Exercises-Fileutils1

Part One

Description

As you know, **`cp`** is used to copy files and directories. We have already covered using **`cp`** to copy files. We will now expand it to copying directories - called *recursive copy*. Here are all the forms of the **`cp`** command - note especially the recursive forms:

Synopsis	***Meaning***
`cp file1 file2`	copy **`file1`** to **`file2`**. **`file2`** is overwritten if it exists.
`cp file1 [... filen] dir`	copy one or more files into an existing directory
`cp -r dir1/. dir2`	copy everything in **`dir1`** to **`dir2`** (including hidden items)
`cp -r dir1 dir2`	copy **`dir1`** and all of its contents to **`dir2`**. If **`dir2`** does not exist, the new copy of **`dir1`** is named **`dir2`**. If **`dir2`** exists, the copy of **`dir1`** is placed in **`dir2`**. If **`dir2/dir1`** exists, the command is the same as **`cp -r dir1/. dir2/dir1`**
`cp -r dir1...dirn dir2`	copy one or more directories and their contents into the existing directory **`dir2`**
other options:	**`-i`** - *interactive copy*. Asks for verification for each individual copy operation before proceeding **`-v`**: *verbose.* Reports each individual copy operation as it is done.

In this first part you will practice recursive copy on a directory structure. It will require some preparation, so listen up and do the preparation carefully:

1. start in your home directory on hills.
2. ensure there is no directory named **`orig`** in your home directory. If there is, rename or remove it. (if you don't, it will be deleted during this exercise)
3. initialize a variable named **`d`** like this: (note there are no spaces on the command-line)
   ```
   d=/pub/cs/gboyd/cs160a/fileutils/part1
   ```
4. initialize a shell alias very carefully like this:
   ```
   alias restart="rm -rf orig;cp -r /pub/cs/gboyd/cs160a/fileutils/part1/orig orig"
   ```
5. invoke the shell alias like this:
   ```
   restart
   ```

Now you should have your starting directory named **`orig`**. Examine its structure and display the contents of each file in the structure. You should also draw a map of it using a tree diagram. Note that each file in the structure has a single line of text that says where the file is. This will be used to identify which files get overwritten by commands you will execute.

In each step of the following Procedure, you will copy one of five tree structures into your **`orig`** structure. Your job is to examine the source tree before the copy, predict what will happen to the **`orig`** structure, then verify the results. After each step, you will use your **`restart`** alias to begin with a fresh **`orig`** tree.

Procedure

At this point, you should have done the **Preparation** section, have your variable and alias correctly set and have a fresh **`orig`** structure in your directory. You should not use the **`cd`** command during this exercise. If you do, you will end up with the next **`orig`** structure in the wrong directory!
The Answers section contains an analysis of the results of each copy operation. You should compare your observations with those answers.

1. Examine the structure of the tree **`$d/A`**. Then execute the command **`cp -r $d/A/* orig`** Predict what happened using your knowledge of **`A`** and **`orig`**, then examine the results. Finally, use your alias **`restart`** to refresh the tree. Here are the commands exactly for this first step:

   ```
    cp -r $d/A/* orig
   <examine the results using ls -RF and cat>
   restart
   ```

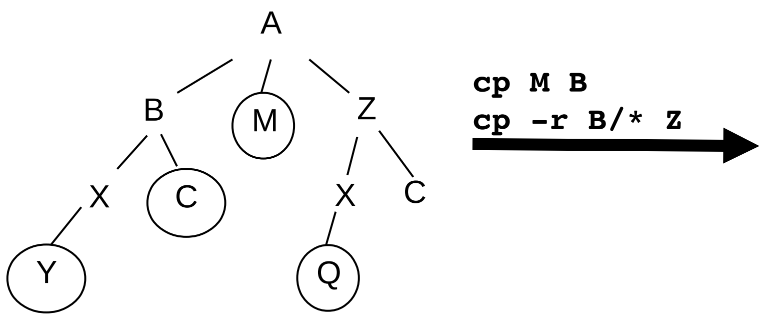
   If you cant figure out what happened, or want to examine the operation in more detail, use **`restart`** to start again, and this time add the -v option to the cp command. This will show the process stepwise.
2. repeat the operation using **`cp -r $d/A/. orig`** What was different about the result compared to 1.? dont forget to **`restart`**
3. repeat the operation using **`$d/B/.`** dont forget to **`restart`**
4. repeat the operation using **`$d/C/.`** dont forget to **`restart`**
5. repeat the operation using **`$d/D/.`** dont forget to **`restart`**
6. repeat the operation using **`$d/E/.`** dont forget to **`restart`**
7. Can you explain why we are using the . at the end of our source paths? What would happen if we didn't? Try it once without the trailing . such as **`cp -r $d/A/ orig`** Can you predict what happened?

Repeat this exercise until you are comfortable with how recursive copy works. When you are done, simply delete the directory **`orig`** to clean up: **`rm -r orig`**

<u>Part Two - using trees</u>

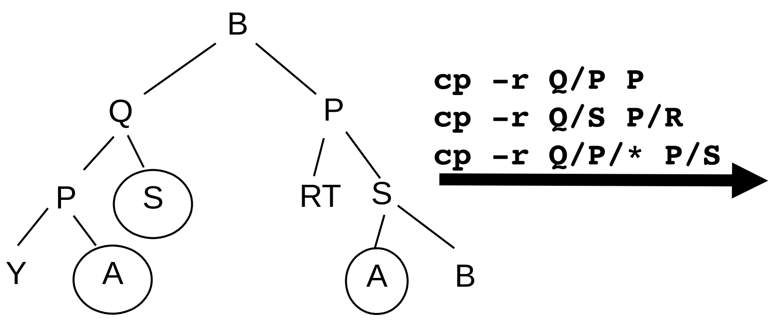
In this exercise you will practice using **`cp`** on directory trees. In each exercise, apply the commands *in order* to the tree on the left and draw the resultant tree on the right. In all trees, you are connected to the top directory (which is **`A`** or **`B`**). To investigate your answers, you can copy the original trees from **`fileutils/part2`** Copy that directory recursively to your home directory. In it you will find **`A`** and **`B.`** **(**You can ignore the **`C`** tree for this exercise. It is only used in FileUtils2.)

1.



```
cp M B
cp -r B/* Z
```

2.



```
cp -r Q/P P
cp -r Q/S P/R
cp -r Q/P/* P/S
```

Answers

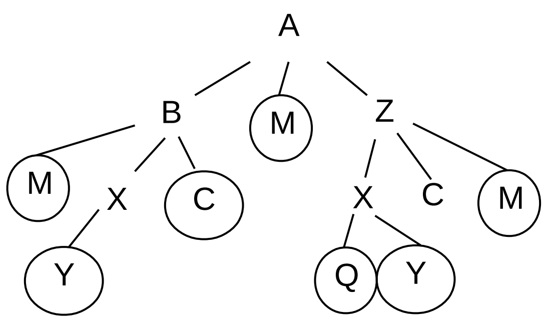
Part One

1. Use **`ls -RF $d/A`** to examine the structure. You can see from the structure, compared to the structure of **`orig`**, that there is a new file **`file1`** in **`A`**, which will be copied to **`dir1`**. There is also a conflicting file **`dir1/a`** that will overwrite the **`dir1/a`** in **`orig`**.
   After the copy, verify that the **`dir1/a`** file was overwritten by **`cat`**-ing it.
   What you may not notice is that there was a hidden file in **`A`**. Did it get copied to **`orig`**? Check it out using **`ls -RFA orig`**
2. This small change of using **`.`** instead of **`*`** affects whether the hidden file got copied to **`orig`**. Verify that it did.
3. In this example, there is a *file* named **`dir2`** in **`A/dir1`**. It conflicts with the existing directory **`dir1/dir2`** in **`orig`**. Since it is illegal for a file to be copied on top of a directory (or vice-versa), that conflict is illegal. The question is whether anything else occurred. Notice that there was a file **`A/dir1/a`**. Did it replace the existing **`dir1/a`** file in **`orig`** even though the copy command got an error? (The answer is yes. A recursive copy consists of many individual copies. Each is done independently. If one gets an error, it goes on and tries the next one.)
4. This time, when you copied the **`C`** directory, **`dir1`** contained a new directory **`dir3`** that did not appear in **`orig`**'s **`dir1`**. The entire directory and its contents got copied to **`orig/dir1`**. Other than this, one file got overwritten (**`orig/dir1/a`**) and one new file got created (**`orig/dir1/abc`**)
5. This time, the file **`dir1/dir2/c`** got overwritten and, again, there is a new directory **`dir1/dir3`** and all its contents.
6. This time, there is a *directory* named **`c`** in **`A/dir1/dir2`**. It conflicts with a file **`orig/dir1/dir2/c`**. Since all file-directory collisions are illegal, you get an error. The file **`dir1/dir2/file1`** did get copied, however.
7. We are using the **`.`** at the end of the source path to indicate *the contents of the directory* instead of *the directory and all its contents*. If the **`.`** is omitted, the directory entry itself is copied, creating a directory **`orig/A`** with the contents of the original **`$d/A`**.

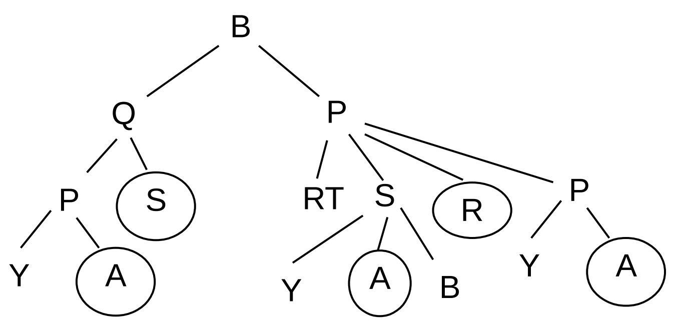
As you can see, recursive copies can be complicated. In general, an attempt is made to merge directories in the two trees, overwriting files when they collide with other files. Note that directory merging may be the result of a mistake rather than what the user wants. Be careful when you are using **`cp -r`** that the paths are correct. You can make a mess of your data!

Part Two

1. (You get an error on the second command due to trying to copy the file **`C`** over the directory **`C`**)



2. (**`P/S/A`** is also overwritten silently with **`Q/P/A`**)


B
Q
P
P
S
RT
S
R
P
Y
A
Y
A
B
Y
A