

Introducing Cron

- Nikhil Sheth

What is Cron?

Cron is very simply a Linux module that allows you to run commands at predetermined times or intervals. In Windows, it's called Scheduled Tasks. The name Cron is in fact derived from the same word from which we get the word chronology, which means order of time.

Using Cron, a developer can automate such tasks as mailing ezines that might be better sent during an off-hour, automatically updating stats, or the regeneration of static pages from dynamic sources. Systems administrators and Web hosts might want to generate quota reports on their clients, complete automatic credit card billing, or similar tasks. Cron has something for everyone!

Breaking Cron Down

Now, let's try to set up the automated running of mailstock.php. If you are a systems administrator, you'll be able to interface directly with Cron. On a standard Redhat setup, jobs are executed from /etc/crontab at predetermined intervals. We set these intervals from within four directories that sort the jobs into hourly, daily, weekly, and monthly intervals.

Sometimes, however, you may want to execute a script overnight, on a weekend, or at some other time at which the server is not likely to be under a great deal of strain. This is where the crontab program comes into play. This executable is actually a go-between that exists between the users, who do not have permissions over the Cron folders and files, and the system that's needed to run the Cron jobs.

Many hosts provide a GUI to administer the Cron jobs. This can simplify the setting up of Cron jobs, however, in the interests of understanding the nuts and bolts of this process, in this tutorial we'll look at cron jobs from the command line perspective. We will approach this from the point of view of a normal user, but the process is essentially identical for System Administrators as well.

The easiest way to use crontab is via the crontab command.

```
# crontab -e
```

This command 'edits' the crontab. Upon employing this command, you will be able to enter the commands that you wish to run. My version of Linux uses the text editor vi. You can find information on using vi [here](#).

The syntax of this file is very important - if you get it wrong, your crontab will not function properly. The syntax of the file should be as follows:

```
minutes hours day_of_month month day_of_week command
```

All the variables, with the exception of the command itself, are numerical constants. In addition to an asterisk (*), which is a wildcard that allows any value, the ranges permitted for each field are as follows:

Minutes: 0-59
Hours: 0-23
Day_of_month: 1-31
Month: 1-12
Weekday: 0-6

We can also include multiple values for each entry, simply by separating each value with a comma.

command can be any shell command and, as we will see momentarily, can also be used to execute a Web document such as a PHP file.

So, if we want to run a script every Tuesday morning at 8:15 AM, our mycronjob file will contain the following content on a single line:

```
15 8 * * 2 /path/to/scriptname
```

This all seems simple enough, right? Not so fast! If you try to run a PHP script in this manner, nothing will happen (barring very special configurations that have PHP compiled as an executable, as opposed to an Apache module). The reason is that, in order for PHP to be parsed, it needs to be passed through Apache. In other words, the page needs to be called via a browser or other means of retrieving Web content.

For our purposes, I'll assume that your server configuration includes wget, as is the case with most default configurations. To test your configuration, log in to shell. If you're using an RPM-based system (e.g. Redhat or Mandrake), type the following:

```
# wget --help
```

If you are greeted with a wget package identification, it is installed in your system.

You could execute the PHP by invoking wget on the URL to the page, like so:

```
# wget http://www.example.com/file.php
```

Summary

There are literally tons of things that can be done with PHP, and as many that can be performed with Cron. Hopefully this gives you a brief overview of some of the possibilities.

You can actually combine more elaborate PHP schemes into a single cron job simply by executing them via a single PHP file. As I mentioned before, it's even possible to bypass wget by using PHP as a shell scripting language. This would require PHP to be compiled as a standalone binary, and that would bring with it a number of risks, but it can (and has!) been done.

The bottom line is that anything you can accomplish with PHP, you can likely automate with Cron. Happy coding!