SA

1.1 More Permission

https://en.wikipedia.org/wiki/File_system_permissions

(1)

Delete / Rename / Create files	List dir (ls)	Read file contents	Write file contents	cd dir	cd subdir	List subdir (ls)	Access subdir files
`---` (0) F	F	F	F	F	F	F	F
`-w-` (2) F	F	F	T	F	F	F	F
`r--` (4) F	T	T	F	F	F	F	F
`rw-` (6) T	T	T	T	F	F	F	F
`--x` (1) F	F	F	F	T	T	T	T
`-wx` (3) T	F	F	T	T	T	T	T
`r-x` (5) F	T	T	F	T	T	T	T
`rwx` (7) T	T	T	T	T	T	T	T

If I have r, I can read. If I have w, I can write. If I have x, I can execute files.
If I want to reach subdir, all intermediate folders need to have the x bit on it.

(2)

man chmod

https://superuser.com/questions/303040/how-do-file-permissions-apply-to-symlinks

Permissions only affect the file, not the symbolic link. The permissions of symbolic links are never used and are irrelevant. chmod never changes the permissions of symbolic links; However, for each symbolic link listed on the command line, chmod changes the permissions of the pointed-to file. In contrast, chmod ignores symbolic links encountered during recursive directory traversals.

(3)

https://linuxconfig.org/how-to-use-special-permissions-the-setuid-setgid-and-sticky-bits

setuid: When setuid permission is set on an executable file, a process that runs this file is granted access based on the owner of the file rather than the

user who is running the executable file. This special permission allows a user to access files and directories that are normally only available to the owner.
setgid: The setgid permission is similar to setuid, except that the process's effective group ID is changed to the group owner of the file, and a user is granted access based on permissions granted to that group.
sticky bits: The sticky bit is a permission bit that protects the files within a directory. If the directory has the sticky bit set, a file can be deleted only by the owner of the file, the owner of the directory, or by root. This special permission prevents users from deleting other users' files from public directories.

1.2 Deeper, deeper

(a).
https://stackoverflow.com/questions/16365130/the-difference-between-usr-bin-env-bash-and-usr-bin-bash
#!/usr/bin/env bash, searches PATH for bash.

(b).
https://stackoverflow.com/questions/16365130/the-difference-between-usr-bin-env-bash-and-usr-bin-bash
If we just use #!/usr/bin/bash, the script may fail to run on the different system because location of bash may vary from system to system. #!/usr/bin/env bash helps to search the path of bash. By using env, a script author could make his script easier to run on different systems. So use #!/usr/bin/env bash is better.

1.3 Copy Monster

https://askubuntu.com/questions/806371/whats-the-difference-between-cp-and-rsync
If you want to make a local duplicate file or directory. cp command is simpler and faster way to do this. And if you want to update a backup of some directory which has already exists on your system. rsync is a much faster choice. rsync command compares directories when copying. This is a much more efficient way of backing up data on a regular basis for it only transfer the changes each time. When we want to copy everything such as one-time backup, we can use cp command which copies everything every time.