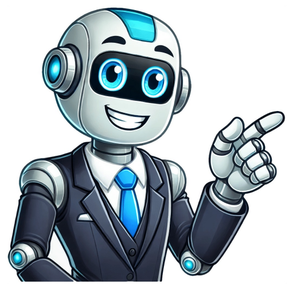
Click to prove you're human



Get a curated assortment of Linux tips, tutorials and memes directly in your inbox. Over 18,000 Linux users enjoy it twice a month. Get a curated assortment of Linux tips, tutorials and memes directly in your inbox. Over 18,000 Linux users enjoy it twice a month. 4.4K As someone who loves tinkering with system settings, one command that I find particularly essential is chmod. This command is a fundamental part of managing file permissions in Linux, and it's a tool every Linux user should master. Understanding chmod can greatly enhance your control over your files and directories, and in this article, my goal is to explain about the effective usage of this command using examples. Introduction to file permissions Linux, like other Unix-based systems, uses a permission system to control who can read, write, or execute a file. Each file and directory has a set of permissions divided into three categories: owner, group, and others. These permissions are represented as a series of letters when you list files using ls -l: -rwxr-xr-- Here's what each part means: -: File type (e.g., - for regular files, d for directories) rwx: Permissions for the file owner r-x: Permissions for the group r--: Permissions for others Basics of chmod The chmod command allows you to change these permissions. The syntax is straightforward: chmod [options] mode file Symbolic mode One way to specify permissions with chmod is using symbolic mode. In symbolic mode, you use letters to represent who you're changing permissions for and what permissions you're changing. Here's a breakdown: u: User (owner) g: Group o: Others a: All (user, group, and others) You can add (+), remove (-), or set (=) permissions. Examples: Adding execute permission for the user: chmod u+x filename If you have a script that you want to make executable by the owner, you'd run this command. The +x part means "add execute permission." Removing write permission for the group: chmod g-w filename This command removes the write permission from the group. Useful if you want to prevent group members from modifying a file. Setting read and write permission for others: chmod o=rw filename This sets the permissions for others to read and write but not execute. Numeric mode Another way to set permissions is using numeric mode. Numeric mode is a bit more efficient once you get the hang of it. Permissions are represented by a three-digit octal number, with each digit representing different permissions. 4: Read (r) 2: Write (w) 1: Execute (x) 0: No permission (–) These numbers are combined to form the permission set. For instance, 7 (4+2+1) means read, write, and execute. Examples: Setting full permissions for the owner and read-only for others: chmod 744 filename This command sets the permissions to rwxr--r--. Setting read and execute permissions for everyone: chmod 755 filename This sets the permissions to rwxr-xr-x. Setting read and write permissions for the owner and the group, and read-only for others: chmod 664 filename This sets the permissions to rw-rw-r--. Recursive changes with chmod If you want to change the permissions of a directory and all its contents, you can use the -R (recursive) option. This is particularly handy when you're setting up a new project and need to apply the same permissions to a bunch of files and subdirectories. Example: chmod -R 755 /path/to/directory Practical examples explaining chmod command usage Let's say you have a directory called my_project with a script inside it called run.sh. You want to ensure that you have full control over the directory and its contents, the group can read and execute files, and others can only read them. List the current permissions: ls -l my_project Output might look like this: drwxr-xr-x 2 user user 4096 Aug 7 12:34 my_project -rw-r--r-- 1 user user 45 Aug 7 12:34 run.sh Make run.sh executable: chmod u+x my_project/run.sh Check the permissions again: ls -l my_project Output: -rwxr--r-- 1 user user 45 Aug 7 12:34 run.sh Set appropriate permissions for the entire directory: chmod -R 755 my_project This command ensures that you can execute files in the directory and subdirectories, while the group and others have read and execute permissions. Takeaways Personally, I find chmod to be an indispensable tool. It provides a granular level of control over file permissions, which is crucial for maintaining system security and functionality. However, one must use it with caution. Incorrect permissions can either lock you out of your files or expose sensitive information to unauthorized users. I dislike when permissions are misconfigured, as it often leads to frustrating troubleshooting sessions. In conclusion, learning chmod is a vital skill for any Linux user. With practice, you'll find it becomes second nature. Hey friend! Learning about permissions in Linux is a key step in your journey as an admin or developer. Let's unravel the mystery around those cryptic rwx codes and dive into what chmod 755 really means. Stick with me and you'll be a permissions pro in no time! Permissions in Linux: A Quick Tour Before we get to chmod 755 specifically, let's understand what file permissions are in Linux. On a multi-user *nix system like Linux, we need a way to control who can access and modify files and directories. That's what permissions provide! There are three basic types of permissions: Read (r) View or copy contents of a file List files in a directory Write (w) Modify or delete a file Create, rename, delete files in a directory Execute (x) Run a executable file/script Change into a directory Now here's the cool part. Permissions are assigned separately to three classes of users: User (u) The owner of the file or directory Group (g) Other users in the same group as the owner Others (o) All other users on the system So each file/directory has 3 sets of rwx permissions that apply to different users. When you run ls -l, it shows the permissions like: -rwxr-xr-x Let's break this down: – File type (d for directory) rwx Owner permissions r-x Group permissions r-x Other users permissions This is the symbolic notation for permissions. But there's also a numeric notation we can use. Each rwx permission is assigned a number: So rwx = 4 + 2 + 1 = 7 Our example would be 755 in numeric notation. Pretty slick! Now that you've got the basics down, let's move on to what exactly chmod 755 means. Understanding chmod 755 Permissions The chmod 755 permission set is commonly used for directories and executable files in Linux. You'll often see it applied to stuff like: /usr/bin /usr/local/bin User home directories Scripts But what does 755 actually represent? Let's break it down. 7 rwx (read, write, execute for owner) 5 r-x (read, execute for group) 5 r-x (read, execute for others) So the owner has full control. But group and others only get read and execute access. This prevents group/others from modifying contents in these locations. The 755 permission set provides a balanced level of access for shared directories and executable programs. Let's look at some common examples of where you'll see this in the wild. User home directories Your home folder like /home/john will be 755. You have full access to your stuff, but others can only view and navigate – not modify. Program directories Locations like /usr/bin contain 755 binary programs that every user can run but not edit. System config files Configs in /etc are 755 so only root can change them but everyone can view. Shared data Shared drives or data folders are often 755 to allow access but prevent tampering. Now that you know where 755 shows up, let's look at how to apply it... Setting the 755 Permission with Chmod The chmod command is used to change permissions of files and directories. To set 755 permissions, you'd run: chmod 755 /home/john/file.txt Let's try a simple example: $ ls -l file.txt -rw-r--r-- 1 john staff 0 Feb 27 15:23 file.txt $ chmod 755 file.txt $ ls -l file.txt -rwxr-xr-x 1 john staff 0 Feb 27 15:23 file.txt Now the owner has rwx while group/others have r-x. Perfect! To recursively apply 755 to all files/dirs under a directory: chmod -R 755 directory Some handy chmod options: -v Verbose output -c List changed files -R Recurse directories Here are a few more examples: # Single file chmod 755 script.sh # Multiple files chmod 755 file1 file2 file3 # Directory recursively chmod -R 755 my_code/ # Verbose output chmod -v 755 file.txt Now that you can set the 755 permission, let's verify it worked... Checking File Permissions in Linux After running chmod, it's good practice to validate the permissions are set correctly. The ls -l command displays the standard permissions for all files/folders. Let's check the permission on a file we changed to 755: $ ls -l script.sh -rwxr-xr-x 1 john staff 875 Feb 28 15:23 script.sh The first 10 characters show the permissions: First character: file/directory flag Next 3 characters: owner permissions Middle 3 characters: group permissions Last 3 characters: other users permissions We can see the owner has rwx while group and others just have r-x. Great success! Here are a some more permission examples: -rw-r--r-- # File: 644 drwxrwsr-x # Directory: 775 -rw------- # Private File: 600 -rwx--x--x # Executable script: 711 Take some time to practice identifying file permissions from the ls output. This is an important Linux skill to master! Now that you know how to set and verify 755 permissions, let's talk about what happens by default when you create new files or directories... Default Permissions: Understanding Umask Whenever a new file or directory is created in Linux, it's given a default permission set: File: 0666 Directory: 0777 This allows full read/write/execute access for user, group and others. But that's usually not what we want for security reasons! Here's where umask comes in... Umask is a value that masks out certain permissions whenever a file is created. The common umask value is 002. What does this do? Value of 002 Masks write for group/others Leaves read and execute So with umask 002, new files get default permissions of 644 instead of 666. And new directories get 755 rather than 777. Much better for security! Each Linux distribution comes with a default umask set in /etc/login.defs Understanding how umask works allows you properly configure your system defaults. Now that you've got permissions and umask down, let's look at some real-world examples... Setting File Permissions for Security Choosing the right permissions is crucial for security and usability. For example: System config files should be 644 to prevent modification by regular users User data might use 700 permissions to restrict access only to the owner Web content is often 755 so the public can read but not write Scripts could be 700 or 755 depending on whether others need access Some key principles to follow: Use the minimum required permissions Start restrictive, open up as needed Remove write access unless absolutely required Here are some common permission guidelines: System config files (/etc) 644 Log files (/var/log) 640 (allow read for admin group) User home directories 700 or 755 Web content 755 Scripts 700 or 755 Setting intentional permissions according to these kinds of best practices will keep your system secure. Now what if you mess up your permissions? Let's talk troubleshooting... Troubleshooting Permission Issues We've all accidentally changed permissions incorrectly and caused issues accessing files. Here are some tips for troubleshooting and correcting permissions problems: Identifying Issues Use ls -l to inspect permissions Try accessing file as various users to pinpoint problem Error messages often indicate permission problem Resetting Permissions Log in as root or file owner to issue chmod Find parent directory with correct permissions and mirror settings Recursively reset permissions on directories using 755 or 775 Restoring Lost Access Boot into recovery mode to gain root access if needed Reset forgotten password to regain access to admin user Take ownership with chown if you have root privileges Preventing Future Problems Set umask to sane default in /etc/login.defs Use ACLs where finer grained control is needed Check for abnormal permissions with find or file commands Script auditing of permissions weekly/monthly Mastering permissions helps avoid many common issues both in development and production environments. Know how to diagnose and recover from problems! Flexing Your Linux Permissions Power We've covered a ton of ground here! Let's recap the key points: Permissions control access to files and directories There are 3 types: read, write, execute And 3 classes: owner, group, others chmod 755 gives full access to owner, read/execute for the rest Use chmod to modify permissions ls -l allows verifying permissions are set Umask influences default permission on new files/dirs Set permissions intentionally for security and access needs Understanding permissions is crucial for both Linux admins and developers. I hope this guide has demystified chmod 755 and how Linux permissions work in general! Let me know if you have any other questions. And good luck on your journey to becoming a Linux permissions pro! Use the octal CHMOD Command: chmod -R 755 folder_name OR use the symbolic CHMOD Command: chmod -R a+rwx,g-w,o-w folder_name The chmod command modifies the permissions of a file or directory on a Linux system. The three numbers after the chmod command represent the permissions assigned to user owner, group owner and others. The numbers 755 assign read-write-execute permissions to the user ower and read-execute permissions to group owner and others. In this article I will explain the basics of the chmod command and you will learn how to use it. It's probably one of the most important Linux commands. We will also go through an example of the Linux command chmod 755. First of all, let's start from the fact that Linux is a multi-user system... ...that's why setting permissions of files and directories is a must know if you work with Linux. There are different types of permissions for users and groups: read permissions write permissions execute permissions And, how can they be set? The chmod command is used in Linux (and Unix-like systems) to set the permissions of files and directories. First of all, here is the generic syntax of the chmod command: chmod The permission part of the command can have different formats. One format is a group of number like the one you see below: chmod 755 So, we have assigned 755 to the permissions part of the command. But, what does it mean? To understand that let's use the ls command in the current directory to look at the details that Linux provides about a file and a directory: [myuser@localhost ~]$ ls -al total 2592 drwxrwxr-x 5 myuser mygroup 4096 Nov 10 16:05 . drwx------ 6 myuser mygroup 225 Mar 10 23:50 .. -rwxrwxr-x 1 myuser mygroup 39 Oct 26 13:13 test_script.sh drwxrwxr-x 6 myuser mygroup 98 Nov 8 18:09 data I have highlighted in bold the file and the directory we are looking at. Which one is the file and which one is the directory? The left hand part of the line starts with a d for a directory and with a dash (–) for a file. The fragment of the lines I have highlighted shows the user owner and group owner, in this specific case the user owner is myuser and the group owner is mygroup. The user owner is the user who has created the file or the directory. The group owner defines the group that has access to the file or directory. The idea of a group is that multiple users can belong to it and hence have the same level of access. For instance, an example of group in a company could be the marketing team whose members should have the same level of access to files and directories. This access is defined by a set of permissions. The same permissions that as mentioned before can be set with the chmod command. So, let's look again at the output of the ls command: [myuser@localhost ~]$ ls -al total 2592 drwxrwxr-x 5 myuser mygroup 4096 Nov 10 16:05 . drwx------ 6 myuser mygroup 225 Mar 10 23:50 .. -rwxrwxr-x 1 myuser mygroup 39 Oct 26 13:13 test_script.sh drwxrwxr-x 6 myuser mygroup 98 Nov 8 18:09 data This time we focus on the first part of the output, the permissions assigned to the file and the directory. Here is how you can break down the permissions: Character 1: as we said before it indicates if this is a file (–) or a directory (d). 2-4: permissions for the user owner (in this case rwx) 5-7: permissions for the group owner (in this case rwx) 8-10: permissions for others (any user who is not the user owner or is not part of the group owner – in this case r-x) It's time to cover the basic values for permissions (characters between 2-10): r: read permission w: write permission x: execute permission The dash (–) shows permissions that are not set (e.g. r-x indicates that the read and execute permissions are set but the write permission is not set). So, if we go back to our file: -rwxrwxr-x 1 myuser mygroup 39 Oct 26 13:13 test_script.sh The user owner has read-write-execute permissions, the group owner has read-write-execute permissions and others have read-execute permissions (the write permission is not set). And the same applies to the directory: drwxrwxr-x 6 myuser mygroup 98 Nov 8 18:09 data So, always remember to read permissions as groups of three letters that always follow the same logical order: r (read), w (write) and x (execute). So, how is this related to the initial question? What does the command chmod 755 mean? The permissions we have seen expressed using the letters r, w, x can also be expressed with an octal representation (using the numbers 0 through 7). And that's exactly what 755 is, the octal representation of a set of permissions for user owner, group owner and others. The octal number comes from the sum of the following numbers: 4 for the read permission 2 for the write permission 1 for the execute permission Where permissions for user owner, group owner and others are considered separately. In other words the sum of the permissions for the user owner will produce one octal number (from 0 to 7) and the same applies for the permissions for the group owner and for others. That's why we end up with 3 octal numbers (there could be a fourth number before them but it's out of scope for this tutorial). Going back to 755, here is what it means: 7: 4 + 2 + 1 = read + write + execute permissions 5: 4 + 1 = read + execute permissions 5: 4 + 1 = read + execute permissions Does it make sense? How would you translate the permissions 644 instead? Also remember to stay away from the permissions 777 that gives full access to everyone. Using the chmod 777 command on files or directories is not a recommended practice for obvious security reasons. Now that we know what 755 means, let's have a look at the effect of this set of permissions on a file and on a directory. We want to understand what read, write and execute permissions do from a user perspective when applied to files or directories. Read permission: file: read the content of the file directory: list the content of the directory Write permission: file: modify the content of the file directory: add, rename and delete files in the directory (assuming that the execute permission is set on the directory) Execute permission: file: define an executable file (e.g. a Bash script) directory: access the directory So, you now know what the command chmod 755 means applied to files and directories. To get full understanding of it try to create a file and a directory and experiment with different permissions. See any errors returned by the Linux shell when you execute an operation not allowed on a file or directory based on the permissions you assigned to it. In all our examples so far we have only used the chmod 755 command followed by the set of permissions and by the file or directory we apply those permissions too. Sometimes you might want to assign the same permissions to a directory and to all the files and subdirectories under it. How can you do it? The chmod command provides a flag that allows to apply the change in permissions recursively to anything under the directory you apply the command to. Let's see an example... ...I have created a directory called test_dir: drwxr-xr-x 5 myuser mygroup 160 Jul 23 23:56 test_dir As you can see the permissions for the directory are rwx (7), r-x (5) and r-x (5). Then inside this directory I have created two subdirectories (test_dir1 and test_dir2) and one file (testfile). [myuser@localhost ~]$ ls -al test_dir/ total 0 drwxr-xr-x 5 myuser mygroup 160 Jul 23 23:56 . drwxr-xr-x 3 myuser mygroup 96 Jul 23 23:56 .. drwxr-xr-x 2 myuser mygroup 64 Jul 23 23:56 test_dir1 drwxr-xr-x 2 myuser mygroup 64 Jul 23 23:56 test_dir2 -rw-r--r-- 1 myuser mygroup 0 Jul 23 23:56 testfile If I want to make sure everything inside the test_dir directory has permissions set to 755, I can use the chmod command with the -R flag: chmod -R 755 test_dir And here is the content of the directory after running the command: [myuser@localhost ~]$ ls -al test_dir/ total 0 drwxr-xr-x 5 myuser mygroup 160 Jul 23 23:56 . drwxr-xr-x 3 myuser mygroup 96 Jul 23 23:56 .. drwxr-xr-x 2 myuser mygroup 64 Jul 23 23:56 test_dir1 drwxr-xr-x 2 myuser mygroup 64 Jul 23 23:56 test_dir2 -rwxr-xr-x 1 myuser mygroup 0 Jul 23 23:56 testfile As you can see the permission of the file testfile have been updated from rw-r–r– (644) to rwxr-xr-x (755). Makes sense? We have covered quite a lot of concepts in this article. Now you should know: The purpose of the chmod command. What is the number 755 applied to chmod. Which command gives user and group owner certain permissions. How to map read, write and execute permissions with their octal representation. The meaning of chmod 755 applied to files and directories. Do you have any questions? Claudio Sabato is an IT expert with over 15 years of professional experience in Python programming, Linux Systems Administration, Bash programming, and IT Systems Design. He is a professional certified by the Linux Professional Institute. With a Master's degree in Computer Science, he has a strong foundation in Software Engineering and a passion for helping others become Software Engineers. Just select the permissions that you want for your files and hit the Calculate button. The permissions are displayed under the calculate button. Linux File Permission is in absolute mode and in symbolic mode. Once you have the file permissions in absolute or symbolic mode, you can use the chmod command to change the file permission. Refer to these chmod command examples if you are not familiar with this command.Chmod ExplanationSome details about the user, group and other you may need to know to clear your basics.User/OwnerUser is the owner of the file. When you create a file, you become the owner of the file. The ownership can be changed.GroupEvery user is part of a certain group(s). A group consists of several users and this is one way to manage users in a multi-user environment.OtherOther can be considered as a super group with all the users on the system. Basically, anyone with access to the system belongs to this group.I hope this nifty chmod calculator helped you to calculate the Linux file permissions easily.Common chmod commands and their meaningHere are some command chmod commands with their explanation:chmod 777This means that owner, group and everyone has all the rights, i.e. to read, write and execute. This is a dangerous permission to have on any file and you should avoid using it.chmod 755The owner can read, write and execute. Group members and everyone else can read and execute but cannot modify (write) the file.chmod +xWith this command, you are adding execute permission for the owner, group and everyone else. This is equivalent to chmod a+x.chmod 600With this, you are giving read and write permission to the owner user. Group members and others cannot read, write or execute. Even the owner cannot execute the file with this permission set.chmod 700You are giving read, write and execute permission to the owner user but the groups members and others have no permissions at all. They cannot read, write or execute.chmod 400The file can only be read by the owner. No one can write or execute it.chmod 775The user and groups can read, write and execute the file. Others can read and execute but cannot write.chmod 644The owner can read and write but cannot execute it. Group members and others can read the file but cannot write or execute it. Get a curated assortment of Linux tips, tutorials and memes directly in your inbox. Over 18,000 Linux users enjoy it twice a month. Control who can access files, search directories, and run scripts using the Linux's chmod command. This command modifies Linux file permissions, which look complicated at first glance but are actually pretty simple once you know how they work. chmod Modifies File Permissions In Linux, who can do what to a file or directory is controlled through sets of permissions. There are three sets of permissions. One set for the owner of the file, another set for the members of the file's group, and a final set for everyone else. The permissions control the actions that can be performed on the file or directory. They either permit, or prevent, a file from being read, modified or, if it is a script or program, executed. For a directory, the permissions govern who can cd into the directory and who can create, or modify files within the directory. You use the chmod command to set each of these permissions. To see what permissions have been set on a file or directory, we can use ls . We can use the -l (long format) option to have ls list the file permissions for files and directories. ls -l 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). There are three characters in each set of permissions. The characters are indicators for the presence or absence of one of the permissions. They are either a dash (-) or a letter. If the character is a dash, it means that permission is not granted. If the character is an r, w, or an x, that permission has been granted. The letters represent: r: Read permissions. The file can be opened, and its content viewed. w: Write permissions. The file can be edited, modified, and deleted. x: Execute permissions. If the file is a script or a program, it can be run (executed). For example: --- means no permissions have been granted at all. rwx means full permissions have been granted. The read, write, and execute indicators are all present. In our screenshot, the first line starts with a d. This line refers to a directory called "archive." The owner of the directory is "dave," and the name of the group that the directory belongs to is also called "dave." The next three characters are the user permissions for this directory. These show that the owner has full permissions. The r, w, and x characters are all present. This means the user dave has read, write and execute permissions for that directory. The second set of three characters are the group permissions, these are r-x. These show that the members of the dave group have read and execute permissions for this directory. That means they can list the files and their contents in the directory, and they can cd (execute) into that directory. They do not have write permissions, so they cannot create, edit, or delete files. The final set of three characters are also r-x. These permissions apply to people who are not governed by the first two sets of permissions. These people (called"others") have read and execute permissions on this directory. So, to summarise, group members and others have read and execute permissions. The owner, a user called dave, also has write permissions. For all of the other files (apart from the mh.sh script file) dave and members of the dave group have read and write properties on the files, and the others have read permissions only. For the special case of the mh.sh script file, the owner dave and the group members have read, write, and execute permissions, and the others have read and execute permissions only. To use chmod to set permissions, we need to tell it: Who: Who we are setting permissions for. What: What change are we making? Are we adding or removing the permission? Which: Which of the permissions are we setting? We use indicators to represent these values, and form short "permissions statements" such as u+x, where "u" means " user" (who), "+" means add (what), and "x" means the execute permission (which). The "who" values we can use are: u: User, meaning the owner of the file. g: Group, meaning members of the group the file belongs to. o: Others, meaning people not governed by the u and g permissions. a: All, meaning all of the above. If none of these are used, chmod behaves as if "a" had been used. 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. The "which " values we can use are: r: The read permission. w: The write permission. x: The execute permission. Let's say we have a file where everyone has full permissions on it. ls -l new_file.txt We want the user dave to have read and write permissions and the group and other users to have read permissions only. We can do using the following command: chmod u=rw,og=r new_file.txt Using the "=" operator means we wipe out any existing permissions and then set the ones specified. let's check the new permission on this file: ls -l new_file.txt The existing permissions have been removed, and the new permissions have been set, as we expected. How about adding a permission without removing the existing permissions settings? We can do that easily too. Let's say we have a script file that we have finished editing. We need to make it executable for all users. Its current permissions look like this: ls -l new_script.sh We can add the execute permission for everyone with the following command: chmod a+x new_script.sh If we take a look at the permissions, we'll see that the execute permission is now granted to everyone, and the existing permissions are still in place. ls -l new_script.sh We could have achieved the same thing without the "a" in the "a+x" statement. The following command would have worked just as well. chmod +x new_script.sh We can apply permissions to multiple files all at once. These are the files in the current directory: ls -l Let's say we want to remove the read permissions for the "other" users from files that have a ".page" extension. We can do this with the following command: chmod o-r *.page Let's check what effect that has had: ls -l As we can see, the read permission has been removed from the ".page" files for the "other" category of users. No other files have been affected. If we had wanted to include files in subdirectories, we could have used the -R (recursive) option. chmod -R o-r *.page 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. Each of the three permissions is represented by one of the bits in the binary equivalent of the decimal number. So 5, which is 101 in binary, means read and execute. 2, which is 010 in binary, would mean the write permission. Using this method, you set the permissions that you wish to have; you do not add these permissions to the existing permissions. So if read and write permissions were already in place you would have to use 7 (111) to add execute permissions. Using 1 (001) would remove the read and write permissions and add the execute permission. Let's add the read permission back on the ".page" files for the others category of users. We must set the user and group permissions as well, so we need to set them to what they are already. These users already have read and write permissions, which is 6 (110). We want the "others" to have read and permissions, so they need to be set to 4 (100). The following command will accomplish this: chmod 664 *.page This sets the permissions we require for the user, group members, and others to what we require. The users and group members have their permissions reset to what they already were, and the others have the read permission restored. ls -l If you read the man page for chmod you'll see there are some advanced options related to the SETUID and SETGID bits, and to the restricted deletion or "sticky" bit. For 99% of the cases you'll need chmod for, the options described here will have you covered. Get a curated assortment of Linux tips, tutorials and memes directly in your inbox. Over 18,000 Linux users enjoy it twice a month. Series of free software licenses "GPL" redirects here. For other uses, see GPL (disambiguation). This article has multiple issues. Please help improve it or discuss these issues on the talk page. (Learn how and when to remove these messages) This article may require copy editing for grammar, style, cohesion, tone, or spelling. You can assist by editing it. (November 2023) (Learn how and when to remove this message) This article may be too long to read and navigate comfortably. Consider splitting content into sub-articles, condensing it, or adding subheadings. Please discuss this issue on the article's talk page. (December 2024) (Learn how and when to remove this message) GNU General Public LicenseAuthorRichard StallmanLatest version3PublisherFree Software FoundationPublished25 February 1989; 36 years ago (1989-02-25)SPDX identifierGPL-3.0-or-laterGPL-3.0-onlyGPL-2.0-or-laterGPL-2.0-onlyGPL-1.0-or-laterGPL-1.0-onlyDebian FSG compatibleYes[1]FSF approvedYes[2]OSI approvedYes (applies to GPLv3-only and GPLv2-only)[3]CopyleftYes[2][4][5]Linking from code with a different licenceSoftware licensed under GPL compatible licenses only, with the exception of the LGPL which allows all programs.[6]Websitewww.gnu.org/licenses/gpl.html The GNU General Public Licenses (GNU GPL, or simply GPL) are a series of widely used free software licenses, or copyleft licenses, that guarantee end users the freedoms to run, study, share, or modify the software.[7] The GPL was the first copyleft license available for general use. It was originally written by Richard Stallman, the founder of the Free Software Foundation (FSF), for the GNU Project. The license grants the recipients of a computer program the rights of the Free Software Definition.[8] The licenses in the GPL series are all copyleft licenses, which means that any derivative work must be distributed under the same or equivalent license terms. It is more restrictive than the Lesser General Public License, and even further distinct from the more widely used permissive software licenses such as BSD, MIT, and Apache. Historically, the GPL license family has been one of the most popular software licenses in the free and open-source software (FOSS) domain.[7][9][10][11][12] Prominent free software programs licensed under the GPL include the Linux kernel and the GNU Compiler Collection (GCC). David A. Wheeler argues that the copyleft provided by the GPL was crucial to the success of Linux-based systems, giving the programmers who contributed to the kernel assurance that their work would benefit the whole world and remain free, rather than being exploited by software companies that would not have to give anything back to the community.[13] In 2007, the third version of the license (GPLv3) was released to address some perceived problems with the second version (GPLv2) which were discovered during the latter's long-time usage. To keep the license current, the GPL license includes an optional "any later version" clause, allowing users to choose between the original terms or the terms in new versions as updated by the FSF. Software projects licensed with the optional "or later" clause include the GNU Project, while projects like the Linux kernel are licensed under GPLv2 only. The "or any later version" clause is sometimes known as a "lifeboat clause" since it allows combinations between different versions of GPL-licensed software to maintain compatibility. Usage of the license has steadily declined since the 2010s, particularly due to these complexities, but also a perception it holds back the modern open source landscape from growth and commercialization.[14][15] The original GPL was written by Richard Stallman in 1989, for use with programs released as part of the GNU Project. It was based on a unification of similar licenses used for early versions of GNU Emacs,[16] the GNU Debugger, and the GNU C Compiler.[17] These licenses contained similar provisions to the modern GPL, but were specific to each program, rendering them incompatible, despite being the same license.[18] Stallman's goal was to produce one license that could be used for any project, thus making it possible for many projects to share code. The second version of the license, GPLv2, was released in 1991. Over the following 15 years, members of the free software community became concerned over problems in the GPLv2 license that could let someone exploit GPL-licensed software in ways contrary to the license's intent.[19] These problems included tivoization (the inclusion of GPL-licensed software in hardware that refuses to run modified versions of its software), compatibility issues similar to those of the AGPL (v1), and patent deals between Microsoft and distributors of free and open-source software, which some viewed as an attempt to use patents as a weapon against the free software community. Version 3 was developed as an attempt to address these concerns and was officially released on 29 June 2007.[20] GNU General Public License, version 1Published25 February 1989Websitewww.gnu.org/licenses/old-licenses/gpl-1.0.htmlDeprecatedyes Version 1 of the GNU GPL,[21] released on 25 February 1989,[22] was written to protect against the two main methods by which software distributors restricted the freedoms that define free software. The first problem is that distributors might publish only binary files that are executable, but not readable or modifiable by humans. To prevent this, the GPLv1 states that copying and distributing copies of any portion of the program must also make the human-readable source code available under the same licensing terms.[a] The second problem is that distributors might add restrictions, either to the license or by combining the software with other software that had other restrictions on distribution. The union of two sets of restrictions would apply to the combined work, thus adding unacceptable constrictions. To prevent this, the GPLv1 states that modified versions, as a whole, had to be distributed under the terms of GPLv1.[b] Therefore, software distributed under the terms of the GPLv1 could be combined with software under more permissive terms, as this would not change the terms under which the whole could be distributed. However, software distributed under GPLv1 could not be combined with software distributed under a more restrictive license, as this would conflict with the requirement that the whole be distributable under the terms of GPLv1. GNU General Public License, version 2PublishedJune 1991Websitewww.gnu.org/licenses/old-licenses/gpl-2.0.html According to Richard Stallman, the major change in version 2 of the GPL was the "Liberty or Death" clause, as he calls it[18] – Section 7. The section says that licensees may distribute a GPL-covered work only if they can satisfy all of the license's obligations, despite any other legal obligations they might have. In other words, the obligations of the license may not be severed due to conflicting obligations. This provision is intended to discourage any party from using a patent infringement claim or other litigation to impair users' freedom under the license.[18] By 1990, it was becoming apparent that a less restrictive license would be strategically useful for the C library and for software libraries that did the same job of existing proprietary ones.[23] When the GPLv2 was released in June 1991, a second license – the GNU Library General Public License – was introduced at the same time and numbered with version 2 to show that both were complementary.[24] The version numbers diverged in 1999 when version 2.1 of the LGPL was released, which renamed it the GNU Lesser General Public License to reflect its place in the philosophy. The GPLv2 was also modified to refer to the new name of the LGPL, but its version number remained the same, resulting in the original GPLv2 not being recognized by the Software Package Data Exchange (SPDX).[25][failed verification] The license includes instructions to specify "version 2 of the License, or (at your option) any later version" to allow the flexible optional use of either version 2 or 3, but some developers change this to specify "version 2" only. GNU General Public License, version 3Published29 June 2007Websitewww.gnu.org/licenses/gpl-3.0.html In late 2005, the Free Software Foundation (FSF) announced work on version 3 of the GPL. On 16 January 2006, the first "discussion draft" of GPLv3 was published, and the public consultation began. The official GPLv3 was released by the FSF on 29 June 2007. GPLv3 was written by Richard Stallman, with legal counsel from Eben Moglen and Richard Fontana from the Software Freedom Law Center.[26][27] According to Stallman, the most important changes were in relation to software patents, free software license compatibility, the definition of "source code", and hardware restrictions on software modifications, such as tivoization.[26][28] Other changes related to internationalization, how license violations are handled, and how additional permissions could be granted by the copyright holder. The concept of "software propagation," was explicitly defined as a term for the copying and duplication of software. The public consultation process was coordinated by the Free Software Foundation with assistance from Software Freedom Law Center, Free Software Foundation Europe,[29] and other free software groups. Comments were collected from the public via the gplv3.fsf.org web portal,[30] using purpose-written software called stet. By the end of the comment period, a total of 2,636 comments had been submitted.[31] The third draft was released on 28 March 2007.[32] This draft included language intended to prevent patent-related agreements such as the controversial Microsoft-Novell patent agreement, and restricted the anti-tivoization clauses to a legal definition of a "user" and a "consumer product". It also explicitly removed the section on "Geographical Limitations", the probable removal of this section having been announced at the launch of the public consultation. Richard Stallman at the launch of the first draft of the GNU GPLv3 at MIT, Cambridge, Massachusetts, United States. To his right is Columbia Law Professor Eben Moglen, chairman of the Software Freedom Law Center. The fourth and final discussion draft[33] was released on 31 May 2007. It introduced Apache License version 2.0 compatibility (prior versions are incompatible), clarified the role of outside contractors, and made an exception to avoid the perceived problems of a Microsoft–Novell style agreement, saying in Section 11 paragraph 6 that: You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license ... This aimed to make such future deals ineffective. The license was also meant to cause Microsoft to extend the patent licenses it granted to Novell customers for the use of GPLv3 software to all users of that GPLv3 software; this was possible only if Microsoft was legally a "conveyor" of the GPLv3 software.[34] Early drafts of GPLv3 also let licensors add an AGPL-like requirement that would have plugged a loophole in the GPL regarding application service providers.[35][36] The freedom to run, study, and share the source code and guarantee copyleft protections is somewhat ambiguous in the context of web services. However, there were concerns expressed about the administrative costs of checking code for the additional requirements in the GPLv3 drafts, and it was decided to keep the GPL and the AGPL license separated.[37] Others, notably high-profile Linux kernel developers such as Linus Torvalds, Greg Kroah-Hartman, and Andrew Morton, commented to the mass media and made public statements about their objections to parts of the GPLv3 drafts.[38] The kernel developers disapproved of GPLv3 draft clauses regarding DRM/tivoization, patents, and "additional restrictions", and warned of a Balkanisation of the "Open Source Universe".[38][39] Linus Torvalds, who decided not to adopt the GPLv3 for the Linux kernel,[40] reiterated his criticism several years later.[41][42] GPLv3 improved compatibility with several free software licenses such as the Apache License, version 2.0, and the GNU Affero General Public License, which GPLv2 could not be combined with.[43] However, GPLv3 software could only be combined and share code with GPLv2 software if the GPLv2 license used had the optional "or later" clause and the software was upgraded to GPLv3. While the "GPLv2 or any later version" clause is considered by FSF as the most common form of licensing GPLv2 software,[44] Toybox developer Rob Landley described it as a lifeboat clause.[c] Software projects licensed with the optional "or later" clause include Joomla[47] and the GNU Project,[48] while a prominent example without the clause is the Linux kernel.[40][49] The final version of the license text was published on 29 June 2007.[50] The terms and conditions of the GPL must be made available to anybody receiving a copy of a work that has a GPL license applied to it ("the licensee"). Any licensee who adheres to the terms and conditions is given permission to modify the work, as well as to copy and redistribute the work or any derivative version. The licensee is allowed to charge a fee for this service or do this free of charge. This latter point distinguishes the GPL from software licenses that prohibit commercial redistribution. The FSF argues that free software should not place restrictions on commercial use,[51] and the GPL explicitly states that GPL works may be sold at any price. The GPL additionally states that a distributor may not impose "further restrictions on the rights granted by the GPL". This forbids activities such as distributing the software under a non-disclosure agreement or contract. The fourth section of the GPLv2 and the seventh section the GPLv3 require that programs distributed as pre-compiled binaries be accompanied by a copy of the source code, a written offer to distribute the source code via the same mechanism as the pre-compiled binary, or the written offer to obtain the source code that the user got when they received the pre-compiled binary under the GPL. The second section of the GPLv2 and the fifth section of the GPLv3 also require distributing the license along with the program. The GPLv3 allows making the source code available in additional ways in fulfillment of the seventh section. These include downloading source code from an adjacent network server or by peer-to-peer transmission, provided that is how the compiled code was available and there are "clear directions" on where to find the source code. The FSF does not hold the copyright for a work released under the GPL unless an author explicitly assigns copyrights to the FSF (which seldom happens except for programs that are part of the GNU Project). Only the individual copyright holders have the authority to sue when a license violation is suspected. Printed GPL statements for consumer entertainment devices which incorporate GPL components Software under the GPL may be run for all purposes, including commercial purposes and even as a tool for creating proprietary software, such as when using GPL-licensed compilers.[52] Users or companies who distribute GPL-licensed works (e.g. software), may charge a fee for copies or give them free of charge. This distinguishes the GPL from shareware software licenses that allow copying for personal use but prohibit commercial distribution or proprietary licenses where copying is prohibited by copyright law. The FSF argues that freedom-respecting free software should also not restrict commercial use and distribution (including redistribution):[51][53] In purely private (or internal) use—with no sales and no distribution—the software code may be modified and parts reused without requiring the source code to be released. For sales or distribution, the entire source code needs to be made available to end users, including any code changes and additions—in that case, copyleft is applied to ensure that end users retain the freedoms defined above. However, software running as an application program under a GPL-licensed operating system such as Linux is not required to be licensed under GPL or to be distributed with source-code availability—the licensing depends only on the used libraries and software components and not on the underlying platform.[54] For example, if a program consists only of original source code, or is combined with source code from other software components,[d] then the custom software components need not be licensed under GPL and need not make their source code available; even if the underlying operating system used is licensed under the GPL, applications running on it are not considered derivative works.[54] Only if GPL-licensed parts are used in a program (and the program is distributed), then all other source code of the program needs to be made available under the same license terms. The GNU Lesser General Public License (LGPL) was created to have a weaker copyleft than the GPL, in that it does not require custom-developed source code (distinct from the LGPL-licensed parts) to be made available under the same license terms. The fifth section of the GPLv3 states that no GPL-licensed code shall be considered an effective "technical protection measure" as defined by Article 11 of the WIPO Copyright Treaty, and that those who convey the work waive all legal power to prohibit circumvention of the technical protection measure "to the extent such circumvention is effected by exercising rights under this License with respect to the covered work". This means that users cannot be held liable for circumventing DRM implemented using GPLv3-licensed code under laws such as the US Digital Millennium Copyright Act (DMCA).[55] Main article: Copyleft The distribution rights granted by the GPL for modified versions of the work are not unconditional. When someone distributes a GPL-licensed work plus their own modifications, the requirements for distributing the whole work cannot be any greater than the requirements that are in the GPL. This requirement is known as copyleft. It earns its legal power from the use of copyright on software programs. Because a GPL work is copyrighted, a licensee has no right to redistribute it, not even in modified form (barring fair use), except under the terms of the license. One is only required to adhere to the terms of the GPL if one wishes to exercise rights normally restricted by copyright law, such as redistribution. Conversely, if one distributes copies of the work without abiding by the terms of the GPL (for instance, by keeping the source code secret), they can be sued by the original author under copyright law. Copyright law has historically been used to prevent distribution of work by parties not authorized by the creator. Copyleft uses the same copyright laws to accomplish a very different goal. It grants rights to distribution to all parties insofar as they provide the same rights to subsequent ones, and they to the next, etc. In this way, the GPL and other copyleft licenses attempt to enforce libre access to the work and all derivatives.[56] Many distributors of GPL-licensed programs bundle the source code with the executables. An alternative method of satisfying the copyleft is to provide a written offer to provide the source code on a physical medium (such as a CD) upon request. In practice, many GPL-licensed programs are distributed over the Internet, and the source code is made available over FTP or HTTP. For Internet distribution, this complies with the license. Copyleft applies only when a person seeks to redistribute the program. Developers may make private modified versions with no obligation to divulge the modifications, as long as they do not distribute the modified software to anyone else. Copyleft applies only to the software, and not to its output (unless that output is itself a derivative work of the program).[e] For example, a public web portal running a modified derivative of a GPL-licensed content management system is not required to distribute its changes to the underlying software, because the modified web portal is not being redistributed but rather hosted, and also because the web portal output is also not a derivative work of the GPL-licensed content management system. There has been debate on whether it is a violation of the GPLv1 to release the source code in obfuscated form, such as in cases in which the author is less willing to make the source code available. The consensus was that while unethical, it was not considered a violation. The issue was clarified when the license was altered with v2 to require that the "preferred" version of the source code be made available.[58] The GPL was designed as a license, rather than a contract.[59] In some common law jurisdictions, the legal distinction between a license and a contract is an important one: contracts are enforceable by contract law, whereas licenses are enforced under copyright law. However, this distinction is not useful in the many jurisdictions where there are no differences between contracts and licenses, such as civil law systems.[60] Those who do not accept the GPL's terms and conditions do not have permission, under copyright law, to copy or distribute GPL-licensed software or derivative works. However, if they do not redistribute the GPL-licensed program, they may still use the software within their organization however they like, and works (including programs) constructed by the use of the program are not required to be covered by this license. Software developer Allison Randal argued that the GPLv3 as a license is unnecessarily confusing for lay readers, and could be simplified while retaining the same conditions and legal force.[61] In April 2017, a US federal court ruled that an open-source license is an enforceable contract.[62] In October 2021 Software Freedom Conservancy sued Vizio over breach of contract as an end user to request source code for Vizio's TVs. A federal judge has ruled in the interim that the GPL is an enforceable contract by end users as well as a license for copyright holders.[63] The text of the GPL is itself copyrighted, and the copyright is held by the Free Software Foundation. The FSF permits people to create new licenses based on the GPL, as long as the derived licenses do not use the GPL preamble without permission. This is discouraged, however, since such a license might be incompatible with the GPL[64] and causes a perceived license proliferation. Other licenses created by the GNU Project include the GNU Lesser General Public License, GNU Free Documentation License, and GNU Affero General Public License. The text of the GPL is not itself under the GPL. The license's copyright disallows modification of the license. Copying and distributing the license is allowed since the GPL requires recipients to get "a copy of this License along with the Program."[65] According to the GPL FAQ, anyone can create a new license using a modified version of the GPL as long as they use a different name for the license, do not mention "GNU," and remove the preamble. However, the preamble can be used in a modified license if permission to use it is obtained from the Free Software Foundation (FSF).[66] This section is written like a personal reflection, personal essay, or argumentative essay that states a Wikipedia editor's personal feelings or presents an original argument about a topic. Please help improve it by rewriting it in an encyclopedic style. (November 2023) (Learn how and when to remove this message) According to the FSF, "The GPL does not require you to release your modified version or any part of it. You are free to make modifications and use them privately, without ever releasing them."[67] However, if one releases a GPL-licensed entity to the public, there is an issue regarding linking: namely, whether a proprietary program that uses a GPL library is in violation of the GPL. This key dispute is whether non-GPL software can legally statically link or dynamically link to GPL libraries. Different opinions exist on this issue. The GPL is clear in requiring that all derivative works of code under the GPL must themselves be under the GPL. Ambiguity arises with regard to using GPL libraries and bundling GPL software into a larger package (perhaps mixed into a binary via static linking). This is ultimately a question not of the GPL per se, but of how copyright law defines derivative works. The following points of view exist: The Free Software Foundation (which holds the copyright of several notable GPL-licensed software products and of the license text itself) asserts that an executable that uses a dynamically linked library is indeed a derivative work. This does not, however, apply to separate programs communicating with one another.[68] The Free Software Foundation also created the LGPL, which is nearly identical to the GPL, but with additional permissions to allow linking for the purposes of "using the library". Richard Stallman and the FSF specifically encourage library writers to license under the GPL so that proprietary programs cannot use the libraries, in an effort to protect the free software world by giving it more tools than the proprietary world.[69] Some people believe that while static linking produces derivative works, it is not clear whether an executable that dynamically links to a GPL code should be considered a derivative work (see weak copyleft). Linux author Linus Torvalds agrees that dynamic linking can create derived works but disagrees over the circumstances.[70] A Novell lawyer has written that dynamic linking not being derivative "makes sense" but is not "clear-cut", and that evidence for good-intentioned dynamic linking can be seen by the existence of proprietary Linux kernel drivers.[71] In Galoob v. Nintendo, the United States Ninth Circuit Court of Appeals defined a derivative work as having "'form' or permanence" and noted that "the infringing work must incorporate a portion of the copyrighted work in some form",[72] but there have been no clear court decisions to resolve this particular conflict. According to an article in the Linux Journal, Lawrence Rosen (a one-time Open Source Initiative general counsel) argues that the method of linking is mostly irrelevant to the question about whether a piece of software is a derivative work; more important is the question about whether the software was intended to interface with client software or libraries.[73] He states, "The primary indication of whether a new program is a derivative work is whether the source code of the original program was used [in a copy-paste sense], modified, translated or otherwise changed in any way to create the new program. If not, then I would argue that it is not a derivative work,"[73] and lists numerous other points regarding intent, bundling, and linkage mechanism. He further argues on his firm's website[74] that such "market-based" factors are more important than the linking technique. There is also the specific issue of whether a plugin or module (such as the NVidia or ATI graphics card kernel modules) must also be GPL if it could reasonably be considered its own work. This point of view suggests that reasonably separate plugins, or plugins for software designed to use plugins, could be licensed under an arbitrary license if the work is GPLv2. Of particular interest is the GPLv2 paragraph: You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions: ... b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. ... These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. The GPLv3 has a different clause: You may convey a work based on the Program or the modifications to produce it from the Program, in the form of source code under the terms of Section 4, provided that you also meet all of these conditions: ... c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable Section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. ... A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. As a case study, some supposedly proprietary plugins and themes/skins for GPLv2 CMS software such as Drupal and WordPress have come under fire, with both sides of the argument taken.[75] The FSF differentiates on how the plugin is being invoked. If the plugin is invoked through dynamic linkage and it performs function calls to the GPL program then it is most likely a derivative work.[76] The mere act of communicating with other programs does not, by itself, require all software to be GPL; nor does distributing GPL software with non-GPL software. However, minor conditions must be followed that ensure the rights of GPL software are not restricted. The following is a quote from the gnu.org GPL FAQ, which describes to what extent software is allowed to communicate with and be bundled with GPL programs:[77] What is the difference between an "aggregate" and other kinds of "modified versions"? An "aggregate" consists of a number of separate programs, distributed together on the same CD-ROM or other media. The GPL permits you to create and distribute an aggregate, even when the licenses of the other software are non-free or GPL-incompatible. The only condition is that you cannot release the aggregate under a license that prohibits users from exercising rights that each program's individual license would grant them. Where's the line between two separate programs, and one program with two parts? This is a legal question, which ultimately judges will decide. We believe that a proper criterion depends both on the mechanism of communication (exec, pipes, rpc, function calls within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged). If the modules are included in the same executable file, they are definitely combined in one program. If modules are designed to run linked together in a shared address space, that almost surely means combining them into one program. By contrast, pipes, sockets, and command-line arguments are communication mechanisms normally used between two separate programs. So when they are used for communication, the modules normally are separate programs. But if the semantics of the communication are intimate enough, exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program. The FSF thus draws the line between "library" and "other program" via 1) "complexity" and "intimacy" of information exchange and 2) mechanism (rather than semantics), but resigns that the question is not clear-cut and that in complex situations, case law will decide. See also: SCO-Linux controversies and SCO v. IBM The first known violation of the GPL was in 1989, when NeXT extended the GCC compiler to support Objective-C, but did not publicly release the changes.[78] After an inquiry they created a public patch. There was no lawsuit filed for this violation.[79] In 2002, MySQL AB sued Progress NuSphere for copyright and trademark infringement in US

federal court. NuSphere had allegedly violated MySQL's copyright by linking MySQL's GPL-licensed code with NuSphere Gemini table without complying with the license. After a preliminary hearing on 27 February 2002, the parties entered settlement talks and eventually settled.[f] After the hearing, FSF commented that the judge "made clear that she sees the GNU GPL to be an enforceable and binding license."[80] In August 2003, the SCO Group stated that they believed the GPL to have no legal validity and that they intended to pursue lawsuits over sections of code supposedly copied from SCO Unix into the Linux kernel. This was a problematic stand for them, as they had distributed Linux and other GPL-licensed code in their Caldera OpenLinux distribution, and there is little evidence that they had any legal right to do so except under the terms of the GPL.[citation needed] In February 2018, after a federal circuit court judgment, appeal, and the case being (partially) remanded to the circuit court, the parties restated their remaining claims and provided a plan to move toward final judgement.[81] The remaining claims revolved around Project Monterey and were finally settled in November 2021 by IBM paying $14.25 million to the TSG (previously SCO) bankruptcy trustee.[82] In April 2004, the netfilter/iptables project was granted a preliminary injunction against Sitecom Germany by Munich District Court after Sitecom refused to desist from distributing Netfilter's GPL-licensed software in violation of the terms of the GPL. Harald Welte of Netfilter was represented by ifrOSS co-founder Till Jaeger. In July 2004, the German court confirmed this injunction as a final ruling against Sitecom.[83] The court's justification was that: Defendant has infringed on the copyright of the plaintiff by offering the software 'netfilter/iptables' for download and by advertising its distribution, without adhering to the license conditions of the GPL. Said actions would only be permissible if the defendant had a license grant. ... This is independent of the questions whether the licensing conditions of the GPL have been effectively agreed upon between plaintiff and defendant or not. If the GPL were not agreed upon by the parties, defendant would notwithstanding lack the necessary rights to copy, distribute, and make the software 'netfilter/iptables' publicly available. This exactly mirrored the predictions given previously by the FSF's Eben Moglen. This ruling was important because it was the first time that a court had confirmed that violating terms of the GPL could be a copyright violation and established jurisprudence as to the enforceability of the GPLv2 under German law.[84] In May 2005, Daniel Wallace filed suit against the Free Software Foundation in the Southern District of Indiana, contending that the GPL is an illegal attempt to fix prices (at zero). The suit was dismissed in March 2006, on the grounds that Wallace had failed to state a valid antitrust claim; the court noted that "the GPL encourages, rather than discourages, free competition and the distribution of computer operating systems, the benefits of which directly pass to consumers".[85] Wallace was denied the possibility of further amending his complaint, and was ordered to pay the FSF's legal expenses. On 8 September 2005, the Seoul Central District Court ruled that the GPL was not material to a case dealing with trade secrets derived from GPL-licensed work.[86] Defendants argued that since it is impossible to maintain trade secrets while being compliant with GPL and distributing the work, they are not in breach of trade secrets. This argument was considered without ground. On 6 September 2006, the gpl-violations.org project prevailed in court litigation against D-Link Germany GmbH regarding D-Link's copyright-infringing use of parts of the Linux kernel in storage devices they distributed.[87] The judgment stated that the GPL is valid, legally binding, and stands in a German court.[88] In late 2007, BusyBox developers and the Software Freedom Law Center embarked upon a program to gain GPL compliance from distributors of BusyBox in embedded systems, suing those who would not comply. These were claimed to be the first US uses of courts for enforcement of GPL obligations. (See BusyBox GPL lawsuits.) On 11 December 2008, the Free Software Foundation sued Cisco Systems, Inc. for copyright violations by its Linksys division, of the FSF's GPL-licensed coreutils, readline, Parted, Wget, GNU Compiler Collection, binutils, and GNU Debugger software packages, which Linksys distributes in the Linux firmware[89] of its WRT54G wireless routers, as well as numerous other devices including DSL and Cable modems, Network Attached Storage devices, Voice-Over-IP gateways, virtual private network devices, and a home theater/media player device.[90] After six years of repeated complaints to Cisco by the FSF, claims by Cisco that they would correct, or were correcting, their compliance problems (not providing complete copies of all source code and their modifications), of repeated new violations being discovered and reported with more products, and lack of action by Linksys (a process described on the FSF blog as a "five-years-running game of Whack-a-Mole"[90]) the FSF took them to court. Cisco settled the case six months later by agreeing "to appoint a Free Software Director for Linksys" to ensure compliance, "to notify previous recipients of Linksys products containing FSF programs of their rights under the GPL," to make source code of FSF programs freely available on its website, and to make a monetary contribution to the FSF.[91] In 2011, it was noticed that GNU Emacs had been accidentally releasing some binaries without corresponding source code for two years, contrary to the intended spirit of the GPL, resulting in a copyright violation.[92] Richard Stallman described this incident as a "very bad mistake",[93] which was promptly fixed. The FSF did not sue any downstream redistributors who also unknowingly violated the GPL by distributing these binaries. In 2017 Artifex, the maker of Ghostscript, sued Hancom, the maker of an office suite that included Ghostscript. Artifex offers two licenses for Ghostscript; one is the AGPL License and the other is a commercial license. Hancom did not acquire a commercial license from Artifex nor did it release its office suite as free software. Artifex sued Hancom in US District Court and made two claims. First, Hancom's use of Ghostscript was a violation of copyright; and second, Hancom's use of Ghostscript was a license violation. The court found the GPL license was an enforceable contract and Hancom was in breach of contract.[94][95] On 20 July 2021, the developers of the open-source Stockfish chess engine sued ChessBase, a creator of chess software, for violating the GPLv3 license.[96] It was claimed that Chessbase had made only slight modifications to the Stockfish code and sold the new engines (Fat Fritz 2 and Houdini 6) to their customers.[97] Additionally, Fat Fritz 2 was marketed as if it was an innovative engine. ChessBase had infringed on the license by not distributing these products as Free Software in accordance with the GPL. A year later on 7 November 2022, the parties reached an agreement and ended the dispute. In the near future ChessBase will no longer sell products containing Stockfish code, while informing their customers of this fact with an appropriate notice on their web pages. However, one year later, Chessbase's license would be reinstated. Stockfish did not seek damages or financial compensation.[98][99][100] Quick guide of license compatibility with GPLv3 according to the FSF. Dashed line indicates that the GPLv2 is only compatible with the GPLv3 with the clause "or any later version". Code licensed under several other licenses can be combined with a program under the GPL without conflict, as long as the combination of restrictions on the work as a whole does not put any additional restrictions beyond what GPL allows.[101] In addition to the regular terms of the GPL, there are additional restrictions and permissions one can apply: If a user wants to combine code licensed under different versions of GPL, then this is only allowed if the code with the earlier GPL version includes an "or any later version" statement.[102] For instance, the GPLv3-licensed GNU LibreDWG library cannot be used by LibreCAD and FreeCAD who have GPLv2-only dependencies.[103] Code licensed under LGPL is permitted to be linked with any other code no matter what license that code has,[104] though the LGPL does add additional requirements for the combined work. LGPLv3 and GPLv2-only can thus commonly not be linked, as the combined Code work would add additional LGPLv3 requirements on top of the GPLv2-only licensed software. Code licensed under LGPLv2.x without the "any later version" statement can be relicensed if the whole combined work is licensed to GPLv2 or GPLv3.[105] FSF maintains a list[106] of GPL-compatible free software licenses[107] containing many of the most common free software licenses, such as the original MIT/X license, the BSD license (in its current 3-clause form), and the Artistic License 2.0.[108] Starting from GPLv3, it is unilaterally compatible for materials (like text and other media) under Creative Commons Attribution-ShareAlike 4.0 International License to be remixed into the GPL-licensed materials (prominently software), not vice versa, for niche use cases like game engine (GPL) with game scripts (CC BY-SA).[109][110] David A. Wheeler has advocated that free/open source software developers use only GPL-compatible licenses, because doing otherwise makes it difficult for others to participate and contribute code.[111] As a specific example of license incompatibility, Sun Microsystems' ZFS cannot be included in the GPL-licensed Linux kernel, because it is licensed under the GPL-incompatible Common Development and Distribution License. Furthermore, ZFS is protected by patents, so distributing an independently developed GPL-ed implementation would still require Oracle's permission.[112] A number of businesses use multi-licensing to distribute a GPL version and sell a proprietary license to companies wishing to combine the package with proprietary code, using dynamic linking or not. Examples of such companies include MySQL AB, Digia PLC (Qt framework, before 2011 from Nokia), Red Hat (Cygwin), and Riverbank Computing (PyQt). Other companies, like the Mozilla Foundation (products include Mozilla Application Suite, Mozilla Thunderbird, and Mozilla Firefox), used multi-licensing to distribute versions under the GPL and some other open-source licenses. It is possible to use the GPL for text documents (or more generally for all kinds of media) if it is clear what constitutes the source code (defined as "the preferred form of the work for making changes in it").[113] For manuals and textbooks, though, the FSF recommends the GNU Free Documentation License (GFDL) instead, which it created for this purpose.[114] Nevertheless, the Debian developers recommended (in a resolution adopted in 2006) to license documentation for their project under the GPL, because of the incompatibility of the GFDL with the GPL (text licensed under the GFDL cannot be incorporated into GPL software).[115][116] Also, the FLOSS Manuals foundation, an organization devoted to creating manuals for free software, decided to eschew the GFDL in favor of the GPL for its texts in 2007.[117] If the GPL is used for computer fonts, any documents or images made with such fonts might also have to be distributed under the terms of the GPL. This is not the case in countries that recognize typefaces (the appearance of fonts) as being a useful article and thus not eligible for copyright, but font files as copyrighted computer software (which can complicate font embedding, since the document could be considered 'linked' to the font; in other words, embedding a vector font in a document could force it to be released under the GPL, but a rasterized rendering of the font would not be subject to the GPL). The FSF provides an exception for cases where this is not desired.[118] Historically, the GPL license family has been one of the most popular software licenses in the FOSS domain.[7][119][9][10][11][120] A 1997 survey of MetaLab, then the largest free software archive, showed that the GPL accounted for about half of the software licensed therein.[119] Similarly, a 2000 survey of Red Hat Linux 7.1 found that 53% of the source code was licensed under the GPL.[9] As of 2003[update], about 68% of all projects and 82.1% of the open source industry certified licensed projects listed on SourceForge.net were from the GPL license family.[121] As of August 2008[update], the GPL family accounted for 70.9% of the 44,927 free software projects listed on Freecode.[10] After the release of the GPLv3 in June 2007, adoption of this new GPL version was much discussed[122] and some projects decided against upgrading. For instance the Linux kernel,[40][42] MySQL,[123] BusyBox,[124] AdvFS,[125] Blender,[126][127] VLC media player,[128] and MediaWiki[129] decided against adopting GPLv3. On the other hand, in 2009, two years after the release of GPLv3, Google open-source programs office manager Chris DiBona reported that the number of open-source project licensed software that had moved from GPLv2 to GPLv3 was 50%, counting the projects hosted at Google Code.[11] In 2011, four years after the release of the GPLv3, 6.5% of all open-source license projects are GPLv3 while 42.5% are GPLv2 according to Black Duck Software data.[130][131] Following in 2011 451 Group analyst Matthew Aslett argued in a blog post that copyleft licenses went into decline and permissive licenses increased, based on statistics from Black Duck Software.[132] Similarly, in February 2012 Jon Buys reported that among the top 50 projects on GitHub five projects were under a GPL license, including dual licensed and AGPL projects.[133] GPL usage statistics from 2009 to 2013 was extracted from Freecode data by Walter van Holst while analyzing license proliferation.[12] Usage of GPL family licenses in % on Freecode[12] 2009 2010 2011 2012 2013 2014-06-18[134][135] 72% 63% 61% 59% 58% approx. 54% In August 2013, according to Black Duck Software, the website's data shows that the GPL license family is used by 54% of open-source projects, with a breakdown of the individual licenses shown in the following table.[120] However, a later study in 2013 showed that software licensed under the GPL license family has increased, and that even the data from Black Duck Software has shown a total increase of software projects licensed under GPL. The study used public information gathered from repositories of the Debian Project, and the study criticized Black Duck Software for not publishing their methodology used in collecting statistics.[136] Daniel German, Professor in the Department of Computer Science at the University of Victoria in Canada, presented a talk in 2013 about the methodological challenges in determining which are the most widely used free software licenses, and showed how he could not replicate the result from Black Duck Software.[137] In 2015, according to Black Duck, GPLv2 lost its first position to the MIT license and is now second, the GPLv3 dropped to fourth place while the Apache license kept its third position.[7] Usage of GPL family licenses in the FOSS domain in % according to Black Duck Software License 2008-05-08[138] 2009-03-11[139] 2011-11-22[130] 2013-08-12[120] 2015-11-19[7] 2016-06-06[140] 2017-01-02[141] 2018-06-04[142] GPLv2 58.69% 52.2% 42.5% 33% 23% 21% 19% 14% GPLv3 1.64% 4.15% 6.5% 12% 9% 9% 8% 6% LGPLv2.1 11.39% 9.84% ? 6% 5% 4% 4% 3% LGPLv3 ? (