Enstabilize & Quickify Yer Tests To Do Better Stuff

Mat Trudel // mat@geeky.net // @mtrudel // github.com/mtrudel/talks

Tests are the foundation of your work

(and you should strive to have good ones)

What makes for good tests?

Tests should be **clear**
Tests should be **correct**
Tests should be **reliable**
Tests should be **ubiquitous**

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable**

Tests should be **ubiquitous**

Tests should be **clear**

Consistent Setup / Trigger / Assertion structure

```
test "getting user details" do
  # Setup
  user = user_fixture()

  # Trigger
  response = Req.get("/users/#{user.id}")

  # Assertions
  assert response.name == user.name
end
```

Tests should be **clear**

Consistent Setup / Trigger / Assertion structure

```
test "getting user details" do
  # Setup
  user = user_fixture()

  # Trigger
  response = Req.get("/users/#{user.id}")

  # Assertions
  assert response.name == user.name
end
```

Tests should be **clear**

Consistent Setup / Trigger / Assertion structure

```
test "getting user details" do
  # Setup
  user = user_fixture()

  # Trigger
  response = Req.get("/users/#{user.id}")

  # Assertions
  assert response.name == user.name
end
```

Tests should be **clear**

Consistent Setup / Trigger / Assertion structure

```
test "getting user details" do
  # Setup
  user = user_fixture()

  # Trigger
  response = Req.get("/users/#{user.id}")

  # Assertions
  assert response.name == user.name
end
```

Tests should be **clear**

Factor up repetitive setup

- `ExUnit.Callback` has a ton of useful functionality
  - We all know about `setup` and `setup_all`
  - There's also `start_supervised`
    - Process lifecycle 'just works'
    - 1.17+ `start_supervised` now sets `$callers` and `$ancestors`
  - My personal favourite though, are named setup callbacks
  - A real-life example from Bandit ties all this together:

```elixir
defmodule PlugTest do
  use ServerHelpers

  setup :http_server

  test "the plumbing all works", context do
    response = Req.get!("#{context.base}/hello_world")

    assert response.body == "Hello, World!"
  end

  def call(conn, _opts) do
    send_resp(conn, 200, "Hello, World!")
  end
end

defmodule ServerHelpers do
  defmacro __using__(_) do
    quote do
      import Plug.Conn

      def http_server(_context, opts \\ [plug: __MODULE__]) do
        {:ok, server_pid} = opts |> Bandit.child_spec() |> start_supervised()
        {:ok, {_ip, port}} = ThousandIsland.listener_info(server_pid)
        [base: "http://localhost:#{port}"]
      end
    end
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/support/server_helpers.ex

```elixir
defmodule PlugTest do
  use ServerHelpers

  setup :http_server

  test "the plumbing all works", context do
    response = Req.get!("#{context.base}/hello_world")

    assert response.body == "Hello, World!"
  end

  def call(conn, _opts) do
    send_resp(conn, 200, "Hello, World!")
  end
end

defmodule ServerHelpers do
  defmacro __using__(_) do
    quote do
      import Plug.Conn

      def http_server(_context, opts \\ [plug: __MODULE__]) do
        {:ok, server_pid} = opts |> Bandit.child_spec() |> start_supervised()
        {:ok, {_ip, port}} = ThousandIsland.listener_info(server_pid)
        [base: "http://localhost:#{port}"]
      end
    end
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/support/server_helpers.ex

```elixir
defmodule PlugTest do
  use ServerHelpers

  setup :http_server

  test "the plumbing all works", context do
    response = Req.get!("#{context.base}/hello_world")

    assert response.body == "Hello, World!"
  end

  def call(conn, _opts) do
    send_resp(conn, 200, "Hello, World!")
  end
end

defmodule ServerHelpers do
  defmacro __using__(_) do
    quote do
      import Plug.Conn

      def http_server(_context, opts \\ [plug: __MODULE__]) do
        {:ok, server_pid} = opts |> Bandit.child_spec() |> start_supervised()
        {:ok, {_ip, port}} = ThousandIsland.listener_info(server_pid)
        [base: "http://localhost:#{port}"]
      end
    end
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/support/server_helpers.ex

Tests should be **clear**

Meaningful assertions

- Your assertions should be clear & organized
  - Multiple *related* assertions in a single test are fine
  - Balance between comprehensive & verbose is tricky
- Your tests should be testing what you think they are
  - Beware of the semantic of = vs ==, e.g.:

    ```
    expected = 123
    assert expected = 123
    ```

    - The compiler will *usually* warn
    - ...or you could just use `Machete`

Tests should be **clear**

You should `use Machete`

- Literate matchers for Elixir
  - Defines the ~> operator
  - Straightforward literal and var matching
  - Powerful & extensible parameteric matching
  - Robust collection support (including parametric matchers)
  - Useful error messages in ExUnit

https://github.com/mtrudel/machete

Tests should be **clear**

You should use `Machete`

```
assert "abc" ~> "abc"

assert "abc" ~> string()

assert 123 ~> integer(positive: true)

assert "2025-02-12T16:45:00Z" ~> iso8601_datetime(roughly: :now)

assert %{a: "abc", b: 123} ~> %{a: "abc", b: integer()}

assert ["def", "abc"] ~> in_any_order(["abc", "def"])

assert ["abc", "def"] ~> list_of(string(), min: 1, max: 5)
```

https://github.com/mtrudel/machete

```elixir
test "it should send `stop` events for normally completing requests", context do
  Req.get!(context.req, url: "/send_200")

  assert_receive {:telemetry, [:bandit, :request, :stop], measurements, metadata}, 500

  assert measurements
         ~> %{
           resp_body_bytes: 0,
           duration: integer(max: System.convert_time_unit(1, :second, :native)),
           monotonic_time: integer(roughly: System.monotonic_time()),
           req_header_end_time: integer(roughly: System.monotonic_time()),
           resp_start_time: integer(roughly: System.monotonic_time()),
           resp_end_time: integer(roughly: System.monotonic_time())
         }

  assert metadata
         ~> %{
           connection_telemetry_span_context: reference(),
           telemetry_span_context: reference(),
           conn: struct_like(Plug.Conn, path_info: ["send_200"]),
           plug: {__MODULE__, []}
         }
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```elixir
test "it should send `stop` events for normally completing requests", context do
  Req.get!(context.req, url: "/send_200")

  assert_receive {:telemetry, [:bandit, :request, :stop], measurements, metadata}, 500

  assert measurements
         ~> %{
           resp_body_bytes: 0,
           duration: integer(max: System.convert_time_unit(1, :second, :native)),
           monotonic_time: integer(roughly: System.monotonic_time()),
           req_header_end_time: integer(roughly: System.monotonic_time()),
           resp_start_time: integer(roughly: System.monotonic_time()),
           resp_end_time: integer(roughly: System.monotonic_time())
         }

  assert metadata
         ~> %{
           connection_telemetry_span_context: reference(),
           telemetry_span_context: reference(),
           conn: struct_like(Plug.Conn, path_info: ["send_200"]),
           plug: {__MODULE__, []}
         }
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```elixir
test "it should send `stop` events for normally completing requests", context do
  Req.get!(context.req, url: "/send_200")

  assert_receive {:telemetry, [:bandit, :request, :stop], measurements, metadata}, 500

  assert measurements
         ~> %{
           resp_body_bytes: 0,
           duration: integer(max: System.convert_time_unit(1, :second, :native)),
           monotonic_time: integer(roughly: System.monotonic_time()),
           req_header_end_time: integer(roughly: System.monotonic_time()),
           resp_start_time: integer(roughly: System.monotonic_time()),
           resp_end_time: integer(roughly: System.monotonic_time())
         }

  assert metadata
         ~> %{
           connection_telemetry_span_context: reference(),
           telemetry_span_context: reference(),
           conn: struct_like(Plug.Conn, path_info: ["send_200"]),
           plug: {__MODULE__, []}
         }
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```
test "it should send `stop` events for normally completing requests", context do
  Req.get!(context.req, url: "/send_200")

  assert_receive {:telemetry, [:bandit, :request, :stop], measurements, metadata}, 500

  assert measurements
          ~> %{
            resp_body_bytes: 0,
            duration: integer(max: System.convert_time_unit(1, :second, :native)),
            monotonic_time: integer(roughly: System.monotonic_time()),
            req_header_end_time: integer(roughly: System.monotonic_time()),
            resp_start_time: integer(roughly: System.monotonic_time()),
            resp_end_time: integer(roughly: System.monotonic_time())
          }

  assert metadata
          ~> %{
            connection_telemetry_span_context: reference(),
            telemetry_span_context: reference(),
            conn: struct_like(Plug.Conn, path_info: ["send_200"]),
            plug: {__MODULE__, []}
          }
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```elixir
test "it should send `stop` events for normally completing requests", context do
  Req.get!(context.req, url: "/send_200")

  assert_receive {:telemetry, [:bandit, :request, :stop], measurements, metadata}, 500

  assert measurements
         ~> %{
           resp_body_bytes: 0,
           duration: integer(max: System.convert_time_unit(1, :second, :native)),
           monotonic_time: integer(roughly: System.monotonic_time()),
           req_header_end_time: integer(roughly: System.monotonic_time()),
           resp_start_time: integer(roughly: System.monotonic_time()),
           resp_end_time: integer(roughly: System.monotonic_time())
         }

  assert metadata
         ~> %{
           connection_telemetry_span_context: reference(),
           telemetry_span_context: reference(),
           conn: struct_like(Plug.Conn, path_info: ["send_200"]),
           plug: {__MODULE__, []}
         }
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable**

Tests should be **ubiquitous**

Tests should be **correct**

A test is meaningless until you've seen it fail

- Red, green, refactor is basically gospel
  - The *very first* thing you should do is build a repro case
  - The *next* thing you should do is to codify that in a test
  - This is so important: **WATCH THAT TEST FAIL**
  - *Then* (and only then) can you get to fixing it
- This helps ensure that you're testing the right thing

Tests should be **correct**

Defence in depth

- More tests are a good thing
    - This *doesn't* mean 'blindly write more test cases'
- Scope your test plan to different levels of abstraction
    - Unit tests for important / subtle modules
    - Acceptance tests for overall behaviour (happy & common sad paths)
    - Mocks take on different roles in these cases

Tests should be **correct**

Mocks considered harmful

- Mocks intentionally diverge the system under test from production
  - You now have two problems
  - Perilously easy to gain false confidence
- If you *really* need to mock, `Mox`, `Mimic` & `ex_vcr` are the way to go
  - Keep your mocks logic free. Input validation and static returns **only**
  - Explicitly test your mocks; they're real code too
- At least `Mox` mandates behaviours. Maybe this will get better with types?

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable**

Tests should be **ubiquitous**

Tests should be **reliable**

Your tests should pass (or fail) 100% of the time

- Non-deterministic tests are a **huge** red flag
  - Is the non-determinism random (almost certainly not)
  - Is it due to isolation (suggests shared resources)
  - Is it due to load / timing (suggests a race condition)
- `--repeat-until-failure 10000` (1.17+) is handy

Tests should be **reliable**

Isolate tests from one another

- Tests within a given module always run sequentially
- Run application anew every test (`start_supervised` in a `setup` block)
- Only one test in the module is ever running so `__MODULE__` is 'safe'
- Use `self()` & `assert_receive` to send messages to the test process
  - This also helps avoid race conditions against async code

```elixir
defmodule PlugTest do
  test "it should send `stop` events for normally completing requests", context do
    TelemetryHelpers.attach_all_events(__MODULE__) |> on_exit()

    Req.get!(context.req, url: "/send_200")

    assert_receive {:telemetry, [:bandit, :request, :start], measurements, metadata}, 500
    assert measurements ~> %{ monotonic_time: integer(roughly: System.monotonic_time()) }
  end
end

defmodule TelemetryHelpers do
  @events [...]

  def attach_all_events(plug) do
    ref = make_ref()
    :telemetry.attach_many(ref, @events, &__MODULE__.handle_event/4, {self(), plug})
    fn -> :telemetry.detach(ref) end
  end

  def handle_event(event, measurements, %{plug: {plug, _}} = metadata, {pid, plug}) do
    send(pid, {:telemetry, event, measurements, metadata})
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```elixir
defmodule PlugTest do
  test "it should send `stop` events for normally completing requests", context do
    TelemetryHelpers.attach_all_events(__MODULE__) |> on_exit()

    Req.get!(context.req, url: "/send_200")

    assert_receive {:telemetry, [:bandit, :request, :start], measurements, metadata}, 500
    assert measurements ~> %{ monotonic_time: integer(roughly: System.monotonic_time()) }
  end
end

defmodule TelemetryHelpers do
  @events [...]

  def attach_all_events(plug) do
    ref = make_ref()
    :telemetry.attach_many(ref, @events, &__MODULE__.handle_event/4, {self(), plug})
    fn -> :telemetry.detach(ref) end
  end

  def handle_event(event, measurements, %{plug: {plug, _}} = metadata, {pid, plug}) do
    send(pid, {:telemetry, event, measurements, metadata})
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

```elixir
defmodule PlugTest do
  test "it should send `stop` events for normally completing requests", context do
    TelemetryHelpers.attach_all_events(__MODULE__) |> on_exit()

    Req.get!(context.req, url: "/send_200")

    assert_receive {:telemetry, [:bandit, :request, :start], measurements, metadata}, 500
    assert measurements ~> %{ monotonic_time: integer(roughly: System.monotonic_time()) }
  end
end

defmodule TelemetryHelpers do
  @events [...]

  def attach_all_events(plug) do
    ref = make_ref()
    :telemetry.attach_many(ref, @events, &__MODULE__.handle_event/4, {self(), plug})
    fn -> :telemetry.detach(ref) end
  end

  def handle_event(event, measurements, %{plug: {plug, _}} = metadata, {pid, plug}) do
    send(pid, {:telemetry, event, measurements, metadata})
  end
end
```

https://github.com/mtrudel/bandit/blob/main/test/bandit/http1/plug_test.exs#L412

Tests should be **reliable**

Quiet & sane test output

- It's 8000x easier to with a test failure when you can see it in isolation
- Tests should be quiet with all output quelled or captured
- `@tag :capture_log` works wonders for this
- If you want to capture log output for tests, `import ExUnit.CaptureLog`
  - Sometimes unavoidable to need a `Process.sleep` in these cases
- Try not to overuse log capture; it can easily hide trouble in otherwise passing tests

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable**

Tests should be **ubiquitous**

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable (& fast)**

Tests should be **ubiquitous**

Tests should be **reliable (& fast)**

Run your tests in parallel

- `use ExUnit.Case async: true`
- Each test file will be run in parallel
- Tests within a single file are always run sequentially
- Overall run speed is limited by your slowest file
  - Prefer more, smaller test files

Bandit's tests ~~should be~~ are fast

8x faster

From ~45s to <6s

https://github.com/mtrudel/bandit/pull/446

Tests should be **clear**

Tests should be **correct**

Tests should be **reliable (& fast)**

Tests should be **ubiquitous**

Tests should be **ubiquitous**

You should be running CI

- GitHub Actions is someone else's computer, but free and amazing
  - Run tests + dialyzer + credo + others on every push to every branch
  - Matrix testing: test on all combinations of recent OTPs and Elixirs (or any other property)
  - GitHub has a bunch of options to gate merges on CI

Tests should be **ubiquitous**

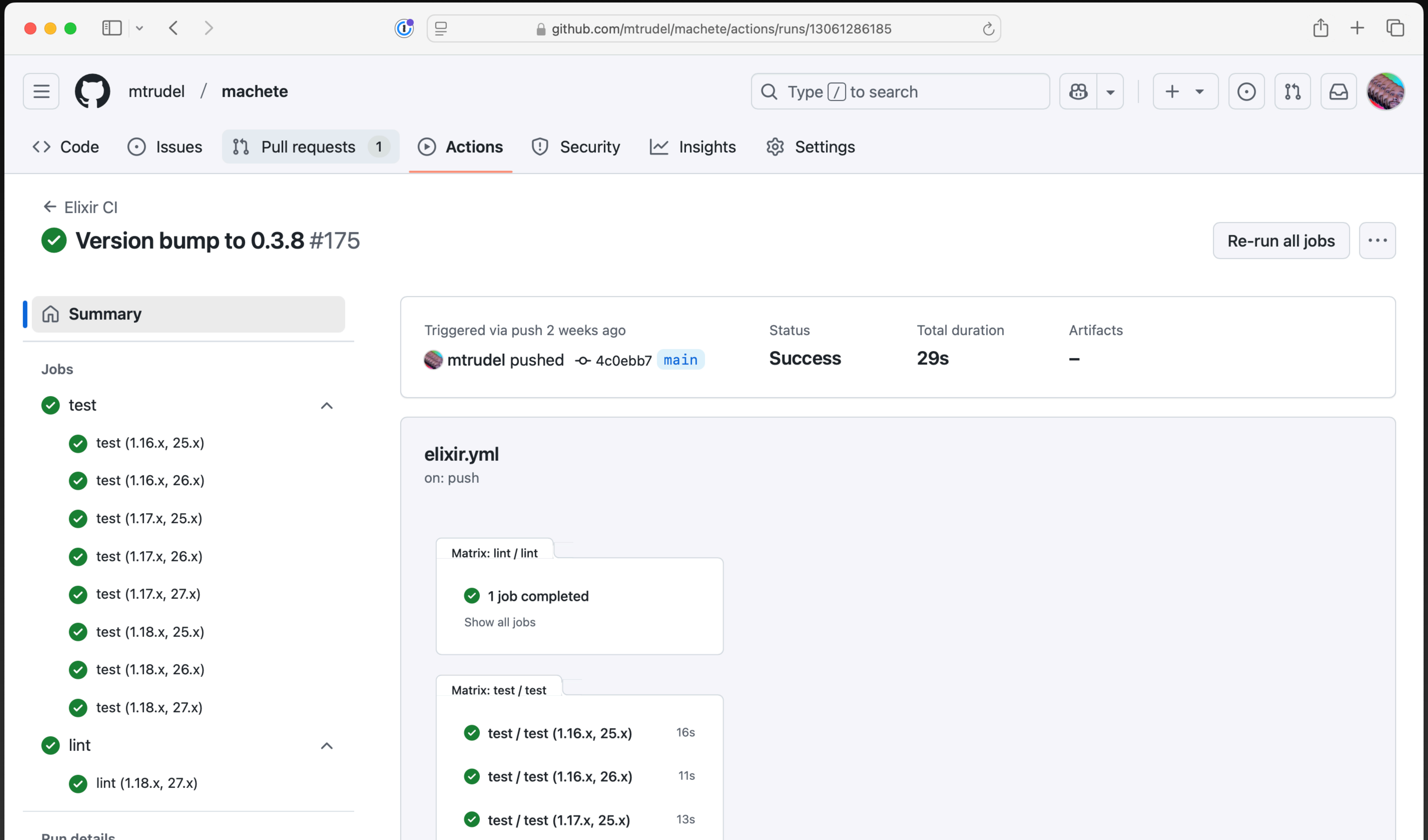
Impossibly easy to get started

```yaml
# Put this in .github/workflows/elixir.yml

name: Elixir CI

on:
  push:
    branches: [ main ]
  pull_request:
  workflow_dispatch:

jobs:
  test:
    uses: mtrudel/elixir-ci-actions/.github/workflows/test.yml@main
  lint:
    uses: mtrudel/elixir-ci-actions/.github/workflows/lint.yml@main
```

https://github.com/mtrudel/elixir-ci-actions

Tests should be **ubiquitous**

Impossibly easy to get started



github.com/mtrudel/machete/actions/runs/13061286185

mtrudel / machete

Type / to search

Code | Issues | Pull requests 1 | Actions | Security | Insights | Settings

← Elixir CI

Version bump to 0.3.8 #175

Re-run all jobs

Summary

Jobs

- test
  - test (1.16.x, 25.x)
  - test (1.16.x, 26.x)
  - test (1.17.x, 25.x)
  - test (1.17.x, 26.x)
  - test (1.17.x, 27.x)
  - test (1.18.x, 25.x)
  - test (1.18.x, 26.x)
  - test (1.18.x, 27.x)
- lint
  - lint (1.18.x, 27.x)

Run details

Triggered via push 2 weeks ago	Status	Total duration	Artifacts
mtrudel pushed 4c0ebb7 `main`	Success	29s	–

elixir.yml
on: push

Matrix: lint / lint

1 job completed
Show all jobs

Matrix: test / test

test / test (1.16.x, 25.x)	16s
test / test (1.16.x, 26.x)	11s
test / test (1.17.x, 25.x)	13s

Tests that are **clear**
Tests that are **correct**
Tests that are **reliable (& fast)**
Tests that are **ubiquitous**

fin