wp-config.php

- Configure Database Settings
  - Default wp-config-sample.php
    - Set Database Name
    - Set Database User
    - Set Database Password
    - Set Database Host
      - MySQL Alternate Port
      - MySQL Sockets or Pipes
  - Database character set
  - Database collation
  - Security Keys
- Advanced Options
  - table_prefix
  - WP_SITEURL
  - Blog address (URL)
  - Moving wp-content folder
  - Moving plugin folder
  - Moving themes folder
  - Moving uploads folder
  - Modify AutoSave Interval
  - Post Revisions
    - Disable Post Revisions
    - Specify the Number of Post Revisions
  - Set Cookie Domain
  - Enable Multisite / Network Ability
  - Redirect Nonexistent Blogs
  - Fatal Error Handler
  - WP_DEBUG

- WP_ENVIRONMENT_TYPE
- SCRIPT_DEBUG
- Disable Javascript Concatenation
- Configure Error Logging
- Increasing memory allocated to PHP
- Cache
- Custom User and Usermeta Tables
- Language and Language Directory
  - WordPress v3.9.6 and below
- Save queries for analysis
- Override of default file permissions
- WordPress Upgrade Constants
  - Enabling SSH Upgrade Access
- Alternative Cron
- Disable Cron and Cron Timeout
- Additional Defined Constants
- Empty Trash
- Automatic Database Optimizing
- DO_NOT_UPGRADE_GLOBAL_TABLES
- View All Defined Constants
- Disable the Plugin and Theme File Editor
- Disable Plugin and Theme Update and Installation
- Require SSL for Admin and Logins
- Block External URL Requests
- Disable WordPress Auto Updates
- Disable WordPress Core Updates
- Cleanup Image Edits

- Double Check Before Saving

One of the most important files in your WordPress installation is the `wp-config.php` file. This file is located in the root of your WordPress file directory and contains your website's base configuration details, such as database connection information.

Configure Database Settings

Important: *Never* use a word processor like Microsoft Word for editing WordPress files!

Locate the file `wp-config-sample.php` in the base directory of your WordPress directory and open in a text editor.

Default wp-config-sample.php

Note: This is an example of a default wp-config-sample.php. The values here are examples to show you what to do.

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'database_name_here' );
/** MySQL database username */
define( 'DB_USER', 'username_here' );
/** MySQL database password */
define( 'DB_PASSWORD', 'password_here' );
/** MySQL hostname */
define( 'DB_HOST', 'localhost' );
```

Note: Text inside /* */ are *comments*, for information purposes only.

Set Database Name

Replace ‘database_name_here’, with the name of your database, e.g. *MyDatabaseName*.

```
define( 'DB_NAME', 'MyDatabaseName' ); // Example MySQL database name
```

Set Database User

Replace ‘username_here’, with the name of your username e.g. *MyUserName*.

```
define( 'DB_USER', 'MyUserName' ); // Example MySQL username
```

Set Database Password

Replace ‘password_here’, with your password, e.g. *MyPassWord*.

```
define( 'DB_PASSWORD', 'MyPassWord' ); // Example MySQL password
```

Set Database Host

If needed, replace '*localhost*', with the name of your database host (e.g. *MyDatabaseHost*). A port number or Unix socket file path may be needed as well.

```
define( 'DB_HOST', 'MyDatabaseHost' ); // Example MySQL Database host
```

Note: There is a good chance you will **NOT** have to change it. If you are unsure, try installing with the default value of 'localhost' and see if it works. If the install fails, contact your web hosting provider.

MySQL Alternate Port

If your host uses an alternate port number for your database, you'll need to change the **DB_HOST** value in the `wp-config.php` file to reflect the alternate port provided by your host.

For localhost:

```
define( 'DB_HOST', '127.0.0.1:3307' );
or
define( 'DB_HOST', 'localhost:3307' );
```

For specified server:

```
define( 'DB_HOST', 'mysql.example.com:3307' );
```

Replace the number **3307** in either of the code examples above with the port number information provided by your host.

MySQL Sockets or Pipes

If your host uses Unix sockets or pipes, you'll need to change the **DB_HOST** value in the `wp-config.php` file to reflect the socket or pipe information provided by your host.

```
define( 'DB_HOST', '127.0.0.1:/var/run/mysqld/mysqld.sock' );
// or define( 'DB_HOST', 'localhost:/var/run/mysqld/mysqld.sock' );
// or define( 'DB_HOST', 'example.tld:/var/run/mysqld/mysqld.sock' );
```

Replace the text string **`/var/run/mysqld/mysqld.sock`** above with the socket or pipe information provided by your host.

Database character set

DB_CHARSET was made available to allow designation of the database character set (e.g. tis620 for TIS620 Thai) to be used when defining the MySQL database tables.

The default value of **utf8** (Unicode UTF-8) is almost always the best option. UTF-8 supports any language, so you typically want to leave DB_CHARSET at **utf8** and use the DB_COLLATE value for your language instead.

This example shows utf8 which is considered the WordPress default value:

```
define( 'DB_CHARSET', 'utf8' );
```

There usually should be no reason to change the default value of DB_CHARSET. If your blog needs a different character set, please read Character Sets and Collations MySQL Supports for valid DB_CHARSET values. **WARNING:** Those performing upgrades.

If DB_CHARSET and DB_COLLATE do not exist in your `wp-config.php` file, **DO NOT** add either definition to your `wp-config.php` file unless you read and understand Converting Database Character Sets. Adding DB_CHARSET and DB_COLLATE to the `wp-config.php` file, for an existing blog, can cause major problems.

Database collation

DB_COLLATE was made available to allow designation of the database collation (i.e. the sort order of the character set). In most cases, this value should be left blank (null) so the database collation will be automatically assigned by MySQL based on the database character set specified by DB_CHARSET. An example of when you may need to set '''DB_COLLATE''' to one of the UTF-8 values defined in UTF-8 character sets for most Western European languages would be when a different language in which the characters that you entered are not the same as what is being displayed. (See also Unicode Character Sets in SQL Manual)

The WordPress default DB_COLLATE value:

```
define( 'DB_COLLATE', '' );
```

UTF-8 Unicode General collation

```
define( 'DB_COLLATE', 'utf8_general_ci' );
```

UTF-8 Unicode Turkish collation

```
define( 'DB_COLLATE', 'utf8_turkish_ci' );
```

There usually should be no reason to change the default value of DB_COLLATE. Leaving the value blank (null) will ensure the collation is automatically assigned by MySQL when the database tables are created. **WARNING:** Those performing upgrades

If DB_COLLATE and DB_CHARSET do not exist in your `wp-config.php` file, **DO NOT** add either definition to your `wp-config.php` file unless you read and understand Converting Database Character Sets. And you may be in need of a WordPress upgrade.

Security Keys

You don’t have to remember the keys, just make them long, random and complicated — or better yet, use the online generator. You can change these at any point in time to invalidate all existing cookies. This does mean that all users will have to login again.

Example (don’t use these!):

```
define( 'AUTH_KEY',         't`DK%X:>xy|e-Z(BXb/f(Ur`8#~UzUQG-^_Cs_GHs5U-&Wb?pgn^p8(2@}IcnCa|'
);
define( 'SECURE_AUTH_KEY',  'D&ovlU#|CvJ##uNq}bel+^MFtT&.b9{UvR]g%ixsXhGlRJ7q!h}XWdEC[BOKXssj'
);
define( 'LOGGED_IN_KEY',    'MGKi8Br(&{H*~&0s;{k0<S(O:+f#WM+q|npJ-+P;RDKT:~jrmgj#/-,[hOBk!ry^'
);
define( 'NONCE_KEY',        'FIsAsXJKL5ZlQo)iD-pt??eUbdc{_Cn<4!d~yqz))&B D?AwK%)+)F2aNwI|siOe'
);
define( 'AUTH_SALT',        '7T-!^i!0,w)L#JK@pc2{8XE[DenYI^BVf{L:jvF,hf}zBf883td6D;Vcy8,S)-&G'
);
define( 'SECURE_AUTH_SALT', 'I6`V|mDZq21-J|ihb u^q0F }F_NUcy`l,=obGtq*p#Ybe4a31R,r=|n#=]@]c #'
);
define( 'LOGGED_IN_SALT',   'w<$4c$Hmd%/*]`Oom>(hdXW|0M=X={we6;Mpvtg+V.o<$|#_}qG(GaVDEsn,~*4i'
);
define( 'NONCE_SALT',       'a|#h{c5|P &xWs4IZ20c2&%4!c(/uG}W:mAvy<I44`jAbup]t=]V<`}.py(wTP%%'
);
```

A **secret key** makes your site harder to successfully attack by adding random elements to the password.

In simple terms, a secret key is a password with elements that make it harder to generate enough options to break through your security barriers. A password like “password” or “test” is simple and easily broken. A random, long password which uses no dictionary words, such as “88a7da62429ba6ad3cb3c76a09641fc” would take a brute force attacker millions of hours to crack. A ‘*salt* is used to further enhance the security of the generated result.

The four keys are required for the enhanced security. The four salts are recommended, but are not required, because WordPress will generate salts for you if none are provided. They are included in `wp-config.php` by default for inclusiveness.

For more information on the technical background and breakdown of secret keys and secure passwords, see:

- Wikipedia’s explanation of Password Cracking
- Lorelle VanFossen – Protect Your Blog With a Solid Password
- Instructables – Security Password Tips
- Huffington Post – 17 Tips You Can Do Today to Protect Your Online Passwords

Advanced Options

The following sections may contain advanced information and some changes might result in unforeseen issues. Please make sure you practice regular backups and know how to restore them before modifying these settings.

table_prefix

The **$table_prefix** is the value placed in the front of your database tables. Change the value if you want to use something other than **wp_** for your database prefix. Typically this is changed if you are installing multiple WordPress blogs in the same database, as is done with the multisite feature.

It is possible to have multiple installations in one database if you give each a unique prefix. Keep security in mind if you choose to do this.

```
$table_prefix = 'r235_'; // Only numbers, letters, and underscores please!
```

WP_SITEURL

WP_SITEURL allows the WordPress address (URL) to be defined. The value defined is the address where your WordPress core files reside. It should include the `http://` part too. Do not put a slash “/

” at the end. Setting this value in `wp-config.php` overrides the wp_options table value for **siteurl**. Adding this in can reduce the number of database calls when loading your site. **Note:** This will **not** change the database stored value. The URL will revert to the old database value if this line is ever removed from `wp-config`. Use the **RELOCATE** constant to change the **siteurl** value in the database.

If WordPress is installed into a directory called “wordpress” for the domain example.com, define WP_SITEURL like this:

```
define( 'WP_SITEURL', 'http://example.com/wordpress' );
```

Dynamically set WP_SITEURL based on $_SERVER[‘HTTP_HOST’]

```
define( 'WP_SITEURL', 'http://' . $_SERVER['HTTP_HOST'] . '/path/to/wordpress' );
```

Note: HTTP_HOST is created dynamically by PHP based on the value of the HTTP HOST Header in the request, thus possibly allowing for file inclusion vulnerabilities. SERVER_NAME may also be created dynamically. However, when Apache is configured as UseCanonicalName “on”, SERVER_NAME is set by the server configuration, instead of dynamically. In that case, it is safer to use SERVER_NAME than HTTP_HOST.

Dynamically set WP_SITEURL based on $_SERVER[‘SERVER_NAME’]

```
define( 'WP_SITEURL', 'http://' . $_SERVER['SERVER_NAME'] . '/path/to/wordpress' );
```

Blog address (URL)

Similar to WP_SITEURL, WP_HOME *overrides the wp_options table value for* home *but does not change it in the database.* **home** is the address you want people to type in their browser to reach your WordPress blog. It should include the `http://` part and should not have a slash “/” at the end. Adding this in can reduce the number of database calls when loading your site.

```
define( 'WP_HOME', 'http://example.com/wordpress' );
```

If you are using the technique described in Giving WordPress Its Own Directory then follow the example below. Remember, you will also be placing an `index.php` in your web-root directory if you use a setting like this.

```
define( 'WP_HOME', 'http://example.com' );
```

Dynamically set WP_HOME based on $_SERVER[‘HTTP_HOST’]

```
define( 'WP_HOME', 'http://' . $_SERVER['HTTP_HOST'] . '/path/to/wordpress' );
```

Moving wp-content folder

You can move the `wp-content` directory, which holds your themes, plugins, and uploads, outside of the WordPress application directory.

Set WP_CONTENT_DIR to the full **local path** of this directory (no trailing slash), e.g.

```
define( 'WP_CONTENT_DIR', dirname(__FILE__) . '/blog/wp-content' );
```

Set WP_CONTENT_URL to the full **URL** of this directory (no trailing slash), e.g.

```
define( 'WP_CONTENT_URL', 'http://example/blog/wp-content' );
```

Moving plugin folder

Set WP_PLUGIN_DIR to the full **local path** of this directory (no trailing slash), e.g.

```
define( 'WP_PLUGIN_DIR', dirname(__FILE__) . '/blog/wp-content/plugins' );
```

Set WP_PLUGIN_URL to the full **URI** of this directory (no trailing slash), e.g.

```
define( 'WP_PLUGIN_URL', 'http://example/blog/wp-content/plugins' );
```

If you have compatibility issues with plugins Set PLUGINDIR to the full **local path** of this directory (no trailing slash), e.g.

```
define( 'PLUGINDIR', dirname(__FILE__) . '/blog/wp-content/plugins' );
```

Moving themes folder

You cannot move the themes folder because its path is hardcoded relative to the `wp-content` folder:

$theme_root = WP_CONTENT_DIR . ‘/themes’;

However, you can register additional theme directories using register_theme_directory.

See how to move the wp-content folder. For more details how the themes folder is determined, see `wp-includes/theme.php`.

Moving uploads folder

Set UPLOADS to :

```
define( 'UPLOADS', 'blog/wp-content/uploads' );
```

This path can not be absolute. It is always relative to ABSPATH, therefore does not require a leading slash.

Modify AutoSave Interval

When editing a post, WordPress uses Ajax to auto-save revisions to the post as you edit. You may want to increase this setting for longer delays in between auto-saves, or decrease the setting to make sure you never lose changes. The default is 60 seconds.

```
define( 'AUTOSAVE_INTERVAL', 160 ); // Seconds
```

Post Revisions

WordPress, by default, will save copies of each edit made to a post or page, allowing the possibility of reverting to a previous version of that post or page. The saving of revisions can be disabled, or a maximum number of revisions per post or page can be specified.

Disable Post Revisions

If you do **not** set this value, WordPress defaults WP_POST_REVISIONS to *true* (enable post revisions). If you want to disable the awesome revisions feature, use this setting:

```
define( 'WP_POST_REVISIONS', false );
```

Note: Some users could not get this to function until moving the command to the first line under the initial block comment in `wp-config.php`.

Specify the Number of Post Revisions

If you want to specify a maximum number of revisions that WordPress stores, change *false* to an integer/number (*e.g.*, 3 or 12).

```
define( 'WP_POST_REVISIONS', 3 );
```

Note: Some users could not get this to function until moving the command to the first line under the initial block comment in `wp-config.php`.

Set Cookie Domain

The domain set in the cookies for WordPress can be specified for those with unusual domain setups. For example, if subdomains are used to serve static content, you can set the cookie domain to only your non-static domain to prevent WordPress cookies from being sent with each request to static content on your subdomain .

```
define( 'COOKIE_DOMAIN', 'www.example.com' );
```

Enable Multisite / Network Ability

WP_ALLOW_MULTISITE can be used to enable multisite functionality. If this setting is absent from `wp-config.php`, it defaults to false.

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

Redirect Nonexistent Blogs

NOBLOGREDIRECT can be used to redirect the browser if the visitor tries to access a nonexistent subdomain or a subfolder.

```
define( 'NOBLOGREDIRECT', 'http://example.com' );
```

Fatal Error Handler

WordPress 5.2 introduced Recovery Mode which displays error message instead of white screen when plugins causes fatal error.

The site is experiencing technical difficulties. Please check your site admin email inbox for instructions.

White screens and PHP error messages are not displayed to users any more. But in a development environment, if you want to enable WP_DEBUG_DISPLAY, you have to disable recovery mode by set true to WP_DISABLE_FATAL_ERROR_HANDLER.

```
define( 'WP_DISABLE_FATAL_ERROR_HANDLER', true );   // 5.2 and later
define( 'WP_DEBUG', true );
define( 'WP_DEBUG_DISPLAY', true );
```

WP_DEBUG

The WP_DEBUG option controls the reporting of some errors and warnings and enables use of the WP_DEBUG_DISPLAY and WP_DEBUG_LOG settings. The default boolean value is false.

```
define( 'WP_DISABLE_FATAL_ERROR_HANDLER', true );   // 5.2 and later
define( 'WP_DEBUG', true );
```

Database errors are printed only if WP_DEBUG is set to true. Database errors are handled by the wpdb class and are not affected by PHP's error settings.

Setting WP_DEBUG to true also raises the error reporting level to E_ALL and activates warnings when deprecated functions or files are used; otherwise, WordPress sets the error reporting level to E_ALL ^ E_NOTICE ^ E_USER_NOTICE.

WP_ENVIRONMENT_TYPE

The WP_ENVIRONMENT_TYPE option controls the environment type for a site.

WordPress supports four environment types:

- `production` (default): a live site connected to and reachable via the internet,
- `staging`: a non-production staging environment connected to and reachable via the the internet
- `development`: a non-production development environment connected to and reachable via the internet
- `local`: a non-production development environment that can reach the internet but is **not** reachable via the internet

`WP_DEBUG` is automatically enabled for sites set to either the `development` or `local` environment types.

For environments reachable via the internet the Application Passwords feature requires the site be served over a secure HTTPS connection. For `local` environments the Application Passwords feature will work over insecure HTTP connections.

The values of environment types are processed in the following order with each sequential method overriding any previous values: the WP_ENVIRONMENT_TYPE PHP environment variable and the WP_ENVIRONMENT_TYPE constant.

For both methods, if the value of an environment type provided is not in the list of allowed environment types, the default `production` value will be returned.

The simplest way to set the value is probably through defining the constant:

```
define( 'WP_ENVIRONMENT_TYPE', 'staging' );
```

SCRIPT_DEBUG

SCRIPT_DEBUG is a related constant that will force WordPress to use the “dev” versions of scripts and stylesheets in `wp-includes/js`, `wp-includes/css`, `wp-admin/js`, and `wp-admin/css` will be loaded instead of the `.min.css` and `.min.js` versions.. If you are planning on modifying some of WordPress’ built-in JavaScript or Cascading Style Sheets, you should add the following code to your config file:

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

Disable Javascript Concatenation

To result in faster administration screens, all JavaScript files are concatenated into one URL. If JavaScript is failing to work in an administration screen, you can try disabling this feature:

```
define( 'CONCATENATE_SCRIPTS', false );
```

Configure Error Logging

Configuring error logging can be a bit tricky. First of all, default PHP error log and display settings are set in the php.ini file, which you may or may not have access to. If you do, they should be set to the desired settings for live PHP pages served to the public. It’s strongly recommended that no error messages are displayed to the public and instead routed to an error log. Furthermore, error logs should not be located in the publicly accessible portion of your server. Sample recommended php.ini error settings:

```
error_reporting = 4339
display_errors = Off
```

```
display_startup_errors = Off
log_errors = On
error_log = /home/example.com/logs/php_error.log
log_errors_max_len = 1024
ignore_repeated_errors = On
ignore_repeated_source = Off
html_errors = Off
```

About Error Reporting 4339 This is a custom value that only logs issues that affect the functioning of your site, and ignores things like notices that may not even be errors. See PHP Error Constants for the meaning of each binary position for 1000011110011, which is the binary number equal to 4339. The far left 1 means report any E_RECOVERABLE_ERROR. The next 0 means do not report E_STRICT, (which is thrown when sloppy but functional coding is used) and so on. Feel free to determine your own custom error reporting number to use in place of 4339.

Obviously, you will want different settings for your development environment. If your staging copy is on the same server, or you don't have access to `php.ini`, you will need to override the default settings at run time. It's a matter of personal preference whether you prefer errors to go to a log file, or you prefer to be notified immediately of any error, or perhaps both. Here's an example that reports all errors immediately that you could insert into your `wp-config.php` file:

```
@ini_set( 'log_errors', 'Off' );
@ini_set( 'display_errors', 'On' );
define( 'WP_DISABLE_FATAL_ERROR_HANDLER', true );   // 5.2 and later
define( 'WP_DEBUG', true );
define( 'WP_DEBUG_LOG', false );
define( 'WP_DEBUG_DISPLAY', true );
```

Because `wp-config.php` is loaded for every page view not loaded from a cache file, it is an excellent location to set `php.ini` settings that control your PHP installation. This is useful if you don't have access to a `php.ini` file, or if you just want to change some settings on the fly. One exception is 'error_reporting'. When WP_DEBUG is defined as true, 'error_reporting' will be set to E_ALL by WordPress regardless of anything you try to set in wp-config.php. If you really have a need to set 'error_reporting' to something else, it must be done after `wp-settings.php` is loaded, such as in a plugin file.

If you turn on error logging, remember to delete the file afterwards, as it will often be in a publicly accessible location, where anyone could gain access to your log.

Here is an example that turns PHP error_logging on and logs them to a specific file. If WP_DEBUG is defined to true, the errors will also be saved to this file. Just place this above any *require_once* or *include* commands.

```
@ini_set( 'log_errors', 'On' );
@ini_set( 'display_errors', 'Off' );
@ini_set( 'error_log', '/home/example.com/logs/php_error.log' );
/* That's all, stop editing! Happy blogging. */
```

Another example of logging errors, as suggested by Mike Little on the wp-hackers email list:

```
/**
 * This will log all errors notices and warnings to a file called debug.log in
 * wp-content (if Apache does not have write permission, you may need to create
 * the file first and set the appropriate permissions (i.e. use 666) )
 */
define( 'WP_DEBUG', true );
define( 'WP_DEBUG_LOG', true );
define( 'WP_DEBUG_DISPLAY', false );
@ini_set( 'display_errors', 0 );
```

A refined version from Mike Little on the Manchester WordPress User Group:

```
/**
 * This will log all errors notices and warnings to a file called debug.log in
 * wp-content only when WP_DEBUG is true. if Apache does not have write permission,
 * you may need to create the file first and set the appropriate permissions (i.e. use 666).
 */
define( 'WP_DEBUG', true ); // Or false
if ( WP_DEBUG ) {
    define( 'WP_DEBUG_LOG', true );
    define( 'WP_DEBUG_DISPLAY', false );
    @ini_set( 'display_errors', 0 );
}
```

Confusing the issue is that WordPress has three (3) constants that look like they could do the same thing. First off, remember that if WP_DEBUG is false, it and the other two WordPress DEBUG constants do not do anything. The PHP directives, whatever they are, will prevail. Except for 'error_reporting', WordPress will set this to 4983 if WP_DEBUG is defined as false. Second, even if WP_DEBUG is true, the other constants only do something if they too are set to true. If they are set to false, the PHP directives remain unchanged. For example, if your `php.ini` file has the directive ('display_errors' = 'On'); but you have the statement define('WP_DEBUG_DISPLAY', false); in your `wp-config.php` file, errors will still be displayed on screen even though you tried to prevent it by setting WP_DEBUG_DISPLAY to false because that is the PHP configured behavior. This is why it's very important to set the PHP directives to what you need in case any of the related WP constants

are set to false. To be safe, explicitly set/define both types. More detailed descriptions of the WP constants is available at Debugging in WordPress.

For your public, production WordPress installation, you might consider placing the following in your `wp-config.php` file, even though it may be partly redundant:

```
@ini_set( 'log_errors', 'On' );
@ini_set( 'display_errors', 'Off' );
define( 'WP_DISABLE_FATAL_ERROR_HANDLER', false );   // 5.2 and later
define( 'WP_DEBUG', false );
define( 'WP_DEBUG_LOG', false );
define( 'WP_DEBUG_DISPLAY', false );
```

The default debug log file is `/wp-content/debug.log`. Placing error logs in publicly accessible locations is a security risk. Ideally, your log files should be placed above your site's public root directory. If you can't do this, at the very least, set the log file permissions to 600 and add this entry to the `.htaccess` file in the root directory of your WordPress installation:

```
<Files debug.log>
    Order allow,deny
    Deny from all
</Files>
```

This prevents anyone from accessing the file via HTTP. You can always view the log file by retrieving it from your server via FTP.

Increasing memory allocated to PHP

WP_MEMORY_LIMIT option allows you to specify the maximum amount of memory that can be consumed by PHP. This setting may be necessary in the event you receive a message such as "Allowed memory size of xxxxxx bytes exhausted".

This setting increases PHP Memory only for WordPress, not other applications. By default, WordPress will attempt to increase memory allocated to PHP to 40MB (code is at the beginning of `/wp-includes/default-constants.php`) for single site and 64MB for multisite, so the setting in `wp-config.php` should reflect something higher than 40MB or 64MB depending on your setup.

WordPress will automatically check if PHP has been allocated less memory than the entered value before utilizing this function. For example, if PHP has been allocated 64MB, there is no need to set this value to 64M as WordPress will automatically use all 64MB if need be.

Note: Some hosts do not allow for increasing the PHP memory limit automatically. In that event, contact your host to increase the PHP memory limit. Also, many hosts set the PHP limit at 8MB.

Adjusting the WordPress memory limit potentially creates problems as well. You might end up hiding the root of the issue for it to happen later down the line as you add in more plugins or functionalities.

If you are facing Out of Memory issues even with an elevated memory limit, you should properly debug your installation. Chances are you have too many memory intensive functions tied to a specific action and should move these functions to a cronjob.

Increase PHP Memory to 64MB

```
define( 'WP_MEMORY_LIMIT', '64M' );
```

Increase PHP Memory to 96MB

```
define( 'WP_MEMORY_LIMIT', '96M' );
```

Administration tasks require may require more memory than usual operation. When in the administration area, the memory can be increased or decreased from the WP_MEMORY_LIMIT by defining WP_MAX_MEMORY_LIMIT. The default value for this is 256MB.

```
define( 'WP_MAX_MEMORY_LIMIT', '512M' );
```

Note: this has to be put before wp-settings.php inclusion.

Cache

The **WP_CACHE** setting, if true, includes the `wp-content/advanced-cache.php` script, when executing `wp-settings.php`.

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

Custom User and Usermeta Tables

CUSTOM_USER_TABLE and **CUSTOM_USER_META_TABLE** are used to designate that the user and usermeta tables normally utilized by WordPress are not used, instead these values/tables are used to store your user information.

```
define( 'CUSTOM_USER_TABLE', $table_prefix.'my_users' );
define( 'CUSTOM_USER_META_TABLE', $table_prefix.'my_usermeta' );
```

Note: Even if ‘CUSTOM_USER_META_TABLE’ is manually set, a usermeta table is still created for each database with the corresponding permissions for each instance. By default, the WordPress

installer will add permissions for the first user (ID #1). You also need to manage permissions to each of the site via a plugin or custom function. If this isn't setup you will experience permission errors and log-in issues.

CUSTOM_USER_TABLE is easiest to adopt during initial Setup your first instance of WordPress. The define statements of the `wp-config.php` on the first instance point to where `wp_users` data will be stored by default. After the first site setup, copying the working `wp-config.php` to your next instance will only require a change the `$table_prefix` variable. Do not use an e-mail address that is already in use by your original install. Once you have finished the setup process log in with the auto generated admin account and password. Next, promote your normal account to the administrator level and Log out of admin. Log back in as yourself, delete the admin account and promote the other user accounts as is needed.

Language and Language Directory

WordPress Version 4.0 allows you to change the language in your WordPress Administration Screens. To change the language in the admin settings screen. Go to Settings > General and select Site Language.

WordPress v3.9.6 and below

WPLANG defines the name of the language translation (.mo) file. **WP_LANG_DIR** defines what directory the WPLANG .mo file resides. If WP_LANG_DIR is not defined WordPress looks first to wp-content/languages and then `wp-includes/languages` for the .mo defined by WPLANG file.

```
define( 'WPLANG', 'de_DE' );
define( 'WP_LANG_DIR', dirname(__FILE__) . 'wordpress/languages' );
```

To find out the WPLANG language code, please refer here. The code in WP Local column is what you need.

Save queries for analysis

The **SAVEQUERIES** definition saves the database queries to an array and that array can be displayed to help analyze those queries. The information saves each query, what function called it, and how long that query took to execute. **Note:** This will have a performance impact on your site, so make sure to turn this off when you aren't debugging.

First, add this to the `wp-config.php` file:

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

Then in the footer of your theme put this:

```
if ( current_user_can( 'administrator' ) ) {
    global $wpdb;
    echo "<pre>";
    print_r( $wpdb->queries );
    echo "</pre>";
}
?>
```

Alternatively, consider using Query Monitor

Override of default file permissions

The **FS_CHMOD_DIR** and **FS_CHMOD_FILE** define statements allow override of default file permissions. These two variables were developed in response to the problem of the core update function failing with hosts running under suexec. If a host uses restrictive file permissions (e.g. 400) for all user files, and refuses to access files which have group or world permissions set, these definitions could solve the problem.

```
define( 'FS_CHMOD_DIR', ( 0755 & ~ umask() ) );
define( 'FS_CHMOD_FILE', ( 0644 & ~ umask() ) );
```

Example to provide setgid:

```
define( 'FS_CHMOD_DIR', ( 02755 & ~umask() ) );
```

Note: '**0755**' and '**02755**' are octal values. Octal values must be prefixed with a 0 and are not delineated with single quotes ('). See Also: Changing File Permissions

WordPress Upgrade Constants

Note: Define as few of the below constants as needed to correct your update issues.

The most common causes of needing to define these are:

Host running with a special installation setup involving symlinks. You may need to define the path-related constants (FTP_BASE, FTP_CONTENT_DIR, and FTP_PLUGIN_DIR). Often defining simply the base will be enough.

Certain PHP installations shipped with a PHP FTP extension which is incompatible with certain FTP servers. Under these rare situations, you may need to define FS_METHOD to “ftpsockets”.

The following are valid constants for WordPress updates:

- **FS_METHOD** forces the filesystem method. It should only be “direct”, “ssh2”, “ftpext”, or “ftpsockets”. Generally, you should only change this if you are experiencing update problems. If you change it and it doesn’t help, **change it back/remove it**. Under most circumstances, setting it to ‘ftpsockets’ will work if the automatically chosen method does not.
  - **(Primary Preference) “direct”** forces it to use Direct File I/O requests from within PHP, this is fraught with opening up security issues on poorly configured hosts, This is chosen automatically when appropriate.
  - **(Secondary Preference) “ssh2”** is to force the usage of the SSH PHP Extension if installed
  - **(3rd Preference) “ftpext”** is to force the usage of the FTP PHP Extension for FTP Access, and finally
  - **(4th Preference) “ftpsockets”** utilises the PHP Sockets Class for FTP Access.
- **FTP_BASE** is the full path to the “base”(ABSPATH) folder of the WordPress installation.
- **FTP_CONTENT_DIR** is the full path to the wp-content folder of the WordPress installation.
- **FTP_PLUGIN_DIR** is the full path to the plugins folder of the WordPress installation.
- **FTP_PUBKEY** is the full path to your SSH public key.
- **FTP_PRIKEY** is the full path to your SSH private key.
- **FTP_USER** is either user FTP or SSH username. Most likely these are the same, but use the appropriate one for the type of update you wish to do.
- **FTP_PASS** is the password for the username entered for **FTP_USER**. If you are using SSH public key authentication this can be omitted.
- **FTP_HOST** is the hostname:port combination for your SSH/FTP server. The default FTP port is 21 and the default SSH port is 22. These do not need to be mentioned.
- **FTP_SSL** TRUE for SSL-connection *if supported by the underlying transport* (not available on all servers). This is for “Secure FTP” not for SSH SFTP.

```
define( 'FS_METHOD', 'ftpext' );
define( 'FTP_BASE', '/path/to/wordpress/' );
define( 'FTP_CONTENT_DIR', '/path/to/wordpress/wp-content/' );
define( 'FTP_PLUGIN_DIR', '/path/to/wordpress/wp-content/plugins/' );
define( 'FTP_PUBKEY', '/home/username/.ssh/id_rsa.pub' );
define( 'FTP_PRIKEY', '/home/username/.ssh/id_rsa' );
define( 'FTP_USER', 'username' );
define( 'FTP_PASS', 'password' );
define( 'FTP_HOST', 'ftp.example.org' );
define( 'FTP_SSL', false );
```

Some configurations should set FTP_HOST to localhost to avoid 503 problems when trying to update plugins or WP itself.

Enabling SSH Upgrade Access

There are two ways to upgrade using SSH2.

The first is to use the SSH SFTP Updater Support plugin. The second is to use the built-in SSH2 upgrader, which requires the pecl SSH2 extension be installed.

To install the pecl SSH2 extension you will need to issue a command similar to the following or talk to your web hosting provider to get this installed:

```
pecl install ssh2
```

After installing the pecl ssh2 extension you will need to modify your PHP configuration to automatically load this extension.

pecl is provided by the pear package in most linux distributions. To install pecl in Redhat/Fedora/CentOS:

```
yum -y install php-pear
```

To install pecl in Debian/Ubuntu:

```
apt-get install php-pear
```

It is recommended to use a private key that is not pass-phrase protected. There have been numerous reports that pass phrase protected private keys do not work properly. If you decide to try a pass phrase protected private key you will need to enter the pass phrase for the private key as FTP_PASS, or entering it in the “Password” field in the presented credential field when installing updates.

Alternative Cron

There might be reason to use an alternative Cron with WP. Most commonly this is done if scheduled posts are not getting published as predicted. This alternative method uses a redirection approach. The users’ browser get a redirect when the cron needs to run, so that they come back to the site immediately while cron continues to run in the connection they just dropped. This method has certain risks, since it depends on a non-native WordPress service.

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

Disable Cron and Cron Timeout

Disable cron entirely by setting DISABLE_WP_CRON to true.

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

Make sure a cron process cannot run more than once every WP_CRON_LOCK_TIMEOUT seconds.

```
define( 'WP_CRON_LOCK_TIMEOUT', 60 );
```

Additional Defined Constants

Here are additional constants that can be defined. These probably shouldn't be set unless other methodologies have been attempted first. The Cookie definitions can be particularly useful if you have an unusual domain setup.

```
define( 'COOKIEPATH', preg_replace( '|https?://[^/]+|i', '', get_option( 'home' ) . '/' ) );
define( 'SITECOOKIEPATH', preg_replace( '|https?://[^/]+|i', '', get_option( 'siteurl' ) . '/'
) );
define( 'ADMIN_COOKIE_PATH', SITECOOKIEPATH . 'wp-admin' );
define( 'PLUGINS_COOKIE_PATH', preg_replace( '|https?://[^/]+|i', '', WP_PLUGIN_URL ) );
define( 'TEMPLATEPATH', get_template_directory() );
define( 'STYLESHEETPATH', get_stylesheet_directory() );
```

Empty Trash

This constant controls the number of days before WordPress permanently deletes posts, pages, attachments, and comments, from the trash bin. The default is 30 days:

```
define( 'EMPTY_TRASH_DAYS', 30 ); // 30 days
```

To disable trash set the number of days to zero.

```
define( 'EMPTY_TRASH_DAYS', 0 ); // Zero days
```

Note: WordPress will not ask for confirmation when someone clicks on "Delete Permanently" using this setting.

Automatic Database Optimizing

There is automatic database repair support, which you can enable by adding the following define to your `wp-config.php` file.

Note: This should only be enabled if needed and disabled once the issue is solved. When enabled, a user does not need to be logged in to access the functionality, since its main intent is to repair a corrupted database and users can often not login when the database is corrupt.

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

The script can be found at `{$your_site}/wp-admin/maint/repair.php`.

DO_NOT_UPGRADE_GLOBAL_TABLES

A **DO_NOT_UPGRADE_GLOBAL_TABLES** define prevents dbDelta() and the upgrade functions from doing expensive queries against global tables.

Sites that have large global tables (particularly users and usermeta), as well as sites that share user tables with bbPress and other WordPress installs, can prevent the upgrade from changing those tables during upgrade by defining **DO_NOT_UPGRADE_GLOBAL_TABLES** to true. Since an ALTER, or an unbounded DELETE or UPDATE, can take a long time to complete, large sites usually want to avoid these being run as part of the upgrade so they can handle it themselves. Further, if installations are sharing user tables between multiple bbPress and WordPress installs you may to want one site to be the upgrade master.

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

View All Defined Constants

PHP has a function that returns an array of all the currently defined constants with their values.

```
print_r( @get_defined_constants() );
```

Disable the Plugin and Theme File Editor

Occasionally you may wish to disable the plugin or theme file editor to prevent overzealous users from being able to edit sensitive files and potentially crash the site. Disabling these also provides an additional layer of security if a hacker gains access to a well-privileged user account.

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

Note: The functionality of some plugins may be affected by the use of `current_user_can('edit_plugins')` in their code. Plugin authors should avoid checking for this capability, or at least check if this constant is set and display an appropriate error message. Be aware that if a plugin is not working this may be the cause.

Disable Plugin and Theme Update and Installation

This will block users being able to use the plugin and theme installation/update functionality from the WordPress admin area. Setting this constant also disables the Plugin and Theme File editor (i.e. you don't need to set DISALLOW_FILE_MODS and DISALLOW_FILE_EDIT, as on its own DISALLOW_FILE_MODS will have the same effect).

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

Require SSL for Admin and Logins

Note: WordPress Version 4.0 deprecated FORCE_SSL_LOGIN. Please use FORCE_SSL_ADMIN.

FORCE_SSL_ADMIN is for when you want to secure logins and the admin area so that both passwords and cookies are never sent in the clear. See also Administration_Over_SSL for more details.

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

Block External URL Requests

Block external URL requests by defining WP_HTTP_BLOCK_EXTERNAL as true and this will only allow localhost and your blog to make requests. The constant WP_ACCESSIBLE_HOSTS will allow additional hosts to go through for requests. The format of the WP_ACCESSIBLE_HOSTS constant is a comma separated list of hostnames to allow, wildcard domains are supported, eg *.wordpress.org will allow for all subdomains of wordpress.org to be contacted.

```
define( 'WP_HTTP_BLOCK_EXTERNAL', true );
define( 'WP_ACCESSIBLE_HOSTS', 'api.wordpress.org,*.github.com' );
```

Disable WordPress Auto Updates

There might be reason for a site to not auto-update, such as customizations or host supplied updates. It can also be done before a major release to allow time for testing on a development or staging environment before allowing the update on a production site.

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

Disable WordPress Core Updates

The easiest way to manipulate core updates is with the WP_AUTO_UPDATE_CORE constant:

```
# Disable all core updates:
define( 'WP_AUTO_UPDATE_CORE', false );
# Enable all core updates, including minor and major:
define( 'WP_AUTO_UPDATE_CORE', true );
# Enable core updates for minor releases (default):
define( 'WP_AUTO_UPDATE_CORE', 'minor' );
```

Reference: Disabling Auto Updates in WordPress 3.7

Cleanup Image Edits

By default, WordPress creates a new set of images every time you edit an image and when you restore the original, it leaves all the edits on the server. Defining IMAGE_EDIT_OVERWRITE as true changes this behaviour. Only one set of image edits are ever created and when you restore the original, the edits are removed from the server.

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

Double Check Before Saving

Be sure to check for leading and/or trailing spaces around any of the above values you entered, and DON’T delete the single quotes!

Before you save the file, be sure to **double-check** that you have not accidentally deleted any of the single quotes around the parameter values.

To save the file, choose **File > Save As > wp-config.php** and save the file in the root of your WordPress install. Upload the file to your web server and you’re ready to install WordPress!

□□□ #2
□ □□□ □□□ 2026-10-31 18:07:21 UTC
□ □□□ □□□ 2026-10-31 18:07:24 UTC