# File Security and Permissions

# File Permissions (1)

- ◆ With respect to a particular file, Unix divides the set of all users on a system into three categories:
  - – user
    - ❖ The owner of the file.
  - – group users
    - ❖ Your group may be 2ndyr, or same as your user name.
    - ❖ Used for easier administration of access control.
    - ❖ Normally only the superuser can set up groups.
    - ❖ Users can be in more than one group.
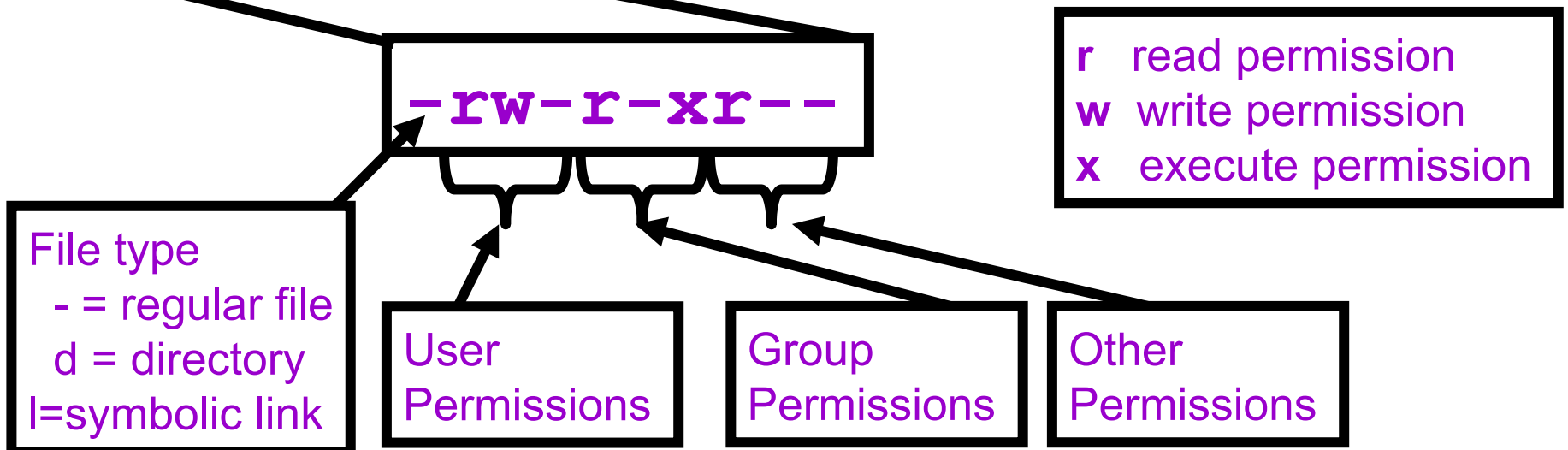  - – others
    - ❖ Everyone else.

# File Permissions (2)

◆ Permissions can be viewed with the **ls -l** command

compute[1] > ls -l

```
total 1247
-rw-------    1 csnow       99    1117   Jul 23 15:49   bad.cpp
drwx--x--x   2 csnow       99       8   Jul 17 10:13   bibd/
drwxr-xr-x   2 csnow  csnow      32   Aug 27 23:18   cache/
-rw-------    1 csnow       99    2081   Jul 23 15:49   tst2.s
-rw-r-xr--   1 csnow       99    1275   Jul 23 15:49   vecexpr.cpp
```

**-rw-r-xr--**

r  read permission
w  write permission
x  execute permission

File type
 - = regular file
 d = directory
 l=symbolic link

User Permissions

Group Permissions

Other Permissions

# File Permissions (3)

◆ Permissions are changed with the chmod command.

◆ There are two syntaxes you can use:

chmod DDD file [file ...]

– DDD are 3 octal digits representing bits of protection

– rwx rwx rwx can be thought of as 111 111 111 in binary

```
rw-r--r--
110100100
  6   4   4        chmod 644 file
```

# File Permissions (4)

◆ chmod [ugoa][+-=][rwx] file [...]

  – This is the "symbolic" method.

  – chmod u+rwx file gives the User Read, Write, and eXecute

  – chmod g+rx file   gives the Group Read and eXecute

  – chmod o-rwx file removes R, W, and X from Others

  – chmod a+x file gives All eXecute permission

  – chmod g=r file gives Group Read permission and makes sure it has nothing else

◆ Symbolic modes can be appended with commas

  – chmod u=rwx,g-w,o-rwx file for instance

# File Permissions (5)

◆ The -R option to chmod is useful when working with directories.

– All files and directories would receive those permissions.

– It recursively changes the mode for each chmod operand that is a directory.

– chmod -R a+rw dir gives everyone read and write permission to each file under dir (not execute though!!!)

– chmod -R a+rwx dir gives the executable access to allow people to actually access the files under dir

❖ Makes all files executable though ...

– chmod -R a+rwX dir gives the executable access only to those files already executable (programs, directories, …)

# The umask command

◆ umask sets the default permissions for any file you will create

◆ Format is **<u>backwards</u>** to the chmod command
  – tells you which permissions will **NOT** be given
    ❖ umask 077 means don't let anyone but the User do anything with my files by default

◆ Generally set umask once in your .bash_profile file and never set it again

# Files and inode (1)

◆ Each file is associated with an inode (index node) in the inode table of a file system

◆ The inode is accessed by the inode number and contains the following attributes of the file
  – File type (regular, directory, device, link, etc.)
  – File permission (the nine permissions and three more)
  – Number of links (number of aliases of the file)
  – The UID of the owner
  – The GID of the group owner
  – File size in byte
  – Date and time of last modification
  – Date and time of last access
  – Date and time of last change of the inode
  – An array of pointers for all disk blocks used by the file

# Files and inode (2)

- Use stat file_name to check inode contents of file_name
- An inode does not store the name of the file
- An inode does not store the inode number of the file
- File name and its inode number are stored in the directory containing the file
- A directory is a file containing file name and its associated inode number of each file in the directory
- A directory is a special file because you can not directly read or write its file content
- Each file system has its inode table stored in a separate area of the disk
- The inode number of a file is unique within a single file system

# Regular File Permissions

◆ The meaning of regular file permissions are simple

  – Requires execute permission to run the file as a command or a program
  – Requires read permission to read the file content
    ❖ example: can use more, vi (read only), or open file for read
  – Requires write permission to write the file or modify the file
    ❖ example: can cp to it or open file for write

# Directory Permissions (1)

◆ The meaning of directory permissions are different from the regular file permissions

- Requires execute permission to access the directory itself and files and subdirectories in the directory
  - ❖ example: cd into the directory
- Requires read permission to list the contents of the directory
  - ❖ example: ls the directory
- Requires write permission to create files, delete files, and rename files in the directory
  - ❖ example: rm a file or mv with a new name

# Directory Permissions (2)

```
compute[1] > ls -l
drwx--x--- 2 csnow 99    8 Jul 17 10:13  bibd/
compute[2] > ls -l bibd
-r--r--rwx 1 csnow 99 173 Jul 17 10:13  readme
```

- ◆ Files in bibd/ are accessible to user

- ◆ Files in bibd/ are accessible by name (if you know the name) for group users

- ◆ Files in bibd/ and subdirectories are not accessible to others.

# Exercise – File permission

- Create a directory dir1 in your home directory
- Edit a file test.txt in dir1
- Remove your own read permission of test.txt
- *Try* to display the content of test.txt by cat
- Add you own read permission back for test.txt
- Remove your own write permission of test.txt
- Make some changes to test.txt with an editor and *try* to save.
- *Try* to delete the file test.txt

# Exercise – Directory Permission

- ◆ Create a directory dir2.
  - – What is the permission of dir2?
  - – What argument is provided to umask in your .bash_profile or .bashrc file?
- ◆ Copy test.txt to dir2/test2.txt
- ◆ Remove your own 'r' permission of dir2.
  - – *Try* to ls dir2.
  - – cat dir2/test2.txt
  - – cd dir2
  - – ls
  - – cd ..

- ◆ Set your own permission of dir2 to be r-x
  - – cp test.txt dir2/test3.txt
  - – rm dir2/test2.txt
  - – edit the file dir2/test2.txt using an editor and save the changes
- ◆ Set your own permission of dir2 to be rw-
  - – cd dir2
  - – cat dir2/test2.txt
  - – cp test.txt dir2/test3.txt
  - – ls dir2