Managing Files and Directories

General Knowledge

Options and Arguments:

- Commands are often followed by options that modify/enhance their behavior.
- Commands are also followed by arguments which are the items open which the command acts on.

```
command -option argument
```

```
ls -l ~/Downloads
```

Creating files and Directories

Creating directories

The *mkdi*r command

- *mkdir* is used for creating a single directory or multiple directories.

  > mkdir + the name of the directory.

- To create multiple directories, separate each directory name with a space.
- If you try to create a directory that already exists, you will get an error notifying you that the file already exists.

Examples of the *mkdir* command

- Create a directory in the present working directory
  - `mkdir wallpapers`
- Create a directory in a different directory using relative path
  - `mkdir wallpapers/ocean`
- Create a directory in a different directory using absolute path
  - `mkdir ~/wallpapers/forest`
- Create a directory with a space in the name
  - `mkdir wallpapers/new\ cars`
  - `mkdir wallpapers/'cities usa'`
- Create a directory with a single quote in the name
  - `mkdir wallpapers/"majora's mask"`
- Create multiple directories
  - `mkdir wallpapers/cars wallpapers/cities wallpapers/forest`
- Create a directory with a parent directory at the same time.
  - `mkdir -p wallpapers_others/movies`

Creating Files

- The touch command
  - touch is used for creating files
    - Examples:
      - To create a file called list
        - **touch** list
      - To create several files:
        - **touch** list_of_cars.txt Script.py names.csv
      - To create a file using absolute path:
        - **touch** ~/Downloads/games.txt

> **Note: Creating files is not the designed purpose of the touch command. The touch command updates any given file's timestamp. But, if the file does not exists, it creates it.

Deleting files and directories

- The **rm** command
  - **rm** removes files. Does not removes directories.
  - To remove a non-empty directory use **rm** with the **-r** option.
  - To remove empty directories use the **rmdir** command.

Examples of the *rm* command

- Remove a file
  - `rm list`
- Remove a file and prompt confirmation before removal
  - `rm -i list`
- Remove all the files inside a directory and ask before removing more than than 3 files
  - `rm -I Downloads/games/*`
- Remove an empty directory
  - `rmdir Downloads/games`
- Remove an non-empty directory
  - `rm -r Downloads/games`

Moving and copying files and directories

Moving files and directoires

The *mv* command

- mv moves and renames directores.
  - > **mv** + source + destination
  - > **nv** + file/directory to rename + new name
- Both source and destination can be an **absolute or a relative path**.

Examples of moving files and directories

- To move a file from a directory to another using relative path
  - `mv Downloads/homework.pdf Documents/`
- To move a directory from one directory to another using absolute path
  - `sudo mv ~/Downloads/theme /usr/share/themes`
    - *Notice that in this command I am using sudo since the destination is owned by root.*
- To move a file from one directory to another combining absolute path and relative path
  - `mv Downloads/english_homework.docx /media/student/flashdrive/`
    - *Notice that in this command I am moving the file "english_homework.docx" to the directory where the flash drive is mounted.*
- To move multiple directories/files to a different directory
  - `mv games/ wallpapers/ rockmusic/ /media/student/flashdrive/`

Examples of renaming files and directories

- To rename a file
  - **mv** homework.docs cis106homework.docx

- To rename a file using absolute path
  - **mv** ~/Downloads/homework.docs ~/Downloads/cis106homework.docx
- To move and rename a file in the same command
  - **mv** Downloads/cis106homework.docs Documents/new_cis106homework.docx

Copying files and directories

The *cp* command

- **cp** copies files/directories from a source o a destination.
- the **cp** command used the same structure as the **mv**
  - > **cp** + files to copy + destination
- To copy directories you must use the **-r** option
  - > **cp -r** + directories to copy + destination

Examples of coying files and directories

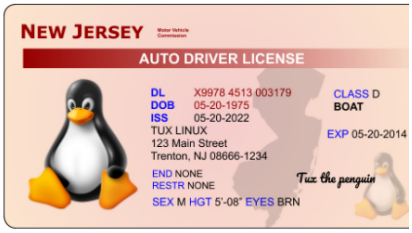
- To copy a file
  - `cp Downloads/wallpapers.zip Pictures/`
- To copy a directory with absolute path
  - `cp -r ~/Downloads/wallpapers ~/Pictures/`
- To copy the content of a directory to another directory
  - `cp Downloads/wallpapers/* ~/Pictures/`
- To copy multiple files in a single command
  - `sudo cp -r script.sh program.py home.html assets/ /var/www/html/`

Working with links

Inodes (index files)

What is an inode?

- A data structure that contains all the information about a file except the file name and its content
- Every file in the file system has an inode
- Each inode is identified by an inode number (index number)
- An inode number references an entry in the inode table
- The inode table is a database of the location of the data on a Linux partition
- To view a file's inode number, use the ls -i command
- To display the inode data on a file or directory use the stat command
- Examples:
  - `stat script.sh`
- Each partition on a hard drive has its own inode table and every file has a unique inode number.



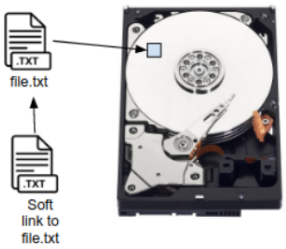
Think of an inode as an ID for each file in the file system that does not have the file name or content.

Hard Links

Soft Links

- **Symbolic links (soft links)** are a special type of file that point to other files instead of data in the hard drive
- Soft links do not share the same inode number as hard link do
- If you modify a soft link, the target file is modified too
- The advantage of soft links is that they can point to files that are store on different partitions
- To create a symbolic link: `ln -s file fileSL`

IMPORTANT: *The file system must support symbolic links in order for you to create links in it. FAT32 does not support links!*



If the concept of soft links is not clicking, think of them as shortcuts in Windows

Getting Help

- Man (manual) pages are documentation files that describe Linux shell commands, executable programs, system calls, special files, and so forth.
- To view the manual of a command type: **man** + command.
    - > Example: **man ls**
- To exit the **man** page press letter "**q**".

Section	Description	Examples
1	Executable programs or shell commands	`man ls`, `man pwd`
2	System calls, which are system requests that programs make to the kernel	`man kill`, `man read`
3	Library calls (to access functions in program libraries)	`man xcrypt`, `man stdin`
4	Special files, such as the floppy disk, that are usually found in `/dev`	`man fd`, `man tty`
5	File formats and conventions	`man passwd`, `man hosts`
6	Games	`man tetravex`, `man AisleRiot`
7	Macro packages and conventions	`man man (7)`, `man gruff (7)`
8	System administration commands	`man yast`, `man suseconfig`

Other ways of getting help

- Most command have a help option built in. Normally that option is accessible by using -h or --h or --help
- Read error messages carefully. Most of the time the answer is in the error itself.
- Some commands may not have a man page but an info page. To read the info page of a command use the info command in a similar way of how you use the man command.
- You can use the apropos command to see the manual page name and description of a particular command
- You can use the whatis command to display a simple description of what a command does.
- There is also a nice program called cheat that you can install in your system to see examples of what a particular command does.
- To learn more about cheat visit the github page: https://github.com/cheat/cheat
- To instal cheat use the command: **`sudo snap install cheat`**
- To use cheat just type cheat + the command. If there is a cheat page available, it will be displayed
- And as it is the case with almost everything, GOOGLE IT! In this case google the error message.

Using *wildcards* (file globbing)

Using Wildcards / File Globbing

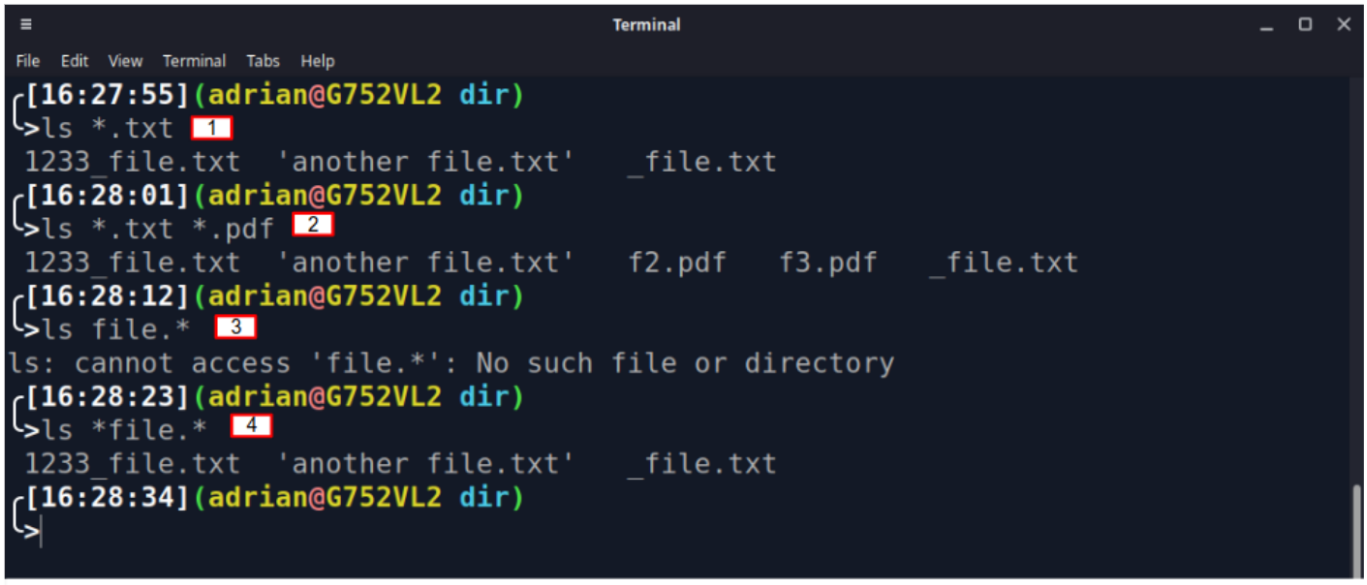
- Wildcard represents letters and characters used to specify a filename for searches.
- File globbing is the processing of pattern matching using wildcards.
- The wildcards are officially called metacharacter wildcards.
- For Example:
  - You can use a wildcard to get a long list of all files in the current directory starting with "new."
  - Use wildcards to manage to directories faster
  - Move or delete a group of files
  - Locate files based on a portion of their filenames
  - Create files and directories quicker

Important note:
Wildcards are not regular expressions. A regular expression is a sequence of characters that define a search pattern.

The *Wildcard

- The main wildcard is a star, or asterisk (*) character.
- The **star alone** matches anything and nothing and matches any number of characters.
  - For example, **ls *.txt**will match all files that end in **.txt**.
  - Examples of when to use **the wildcard:*
    - to list all files with a particular file extension.
    - Do not remember the complete name of a file but you remember a prtion of the name.
    - Want to copy, move, or remove all files that match a particular naming convention.

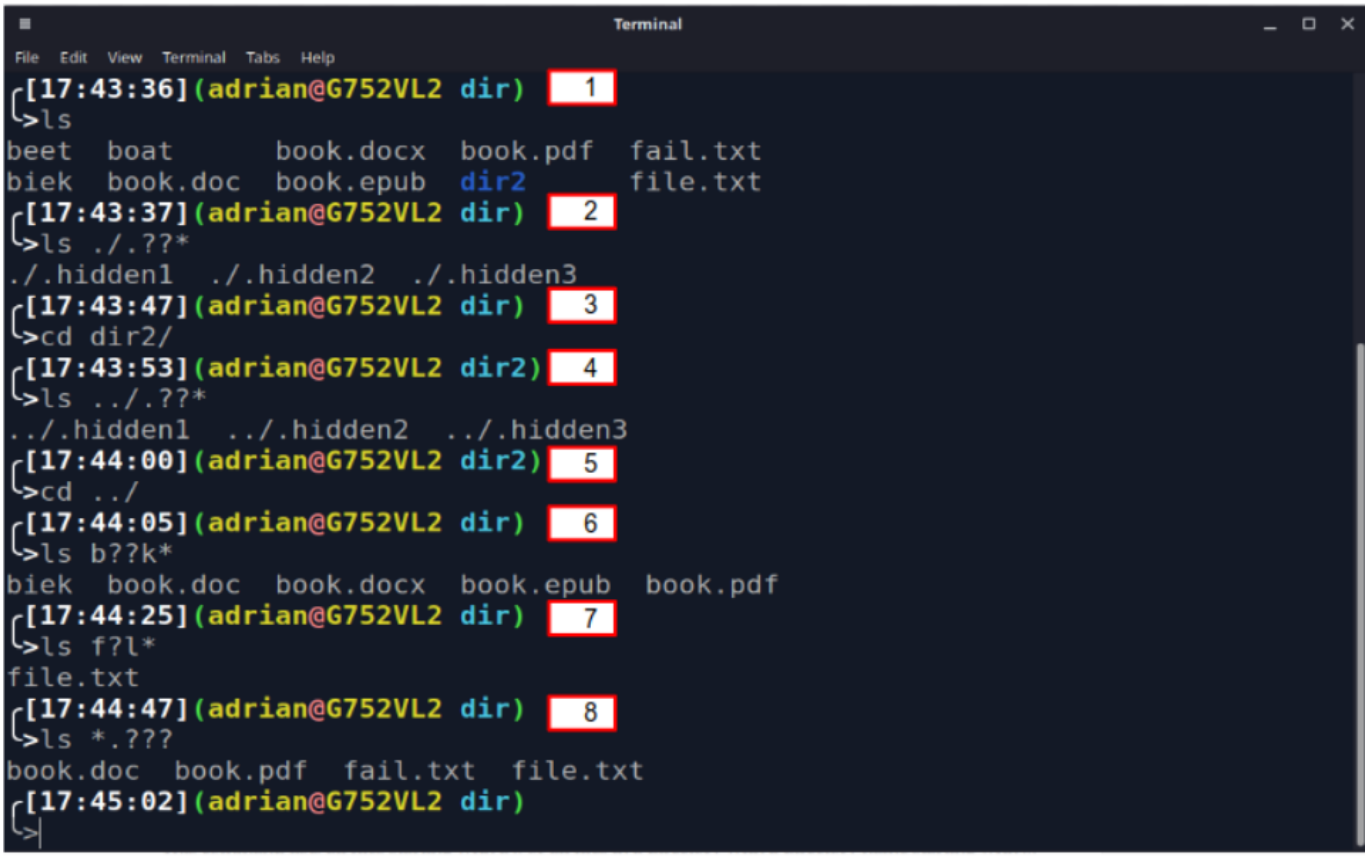
Examples of *Wildcard



1. ls *.txt lists all the files that end in .txt
2. ls *.txt *.pdf list all the files that end in .txt and .pdf
3. ls file.* lists all the files that start with the string "file." regardless of their file extension. In this example, there were no files in the directory that matched this criteria.
4. ls *file.* list all the files that have any letter before the string "file." and after as well.

The *? Wildcard*

- The ? wildcard metacharacter matches **precisely one character**. You might need the question mark to minimize a long list of files names down to a few.
- In addition, the question mark proves very useful when working with hidden files (hidden files are also called **dot files**).
- If you want to list all hidden files you can use: `ls .??*` which will match all files that start with a . or .. and have any character after it.
- Isn't this the same as using the * character on its own? **NO!**
  - The problem with dot files and wildcard expressions is that the **current directory** and the **parent directory** have a name.
  - The current directory is called/represented with a single dot (.) and the parent directory is called/represented with two dots (..)
    - *Remember cd ../../../ that's what I am talking about!*
  - If you use a wildcard expression such as `.*` to list all files that start with a dot. The shell will also match . and ..
  - To go around this problem you can use `./.*` to match all the files in the current directory with a file name starting with a dot.
  - You can also match all the files that start with a . in the parent directory using `../.*`



1. List all the files in the current directory (excluding hidden files)
2. List all the hidden files in the current directory
3. Changes the current working directory to dir2
4. List all the hidden files in the parent directory
5. Changes the current working directory to the previous directory (dir)
6. List all the files that have a two character between letter b and k.
7. List all the files that have a single character between letter f and l.
8. List all the files that have a 3 letter file extension.

The *[] Wildcard*

- The brackets wildcard match a single character in a range.
- The brackets wildcard use the exclamation mark to reverse the match. For example, match everything except vowels [!aeiou] or any character except numbers [!0-9]
- **Examples:**
  - To match all files that have a vowel after letter f:
    - `ls f[aeiou]*`
  - To match all files that do not have a vowel after letter f:
    - `ls f[!aeiou]*`
  - To match all files that have a range of letters after f:
    - `ls f[a-z]*`
  - To match all files whose name has at least one number:
    - `ls *[0-9]*`
  - To match all the files whose name does not have a number in their file name:
    - `ls *[!0-9].*`
  - To match all files whose name begins with a letter from a-p or start with letters s or c:
    - `ls [a-psc]*`
  - To match all files whose name begins with any of these two sets of characters: letters from a-f or p-z:
    - `ls [a-fp-z]*`
  - To match all files whose name begins with any 3 combination of numbers and the current user's username:
    - `ls [0-9][0-9][0-9]$USER`

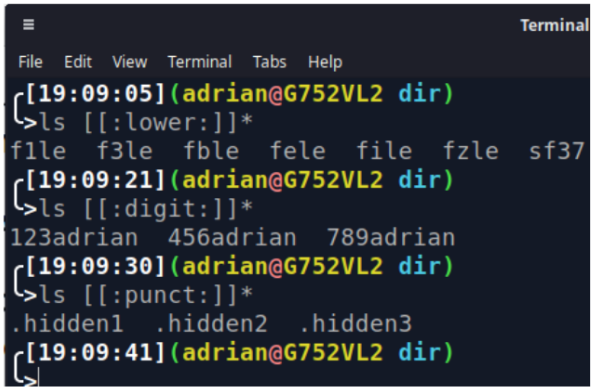
31

![exampleBrak](../notes/imgs4/exampleBrak.png

The [] wildcard with Regex Character Classes

You can use POSIX or Character Classes with the [] wildcard

POSIX class	Represents	Means
[:upper:]	[A-Z]	Upper case letters
[:lower:]	[a-z]	Lower case letters
[:alpha:]	[A-Za-z]	Upper and Lower case letters
[:alnum:]	[A-Za-z0-9]	Lower case, upper case, and digits
[:digit:]	[0-9]	digits
[:xdigit:]	[0-9A-Fa-f]	hexadecimal digits
[:punct:]	[.,!?:...]	puctuation
[:blank:]	[\t]	space and tabs
[:cntrl:]	n/a	control characters
[:graph:]	[^\t\n\r\g\v]	printed characters without spaces
[:print:]	[^\t\n\r\g\v]	printed characters including spaces
[:space:]	[\t\n\r\g\v]	whitespace characters



Using Wildcards / File Globbing (quick reference)

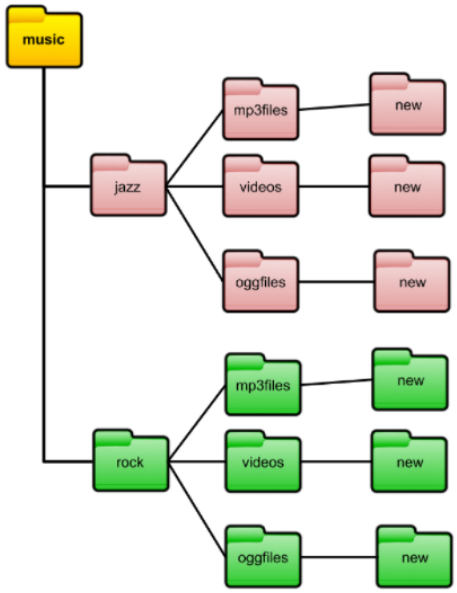
Wildcard	Description
*	Matches zero or more characters in a filename
?	Matches any one character in a filename
[acf]	Matches one of multiple characters in a filename; in this example, a, c, or f
[a-f]	Matches one of a range of characters in a filename; in this example, any character from a through f
[!a-f]	Matches filenames that don't contain a specified range of characters; in this example, filenames that don't contain a through f

Using Brace Expansion

- Brace expansion {} is not a wildcard but another feature of bash that allows you to generate arbitrary strings to use with commands.
- For example,
  - To create a whole directory structure in a single command:
    - `mkdir -p music/{jazz,rock}/{mp3files,vidoes,oggfiles}/new{1..3}`
  - To create a N number of files use:
    - `touch website{1..5}.html`
    - `touch file{A..Z}.txt`
    - `touch file{001..10}.py`
    - `touch file{{a..z},{0..10}}.js`
  - Remove multiple files in a single directory
    - `rm -r {dir1,dir2,dir3,file.txt,file.py}`

Here is a fun example!

Try to replicate this file structure in a single command



Bonus

- **ranger** - visual file manger
  - **Installation:** *sudo apt install ranger*
  - **Usage: ranger**
- **Nnn**
  - **Installation:** *sudo apt install ranger*
  - **Usage: ranger**