A Tasty Taste of Tezt Tests

Romain Bardou

OUPS 2022-05-12

2019 — Testing in Tezos

Multiple frameworks:

- unit tests using Alcotest (OCaml framework)
- lots of integration tests in Pytest (Python framework)
- some integration tests in Flextesa (OCaml framework)

Being a release manager involved lots of cherry-picking from `master` to release branches.

- tests would fail
- hard to find why
- even harder as I was not the author
- even harder as I barely ever used Python (wth is a *fixture*??)

As a Wheel Reinventor™, I decided to make my own framework.

2020 — Tezt Proof of Concept (1)

Focus on UX:

- immediately visible helpful error messages
  - in red
  - at the end of logs
  - copy-pastable command to reproduce
- easy test selection from command-line
  - from test tags
  - from test title
  - from source file
  - `--list` to list all tests

2020 — Tezt Proof of Concept (2)

Focus on integration tests and UX again:

- easy to launch external processes (with Lwt)
- one color per process
- log commands
- log process stdout and stderr
- log exit codes
- declare temporary files for automatic cleanup (optional)

Goal: make it easy for me to debug other people's tests.

2020 — Tezt Proof of Concept (3)

Focus on CI integration:

- if all goes well, only log one "success" line per test
- in case of error, log lines that lead to the error
- log everything to a log file (stored as CI artifact)

Constraint: CI log cannot be too large.

Common theme up to now: **good logs are important**

2020-2022 — Improvements

- devs happy with proof of concept
- lots of tests got written
- everyday use leads to improvements

2020-2022 — Improvements — Regression Tests

- capture output to file
- compare with previous output
- regexps to substitute non-deterministic parts

Used to test Tezos RPCs and encodings.

2020-2022 — Improvements — Auto-Balancing

- store test duration in files
- use those files to split tests in batches of roughly the same time
- run one batch per CI job
- easily increase the number of CI jobs

Used to reduce CI pipeline total times.

2020-2022 — Improvements — Parallel Tests

A la `make -j`:

- fork process for each test
- main process becomes scheduler
- scheduler limits maximum number of simultaneous forks

Provides significant speedup, both locally and in the CI.

Caveats:

- make sure network ports are not shared between tests
- make sure temporary files are not shared (this is automatic)

2020-2022 — Improvements — Remote Processes

You can spawn your processes remotely through SSH.

```
let runner =
  Runner.create
    ~ssh_port: 2222
    ~address: "192.168.0.42"
    ()
in
Process.spawn ~runner "tezos-node" [ "run" ]
|> Process.check
```

Example: spawn a Tezos test network on multiple machines

2022 — Testing in Octez (1)

(OCaml implementation of Tezos now called Octez)

Multiple frameworks:

- unit tests using Alcotest (OCaml framework)
- lots of integration tests in Pytest (Python framework)
- some integration tests in Flextesa (OCaml framework)
- lots of integration tests in Tezt

2022 — Testing in Octez (2)

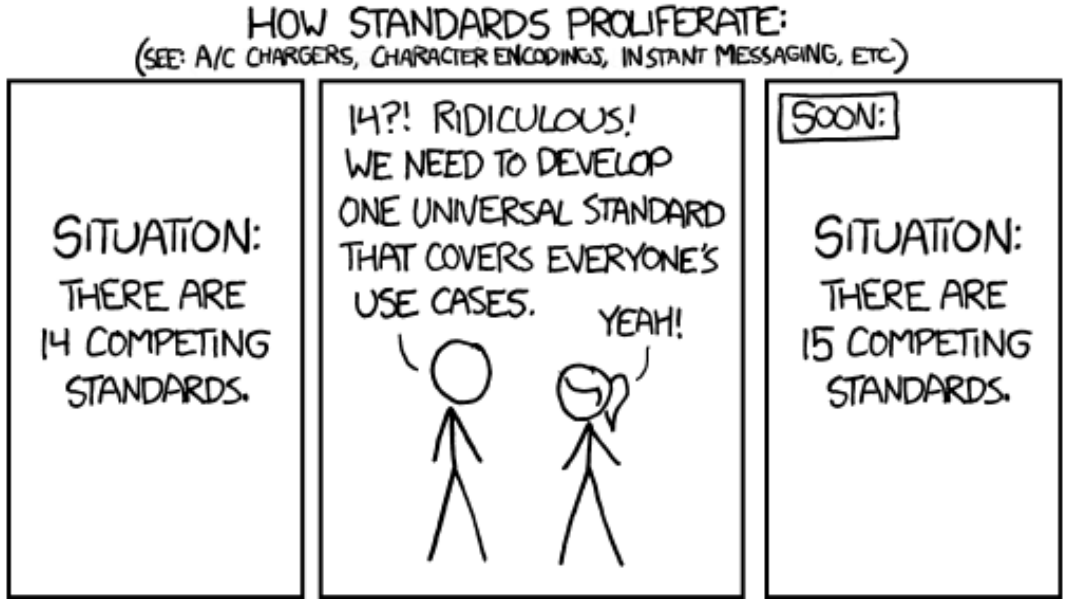

Figure 1: Standards

Title text: *Fortunately, the charging one has been solved now that we've all standardized on mini-USB. Or is it micro-USB? Shit.*

2022 — Testing in Octez (3)

But:

- only one Flextesa test left
- no new Python tests
- considering using Tezt for unit tests

Tezt initially marketed as integration test framework.

But has all features we need from Alcotest and more.

A Tasty Taste of Tezt (1) — Basic Unit Test

```
let () =
  Test.register
    ~__FILE__
    ~title: "basic test"
    ~tags: ["basic"]
  @@ fun () ->
  Log.info "hello I'm a test";
  if 1 = 2 then Test.fail "universe exploded, sorry";
  unit

let () = Test.run ()
```

In dune:

```
(test
 (name main)
 (libraries tezt)
 (flags (:standard -open Tezt -open Tezt.Base)))
```

A Tasty Taste of Tezt (2) — Running Tests

Run test:

```
dune runtest
dune exec main.exe
```

Tip:

```
alias tezt='dune exec main.exe --'
```

Get list of registered tests (files, titles and tags):

```
tezt --list
```

Run only our basic test:

```
tezt basic
tezt --title 'basic test'
```

A Tasty Taste of Tezt (3) — Basic Integration Test

```
let () =
  Test.register
    ~__FILE__
    ~title: "basic integration test"
    ~tags: ["basic"; "integration"]
  @@ fun () ->
  Process.run "git" [ "--help" ]
```

You can also:

- read stdout (`Process.run_and_read_stdout`)
- read stderr (`Process.run_and_read_stderr`)
- not wait for process to exit (`Process.spawn`)
  - then check exit code is 0 (`Process.check`)
  - or check for errors (`Process.check_error`)
  - or just read the exit code (`Process.wait`)
  - and the output (`Process.stdout`, `Process.stderr`)
    - as `Lwt_io.input_channel`

A Tasty Taste of Tezt (3) — Basic Regression Test

```
let () =
  Regression.register
    ~__FILE__
    ~title: "basic regression test"
    ~tags: ["basic"]
    ~output_file: "regression"
  @@ fun () ->
  Regression.capture "some constant string";
  Process.run "git" [ "--help" ]
    ~hooks: Regression.hooks
```

Initialize output file with:

```
tezt --reset-regressions regression
```

(tag `regression` is automatically added by `Regression.register`)

A Tasty Taste of Tezt (4) — Successful Output

```
$ dune exec main.exe
[09:41:40.582] [SUCCESS] (1/3) basic unit test
[09:41:40.583] [SUCCESS] (2/3) basic integration test
[09:41:40.585] [SUCCESS] (3/3) basic regression test
```

A Tasty Taste of Tezt (5) — Failed Output

```
$ dune exec main.exe
[09:44:20.166] Starting test: basic unit test
[09:44:20.166] hello I'm a test
[09:44:20.166] [error] universe exploded, sorry
[09:44:20.166] [FAILURE] (1/3, 1 failed) basic unit test
[09:44:20.166] Try again with: main.exe --verbose
--file main.ml --title 'basic unit test'
```

A Tasty Taste of Tezt (6) — List

```
$ dune exec main.exe -- --list
+---------+-------------------------+---------------------+
|  FILE   |          TITLE          |        TAGS         |
+---------+-------------------------+---------------------+
| main.ml | basic unit test         | basic, unit         |
| main.ml | basic integration test  | basic, integration  |
| main.ml | basic regression test   | regression, basic   |
+---------+-------------------------+---------------------+
```

A Tasty Taste of Tezt (7) — Select

```
$ dune exec main.exe -- --file main.ml /regression
[09:42:57.576] [SUCCESS] (1/2) basic unit test
[09:42:57.577] [SUCCESS] (2/2) basic integration test
```

A Tasty Taste of Tezt (8) — Auto-Balancing

```
$ dune exec main.exe -- --list --job 1/2
+---------+-------------------------+---------------------+
|  FILE   |          TITLE          |        TAGS         |
+---------+-------------------------+---------------------+
| main.ml | basic unit test         | basic, unit         |
| main.ml | basic integration test  | basic, integration  |
+---------+-------------------------+---------------------+
$ dune exec main.exe -- --list --job 2/2
+---------+-------------------------+---------------------+
|  FILE   |          TITLE          |        TAGS         |
+---------+-------------------------+---------------------+
| main.ml | basic regression test   | regression, basic   |
+---------+-------------------------+---------------------+
```

Tezt — Other Features

A `JSON` module to easily parse JSON:

```
let* rpc_response = ... in
let json =
  JSON.parse ~origin: "RPC response" rpc_response
in
let name =
  JSON.(json |-> "name" |-> "firstname" |> as_string)
in
```

Tezt — A General Testing Framework

Tezt supports:

- unit tests
- integration tests
- regression tests

Using the same framework for everything is nice:

- same UX
- only one lib to learn
- less dependencies

Try it!

```
opam install tezt
```