Devlog and thoughts on the `lintestor` project: How to get the most out of the (albeit amateur) development experience

aisuneko icecat @ SZU, PLCT QA team

10/30/2024

`lintestor` - linear tester? for what?

- Rust framework for automated software package availability testing
  - Perhaps the first of its kind?
  - github.com/255doesnotexist/lintestor
- I joined its development along with @255doesnotexist in August
- Watching it gradually grow from an utter toy to something barely usable

Show me a demo:

```
    Running `target/debug/lintestor -tas -D tests/test_files/`
[2024-10-30T02:38:05Z INFO  lintestor] Distros: ["distro_a", "distro_b"]
[2024-10-30T02:38:05Z INFO  lintestor] Packages: ["test1", "test2", "test3", "test3-ill", "test4"]
[2024-10-30T02:38:05Z INFO  lintestor] Running tests
[2024-10-30T02:38:05Z INFO  lintestor] Connection method: None (Locally)
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_a/test1, locally.
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_a/test1
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_a/test2, locally.
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_a/test2
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_a/test3, locally.
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_a/test3
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_a/test3-ill, locally.
[2024-10-30T02:38:05Z WARN  lintestor::testscript_manager] Missing metadata.sh for distro_a/test3-ill, its metadata will not be recorded
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_a/test3-ill
[2024-10-30T02:38:05Z INFO  lintestor] Connection method: None (Locally)
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_b/test1, locally.
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_b/test1
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_b/test2, locally.
[2024-10-30T02:38:05Z INFO  lintestor] Test passed for distro_b/test2
[2024-10-30T02:38:05Z INFO  lintestor] Running test for distro_b/test4, locally.
[2024-10-30T02:38:05Z ERROR lintestor] Test failed for distro_b/test4: Not all tests passed for distro_b/test4
[2024-10-30T02:38:05Z INFO  lintestor] Aggregating reports
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_a/test1/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_a/test2/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_a/test3/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_a/test3-ill/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_b/test1/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_b/test2/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregating /home/aisuneko/src/rust/lintestor/tests/test_files/distro_b/test4/report.json
[2024-10-30T02:38:05Z INFO  lintestor::aggregator] Aggregated report generated at /home/aisuneko/src/rust/lintestor/tests/test_files/reports.json
[2024-10-30T02:38:05Z INFO  lintestor] Generating summary report
[2024-10-30T02:38:05Z INFO  lintestor::markdown_report] Markdown report generated at /home/aisuneko/src/rust/lintestor/tests/test_files/summary.md
```

(this one is from our integration test; the next one is an actual report)

软件包测试结果矩阵 Software package test results

> 图标说明 Legend: ✅ = 通过 Passed; ⚠️ = 部分测试不通过 Not all tests passed; ❌ = 全部测试不通过 All tests failed; ❓ = 未知 Unknown

软件包 Package	种类 Type	debian	bianbu
Apache HTTP Server	Web Server	✅ apache=2.4.62-1	✅ apache=2.4.62-1
Clang C/C++ Compiler	Compiler Toolchain	✅ clang=16.0.6 (27+b1)	✅ clang=16.0.6 (27+b1)
CMake	Build System	✅ cmake=3.30.2	✅ cmake=3.30.2
Docker	Containerization Platform	✅ docker=5:27.1.1-1~~debian.12~~bookworm	✅ docker=5:27.1.1-1~~debian.12~~bookworm
Erlang Programming Language	Programming Language	✅ erlang=1:25.3.2.12+dfsg-1	✅ erlang=1:25.3.2.12+dfsg-1
GNU Compiler Collection	Compiler Toolchain	⚠️ gcc=14.2.0	⚠️ gcc=14.2.0
GNU Debugger	Debugging Tool	✅ gdb=GNU gdb (Debian 15.1-1) 15.1	✅ gdb=GNU gdb (Debian 15.1-1) 15.1
Go Programming Language	Programming Language	✅ golang=2:1.22~3	✅ golang=2:1.22~3
HAProxy	Load Balancer	✅ haproxy=2.9.9-1	✅ haproxy=2.9.9-1
libmemcached	Caching Library	✅ libmemcached=1.1.4-1.1+b1	❌ libmemcached=1.1.4-1.1+b1
Lighttpd	Web Server	✅ lighttpd=1.4.76-1	✅ lighttpd=1.4.76-1
LLVM Compiler Infrastructure	Compiler Toolchain	✅ llvm=1:16.0.6-27+b1	✅ llvm=1:16.0.6-27+b1
MariaDB database server	Database	✅ mariadb=1:11.4.3-1	✅ mariadb=1:11.4.3-1
Nginx	Web Server	✅ nginx=1.26.0-2	✅ nginx=1.26.0-2
Node.js	Javascript Runtime	✅ nodejs=20.16.0+dfsg-1	✅ nodejs=20.16.0+dfsg-1

Let's talk about project structure

I mean source code:

```
src/
├── aggregator.rs
├── config
│   ├── connection_config.rs
│   ├── distro_config.rs
│   └── mod.rs                # really necessary?
├── main.rs
├── markdown_report.rs
├── testenv_manager.rs
├── test_runner
│   ├── local.rs
│   ├── mod.rs
│   └── remote.rs
├── testscript_manager.rs
└── utils.rs
```

... No I mean how the *tests* are organized

e.g. in a **working directory**:

```
cwd/
├── distro1
│   ├── package1
│   │   ├── metadata.sh         # more on that later
│   │   ├── test1.sh
│   │   ├── test2.sh
│   │   ├── ...
│   ├── package2
│   ├── ...
│   └── prerequisite.sh (optional), startup/stop scripts
├── distro2
└── ...
```

Testing environment connection methods

- None (Locally), Remote (SSH), Remote (QEMU)
- This is also reflected in our design for distro configs:

```
enabled = true
testing_type = "qemu-based-remote"
startup_script = "./debian/start_qemu.sh"
stop_script = "./debian/stop_qemu.sh"
skip_packages = ["docker"]

[connection]
method = "ssh"
ip = "localhost"
port = 2222
username = "root"
password = "root"
```

- Bug: config fails to parse when [connection] isn't present (should be optional)
- What is wrong with the following code?

```rust
#[derive(Debug, serde::Deserialize)]
pub struct DistroConfig {
    pub enabled: bool,
    pub testing_type: String, // 'locally' or 'remote' or 'qemu-based-remote'
    #[serde(rename = "startup_script")]
    #[serde(default, skip_serializing_if = "is_not_qemu_based_remote")]
    pub startup_script: String,

    #[serde(rename = "stop_script")]
    #[serde(default, skip_serializing_if = "is_not_qemu_based_remote")]
    pub stop_script: String,

    #[serde(rename = "connection")]
    #[serde(default, skip_serializing_if = "is_not_remote")]
    pub connection: ConnectionConfig,

    pub skip_packages: Option<Vec<String>>,
}
```

1. Where's `is_not_qemu_based_remote`?

```rust
#[allow(dead_code)]
fn is_not_qemu_based_remote(value: &String) -> bool {
    // keep this function as it is, just for serde plz
    value != "qemu-based-remote"
}
```

2. use `pub connection: Option<ConnectionConfig>` plz

- `serde`'s macro mechanism is hacky and lacks concise documentation
- https://serde.rs/field-attrs.html

Flag design

`clap-rs` is your best friend

```
Usage: lintestor [OPTIONS]

Options:
  -t, --test                          Run tests (for all distributions by default)
  -a, --aggr                          Aggregate multiple report.json files into a single reports.json
  -s, --summ                          Generate a summary report
  -D, --directory <working_directory>  Specify working directory with preconfigured test files
  -d, --distro <distro>               Specify distributions to test
  -p, --package <package>             Specify packages to test
      --skip-successful               Skip previous successful tests (instead of overwriting their results)
  -h, --help                          Print help
  -V, --version                       Print version
```

What is that `metadata.sh` thingy?

- Our current implementation of **package metadata**
  - Definitely not perfect but it works for now :(

```
PACKAGE_NAME="llvm"
PACKAGE_VERSION=$(dpkg -l | grep $PACKAGE_NAME | head -n 1 | awk '{print $3}')
PACKAGE_PRETTY_NAME="LLVM Compiler Infrastructure"
PACKAGE_TYPE="Compiler Toolchain"
PACKAGE_DESCRIPTION="The LLVM Compiler Infrastructure"
```

Why all this nonsense? Because we need those to appear in that Markdown report (aka. the "support matrix")

Okay but how should you parse it then?

- Sad story...
- It isn't simple to expose bash variables to Rust
  - `std::process::Command` is limited as it doesn't provide a **persistent env**
  - No alternatives either;
    `rexpect` `ptyprocess` `subprocess`... none of those crates work
  - We even tried writing our own shell implementation

We ended up with a very stupid approach:

```rust
let metadata_command = format!(
    "source {} && echo $PACKAGE_VERSION && echo $PACKAGE_PRETTY_NAME && echo $PACKAGE_TYPE && echo $PACKAGE_DESCRIPTION",
    metadata_script);
let metadata_output = Command::new("bash").arg("-c").arg(metadata_command).output()?;
let metadata_vec: Vec<String> = String::from_utf8_lossy(&metadata_output.stdout)
    .lines().map(|line| line.to_string()).collect();
if let [version, pretty_name, package_type, description] = &metadata_vec[..] {
    PackageMetadata {
        package_version: version.to_owned(),
        package_pretty_name: pretty_name.to_owned(),
        package_type: package_type.to_owned(),
        package_description: description.to_owned(),
    }
} else {
    panic!("Unexpected metadata format: not enough elements in metadata_vec");
}
```

Thoughts on the organization of test files

- They are all tons of `.sh` scripts that are tricky to manage!
  - ~~So why not just a simple wrapper and get away with it?~~
- Supported distros at the moment: `debian` `bianbu` `openkylin`
  - And we have to port our scripts to each new distros explicitly

Q: how to deal with fixed "config params"?

```rust
pub fn many_of_the_functions_used_in_lintestor(
    distros: &str,
    packages: &str,
    skip_successful: bool,
    dir: &Path,
    ...
)
```

How to reduce code redundancy?

1. Builder pattern (the Rust way?)
2. Pin several fields explicitly (our current approach)
3. construct a global `Config` or `Context` class
   aka "Parameter Object / Options Object" pattern

CI/CD

- Generate documentation
- `cargo clippy -- -D warnings`
- Publish releases
  - Oh shoot we're still living on nightly
- Automated tests

CI/CD: multi-arch builds

- Building failed for OpenSSL when attempting to cross-compile to riscv64gc with `cross-rs`: `crypto/riscv64cpuid.s:67: Error: Instruction csrr requires absolute expression`
- openssl/openssl#23011
- `docker run -it ghcr.io/cross-rs/riscv64gc-unknown-linux-gnu /bin/bash && gcc -v` yields...
  `gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)`
- cross-rs/cross#229 "Remove OpenSSL"
  - okay looks like they deprecated it
  - we scrapped `cross` and built a docker image on our own instead

CI/CD: ~~Unit~~ Integration Tests

```rust
use assert_cmd::Command;
#[test]
fn integration_test() {
    let mut cmd = Command::cargo_bin(env!("CARGO_PKG_NAME")).unwrap();
    let output = cmd.arg("-tas").arg("-D").arg("tests/test_files")
    .env("RUST_LOG", "debug").output().expect("failed to execute process");
    // assert and outputs...
}
```

Unit tests are not possible at the moment:

> If our project is a binary crate that only contains a src/main.rs file and doesn't have a src/lib.rs file, we can't create integration tests in the tests directory and bring functions defined in the src/main.rs file into scope with a use statement. Only library crates expose functions that other crates can use; binary crates are meant to be run on their own.
> https://doc.rust-lang.org/book/ch11-03-test-organization.html

CI/CD: ~~Code~~ Documentation Coverage

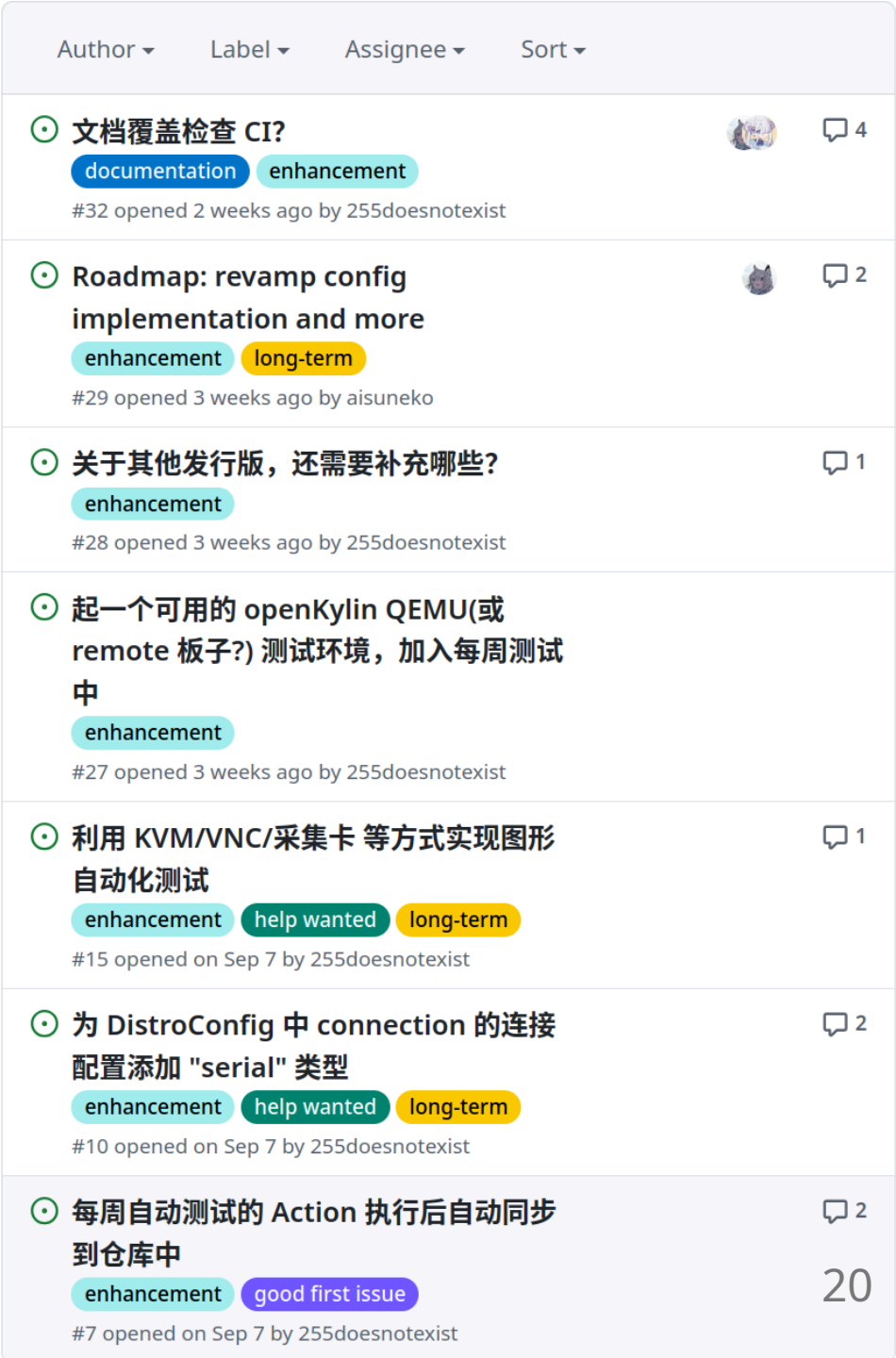
- Code Coverage is also not possible due to the reason above
  - Should we refactor the code to allow more extensive testing?
- Documentation Coverage could be realized by (nightly toolchain required):

  ```
  RUSTDOCFLAGS="-Z unstable-options --show-coverage" cargo doc --no-deps
  ```

- Speaking of documentation, it looks like we are striving to provide comprehensive docs as if we are a library project ^^
  - https://255doesnotexist.github.io/lintestor/

Collaboration

- Good ol' Issues (https://github.com/255doesnotexist/lintestor/issues)
  - Issues (X) Team Kanban (O)
- Random feature proposals and interest-driven development
- Use of `dependabot` and `coderabbitai` also helped ;)



Conclusion / Reflection

- `lintestor` is about usable, yet immature and perhaps bug-prone
  - e.g. we literally hardcoded filesystem paths (with `format!()`) before switching to `Path` and `PathBuf`
- A lot of refactor and overhaul has been done before it reached the aforementioned "barely usable" state
- (almost) no predefined roadmaps nor deadlines
- *Still, we learned a lot about writing real-world Rust projects!*

Thanks for listening :)

Feel free to check out my GitHub page...
or mailto:iceneko@protonmail.ch