Exercises-Fileutils3

Filenames and directory structure are an illusion. They don't really exist. Sorry.

In actuality, all data on a computer is represented as numbers. This includes where data exists. What we might know as a file named **`resume`** in our home directory might be known to the system as data object **`#45663`** on device **`#5`**. Our home directory is really a data object that is simply interpreted as a table associating data object numbers with some text names: irrelevant as far as the system is concerned. (These data object numbers are called *inode numbers*, since they refer to the master table that keeps track of all data objects: the *inode table*. The term inode stands for *information node*.) In order to truly understand the subject of this exercise set - links - we must be aware of this distinction.

A link is often referred to as another name for a file. A better definition would be an association between a name and a data object [number]. In this sense, we see that all filenames are links; in fact, links are essentially what are placed in directories. Most data objects that are regular files have a single link. Some regular files have multiple links. All directories have at least two links.

Unix has two types of links: hard links and symbolic (or soft) links. We have been discussing hard links. This is the topic of part one. Symbolic links are the topic of part two. Traditionally, use of the term link implied hard link, which is the origin of the term. Symbolic links, which were added to Unix later, however, have become so common that it is wise to always qualify what you mean if it is important.

Begin by creating a **`links`** directory to work in and connecting to it.

Part One

Description

In this exercise you will practice creating and using hard links. Every association between a name and a data object is a hard link, so a hard link can refer to anything. A user, however, can only create a hard link to a regular file. Creating hard links to other types of objects is reserved to the operating system.

Exercise

You should be in your **`links`** directory.

1. Create a single file named **`resume`** by placing a string of text in it using **`cat`** or **`echo`**. Then list the file with the options **`-li`** Notice the inode number. This is the number that is used internally in Unix to identify the data object. (If we were strict about our terminology here we would say that the name **`resume`** is a link to the data whose inode number is shown)
2. Add a second link to the data using the **`ln`** command. Name the link **`resume_ln`**. List both names with the **`-li`** option. What do you notice? Using the **`cat`** command, display the contents of **`resume_ln`**.
3. Remove the name **`resume`** using **`rm`**. Then list the directory contents again using **`-li`**. Display the contents of the file **`resume_ln`**.
4. Rename **`resume_ln`** to **`resume`**. If a friend walked up to your terminal now, would he know that steps 2 and 3 had occurred? In other words, could he tell that the original **`resume`** had been deleted?
5. Again, create a second link to **`resume`**. Name it **`resume_ln`** again. Then create a directory named **`subdir`** and move **`resume`** to it. Can you **`cat resume_ln`** now? How about **`subdir/resume`**?
6. Try to create a hard link to **`subdir`**. Can you?
7. Change the permissions of **`resume_ln`** to **`777`**. Then list **`resume_ln`** and **`subdir/resume`** using **`-li`**. What happened to the permissions of **`subdir/resume`**? Can you explain this?
8. Overwrite the file **`resume_ln`** with a different string of text. Then use **`cat`** to display both **`resume_ln`** and **`subdir/resume`**. Are you surprised?
9. Using the **`mv`** command, move **`resume_ln`** to the directory **`/tmp`** (If another student doing this assignment did not clean up after themselves, you will find this difficult. You may have to rename **`resume_ln`** as you move it.) Display the contents of your file on **`/tmp`** and the file **`subdir/resume`**.

10. Overwrite **`subdir/resume`** with a different string of text and redisplay both files. Can you explain this? List both using the **`-li`** option. What do you notice about the inode numbers?
11. Last, try to create a hard link named **`link`** to the [non-existent] file **`foo`**. Can you?

Remove the file you created on the /tmp directory when you are finished.

<u>Part Two</u>

<u>Description</u>

In this exercise you will practice creating and using symbolic (soft) links, also called symlinks. A symbolic link acts as if it is a small file that contains a string of text. The string of text could be anything, but it will be interpreted as a path when the link is used, and the system will substitute the path for the name of the symlink. Thus, if a symbolic link named **`myfile`** contains **`dir1/file2`**, the command **`cat myfile`** will result in an attempt to **`cat dir1/file2`**. No checking of the path is performed when the link is created: you must use the link to see if it works.

Unlink hard links, which can only refer to regular files, a symlink can have any path in it. Thus you can have symbolic links to directories, devices, etc.

Note: in this exercise, the options you will be using for **`ls`** are both upper- and lower-case **`L`**s.

<u>Exercise</u>

You should be in your **`links`** directory. You will need a **`resume`** file in your **`links`** directory when you start this exercise. You could either retrieve the one from your **`subdir`** directory or create a new one.

1. Set the permissions of resume to **`644`**. Then make two symbolic links to **`resume`**. The first one should contain a relative path to resume (e.g., **`resume`**). This link should be named **`rel_symlin`**k. The second should contain an absolute path to **`resume`**. It should be named **`abs_symlink`**. You can easily do this by using **`$PWD/resume`** (**`PWD`** is a variable that contains the path to the current directory). Using the **`cat`** command, ensure that the symbolic links work.
2. List the symbolic links and the file they point to using the **`-l`** option. Note the permissions, owner, group, size, and what appears in the name field of the symbolic links. List the symbolic links and resume again using the **`-lL`** options. What is different?
3. Move the **`resume`** file to the parent directory. Try to **`cat`** the file using the symbolic links again. Do they work? What does **`ls -l`** show now? How about if you add the **`-L`** option? Move the **`resume`** file back when you are finished
4. Move the symbolic links to the parent directory. Do the links work now? What happens when you try to list them using **`-Ll`**?
5. Now move the resume file back to the parent directory. Try using the symbolic links again. What happens? Move resume and the symbolic links back to the links directory and ensure that they work correctly.
6. Change permissions of the relative symbolic link to **`600`**. Then list it with **`-l`**. What do you see? List it again after adding the **`-L`** option. What permissions changed?
7. Create a symbolic link and put the string **`unix is weird`** in it. Try to **`cat`** the symlink. Then create a file named **`unix is weird`**, putting a string of text in it. Try to **`cat`** the symlink again. This shows that a symlink and the item it refers to are independent.
8. Create a symbolic link to the current directory. What happens when you run **`ls`** on it?

Delete the **`links`** directory and its contents when you are finished.

Answers

Part One

1.
```
echo "This is resume" > resume
ls -li resume
```
2.
```
ln resume resume_ln
ls -li resume resume_ln
cat resume_ln
Both links have exactly the same information, as they reference the same
object (as indicated by the inode number).
```
3.
```
rm resume
ls -li
cat resume_ln
```
4.
```
mv resume_ln resume
It would be impossible to tell that this was not the original link to
resume.
```
5.
```
ln resume resume_ln
mkdir subdir
mv resume subdir
cat resume_ln subdir/resume   # both still succeed
```
6.
```
ln subdir subdir_ln  # this command fails
```
7.
```
chmod 777 resume_ln
ls -li resume_ln subdir/resume
The permissions of the object changed, so both links show the new
permissions.
```
8.
```
echo "This is the new resume_ln" > resume_ln
cat subdir/resume resume_ln
```
9.
```
mv resume_ln /tmp/resume_ln
cat /tmp/resume_ln subdir/resume  # both files look the same
```
10.
```
echo "This is the newer resume file" > subdir/resume
cat /tmp/resume_ln subdir/resume   # now they have different contents
ls -li /tmp/resume_ln subdir/resume  # now they have different i-numbers
# Note: since /tmp is on a different partition than the original file,
# the mv command became a combination of cp and rm
```
11.
```
ln foo link   # this fails, since foo doesn't exist
```

```
rm /tmp/resume_ln
```

Part Two

```
mv subdir/resume .
```

1.
```
chmod 644 resume
ln -s resume rel_symlink
ln -s $PWD/resume abs_symlink
cat rel_symlink abs_symlink
```
2.
```
ls -l rel_symlink abs_symlink resume
ls -lL rel_symlink abs_symlink resume
# adding -L shows what the link points to, instead of the symlink itself
```
3.
```
mv resume ..
```

```
cat rel_symlink
cat abs_symlink   # neither symbolic link works
ls -l rel_symlink abs_symlink  # cant tell anything
ls -lL rel_symlink abs_symlink # this is what a 'broken' symlink looks
like
mv ../resume .
```

4. `mv *symlink ..`
```
cat ../*symlink # the absolute symlink now works, but not the relative one
```

5. `mv resume ..`
```
cat ../*symlink # now the relative symlink works, but the abs one fails
mv ../*symlink .
mv ../resume .
```

6. `chmod 600 rel_symlink`
```
ls -l rel_symlink    # the chmod didn't seem to work
ls -Ll rel_symlink   # the chmod affected the resume file, not the symlink
# (try listing resume with -l)
```

7. `ln -s 'unix is weird' slink`
```
cat slink          # of course,  this fails
echo hello > 'unix is weird'
cat slink          # now it succeeds
```

8. `ln -s . dot_slink`
```
ls dot_slink  # this lists the current directory
```

```
cd ..
rm -r links
```