

WARNING

This lecture contains usages of the
Papyrus and Comic Sans fonts.
Observer discretion is advised.

Everything is a File

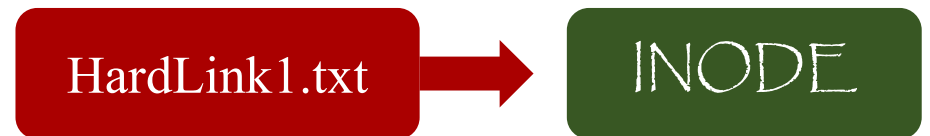
Benjamin Brewster

Except as noted, all images copyrighted with Creative Commons licenses,
with attributions given whenever available

TRIGGER WARNING
Papyrus font on next slide

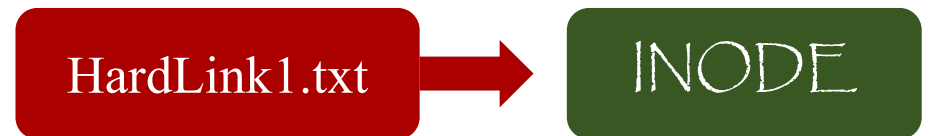
Files!

- Files are “inodes” with “hard links” that point to them.
- Inodes are maintained by the file system itself and contain:
 - Pointers to actual file data
 - All meta-information (size, permissions, etc)
 - A “reference count”: how many hard links point to the inode
 - A unique “inode number”



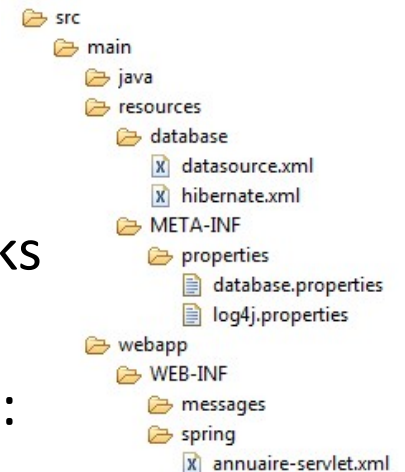
Hard Links

- A text entry in a file system directory that points to an inode
- Connects a **text filename** to an inode on disk



Directories!

- Directories are essentially text files that organize hard links hierarchically
- Create or remove directories with these UNIX commands:
 - mkdir, rmdir
- Because directories *are* files, you can also read the contents of a directory (in C):
 - opendir(), closedir(), readdir(), rewinddir()
- Directory hard link that represents itself: .
- Directory hard link that represents its parent directory: ..



What's in a directory file?

```
$ vim .
" =====
" Netrw Directory Listing                                     (netrw v149)
" /nfs/stak/faculty/b/brewsteb/codesamples
" Sorted by      name
" Sort sequence: [\/]$,\<core\%(\\.d\\+\\)\|=\\>,.h$,\\.c$,\\.cpp$,\\~\\|=\\*$,*,.o$,\\
" Quick Help: <F1>:help -:go up dir D:delete R:rename s:sort-by x:exec
" =====
../
./
Curses.pdf
IntroToUnixShell.html
OnE1.txt
OnE1FAQ.txt
OnE1_sol.txt
Prog1.html
Prog1.test
Prog1FAQ.txt
Prog2.html
Prog2FAQ.txt
cursesDemo.c
index.html
```



Directories Can Be Read Like a File

```
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void main() {
    int newestDirTime = -1; // Modified timestamp of newest subdir examined
    char targetDirPrefix[32] = "brewsteb.rooms."; // Prefix we're looking for
    char newestDirName[256]; // Holds the name of the newest dir that contains prefix
    memset(newestDirName, '\0', sizeof(newestDirName));

    DIR* dirToCheck; // Holds the directory we're starting in
    struct dirent *fileInDir; // Holds the current subdir of the starting dir
    struct stat dirAttributes; // Holds information we've gained about subdir

    dirToCheck = opendir("."); // Open up the directory this program was run in

    if (dirToCheck > 0) { // Make sure the current directory could be opened
        while ((fileInDir = readdir(dirToCheck)) != NULL) { // Check each entry in dir
            if (strstr(fileInDir->d_name, targetDirPrefix) != NULL) { // If entry has prefix
                printf("Found the prefex: %s\n", fileInDir->d_name);
                stat(fileInDir->d_name, &dirAttributes); // Get attributes of the entry

                if ((int)dirAttributes.st_mtime > newestDirTime) { // If this time is bigger
                    newestDirTime = (int)dirAttributes.st_mtime;
                    memset(newestDirName, '\0', sizeof(newestDirName));
                    strcpy(newestDirName, fileInDir->d_name);
                    printf("Newer subdir: %s, new time: %d\n", fileInDir->d_name, newestDirTime);
                }
            }
        }
    }

    closedir(dirToCheck); // Close the directory we opened
    printf("Newest entry found is: %s\n", newestDirName);
}
```

This code searches for the most recently modified/created directory whose name matches a certain prefix

Note the funky single equals sign!

Creating Links

- When you create a file (using `touch`, a C function, etc.), an inode is allocated and a hard link is automatically created
- However, you can create multiple hard links to the same inode
 - So a file can appear in multiple directories at the same time!
 - The same file can also appear under different names
 - Even in the same directory
 - That's pretty weird
- To create a link, use the `ln` or `link` commands



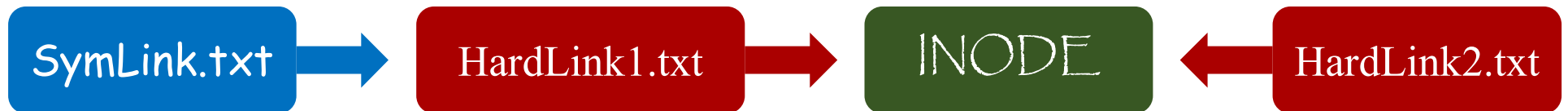
Removing Files in C

- Removing is approximately unlinking everything
 - The inode is garbage collected when ref count == 0
- One way to "remove" a file:
 - `unlink(file_name)`
 - can't unlink directories
- Another way to "remove" a file
 - `remove(file_name)`
 - unlike `unlink()`, `remove` will delete empty directories
 - if file, `remove()` is identical to `unlink()`
 - if directory, `remove()` is identical to `rmdir()`

TRIGGER WARNING
Comic Sans font on next slide

Symbolic Link

- A symbolic link is like a Windows shortcut - it's not actually a file, it points to a hard link
- If you delete, rename, or move the hard link a symbolic link points to, the symbolic link become unusable
- You can symbolically link to directories or to files across the network
- Hard links are only available on the local file system

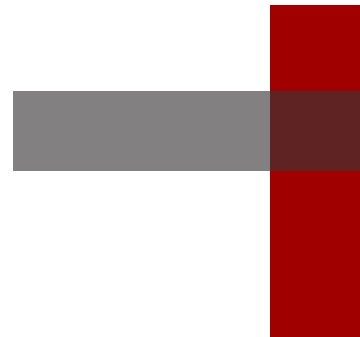


Symbolic Link Example

- I'm on a new server and want access to my home directory's files stored on my original server

```
NewServer$ ln -s /nfs/rack/u2/b/brewsteb ./myFilesLink  
NewServer$ cd myFilesLink
```

- I am now in my filesystem on the new server



Linking Example

```
$ mkdir 1
$ mkdir 2
$ touch ./1/hardlink1
$ ln ./1/hardlink1 ./2/hardlink2
$ ln -s ./1/hardlink1 ./symlink
$ find -samefile ./1/hardlink1
```

Find all files that share an inode with this hard link

```
./1/hardlink1
./2/hardlink2
```

```
$ ls -laR
```

```
..:
```

```
drwxrwx---. 1 brewsteb upg57541 124 Aug 29 15:11 ./
drwxrwx---. 1 brewsteb upg57541 636 Aug 29 14:56 ../
drwxrwx---. 1 brewsteb upg57541  84 Aug 29 15:11 1/
drwxrwx---. 1 brewsteb upg57541  84 Aug 29 15:11 2/
lrwxrwxrwx. 1 brewsteb upg57541  13 Aug 29 15:11 symlink -> ./1/hardlink1
```

```
./1:
```

```
drwxrwx---. 1 brewsteb upg57541  84 Aug 29 15:11 ./
drwxrwx---. 1 brewsteb upg57541 124 Aug 29 15:11 ../
-rw-rw----. 2 brewsteb upg57541   0 Aug 29 15:11 hardlink1
```

```
./2:
```

```
drwxrwx---. 1 brewsteb upg57541  84 Aug 29 15:11 ./
drwxrwx---. 1 brewsteb upg57541 124 Aug 29 15:11 ../
-rw-rw----. 2 brewsteb upg57541   0 Aug 29 15:11 hardlink2
```

Reference count: 2

What's in a directory?

```
% ls -pla
drwxr-xr-x  2 brewsteb upg22026   512 Jun 22 16:44 ./
drwxr-xr-x  8 brewsteb ftp       1024 Jun 22 15:46 ../
-rw-r--r--  1 brewsteb upg22026  1027 Jun 22 15:47 cursesDemo.c
-rw-r--r--  1 brewsteb upg22026 42558 Jun 22 15:55 Curses.pdf
-rw-r--r--  1 brewsteb upg22026  4208 Jun 22 16:24 index.html
-rw-r--r--  1 brewsteb upg22026 61554 Jun 22 15:46 IntroToUnixShell.html
-rw-r--r--  1 brewsteb upg22026    38 Jun 22 15:46 OnElFAQ.txt
-rw-----  1 brewsteb upg22026   467 Jun 22 15:46 OnEl_sol.txt
-rw-r--r--  1 brewsteb upg22026   288 Jun 22 15:46 OnEl.txt
-rw-r--r--  1 brewsteb upg22026    38 Jun 22 15:55 Prog1FAQ.txt
-rw-r--r--  1 brewsteb upg22026  8098 Jun 22 15:46 Prog1.html
-rw-r--r--  1 brewsteb upg22026  7114 Jun 22 15:46 Prog1.test
-rw-r--r--  1 brewsteb upg22026    38 Jun 22 15:46 Prog2FAQ.txt
-rw-r--r--  1 brewsteb upg22026  4517 Jun 22 16:14 Prog2.html
```

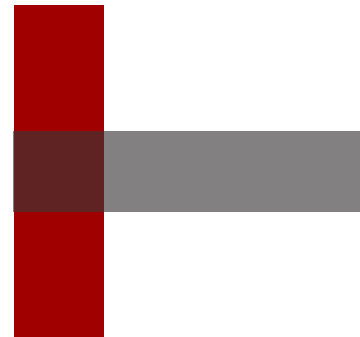
permissions	hard link count	owner	group owner	size (bytes)	last modified	name
-------------	-----------------	-------	-------------	--------------	---------------	------

Possibilities include:

- Hard links
- Symbolic links
- Named pipes
- Device character special file
- Device block special file
- Named socket

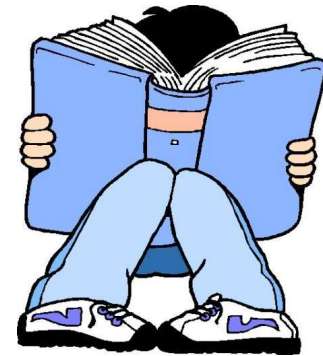
Permissions

- Files in UNIX have access permissions for three classes of users:
 - **u**ser (the owner of the file - can set all permissions)
 - **g**roup
 - all **o**thers
- Three kinds of access permissions for each:
 - **r**ead
 - **w**rite
 - **x**ecute
- Every file belongs to exactly one user and one group



Read Permissions

- File
 - The file's contents can be read
- Directory
 - The directory's contents can be read (ie, a listing of the files in the directory can be returned)



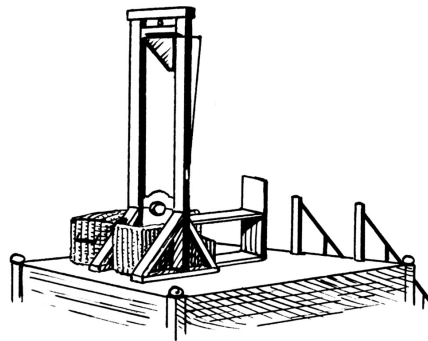
Write Permissions

- File
 - The file can be written to (ie, the contents of the file can be changed)
- Directory
 - Files can be added/removed/renamed/etc. to/in the directory



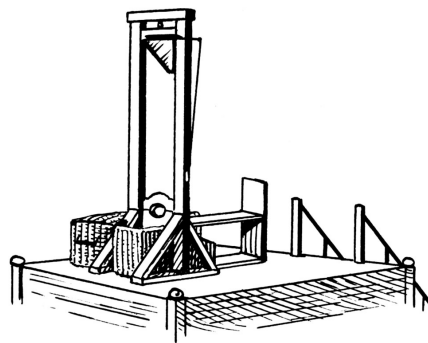
Execute Permissions

- File
 - The file can be executed (program, shell script)
- Directory
 - The directory can be cd'd into
 - File contents can be listed, and meta-information accessed, if name is known



Execute Permissions

- File
 - The file can be executed (program, shell script)
- Directory
 - The directory can be cd'd into
 - File contents can be listed, and meta-information accessed, if name is known



chmod

- You can change the permissions on a file by using the `chmod` (**change mode**) command
- Here is a sample file listing (generated by `ls -pla`) of a file and dir:

The diagram illustrates the components of a file's permissions as shown in the output of the `ls -pla` command. It features a sample file listing with two entries: a regular file named `Prog2.html` and a directory named `tempDir/`. Red dashed vertical lines are used to group the permission characters for each entry. A red callout box labeled "Filetype" points to the first character of the first entry. Three red callout boxes labeled "User", "Group", and "Others" point to the next three characters of the first entry's permissions.

```
-rw-r--r-- 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html
drwx--x--x 1 brewsteb upg22026  512 Jun 22 17:48 tempDir/
```

Filetype

User Group Others

chmod - octal math

- The traditional method of assigning permissions with chmod uses octal
- Why does everything in UNIX have to be so hard?
 - $r = 4$
 - $w = 2$
 - $x = 1$

```
-rw-r--r-- 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html  
drwxr-xr-x 1 brewsteb upg22026 512 Jun 22 17:48 tempDir/
```


$$4 + 2 + 1 = 7$$


$$4 + 0 + 1 = 5$$

```
$ chmod 644 Prog2.html
```

Standard rights for a publicly viewable **webpage**

```
$ chmod 755 tempDir
```

Standard rights for a publicly viewable **directory**

Setting Permissions the Easy Way

- With this file:

```
----- 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html
```

- Set permissions like this:

```
$ chmod u+rx Prog2.html
```

```
-rwx----- 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html
```

```
$ chmod g+rx,o+rx Prog2.html
```

```
-rwxr-xrwx 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html
```

```
$ chmod o-w Prog2.html
```

```
-rwxr-xr-x 1 brewsteb upg22026 4517 Jun 22 16:14 Prog2.html
```

umask

- The *creation mask* setting defines the *default permissions* for new files.
- You can set this mask with the UNIX utility `umask`
- If no argument is included, `umask` displays the current setting



umask

- Since this is a mask we're talking about, it's inverted from what we saw with `chmod`
- Set the default permissions on new files to give the owner full privileges, while the group and all others do not have write privileges:

```
$ umask 022
```

- Note that execute permissions still are often not set by default, even if your umask indicates that they should



umask

```
$ umask
```

```
0007
```

```
$ touch tempfile
```

```
$ ls -pla tempfile
```

```
-rw-rw----. 1 brewsteb upg57541 0 Mar 30 08:17 tempfile
```

```
$ rm tempfile
```

```
rm: remove regular empty file `tempfile'? y
```

```
$ umask 022
```

```
$ umask
```

```
0022
```

```
$ touch tempfile
```

```
$ ls -pla tempfile
```

```
-rw-r--r--. 1 brewsteb upg57541 0 Mar 30 08:18 tempfile
```

Yes, there are actually four digits: the farthest left controls the SUID, SGID, and sticky bits, which are rarely used - you can always leave them off

Note that not all leading zeroes are necessary

What are UNIX groups?

The diagram illustrates the output of the `id` command. It features four red callout boxes with white text, each pointing to a specific part of the command's output:

- My user ID**: Points to the `uid=57541(brewsteb)` part of the output.
- My ID that I am known by in groups**: Points to the `gid=20805(upg57541)` part of the output.
- Personal group consisting of just me**: Points to the `upg57541` part of the `gid` field.
- Other groups I'm a part of**: Points to the list of other groups: `,14013(ecampus-video),19070(ecampusfiles),12028(transfer)`.

```
$ id
uid=57541(brewsteb) gid=20805(upg57541)
groups=20805(upg57541),14013(ecampus-video),19070(ecampusfiles),12028(transfer)
```

- These groups are the same groups referred to when using the `chmod` command

Common UNIX Commands - Directories & Files

- **du** - Returns the total usage in kilobytes of the specified directory

```
$ du .  
4      ./inodetest/1  
4      ./inodetest/2  
16     ./inodetest  
8      ./permissionstests  
96     .
```

Common UNIX Commands - Directories & Files

- **df** - Returns the total usage in kilobytes of filesystems

```
$ pwd
```

```
/nfs/stak/faculty/b/brewsteb/tempdir
```

```
$ df .
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
128.193.40.234:/stak_faculty/data	2147483648	1404256256	743227392	66%	/nfs/stak/faculty

Common UNIX Commands - Directories & Files

- **stat** - Get details about a file

```
$ stat testfile
  File: 'testfile'
  Size: 9          Blocks: 8          IO Block: 262144 regular file
Device: 35h/53d Inode: 3238033175  Links: 1
Access: (0000/-----)  Uid: (57541/brewsteb)   Gid: (20805/upg57541)
Context: system u:object_r:nfs t:s0
Access: 2016-08-30 09:38:11.386951000 -0700
Modify: 2016-08-30 09:40:01.075970000 -0700
Change: 2016-08-30 09:40:11.156727000 -0700
Birth: -

$ touch testfile

$ stat testfile
  File: 'testfile'
  Size: 9          Blocks: 8          IO Block: 262144 regular file
Device: 35h/53d Inode: 3238033175  Links: 1
Access: (0000/-----)  Uid: (57541/brewsteb)   Gid: (20805/upg57541)
Context: system u:object_r:nfs t:s0
Access: 2016-08-30 11:19:33.386465000 -0700
Modify: 2016-08-30 11:19:33.386465000 -0700
Change: 2016-08-30 11:19:33.386480000 -0700
Birth: -
```

Timestamp types

- **Access:** When file was last read
- **Modify:** When file contents were last modified
- **Change:** When meta data about file was last changed (renaming, permissions, etc.)
- **Birth:** Creation date of file (unsupported on Linux, but works on BSD & Windows)

Common UNIX Commands - Directories & Files

- **touch** - Create files and modify time stamps

```
$ ls -l
$ touch testfile
$ ls -l
-rw-rw----. 1 brewsteb upg57541 0 Aug 30 12:02 testfile
$ echo "testtext" > testfile
$ ls -l
-rw-rw----. 1 brewsteb upg57541 9 Aug 30 12:02 testfile
$ touch -d 20120101 fakefile
$ ls -l
-rw-rw----. 1 brewsteb upg57541 0 Jan  1  2012 fakefile
-rw-rw----. 1 brewsteb upg57541 9 Aug 30 12:02 testfile
$ touch -r fakefile testfile
$ ls -l
-rw-rw----. 1 brewsteb upg57541 0 Jan  1  2012 fakefile
-rw-rw----. 1 brewsteb upg57541 9 Jan  1  2012 testfile
```

- Security warning: this is how easy it is to modify the Access and Modify values!
- Only the “Change” value remains unchanged
- You can also arbitrarily update “Change” by setting the system time (with root) to whatever you want, running these commands, then changing the time back

Standard Directories

- Root dir:

- /

- Home dir:

- ~

- Bad idea:

- `rm -rf /*`

Easier to delete everything than you might think!

If you want to delete all of the files in your current directory, and all directories underneath, the command is:

```
$ rm -rf /*
```

I know because I've done it. :(