UCLA CS35L

Week 1

Wednesday

Reminders

- Homework 1 is due next Monday (April 6) by 11:55 PM
- I added a Zoom meeting password to the session to prevent some of the Zoom-bombing issues.

- Longer-Term things but wanted you to have a heads-up now
  - Friday of Week 10 is the last day to turn in homework, the typical late policy deduction won't apply past that
  - The final presentations go better with an audience who will ask questions at the end. So I do expect everyone to actively attend at least one day's worth of presentations. (Can be the same day you present).
- Anonymous feedback for me- https://forms.gle/tZwuMbALe825DBVn8

Questions from Last Lecture?

Note for Class

Add **export PATH="/usr/local/cs/bin:$PATH"** to your **~/.bash_profile** or the **~/.profile file.**

Shell Variables

- The shell allows us to assign variables by assigning things like:

```
a=1
echo $a
```

- But shell variables disappear on logoff, they are specific to the session
- Will come back to these more next week

PATH Variable

- The PATH variable is an environment variable that specifies where executable programs are located
  - Whenever we run a program (like ls) our shell searches through the directory in the PATH variable from left to right
  - If it never finds anything – we get an error
  - Look at your path variable with `echo $PATH`

- Environment variables are a set of variables that the current shell and any child processes of that shell will have access to
  - Look at all of your environment variables with `printenv`

Process

- A process is a running instance of a program. Even the shell itself is a process.
- Each process is associated with a Process ID (PID).
- A process can also spawn other processes, which will be child processes.

Note for Class

Add **export PATH="/usr/local/cs/bin:$PATH"** to your **~/.bash_profile** or the **~/.profile file.**

Export

- Each process has its own memory space.
- So when we create a variable x=1 in the shell process, other programs will not know about the variable x, even if they are spawned by the same shell process.
- However, when we use export, the variables being exported will be visible in all the child processes.

I/O Redirection in the Shell

Most programs read from `stdin` (input to terminal)

Then write to `stdout` (output to terminal)

Send error messages to `stderr`

echo hello -> writes to stdout

cat non-existent-file -> writes to stderr

cat -> waits for stdin, and then writes to stdout

I/O Redirection – Pipeline Operator |

Lets you **PIPE** output from one command as input to a second command.

```
$ who
george              pts/2        Dec 31 16:39 (valley-forge.example.com)
betsy               pts/3        Dec 27 11:07 (flags-r-us.example.com)
benjamin            dtlocal      Dec 27 17:55 (kites.example.com)
jhancock            pts/5        Dec 27 17:55 (:32)
Camus               pts/6        Dec 31 16:22
tolstoy             pts/14       Jan 2 06:42

$ who | wc -l          Count users
6
```

I/O Redirection – <, >, and >>

< takes the file instead of the terminal as `stdin`

```
tr a b < some-file.txt
```

> Redirect `stdout` and make it overwrite the file (erases previous contents)

```
echo 'this will be the new text in the file' > myFile
```

>> Redirect stdout and appends to the file (preserves previous contents)

```
echo 'new line at end of file' >> myFile
```

2> and 2>> same as above, but redirects `stderr` instead.

```
cat non-existent-file > out.log 2> error.log
```

Note for Class

Add **export PATH="/usr/local/cs/bin:$PATH"** to your **~/.bash_profile** or the **~/.profile file.**

Can use stdout redirection like so:

```
cd ~
echo "export PATH=/usr/local/cs/bin:\$PATH" >> .profile
```

NOTE – the \ in front on $PATH is just to prevent the variable from expanding when we execute the command. More on escape characters in Week 3

Linux Wildcards and String Matching

- Useful in searches to match against multiple names
- ? Matches a single occurrence of preceding character
- * Matches zero or more occurrences of preceding character
- [] Matches any of the characters in brackets
  - [Aa] matches ‘A’ or ‘a’
  - [A-Z] matches all uppercase letters
  - [1-5] matches 1, 2, 3, 4, and 5

- Example

```
find . -name *.txt
```

File System

File System Details

- A file system is a mechanism that organizes physical data on disk and provides an interface for their manipulation.
- In Unix file systems, the metadata of every file, such as its location on disk, its owner, permissions, size, modification time and the number of links, are stored in an entity called inode that's particular to the file.
- Each file in the filesystem has a unique inode number.
- All inodes are collected to form an inode table, of which a lightweight version is stored in memory, while the full one is stored on disk

File System Details Cont

- To view inodes we use the command **`ls -i`**
- Reference
  - http://www.tldp.org/LDP/tlk/fs/filesystem.html#tth_sEc9.1.4

Directory Structure Details

- The directory structure is a many-to-one mapping from a representation of files to inodes of the file systems.
- Two entries in the directory structure can map to the same inode in the file system, which is why the directory structure is a **many-to-one** mapping.
- The files and directories we see in the shell is the directory structure representation.

Directory Entry

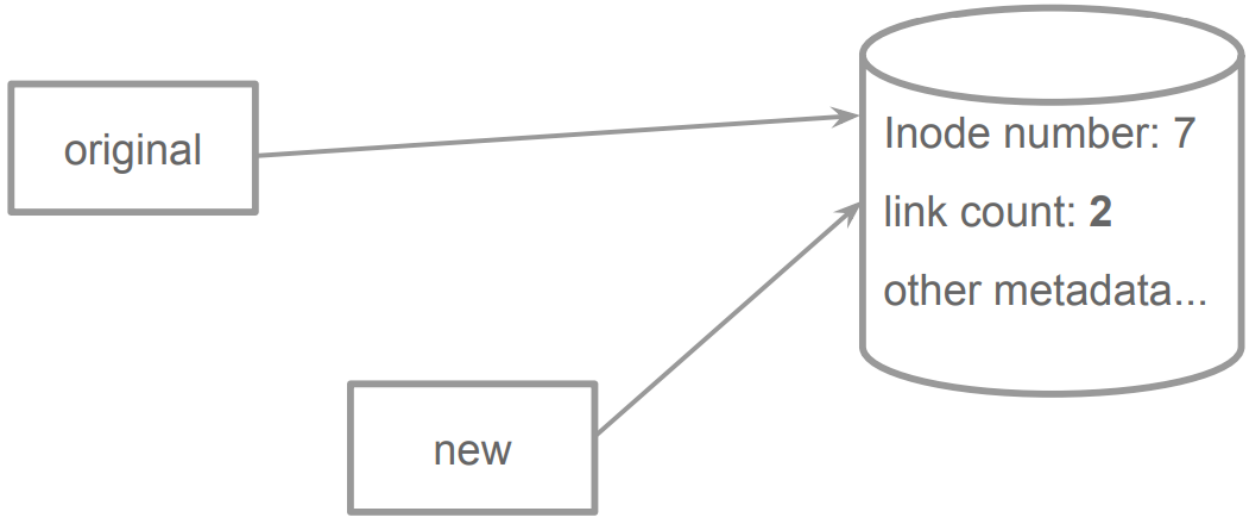
- A Unix directory is a list of association structures, each of which maps one filename to one inode number.
- For example, in the EXT2 file system, a directory is a list of entries, each of which containing the following:
  - Inode - the inode for this directory entry
  - name length - the length of this directory entry in bytes
  - name - the name of this directory entry.
- e.g.
  - inode: 3047909290
  - name length: 20
  - name: /home/meirovit/test.txt

Hard Links

- Two directory entries can map to the same inode, in which case the inode will have a link count of 2.
  - That is, the link count of an inode tracks the number of directory entries being mapped to the inode.

- You create one with

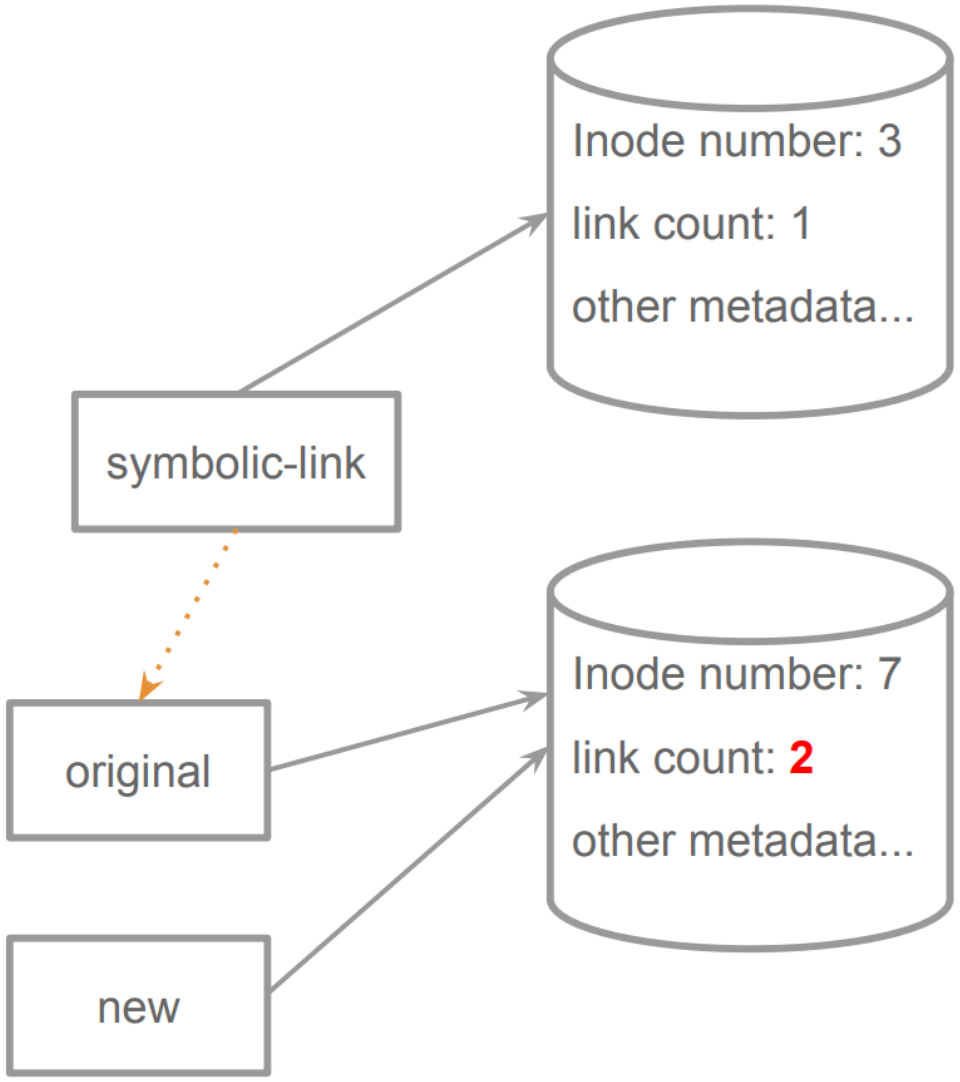
```
ln original new
```



Symbolic Link

- Links to the directory filename instead of the inode.
  - Thus does not increase the link count in the inode
  - Will be dangling if the actual file is deleted.
  - Symbolic links have their own inodes
- You create one with

```
ln -s source target
```



Hard Link Advantages

- Less indirection compared to symbolic links when trying to locate the file/inode, leading to faster speed.
- Deleting the original does not delete the file, since the inode only gets deleted when the link count reaches 0
  - The link and the original have equal footing.

Hard Link Limitations

- Does not work across file systems
  - Inodes and their internal workings are unique to their file system, so they can not be shared.

- Directories cannot be hard linked
  - This can create a cycle between parent and child directories which will create problems for the directory structure.
  - Directory Structure must be a Directed Acyclic Graph (DAG)

ls Revisited, Permissions and chmod

- The first character in the output of ls -l will be:
  - l for symbolic links
  - - for a regular file
  - d for a directory.
- The rest of the characters in the first field indicate the permission bits.
  - The first three bits are permissions for the owner of the file.
  - The next three bits are for the user group that owns with the file.
  - The last three bits are for all other users on the system.
    - r can read
    - w can write
    - x can execute

Permissions and chmod

- Reading the bits
  - **u** owner, first 3 bits
  - **g** user group that owns the file, next 3 bits
  - **o** users not in u and g, last 3 bits
  - **a** all three groups, all 9 bits
- Commands to modify permissions
  - **+** add permissions
  - **-** remove permissions
  - **=** set permissions equal to
- Corresponding flags
  - **r** can read
  - **w** can write
  - **x** can execute
- e.g.
  - chmod u+x some-file
  - chmod go-wx some-file

Permissions and chmod – Binary and Decimal

- Another way to represent permissions is with a number, based on the binary representation of the desired bits.

```
Desired permissions                                  r-xr-x---
Translate to bits                                    101101000
Each three bits becomes the corresponding decimal    5  5  0
```

- For example:
  - `chmod 550 myFile`
  - Will set the permission of myFile to be **`r-xr-x---`**

How to copy Server Files to Local

- If you know **git** already – feel free to use git to create a directory on your local device and the school server to push/pull files
  - If you don’t know git – we will learn it later in this class so don’t worry

- GUI option – Tools like **WinSCP** and **FileZilla** let you create a session to the server and give you a file explorer view. You can then drag and drop files between local and server.

- CLI option – The **scp** command lets you copy files from remote servers to local. The format will be:

```
scp username@lnxsrv#.seas.ucla.edu:foobar.txt /absolute/path/to/directory
```

Assignment 1 Tips

- There is a Hints slide on CCLE which goes into detail on each question
- Most common question – you do not have to actively record each individual keystrokes. Just write down which commands or Emacs shortcuts you used so the grader gets a good idea of what happened.
- Make use of the Emacs reference sheet - https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

- And above all – if any questions than please feel free to ask in Office Hours or through Questionly