soft links

Hard link -> Inode = information node -> file

soft link -> file = windows shortcut

KIT104 ICT Architecture and Operating Systems Tutorial Week 6

1. The UNIX Links

Sometimes it's convenient to have the same file appear in different locations of a file system. For example, when a group of people are working together on the same project, it's convenient to have the project files appear in each group member's account, so that any updates to the files are immediately available to all members of the group. In UNIX, this can be implemented by creating Hard Links, or Soft Links (also called Symbolic Links).

Hard links are duplicate directory entries pointing to the same storage area. Any update to a file is immediately available to all the directories where links are housed. Hard links share the same inode, which means that there is only one copy of the file content on a storage, but it is visible in all the directories which contain its hard links.

Please note, an ordinary file with a hard link count of 2 (one file on disk, with 2 hard links) is different from having 2 copies of the file.

The command line for creating hard link is:

```
$ ln source_file destination_file
```

Soft Links or Symbolic Links are shortcuts pointing to other files. A shortcut and the file it points to are two different files (with different inode numbers). A shortcut is pointing to nowhere if its target file has moved or been removed.

The command line for creating soft links is:

```
$ ln -s source_file destination_file
```

2. The `find` Utility for Locating Files

The `find` command recursively examines a directory tree to look for files either by name or by matching one or more file attributes. The general command line is:

```
$ find path_list selection_criteria action
```

The following are some examples.

```
$ find  /home   -name index.html -print
```

(In the `/home` directory, find all the files which are named `index.html`, and display them on the computer screen. Linux prints the output to screen by default, so `-print` is optional on Linux.)

A sample output of the above command line is as follows:

```
/home/andrew/scripts/reports/index.html
/home/jane/emails/index.html
/home/john/www/index.html
```

the follow path no long exist

use / instead of the file path

```
$ find  .  -name  "*.java"  -print
```
(In the current directory, find all filenames having the `.java` extension and display them on the computer screen. Note the use of double quotes)

```
$ find  .  -name  "[A-Z]*"  -print
```
(In the current directory, search for all filenames beginning with an uppercase letter. `[A-Z]` matches a *single* character that is within the ASCII range of the characters A and Z. Note the use of double quotes)

```
$ find  .  -name  "*.[Dd][Oo][Cc]"  -print
```
(In the current directory, locate all the `.doc` files irrespective of their case. `[Dd]` matches D or d, `[Oo]` matches O or o, `[Cc]` matches C or c. Note the use of double quotes)

```
$ find  /home/adam  -type d  -print
```
(In the `/home/adam` directory, find all directory files. This is for finding files by file type, `d` for directory file, `f` for ordinary file, and `l` for soft link)

```
$ find  /home  -perm 777  -type f  -print
```
(Find files by permissions, and type. This is to search for all ordinary files with permissions `777` in `/home`) type f is ordinary file

```
$ find  /home  -mtime  -2  -print
```
(Find files by modification time. Search for all files that have been modified in less than two days in `/home`)

```
$ find  /home  -mtime +2 -mtime -5  -ls
```
(List those files that were modified in more than two days and less than five days)

```
$ find  /home -mtime +90  -exec rm -f {} \;
```
(Remove all files in `/home` that are more than 90 days old, ie, have not been modified during the past 90 days. The curly braces `{}` represents file names, the `exec` sequence is terminated with `\;`)

```
$ find /home  -size +2000k -atime +90  -exec rm -f {} \;
```
(Find files by size and access time. Remove all files in `/home` that are larger than 2000 Kilobytes and have not been accessed in 90 days. Here `k` is for Kilobytes. You can also use `M` for Megabytes, or `G` for Gigabytes.)

Log in to your UNIX account on alacritas to perform the following actions. find / "*.c" -print
A). Find all C program files (*.c) stored under the root directory.
B). Do you see a lot of "Permission denied" error messages? This is the case because you are searching for files stored on the entire system from a student's account.

The solution is to attach `2>/dev/null` (this is a null device arrangement which will be introduced later) to the end of your command line to eliminate the "Permission denied" error messages. Do this for your task in step A.

C) Are there ordinary files named `ff` stored on alacritas?

```
find / -name ff -type f -print 2>/dev/null
```

D) Are there directory files named `ff` stored on alacritas?

```
find / -name ff -type d -print 2>/dev/null
```

E) Ensure that you are under your home directory. What are all the directory files stored within your home directory tree? What are all the ordinary files stored within your home directory tree? (You may wish to create some sub-directories and under the sub-directories create some ordinary files, and then try to find them)

F) Find all the `.htm` or `.HTM` files stored on alacritas using a single `find` command line.

```
find / -name *.htm -o -name *.html 2>/dev/null
```

G) Find all 2-uppercase character directory filenames (eg, AB, BG) on alacritas.

```
find / -name [A-Z][A-Z] 2>/dev/null
```

H) Are there any ordinary files on alacritas which have the permissions of 777?

```
find / -perm 777 -type f 2>/dev/null
```

I) Are there any directory files on alacritas which have the permissions of 711?

```
find / -perm 711 -type d 2>/dev/null
```

J). Find all ordinary files on alacritas which have been modified in more than 30 days and less than 60 days.

```
find / -mtime +30 -mtime -60 -type f 2>/dev/null
```

K) Find all ordinary files on alacritas which are larger than 10 Megabytes in size.

```
find / -size 10M -type f 2>/dev/null
```

L) Find all ordinary files on alacritas which have not been modified and not been accessed during the past 180 days and are larger than 10 Megabytes. Do not attempt to remove them. (5 marks)

```
find / -atime +180 -mtime +180 -size 10M 2>/dev/null
```

don’t remove files

3. UNIX Shell

3.1 Common UNIX Shells

A UNIX shell’s interpretive cycle is this:

- It issues a prompt and sleeps till you enter a command and hit `[Enter]`;
- It scans the command line for some special characters such as `*` and `?` (called meta-characters, or wild cards);
- It then creates a simplified command line and passes it onto the kernel for execution, and waits for its completion;
- The prompt reappears, the shell returns to its sleeping state.

There are five common shells on UNIX:

- the Bourne shell `(/bin/sh)`
- the Korn shell `(/bin/ksh)`
- the bash shell `(/bin/bash)`
- the C shell `(/bin/csh)`
- the T C shell `(/bin/tcsh)`

A UNIX shell is a system program itself, usually stored under the `/bin` directory. The Bourne shell is the earliest shell that came with UNIX.

Korn shell and Bash shell are supersets of Bourne shell – the Bourne family. Anything that applies to Bourne also applies to Korn and Bash. Some of them do not apply to the C shell and the T C shell. Use the following command to get to know which shell you are using:

```
$ echo $SHELL
```

The Bourne shell was weak as an interpreter but had reasonably strong programming features. The C shell was created to improve the interpretive features of Bourne, but was not suitable for programming. For some time, it was normal to use the C shell for interpretive work and the Bourne shell for programming. Bash was created as a Bourne-again shell, a grand superset in that it combines features of the Korn and the C shells. **Bash is arguably the best shell to use**.

Your login shell is set at the time of your account creation, determined by the last field of your entry in `/etc/passwd`. It is also available in the `SHELL` environment variable. You can make a temporary switch to another shell by running the shell name as a command (to run it as a child process). Use the `exit` command to terminate the child process and return to your login shell.

3.2 The Wild Cards on UNIX Shell

The `*` and `?` which you have previously seen are known as meta-characters, or wild cards:

***** (asterisk) matches zero or more character(s) in a file (or directory) name.
? (question mark) matches a single character in a filename.

For example, the command `ls -l chap*` is equivalent to

```
$ ls -l chap chapx chapy chap01 chap02 chap001
```
(if these files beginning with `chap` exist in the current directory)

Another wild card is `[]`:

- `[ijk]` matches a *single* character - either an `i`, `j`, or `k`
- `[!ijk]` matches a *single* character that is not an `i`, `j`, or `k`
- `[x-z]` matches a *single* character that is within the ASCII range of the characters `x` and `z`
- `[!x-z]` matches a *single* character that is not within the ASCII range of the characters `x` and `z`

Here are some examples:

`$ rm note0[2468]` - removes 4 files: `note02`, `note04`, `note06`, `note08`
`$ rm note[0-1][0-9]` - removes 20 files
`$ rm *.[!o]` – removes all files with a single letter file name extension, except `.o` files

Note the `*` does not match all file names beginning with a dot (`.`). In your home directory, there are many hidden filenames (related to your account configuration) which begin with a dot. To list all hidden files in your home directory having at least two characters after the dot, the dot must be matched explicitly:
```
$ ls  -x  .??*
```

You should now have a better understanding about some of the `find` commands introduced in the previous section of this tutorial:
```
$ find  .  -name  "[A-Z]*"  -print
$ find  .  -name  "*.[Dd][Oo][Cc]"  -print
```

3.3 Escaping on UNIX Shell

A general principle for naming files is that they should not contain shell meta-characters (such as `*` and/or `?`). Imagine that you had a file named `chap*`, and you want to delete it:

`$ rm chap*` - this will delete all the filenames beginning with `chap`.

The correct way to delete this single file named `chap*` is this:

```
$ rm chap\*
```

The backslash (\) here removes the special meaning of `*` (meta-character) so that it becomes an ordinary character. This is called **escaping** on a UNIX shell. The following are more examples related to escaping:

`$ ls chap0\[1-3\]` - This lists one file, which is named `chap0[1-3]`
`$ rm chap0\[1-3\]` - This deletes one file, which is named `chap0[1-3]`

An alternative approach for escaping is to use double or single quotes instead of backslash. In the following examples, the special meaning of any special character enclosed within double or single quotes is removed:

```
$ rm "chap*"  -  this deletes a single file named chap*
$ rm 'chap*'  -  this deletes a single file named chap*
$ rm "chap0[1-3]"  -  this deletes a single file named chap0[1-3]
$ rm 'chap0[1-3]'  -  this deletes a single file named chap0[1-3]
```

4. Hands-on Exercises

A)	Change to the directory `kit104` to make it your current directory. This step will ensure that you do not disturb your other files on the system. If you have removed `kit104` you should create it now and then change to this directory.
	
	But before performing the following steps, first check to see which UNIX shell you are currently using:
	`$ echo $SHELL`
	
	The `echo` command is used to display messages, for example:
	`$ echo Hello World`
	`$ echo This is a test message`

B)	Use the `joe` or `nano` text editor to create a file named `chap01`, with the following C program statements as its content.
	
	`#include <stdio.h>`
	`main()`
	`{`
	`   printf("You are the best in the class.\n");`
	`}`
	
	Note: This is a simple C program which prints a simple message.

C)	Use command `cp` to copy `chap01` into files named `chap02`, `chap03`, `chapx1`, `chapx2`, `chapx3`, `chapy01` and `chapz01`. And, also do the following command: `$ cat chap01 > one.c` This saves the output of the `cat` command into file `one.c`. This is another way for making another copy of `chap01`. Make a copy of `one.c` and name it as `two.c`.
D)	Work out the appropriate pattern in the following command to find the answers to the questions listed below: `$ls` *Pattern* 1. Display all the C programs (indicated by extension `.c`); 2. Display all the files having names beginning with `chapx`;

3. Display all the files having names beginning with `chap` followed by three characters;
4. Run the following commands and think about their outputs:

```
$ ls chap[0-3][0-3]
$ ls chap[x-z][0-3][0-3]
```

5. Create a new file such that the following commands return different outputs:

```
$ls chapx?
$ls chapx*
```

E) Remove all files starting with `chap` using a single and short command line.

F) Run the following command lines to create three ordinary files which are named as `chap*`, `chap1`, and `chap2`, respectively.

```
$ man man > chap*
$ man pwd > chap1
$ man ls > chap2
$ ls
```

G) Use the `ls` command to check the file attributes for `chap*`

```
$ ls -l chap*
```

Do you only see the attributes for the file named `chap*`?

H) Use 3 different approaches to display the file attributes for `chap*` only.

I) All of the following characters have special meanings on a UNIX shell. Can you display each of them using the `echo` command? If you are stuck, press CTRL-C to exit.

```
|    <    >    ′    ″
```

```
$ echo |
$ echo <
$ echo >
$ echo ′
$ echo ″
```

Think about how you can display each of them on the UNIX shell.

J) By default, the `echo` command condenses multiple spaces into one space in a message. In the following example, you want to display a message with 3 spaces between consecutive words, however, in the output, multiple spaces have been condensed into one:

	`$ echo Mon    Tue    Wed`
	`Mon Tue Wed`
	
	Think about what you can do to preserve spaces so that the following message can be displayed with the `echo` command:
	
	`Mon    Tue    Wed`
	(There are 3 spaces between consecutive words)

K)	Under the current directory, create one new sub-directory named `My Docs`. Note that a space is required between `My` and `Docs`. Run `ls -l` to verify its existence. Remove this new sub-directory.
L)	**Run the following command to display a message:** **`$ echo The newline character is \n`** **Observe the output. Why has the backslash (\\) disappeared?** **Use 3 different approaches to display the following message with the `echo` command (5 marks):** **`The newline character is \n`**

echo 'The new line character is \n'

echo The new line character is \\n

(The End)