PHP Cheatsheet

By Dejan Panovski • Updated on Sep 1, 2026 •

PHP 8 quick reference for syntax, arrays, functions, classes, request data, files, JSON, exceptions, CLI commands, Composer, configuration, and PHP-FPM.

Use this PHP 8 cheatsheet while writing application code or managing PHP on Linux. It pairs everyday language syntax with the CLI, Composer, configuration, and PHP-FPM commands.

Basic Syntax

Every PHP file starts with an opening tag. Closing tags are omitted in pure PHP files to avoid stray output.

`<?php ... ?>`	Standard PHP tags
`<?= $name ?>`	Short echo tag (always available)
`declare(strict_types=1);`	Use strict scalar checks for calls and returns in this file
`// comment, # comment`	Single-line comment
`/* comment */`	Multi-line comment
`echo "text";`	Output one or more strings
`print_r($var)`	Human-readable dump of an array or object
`var_dump($var)`	Dump value with type and length
`require 'file.php'`	Include file, fatal error if missing
`include 'file.php'`	Include file, warning if missing
`require_once 'file.php'`	Include only once

Variables and Types

PHP variables start with `$` and are dynamically typed. Type declarations are optional but recommended.

`$name = "Alice";`	Assign a value
`int, float, bool, string`	Scalar types
`array, object, callable, iterable`	Compound types
`null`	Absence of a value
`const MAX = 10;`	Compile-time constant
`define('MAX', 10);`	Runtime constant
`gettype($x)`	Return the type name
`is_int($x), is_string($x), is_array($x)`	Type checks
`(int) $x, (string) $x, (bool) $x`	Explicit casts
`intval($x), floatval($x)`	Convert to int or float
`isset($x)`	`true` if set and not `null`
`empty($x)`	`true` when unset or equal to `""`, `"0"`, `0`, `0.0`, `[]`, `null`, or `false`
`unset($x)`	Destroy a variable
`?int $x`	Nullable type (int or null)
`int|string $x`	Union type

Operators

Arithmetic, comparison, logical, and null-handling operators.

`+, -, *, /, %`	Arithmetic and modulo
`**`	Exponentiation
`.`	String concatenation
`.=, +=, -=, *=`	Compound assignment
`==, !=`	Loose comparison (type juggling)
`===, !==`	Strict comparison (value and type)
`<, >, <=, >=`	Relational comparison
`<=>`	Spaceship, returns -1, 0, or 1
`&&, ||, !`	Logical AND, OR, NOT
`and, or, xor`	Low-precedence logical operators
`? :`	Ternary conditional
`?:`	Elvis, returns left side if truthy
`??`	Null coalescing, returns right side if left is null
`??=`	Null coalescing assignment
`?->`	Nullsafe method or property access
`|>`	Pipe operator, PHP 8.5 and later

Strings

Double-quoted strings interpolate variables; single-quoted strings do not.

`strlen($s)`	String length in bytes
`mb_strlen($s)`	Character length (requires the mbstring extension)
`strtolower($s)` / `strtoupper($s)`	Change case
`ucfirst($s)` / `ucwords($s)`	Capitalize first letter / each word
`trim($s), ltrim($s), rtrim($s)`	Strip whitespace
`str_contains($s, "x")`	`true` if substring is present
`str_starts_with($s, "x")`	`true` if string starts with prefix
`str_ends_with($s, "x")`	`true` if string ends with suffix
`strpos($s, "x")`	First index of substring, `false` if absent
`substr($s, 0, 5)`	Extract part of a string
`str_replace("a", "b", $s)`	Replace all occurrences
`explode(",", $s)`	Split into an array
`implode(", ", $arr)`	Join array elements
`sprintf("%s has %d", $a, $b)`	Format into a string
`number_format(1234.5, 2)`	Format a number with separators
`str_pad($s, 10, "0", STR_PAD_LEFT)`	Pad to a fixed width
`str_repeat($s, 3)`	Repeat a string
`htmlspecialchars($s)`	Escape HTML before output
`nl2br($s)`	Convert newlines to ` `
`"Hello $name" / "Sum: {$a['b']}"`	Interpolation, braces for complex expressions
`<<<EOT ... EOT;`	Heredoc (interpolates) and nowdoc `<<<'EOT'` (does not)

Arrays

PHP arrays are ordered maps and cover both lists and dictionaries.

`$a = [1, 2, 3];`	Indexed array
`$a = ["k" => "v"];`	Associative array
`$a[] = 4;`	Append an element
`$a["k"]`	Access by key
`$a[0][1]`	Nested access
`count($a)`	Number of elements
`array_key_exists("k", $a)`	`true` if the key exists, even when null
`in_array(4, $a)`	`true` if the value exists
`array_search(4, $a)`	Return the key of a value
`[$x, $y] = $a;`	Destructuring assignment
`["k" => $v] = $a;`	Destructure by key
`[...$a, ...$b]`	Spread into a new array
`foreach ($a as $k => $v)`	Iterate keys and values

Array Functions

Transform, filter, and sort arrays without writing loops.

`array_push($a, $v)` / `array_pop($a)`	Add or remove at the end
`array_unshift($a, $v)` / `array_shift($a)`	Add or remove at the start
`array_merge($a, $b)`	Merge arrays, reindex numeric keys
`array_keys($a)` / `array_values($a)`	Extract keys or values
`array_slice($a, 1, 3)`	Extract a portion
`array_splice($a, 1, 2)`	Remove or replace a portion in place
`array_map(fn($x) => $x * 2, $a)`	Apply a callback to each element
`array_filter($a, fn($x) => $x > 2)`	Keep elements passing a test
`array_reduce($a, fn($c, $x) => $c + $x, 0)`	Reduce to a single value
`array_column($rows, "name", "id")`	Pull one column, optionally keyed
`array_unique($a)`	Remove duplicate values
`array_combine($keys, $vals)`	Build an array from two arrays
`array_flip($a)`	Swap keys and values
`array_sum($a)` / `array_product($a)`	Sum or product of values
`min($a)` / `max($a)`	Smallest or largest value
`range(1, 10)`	Build a sequence
`sort($a)` / `rsort($a)`	Sort values ascending / descending
`asort($a)` / `ksort($a)`	Sort by value / key, keep keys
`usort($a, fn($x, $y) => $x <=> $y)`	Sort with a custom comparator
`array_find($a, $fn)`, `array_any($a, $fn)`, `array_all($a, $fn)`	Search and test, PHP 8.4 and later
`array_first($a)` / `array_last($a)`	First or last value, PHP 8.5 and later

Control Flow

Conditionals and loops. The alternative syntax with `endif` and `endforeach` reads better inside HTML templates.

`if (...) { } elseif (...) { } else { }`	Standard branching
`if (...): ... endif;`	Alternative syntax for templates
`switch ($x) { case 1: ...; break; default: ...; }`	Multi-branch on loose comparison
`match($x) { 1 => "a", default => "b" }`	Strict comparison, returns a value
`for ($i = 0; $i < 10; $i++)`	Counter loop
`foreach ($arr as $value)`	Iterate values
`foreach ($arr as $key => $value)`	Iterate keys and values
`foreach ($arr as &$value)`	Iterate by reference (unset after)
`while (cond) / do { } while (cond);`	Conditional loops
`break; / break 2;`	Exit the loop, or two levels of loops
`continue;`	Skip to the next iteration
`return $x;`	Return from a function

Functions

Functions support default values, type declarations, named arguments, and variadics.

`function name($a, $b) { }`	Function declaration
`function name(int $a): string { }`	Typed parameters and return type
`function name($a = 10) { }`	Default parameter value
`function name(...$args) { }`	Variadic parameters
`name(b: 2, a: 1)`	Named arguments
`function name(&$a) { }`	Pass by reference
`function (): void { }`	No return value
`fn($x) => $x * 2`	Arrow function, captures scope automatically
`function ($x) use ($y) { }`	Closure with an explicit captured variable
`$fn = strlen(...);`	First-class callable syntax
`call_user_func($fn, $arg)`	Call a callable dynamically
`function gen() { yield $x; }`	Generator function
`static function () { }`	Closure without `$this` binding

Classes and Objects

PHP 8 adds constructor promotion, enums, readonly properties, and property hooks.

`class User { }`	Class declaration
`new User()`	Instantiate
`public, protected, private`	Visibility modifiers
`public function __construct(private string $name) { }`	Constructor property promotion
`public readonly int $id;`	Write once, then immutable
`private(set) string $name;`	Asymmetric visibility, PHP 8.4 and later
`public string $full { get => "$this->a $this->b"; }`	Property hook, PHP 8.4 and later
`$this->name`	Access a property on the instance
`self::CONST, static::method()`	Class and late static binding access
`parent::__construct()`	Call the parent constructor
`class Admin extends User { }`	Inheritance
`interface Jsonable { } / implements Jsonable`	Interfaces
`abstract class Base { }`	Abstract class
`trait Loggable { } / use Loggable;`	Trait reuse
`enum Status: string { case Active = 'active'; }`	Backed enum
`Status::from('active'), Status::tryFrom($x)`	Enum lookup, `tryFrom` returns null
`$obj instanceof User`	Type check
`User::class`	Fully qualified class name as a string
`__get, __set, __call, __toString`	Magic methods
`namespace App\Models; / use App\Models\User;`	Namespaces and imports

Superglobals and Request Data

Superglobals are available in every scope. Treat all of them as untrusted input.

`$_GET`	Query string parameters
`$_POST`	Form body parameters
`$_REQUEST`	Merge of GET, POST, and cookies
`$_SERVER`	Request and server metadata
`$_SERVER['REQUEST_METHOD']`	HTTP method
`$_SERVER['REMOTE_ADDR']`	Client IP address
`$_FILES`	Uploaded file metadata
`$_COOKIE`	Cookies sent by the client
`$_SESSION`	Session data, after `session_start()`
`$_ENV, getenv('NAME')`	Environment variables
`filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT)`	Read and validate in one call
`filter_var($email, FILTER_VALIDATE_EMAIL)`	Validate a value
`htmlspecialchars($v, ENT_QUOTES)`	Escape before printing to HTML
`header('Location: /home'); exit;`	Redirect and stop further execution
`http_response_code(404)`	Set the response status
`password_hash($p, PASSWORD_DEFAULT)`	Hash a password
`password_verify($p, $hash)`	Verify a password against a hash

Files and JSON

File helpers for small payloads, plus JSON encoding and decoding.

`file_get_contents($path)`	Read a whole file into a string
`file_put_contents($path, $data)`	Write a string to a file
`file($path, FILE_IGNORE_NEW_LINES)`	Read a file into an array of lines
`fopen($path, 'r')`, `fgets($fh)`, `fclose($fh)`	Streamed reads for large files
`fwrite($fh, $data)`	Write to an open handle
`file_exists($p)`, `is_file($p)`, `is_dir($p)`	Existence and type checks
`is_readable($p)`, `is_writable($p)`	Permission checks
`unlink($p)`, `rename($a, $b)`, `copy($a, $b)`	Delete, move, copy
`mkdir($p, 0755, true)` / `rmdir($p)`	Create or remove directories
`scandir($p)` / `glob("*.log")`	List directory entries or match a pattern
`dirname($p)`, `basename($p)`, `pathinfo($p)`	Split a path
`__DIR__`, `__FILE__`	Directory and path of the current file
`realpath($p)`, `filesize($p)`, `filemtime($p)`	Resolve path, size, modification time
`json_encode($data, JSON_PRETTY_PRINT)`	Encode to JSON
`json_decode($s, true)`	Decode to an associative array
`json_encode($d, JSON_THROW_ON_ERROR)`	Throw `JsonException` on failure
`json_validate($s)`	Check validity without decoding, PHP 8.3 and later

Dates and Times

`DateTimeImmutable` is the safer default because arithmetic returns a new object instead of mutating the original.

`time()`	Current Unix timestamp
`date('Y-m-d H:i:s')`	Format the current time
`date('Y-m-d', $ts)`	Format a given timestamp
`strtotime('+1 day')`	Parse a relative or absolute string
`mktime($h, $m, $s, $mo, $d, $y)`	Build a timestamp from parts
`new DateTimeImmutable('2026-09-01')`	Create an immutable date object
`$d->format('D, d M Y')`	Format a date object
`$d->modify('+2 weeks')`	Return a shifted copy
`$d->add(new DateInterval('P1M'))`	Add an interval
`$a->diff($b)->days`	Difference in days
`new DateTimeZone('Europe/Berlin')`	Time zone object
`date_default_timezone_set('UTC')`	Set the script time zone
`checkdate($m, $d, $y)`	Validate a calendar date
`Y m d H i s, D M, N, U`	Common format characters

Errors and Exceptions

`Error` covers engine failures and `Exception` covers application failures; both implement `Throwable`.

`try { } catch (Exception $e) { } finally { }`	Handle and clean up
`catch (TypeError | ValueError $e)`	Catch multiple types
`catch (Exception)`	Catch without capturing the object
`throw new RuntimeException("msg", 500);`	Throw an exception
`$e->getMessage(), $e->getCode()`	Read the message and code
`$e->getFile(), $e->getLine()`	Where the exception was thrown
`$e->getTraceAsString()`	Stack trace as text
`$e->getPrevious()`	Chained exception
`class MyException extends Exception { }`	Custom exception
`error_reporting(E_ALL)`	Report every error level
`ini_set('display_errors', '1')`	Show errors, development only
`ini_set('log_errors', '1')`	Write errors to the log
`set_error_handler($fn)`	Register a user-defined error handler
`set_exception_handler($fn)`	Catch uncaught exceptions
`trigger_error("msg", E_USER_WARNING)`	Raise a user-level error
`@$value`	Error suppression operator, avoid it

Regular Expressions

PHP uses PCRE. Patterns need delimiters, usually `/` or `#`.

`preg_match('/^a/', $s, $m)`	Match once, fill `$m` with captures
`preg_match_all('/\d+/', $s, $m)`	Match every occurrence
`preg_replace('/\s+/', ' ', $s)`	Replace matches
`preg_replace_callback('/\d/', $fn, $s)`	Replace using a callback
`preg_split('/[\s,]+/', $s)`	Split on a pattern
`preg_quote($s, '/')`	Escape user input used in a pattern
`preg_grep('/^a/', $arr)`	Filter array entries by pattern
`/pattern/i`	Case-insensitive
`/pattern/m`	Multiline, ^ and $ match each line
`/pattern/s`	Dot matches newlines
`/pattern/u`	Treat pattern and subject as UTF-8
`(?<name>...)`	Named capture group, read as `$m['name']`

PHP CLI

The `php` binary runs scripts, checks syntax, and starts a development server without a web server in front of it.

`php script.php`	Run a script
`php -v`	Show the PHP version
`php -m`	List compiled and loaded modules
`php -i`	Print the full `phpinfo()` output
`php --ini`	Show which `php.ini` files are loaded
`php -l script.php`	Syntax check without executing
`php -r 'echo PHP_VERSION;'`	Run inline code
`php -a`	Interactive shell
`php -S localhost:8000`	Built-in development server
`php -S localhost:8000 -t public`	Serve a specific document root
`php -d memory_limit=512M script.php`	Override an `ini` setting for one run
`php -c /path/to/php.ini script.php`	Use a specific config file
`php --rf str_replace`	Reflect on a function signature
`php --rc DateTimeImmutable`	Reflect on a class
`php --re json`	Reflect on an extension
`php -n script.php`	Run without loading any `php.ini`

Configuration and Extensions

Configuration paths depend on the SAPI and distribution. Use `php --ini` for the CLI or `phpinfo()` through the web server to confirm which files are active.

`php --ini`	Locate the CLI configuration files
`/etc/php/<version>/cli/php.ini`	CLI config on Debian and Ubuntu
`/etc/php/<version>/fpm/php.ini`	PHP-FPM config on Debian and Ubuntu
`/etc/php/<version>/apache2/php.ini`	Apache module config on Debian and Ubuntu
`/etc/php.ini`, `/etc/php.d/`	Main config and snippets on Fedora and RHEL
`/etc/php/<version>/mods-available/`	Extension snippets on Debian and Ubuntu
`memory_limit`, `max_execution_time`	Per-script resource limits
`upload_max_filesize`, `post_max_size`	Upload limits, raise both together
`display_errors`, `error_log`	Error output and log destination
`date.timezone`	Default time zone
`opcache.enable`, `opcache.memory_consumption`	Bytecode cache settings
`sudo apt install php-gd php-curl php-mbstring`	Install default-version extensions on Ubuntu or Debian
`sudo dnf install php-gd php-curl php-mbstring`	Install extensions on Fedora or RHEL
`php -m | grep -E 'curl|gd|mbstring'`	Confirm extensions are loaded
`sudo phpenmod curl` / `sudo phpdismod curl`	Enable or disable an extension on Debian or Ubuntu
`sudo update-alternatives --config php`	Select an installed CLI version on Debian or Ubuntu

PHP-FPM

PHP-FPM is the process manager that Nginx and Apache hand PHP requests to. The versioned examples below use PHP 8.5 on Ubuntu 26.04; Debian 13 uses 8.4, while Fedora and RHEL use unversioned `php-fpm` names. Reload or restart FPM after config or extension changes.

`sudo systemctl status php8.5-fpm`	Check the Ubuntu 26.04 service state
`sudo systemctl status php-fpm`	Check the Fedora or RHEL service state
`sudo systemctl restart php8.5-fpm`	Restart the versioned service
`sudo systemctl reload php8.5-fpm`	Gracefully reload FPM workers
`sudo php-fpm8.5 -t`	Test Ubuntu 26.04 configuration before reloading
`/etc/php/8.5/fpm/pool.d/www.conf`	Ubuntu 26.04 default pool configuration
`listen = /run/php/php8.5-fpm.sock`	Unix socket the web server connects to
`user / group`	System account the workers run as
`pm = dynamic`	Process manager mode: `static`, `dynamic`, or `ondemand`
`pm.max_children`	Hard cap on worker processes
`pm.start_servers, pm.min_spare_servers`	Warm pool sizing for `dynamic`
`pm.max_requests`	Recycle a worker after N requests
`php_admin_value[memory_limit] = 256M`	Override an `ini` value per pool
`slowlog, request_slowlog_timeout`	Log requests that run too long
`sudo journalctl -u php8.5-fpm -f`	Follow the versioned service log

Composer

Composer manages dependencies and the autoloader for almost every modern PHP project.

`composer init`	Create a `composer.json` interactively
`composer require vendor/package`	Add a dependency
`composer require --dev phpunit/phpunit`	Add a development dependency
`composer install`	Install from `composer.lock`
`composer update`	Update dependencies and the lock file
`composer update vendor/package`	Update a single package
`composer remove vendor/package`	Remove a dependency
`composer install --no-dev --optimize-autoloader`	Production install
`composer dump-autoload -o`	Regenerate an optimized autoloader
`composer show / composer show -t`	List packages, or show the dependency tree
`composer outdated`	List packages with newer versions
`composer why vendor/package`	Explain why a package is installed
`composer audit`	Check dependencies for known vulnerabilities
`composer create-project vendor/skeleton app`	Start a project from a skeleton
`require 'vendor/autoload.php';`	Load the autoloader in your entry script

Related Guides

Use these guides to install PHP, check the running version, and debug errors on a server.

How to Install PHP on Ubuntu 26.04	Install PHP 8.5 with Apache or Nginx and PHP-FPM
How to Check the PHP Version	Find the CLI and web server PHP versions
PHP Error Reporting	Show, log, and control PHP errors
How to Install a LAMP Stack on Ubuntu 26.04	Apache, MySQL, and PHP on one server
MySQL and MariaDB Cheatsheet	Database commands for the data layer behind PHP