**File Permission**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **File Permission Types**   |  |  |  | | --- | --- | --- | | User Owner | Group Owner | Other | | r w x | r w x | r w x |   **Read**  The first character of each group represents the *read* permission. There is an r character if the group has the read permission, or a - character if the group does not.   * On a *file*, this allows processes to read the contents of the file, meaning the contents can be viewed and copied. * On a *directory*, file names in the directory can be listed, but other details are not available.   **Write**  The second character of each group represents the *write* permission. There is a w character if the group has the write permission, or a - character if the group does not.   * A *file* can be written to by the process, so changes to a file can be saved. Note that w permission really requires r permission on the file to work correctly. * On a *directory*, files can be added to or removed from the directory. Note that w permission requires x permission on the directory to work correctly.   **Execute**  The third character of each group represents the *execute* permission. There is an x character if the group has the execute permission, or a - character if the group does not.   * A *file* can be executed or run as a process. * On a *directory*, the user can use the cd command to "get into" the directory and use the directory in a pathname to access files and, potentially, subdirectories under this directory. | |
| To change the group owner of an existing file the chgrp command can be used.  To change the group ownership of all of the files of a directory structure, use the recursive -R option to the chgrp command. | $touch sample  $ls -l sample  -rw-rw-r-- 1 sysadmin sysadmin 0 Oct 23 22:12 sample‌⁠​​⁠​  **$ chgrp research sample**  $ ls -l sample  -rw-rw-r--. 1 sysadmin research 0 Oct 23 22:12 sample‌⁠​​⁠​  $ chgrp -R development test\_dir |
| The chown command allows the root user to change the user ownership of files and directories. A regular user cannot use this command to change the user owner of a file, even to give the ownership of one of their own files to another user. However, the chown command also permits changing group ownership, which can be accomplished by either root or the owner of the file. | **# chown jane /tmp/filetest1**  # ls -l /tmp/filetest1  -rw-rw-r-- 1 jane sysadmin 0 Dec 19 18:44 /tmp/filetest1  **# chown jane:users /tmp/filetest2**  # ls -l /tmp/filetest2  -rw-r--r-- 1 jane users 0 Dec 19 18:53 /tmp/filetest2  **jane@localhost:~$ chown .users /tmp/filetest1**  jane@localhost:~$ ls -l /tmp/filetest1  -rw-rw-r-- 1 jane users 0 Dec 19 18:44 /tmp/filetest1 |
| The chmod (change mode) command is used to change permissions on files and directories. Only the root user or the user who owns the file is able to change the permissions of a file. There are two techniques that can be used with this command: symbolic and numeric. Both techniques use the following basic syntax:  chmod new\_permission file\_name  Octal method  Octal Value Permission  4 Read  2 Write  1 Execute | # touch abc.txt  # ls -l abc.txt  -rw-r--r-- 1 root root 0 Dec 19 18:58 abc.txt  **# chmod g+w abc.txt**  # ls -l abc.txt  -rw-rw-r-- 1 root root 0 Dec 19 18:58 abc.txt  **# chmod ug+x,o-r abc.txt**  # ls -l abc.txt  -rwxrwx--- 1 root root 0 Dec 19 18:58 abc.txt  **# chmod u=rx abc.txt**  # ls -l abc.txt  -r-xrwx--- 1 root root 0 Dec 19 18:58 abc.txt  **# chmod 754 abc.txt**  # ls -l abc.txt  -rwxr-xr-- 1 root root 0 Dec 19 18:58 abc.txt |
| Special permission  Typically, special permissions are only set by the administrator (the root user) and they perform very specialized functions. They can be set using the chmod command, using either the symbolic or octal method. | | **Permission** | **Symbol** | **Symbolic** | **Octal Value** | **Purpose** | | --- | --- | --- | --- | --- | | setuid on a file | s | u+s | 4000 | Causes an executable file to execute under user owner identity, instead of the user running the command. | | setgid on a file | s | g+s | 2000 | Causes an executable file to execute under group owner identity, instead of the user running the command. | | setgid on a directory | s | g+s | 2000 | Causes new files and directories that are created inside to be owned by the group that owns the directory.  In addition, any directories created within a directory with the setgid permission set are not only owned by the group that owns the setgid directory, but the new directory automatically has setgid set on it as well. | | sticky on a directory | t | o+t | 1000 | Causes files inside a directory to be able to be removed only by the user owner, or the root user. |   $ ls -l /usr/bin/passwd  -rwsr-xr-x 1 root root 59640 Jan 25 2018 /usr/bin/passwd  To add the setuid permission symbolically, run:  **chmod u+s file**  To add the setuid permission numerically, add 4000 to the file's existing permissions (assume the file originally had 775 for its permission in the following example):  **chmod 4775 file**  To remove the setuid permission symbolically, run:  **chmod u-s file**  To remove the setuid permission numerically, subtract 4000 from the file's existing permissions:  **chmod 0775 file** |
| The umask command is a feature that is used to determine default permissions that are set when a file or directory is created. Default permissions are determined when the *umask value* is subtracted from the maximum allowable default permissions. The maximum default permissions are different for files and directories.  **File rw-rw-rw-**  **Directories rwxrwxrwx**  The new umask is only applied to file and directories created during that session. When a new shell is started, the default umask will again be in effect.  Permanently changing a user's umask requires modifying the .bashrc file located in that user's home directory. | **$ umask**  0002   * The first 0 indicates that the umask is given as an octal number. * The second 0 indicates which permissions to subtract from the default user owner's permissions. * The third 0 indicates which permissions to subtract from the default group owner's permissions. * The last number 2 indicates which permissions to subtract from the default other's permissions.   **$ umask 027**  $ touch sample  $ ls -l sample  -rw-r-----. 1 sysadmin sysadmin 0 Oct 28 20:14 sample  $ mkdir test-dir  $ ls -ld test-dir  drwxr-x---. 1 sysadmin sysadmin 4096 Oct 28 20:25 test-dir  $vi ~/.bashrc  # .bashrc  # Source global definitions  if [ -f /etc/bashrc ]; then  . /etc/bashrc  fi  # User specific environment  PATH="$HOME/.local/bin:$HOME/bin:$PATH"  export PATH  # Uncomment the following line if you don't like systemctl's auto-paging feature:  # export SYSTEMD\_PAGER=  # User specific aliases and functions  umask 0002 |

**Special Permission Scenario**

First, consider the following user accounts:

* The user joe is a member of the staff group.
* The user maya is a member of the payroll group.
* The user steve is a member of the acct group.

In this scenario, these three users need to work on a joint project. They approach the administrator to ask for a shared directory in which they can work together, but that no one else can access their files. The administrator does the following:

* Creates a new group called team.
* Adds joe, maya, and steve to the team group.
* Makes a new directory called shared under the /srv directory.

**root@localhost:~#** mkdir /srv/shared

* Makes the team group the group owner of the shared directory.

**root@localhost:~#** chgrp team /srv/shared

* Make the shared directory writable for the group by giving it the rwxrwx--- permissions:

**root@localhost:~#** chmod 770 /srv/shared

**root@localhost:~#** ls -ld /srv/shared

drwxrwx--- 2 root team 4096 Mar 28 02:14 /srv/shared

This solution is almost perfect: the users joe, maya and steve can now access the /srv/shared directory and add new files. However, there is a potential problem, because the user joe is a member of the staff group, when joe makes a new file in the /srv/shared directory, the new file is owned by the staff group:

-rw-rw----. 2 joe staff 8987 Jan 10 09:08 staffdocument.txt

The problem with this is that neither maya or steve are members of the staff group. As a result, their permissions are ---, which means they can't access the contents of this file.

The user joe could have used the newgrp command to switch to the team group before creating the file. Or, after creating the file, the user joe could have used the chgrp command to change the group ownership to the common group. However, users won't always remember to run these commands; some may not even know that these commands exist.

Instead, the administrator can set up a directory with setgid permission. If a directory is setgid, then all new files created or copied into the directory will *automatically* be owned by the group that owns the directory. This means users don't have to use the newgrp or chgrp commands because the group ownership will be managed automatically.

So, a better solution to this scenario would be to set the setgid permission on the /srv/shared directory using the following command:

**root@localhost:~#** chmod 2775 /srv/shared

Listing the details of the directory after running the previous command would result in the following output:

drwxrwsr-x. 2 root team 4096 Jan 10 09:08 /srv/shared

After completing these steps, the joe, maya, and steve users would now be able to easily create files in the /srv/shared directory that would automatically be owned by the team group.