In Unix-like operating systems, the **chmod** command is used to change the access mode of a file. The name is an abbreviation of change mode.

755 means **full permissions for the owner and read and execute permission for others**

**Chmod + x also works**

1. + means add this permission to the other permissions that the file already has.
2. = means ignore all permissions, set them exactly as I provide.
   * So all of the "read, write, execute, sticky bit, suid and guid" will be ignored and only the ones provided will be set.
3. read = 4, write = 2, execute = 1
   * Here is the binary logic behind it (if you're interested):
   * Symbolic: r-- -w- --x | 421
   * Binary: 100 010 001 | -------
   * Decimal: 4 2 1 | 000 = 0
   * | 001 = 1
   * Symbolic: rwx r-x r-x | 010 = 2
   * Binary: 111 101 101 | 011 = 3
   * Decimal: 7 5 5 | 100 = 4
   * / / / | 101 = 5
   * Owner ---/ / / | 110 = 6
   * Group ------/ / | 111 = 7
   * Others ---------/ | Binary to Octal chart

Using +x you are telling to add (+) the executable bit (x) to the owner, group and others.

* it's equal to ugo+x or u+x,g+x,o+x
* When you don't specify which one of the owner, group or others is your target, in case of x it will considers all of them. And as @Rinzwind pointed out, it's based on umask value, it adds the bit to the ones umask allows. remember if you specify the target like o+r then umask doesn't have any effect anymore.
* It doesn't touch the other mods (permissions).
* You could also use u+x to only add executable bit to the owner.

Using 755 you are specifying:

* 7 --> u=rwx (4+2+1 for owner)
* 5 --> g=rx (4+1 for group)
* 5 --> o=rx (4+1 for others)

So chmod 755 is like: chmod u=rwx,g=rx,o=rx or chmod u=rwx,go=rx.

WHAT IS THIS LINE CALLED?

This first line (**#!/bin/bash or #!/bin/sh**) has a name. It is known as ‘**she-bang**‘(**shabang**). This derives from the concatenation of the tokens *sharp* (#) and *bang* (!). It is also called as **sh-bang**, **hashbang**, **poundbang** or **hash-pling**. In computing, a she-bang is the character sequence consisting of the characters number sign and exclamation mark (#!) at the beginning of a script.

/bin/bash VS /bin/sh

We have often seen variety of she-bang or script header. We often wonder why is that particular script using that particular she-bang, why not some other. On Unix-like Operating systems we have a choice of multiple shells. The shell is responsible not only for the little prompts but also interpreting the commands of the script. Thus the shell plays an important role specially when we implement big and complex logics using conditions, pipes, loops , etc.

**/bin/sh** is an executable representing the system shell and usually implemented as a symbolic link pointing to the executable for whichever shell is the system shell. The system shell is basically the default shell that the script should use. In last couple of years, Debian (and Ubuntu) decided to switch the system shell from bash to dash — a similar shell but lighter and much faster.

Dash is fairly well compatible with bash, being based on the same **POSIX standard**. However, it doesn’t implement the bash-specific extensions. **POSIX standard** is **P**ortable **O**perating **S**ystem *I*nterface, an attempt to standardize UNI**X**-like OSes. Even though Ubuntu’s system shell is pointing to dash, your login shell as a user continues to be bash at this time.

**/bin/bash** is the most common shell used as default shell for user login of the linux system. The shell’s name is an acronym for **Bourne-again shell**. Bash can execute the vast majority of scripts and thus is widely used because it has more features, is well developed and better syntax