Permission Scheme for WordPress , and folder security of wordpress

Example Permission Modes

Mode	Str Perms	Explanation
0477	-r—rwxrwx	owner has read only (4), other and group has rwx (7)
0677	-rw-rwxrwx	owner has rw only(6), other and group has rwx (7)
0444	-r—r—r—	all have read only (4)
0666	-rw-rw-rw-	all have rw only (6)
0400	-r———	owner has read only(4), group and others have no permission(0)
0600	-rw——-	owner has rw only, group and others have no permission
0470	-r—rwx—	owner has read only, group has rwx, others have no permission
0407	-r——rwx	owner has read only, other has rwx, group has no permission
0670	-rw-rwx—	owner has rw only, group has rwx, others have no permission
0607	-rw—-rwx	owner has rw only, group has no permission and others have rwx

.

Permissions will be different from host to host, so this guide only details general principles. It cannot cover all cases. This guide applies to servers running a standard setup (note,

for shared hosting using “suexec” methods, see below).

Typically, all files should be owned by your user (ftp) account on your web server, and should be writable by that account. On shared hosts, files should never be owned by the webserver process itself (sometimes this is **www**, or **apache**, or **nobody** user).

Any file that needs write access from WordPress should be owned or group-owned by the user account used by the WordPress (which may be different than the server account). For example, you may have a user account that lets you FTP files back and forth to your server, but your server itself may run using a separate user, in a separate usergroup, such as **dhapache** or **nobody**. If WordPress is running as the FTP account, that account needs to have write access, i.e., be the owner of the files, or belong to a group that has write access. In the latter case, that would mean permissions are set more permissively than default (for example, 775 rather than 755 for folders, and 664 instead of 644).

The file and folder permissions of WordPress should be the same for most users, depending on the type of installation you performed and the umask settings of your system environment at the time of install.

Typically, all core WordPress files should be writable only by your user account (or the httpd account, if different). (Sometimes though, multiple ftp accounts are used to manage an install, and if all ftp users are known and trusted, i.e., not a shared host, then assigning group writable may be appropriate. Ask your server admin for more info.) However, if you utilize mod_rewrite Permalinks or other .htaccess features you should make sure that WordPress can also write to your /.htaccess file.

If you want to use the built-in theme editor, all files need to be group writable. Try using it before modifying file

permissions, it should work. (This may be true if different users uploaded the WordPress package and the Plugin or Theme. This wouldn't be a problem for Plugin and Themes installed via the admin. When uploading files with different ftp users group writable is needed. On shared hosting, make sure the group is exclusive to users you trust… the apache user shouldn't be in the group and shouldn't own files.)

Some plugins require the /wp-content/ folder be made writeable, but in such cases they will let you know during installation. In some cases, this may require assigning 755 permissions. The same is true for /wp-content/cache/ and maybe /wp-content/uploads/(if you're using MultiSite you may also need to do this for /wp-content/blogs.dir/)

Additional directories under /wp-content/ should be documented by whatever plugin / theme requires them. Permissions will vary.

Shared Hosting with su exec

The above may not apply to shared hosting systems that use the "suexec" approach for running PHP binaries. This is a popular approach used by many web hosts. For these systems, the php process runs as the owner of the php files themselves, allowing for a simpler configuration and a more secure environment for the specific case of shared hosting.

Note: suexec methods should NEVER be used on a single-site server configuration, they are more secure **only** for the specific case of shared hosting.

In such an suexec configuration, the correct permissions scheme is simple to understand.

- All files should be owned by the actual user's account, not the user account used for the httpd process.
- Group ownership is irrelevant, unless there's specific

group requirements for the web-server process permissions checking. This is not usually the case.
- All directories should be 755 or 750.
- All files should be 644 or 640. Exception: wp-config.php should be 600 to prevent other users on the server from reading it.
- No directories should ever be given 777, even upload directories. Since the php process is running as the owner of the files, it gets the owners permissions and can write to even a 755 directory.

In this specific type setup, WordPress will detect that it can directly create files with the proper ownership, and so it will not ask for FTP credentials when upgrading or installing plugins.

Using an FTP Client

FTP programs ("clients") allow you to set permissions for files and directories on your remote host. This function is often called chmodor set permissions in the program menu.

In a WordPress install, two files that you will probably want to alter are the index page, and the css which controls the layout. Here's how you change index.php — *the process is the same for any file*.

In the screenshot below, look at the last column — that shows the permissions. It looks a bit confusing, but for now just note the sequence of letters.

Initial permissions

Right-click 'index.php' and select 'File Permissions'
A popup screen will appear.

Altering file permissions

Don’t worry about the check boxes. Just delete the ‘Numeric value:’ and enter the number you need – in this case it’s 666. Then click OK.

Permissions have been altered

You can now see that the file permissions have been changed.

Unhide the hidden files

By default, most FTP Clients, including FileZilla, keep hidden files, those files beginning with a period (.), from being displayed. But, at some point, you may need to see your hidden files so that you can change the permissions on that file. For example, you may need to make your .htaccess file, the file that controls permalinks, writeable.

To display hidden files in FileZilla, in it is necessary to select ‘View’ from the top menu, then select ‘Show hidden files’. The screen display of files will refresh and any previously hidden file should come into view.

To get FileZilla to always show hidden files – under Edit, Settings, Remote File List, check the Always show hidden files box.

In the latest version of Filezilla, the ‘Show hidden files’ option was moved to the ‘Server’ tab. Select ‘Force show hidden files.’

Using the Command Line

If you have shell/SSH access to your hosting account, you can use chmod to change file permissions, which is the preferred method for experienced users. Before you start using chmod it would be recommended to read some tutorials to make sure you understand what you can achieve with it. Setting incorrect permissions can take your site offline, so please take your time.

- Unix Permissions
- Apple Chmod Reference

You can make **all** the files in your wp-content directory writable in two steps, but before making every single file and folder writable you should first try safer alternatives like modifying just the directory. Try each of these commands first and if they dont work then go recursive, which will make even your themes image files writable. Replace **DIR** with the folder you want to write in
[crayon-669e34196cd41127878469/]
If those fail to allow you to write, try them all again in order, except this time replace -v with -R, which will recursively change each file located in the folder. If after that you still cant write, you may now try 777.

About Chmod

chmod is a unix command that means "**ch**ange **mod**e" on a file. The -R flag means to apply the change to every file and directory inside of wp-content. 766 is the mode we are changing the directory to, it means that the directory is readable and writable by WordPress and any and all other users on your system. Finally, we have the name of the directory we are going to modify, wp-content. If 766 doesn't work, you can try 777, which makes all files and folders readable, writable,

and executable by all users, groups, and processes.

If you use Permalinks you should also change permissions of .htaccess to make sure that WordPress can update it when you change settings such as adding a new page, redirect, category, etc.. which requires updating the .htaccess file when mod_rewrite Permalinks are being used.

1. Go to the main directory of WordPress
2. Enter chmod -v 666 .htaccess

NOTE: From a security standpoint, even a small amount of protection is preferable to a world-writeable directory. Start with low permissive settings like 744, working your way up until it works. Only use 777 if necessary, and hopefully only for a temporary amount of time.

The dangers of 777

The crux of this permission issue is how your server is configured. The username you use to FTP or SSH into your server is most likely not the username used by the server application itself to serve pages.
[crayon-669e34196cd49676510550/]
Often the Apache server is 'owned' by the **dhapache** or **nobody** user accounts. These accounts have a limited amount of access to files on the server, for a very good reason. By setting your personal files and folders owned by your user account to be World-Writable, you are literally making them World Writable. Now the dhapache and nobody users that run your server, serving pages, executing php interpreters, etc.. will have full access to your user account files.

This provides an avenue for someone to gain access to your files by hijacking basically any process on your server, this also includes any other users on your machine. So you should think carefully about modifying permissions on your machine.

I’ve never come across anything that needed more than 767, so when you see 777 ask why its necessary.

The Worst Outcome

The worst that can happen as a result of using 777 permissions on a folder or even a file, is that if a malicious cracker or entity is able to upload a devious file or modify a current file to execute code, they will have complete control over your blog, including having your database information and password.

Find a Workaround

Its usually pretty easy to have the enhanced features provided by the impressive WordPress plugins available, without having to put yourself at risk. Contact the Plugin author or your server support and request a workaround.

Finding Secure File Permissions

The .htaccess file is one of the files that is accessed by the owner of the process running the server. So if you set the permissions too low, then your server won’t be able to access the file and will cause an error. Therein lies the method to find the most secure settings. Start too restrictive and increase the permissions until it works.

Example Permission Settings

The following example has a *custom compiled php-cgi binary* and

a *custom php.ini* file located in the cgi-bin directory for executing php scripts. To prevent the interpreter and php.ini file from being accessed directly in a web browser they are protected with a .htaccess file.

Default Permissions (umask 022)
[crayon-669e34196cd4d395162095/]
Secured Permissions
[crayon-669e34196cd4f430724721/]

.htaccess permissions

644 > 604 – The bit allowing the group owner of the .htaccess file read permission was removed. 644 is normally required and recommended for .htaccess files.

php.ini permissions

644 > 600 – Previously all groups and all users with access to the server could access the php.ini, even by just requesting it from the site. The tricky thing is that because the php.ini file is only used by the php.cgi, we only needed to make sure the php.cgi process had access. The php.cgi runs as the same user that owns both files, so that single user is now the only user able to access this file.

php.cgi permissions

755 > 711 This file is a compiled php-cgi binary used instead of mod_php or the default vanilla php provided by the hosting company. The default permissions for this file are 755.

php5.cgi permissions

755 > 100 – Because of the setup where the user account is the

owner of the process running the php cgi, no other user or group needs access, so we disable all access except execution access. This is interesting because it really works. You can try reading the file, writing to the file, etc.. but the only access you have to this file is to run php scripts. And as the owner of the file you can always change the permission modes back again.

[crayon-669e34196cd52375495762/]