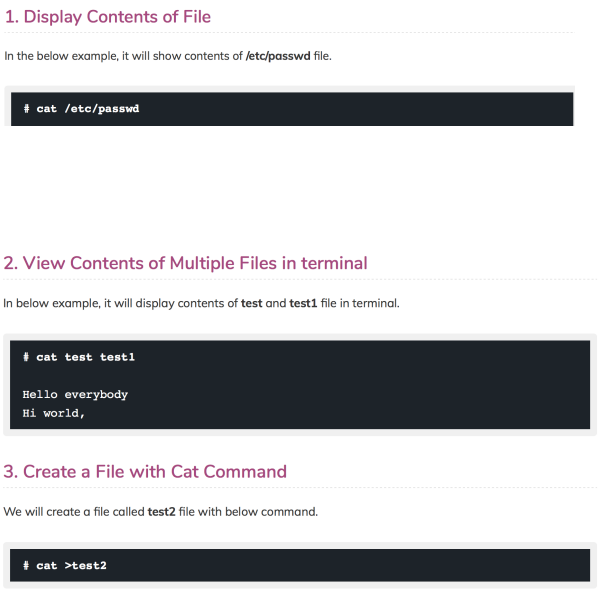
BASH

https://www.thegeekdiary.com/view-files-using-cat-more-tail-head-and-wc-commands/

Cat - cat, > >> 2> < |
The cat (short for "**concatenate**") command is one of the most frequently used command in Linux/Unix like operating systems. **cat** command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.



https://www-uxsup.csx.cam.ac.uk/pub/doc/redhat/AS2.1/rhl-gsg-en-7.2/s1-navigating-usingcat.html

apropos
The apropos command displays a list of all topics in the man pages (i.e., the standard manual that is built into Unix-like operating systems) that are related to the subject of a query.

vi
http://www.atmos.albany.edu/daes/atmclasses/atm350/vi_cheat_sheet.pdf

Sudo
https://www.sudo.ws/man/1.8.28/sudo.man.html
sudo allows a permitted user to execute a command as the superuser or another user, as specified by the security policy. The invoking user's real (not effective) user ID is used to determine the user name with which to query the security policy.

ps
https://www.geeksforgeeks.org/ps-command-in-linux-with-examples/

grep
The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern.
https://www.geeksforgeeks.org/grep-command-in-unixlinux/

Chmod
https://www.howtogeek.com/437958/how-to-use-the-chmod-command-on-linux/

On each line, the first character identifies the type of entry that is being listed. If it is a dash (-) it is a file. If it is the letter d it is a directory.
The next nine characters represent the settings for the three sets of permissions.

> The first three characters show the permissions for the user who owns the file (user permissions).
> The middle three characters show the permissions for members of the file's group (group permissions).
> The last three characters show the permissions for anyone not in the first two categories (other permissions).

Another way to use `chmod` is to provide the permissions you wish to give to the owner, group, and others as a three-digit number. The leftmost digit represents the permissions for the owner. The middle digit represents the permissions for the group members. The rightmost digit represents the permissions for the others.

The digits you can use and what they represent are listed here:

- 0: (000) No permission.
- 1: (001) Execute permission.
- 2: (010) Write permission.
- 3: (011) Write and execute permissions.
- 4: (100) Read permission.
- 5: (101) Read and execute permissions.
- 6: (110) Read and write permissions.
- 7: (111) Read, write, and execute permissions.

The “what” values we can use are:

- –: Minus sign. Removes the permission.
- +: Plus sign. Grants the permission. The permission is added to the existing permissions. If you want to have this permission and only this permission set, use the = option, described below.
- =: Equals sign. Set a permission and remove others.

xargs

How to use xargs

By default `xargs` reads items from standard input as separated by blanks and executes a command once for each argument. In the following example standard input is piped to xargs and the `mkdir` command is run for each argument, creating three folders.

```
echo 'one two three' | xargs mkdir
ls
one two three
```

How to use xargs with find

The most common usage of `xargs` is to use it with the `find` command. This uses `find` to search for files or directories and then uses `xargs` to operate on the results. Typical examples of this are changing the ownership of files or moving files.

`find` and `xargs` can be used together to operate on files that match certain attributes. In the following example files older than two weeks in the temp folder are found and then piped to the xargs command which runs the `rm` command on each file and removes them.

```
find /tmp –mtime +14 | xargs rm
```

grep
Print lines matching a pattern.
A simple example: find all the lines that contains ’test’ in all the files in the current directory
grep test *

Find
Print lines matching a pattern.
A simple example:
find all the lines that contains ’test’ in all the files in the current directory

Control FLow - if statement

where is the linebreak, semi-colon? semicolon in fact starts a new line in one-line command
Spaces right inside brackets are significant
You can write multiple commands in one line for a format choice or style preference.

```
a=6
if [ $a -eq 5 ]; then echo x; else echo 'no'; fi
```