Shell Command

Author	Md. Zarif Ul Alam
Date Created	December 2, 2021 2:24 PM
Tag	CSE 313 Operating System
Property	Sessional

Shell Commands

pwd

- `pwd` : prints ***current path***

cd

- `cd` : directs to `/home/username` in unix or `/c/Users/username` in windows
- `cd ..` : Goes to ***parent*** of current directory
- `cd /abc` : using absolute paths, `abc` start from the root folder `/`
- `cd /mnt` : ***Mount*** Drives
- `cd abc` : Goes to *folder* `abc` in this directory
- `cd "abc"` : goes to path `"abc"` in this directory
- `explorer.exe .` : opens in **windows file explorer**

ls

- `ls` : ***Lists*** all *folders* in this directory
- `ls -a` : lists hidden files also
- `ls -l` : lists in details

  You have, from left to right:

  - the file *permissions* (and if your system supports ACLs, you get an ACL flag as well)
  - the *number of links* to that file
  - the *owner* of the file
  - the *group* of the file
  - the file *size* in bytes
  - the file *modified datetime*
  - the file *name*

  Note

  - This set of data is also generated by the `l` option.
  - The `a` option instead also shows the ***hidden files***. Hidden files are files that start with a dot (.)
  - For file permission : r = read , w = write , x = executions access
- `ls -R` : recursively show all files and folders
- `ls -al` : possible to concatenate multiple options together
- `ls *.file_type` : **anything that matches the file type**
- `ls demo*` : anything starts with the name ***demo***
- `ls demo?` : has one character after ***demo***
- `tree` // extra tool
- `broot` // extra tool

mkdir

- `mkdir abc` : ***creates*** *folder* named `abc`

- `mkdir fruits cars` : ***creates*** *multiple folders* at once
- `mkdir new_folder/sub_folder` : won't work unless **`new_folder`** is already created
- `mkdir -p fruits/apples` : ***creates*** *multiple* ***nested*** *folders* by adding the `p` option
- `mkdir a/_a b/_b` : create multiple folders at once given that `a` and `b` already exists

rm

- `rm abc` : ***deletes*** the file `abc` (can't delete folder with this)
  - `-f` : ignores non-existent files, never prompts
  - `-i` : interactive. prompts before deleting every file and folder
  - `-r` : removes content recursively
  - `-v` : tells deletions step by step
- `rm -rf abc` : force ***deletes*** *files* and *folders* and everything included
- `rmdir abc` : ***deletes*** *folder* ***(The folder you delete must be empty.)***

touch

- `touch abc` : ***creates*** an *empty file* (If the file already exists, silently fails)

cp

- `cp src dst` : copies file
- `cp -r src dst` : copies directories recursively
- `cp -i src dst` : interactive prompt before overwriting

mv

- `mv abc new_abc` : if there is no folder , `abc` will be ***renamed*** to `new_abc` . Otherwise `abc` will be moved to that *folder*
    - also possible to rename files using this same technique
- `mv a1 a2 abc` : ***moves*** `a1` , `a2` *files* to *folder* `abc`

pushd

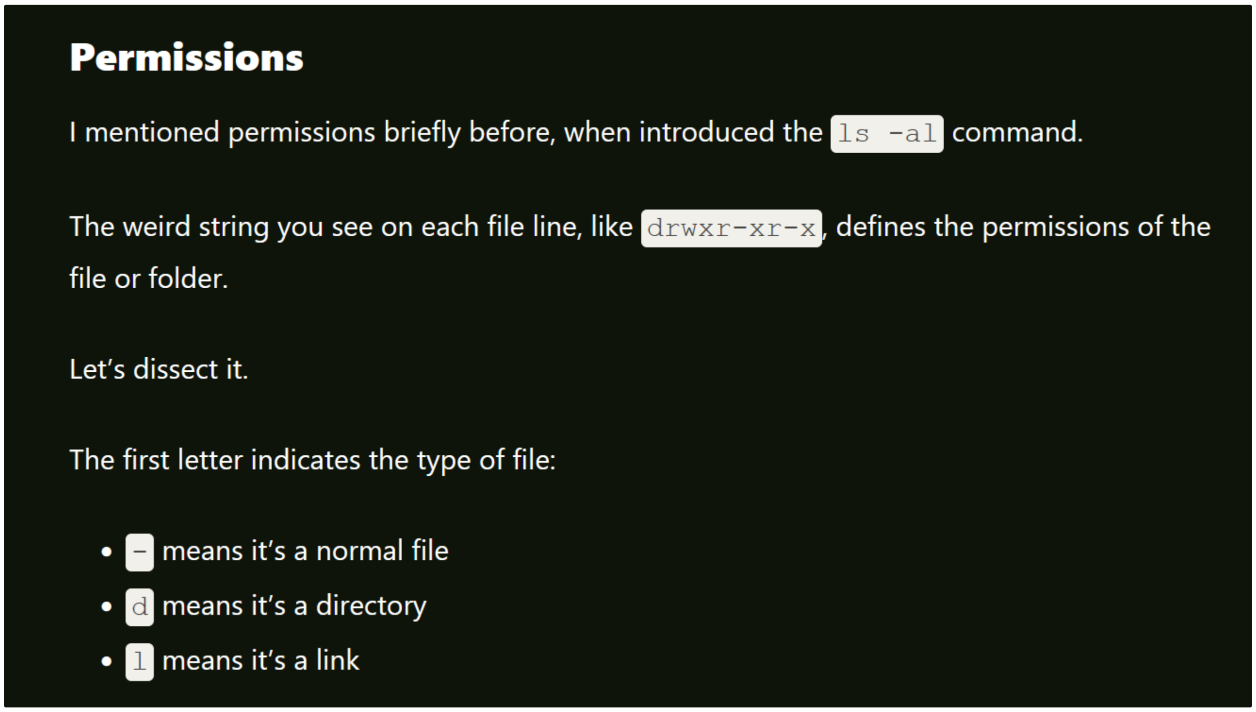
- `pushd .` : pushes current directory to the stack

popd

- `popd` : is equivalent to cd of the last directory that was pushed

chmod

- `chmod` : changes read write execute ***permissions*** (**Details**)



Permissions

I mentioned permissions briefly before, when introduced the `ls -al` command.

The weird string you see on each file line, like `drwxr-xr-x`, defines the permissions of the file or folder.

Let's dissect it.

The first letter indicates the type of file:

- `-` means it's a normal file
- `d` means it's a directory
- `l` means it's a link

Then you have 3 sets of values:

- The first set represents the permissions of the **owner** of the file
- The second set represents the permissions of the members of the **group** the file is associated to
- The third set represents the permissions of the **everyone else**

Those sets are composed by 3 values. `rwx` means that specific *persona* has read, write and execution access. Anything that is removed is swapped with a `-`, which lets you form various combinations of values and relative permissions: `rw-`, `r--`, `r-x`, and so on.

You can change the permissions given to a file using the `chmod` command.

`chmod` can be used in 2 ways. The first is using symbolic arguments, the second is using numeric arguments. Let's start with symbols first, which is more intuitive.

You type `chmod` followed by a space, and a letter:

- `a` stands for *all*
- `u` stands for *user*
- `g` stands for *group*
- `o` stands for *others*

Then you type either `+` or `-` to add a permission, or to remove it. Then you enter one or more permissions symbols (`r`, `w`, `x`).

All followed by the file or folder name.

Here are some examples:

```
chmod a+r filename #everyone can now read
chmod a+rw filename #everyone can now read and write
chmod o-rwx filename #others (not the owner, not in the same group of t
he file) cannot read, write or execute the file
```

You can apply the same permissions to multiple personas by adding multiple letters before the `+`/`-`:

```
chmod og-r filename #other and group can't read any more
```

In case you are editing a folder, you can apply the permissions to every file contained in that folder using the `-r` (recursive) flag.

Numeric arguments are faster but I find them hard to remember when you are not using them day to day. You use a digit that represents the permissions of the persona. This number value can be a maximum of 7, and it's calculated in this way:

- `1` if has execution permission
- `2` if has write permission
- `4` if has read permission

This gives us 4 combinations:

- `0` no permissions
- `1` can execute
- `2` can write
- `3` can write, execute
- `4` can read
- `5` can read, execute
- `6` can read, write
- `7` can read, write and execute

We use them in pairs of 3, to set the permissions of all the 3 groups altogether:

```
chmod 777 filename
chmod 755 filename
chmod 644 filename
```

Owner and group

You can change the owner of a file using the `chown` command:

```
chown <username> <filename>
```

You can change the group of a file using the `chgrp` command:

```
chgrp <group> <filename>
```

chown | chgrp

- `chown` : change ow]ner
- `chgrp` : change group

cat

- `cat in1` or `cat < in1` : ***Reads*** file
- `cat -n in1` : prints content of `in1` with line numbers
- `cat in1 in2` : print contents of multiple files

- `cat > in1` : ***Writes*** to file and then `ctrl+c` to exit editing
- `cat >> in1` : ***Appends*** to file and then `ctrl+c` to exit editing
- `cat < in1 > in2` concatenate contents of single file into new file , takes from `in1` and copies to `in2`
- `cat in1 in2 > in3` : concatenate contents of multiple files into new file
- `cat in3 | anothercommand` : add another command using `|`

wc

word count

- `wc -l file1` : count ***lines***
- `wc -w file1` : count ***words***
- `wc -c file1` : count ***characters***
- `wc -m file1` : count ***characters with multibyte support*** (i.e. emojis count as 1, not as multiple characters)
- `wc -m in*` : operate wc -m on all files starting with `in`

find

- `find . -name '*.txt'` : Finds all *files* with `.txt` extension

- `find . -name 'in*'` : Finds all *files* and *folders* starting with in
    - `find . -type d -name "in"` : search only *directory*
    - `find . -type f -name "in"` : search only *files*
    - `find . -type l -name "in"` : search only *links*
    - `find . -type f -name 'in*' -mtime +3` : Search files ***edited more than 3 days ago***
    - `find . -type f -name 'in*' -mtime -1` : Search files ***edited in the last 24 hours***
    - `find . -type f -mtime -1 -delete` : delete those who meet the criterias
    - `find . -type f -size +100c` : search files that have ***more than 100 characters (bytes) in them***
    - `find . -type f -size +100k -size -1M` : search ***file size bigger than 100KB but smaller than 1MB***
- `find . -type f -exec cat {} \;` execute a command on each result of the search . ***{} is filled with the file name at execution time***.
- `fd "*.py"` // ?

locate

- `locate py`

grep

- `grep foobar mcd.sh` : search `foobar` in `mcd.sh`
- `grep -R foobar .` : search `foobar` in all files

```
# Find all python files where I used the requests library rg -t py
'import requests' # Find all python files where I used the requests
library with 5 lines of context rg -t py -C 5 'import requests' # Find
all files (including hidden files) without a shebang line rg -u --
files-without-match "^#!" # Find all matches of foo and print the
following 5 lines rg foo -A 5 # Print statistics of matches (# of
matched lines and files ) rg --stats PATTERN
```

cut

- `cut -d ' ' -f 1,3 test.txt -output-delimiter='__'` : takes every line from file, uses space as delimiter and prints 1st and 3rd element after tokenization, also output delimiter is defined as `__`
- `echo 'drüberspringen' | cut -b 1-5` : cuts bytewise , `ü` takes 2 byte , `-b` and `-c` behaves same