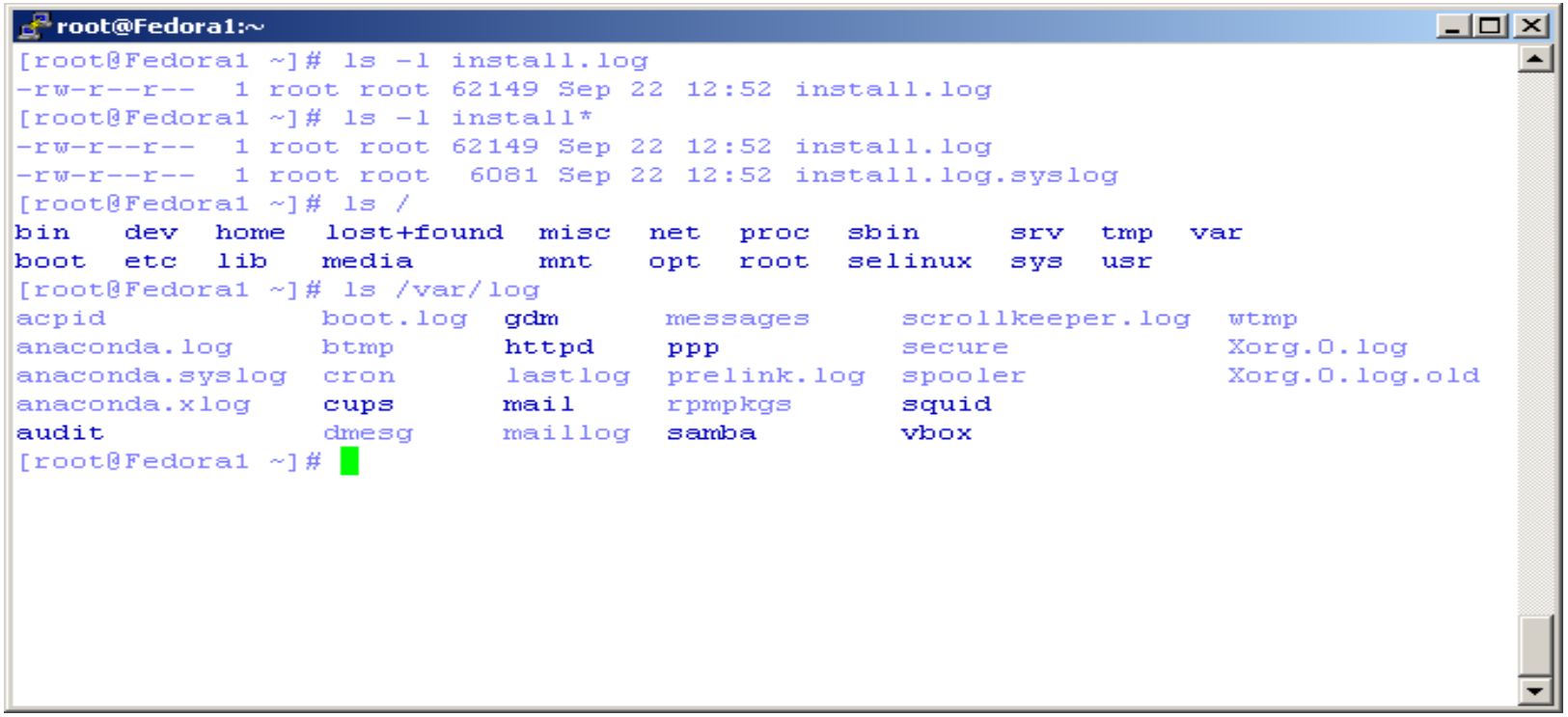
- You can include a path instead of just a filename to list. For example to view the contents of the / directory you can type:

  ls /

- Notice the –l option was not used so the directory will display only the file and directory names, not the properties.
- To view the contents of the /var/log directory type:

  ls /var/log

# Changing and Viewing Directories from the CLI



# Changing and Viewing Directories from the CLI 3

- Although you can view the contents of any directory from anywhere in the filesystem hierarchy you can also change to the individual directories.
- To change the current directory to the / directory type:

```
cd /
```

- To change to jsmith's Documents directory within his home directory type:

```
cd /home/jsmith/Documents
```

# Changing and Viewing Directories from the CLI 3

- There are shortcut commands you can also use. For example, you can use the shortcut .. change to the parent directory of the current directory by typing:

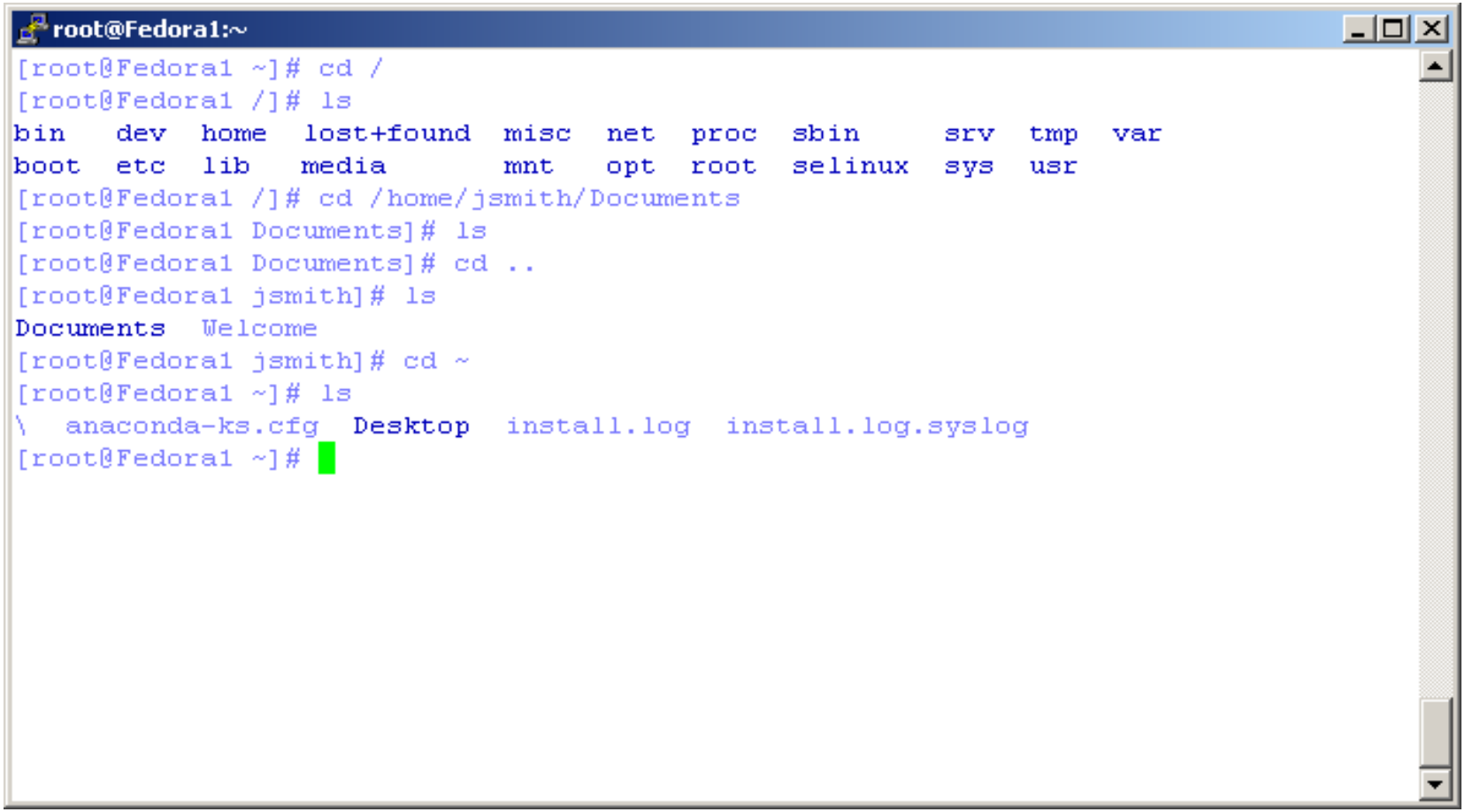
```
cd ..
```

- You can change to your home directory with the shortcut ~ by typing:

```
cd ~
```

- You can display the current directory by typing the pwd command.

# Changing and Viewing Directories from the CLI 3



# Creating a Directory and File from the CLI

- You create a directory with the mkdir command. To create a directory called scripts in the current directory type:

  mkdir scripts

- You can create a directory anywhere in the Linux filesystem hierarchy by including the full path in the directory name. To create a directory called cron in the /etc/skel directory type:

  mkdir /etc/skel/cron

- You must have permission to write in a directory to create a subdirectory in that directory.

# Creating a Directory and File from the CLI

- You already know you can create a file with vi and this is a preferred method if you are including content.
- If you only want the file and no content use the touch command.
- To create a file in the current directory called myFile type:

  touch myFile

  Notice the file size is 0.

# Creating a Directory and File from the CLI

```
root@Fedora1:~
[root@Fedora1 ~]# ls
\  anaconda-ks.cfg  Desktop  install.log  install.log.syslog
[root@Fedora1 ~]# mkdir scripts
[root@Fedora1 ~]# ls
\  anaconda-ks.cfg  Desktop  install.log  install.log.syslog  scripts
[root@Fedora1 ~]# ls /etc/skel
Documents  Welcome
[root@Fedora1 ~]# mkdir /etc/skel/cron
[root@Fedora1 ~]# ls /etc/skel
cron  Documents  Welcome
[root@Fedora1 ~]# touch myFile
[root@Fedora1 ~]# ls -l
total 120
-rw-r--r--  1 root root  1663 Sep 22 13:59 \
-rw-------  1 root root  1205 Sep 22 12:52 anaconda-ks.cfg
drwxr-xr-x  2 root root  4096 Sep 22 14:02 Desktop
-rw-r--r--  1 root root 62149 Sep 22 12:52 install.log
-rw-r--r--  1 root root  6081 Sep 22 12:52 install.log.syslog
-rw-r--r--  1 root root     0 Sep 23 09:01 myFile
drwxr-xr-x  2 root root  4096 Sep 23 08:59 scripts
[root@Fedora1 ~]#
```

# Remove a Directory or File from the CLI

- To remove a directory use the rmdir command. To remove the scripts directory you created in the current directory type:

```
rmdir scripts
```

- You must have write permission to remove a directory.

# Remove a Directory or File from the CLI

- To remove a directory that has other files within it use the rm command with the –r option (recursive).
- The rm command is used to delete files and, with the recursive option, directories and their contents as well.
- To remove the /etc/skel/cron directory and any files and directories that may be within it type:

  rm –r /etc/skel/cron

- To remove one or more files use the rm command. To remove the file Welcome from the /etc/skel directory type:

  rm /etc/skel/Welcome

# Remove a Directory or File from the CLI

```
root@Fedora1:~
[root@Fedora1 ~]# ls
\  anaconda-ks.cfg  Desktop  install.log  install.log.syslog  myFile  scripts
[root@Fedora1 ~]# rmdir scripts
[root@Fedora1 ~]# ls
\  anaconda-ks.cfg  Desktop  install.log  install.log.syslog  myFile
[root@Fedora1 ~]# ls /etc/skel
cron  Documents  Welcome
[root@Fedora1 ~]# rm -r /etc/skel/cron
rm: remove directory `/etc/skel/cron'? y
[root@Fedora1 ~]# ls /etc/skel
Documents  Welcome
[root@Fedora1 ~]# rm /etc/skel/Welcome
rm: remove regular file `/etc/skel/Welcome'? y
[root@Fedora1 ~]# ls /etc/skel
Documents
[root@Fedora1 ~]#
```

# Copying a file from the CLI

- Copy files with the cp command.
- When you copy a file you duplicate the file and it's contents to another file.
- Filenames must be unique so if you copy the file to the same directory it must have a different name.
- To copy /root/myFile to the / directory type:

  cp /root/myFile /myFile

- To copy /root/myFile to the / directory with the name myFile2 type:

  cp /root/myFile /myFile2

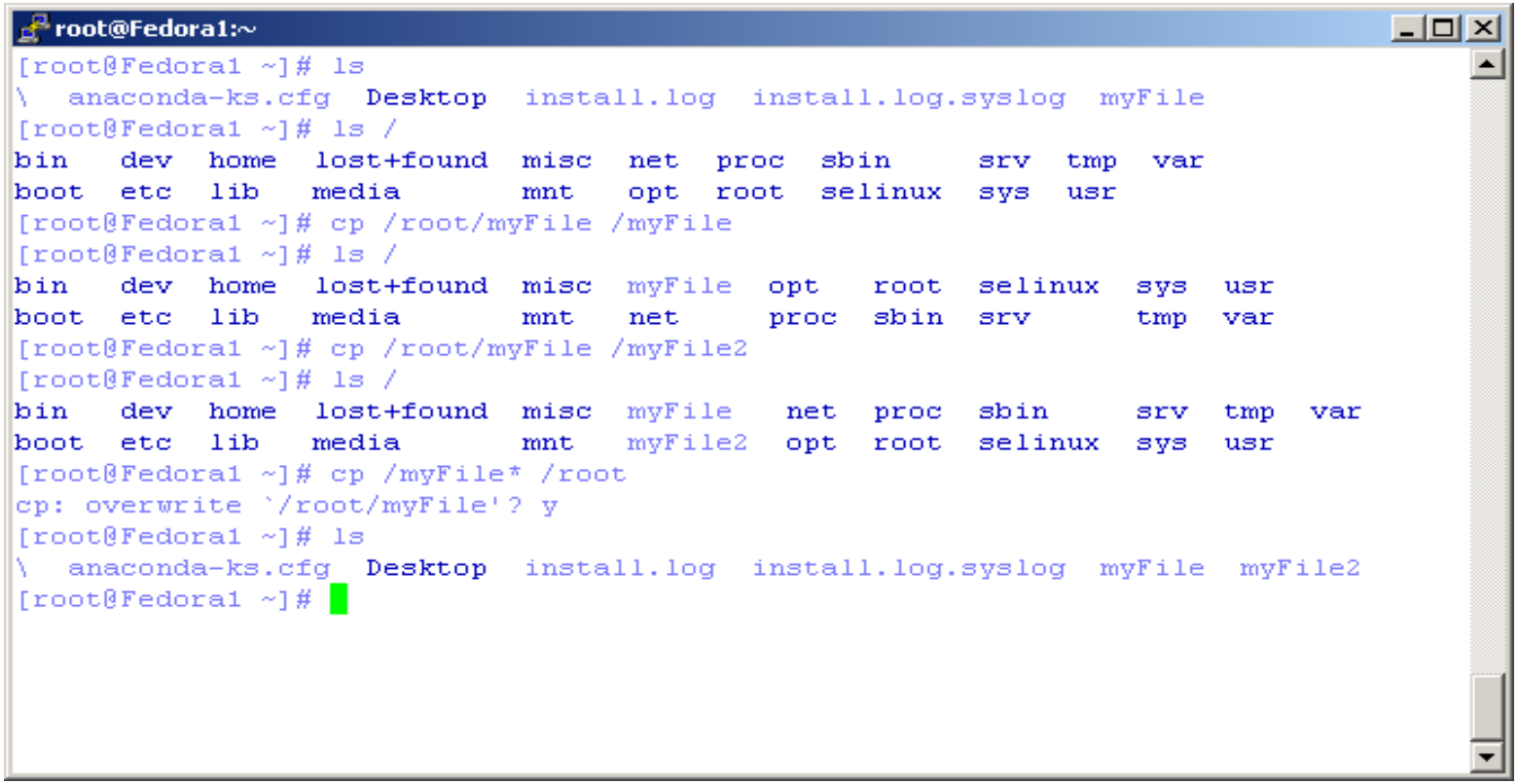
# Copying a file from the CLI

- You can use wildcards with filenames as you did with the ls command.
- To copy /myFile and /myFile2 to the /root directory type:

  cp /myfile* /root

- If you are replacing an existing file you will be prompted if it can be overwritten.

# Copying a file from the CLI



# Moving a File from the CLI

- To move a file use the mv command. When you move a file to another file in the same directory you are essentially renaming the file.
- To move /root/myFile to /root/myFile2 type:

```
mv /root/myFile /root/myFile2
```

# Moving a File from the CLI

- To move /root/myFile2 to /myFile type:

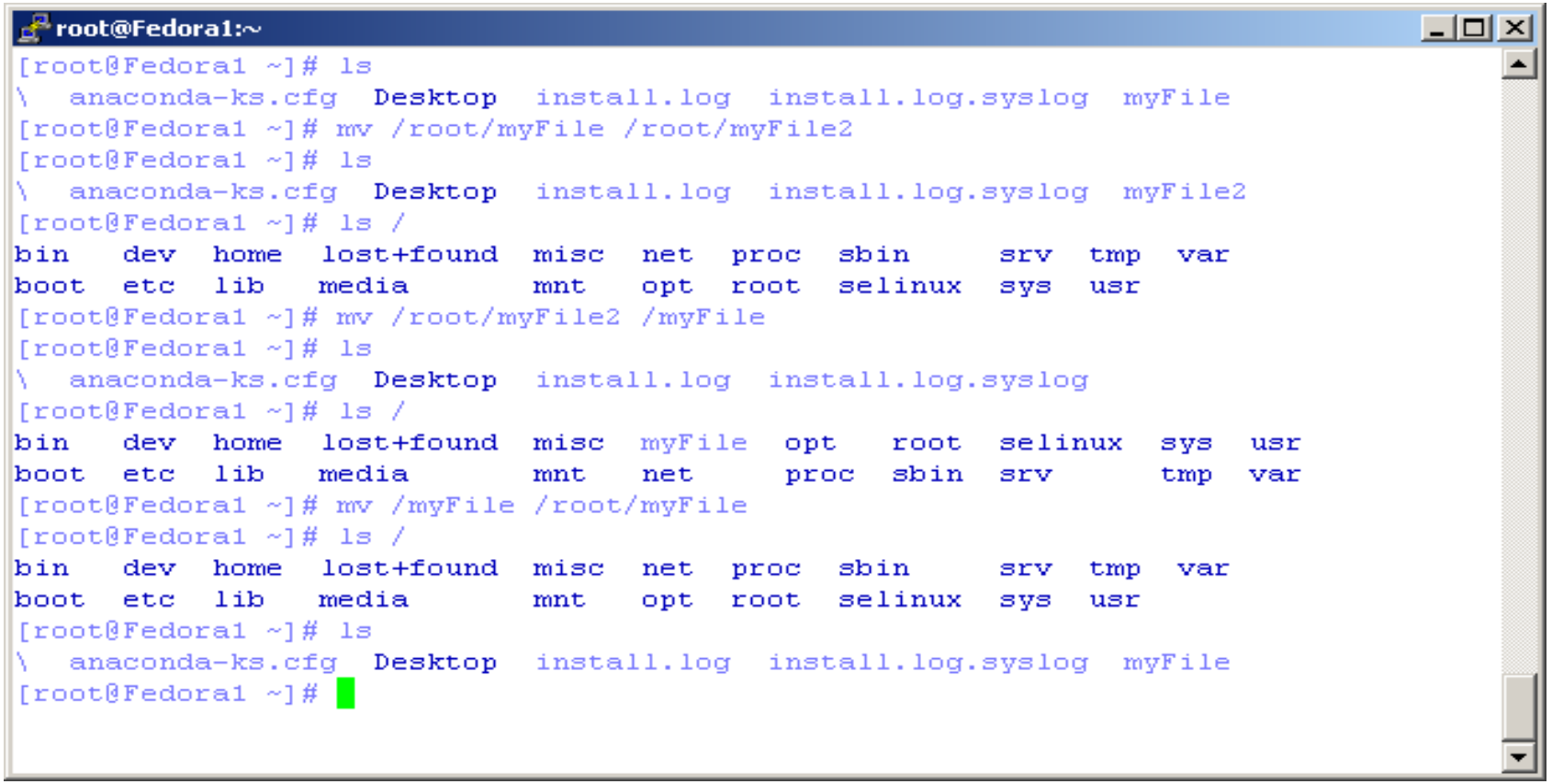
  mv /root/myFile2 /myFile

- To move /myFile to /root/myFile type:

  mv /myFile /root/myFile

You must have read and write permissions to move a file.

# Moving a File from the CLI



# Shell Programming

A *shell* is a program that acts as the interface between you and the Linux system,

- Allowing you to enter commands for the operating system to execute.
- it resembles the Windows command prompt
- Linux shells are very powerful.
- On Linux it's quite feasible to have multiple shells installed, with different users able to pick the one they prefer
- Linux is so modular, you can slot in one of the many different shells in use, although most of them are derived from the original Bourne shell.

# Shell Programming

- To check shell:
  - $ echo $SHELL (shell is a pre-defined variable)
- switch shell:
  - You can switch from one shell to another by just typing the name of the shell. exit return you back to previous shell.

# Shell Scripts

- Text files that contain sequences of UNIX commands , created by a text editor
- No compiler required to run a shell script, because the UNIX shell acts as an interpreter when reading script files
- After you create a shell script, you simply tell the OS that the file is a program that can be executed, by using the chmod command to change the files' mode to be executable
- Shell programs run less quickly than compiled programs, because the shell must interpret each UNIX command inside the executable script file before it is executed

# Commenting

- Lines starting with # are comments except the very first line where #! Indicates the location of the shell that will be run to execute the script.
- On any line characters following an unquoted # are considered to be comments and ignored.
- Comments are used to;
  - Identify who wrote it and when
  - Identify input variables
  - Make code easy to read
  - Explain complex code sections
  - Version control tracking
  - Record modifications

# Quote Characters

There are three different quote characters with different behaviour. These are:

“ : double quote, weak quote. If a string is enclosed in “ s the references to variables (i.e $*variable* ) are replaced by their values. Also back-quote and escape \ characters are treated specially.

‘ : single quote, strong quote. Everything inside single quotes are taken literally, nothing is treated as special.

` : back quote. A string enclosed as such is treated as a command and the shell attempts to execute it. If the execution is successful the primary output from the command replaces the string.

Example: echo “Today’s date is:” `date`

# Shell Programming

- **Programming features of the UNIX/LINUX shell:**
  - ***Shell variables***: **Your scripts often need to keep values in memory for later use. Shell variables are symbolic names that can access values stored in memory**
  - ***Operators***: **Shell scripts support many operators, including those for performing mathematical operations**
  - ***Logic structures***: **Shell scripts support sequential logic (for performing a series of commands), decision logic (for branching from one point in a script to another), looping logic (for repeating a command several times), and case logic (for choosing an action from several possible alternatives)**

# Variables

- Variables are symbolic names that represent values stored in memory
- Three different types of variables
  - Global Variables:
    - Environment and configuration variables, capitalized, such as HOME, PATH, SHELL, USERNAME, and PWD.
    - When you login, there will be a large number of global System variables that are already defined. These can be freely referenced and used in your shell scripts.
  - Local Variables
    - Within a shell script, you can create as many new variables as needed. Any variable created in this manner remains in existence only within that shell.

# A few global (environment) variables

| | |
|---|---|
| SHELL | Current shell |
| DISPLAY | Used by X-Windows system to identify the display |
| HOME | Fully qualified name of your login directory |
| PATH | Search path for commands |
| MANPATH | Search path for <man> pages |
| PS1 & PS2 | Primary and Secondary prompt strings |
| USER | Your login name |
| TERM | terminal type |
| PWD | Current working directory |

# Permissions

- **Every file or directory has a set of permissions that determines**
  - who can access and
  - what they can do to the file.
- **Permissions are represented by the characters r, w, and x which represent read, write, and execute.**
- **The permissions are slightly different when assigned to a file or a directory.**

# Permissions

- **The r, w, and X have a different meaning depending if it is assigned to a file or a directory.**
  - r
    - with a file means the file can be displayed or copied,
    - with a directory means it can be listed.
  - w
    - with a file means the file can be changed,
    - with a directory means it can be created or deleted.
  - x
    - means a file can be executed (script or program),
    - with a directory allows files within it to be copied or moved.

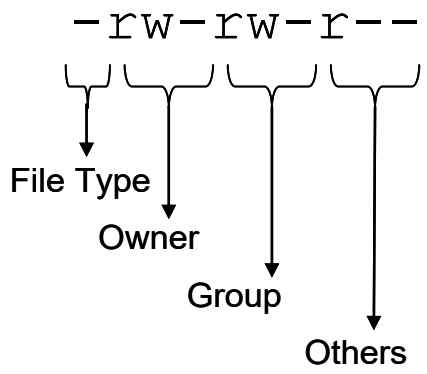
# Permissions

- The first item, which specifies the file type, can show one of the following:
  - d — a directory
  - -(dash) — a regular file (rather than directory or link)
  - l — a symbolic link to another program or file elsewhere on the system
- Beyond the first item, in each of the following three sets, you will see one of the following:
  - r — file can be read
  - w — file can be written to
  - x — file can be executed (if it is a program)

# Permissions

- For example:
- -rwx------ indicates a file that has read, write, and execute permissions for the owner only.
- -r-xr-xr-x indicates a file that has read and execute permissions for the owner, group, or anyone else.
- drwxr-x—x indicates a directory that allows read, write, and execute permission for the owner, read and execute for the group, and execute only for everyone else.

```
-rw-rw-r--    1 dengle   dengle          0 Sep 15 15:59 mydocument
drwxrwxr-x    2 dengle   dengle       4096 Sep 15 15:59 my-files
```



# Changing Permissions with the CLI

## The chmod Command

- **Think of these settings as a kind of shorthand when you want to change permissions with chmod, because all you really have to do is remember a few symbols and letters with the chmod command.**
- **Here is a list of what the shorthand represents:**
- **Identities**
  - u — the user who owns the file (that is, the owner)
  - g — the group to which the user belongs
  - o — others (not the owner or the owner's group)
  - a — everyone or all (u, g, and o)

# Changing Permissions with the CLI

## The chmod Command

- **Permissions**
  - r — read access
  - w — write access
  - x — execute access
- **Actions**
  - + — adds the permission
  - - — removes the permission
  - = — makes it the only permission

# Changing Permissions with the CLI

## The chmod Command

- Here are some common examples of settings that can be used with chmod:
- g+w — adds write access for the group
- o-rwx — removes all permissions for others
- u+x — allows the file owner to execute the file
- a+rw — allows everyone to read and write to the file
- ug+r — allows the owner and group to read the file
- g=rx — allows only the group to read and execute (not write)
- By adding the -R option, you can change permissions for entire directory trees.

# Changing Permissions with the CLI

- Changing Permissions With Numbers
  - Remember the reference to the shorthand method of chmod? Here is another way to change permissions, although it may seem a little complex at first.
  - Go back to the original permissions for sneakers.txt:
  - -rw-rw-r-- 1 sam sam 150 Mar 19 08:08 sneakers.txt
- Each permission setting can be represented by a numerical value:

```
» r = 4
» w = 2
» x = 1
» - = 0
```

# Changing Permissions with the CLI

- Changing Permissions With Numbers
- Here is a list of some common settings, numerical values and their meanings:
- -rw------- (600) — Only the owner has read and write permissions.
- -rw-r--r-- (644) — Only the owner has read and write permissions; the group and others have read only.
- -rwx------ (700) — Only the owner has read, write, and execute permissions.

# Changing Permissions with the CLI

- Changing Permissions With Numbers
- Here is a list of some common settings, numerical values and their meanings:
- -rwxr-xr-x (755) — The owner has read, write, and execute permissions; the group and others have only read and execute.
- -rwx--x--x (711) — The owner has read, write, and execute permissions; the group and others have only execute.
- -rw-rw-rw- (666) — Everyone can read and write to the file. (Be careful with these permissions.)
- -rwxrwxrwx (777) — Everyone can read, write, and execute. (Again, this permissions setting can be hazardous

# Changing the Owner of a File or Directory

You may want to change the owner of a file so someone else can be responsible for the file.

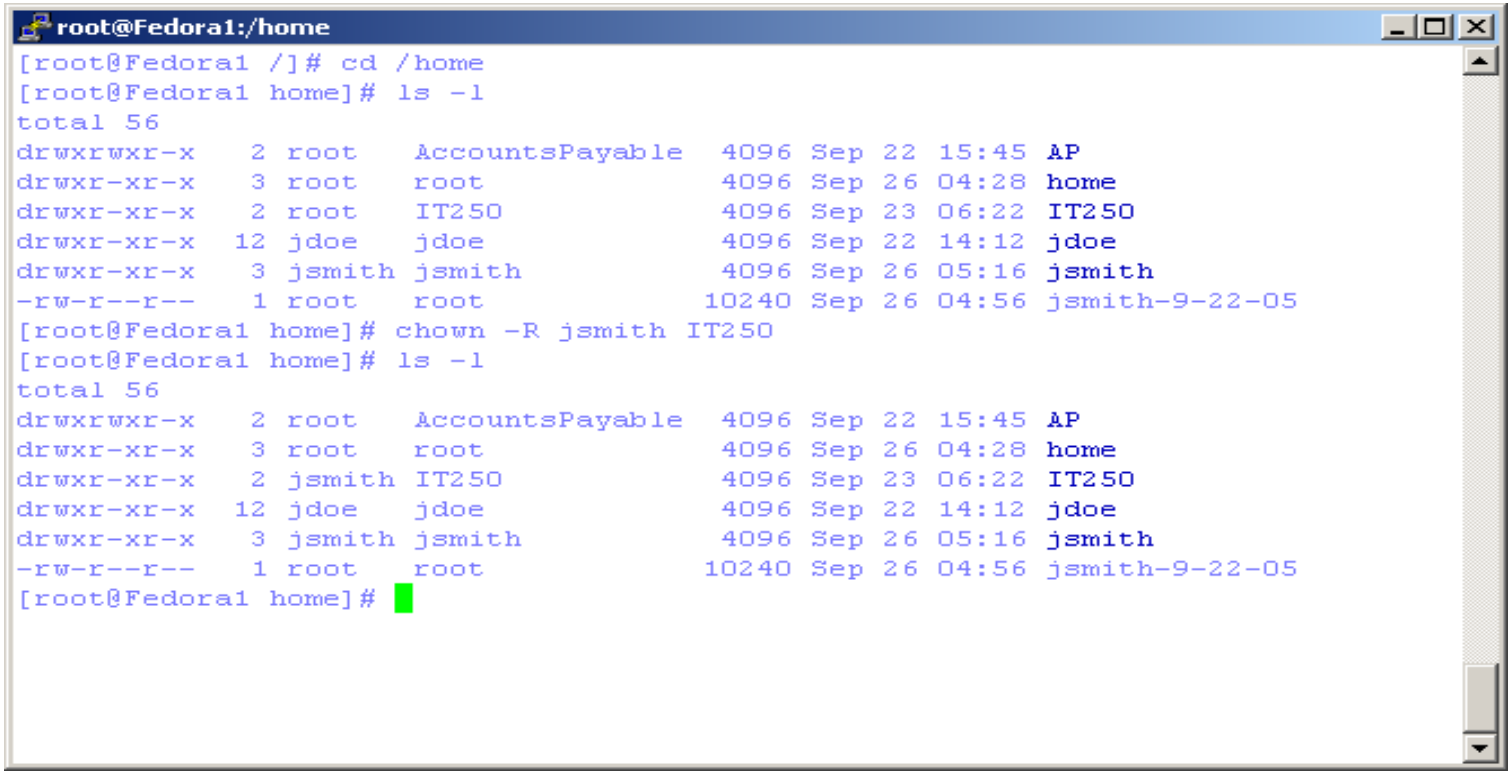
For example, the IT250 directory is owned by root but the root user has decided jsmith should be the owner.

To change ownership you can use chown. The command to the right:

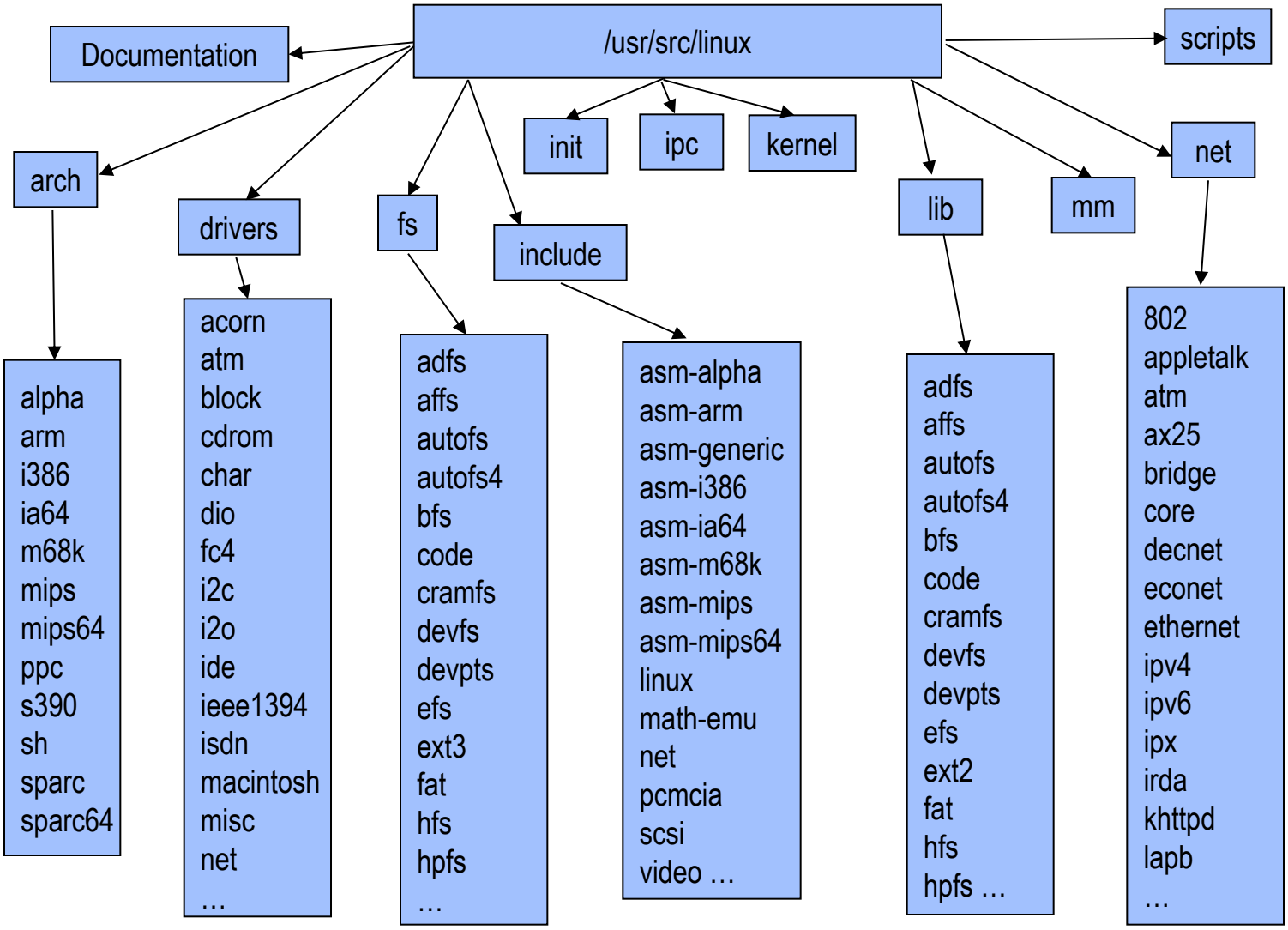
```
chown –R jsmith IT250
```

uses the –R option to act recursively on the IT250 directory and make jsmith owner of all the files contained within IT250 as well.

# Changing the Owner of a File or Directory



# Linux Source Tree Layout



# linux/arch

- Subdirectories for each current port.
- Each contains kernel, lib, mm (memory management), boot and other directories whose contents override code stubs in architecture independent code.
- lib contains highly-optimized common utility routines such as memcpy, checksums, etc.
- arch as of 2.4:
  - alpha, arm, i386, ia64, m68k, mips, mips64.
  - ppc, s390, sh, sparc, sparc64.

# linux/drivers

- Largest amount of code in the kernel tree (~1.5M).
- device, bus, platform and general directories.
- drivers/char – n_tty.c is the default line discipline.
- drivers/block – elevator.c, genhd.c, linear.c, ll_rw_blk.c, raidN.c.
- drivers/net –specific drivers and general routines Space.c and net_init.c.
- drivers/scsi – scsi_*.c files are generic; sd.c (disk), sr.c (DVD-ROM), st.c (tape), sg.c (generic).
- General:
  - cdrom, ide, isdn, parport, pcmcia, pnp, sound, telephony, video.
- Buses – fc4, i2c, nubus, pci, sbus, tc, usb.
- Platforms – acorn, macintosh, s390, sgi.

# linux/fs

- Contains:
  - virtual filesystem (VFS) framework.
  - subdirectories for actual filesystems.
- vfs-related files:
  - exec.c, binfmt_*.c - files for mapping new process images.
  - devices.c, blk_dev.c – device registration, block device support.
  - super.c, filesystems.c.
  - inode.c, dcache.c, namei.c, buffer.c, file_table.c.
  - open.c, read_write.c, select.c, pipe.c, fifo.c.
  - fcntl.c, ioctl.c, locks.c, dquot.c, stat.c.

# linux/include

- include/asm-*:
  - Architecture-dependent include subdirectories.
- include/linux:
  - Header info needed both by the kernel and user apps.
  - Usually linked to /usr/include/linux.
  - Kernel-only portions guarded by #ifdefs
    - #ifdef __KERNEL__
    - /* kernel stuff */
    - #endif
- Other directories:
  - math-emu, net, pcmcia, scsi, video.

# linux/init

- Just two files: version.c, main.c.
- version.c – contains the version banner that prints at boot.
- main.c – architecture-independent boot code.
- start_kernel is the primary entry point.

# linux/ipc

- System V IPC facilities.
- If disabled at compile-time, util.c exports stubs that simply return –ENOSYS.
- One file for each facility:
  - sem.c – semaphores.
  - shm.c – shared memory.
  - msg.c – message queues.

# linux/kernel

- The core kernel code.
- sched.c – “the main kernel file”:
  - scheduler, wait queues, timers, alarms, task queues.
- Process control:
  - fork.c, exec.c, signal.c, exit.c etc…
- Kernel module support:
  - kmod.c, ksyms.c, module.c.
- Other operations:
  - time.c, resource.c, dma.c, softirq.c, itimer.c.
  - printk.c, info.c, panic.c, sysctl.c, sys.c.

# linux/lib

- **kernel code cannot call standard C library routines.**
- **Files:**
  - **brlock.c – “Big Reader” spinlocks.**
  - **cmdline.c – kernel command line parsing routines.**
  - **errno.c – global definition of errno.**
  - **inflate.c – “gunzip” part of gzip.c used during boot.**
  - **string.c – portable string code.**
    - Usually replaced by optimized, architecture-dependent routines.
  - **vsprintf.c – libc replacement.**

# linux/mm

- Paging and swapping:
  - swap.c, swapfile.c (paging devices), swap_state.c (cache).
  - vmscan.c – paging policies, kswapd.
  - page_io.c – low-level page transfer.
- Allocation and deallocation:
  - slab.c – slab allocator.
  - page_alloc.c – page-based allocator.
  - vmalloc.c – kernel virtual-memory allocator.
- Memory mapping:
  - memory.c – paging, fault-handling, page table code.
  - filemap.c – file mapping.
  - mmap.c, mremap.c, mlock.c, mprotect.c.