Manipulate files and data

OS Practice CSC 420 Spring 2022

README

- You can hide/open headers and codeblocks with the <TAB> key
- You can get this file from tinyurl.com/2-manipulate-files-org
- You can get the Emacs configuration file for Pi OS from tinyurl.com/EmacsLyonPi
- The solutions and results herein were all obtained on a Pi 3 running Raspbian GNU/Linux 11 (bullseye).

Emacs setup

You need two open Emacs buffers, ideally:

1. One buffer shows this file while you work on it.
2. The second buffer shows the file system as you change it.
3. Open a second buffer with `C-x 5 2`.
4. You can close it again with `C-x 5 0` or just delete the window.
5. Try this a couple of times now without losing your mind.
6. Remember: to update the Dired buffer enter `g`.

Building a playground

1. Change to your `$HOME` directory
2. Print your working directory
3. Make a directory `playground`
4. Switch on the verbose option `-v`
5. Check that it worked using `ls`, a filter and `grep`

```
cd $HOME
pwd
mkdir --verbose playground
ls ~ | grep playground
```

- `[ ]` Check task group

Creating directories

1. Change directory to `playground`[1] - do this in every code block!
2. Print your working directory
3. Make two directories `dir1` and `dir2`
4. Switch on the verbose option `-v`
5. Check that it worked using `echo` and the regex `*[0-9]`

```
cd ~/playground
pwd
```

```
mkdir --verbose dir1 dir2
echo "found: " *[0-9]
```

- [X] Check task group

Copying files

1. Copy `/etc/passwd` into the current working directory (`playground`)
2. Switch on the verbose option `-v`
3. Check that it worked using `echo` and the regex `pass??`

```
cd ~/playground
pwd
cp -v /etc/passwd .
echo "found: " pass??
```

- [X] Check task group

Moving and renaming files

1. Change the name of `passwd` to `fun`
2. Switch on the verbose option
3. Check that it worked with the wildcard `*fun*`

```
cd ~/playground
pwd
mv -v passwd fun
echo "found: " *fun*
```

1. Move the renamed file `fun` to directory `dir1`
2. Check that it worked with `ls -l`
3. Move `fun` from `dir1 to ~dir2` in one command
4. Check that it worked with `ls -l`
5. Move `fun` back to the current working directory
6. Check that it worked with `ls -l`

```
cd ~/playground
pwd
mv -v fun dir1
mv -v dir1/fun dir2
mv -v dir2/fun .
ls -l .
```

1. Move file `fun` into `dir1` again
2. Move directory `dir1` into `dir2`
3. Confirm that the file is there with `ls -l`

```
cd ~/playground
pwd
mv -v fun dir1
```

```
mv -v dir1 dir2
ls -l dir2/dir1
```

- Note that `dir1` was moved into `dir2` because it existed
- If it had not existed, `dir1` would have been renamed `dir2`
- Put everthing back and confirm at the end with `ls -l`:
    1. move `dir1` back to `playground`
    2. move `fun` from `dir1` back to `playground`
    3. always use the verbose flag `-v`

```
cd ~/playground
pwd
mv -v dir2/dir1 ./dir1
mv -v dir1/fun .
ls -l .
```

- `[X]` Check task group

Creating hard links

1. Create a hard link `fun-hard` to `fun` in `./`
2. Create a hard link `fun-hard` to `fun` in `dir1`
3. Create a hard link `fun-hard` to `fun` in `dir2`
4. Switch on the verbose option for `ln`
5. Confirm with `ls -l ./` and with `ls -l dir*`

```
cd ~/playground
pwd
ln -v fun fun-hard
ln -v fun dir1/fun-hard
ln -v fun dir2/fun-hard
ls -l .
ls -l dir*
```

- The number `4` in the listing is the number of hard links that exist for the file (including the default link)
- Show that `fun` and `fun-hard` are identical with `ls -li`
- The first column shows the file's `inode` (meta data)

```
cd ~/playground
pwd
ls -li fun*
```

- `[X]` Check task group

Creating symbolic links

1. Create a symlink `fun-sym` to `fun` in `./`
2. Create a symlink `fun-sym` to `fun` in `dir1`
3. Create a symlink `fun-sym` to `fun` in `dir2`
4. Switch on the verbose option for `ln`
5. Confirm with `ls -l ./` and with `ls -l dir*`

```
cd ~/playground
pwd
ln -vs fun fun-sym
ln -vs fun dir1/fun-sym
ln -vs fun dir2/fun-sym
ln -l .
ls -l dir*
```

1. Create a symlink `dir1-sym` to `dir1` in `./`

```
cd ~/playground
pwd
ln -vs dir1 dir1-sym
ls -l ./dir1*
```

1. Check the `inode` values in `playground`.

```
ls -li ~/playground
```

1. Test the links by changing to the Dired buffer (`C-x 5 o`)
2. `[X]` Check task group

Removing files and directories

1. Remove the hard link `fun-hard` in `./` (with verbose option)
2. Confirm with `ls -l~`
3. Check in the Dired buffer

```
cd ~/playground
rm -v fun-hard
```

1. Create a file `y` and put `y` into it: `echo "y" > y`
2. Remove `fun` and switch on verbose option[2]
3. Confirm with `ls -l`

```
cd ~/playground
echo y > y
rm -iv fun
ls -l
```

- `[ ]`

  In a shell, check that `fun-sym` is broken now with `cat`. You should get:

  ```
  fun-sym: No such file or directory
  ```

- `[ ]` Make sure that you understand what "broken symbolic link" in this context means, and why `fun-sym` is now broken
- Remove the symbolic links (switch on verbose option)
- Confirm with `ls -l`

```
cd ~/playground
pwd
rm -v fun-sym dir1-sym
ls -l
```

- Go `$HOME` and remove the playground (with verbose option)
- Check with `ls -vl`

```
cd ~/
pwd
rm -vr playground
ls -vl
```

- `[X]` Check this last task group
- Save this file with `C-x C-s`
- Kill the buffer with `C-x k` (confirm)

You may close Emacs!

Command summary

- `[ ]` Complete the table!

COMMAND	MEANING	EXAMPLE
cd		
pwd		
mkdir		
echo		
mv -v		
rm -vr		
ln -vs		
ls -l		

Footnotes:

[1] If you work with code blocks inside Emacs, you may have to resort to absolute filenames to make sure that you are where you want to be.

[2] In Org-mode, you need to use the `:cmdline` header argument and redirect the input, in this case from a file `y` that only contains the character `y`, which I created for this purpose.

Author: [yourname]
Created: 2024-02-01 Thu 19:50