Plugin Basics

- Getting Started
- Hooks: Actions and Filters
  - Basic Hooks
  - Adding Hooks
  - Removing Hooks
- WordPress APIs
- How WordPress Loads Plugins
- Sharing your Plugin

Getting Started

At its simplest, a WordPress plugin is a PHP file with a WordPress plugin header comment. It's highly recommended that you create a directory to hold your plugin so that all of your plugin's files are neatly organized in one place.

To get started creating a new plugin, follow the steps below.

1. Navigate to the WordPress installation's `wp-content/` directory.
2. Open the `plugins/` directory.
3. Create a new directory and name it after the plugin (e.g. `plugin-name/`).
4. Open the new plugin's directory.
5. Create a new PHP file (it's also good to name this file after your plugin, e.g. `plugin-name.php`).

Here's what the process looks like on the Unix command line:

```
$ cd wp-content/
$ cd plugins/
$ mkdir plugin-name/
$ cd plugin-name/
$ vim plugin-name.php
```

In the example above, `vim` is the name of the text editor. Use whichever editor is comfortable for you.

Now that you're editing your new plugin's PHP file, you'll need to add a plugin header comment. This is a specially formatted PHP block comment that contains metadata about the plugin, such as its name, author, version, license, etc. The plugin header comment must comply with the header requirements, and at the very least, contain the name of the plugin.

Only one file in the plugin's folder should have the header comment — if the plugin has multiple PHP files, only one of those files should have the header comment.

After you save the file, you should be able to see your plugin listed in your WordPress site. Log in to your WordPress site, and click **Plugins** on the left navigation pane of your WordPress Admin. This page displays a listing of all the plugins your WordPress site has. Your new plugin should now be in that list!

Hooks: Actions and Filters

WordPress hooks allow you to tap into WordPress at specific points to change how WordPress behaves without editing any core files.

There are two types of hooks within WordPress: *actions* and *filters*. Actions allow you to add or change WordPress functionality, while filters allow you to alter content as it is loaded and displayed to the website user.

Hooks are not just for plugin developers; hooks are used extensively to provide default functionality by WordPress core itself. Other hooks are unused place holders that are simply available for you to tap into when you need to alter how WordPress works. This is what makes WordPress so flexible.

Basic Hooks

The 3 basic hooks you'll need when creating a plugin are the register_activation_hook(), the register_deactivation_hook(), and the register_uninstall_hook().

The activation hook is run when you *activate* your plugin. You would use this to provide a function to set up your plugin — for example, creating some default settings in the `options` table.

The deactivation hook is run when you *deactivate* your plugin. You would use this to provide a function that clears any temporary data stored by your plugin.

These uninstall methods are used to clean up after your plugin is *deleted* using the WordPress Admin. You would use this to delete all data created by your plugin, such as any options that were added to the `options` table.

Adding Hooks

You can add your own, custom hooks with do_action() , which will enable developers to extend your plugin by passing functions through your hooks.

Removing Hooks

You can also use invoke remove_action() to remove a function that was defined earlier. For example, if your plugin is an add-on to another plugin, you can use remove_action() with the same function callback that was added by the previous plugin with add_action(). The priority of actions is important in these situations, as remove_action() would need to run after the initial add_action().

You should be careful when removing an action from a hook, as well as when altering priorities, because it can be difficult to see how these changes will affect other interactions with the same hook. We highly recommend testing frequently.

You can learn more about creating hooks and interacting with them in the Hooks section of this handbook.

WordPress APIs

Did you know that WordPress provides a number of Application Programming Interfaces (APIs)? These APIs can greatly simplify the code you need to write in your plugins. You don't want to reinvent the wheel, especially when so many people have done a lot of the work and testing for you.

The most common one is the Options API, which makes it easy to store data in the database for your plugin. If you're thinking of using cURL in your plugin, the HTTP API might be of interest to you.

Since we're talking about plugins, you'll want to study the Plugin API. It has a variety of functions that will assist you in developing plugins.

How WordPress Loads Plugins

When WordPress loads the list of installed plugins on the Plugins page of the WordPress Admin, it searches through the `plugins/` folder (and its sub-folders) to find PHP files with WordPress plugin header comments. If your entire plugin consists of just a single PHP file, like Hello Dolly, the file could be located directly inside the root of the `plugins/` folder. But more commonly, plugin files will reside in their own folder, named after the plugin.

Sharing your Plugin

Sometimes a plugin you create is just for your site. But many people like to share their plugins with the rest of the WordPress community. Before sharing your plugin, one thing you need to do is choose a license. This lets the user of your plugin know how they are allowed to use your code. To maintain compatibility with WordPress core, it is recommended that you pick a license that works with GNU General Public License (GPLv2+).

□□□ #2
□ □□□ □□□ 2026-10-26 07:37:07 UTC
□ □□□ □□□ 2026-10-26 07:37:07 UTC