

Common Issues



- [Error: Can't connect to the database](#)
 - [Running wp --info produces HTML output](#)
 - [PHP Fatal error: Cannot redeclare wp_unregister_GLOBALS\(\)](#)
 - [PHP Fatal error: Call to undefined function <WordPress function>](#)
 - [PHP Fatal error: Call to undefined function cli\posix_isatty\(\)](#)
 - [PHP Fatal error: Allowed memory size of 999999 bytes exhausted \(tried to allocate 99 bytes\)](#)
 - [Error: YIKES! It looks like you're running this as root.](#)
 - [PHP notice: Undefined index on \\$_SERVER superglobal](#)
 - [PHP notice: Use of undefined constant STDOUT](#)
 - [PHP Parse error: syntax error, unexpected '?' in ... /php/WP_CLI/Runner.php ... eval\(\)'d code on li](#)
 - [Can't find wp-content directory / use of \\$_SERVER\['document_root'\]](#)
 - [Conflict between global parameters and command arguments](#)
 - [Warning: Some code is trying to do a URL redirect](#)
 - [Cannot create a post with Latin characters in the title on Windows](#)
 - [Arguments containing newlines are truncated on Windows](#)
 - [The installation hangs](#)
 - [W3 Total Cache Error: some files appear to be missing or out of place.](#)
 - [The automated updater doesn't work for versions before 3.4](#)
 - [PHP Fatal error: Maximum function nesting level of '#' reached, aborting!](#)
-

Error: Can't connect to the database

A few possibilities:

a) you're using MAMP, but WP-CLI is not using the MAMP PHP binary.

You can check which PHP WP-CLI is using by running `wp --info`.

If you need to specify an alternate PHP binary, see [using a custom PHP binary](#).

b) it's a WordPress multisite install.

c) the database credentials in `wp-config.php` are actually incorrect.

Running `wp --info` produces HTML output

If you run `wp --info` on a server with Phar support disabled, you may see:

```
$ wp --info
Content-type: text/html; charset=UTF-8
```

When using the WP-CLI Phar, you'll need to whitelist Phar support in your `php.ini`:

```
suhosin.executor.include.whitelist = phar
```

PHP Fatal error: Cannot redeclare wp_unregister_GLOBALS()

If you get this fatal error running the `wp` command, you may have moved or edited `wp-config.php` beyond what wp-cli supports:

```
PHP Fatal error: Cannot redeclare wp_unregister_GLOBALS() (previously declared in
/var/www/foo.com/wp-includes/load.php:18) in /var/www/foo.com/wp-includes/load.php on line 33
```

One of WP-CLI's requirements is that the line:

```
require_once(ABSPATH . 'wp-settings.php');
```

remains in the `wp-config.php` file, so if you've modified or moved it, put it back there. It gets matched by a regex when WP-CLI runs.

PHP Fatal error: Call to undefined function <WordPress function>

Before WP-CLI can load `wp-settings.php`, it needs to know all of the constants defined in `wp-config.php` (database connection details and so on). Because WP-CLI doesn't want WordPress to load yet when it's [pulling the constants](#) out of `wp-config.php`, [it uses regex](#) to strip the `require_once(ABSPATH . 'wp-settings.php');` statement.

If you've modified your `wp-config.php` in a way that calls WordPress functions, PHP will fail out with a fatal error, as your `wp-config.php` is calling a WordPress function before WordPress has been loaded to define it.

Example:

```
$ wp core check-update
PHP Fatal error: Call to undefined function add_filter() in
phar:///usr/local/bin/wp/php/WP_CLI/Runner.php(952) : eval()'d code on line N
```

Modifying `wp-config.php` beyond constant definitions is not best practice. You should move any modifications to a [WordPress mu-plugin](#), which will retain the functionality of your modifications while allowing wp-cli to parse your wp-config.php without throwing a PHP error, as well as preventing other errors.

See: [#1631](#)

PHP Fatal error: Call to undefined function cli\posix_isatty()

Please ensure you have the php-process extension installed. For example for Centos 6: `yum install php-process`

PHP Fatal error: Allowed memory size of 999999 bytes exhausted (tried to allocate 99 bytes)

If you run into a PHP fatal error relating to memory when running `wp package install`, you're likely running out of memory.

WP-CLI uses Composer under the hood to manage WP-CLI packages. However, Composer is a bit of a memory hog, so you'll need to increase your memory limit to accommodate it.

Edit your `php.ini` as a permanent fix:

```
# Find your php.ini for PHP-CLI
$ php -i | grep php.ini
Configuration File (php.ini) Path => /usr/local/etc/php/7.0
Loaded Configuration File => /usr/local/etc/php/7.0/php.ini
# Increase memory_limit to 512M or greater
$ vim /usr/local/etc/php/7.0/php.ini
memory_limit = 512M
```

Set `memory_limit` on the fly as a temporary fix:

```
$ php -d memory_limit=512M "$(which wp)" package install <package-name>
```

If your PHP process is still running out of memory, try these steps:

1. Restart PHP.
2. Check for additional `php.ini` files:

```
$ php -i | grep additional
Scan this dir for additional .ini files => /usr/local/etc/php/7.1/conf.d
# Edit the additional file(s) and increase the memory_limit to 512M or greater
$ vim /usr/local/etc/php/7.1/conf.d
memory_limit = 512M
```

Error: YIKES! It looks like you're running this as root.

Running WP-CLI as root is extremely dangerous. When you execute WP-CLI as root, any code within your WordPress instance (including third-party plugins and themes you've installed) will have full privileges to the entire server. This can enable malicious code within the WordPress instance to compromise the entire server.

The WP-CLI project strongly discourages running WP-CLI as root.

See also: [#973](#)

PHP notice: Undefined index on `$_SERVER` superglobal

The `$_SERVER` superglobal is an array typically populated by a web server with information such as headers, paths, and script locations. PHP CLI doesn't populate this variable, nor does WP-CLI, because many of the variable details are meaningless at the command line.

Before accessing a value on the `$_SERVER` superglobal, you should check if the key is set:

```
if ( isset( $_SERVER['HTTP_X_FORWARDED_PROTO'] ) && 'https' ===  
$_SERVER['HTTP_X_FORWARDED_PROTO'] ) {  
    $_SERVER['HTTPS']='on';  
}
```

When using `$_SERVER['HTTP_HOST']` in your `wp-config.php`, you'll need to set a default value in WP-CLI context:

```
if ( defined( 'WP_CLI' ) && WP_CLI && ! isset( $_SERVER['HTTP_HOST'] ) ) {  
    $_SERVER['HTTP_HOST'] = 'example.com';  
}
```

See also: [#730](#)

PHP notice: Use of undefined constant STDOUT

The `STDOUT` constant is defined by the PHP CLI. If you receive an error notice that `STDOUT` is missing, it's likely because you're not running WP-CLI by PHP CLI. Please review your server configuration accordingly.

PHP Parse error: syntax error, unexpected '?' in ... /php/WP_CLI/Runner.php ... eval()'d code on line 1

If you get this error running the `wp` command, the most likely cause is a [Unicode BOM](#) at the start of your `wp-config.php`. This issue will be addressed in a future release of WP-CLI, but in the meantime you can solve the issue by running:

```
$ sed -i '1s/^\xEF\xBB\xBF//' $(wp config path)
```

or by manually removing the `BOM` using your favorite editor.

See also: [wp-cli/search-replace-command#71](#)

Can't find wp-content directory / use of `$_SERVER['document_root']`

`$_SERVER['document_root']` is defined by the webserver based on the incoming web request. Because this type of context is unavailable to PHP CLI, `$_SERVER['document_root']` is unavailable to WP-CLI. Furthermore, WP-CLI can't safely mock `$_SERVER['document_root']` as it does with `$_SERVER['http_host']` and a few other `$_SERVER` values.

If you're using `$_SERVER['document_root']` in your `wp-config.php` file, you should instead use `dirname( __FILE__ )` or similar.

See also: [#785](#)

Conflict between global parameters and command arguments

All of the [global parameters](#) (e.g. `--url=<url>`) may conflict with the arguments you'd like to accept for your command. For instance, adding a RSS widget to a sidebar will not populate the feed URL for that widget:

```
$ wp widget add rss sidebar-1 1 --url="http://www.smashingmagazine.com/feed/" --items=3
Success: Added widget to sidebar.
```

- Expected result: widget has the feed URL set.
- Actual result: widget is added with the number of items set to 3, but with empty feed URL.

Use the `WP_CLI_STRICT_ARGS_MODE` environment variable to tell WP-CLI to treat any arguments before the command as global, and after the command as local:

```
WP_CLI_STRICT_ARGS_MODE=1 wp --url=wp.dev/site2 widget add rss sidebar-1 1 --url="http://wp-cli.org/feed/"
```

In this example, `--url=wp.dev/site2` is the global argument, setting WP-CLI to run against 'site2' on a WP multisite install. `--url="http://wp-cli.org/feed/"` is the local argument, setting the RSS feed widget with the proper URL.

See also: [#3128](#)

Warning: Some code is trying to do a URL redirect

Most of the time, it's some plugin or theme code that disables wp-admin access to non-admins.

Quick fix, other than disabling the protection, is to pass the user parameter: `--user=some_admin`

See also: [#477](#)

Cannot create a post with Latin characters in the title on Windows

Considering the following example:

```
wp post create --post_title="Perícias Contábeis"
```

Using UTF-8 in PHP arguments doesn't work on Windows for PHP <= 7.0, however it will work for PHP >= 7.1, as it was fixed as part of [Support for long and UTF-8 path](#). A workaround for PHP <= 7.0 is to use the `--prompt` option:

```
echo "Perícias Contábeis" | wp post create --post_type=page --post_status=publish --prompt=post_title
```

See also: [#4714](#)

Arguments containing newlines are truncated on Windows

On Windows, `wp` is a `.bat` wrapper around `php wp-cli.phar`. Batch files terminate a command at a newline, including a newline inside a quoted argument, because `cmd.exe` splits the line before quoting is evaluated.

Any flags positioned after the multi-line value never reach WP-CLI. The command still succeeds with the arguments it did receive, so there is no error — WordPress silently applies its defaults for the missing values. In the example below the post is created, but as a draft under the site's default category and attributed to the current user, because everything after `--post_content` (`--post_category`, `--post_author` and `--post_status`) was dropped:

```
wp post create --post_content="<multi-line content>" --post_category=3 --post_author=2 --post_status=publish
```

Pass the content over `STDIN` instead, using `-` as the positional argument. The newline then lives in the piped stream rather than in the command line, so nothing is truncated:

```
type content.html | wp post create - --post_status=publish --post_category=3 --post_author=2
```

The same applies in PowerShell:

```
$content | wp post create - --post_status=publish --post_category=3 --post_author=2
```

`wp post create` and `wp post update` also accept a file path in place of `-`.

See also: [#2712](#)

The installation hangs

If the installation seems to hang forever while trying to clone the resources from GitHub, please ensure that you are allowed to connect to Github using SSL (port 443) and Git (port 9418) for outbound connections.

W3 Total Cache Error: some files appear to be missing or out of place.

W3 Total Cache object caching can cause this problem. Disabling object caching and optionally removing `wp-content/object-cache.php` will allow WP-CLI to work again.

See also: [#587](#)

The automated updater doesn't work for versions before 3.4

The `wp core update` command is designed to work for WordPress 3.4 and above. To be able to update an older website to latest WordPress, you could try one of the following alternatives:

1. **Fully-automated:** Run `wp core download --force` to download latest WordPress and replace it with your files (don't worry, `wp-config.php` will remain intact). Then, run `wp core update-db` to update the database. Since the procedure isn't ideal, run once again `wp core download --force` and the new version should be available.
2. **Semi-automated:** Run `wp core download --force` to download all files and replace them in your current installation, then navigate to `/wp-admin/` and run the database upgrade when prompted.

PHP Fatal error: Maximum function nesting level of '#' reached, aborting!

You're encountering a limitation associated with [PHP Xdebug](#). Here are a few solutions to the problem:

1. Temporarily turn off Xdebug while executing the CLI command: `XDEBUG_MODE=off wp <command> ...`
2. Increase the value of `xdebug.max_nesting_level` in your `php.ini` file. You can learn more about it here - [xdebug.max-nesting-level](#)

If you're looking for a quick fix while still retaining debugging tools, you can change the `max_nesting_level` value to execute the command just once:

```
php -d xdebug.max_nesting_level=512 wp <command> ...
```

If you're not in the process of debugging code, it's advisable to completely disable Xdebug. For instructions on how to permanently switch off Xdebug, please refer to the documentation of your local environment or your hosting provider.

You might experience similar issues with the alternate error message: `Error: Xdebug has detected a possible infinite loop, and aborted your script with a stack depth of '#' frames`

📄 #3

📄 cocobook-sync 📄 2026-10-28 10:58:37 UTC

📄 cocobook-sync 📄 2026-10-28 11:45:21 UTC