Cron

- Cron
- Hooking WP-Cron Into the System Task Scheduler
- Scheduling WP Cron Events
- Testing of WP-Cron
- Understanding WP-Cron Scheduling

Cron

What is WP-Cron

WP-Cron is how WordPress handles scheduling time-based tasks in WordPress. Several WordPress core features, such as checking for updates and publishing scheduled post, utilize WP-Cron. The "Cron" part of the name comes from the cron time-based task scheduling system that is available on UNIX systems.

WP-Cron works by checking, on every page load, a list of scheduled tasks to see what needs to be run. Any tasks due to run will be called during that page load.

[info]WP-Cron does not run constantly as the system cron does; it is only triggered on page load.[/info]

Scheduling errors could occur if you schedule a task for 2:00PM and no page loads occur until 5:00PM.

Why use WP-Cron

- WordPress core and many plugins need a scheduling system to perform time-based tasks. However, many hosting services are shared and do not provide access to the system scheduler.
- Using the WordPress API is a simpler method for setting scheduled tasks than going outside of WordPress to the system scheduler.
- With the system scheduler, if the time passes and the task did not run, it will not be re-attempted. With WP-Cron, all scheduled tasks are put into a queue and will run at the next opportunity (meaning the next page load). So while you can’t be 100% sure *when* your task will run, you can be 100% sure that it will run *eventually*.

Hooking WP-Cron Into the System Task Scheduler

As previously mentioned, WP-Cron does not run continuously, which can be an issue if there are critical tasks that must run on time. There is an easy solution for this. Simply set up your system's task scheduler to run on the intervals you desire (or at the specific time needed). The easiest solution is to use a tool to make a web request to the `wp-cron.php` file.

After scheduling the task on your system, there is one more step to complete. WordPress will continue to run WP-Cron on each page load. This is no longer necessary and will contribute to extra resource usage on your server. WP-Cron can be disabled in the `wp-config.php` file. Open the `wp-config.php` file for editing and add the following line:

```
define( 'DISABLE_WP_CRON', true );
```

Windows

Windows calls their time based scheduling system the Task Scheduler. It can be accessed via the **Administrative Tools** in the control panel.

How you setup the task varies with server setup. One method is to use PowerShell and a Basic Task. After creating a Basic Task the following command can be used to call the WordPress Cron script.

```
powershell "Invoke-WebRequest http://YOUR_SITE_URL/wp-cron.php"
```

MacOS and Linux

Mac OS X and Linux both use cron as their time based scheduling system. It is typically access from the terminal with the `crontab -e` command. It should be noted that tasks will be run as a regular user or as root depending on the system user running the command.

Cron has a specific syntax that needs to be followed and contains the following parts:

- Minute

- Hour
- Day of month
- Month
- Day of week
- Command to execute

Cron Scheduling

If a command should be run regardless of one of the time sections an asterisk (`*`) should be used. For example if you wanted to run a command every 15 minutes regardless of the hour, day, or month it would look like:

Many servers have `wget` installed and this is an easy tool to call the WordPress Cron script.

```
wget --delete-after https://example.com/wp-cron.php
```

[info]Without –delete-after option, wget would save the output of the HTTP GET request.[/info]

A daily call to your site's WordPress Cron that triggers at midnight every night could look similar to:

```
0 0 * * * wget --delete-after https://example.com/wp-cron.php
```

Scheduling WP Cron Events

The WP Cron system uses hooks to add new scheduled tasks.

Adding the Hook

In order to get your task to run you must create your own custom hook and give that hook the name of a function to execute. This is a very important step. Forget it and your task will never run.

The following example will create a hook. The first parameter is the name of the hook you are creating, and the second is the name of the function to call.

```
add_action( 'bl_cron_hook', 'bl_cron_exec' );
```

[info]Remember, the `bl_` part of the function name is a *function prefix*. You can learn why prefixes are important here.[/info]

[info]You can read more about actions here.[/info]

Scheduling the Task

An important note is that WP-Cron is a simple task scheduler. As we know, tasks are added by the hook created to call the function that runs the desired task. However if you call `wp_schedule_event()` multiple times, even with the same hook name, the event will be scheduled multiple times. If your code adds the task on each page load this could result in the task being scheduled several thousand times. This is not what you want.

WordPress provides a convenient function called `wp_next_scheduled()` to check if a particular hook is already scheduled. `wp_next_scheduled()` takes one parameter, the hook name. It will return either a string containing the timestamp of the next execution or false, signifying the task is not scheduled. It is used like so:

```
wp_next_scheduled( 'bl_cron_hook' );
```

Scheduling a recurring task is accomplished with `wp_schedule_event()`. This function takes three required parameters, and one additional parameter that is an array that can be passed to the function executing the wp-cron task. We will focus on the first three parameters. The parameters

are as follows:

1. `$timestamp` – The UNIX timestamp of the first time this task should execute.
2. `$recurrence` – The name of the interval in which the task will recur in seconds.
3. `$hook` – The name of our custom hook to call.

We will use the 5 second interval we created here and the hook we created above, like so:

```
wp_schedule_event( time(), 'five_seconds', 'bl_cron_hook' );
```

Remember, we need to first ensure the task is not already scheduled. So we wrap the scheduling code in a check like this:

```
if ( ! wp_next_scheduled( 'bl_cron_hook' ) ) {
  wp_schedule_event( time(), 'five_seconds', 'bl_cron_hook' );
}
```

Unscheduling tasks

When you no longer need a task scheduled you can unschedule tasks with `wp_unschedule_event()`. This function takes the following two parameters:

1. `$timestamp` – Timestamp of the next occurrence of the task.
2. `$hook` – Name of the custom hook to be called.

This function will not only unschedule the task indicated by the timestamp, it will also unschedule all future occurrences of the task. Since you probably will not know the timestamp for the next task, there is another handy function, `wp_next_scheduled()` that will find it for you. `wp_next_scheduled()` takes one parameter (that we care about):

1. `$hook` – The name of the hook that is called to execute the task.

Put it all together and the code looks like:

```
$timestamp = wp_next_scheduled( 'bl_cron_hook' );
wp_unschedule_event( $timestamp, 'bl_cron_hook' );
```

It is very important to unschedule tasks when you no longer need them because WordPress will continue to attempt to execute the tasks, even though they are no longer in use (or even after your plugin has been deactivated or removed). An important place to remember to unschedule your tasks is upon plugin deactivation.

Unfortunately there are many plugins in the WordPress.org Plugin Directory that do not clean up after themselves. If you find one of these plugins please let the author know to update their code. WordPress provides a function called `register_deactivation_hook()` that allows developers to run a function when their plugin is deactivated. It is very simple to setup and looks like:

```
register_deactivation_hook( __FILE__, 'bl_deactivate' );

function bl_deactivate() {
  $timestamp = wp_next_scheduled( 'bl_cron_hook' );
  wp_unschedule_event( $timestamp, 'bl_cron_hook' );
}
```

[info]You can read more about activation and deactivation hooks here.[/info]

Testing of WP-Cron

WP-CLI

Cron jobs can be tested using WP-CLI. It offers commands like `wp cron event list` and `wp cron event run {job name}`. Check the documentation for more details.

WP-Cron Management Plugins

Several plugins are available on the WordPress.org Plugin Directory for viewing, editing, and controlling the scheduled cron events and available schedules on your site.

`_get_cron_array()`

The `_get_cron_array()` function returns an array of all currently scheduled cron events. Use this function if you need to inspect the raw list of events.

`wp_get_schedules()`

The `wp_get_schedules()` function returns an array of available event recurrence schedules. Use this function if you need to inspect the raw list of available schedules.

Understanding WP-Cron Scheduling

Unlike a traditional system cron that schedules tasks for specific times (e.g. "every hour at 5 minutes past the hour"), WP-Cron uses intervals to simulate a system cron.

WP-Cron is given two arguments: the time for the first task, and an interval (in seconds) after which the task should be repeated. For example, if you schedule a task to begin at 2:00PM with an interval of 300 seconds (five minutes), the task would first run at 2:00PM and then again at 2:05PM, then again at 2:10PM, and so on, every five minutes.

To simplify scheduling tasks, WordPress provides some default intervals and an easy method for adding custom intervals.

The default intervals provided by WordPress are:

- hourly
- twicedaily
- daily
- weekly (since WP 5.4)

Custom Intervals

To add a custom interval, you can create a filter, such as:

```
add_filter( 'cron_schedules', 'example_add_cron_interval' );
function example_add_cron_interval( $schedules ) {
  $schedules['five_seconds'] = array(
    'interval' => 5,
    'display'  => esc_html__( 'Every Five Seconds' ),
  );
  return $schedules;
}
```

This filter function creates a new interval that will allow us to run a cron task every five seconds.

[info]All intervals are in seconds.[/info]