UNIX File Types, File System Hierarchy and File Permissions

CS2600 Systems Programming

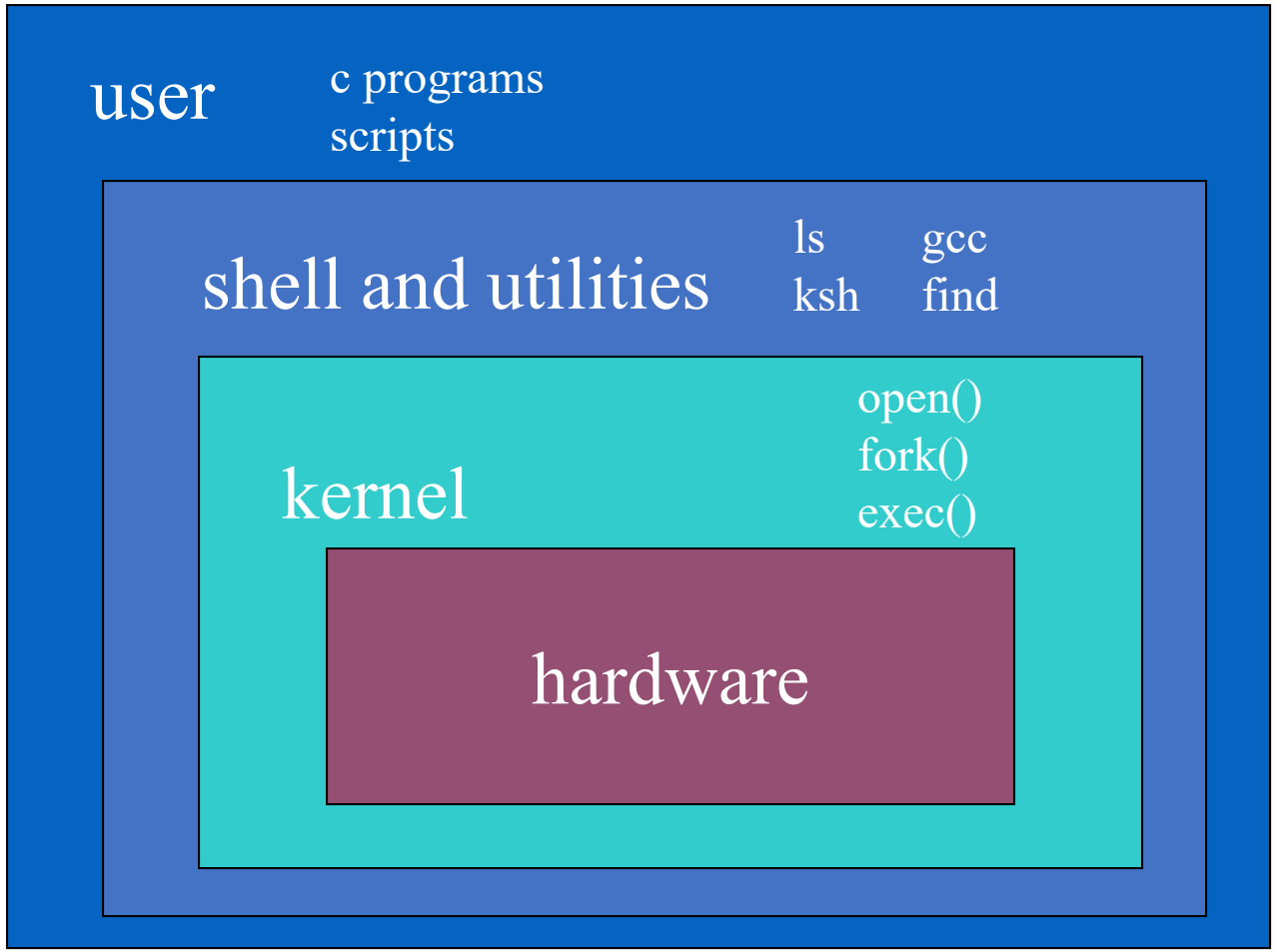
Dr. Amar Raheja

Lecture 3

Unix File Systems

- Topics Covered
  - Unix File Types
  - Unix File System Hierarchy Standard (FSH)
  - Unix File Permissions and how to change them

Unix System Structure



Two Primary Kernel Subsystems

- File system
  - Deals with all input and output
    - Includes files and terminals
    - Integration of storage devices
- Process management
  - Deals with programs and program interaction
    - How processes share CPU, memory and signals
    - Scheduling
    - Inter Process Communication (IPC)
    - Memory management
- UNIX variants have slightly different implementations of different subsystems.

Unix File Types (1 of 2)

- The UNIX filesystem contains several different types of files:
- Ordinary Files
  - Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
  - Always located within/under a directory file
  - Do not contain other files
- Directories
  - Branching points in the hierarchical tree
  - Used to organize groups of files
  - May contain ordinary files, special files or other directories
  - Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
  - All files are descendants of the root directory, (named /) located at the top of the tree.

Unix File Types (2 of 2)

- Special Files
  - Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Ouput (I/O) operations
  - Unix considers any device attached to the system to be a file - including your terminal:
    - By default, a command treats your terminal as the standard input file (stdin) from which to read its input
    - Your terminal/console is also treated as the standard output file (stdout) to which a command's output is sent
  - Two types of I/O: character and block
    - Usually only found under directories named /dev
- Pipes
  - UNIX allows you to link commands together using a pipe. (IPC)
  - The pipe acts a temporary file which only exists to hold data from one command until it is read by another

Finding File types using file

- Identifies the "type" of file. The command syntax is: file <filename>

```
Amars-MacBook-Pro:CS2600 amarraheja$ file *
a.out:          Mach-O 64-bit executable x86_64
data:           ASCII text
data1:          ASCII text
data2:          ASCII text
data3:          ASCII text
database.txt:   ASCII text
datafile.txt:   ASCII text
datafile.txt~:  ASCII text
datebook.txt:   ASCII text
foo:            ASCII text
foo.txt:        ASCII text
foo2:           ASCII text
foo3:           ASCII text
foo4:           ASCII text
hello.c:        c program text, ASCII text
hello.c~:       c program text, ASCII text
junk:           directory
test:           ASCII text
test2:          ASCII text
test2~:         ASCII text
Amars-MacBook-Pro:CS2600 amarraheja$
```

File Names (1 of 2)

- UNIX permits file names to use most characters, but avoid spaces, tabs and characters that have a special meaning to the shell, such as:
  - & ; ( ) | ? \ ' " ` [ ] { } < > $ - ! /
- Case Sensitivity: uppercase and lowercase are not the same! These are three different files:
  - NOVEMBER November november
  - Best Advice: Stick to lower case for every file name with occasional use of upper case
- Length: can be up to 256 characters
- Extensions: may be used to identify types of files
  - libc.a - *archive, library file*
  - program.c - *C language source file*
  - alpha2.f - *Fortran source file*
  - xwd2ps.o - *Object/executable code*
  - mygames.Z - *Compressed file*

File Names(2 of 2)

- Hidden Files: have names that begin with a dot (.)
  - For example: .cshrc , .login, .mailrc , .mwmrc
  - Mostly resource configuration files begin with a .
- Uniqueness: as children in a family, no two files with the same parent directory can have the same name. Files located in separate directories can have identical names.
- Reserved Filenames:

  / *the root directory (slash)*

  . *current directory (period)*

  .. *parent directory (double period)*

  ~ *your home directory (tilde)*

Pathnames

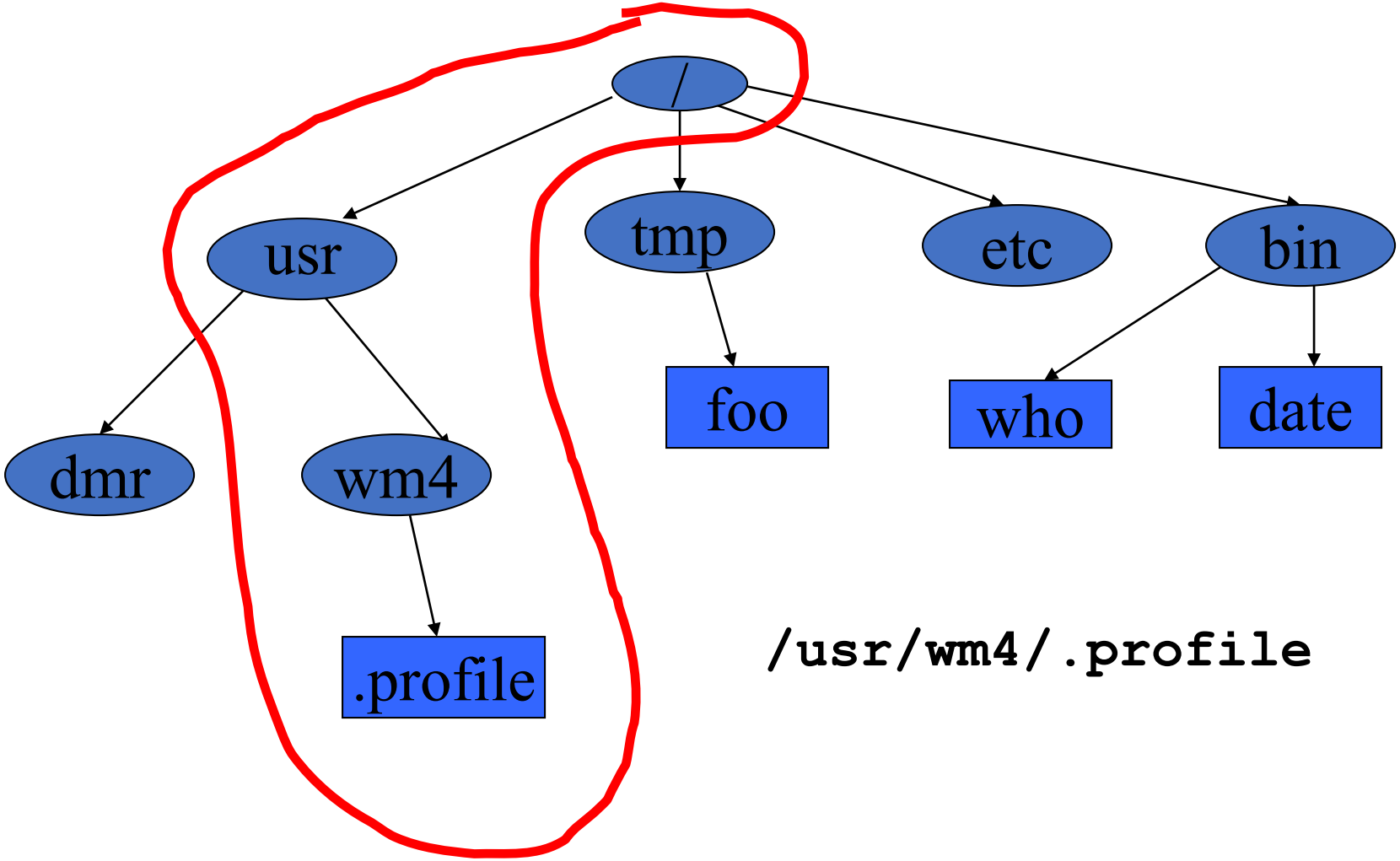
- Specify where a file is located in the hierarchically organized file system
- Must know how to use pathnames to navigate the UNIX file system
- Absolute Pathname: tells how to reach a file begining from the root; always begins with / (slash).
  - For example: /usr/local/doc/training/sample.f
- Relative Pathname: tells how to reach a file from the directory you are currently in (current or working directory); never begins with / (slash).
  - For example: training/sample.f ../bin ~/projects/report.001
- For example, if your current directory is /usr/home/raheja and you wanted to change to the directory /usr/home/amar, you could use either of these commands:

  cd ../amar - *relative pathname*

  cd /usr/home/raheja - *absolute pathname*

Pathname Example

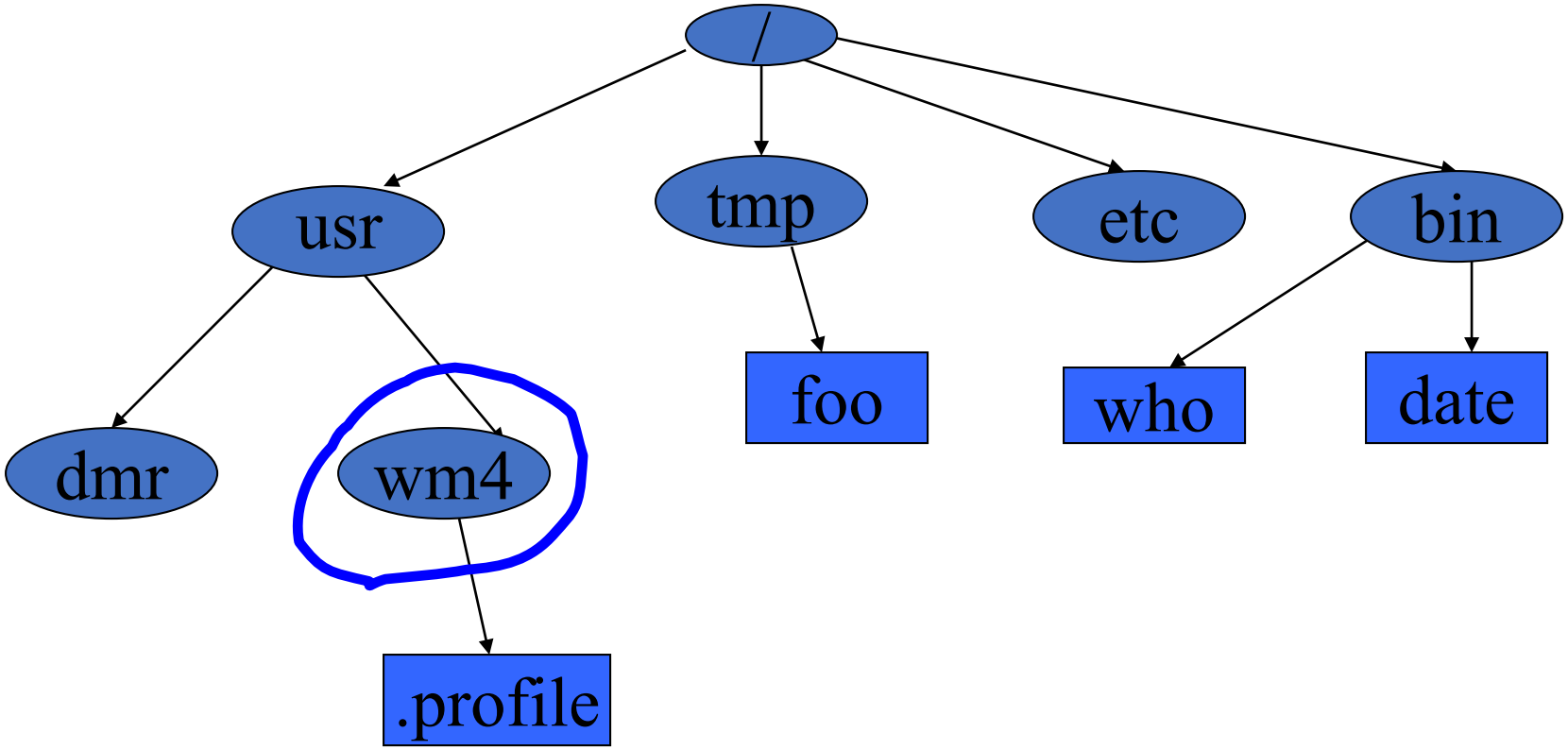
- A sequence of directory names followed by a simple filename, each separated from the previous one by a /



`/usr/wm4/.profile`

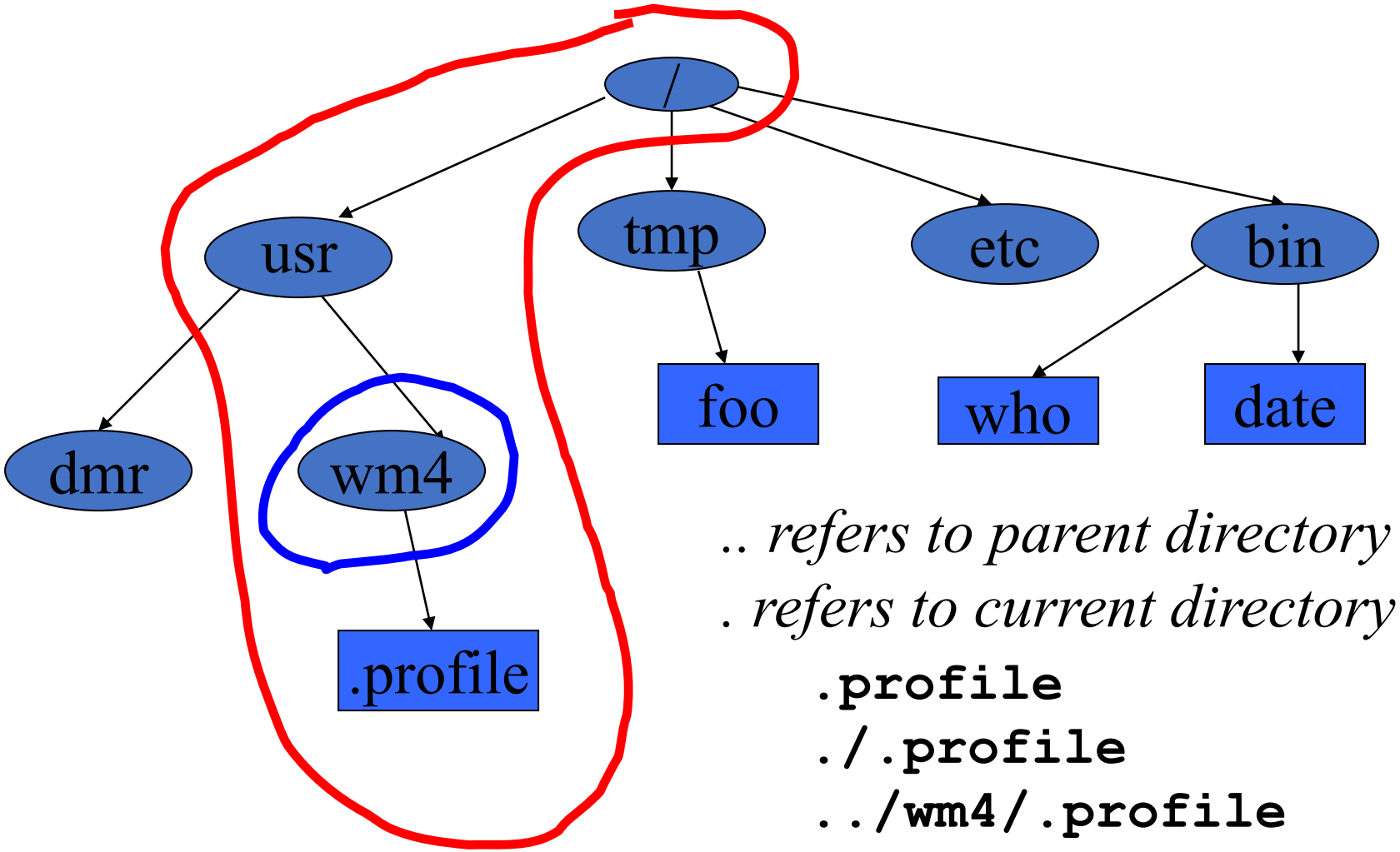
Definition: Current Working Directory

- A directory that you currently reside in, use command *pwd*



Definition: Relative Pathname

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



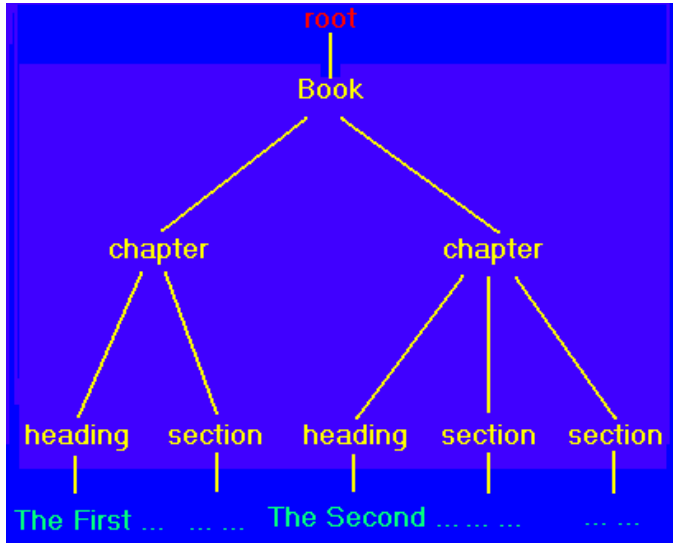
.. refers to parent directory

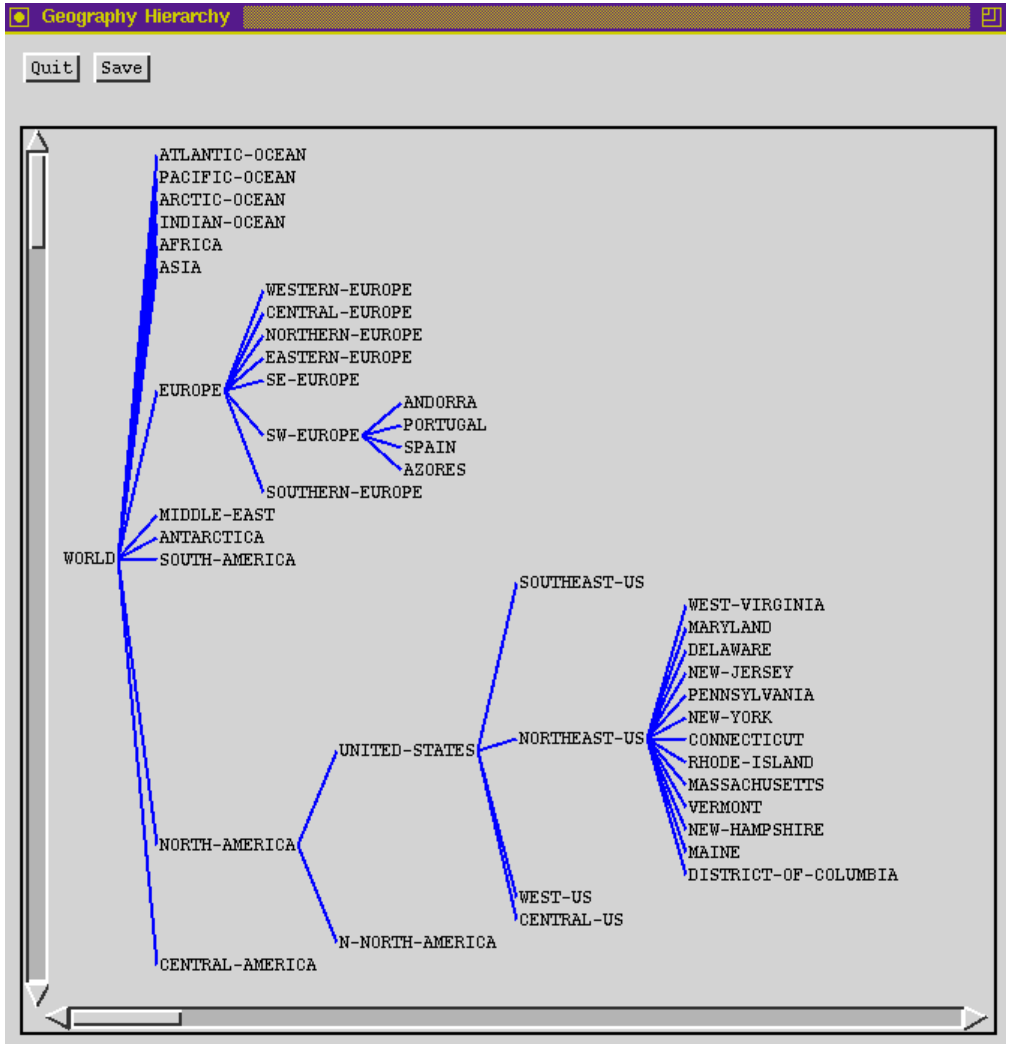
. refers to current directory

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

Hierarchal File System

- Hierarchies are ubiquitous





Unix Hierarchical File Structure

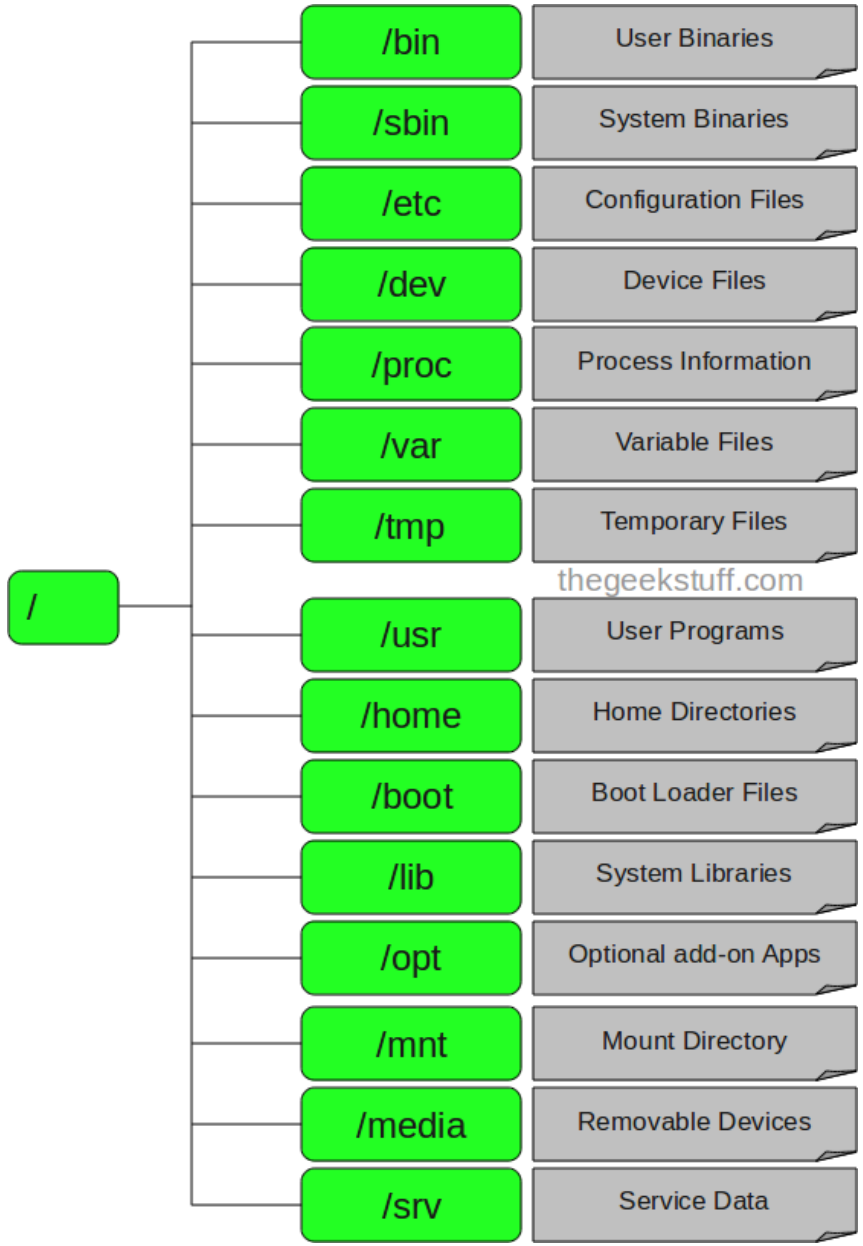
- All of the files in the UNIX file system are organized into a multi-leveled hierarchy called a directory tree.
- A family tree is an example of a hierarchical structure that represents how the UNIX file system is organized. The UNIX file system might also be envisioned as an inverted tree or the root system of plant.
- At the very top of the file system is single directory called "root" which is represented by a / (slash). All other files are "descendents" of root.
- The number of levels is largely arbitrary, although most UNIX systems share some organizational similarities.
  - A "standard" UNIX file system is usually maintained as a standard by a working group

Filesystem Hierarchy Standard (FHS)

- The Linux File Hierarchy Structure defines the directory structure (FHS) and directory contents in Unix-like operating systems.
  - Maintained by the Linux Foundation, current version 3.0
- In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.
- Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.
- Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS, and are not considered authoritative for platforms other than Linux.

General FHS for Linux

- why certain programs are located under /bin, or /sbin, or /usr/bin, or /usr/sbin ?



Source: https://www.thegeekstuff.com/2010/09/linux-file-system-structure/

FSH Specifics

- **/ – Root**
  - Every single file and directory starts from the root directory.
  - Only root user has write privilege under this directory.
  - Please note that /root is root user's home directory, which is not same as /.
- **/bin – User Binaries**
  - Contains binary executables.
  - Common linux commands you need to use in single-user modes are located under this directory.
  - Commands used by all the users of the system are located here.
  - For example: ps, ls, ping, grep, cp.
- **/sbin – System Binaries**
  - Just like /bin, /sbin also contains binary executables.
  - But, the linux commands located under this directory are used typically by system administrator, for system maintenance purpose.
  - For example: iptables, reboot, fdisk, ifconfig, swapon

FSH Specifics (1 of 4)

- **/etc – Configuration Files**
  - Contains configuration files required by all programs.
  - This also contains startup and shutdown shell scripts used to start/stop individual programs.
  - For example: /etc/resolv.conf, /etc/logrotate.conf
- **/dev – Device Files**
  - Contains device files.
  - These include terminal devices, usb, or any device attached to the system.
  - For example: /dev/tty1, /dev/usbmon0
- **/proc – Process Information**
  - Contains information about system process.
  - This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
  - This is a virtual filesystem with text information about system resources. For example: /proc/uptime

FSH Specifics (2 of 4)

- **/var – Variable Files**
  - var stands for variable files.
  - Content of the files that are expected to grow can be found under this directory.
  - This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);
- **/tmp – Temporary Files**
  - Directory that contains temporary files created by system and users.
  - Files under this directory are deleted when system is rebooted.
- **/usr – User Programs**
  - Contains binaries, libraries, documentation, and source-code for second level programs.
  - /usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
  - /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
  - /usr/lib contains libraries for /usr/bin and /usr/sbin
  - /usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2

FSH Specifics (3 of 4)

- **/home – Home Directories**
  - Home directories for all users to store their personal files.
  - For example: /home/john, /home/nikita
- **/boot – Boot Loader Files**
  - Contains boot loader related files.
  - Kernel initrd, vmlinux, grub files are located under /boot
  - For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic
- **/lib – System Libraries**
  - Contains library files that supports the binaries located under /bin and /sbin
  - Library filenames are either ld* or lib*.so.*
  - For example: ld-2.11.1.so, libncurses.so.5.7

FSH Specifics (4 of 4)

- **/opt – Optional add-on Applications**
  - opt stands for optional.
  - Contains add-on applications from individual vendors.
  - add-on applications should be installed under either /opt/ or /opt/ sub-directory.
- **/mnt – Mount Directory**
  - Temporary mount directory where sysadmins can mount filesystems.
- **/media – Removable Media Devices**
  - Temporary mount directory for removable devices.
  - For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer
- **/srv – Service Data**
  - srv stands for service.
  - Contains server specific services related data.
  - For example, /srv/cvs contains CVS related data.

Typical Filesystem Directory Contents

/	The "root" directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/users or /home	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	UNIX system configuration and information files
/dev	Hardware devices
/var	Content of the files in this directory are expected to grow/shrink like user's mail spool and print spool files
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

Fundamentals of Security

- UNIX systems have one or more users, identified with a number and name.
  - UNIX is a multi-user system.
- A set of users can form a group. A user can be a member of multiple groups.
  - A special user (id 0, name **root**) has complete control.
  - Every user has a **username**, numeric **uid** (user identification),a **default group** association, optional associations with other groups are possible
- **`id`** view your uid and default group and which groups you belong to

```
Amars-MacBook-Pro:CS2600 amarraheja$ id
uid=504(amarraheja) gid=20(staff) groups=20(staff),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),702(com.apple.sharepoint.group.2),704(com.apple.sharepoint.group.4),33(_appstore),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(com.apple.access_screensharing),399(com.apple.access_ssh),701(com.apple.sharepoint.group.1),703(com.apple.sharepoint.group.3)
Amars-MacBook-Pro:CS2600 amarraheja$
```

File/Directory Access Permissions

- Every file and directory in your account is protected from others
- Every file has:
  - A single owner
  - An association with a single group
  - A set of access permissions associated with it
- For a File, permissions control what can be done to the file contents
- For a Directory, permissions control whether a file in that directory can be listed, searched, renamed or removed

How are Users & Groups used?

- Used to determine if file or process operations can be performed:
  - Can a given file be read? written to?
  - Can this program be run?
  - Can I use this piece of hardware?
  - Can I stop a particular process that's running?
- UNIX provides a way to protect files based on users and groups.
- Three **types** of permissions:
    - read, process may read contents of file
    - write, process may write contents of file
    - execute, process may execute file
- Three **sets** of permissions:
    - permissions for owner
    - permissions for group (1 group per file)
    - permissions for other

File Access Permissions

- Every user has responsibility for controlling access to their files and directories.
  - can be made accessible to other users by changing its access permissions only by the owner of file
- Permissions for a file or directory may be any or all of the following:

  r - read

  w - write

  x - execute = running a program
- Each permission (rwx) can be controlled at three levels:

  u - user = yourself

  g - group = can be people in the same project

  o - other = everyone on the system

Directory Access Permissions

- Same types and sets of permissions as for files:

  read: process may a read the directory *contents* (i.e., list files)

  write: process may add/remove files in the directory

  execute: process may open files in directory or subdirectories

Permission	**For a File**	**For a Directory**
r (read)	Contents can be viewed or printed.	Contents can be listed, but not searched. Normally r and x are used together.
w (write)	Contents can be changed or deleted.	File entries can be added or removed.
x (execute)	File can be run as a program.	Directory can be searched and you can cd to it.

Viewing File Permissions

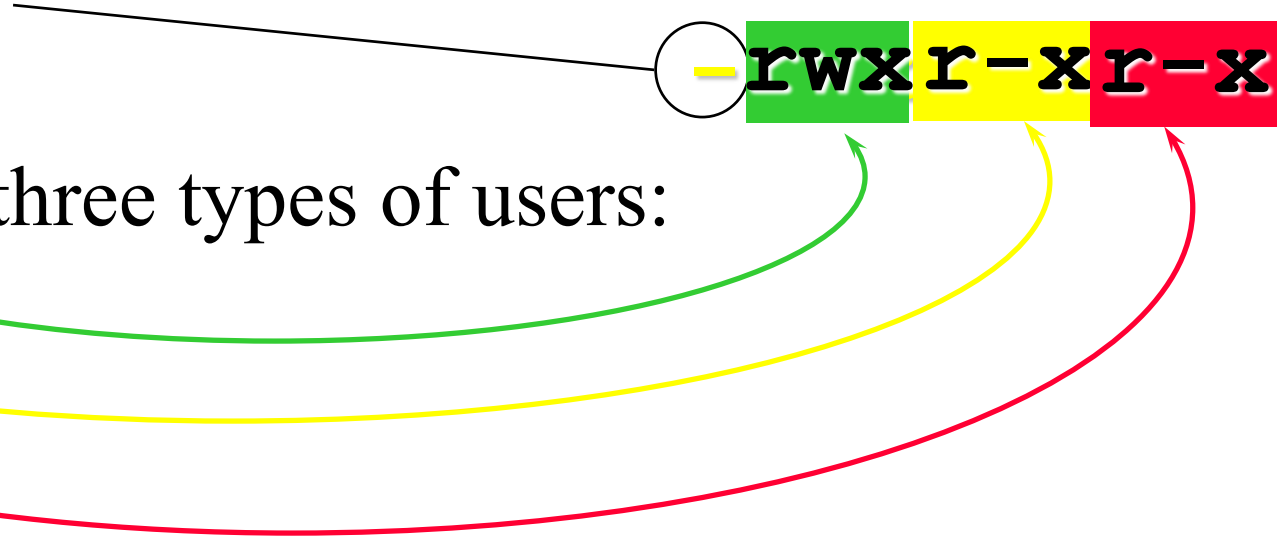
- owner read (r)
- owner write (w)
- owner execute (x)
  - group read (r)
  - group write (w)
  - group execute (x)
    - public read (r)
    - public write (w)
    - public execute (x)

  which are displayed as: -rwxrwxrwx

  -rwxr-xr-x 1 amarraheja staff 8432 May 14 15:22 a.out
  -rw-r--r-- 1 amarraheja staff   105 May 25 19:09 data
  drwxr-xr-x 2 amarraheja staff   64 May 26 23:23 junk

Access Permissions

- First character shows the file type:
  - directory (d)
  - plain file (-)
  - link (l)



  - Rest specify three types of users:
    - user owner
    - group
    - others
  - who are allowed to:
    - (r) read
    - (w) write
    - (x) execute

Utilities for Manipulating File Attributes

- **chmod:** change file permissions

  chmod [permission] [file_name]

- **chown:** change file owner

  chown [user_name] [file_name]

- **chgrp:** change file group

  chgrp [group_name] [file_name]

- **umask**: user file creation mode mask
  - only owner or super-user can change file attributes
  - upon creation, default permissions given to file modified by process **umask** value

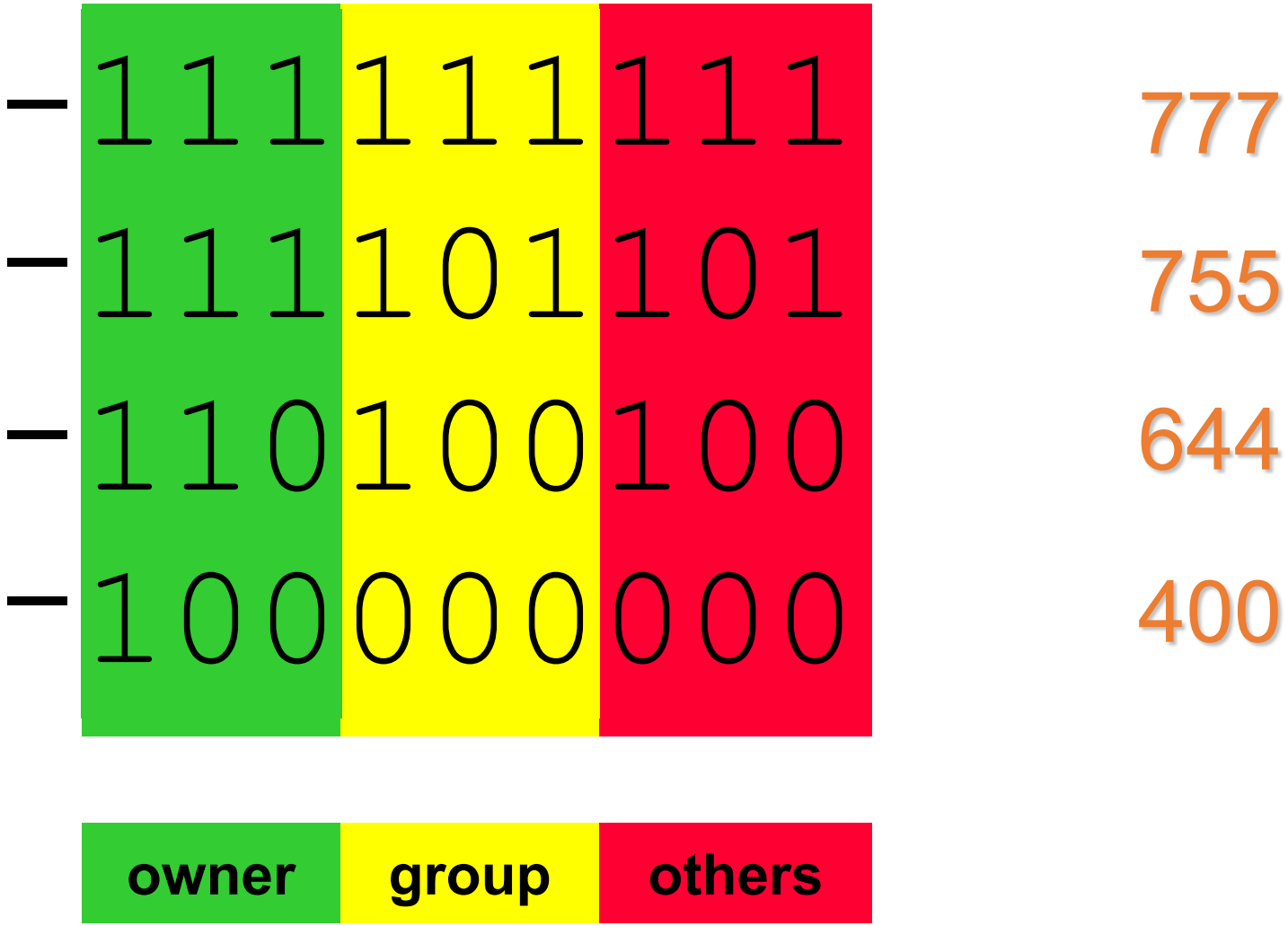
  `$umask ddd`

Permission Settings for Octal Mode

- Permission settings use octal numbers.
  - **r** = 100 = 4
  - **w** = 010 = 2
  - **x** = 001 = 1
  - **None** = 000 = 0

- These numbers are **additive**.
  - **rwx** = 7 (4 + 2 + 1) = 111
  - **rw** = 6 (4 + 2) = 110
  - **rx** = 5 (4 + 1) = 101

Numerical Access Permissions



Chmod command

- Symbolic access modes {`u`,`g`,`o`} / {`r`,`w`,`x`}
    - example: `chmod +r` *file*
- Octal access modes

octal	**read**	**write**	**execute**
0	no	no	no
1	no	no	yes
2	no	yes	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	yes	no
7	yes	yes	yes

Permission Settings: Examples

Use **`ls -l`** to view permission settings

`-r-- --- ---`	(400)	protect it from accidental editing
`-rw- --- ---`	(600)	only you can edit/read the file
`-rw- r-- r--`	(644)	only you can edit, others can read
`-rw- rw- rw-`	(666)	public file!
`dr-x r-x r-x`	(555)	anyone can list but can't create/delete/rename files
`dr-x --- ---`	(500)	only you can list
`drwx --- ---`	(700)	you can do anything, but not others
`drwx r-x r-x`	(755)	you can do anything, others only list
`drwx rwx rwx`	(777)	anyone can do anything!

Changing Permissions

- **`chmod`** command is used to modify permissions. can only be used by the owner of a file/dir (or the administrator *root*).

Format: **`chmod [ugoa] [+-=] [rwx] [file/dir]`**

- Optionally, one of the characters: *u* (user/owner), *g* (group), *o* (other), or *a* (all).
- Optionally, one of the characters: *+* (add permission), - (remove permission), or = (set permission).
- Any combination of the characters *r* (read), *w* (write), or *x* (execute).

chmod Examples

- Character Method

```
$ chmod a=r-x file
$ chmod u=rw- file
$ chmod ugo+r file
$ chmod go-w file
$ chmod g:cs2600-001+rx file
```

- Numerical Method

```
$ chmod 744 file
$ chmod 600 file
```

Advanced Permissions Flags

- The special permissions flag can be marked with any of the following:

  _ no special permissions

  d directory

  l The file or directory is a symbolic link

  s This indicated the setuid/setgid permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a **s** in the execute portion of the owner or group permissions.

  t This indicates the sticky bit permissions. This is not set displayed in the special permission part of the permissions display, but is represented as a **t** in the executable portion of the all user permissions

What is SUID and SGID?

- **SUID** is a special file permission for executable files which enables other users to run the file with effective permissions of the file owner. Instead of the normal **x** which represents execute permissions, you will see an **s** (to indicate **SUID**) special permission for the user.
- **SGID** is a special file permission that also applies to executable files and enables other users to inherit the effective **GID** of file group owner. Likewise, rather than the usual **x** which represents execute permissions, you will see an **s** (to indicate **SGID**) special permission for group user.

Examples of SUID and SGID

$ find . -perm /4000

```
aaronkilik@tecmint ~ $ find . -perm /4000
./backup.sh
./update.sh
./diskusage.sh
aaronkilik@tecmint ~ $ ls -l backup.sh
-rwsrwx--- 1 aaronkilik aaronkilik 0 Aug  3 22:06 backup.sh
aaronkilik@tecmint ~ $ ls -l update.sh
-rwsrws--- 1 aaronkilik aaronkilik 0 Aug  3 22:06 update.sh
aaronkilik@tecmint ~ $ ls -l diskusage.sh
-rwsrws--- 1 aaronkilik aaronkilik 20 Aug  3 22:04 diskusage.sh
aaronkilik@tecmint ~ $
```

$ find . -perm /2000

```
aaronkilik@tecmint ~ $ find .  -perm /2000
./add.c
./update.sh
./diskusage.sh
./del.py
aaronkilik@tecmint ~ $ ls -l add.c
-rwxrws--- 1 aaronkilik aaronkilik 0 Aug  3 22:12 add.c
aaronkilik@tecmint ~ $ ls -l update.sh
-rwsrws--- 1 aaronkilik aaronkilik 0 Aug  3 22:06 update.sh
aaronkilik@tecmint ~ $ ls -l diskusage.sh
-rwsrws--- 1 aaronkilik aaronkilik 20 Aug  3 22:04 diskusage.sh
aaronkilik@tecmint ~ $ ls -l del.py
-rwxrws--- 1 aaronkilik aaronkilik 0 Aug  3 22:11 del.py
aaronkilik@tecmint ~ $
```

Source: https://www.tecmint.com/how-to-find-files-with-suid-and-sgid-permissions-in-linux/

Class Exercise

- Use id command to check your user id and groups you belong to
- Create a few empty files (foo1, foo2 and foo3) using the touch command
- Look at the default permissions on creation
- Make foo1 executable
- Change permissions for foo2 to have read and execute for group as well as read for others using u,g,o and r,w,x
- Do the above to foo3 but using octal numbers
- Use the = symbol to assign a read and write permission to all
- Can you find any listing of files/directories in your home directory that has any other symbol like s or t in place of the x?
- Remove all 3 files after this exercise