tmt

test management tool

•••

miloš prchlík & petr šplíchal

we

- petr šplíchal
  - psss / psplicha@redhat.com
  - principal software quality engineer
- miloš prchlík
  - happz / mprchlik@redhat.com
  - principal software quality engineer
- what are we doing?
  - testing the operating system
  - improving testing tools & processes

roots

at the beginning...

- there were ugly makefiles
- dependency on beaker
- and the test case management system
- and another test case management system
- and possibly yet another test case management system
- different ways how tests were executed
- and how they were enabled in ci
- hard to debug test failures

makefile

```
export TEST=/tmt/smoke
export TESTVERSION=1.0

BUILT_FILES=

FILES=$(METADATA) runtest.sh Makefile PURPOSE

.PHONY: all install download clean

run: $(FILES) build
	./runtest.sh

build: $(BUILT_FILES)
	test -x runtest.sh || chmod a+x runtest.sh

clean:
	rm -f *~ $(BUILT_FILES)

include /usr/share/rhts/lib/rhts-make.include
```

makefile

```
$(METADATA): Makefile
	@echo "Owner:             Petr Splichal <psplicha@redhat.com>" > $(METADATA)
	@echo "Name:              $(TEST)" >> $(METADATA)
	@echo "TestVersion:       $(TESTVERSION)" >> $(METADATA)
	@echo "Path:              $(TEST_DIR)" >> $(METADATA)
	@echo "Description:       Simple smoke test" >> $(METADATA)
	@echo "Type:              Sanity" >> $(METADATA)
	@echo "TestTime:          5m" >> $(METADATA)
	@echo "RunFor:            tmt" >> $(METADATA)
	@echo "Requires:          tmt" >> $(METADATA)
	@echo "RhtsRequires:      fmf" >> $(METADATA)
	@echo "Priority:          Normal" >> $(METADATA)
	@echo "License:           MIT" >> $(METADATA)
	@echo "Confidential:      no" >> $(METADATA)
	@echo "Destructive:       no" >> $(METADATA)

	rhts-lint $(METADATA)
```

we had a dream...

- easily open source tests
- prevent code duplication
- minimize maintenance
- easily develop tests
- reproduce issues
- execute in preferred environment
- flexible test execution metadata
- stored at a single place
- unified, concise and human-friendly
- simple

main.fmf

```
summary: Verify that test debugging works fine
description:
    Prepare the environment for testing using provision & prepare,
    then debug test code using repeated discover & execute.
contact: Petr Splichal <psplicha@redhat.com>
test: ./test.sh
duration: 5m
component: tmt
require: tmt
recommend: fmf
```

main.fmf

```
summary: Verify that test debugging works fine
description:
    Prepare the environment for testing using provision & prepare,
    then debug test code using repeated discover & execute.
```

trees

fmf

- flexible metadata format
  - data stored in yaml
  - inheritance
  - elasticity
- python module
  - `fmf.Tree()`
- command line tool
  - fmf init
  - fmf ls
  - fmf show
  - ...

inheritance

main.fmf

```
require: [wget, httpd]
tier: 1

/download:
    description: Check basic download options
    test: ./test.sh --simple
    duration: 3m

/recursion:
    description: Check recursive download options
    test: ./test.sh --recursive
    duration: 20m
```

elasticity

main.fmf

```
require: [wget, httpd]
tier: 1
```

download.fmf

```
description: Check basic download options
test: ./test.sh --simple
duration: 3m
```

recursion.fmf

```
description: Check recursive download options
test: ./test.sh --recursive
duration: 20m
```

tmt

- metadata specification
  - core
  - tests
  - plans
  - stories
- python module
  - `tmt.Tree()`
- command line tool
  - tmt test ls
  - tmt plan show
  - tmt story create
  - tmt run
  - ...

specification

- core attributes such as summary or description common across all levels
- tests define attributes which are closely related to individual test cases such as test script, framework or maximum test duration
- plans group relevant tests and enable them in the ci, they describe how to discover tests for execution, how to provision the environment, how to prepare it for testing, how to execute tests and report test results
- stories can be used to track implementation, test and documentation coverage for individual features or requirements

branches

inherit common stuff

```
├── plans
│   ├── main.fmf
│   ├── features
│   ├── install
│   ├── provision
│   └── sanity
├── spec
│   ├── main.fmf
│   ├── core
│   ├── plans
│   ├── stories
│   └── tests
└── tests
    ├── main.fmf
    ├── login
    ├── run
    ├── status
    └── unit
```

leaves

tests in leaves

```
├── install
│   ├── data
│   ├── main.fmf
│   └── test.sh
├── recommend
│   ├── data
│   ├── main.fmf
│   └── test.sh
├── require
│   ├── data
│   ├── main.fmf
│   └── test.sh
└── shell
    ├── data
    ├── simple.fmf
    ├── complex.fmf
    └── test.sh
```

testing farm

- testing system as a service
  - docs.testing-farm.io
- integration enabled across
  - packit
  - github
  - gitlab
  - fedora ci
  - centos stream ci
  - rhel ci

demo

code

plugins & options

- command line options
  - handle options for individual plugins based on `--how`
  - `create_method_class()`
- storing & loading plugin data
  - fmf + command line
  - from `dict` to `dataclasses`: `StepData` class
  - validation: `ValidateFmfMixin`
  - normalization: `NormalizeKeysMixin`
  - serialization: `SerializableContainer`

type annotations

- how it went?
  - gradual addition with `--files`
  - large central packages - `tmt.base`, `tmt.utils` - take sooo long to annotate
- what's left?
  - `Any`
  - type-less getters: `get()` and `opt()`
  - iterating over many tests/plans/stories: `Generator`
  - stricter parent/child checking with `Generic`

future

multi host tests

- teemtee/tmt/pull/1790
- everything, everywhere, at once...
  - server vs clients, one plan on different architectures, fedora server vs `podman` client, ...
- traps
  - single thread of execution explodes
  - step and test synchronization
  - reporting

hardware requirements

- teemtee/tmt/pull/1316
- common specification over many different services
  - openstack, beaker, aws, azure, google cloud, `libvirt`, `podman`, ...

```
provision:
  how: ...
  hardware:
      boot:
          method: uefi
      cpu:
          model-name: Haswell .+
          cores: >= 8
      virtualization:
          is-vitualized: true
```

tmt try

- wizard like mode for quick experimenting
- with simplified syntax for the most common use cases

```
# quickly debug a test failure
cd tests/feature
tmt try fedora

# provision a guest for experimenting
tmt try centos-stream-9
```

... and more

- consistent test environment in ci and in a vm on my laptop
- all mighty prepare install plugin
- handling environment combinations

questions?

links

- docs
  - tmt.readthedocs.io
  - fmf.readthedocs.io
  - docs.testing-farm.io
  - packit.dev
- talks
  - enjoy creating, executing and enabling tests using tmt — nest with fedora 2020
  - easily enable tests in fedora, rhel, github — devconf.cz 2021
  - tmt = freedom for tests + comfort for users — devconf.cz 2022

thanks!