

# COMP 2021

## Unix and Script Programming



Unix File, Security and Link



# Tools: Archiving & Zipping

---

## ▶ Compress

- ▶ Tar the directory `Packages` to create a file `NewPackages.tar` ; **c=create, v=verbose, f=file**
- ▶ `tar cvf NewPackages.tar Packages`
- ▶ **Zip up** `NewPackage.tar` to reduce its size, this will create the file `NewPackages.tar.gz`
- ▶ `gzip NewPackages.tar`

## ▶ Decompress

- ▶ `gunzip NewPackages.tar.gz`
- ▶ `Tar xvf NewPackages.tar`
- ▶ **x=extract, v=verbose, f=file**



# File System

---

- ▶ **What is a file system?**

- ▶ A means of organizing information on the computer.
- ▶ A file system is a logical view, not necessarily a physical view.

- ▶ **What does the file system provide:**

- ▶ Ways to create, move, and remove files
- ▶ Ways to order files
- ▶ Security

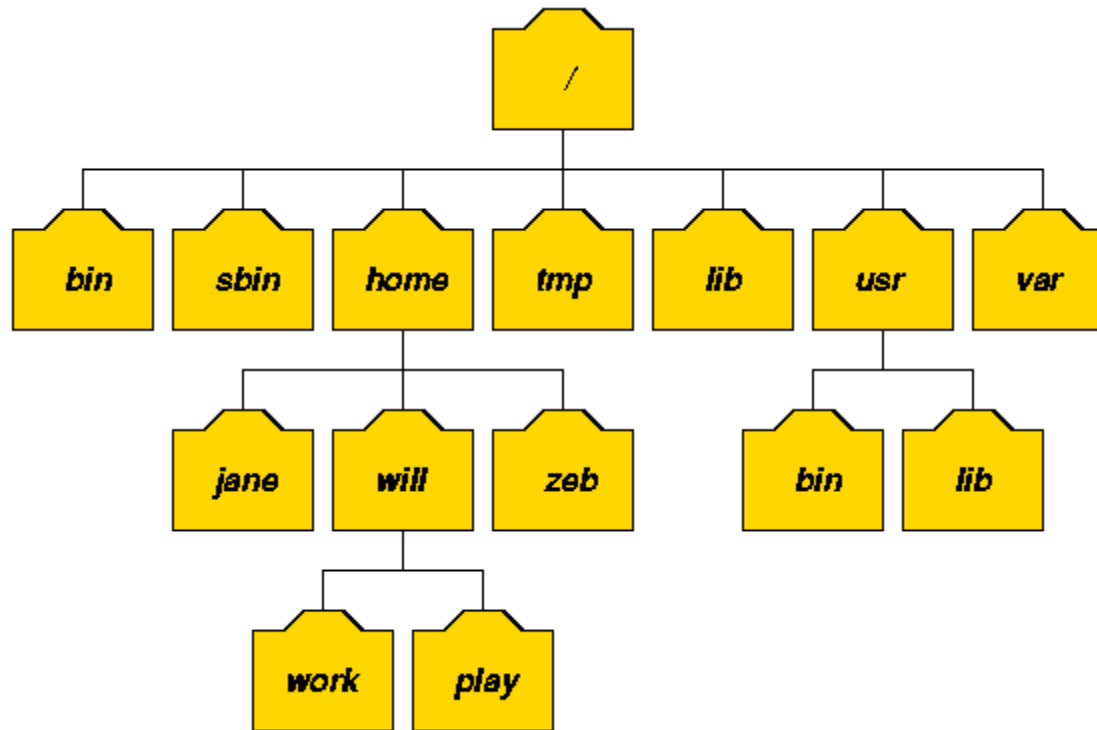
- ▶ **Examples of file systems:**

- ▶ Windows: DOS, FAT (File Allocation Table), NTFS (New Technology File System)
- ▶ MacOS: HFS+ (Hierarchical File System)

- 
- ▶ ▶ Unix, BSD, Solaris: UFS (Unix File System)

# Unix File System

---



- Hierarchical Organization

# Recap: UNIX File Utilities

---

- ▶ `ls`            **list** directory contents
- ▶ `cd`            **change** **directory**
- ▶ `pwd`           **print** **w**orking **d**irectory
- ▶ `cat`            display file
- ▶ `more`           display one screen of file
- ▶ `rm`            **remove** (delete) a file
- ▶ `rmdir`          **remove** (delete) **directory**
- ▶ `cp`            **copy** source file to target file
- ▶ `mv`            rename or **move** a file



# File? Device?

---

- ▶ In UNIX a file is a sequence of bytes of data that reside in semi permanent form on some stable medium.
- ▶ This file can contains anything you can represent as a stream of bytes.
  - ▶ A network interface, a disk drive, a keyboard, and a printer
- ▶ All input and output devices are treated as files in UNIX.
  - ▶ Described under file types and file system structure.



# File Name and Extension

---

- ▶ File names are limited to 14 chars in system V and 255 chars in BSD.
  - ▶ To avoid any problem use only letters, numbers, period and underscore.
- ▶ **Extensions (usually) have no meaning to UNIX**, they are parts of files names, used by users to mark their files types.
  - ▶ common files extensions:

.c	c source code.
.C, .cc	c++ source code
.o	object file
.ps	PostScript format document
.gz	gzip compressed file
.tar	Tape archive, used by tar command
.dat	Data or other information
.pl	Perl Program
.bak	Backup cope of file
.asc	ASCII (text) file, often containing ANSI codes



# Types of Files in Unix

---

## ▶ Simple/ordinary file

- ▶ Store info and data on secondary storage device, typically a disk.
- ▶ It can contain a source program, an executable program such as compilers, DB tools, pictures, audio, graphics, and so on.
- ▶ Unix does not treat any kind of these files differently from others. For example a C++ file is no different to UNIX than an HTML file, however the files are treated differently by C++ compiler and WEB browser.

## ▶ Directory files

- ▶ Store (inode, filename) for each file or subdirectory it contains.





# Unix Inode

---

- ▶ A Unix file is "stored" in two different parts of the disk - the **data blocks** and the **inode**.
  - ▶ Data blocks: contents of the file
  - ▶ inode: information about the file
- ▶ Every file has a **unique** inode
- ▶ An inode contains the metadata for UNIX files, necessary for a process to access a file
- ▶ Exist in a static form on disk
- ▶ Kernel reads them in to an in-core inode to manipulate them



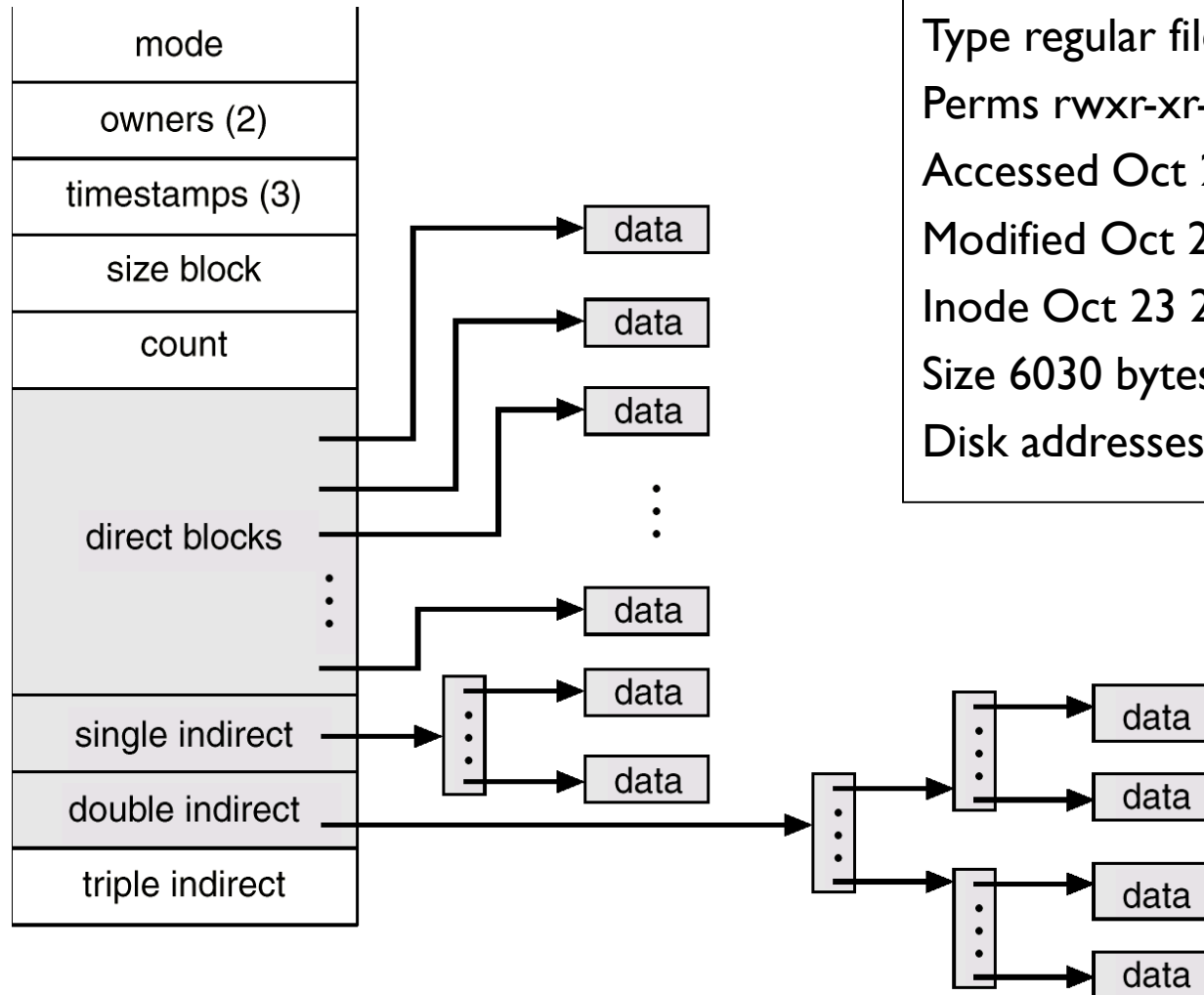
# Contents of Disk Inodes

---

- ▶ File owner identifier (individual/group owner)
  - ▶ File type (regular, directory,...)
  - ▶ File access permission (owner,group,other)
  - ▶ File access time
  - ▶ Number of links to the file (chap5)
  - ▶ Table of contents for the disk address of data in a file (byte stream vs discontinuous disk blocks)
  - ▶ File size
- \* Inode does **not specify** the **path name** that access the file



# Inode Structure



## Sample Inode:

Owner cindy

Group comp2021

Type regular file

Perms rwxr-xr-x

Accessed Oct 23 2015 1:45 P.M

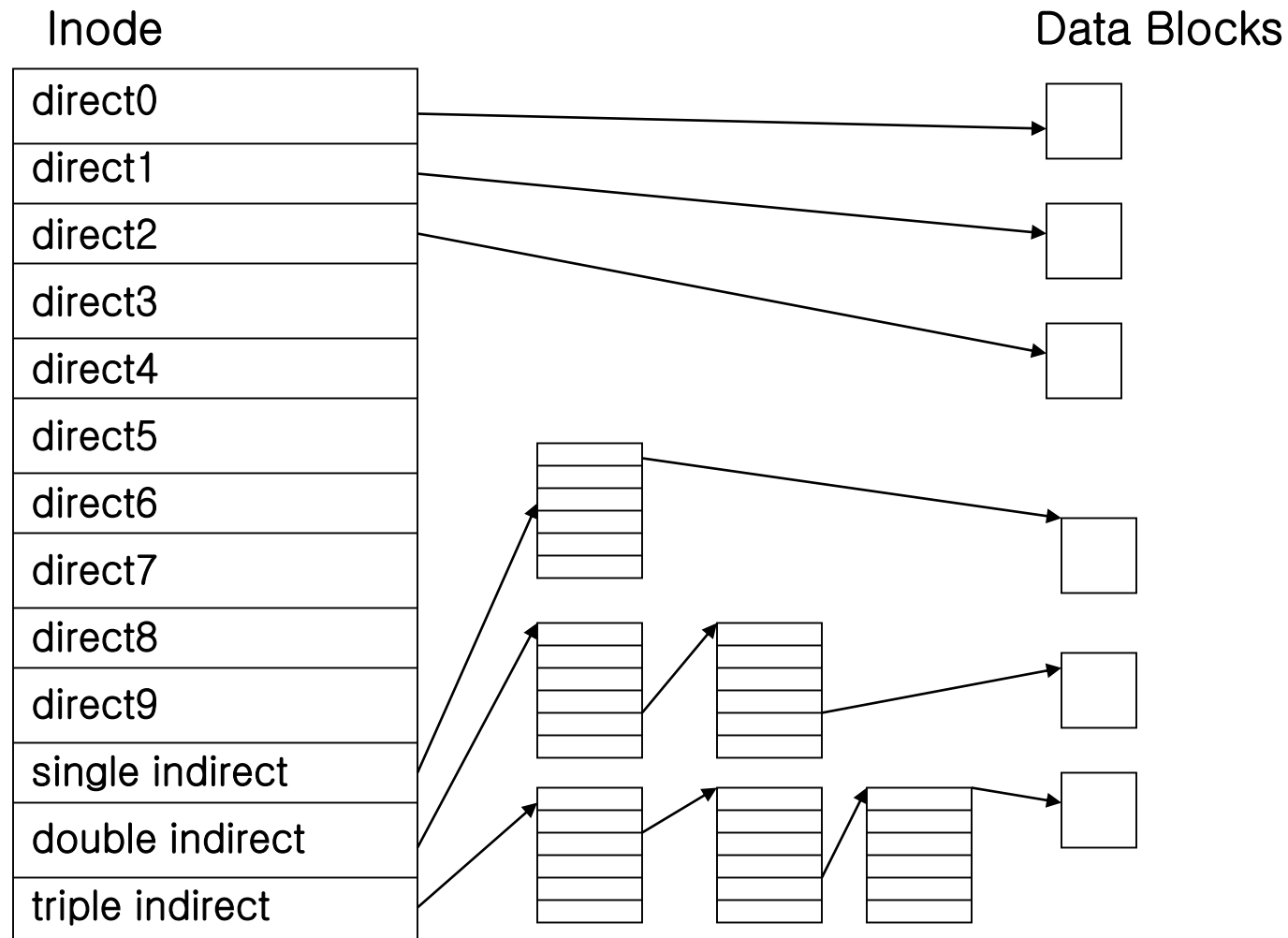
Modified Oct 22 2015 10:3 A.M

Inode Oct 23 2015 1:30 P.M

Size 6030 bytes

Disk addresses

# Direct and Indirect Blocks in Inode



# Byte Capacity of a File

---

## ▶ System V UNIX. Assume that

- ▶ Run with 13 entries
- ▶ 1 logical block : 1K bytes
- ▶ Block number address : a 32 bit (4byte) integer

- ▶ 1 block can hold up to 256 block number (1024byte / 4byte)
- ▶ 10 direct blocks with 1K bytes each=10K bytes
- ▶ 1 indirect block with 256 direct blocks= 1K\*256=256K bytes
- ▶ 1 double indirect block with 256 indirect blocks=256K\*256=64M bytes
- ▶ 1 triple indirect block with 256 double indirect blocks=64M\*256=16G
- ▶ Size of a file : 4G ( $2^{32}$ ), if file size field in inode is 32bits



# Directory

---

- ▶ A directory is a file
- ▶ Its data is a sequence of entries, each consisting of an inode number and the name of a file contained in the directory

- ▶ Example:

Byte Offset in Directory	Inode Number (2 bytes)	File Names
0	83	.
16	2	..
32	1798	init
48	1276	fsck
...	...	...
224	<b>0</b>	crash
240	95	<b>mkfs</b>
256	188	inittab



# Unix File System (revisit)

---

- ▶ File system contains a linear list of inodes

boot block	super block	inode list	data blocks
------------	-------------	------------	-------------

- ▶ Inode is free if its type field is zero (0), can be assigned to a new file

- ▶ **Super block**

- ▶ the size of the file system
- ▶ the number of free blocks in the file system
- ▶ a list of free blocks available on the file system
- ▶ the index of the next free block in the free block list
- ▶ the size of the inode list
- ▶ the number of free inodes in the file system
- ▶ a list of free inodes in the file system
- ▶ the index of the next free inode in the free inode list
- ▶ lock fields for the free block and free inode lists

- 
- ▶ ▶ a flag indicating that the super block has been modified

# Other Types of Files in Unix

---

## ▶ Pipe

- ▶ First-in-first –out (FIFO)
- ▶ Its data is transient: once data is read from a pipe, it cannot be read again
- ▶ Use only direct block (not the indirect block)

## ▶ Link files

- ▶ Point to the existing file, allowing you to rename an existing file and share it without duplicating its contents.
- ▶ Establish connection between the file to be shared
- ▶ `ln` command links the file to a directory.





# Other Types of Files in Unix (cont.)

---

## ▶ Special files (devices)

- ▶ A mean of accessing hardware device, including the keyboard, hard disk, CD-ROM drive...etc.
- ▶ Each hardware device is associated with at least one special file, and hence an inode number
- ▶ Usually reside in the `/dev` directory.



# User and Group

---

- ▶ Unix was designed to allow multiple people to use the same machine at once
- ▶ Security and sharing are important issues to be dealt with
  - ▶ Access to files depends on the users account
  - ▶ All accounts are presided over by the Superuser, or “root” account
  - ▶ Each use has absolute control over any files he/she owns, which can only be superseded by root
  - ▶ Files can also be assigned to groups of users, allowing reading, modifications and/or execution to be restricted to a subset of users



# File Ownership

---



- ▶ Each file is assigned to a single user and a single group (usually written user : group)
  - ▶ cindy's file belongs to `cindy : cs`, and roots files belong to `root : root`
- ▶ Needs root privilege to change file ownership – a regular user can't take ownership of their files to another user or a group they don't belong to

## Class Exercises

Find out who you are (login name) and which groups do you belong to?

`whoami`

`groups`



# Discovering Permissions

---

- ▶ Use `ls -l` to tell about ownership and permissions of files
  - ▶ `ls -l` lists files and directories in the long format

## Example

```
-rw----- 1 lixin cs 1355 Feb 1 21:32 2021StuList
```

## Question: What permission is allowed?

```
-rw-r--r-- 1 lixin cs 1355 Feb 1 21:32 2021OpenList
```

- ▶ Can use `ls -ld` to lists a directory's information



# Security and Access Permissions

---

- ▶ There are three types of users:
  - ▶ The owner of the file (*user*)
  - ▶ The group of the file (*group*)
  - ▶ Anyone else (*other*)
- ▶ There are three types of permission (independent of each other):
  - ▶ Read permission
  - ▶ Write permission
  - ▶ Execute permission



# Crack the Format

`-rw-r--r-- 1 cindy cs 154 Feb 4 15:00 2021 OpenList`

Permissions: `-rw-r--r--`  
#links: `1`  
User: `cindy`  
Group: `cs`  
Byte size: `154`  
Last modification: `Feb 4 15:00 2021`  
Name: `OpenList`

- ▶ There are four sets of items in the permissions:

`-rwxrwxrwx`

type

**user** **group** **other**

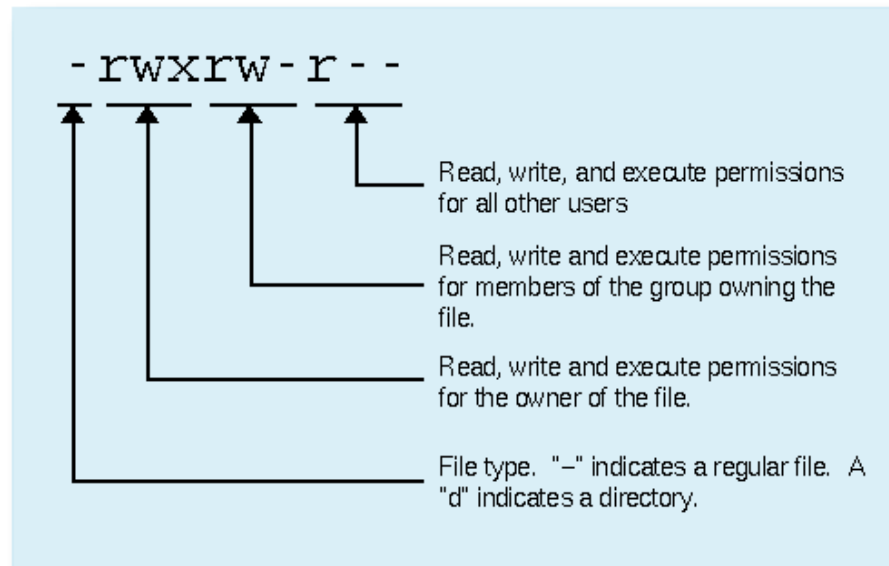
- ▶ The type is:  
“-” regular files, “d” directories, “l” symbolic links.
- ▶ The next nine characters indicate if the file is readable, writable, or executable for the file owner, the file group, or other users, respectively.

# Changing Permissions

## Change Mode

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

- Changes file/directory permissions
- 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).



# Permission Example

---

- ▶ To let everybody read or write the file

```
$ chmod a+rw file
```

```
$ ls -l file
```

```
-rw-rw-rw- 1 cindy cs 154 Feb 4 15:00 file
```

- ▶ To allow user to execute file

```
$ chmod u+x file
```

```
$ ls -l file
```

```
-rwxrw-rw- 1 cindy cs 154 Feb 4 15:00 file*
```

- ▶ To not let “other” to read or write file file

```
$ chmod o-rw file
```

```
$ ls -l file
```

```
-rwxrw---- 1 cindy cs 154 Feb 4 15:00 file*
```

- ▶ To let “group” only read the file file

```
$ chmod g=r file
```

```
$ ls -l file
```

```
-rwxr----- 1 cindy cs 154 Feb 4 15:00 file*
```





# Permission Shortcut

- ▶ Think of r, w, and x as binary variables
  - ▶ 1 ON, 0 OFF

Binary	Symbolic	Allowed file accesses
111000000	rwX-----	Owner can read, write, and execute
111111000	rwXrwX---	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwXr-Xr-X	Owner can do everything, rest can read and execute
000000000	-----	Nobody has any access
000000111	-----rwx	Only outsiders have access (strange, but legal)

## Decimal Permission

chmod 755:    rwXr-Xr-X

chmod 600:    rw-----

chmod 777:    rwXrwXrwx

# Permission Example

---

Note: CSE homepage is not available for UG students

## Setup your homepage at Department of Computer Science and Engineering

First of all, logon to any UNIX workstations, e.g. ras, make a directory "public\_html" under your home, i.e. "**mkdir ~/public\_html**", put an index file, "index.html" under the "public\_html" directory.

Make sure the permissions have been set appropriately, for instance:

- Your home directory has to be at least world-accessible, i.e. "**chmod 701 ~**"
- The same for "public\_html" directory, i.e. "**chmod 701 ~/public\_html**"
- The files under "public\_html" directory should be world-readable, i.e. "**chmod 604 ~/public\_html/\*.html**"

Now, your homepage should be ready and can be accessed as:

**<http://www.cse.ust.hk/~<your name>>**

You may setup your personal homepage under ITSC ihome service

<http://itsc.ust.hk/services/general-it-services/communication-collaboration/ihome>



# Directory Permissions

---

- ▶ Read determines if a user can view directory's contents
  - ▶ i.e. do `ls` in it.
- ▶ Write determines if a user can create new files or delete file in the directory
  - ▶ Note a user with write access to a directory can delete files in the directory even if he doesn't have write permission to the file
- ▶ Execute determines if the user can `cd` into the directory
  - ▶ You can use the directory name when accessing files inside it
  - ▶ you can do `ls` and `cp` on individual files in the directory.



# Directory Permissions

---

```
$ ls -ld secret*
```

drwxr-xr-x	2	cindy	cs	512	Feb	4	16:38	secret/
d-----	2	cindy	cs	512	Feb	4	16:39	secret1/
dr--r--r--	2	cindy	cs	512	Feb	4	16:39	secret2/
d--x--x--x	2	cindy	cs	512	Feb	4	16:38	secret3/

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

```
secret:
```

```
total 2
```

-rw-r--r--	1	cindy	cs	1054	Feb	4	16:38	letter1
------------	---	-------	----	------	-----	---	-------	---------

```
secret1 unreadable
```

```
ls: secret2/letter1: Permission denied
```

```
secret2:
```

```
total 0
```

```
secret3 unreadable
```



# Brian Storming

---

- ▶ Real-life Example: What if you want your friend to get a file and no one else ?
  - ▶ Hint: with the trick of permission



# Links

---

- ▶ **A link is a directory entry that points to blocks on disk.**
  - ▶ In other words, every file on your system has at least one link
- ▶ In fact, in UNIX *all* filenames are just links to a file. Most files only have one link.

```
-rw----- 1 lixin cs 4 Feb 7 11:57 original1  
-rw----- 1 lixin cs 4 Feb 7 11:57 original2
```

## Number of links

- ▶ Additional links to a file allow the file to be shared.
- ▶ The `ln` command creates new links.

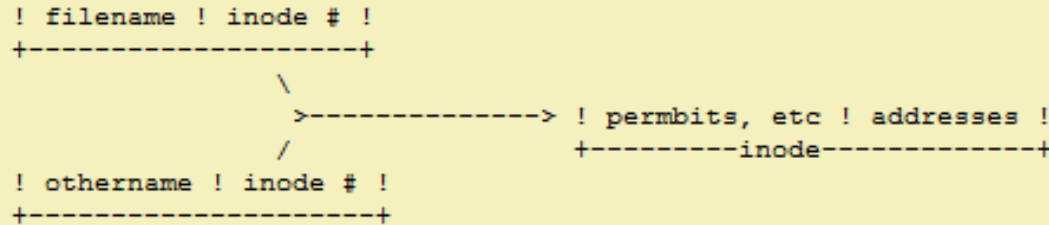
```
$ ln original1 original1-hard  
$ ls -l  
total 12  
-rw----- 2 lixin cs 4 Feb 7 11:57 original1  
-rw----- 2 lixin cs 4 Feb 7 11:57 original1-hard  
-rw----- 1 lixin cs 4 Feb 7 11:57 original2
```



# Hard Link

---

- ▶ More than one filename can reference the same inode number; these files are said to be 'hard linked' together



```
$ ls -li
```

```
total 12
```

```
1744963559 -rw----- 2 lixin cs 4 Feb 7 11:57 original1
```

```
1744963559 -rw----- 2 lixin cs 4 Feb 7 11:57 original1-hard
```

```
1765251893 -rw----- 1 lixin cs 4 Feb 7 11:57 original2
```

**inode number**



# ln command

---

- ▶ **ln** make links between files

## The ln command

- `ln [option] TARGET LINK_NAME`

Create a link to **TARGET** with the name **LINK\_NAME**

- `ln [option] TARGET DIRECTORY`

Create a link to **TARGET** in **DIRECTORY**

- ▶ **ln** creates hard link by default
- ▶ The last argument is the link destination, and can be:
  - ▶ A pathname of a new regular file

```
$ ln original1 original1-hard
```
  - ▶ A pathname of an existing directory (a link with the same basename as the original file is created in the directory)

```
$ ln original1 subdir
```





# Soft Link

- ▶ There's a special file type whose data part carries a path to another file (a file that contains the name of another file)
- ▶ OS recognizes the data as a path, and redirects opens/reads/writes
- ▶ This special file is called a 'soft link', or a 'symbolic link' (aka a 'symlink')

```
! filename ! inode # !
+-----+
\
.-----> ! permbits, etc ! addresses !
+-----inode-----+

'-----'
(
'--> !"/path/to/some/other/file"!
+-----data-----+
/
}
~ ~ ~ ~ ~ }-- (redirected at open() time)
(
'~~> ! filename ! inode # !
+-----+
\
'-----> ! permbits, etc ! addresses !
+-----inode-----+

'-----'
(
'-> ! data ! ! data ! etc.
+-----+ +-----+

```

# Soft Link (cont.)

---

- ▶ A soft link is a pointer to a pathname, not a pointer to the file itself.

- ▶ `ln -s TARGET LINK_NAME` creates a soft link.

- ▶ The symbolic link has a different *inode*.

```
$ ln -s original2 original2-soft
```

```
$ ls -li
```

```
total 12
```

```
1744963559 -rw----- 2 lixin cs 4 Feb 7 11:57 original1
```

```
1744963559 -rw----- 2 lixin cs 4 Feb 7 11:57 original1-hard
```

```
1765251893 -rw----- 1 lixin cs 4 Feb 7 11:57 original2
```

```
1770683694 lrwxrwxrwx 1 lixin cs 9 Feb 7 12:55 original2-soft -  
    > original2
```



# Hard vs. Soft Links

---

```
csl2wk09:lixin:182> touch original1; touch original2
csl2wk09:lixin:183> echo "cat" >> original1
csl2wk09:lixin:184> echo "dog" >> original2
csl2wk09:lixin:185> cat original1; cat original2
cat
dog
csl2wk09:lixin:186> ln original1 original1-hard
csl2wk09:lixin:187> ln -s original2 original2-soft
csl2wk09:lixin:188> ls -li
total 12
1744963559 -rw----- 2 lixin cs 4 Feb 17 13:09 original1
1744963559 -rw----- 2 lixin cs 4 Feb 17 13:09 original1-hard
1765251893 -rw----- 1 lixin cs 4 Feb 17 13:09 original2
1747919295 lrwxrwxrwx 1 lixin cs 9 Feb 17 13:10 original2-soft -> original2
csl2wk09:lixin:189> cat original1-hard
cat
csl2wk09:lixin:190> cat original2-soft
dog
csl2wk09:lixin:191> chmod g+rw original1-hard
csl2wk09:lixin:192> ls -li
total 12
1744963559 -rw-rw---- 2 lixin cs 4 Feb 17 13:09 original1
1744963559 -rw-rw---- 2 lixin cs 4 Feb 17 13:09 original1-hard
1765251893 -rw----- 1 lixin cs 4 Feb 17 13:09 original2
1747919295 lrwxrwxrwx 1 lixin cs 9 Feb 17 13:10 original2-soft -> original2
csl2wk09:lixin:193> rm original1
csl2wk09:lixin:194> cat original1-hard
cat
csl2wk09:lixin:195> rm original2
csl2wk09:lixin:197> cat original2-soft
cat: original2-soft: No such file or directory
csl2wk09:lixin:198> █
```

When you `rm` a file the actual system call is `unlink`.

---

It removes the directory entry.

## Hard vs. Soft Links (Cont.)

---

- ▶ The most important difference between hard and soft links occur when a link is *removed*.
- ▶ When deleting files, the data part isn't disposed of until all the filename parts have been deleted.
  - ▶ *inode* keeps how many filenames point to this file
  - ▶ Count is decremented by 1 each time one of those filenames is deleted
  - ▶ When the count makes it to zero, the *inode* and its associated data are deleted.



# Hard vs. Soft Links (Cont.)

---

- ▶ You can't make a hard link to a directory, but you can make a symbolic link to a directory.

```
$ mkdir subdir
$ cd subdir
$ touch original3
$ cd ..
$ ln subdir subdir-hard
ln: `subdir': hard link not allowed for directory
$ ln -s subdir subdir-soft
$ cd subdir-soft
$ ls
original3
```

- ▶ You can also make symbolic links across file systems.

```
$ ln /usr/include/stdio.h stdio.h
ln: creating hard link `stdio.h' to `/usr/include/stdio.h': Invalid
cross-device link
$ ln -s /usr/include/stdio.h stdio.h
$ ls -li
317906316 lrwxrwxrwx 1 lixin cs 20 Feb 17 14:10 stdio.h ->
/usr/include/stdio.h
```

- ▶ There is no way to tell how many symbolic links there are to a file
  - ▶ The permission is not correctly reflected using `-l` option for soft links, use `-L` instead
-