PHPUnit PHP testing framework

START HERE

`composer require --dev phpunit/phpunit`
Install it

`vendor/bin/phpunit --generate-configuration`
Create phpunit.xml

`vendor/bin/phpunit`
Run all tests

`vendor/bin/phpunit --coverage-text`
Coverage summary

`vendor/bin/phpunit <path>`
Run a file or dir

`vendor/bin/phpunit --filter <name>`
Run matching tests

`vendor/bin/phpunit --testdox`
Readable results

`vendor/bin/phpunit --stop-on-failure`
Stop at first failure

phpunit.xml suites, source → `discover` → **Test classes** *Test.php → `run` → **Result** dots, F, E, R → `report` → **Reports** coverage, JUnit

CONCEPTS

A test is a public method named test* or marked #[Test] in a class extending TestCase whose file ends in Test.php.

Without a phpunit.xml PHPUnit takes the path argument as the tests to run; with one, a bare call runs the whole config.

Coverage needs a driver: Xdebug (xdebug.mode=coverage) or PCOV. Without one, coverage options only warn.

Metadata such as groups and providers is read from PHP attributes. Doc-comment annotations were dropped in PHPUnit 12.

RUN AND SELECT

`vendor/bin/phpunit`
Run the configured tests

`vendor/bin/phpunit <path>`
Run one file or directory

RUN AND SELECT (cont.)

`vendor/bin/phpunit --filter <pattern>`
Run tests whose name matches
Matches class, method and data set names; repeating --filter combines with AND.

`--testsuite <name>`
Run one suite from the config

`--group <name>`
Run tests in a group

`--exclude-group <name>`
Skip a group

EXECUTION

`--stop-on-failure`
Stop at the first failure

`--fail-on-warning`
Exit non-zero on warnings

`--order-by <order>`
Reorder tests
Orders: default, defects, depends, duration, no-depends, random, reverse, size.

OUTPUT

`--testdox`
Show results as readable sentences

`--display-deprecations`
Show deprecation details

OUTPUT (cont.)

`--display-warnings`
Show warning details

COVERAGE

`--coverage-text`
Print a coverage table

`--coverage-html <dir>`
Write an HTML report

`XDEBUG_MODE=coverage vendor/bin/phpunit --coverage-html <dir>`
Coverage with Xdebug

CONFIGURATION

`--generate-configuration`
Interactively create phpunit.xml

`-c <file>`
Use another config file

Minimal phpunit.xml

```
<phpunit bootstrap="vendor/
autoload.php" colors="true"
         failOnWarning="true"
failOnRisky="true"
         cacheDirectory=".
phpunit.cache">
  <testsuites>
    <testsuite name=
"unit"><directory>tests/unit</
directory></testsuite>
  </testsuites>
  <source><include><directory
suffix=".php">src</directory></
include></source>
</phpunit>
```

WRITING TESTS

A basic test

```
use PHPUnit\Framework\TestCase;

final class CartTest extends
TestCase
{
    protected function setUp():
void { $this->cart = new
Cart; }

    #[Test]
    public function
adds_items(): void
    {
        $this->cart-
>add('apple');
        $this->assertCount(1,
$this->cart);
    }
}
```

`assertSame($expected, $actual)`
Same type and value (===)

`assertEquals($expected, $actual)`
Loosely equal (==)

`assertTrue($cond)` / `assertFalse($cond)`
Boolean checks

`assertNull($v)` / `assertNotNull($v)`
Null checks

`assertCount($n, $items)`
Number of elements

`assertInstanceOf(<Class>::class, $v)`
Value is of a class

WRITING TESTS (cont.)

`assertEqualsWithDelta($a, $b, $delta)`
Floats within tolerance

`$this->expectException(<Class>::class)`
Expect a thrown exception
Call it before the code that throws.

`setUp()` / `tearDown()`
Run before and after each test

ATTRIBUTES

`#[Test]`
Mark a method as a test

`#[DataProvider('<method>')]`
Feed data sets from a static method

`#[TestWith([<args>])]`
Inline data set

`#[Group('<name>')]`
Put a test in a group

Data provider

```
#[DataProvider('sums')]
public function test_sum(int
$a, int $b, int $want): void
{
    $this->assertSame($want, $a
+ $b);
}

public static function sums():
array
{
    return ['zero' => [0, 0,
0], 'one' => [1, 0, 1]];
}
```

ATTRIBUTES (cont.)

Provider methods must be public static. String keys name the data sets, so --filter 'test_sum with data set "zero"' hits one.

TEST DOUBLES

`$this->createStub(<Class>::class)`
Canned-answer double, no expectations

`$this->createMock(<Class>::class)`
Double that can verify calls

`$stub->method('<m>')->willReturn($v)`
Fix a return value

`->willReturnMap([[<args>, $ret]])`
Return by argument

`->willThrowException(new <Ex>)`
Make the method throw

`$mock->expects($this->once())->method('<m>')`
Require exactly one call

Only call expects() on a mock. A createStub() double ignores how it is used and never fails the test.