Xdebug Cheatsheet

A comprehensive cheat sheet for Xdebug, covering installation, configuration, debugging features, and common use cases to streamline PHP debugging.

Installation and Configuration

Installation

1. **Download Xdebug:** Obtain the appropriate Xdebug version for your PHP version and architecture from the official Xdebug website.
2. **Locate PHP Extension Directory:** Find the directory where PHP extensions are stored (e.g., `/usr/lib/php/20190902/`). You can find this with `php -i | grep extension_dir`.
3. **Move the Xdebug Extension:** Place the downloaded Xdebug extension file (e.g., `xdebug.so`) into the PHP extension directory.
4. **Configure PHP:** Edit the `php.ini` file to enable Xdebug.
5. **Restart Web Server:** Restart your web server (e.g., Apache, Nginx) for the changes to take effect.

Basic Configuration

`zend_extension=xdebug.so`	Enables the Xdebug extension.
`xdebug.mode=debug`	Sets the Xdebug mode to 'debug' for debugging features. Can also use 'profile', 'coverage', 'develop', 'coverage,debug'.
`xdebug.start_with_request=yes`	Automatically starts a debugging session for every request.
`xdebug.client_host=localhost`	Specifies the host where the debugging client (IDE) is running.
`xdebug.client_port=9003`	Specifies the port on which Xdebug attempts to connect to the debugging client.
`xdebug.log=/tmp/xdebug.log`	Specifies the file where Xdebug logs its activities. Useful for troubleshooting.

Configuration Verification

1. **Check PHP Info:** Create a `phpinfo()` page in your web directory and access it through your browser.
2. **Search for Xdebug:** Look for the Xdebug section in the `phpinfo()` output to verify that Xdebug is loaded and configured correctly.
3. **Verify Settings:** Confirm that the settings you configured in `php.ini` are reflected in the `phpinfo()` output.

Debugging with Xdebug

IDE Integration

Xdebug integrates with various IDEs like VS Code, PhpStorm, and NetBeans. Configure your IDE to listen for Xdebug connections on the specified port (e.g., 9003).

VS Code: Install the 'PHP Debug' extension and configure the `launch.json` file.
PhpStorm: Configure the server and debugging settings in the 'Preferences' or 'Settings' panel.
NetBeans: Enable Xdebug support and set the debugging port in the 'Options' panel.

Breakpoints

Setting Breakpoints	Place breakpoints in your code where you want the execution to pause. Use your IDE to set breakpoints by clicking in the editor's gutter.
Conditional Breakpoints	Set breakpoints that trigger only when a specific condition is met. This helps in debugging complex scenarios.
Line Breakpoints	Break on specific lines of code to examine variables and execution flow.
Exception Breakpoints	Break when specific exception is thrown.

Debugging Operations

Step Over: Execute the current line and move to the next line in the same scope.
Step Into: Enter a function or method call to debug its execution.
Step Out: Exit the current function or method and return to the calling scope.
Continue: Resume normal execution until the next breakpoint or the end of the script.
Evaluate Expression: Evaluate the value of a variable or expression at the current breakpoint.

Watch Variables: Monitor the values of specific variables as you step through the code. This helps track changes and identify issues.
Inspect Variables: Examine the current state of all variables in the current scope. Useful for understanding the context of the execution.

Advanced Xdebug Features

Profiling

Xdebug can generate profiling information to analyze the performance of your PHP code. Profiling helps identify bottlenecks and optimize slow-running code.
Enable profiling by setting `xdebug.mode=profile` in `php.ini`. Specify the output file location using `xdebug.output_dir` and `xdebug.profiler_output_name`.
Use tools like KCachegrind or Webgrind to visualize and analyze the profiling data. These tools provide insights into function call durations, memory usage, and other performance metrics.

Code Coverage Analysis

Enabling Coverage	Configure Xdebug to collect code coverage data by setting `xdebug.mode=coverage` in `php.ini`.
Generating Reports	Use PHPUnit with the `--coverage-html` or `--coverage-clover` options to generate HTML or XML reports showing which lines of code are covered by your tests.
Analyzing Coverage	Review the generated reports to identify uncovered code and improve your test suite.

Remote Debugging

Xdebug supports remote debugging, allowing you to debug code running on a remote server from your local IDE. Configure `xdebug.client_host` to point to your local machine's IP address.
Ensure that the remote server can connect to your local machine on the specified port (e.g., 9003). You may need to configure firewall rules to allow the connection.
Use SSH tunneling to create a secure connection between your local machine and the remote server. This is especially useful when debugging code in production environments.

Troubleshooting

Common Issues

Xdebug Not Loading: Verify that the `zend_extension` directive is correctly configured in `php.ini` and that the Xdebug extension file exists in the specified directory. Also, verify that the loaded `php.ini` is the one you are modifying (use `phpinfo()` to check). **Connection Refused:** Ensure that your IDE is listening for Xdebug connections on the correct port and that there are no firewall rules blocking the connection. **Breakpoints Not Hitting:** Confirm that the file paths in your IDE match the actual file paths on the server. Check that the Xdebug settings in your IDE are correctly configured. **Slow Performance:** Xdebug can slow down script execution. Disable Xdebug when not debugging, or use the `develop` mode introduced in Xdebug 3.

Debugging Techniques

Logging	Use `xdebug_info()` and `xdebug_var_dump()` to output debugging information to the browser or log files. These functions can help you inspect variables and execution flow.
Remote Session	Start a debugging session from your browser by adding `XDEBUG_SESSION_START=name` to the URL query string. This triggers Xdebug to connect to your IDE and start debugging.
Error Handling	Use `try-catch` blocks to handle exceptions and set breakpoints in the `catch` block to debug error conditions. Configure Xdebug to break on exceptions to catch errors early.
Code Review	Sometimes, the best debugging technique is to carefully review your code and look for potential errors. Use Xdebug in conjunction with code review to identify and fix issues.

Xdebug 3 Migration

Xdebug 3 introduces significant changes in configuration. Replace the old `xdebug.remote_*` settings with the new `xdebug.client_*` and `xdebug.mode` settings.
Use `xdebug.mode` to specify the desired functionality (e.g., `debug`, `profile`, `coverage`). The `develop` mode is useful for general development and includes error reporting and stack traces.
Ensure that your IDE is compatible with Xdebug 3 and that you have updated your debugging configuration accordingly.