# Bats Bash test runner

**START HERE**

`bats test/`
Run all .bats files in a dir

`[ "$status" -eq 0 ]`
Assert the exit status

`bats <file>.bats`
Run one test file

`setup() { ... }`
Runs before every test

`@test "name" { ... }`
Define a test

`bats -r test/`
Include subdirectories

`run <cmd>`
Capture status and output

`bats -f <regex> test/`
Run tests by name

**setup_suite** once per run → then **setup_file** once per file → then **setup** each test → @test **teardown** each test → then **teardown_file** once per file

## CONCEPTS

A test passes if every command in it succeeds (Bats runs tests with set -e); run never fails, so you assert on $status and $output afterwards.

Each .bats file is evaluated once per test plus once more, so keep top-level code free of output and side effects.

## RUNNING

`bats <path>...`
Run files or directories of .bats files

`bats -r <dir>`
Recurse into subdirectories

`bats -f <regex> <path>`
Run only tests whose name matches

`bats --negative-filter <regex> <path>`
Skip tests whose name matches

`bats -c <path>`
Count tests without running them

`bats --abort <path>`
Stop at the first failing test

`bats -j <n> <path>`
Run tests in parallel (needs GNU parallel)

`bats -x <path>`
Trace commands like set -x

`bats --print-output-on-failure <path>`
Show $output of failed tests

`bats --show-output-of-passing-tests <path>`
Show output of passing tests too

`bats -T <path>`
Add timing to each result

`bats --filter-status failed <path>`
Rerun failed or missing tests

`bats -v / bats -h`
Print version / help

## OUTPUT

`bats -p <path>`
Pretty output (default on a terminal)

`bats -t <path>`
TAP output (default when piped)

`bats -F junit <path>`
JUnit XML for CI

## OUTPUT (cont.)

`bats --report-formatter junit -o <dir> <path>`
Write a report file to <dir>
*The output directory must already exist.*

`bats --no-tempdir-cleanup <path>`
Keep the temp dir for debugging

Exit code is 0 when all tests pass, 1 on any failure or when no tests are found.

## WRITING TESTS

`@test "<name>" { <commands> }`
Define a test

`load <file>`
Source a helper relative to the test file
*Give the name without the .bash extension.*

`skip ["<reason>"]`
Skip this test

`bats_require_minimum_version`
Fail on a too-old Bats: pass <ver>

`bats_load_library <name>`
Load a library from BATS_LIB_PATH

`echo "# <msg>" >&3`
Print a message during a test
*Plain stdout and stderr show only when a test fails.*

`# bats test_tags=<a>,<b>`
Tag the next test

`# bats file_tags=<a>,<b>`
Tag all tests in the file

`bats --filter-tags <a>,<b> <path>`
Run tests that have all these tags
*Repeat the flag for OR; prefix a tag with ! to negate.*

`# bats test_tags=bats:focus`
Run only focused tests
*A focused run still exits 1 unless BATS_NO_FAIL_FOCUS_RUN=1.*

`bats_test_function --description <d> -- <fn>`
Register tests from a loop

Skipped tests still run setup and teardown.

## RUN AND ASSERT

`run <cmd> [args]`
Capture status and output; never fails

## RUN AND ASSERT (cont.)

`run -<N> <cmd>`
Fail unless exit status is N

`run ! <cmd>`
Fail unless exit status is nonzero
*A bare ! in a test does not fail it reliably.*

`run --separate-stderr <cmd>`
Split stdout and stderr

`run --keep-empty-lines <cmd>`
Keep blank lines in ${lines[@]}

`run bats_pipe <cmd1> \| <cmd2>`
Run a pipeline under run

`[ "$status" -eq 0 ]`
Assert exit status

`[ "$output" = "<text>" ]`
Assert combined output

`[ "${lines[0]}" = "<text>" ]`
Assert one output line

`[[ "$output" == *"<part>"* ]]`
Assert output contains text

`$stderr / ${stderr_lines[@]}`
Stderr (with --separate-stderr)

run uses a subshell, so variables set by the command are gone afterwards. Call the function directly if you need them.

## SETUP AND TEARDOWN

`setup() { ... }`
Before each test

`teardown() { ... }`
After each test, even on failure

`setup_file() { ... }`
Once before all tests in a file

`teardown_file() { ... }`
Once after all tests in a file

`setup_suite() { ... }`
Once per run, in setup_suite.bash

`teardown_suite() { ... }`
Once per run after the last test

`$BATS_TEST_TMPDIR`
Fresh temp dir for each test

`$BATS_FILE_TMPDIR`
Temp dir shared within a file

`$BATS_SUITE_TMPDIR`
Temp dir shared across the run

`$BATS_TEST_DIRNAME`
Directory of the test file

## SETUP AND TEARDOWN (cont.)

`$BATS_TEST_TIMEOUT`
Per-test timeout in seconds

Teardown runs with set -e off, but a failing command in it still fails the test.

## HELPER LIBRARIES

bats-support and bats-assert are separate repos, not part of Bats itself. Check the version your project installs before using them.

`load 'test_helper/bats-support/load'`
Load bats-support first

`load 'test_helper/bats-assert/load'`
Then bats-assert

`assert_success / assert_failure`
Check exit status after run

`assert_output "<text>"`
Check output (--partial, --regexp)

## RECIPES

**A minimal test file**

```
#!/usr/bin/env bats

setup() {
  cd "$BATS_TEST_TMPDIR"
}

@test "echo prints its argument" {
  run echo hello
  [ "$status" -eq 0 ]
  [ "$output" = "hello" ]
}
```

**Test a script in the repo**

```
setup() {
  DIR="$(cd "$BATS_TEST_DIRNAME/.." && pwd)"
  PATH="$DIR/bin:$PATH"
}

@test "fails without args" {
  run -1 mytool
  [[ "$output" == *"usage"* ]]
}
```

**CI run with JUnit report**

```
mkdir -p reports
bats -r --report-formatter junit -o reports test/
# or machine-readable on stdout
bats -r -t test/ > results.tap
```

**Fast local loop**

```
bats -r -j 4 --print-output-on-failure test/
# rerun only what failed last time
bats --filter-status failed test/
```