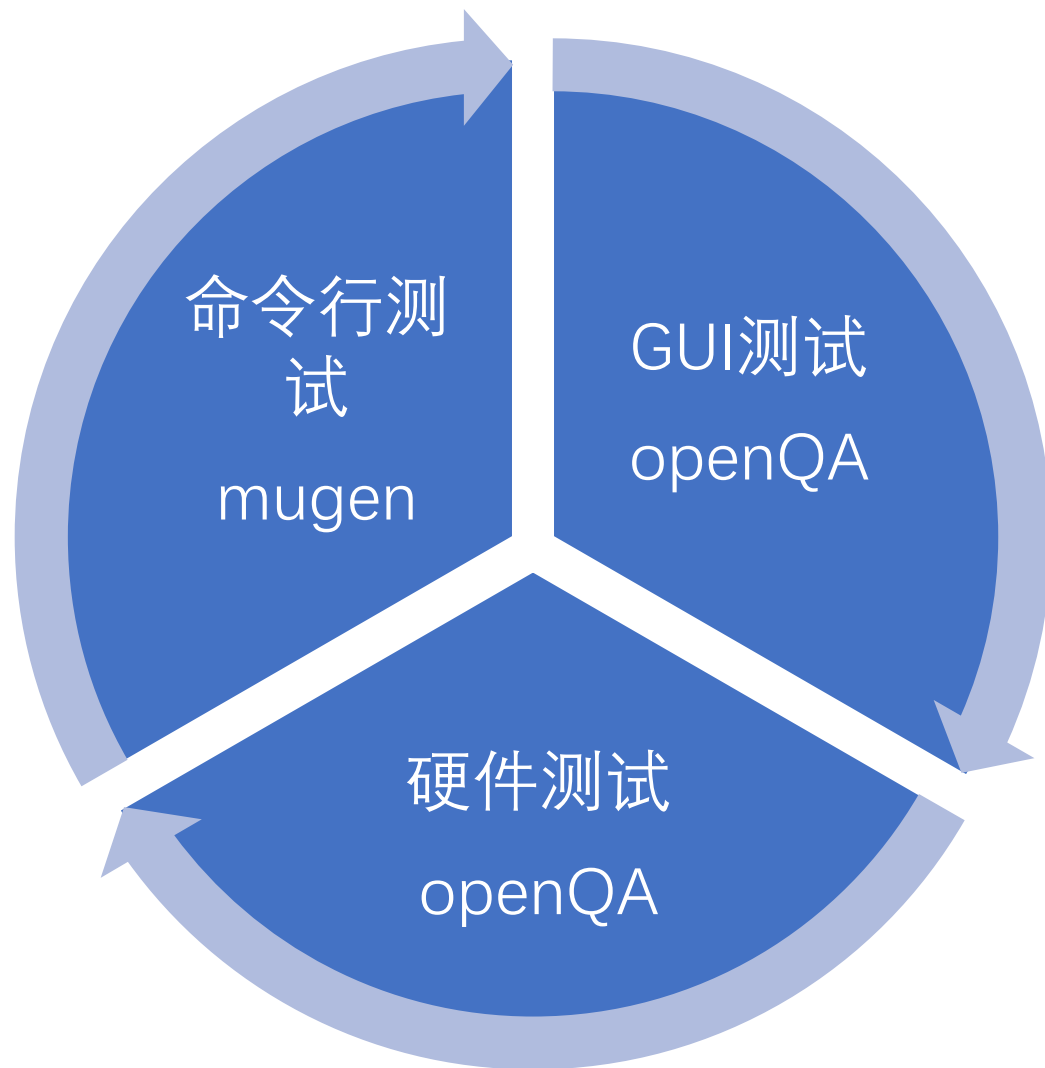


# 开源自动化测试工具Mugen和openQA在 openEuler RISC-V测试中的使用

2024年1月13日

## 测试工具的开发



# 测试环境的基础设施建设



PLCT云测试中心

命令行

GUI

硬件  
(启动和内核)

# 目 录

01

openEuler RISC-V 测试

02

Mugen

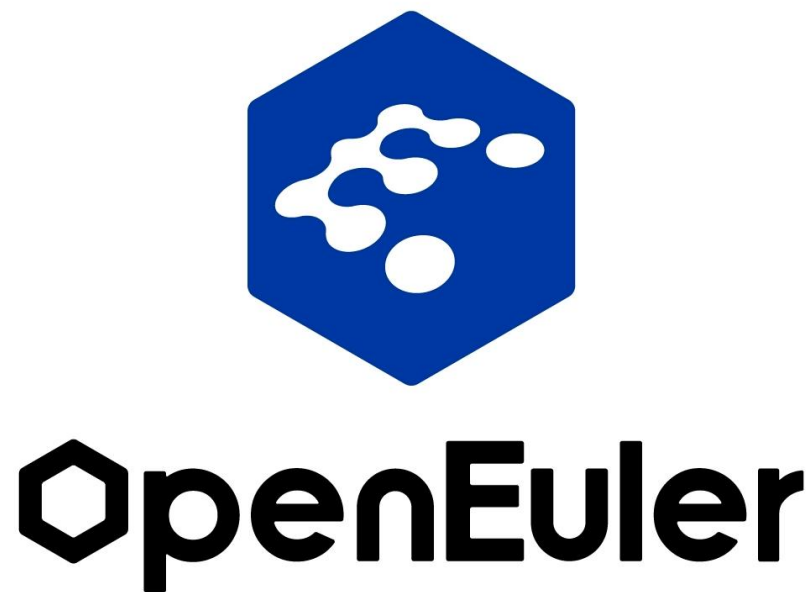
03

openQA

04

未来的构想

openEuler是一款开源、免费的操作系统，由openEuler社区运作。当前openEuler内核源于Linux，支持鲲鹏及其它多种处理器，能够充分释放计算芯片的潜能，是由全球开源贡献者构建的高效、稳定、安全的开源操作系统，适用于数据库、大数据、云计算、人工智能等应用场景。



## openEuler RISC-V 简介

- openEuler在支持RISC-V基础架构方面，已进入世界先进行列。从2020年4月开始进行RISCV-V适配，成立专门的RISC-V SIG组，聚合全球RISC-V上下游开发者和生态伙伴，构建了十分活跃的RISC-V生态，同时也有利地支持了openEuler开源生态的建设。目前，openEuler社区已成为全球在RISC-V基础软件领域应用推广热度最高的社区发行版。
- openEuler RISC-V测试主要对openEuler RISC-V的系统和重要组件测试进行测试。通过结合社区release-manager团队制定的版本计划规划相应的测试活动。整体测试覆盖新需求、继承需求的测试分析和执行，明确各个测试周期的测试策略及出入口标准。其中23.09 openEuler RISC-V baseOS版本已于2023年合入主线。

# 01

## openEuler RISC-V 测试

### 1.1

测试策略

### 1.2

开源测试工具的使用

### 1.3

问题和挑战

## 测试策略

openEuler 24.03 RISC-V 按照社区 release-manger 团队既定的版本计划，共有 5 轮测试，按照社区研发模式，所有的需求已在拉分支前完成合入，因此本次版本测试采取 1+3+1 的测试方式，即 Round 1 发布的 beta 版本可提供外部开发者基本功能（baseOS 编译器）及测试条件（内核），Round 2~4 全量保障本次版本发布所有特性（新增&继承）以及其他能力，Round 5 进行回归测试。

测试轮次	测试内容
Round1	功能测试 编译器测试 内核测试
Round2	功能测试 长稳测试
Round3	功能测试 长稳测试 文档测试 问题单回归
Round4	功能测试 长稳测试 文档测试 问题单回归
Round5	功能测试 问题单全量回归 文档测试



# 测试能力执行

序号	测试类型	测试项目	在riscv64设备上是否可以测试	无法测试的原因	2309版本是否测试	2309版本未测试的原因	2309测试通过标准
1	功能测试	mugen	Yes		Yes		1.全部Pass2.对于不支持/非问题的失败给出解释，QA SIG评审通过3.对于失败项目，确定影响范围，QA SIG 评审
		LTP	Yes		Yes		1.全部Pass2.对于不支持/非问题的失败给出解释，QA SIG评审通过3.对于失败项目，确定影响范围，QA SIG 评审
2	基础性能测试	unixbench	Yes		No	2309版本待测设备是qemu，经和QA SIG沟通后无需执行性能测试；创新版本不涉及性能规格（同上游测试策略）	
		netperf	Yes		No		
		iozone	Yes		No（24.03 新增，SG2042 测试可用）		
		fio	Yes		No		
		stream	Yes		No		
		lmbench	Yes*	+需要对 scripts/gnu-os 打 patch / 直接替换支持 riscv64 的最新版本	No		
3	DFX测试				No		
4	安全测试	oss-fuzz	No	安装部署依赖的 Docker 镜像缺少 riscv64 架构适配	No	2309版本待测设备是qemu，经和QA SIG沟通后无需执行安全测试	
		syzkaller	Yes*	+执行速度过慢	No		
		nmap	Yes		No		
		mugen/security_test	Yes		No		
5	虚拟化测试	virttest-avocado-vt	<a href="#">Yes（详见此处）</a>	目前没有支持部署测试工具的硬件设备(支持gcc H扩展)，目前只有 QEMU 支持，效率过低	No	2309版本待测设备是qemu，经和QA SIG沟通后无需执行安全测试	
6	内核测试	trinity	Yes	<a href="https://gitee.com/openeuler/RISC-V/issues/I7IIBH">虚拟机环境下测试不通过，https://gitee.com/openeuler/RISC-V/issues/I7IIBH</a>	No	2309版本待测设备是qemu，而 trinity 虚拟机环境下测试不可靠	
7	接口测试	api sanity checker	NA	oe社区门禁测试	NA	oe社区门禁测试	
8	长稳测试	long stress(ltp stress)	Yes		Yes		连续执行24小时测试，不出现异常
9	GUI测试	openQA	Yes		No	QA SIG 不要求	
10	编译器测试	dejagnu	Yes		Yes		全部 Pass；失败用例给出原因/影响范围，QA 评审
		llvmcase	No	23.09 版本未将 LLVM 作为支持特性	No	23.09 版本未将 LLVM 作为支持特性；QA 提供的测试步骤为使用 GCC 编译 LLVM 本身，实际上在 OBS 构建 LLVM 包时已经覆盖，OBS 构建成功&软件包打出来了 = 此项测试通过	
		Anghabench	Yes		Yes		
		jotai	<a href="#">Yes</a>		No	<a href="https://gitee.com/yunxiangluo/openeuler-riscv-2303-test/tree/master/BasicTest/%E7%BC%96%E8%AF%91%E5%99%A8%E6%B5%8B%E8%AF%95%jotai">与 AnghaBench 功能重叠，采用的测试源代码库也是相同的https://gitee.com/yunxiangluo/openeuler-riscv-2303-test/tree/master/BasicTest/%E7%BC%96%E8%AF%91%E5%99%A8%E6%B5%8B%E8%AF%95%jotai</a>	
		csmith	Yes*	QA 测试能力执行指南要求百万量级测试用例，csmith 默认为单线程执行，效率过低，需要更改测试执行方式	Yes		
		yarpgen	Yes*	同 csmith	No		
11	自编译测试						
12	北向兼容性测试	oecp	No	测试工具未适配riscv64	No	测试工具未适配riscv64	
13	南向兼容性测试	oech-ci	No	测试工具未适配riscv64	No	测试工具未适配riscv64	
14	软件包专项测试		NA	在 obs 包构建过程中已进行测试	NA	在 obs 包构建过程中已进行测试	软件包可正常安装卸载
15	资料测试		Yes		Yes		文档内容正确无误
16	特性测试（继承特性+新特性）	各项目SIG独立负责，测试方法自定义	NA		No	测试范围未baseOS，不涉及相关项目	各SIG提交测试报告，QA SIG评审通过

# 开源测试工具的使用

## 基础性能测试

- Unixbench
- Netperf
- Fio
- Stream
- Lmbench

## 内核及安全测试

- LTP
- Trinity
- nmap
- syzkaller

## 编译器测试

- anghaBench
- csmith
- dejagnu
- jotai
- yarpgen

## 自动化测试

- Mugen
- openQA

# 开源测试工具的使用

测试名称	测试工具	主要功能
基础性能测试	Unixbench	测试性能 (CPU为主)
	Netperf	测试网络性能
	Fio	对磁盘进行压力测试和性能验证
	Stream	测试内存带宽性能
	Lmbench	评价系统综合性能（带宽，网络内存为主）

# 开源测试工具的使用

- Unixbench

Unixbench是一个用于测试系统性能的开源工具，也是一个比较通用的benchmark, 测试包含了系统调用、读写、进程、管道、运算、c库等系统基本性能，以及一些简单的2D、3D图形测试。

- 操作步骤

```
git clone https://github.com/kdlucas/byte-unixbench
cd byte-unixbench/UnixBench
make -j$(nproc)
./Run -c $(nproc)
```

- 与上游测试方式不同之处

- unixbench 直接拉取 GitHub 主线最新源码
- 未修改最大线程数

# 开源测试工具的使用

## • Fio

Fio是Linux下开源的一款IOPS测试工具，主要用来对磁盘进行压力测试和性能验证。它可以产生许多线程或进程来执行用户特定类型的I/O操作，相当于是一个多线程的IO生成工具，用于生成多种IO模式来测试硬盘设备的性能。

硬盘I/O测试主要有以下四个类型：随机读，随机写，顺序读，顺序写

## • 操作步骤

```
cat << 'EOF' > fio.sh
#!/bin/bash
numjobs=10
iodepth=10
mkdir test
for rw in read write randread randwrite randrw;do
for bs in 4 16 32 64 128 256 512 1024;do
if [ $rw == "randrw" ];then
fio -filename=test/fio -direct=1 -iodepth ${iodepth} -thread -rw=$rw -rwmixread=70 -ioengine=libaio
-bs=${bs}k -size=1G -numjobs=${numjobs} -runtime=30 -group_reporting -name=job1
else
fio -filename=test/fio -direct=1 -iodepth ${iodepth} -thread -rw=$rw -ioengine=libaio -bs=${bs}k -
size=1G -numjobs=${numjobs} -runtime=30 -group_reportin -name=job1
fi
sleep 30
done
done
EOF
bash fio.sh
```

## • 与上游测试方式不同之处

- fio 从软件源直接获取而不是编译安装。
- 测试文件大小改为 1G。

# 开源测试工具的使用

- Lmbench

Lmbench是一款评价系统综合性能的多平台开源工具，它简易且可移植性强。Lmbench主要对文件读写，进程创建销毁开销，网络建立，内存操作等性能进行测试。它主要衡量系统的两个关键特征，分别为反应时间和带宽。

- 操作步骤

```
sudo dnf install -y lmbench libtirpc libtirpc-devel
cp -R /opt/lmbench .
sudo chown -R openeuler:openeuler lmbench
wget https://git.savannah.gnu.org/cgit/config.git/plain/config.guess
cp -f config.guess lmbench/scripts/gnu-os
cd lmbench
make -j$(nproc)
make results
```

- 与上游测试方式不同之处

- lmbench 软件包为从软件源获取，而非直接编译安装。
- 内存测试范围选定为 1024M。部分小内存开发板设置更低值。
- 使用新版本的 config.guess 替换了 scripts/gnu-os 这一系统检测脚本，lmbench 附带的版本过旧不支持 RISC-V 架构，直接运行会导致报错。
- 使用 make see, make summary, make percent 查看详细统计结果。

# 开源测试工具的使用

- ## Netperf

Netperf是一种网络性能测试工具，主要基于TCP或UDP的传输。netperf根据应用的不同，可以进行不同模式的网络性能测试，即批量数据传输模式和请求/应答模式。可以测量TCP和UDP传输的吞吐量、时延、CPU 占用率等性能参数。Netperf测试结果所反映的是一个系统能够以多快的速度向另一个系统发送数据，以及另一个系统能够以多快的速度接收数据。

- ## 操作步骤

在服务器中运行下列指令启动服务器。

```
dnf install netperf -y
netserver
```

客户端运行右侧脚本开始测试。

```
bash client.sh {Server-IP}
```

```
<client.sh>
#!/bin/bash
host_ip=$1
for i in 1 64 128 256 512 1024 1500 2048 4096 9000 16384 32768
65536;do
./netperf -t TCP_STREAM -H $host_ip -l 60 -- -m $i
done
for i in 1 64 128 256 512 1024 1500 2048 4096 9000 16384 32768;do
./netperf -t UDP_STREAM -H $host_ip -l 60 -- -m $i -R 1
done
./netperf -t TCP_RR -H $host_ip
./netperf -t TCP_CRR -H $host_ip
./netperf -t UDP_RR -H $host_ip
```

# 开源测试工具的使用

- **Stream**

Stream是内存带宽性能测试的基准工具，它支持复制（Copy），尺寸变换（Scale），矢量求和（Add），复合矢量求和（Triad）四种运算方式测试内存带宽的性能。

- **操作步骤**

```
git clone
https://gitee.com/thesamename/STREAM.git
cd STREAM
sudo dnf install -y gcc gfortran
sed -i "s/CC =.*/CC = gcc/" Makefile
sed -i "s/FC =.*/FC = gfortran/" Makefile
make
./stream_c.exe
```



# 开源测试工具的使用

测试名称	测试工具	主要功能
内核及安全测试	LTP	压力测试判断系统的稳定性和可靠性
	Trinity	对内核 API 进行冒烟测试
	nmap	对端口进行扫描
	syzakaller	对内核进行冒烟测试

# 开源测试工具的使用

- LTP

LTP是由 Linux Test Project 所开发的一套系统测试套件。它基于系统资源的利用率统计开发了一个测试的组合, 为系统提供足够的压力。通过压力测试来判断系统的稳定性和可靠性。

- 操作步骤

```
wget https://github.com/linux-test-  
project/ltp/releases/download/20230127/ltp-full-  
20230127.tar.xz  
tar -xvf ltp-full-20230127.tar.xz  
cd ltp-full-20230127  
make autotools  
./configure --with-bash --with-expect --with-perl --  
with-python  
make  
make install
```

# 开源测试工具的使用

- Trinity

Trinity 是对内核 API 进行冒烟测试的套件。

- 操作步骤

- 从 GitHub 仓库 获取 Trinity 源代码。以非 root 用户使用 `./configure && make -j4` 进行编译后,
- 使用 `./trinity` 运行测试。如需减少日志输出, 节省硬盘空间, 可添加 `-q` 或 `-qq` 参数: `./trinity -qq`
- 运行结束后进入 `tmp` 目录查看日志。其中, `trinity.log` 为主进程的日志, `trinity-child*.log` 为各子进程的日志。

# 开源测试工具的使用

- **nmap**

nmap (Network Mapper) 是一款开源免费的针对大型网络的端口扫描工具，nmap可以检测目标主机是否在线、主机端口开放情况、检测主机运行的服务类型及版本信息、检测操作系统与设备类型等信息。

- **操作步骤**

- 安装依赖包: `sudo apt install -y vde2`
- 利用脚本配置 VDE 网络
- 将两台虚拟机的 `ssh_port` 修改为不同端口以免冲突。
- 在其中一台虚拟机上安装 `nmap` 并运行端口扫描:

# 开源测试工具的使用

- **syzkaller**

syzkaller 是由 Google 开发的内核冒烟测试工具，可以对多种内核进行冒烟测试。

- **操作步骤**

- 获取 Golang 工具链
- 制作 RISC-V 交叉编译工具链
- 安装 QEMU
- 构建 syzkaller
- 配置受测镜像
- 配置 fstab和OpenSSH
- 生成 SSH 密钥对
- 编写 syzkaller 配置文件
- 运行测试

# 开源测试工具的使用

测试名称	测试工具	主要功能
编译器测试	AnghaBench	AnghaBench 包含了一百万个可编译的 C 文件，可用于对 C 编译器进行编译测试。
	csmith	csmith 是适用于编译器的冒烟测试工具，能够生成大量合法的 C 文件作为测试用例，进而测试 C 编译器的标准性
	dejagnu	DejaGnu 是一个用于测试程序的框架, 主要负责测试gcc， g++， gfortran 的测试套
	jotai	jotai 是从 AnghaBench 中抽取修改的可编译可运行的代码测试集
	yarpgen	yarpgen 是由 Intel 编写的编译器冒烟测试套件。调用 clang 和 gcc 分别使用 -O0， -O1， -O2， -O3 与 -Os 对 yarpgen 生成的代码进行 编译运行并对比结果。

## oe 24.03 RISC-V LTS测试问题和挑战

- 独立项目SIG测试任务的归属
- [mugen](#)测试问题现有问题的解决（测试套750+，测试用例5300+）
- 开发板的内核问题
- 测试工具架构不支持
  - 安全测试中oss-fuzz：安装部署依赖的 Docker 镜像缺少 riscv64 架构适配
  - 北向兼容性测试 oecp/oec-application：未适配 riscv64
  - 南向兼容性 oec-hardware/oec-ci：未适配 riscv64
- 虚拟化测试中目前没有支持 H 扩展的硬件设备，目前只有 QEMU 支持，效率过低

参考[openEuler-22.03-LTS-龙芯版测试策略](#)

# 目 录

01

openEuler RISC-V 测试

02

Mugen

03

openQA

04

未来的构想



## 02

## Mugen

### 2.1

Mugen 简介

### 2.2

工作原理和实例

### 2.3

测试使用场景

### 2.4

挑战和尝试

### 2.5

今后的工作

# Mugen 简介

mugen 使用 `bash` (测试用例) 与 `python` (测试功能, 如多线程、QEMU配置) 语言编写框架和测试用例脚本, 在实际系统环境下对软件包进行安装、卸载以及功能测试。测试用例将模拟实际使用场景对软件进行配置、调用并检查结果, 包括但不限于命令行的直接调用测试、`systemd` 服务启停测试、终端输出的检查、日志输出的检查以及网络连通性的检查。mugen 只适用于命令行程程序的测试, 主程序为 `mugen.sh`。

- 官方的介绍

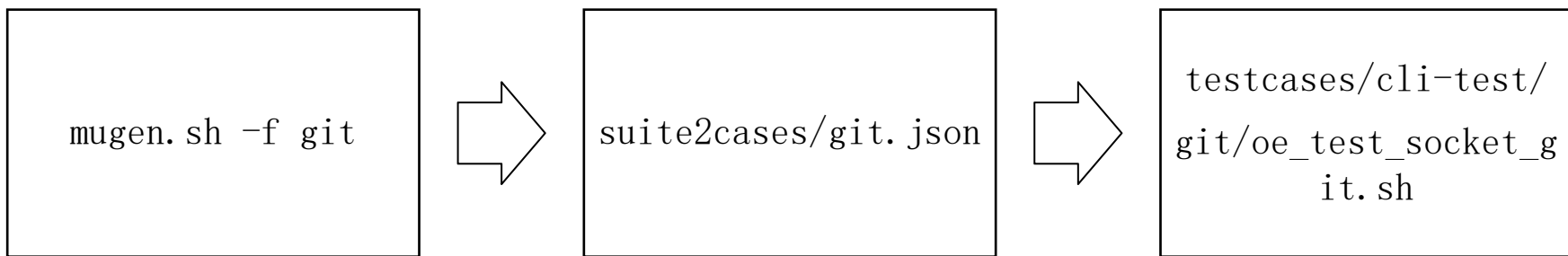
mugen 是 openEuler 社区开放的测试框架, 提供公共配置和方法以便社区开发者进行测试代码的编写和执行。

- 当前的角色

mugen 是 openEuler 上游要求的测试, 其测试结果是软件包质量的一个体现。

## Mugen 工作原理和实例

- 测试套配置



以 git 测试套为例，`mugen.sh -f git` 指定测试 git 测试套，mugen 会寻找对应的测试套配置文件 `suite2cases/git.json`。

测试套的 json 配置文件定义了这个测试套中所有测试用例所在的路径、测试所需的测试机数量，以及每台测试机的磁盘数量、每个磁盘的容量、网卡数量等信息。

## Mugen 工作原理和实例

- 测试套配置实例

```
1 {  
2   "path": "$OET_PATH/testcases/cli-test/git",  
3   "cases": [  
4     {  
5       "name": "oe_test_socket_git"  
6     }  
7   ]  
8 }
```

从 `suite2cases/git.json` 中可以看出，`git` 测试套是一个单用例单测试机的测试，`mugen` 会根据测试用例名到配置指定的目录查找 `bash` 测试脚本（`path/oe_test_socket_git.sh`）。

## Mugen 工作原理和实例

- 测试环境配置

测试环境由 `conf/env.json` 配置文件描述，该文件由 `mugen` 生成，用户需要提供四个参数，其他参数将被自动获取。（`host`主机上执行）

```
$ bash mugen.sh -c --port 22 --user root --password openEuler12#$ --ip 10.0.6.17
```

多台测试机环境的配置只需依次运行命令生成，在配置中虚拟机 ID 将从 1 开始自动累加。

# Mugen 工作原理和实例

- 测试环境配置实例

```
{
  "NODE": [
    {
      "ID": 1,
      "LOCALTION": "local",
      "MACHINE": "physical",
      "IPV6": "fe80::c58d:8909:5272:2dc9",
      "FRAME": "riscv64",
      "NIC": "enp0s2",
      "MAC": "52:54:00:11:45:31",
      "IPV4": "10.0.6.29",
      "USER": "root",
      "PASSWORD": "openEuler12#$",
      "SSH_PORT": 22,
      "BMC_IP": "",
      "BMC_USER": "",
      "BMC_PASSWORD": ""
    },
    {
      "ID": 2,
      "LOCALTION": "remote",
      "MACHINE": "physical",
      "IPV6": "fe80::8961:6f59:4c20:54a5",
      "FRAME": "riscv64",
      "NIC": "enp0s2",
      "MAC": "52:54:00:11:45:28",
      "IPV4": "10.0.6.17",
      "USER": "root",
      "PASSWORD": "openEuler12#$",
      "SSH_PORT": 22,
      "BMC_IP": "",
      "BMC_USER": "",
      "BMC_PASSWORD": ""
    }
  ]
}
```

这是具有 2 个 RISC-V QEMU 实例的测试环境，测试时 `mugen.sh` 测试进程将被运行在 1 号机上，并使用配置中所载的用户名和密码通过 `ssh` 调用 2 号机。

在 x86\_64 KVM 环境下，`MACHINE` 值为 `kvm`。

从 “`MACHINE`” : “`physical`”，可以看出 `mugen` 没有正常识别到 RISC-V QEMU 环境。

## Mugen 工作原理和实例

- 测试用例架构

一个 mugen 测试用例通常有三个部分

- `pre_test`
- `run_test`
- `post_test`

其中 `pre_test` 和 `post_test` 负责测试环境的准备和还原，`run_test` 存放整个测试的测试过程，并在整个过程中插入检查点，判断某个变量值/命令返回值是否符合预期，并可以选择是否提前中断测试。

# Mugen 工作原理和实例

- 测试用例实例

这里给出一个简单的实例： `testcases/cli-test/git/oe_test_socket_git.sh`

```
22 function pre_test() {
23     LOG_INFO "Start environmental preparation."
24     DNF_INSTALL git-daemon
25     LOG_INFO "End of environmental preparation!"
26 }
27
28 function run_test() {
29     LOG_INFO "Start to run test."
32     systemctl reload git.socket 2>&1 | grep "Job type reload is not applicable for unit git.socket"
33     CHECK_RESULT $?
34     systemctl status git.socket | grep "Active: active"
35     CHECK_RESULT $?
36     LOG_INFO "End of the test."
37 }
38
39 function post_test() {
40     LOG_INFO "start environment cleanup."
41     DNF_REMOVE
42     LOG_INFO "Finish environment cleanup!"
43 }
```

最简单的 `pre_test` 与 `post_test` 函数通常分别包含 `DNF_INSTALL` 与 `DNF_REMOVE`，它们在测试前安装测试所需的软件包，并在测试结束后读取软件包安装记录并移除这些软件包。



- 命令行测试用例实例

以 curl 测试套的 oe\_test\_curl\_curl\_001 为例

```
27 function run_test() {
28     LOG_INFO "Start to run test."
29     curl --anyauth -u root:pass http://www.baidu.com
30     CHECK_RESULT $? 0 0 "check curl --anyauth failed"
31     curl -a https://httpbin.com/anything
32     CHECK_RESULT $? 0 0 "check curl -a failed"
33     curl --basic https://httpbin.com/anything
34     CHECK_RESULT $? 0 0 "check curl --basic failed"
35     curl --cacert /etc/pki/tls/certs/ca-bundle.crt https://httpbin.com/anything
36     CHECK_RESULT $? 0 0 "check curl --cacert failed"
37     curl --capath /etc/pki/tls/certs/ https://httpbin.com/
38     CHECK_RESULT $? 0 0 "check curl --capath failed"
39     curl --connect-timeout 1 https://httpbin.com/anything
40     CHECK_RESULT $? 0 0 "check curl --connect-timeoutfailed"
41     curl -C - https://www.baidu.com
42     CHECK_RESULT $? 0 0 "check curl -C failed"
43     curl https://www.baidu.com --cookie "user=root;pass=123456" -v 2>&1 | grep "Set-Cookie"
44     CHECK_RESULT $? 0 0 "check curl https://www.baidu.com failed"
45     curl https://www.baidu.com -c cookiefile && test -f cookiefile
46     CHECK_RESULT $? 0 0 "check curl -c failed"
47     curl --create-dirs https://httpbin.com/anything
48     CHECK_RESULT $? 0 0 "check curl --create-dirs failed"
49     LOG_INFO "End of the test."
50 }
```

测试套对 curl 命令的一些参数进行测试，并插入 CHECK\_RESULT 函数对这些测试点进行检查

CHECK\_RESULT 的第二个参数为预期值，第三个参数代表了判断方式为等还是不等，第四个参数为错误信息。

## Mugen 工作原理和实例

- 命令输出测试用例实例

这种测试用例包括命令行输出的测试和日志输出的测试，以 file 测试套的 file01 用例为例

```
39 function run_test() {
40     LOG_INFO "Start to run test."
41     file -sL "$root_blk" | grep "$root_fs"
42     CHECK_RESULT $? 0 0 "file cannot determin $root_blk file system type."
43     file -b hello.txt | grep "ASCII text"
44     CHECK_RESULT $? 0 0 "file cannot determin hello.txt file type."
45     file -b hello.tar.gz | grep "gzip"
46     CHECK_RESULT $? 0 0 "file cannot determin hello.tar.gz file type."
47     file -b "$(whereis ls | cut -d ' ' -f 2)" | grep "$endian pie executable, $arch"
48     CHECK_RESULT $? 0 0 "file cannot determin $(whereis ls | cut -d ' ' -f 2) file type."
49     LOG_INFO "End to run test."
50 }
```

可以看到每个测试点都是使用 grep 、 awk 等文本处理工具对输出文本进行匹配和判断，随后调用 CHECK\_RESULT 函数检查。

## Mugen 工作原理和实例

- 服务启停测试用例实例

这种测试用例调用 `systemctl` 或 `service` 命令对 `service` 或 `socket` 进行测试这里以 `nginx` 测试套的 `oe_test_service_nginx` 测试为例

```
28 function run_test() {
29     LOG_INFO "Start testing..."
30     test_execution nginx.service
31     systemctl start nginx.service
32     sed -i 's\ExecStart=/usr/sbin/nginx\ExecStart=/usr/sbin/nginx -q\g' /usr/lib/systemd/system/nginx.service
33     systemctl daemon-reload
34     systemctl reload nginx.service
35     CHECK_RESULT $? 0 0 "nginx.service reload failed"
36     systemctl status nginx.service | grep "Active: active"
37     CHECK_RESULT $? 0 0 "nginx.service reload causes the service status to change"
38     LOG_INFO "Finish test!"
39 }
```

测试对 `nginx.service` 进行启停测试，其中 `test_execution` 为 `mugen` 框架定义的函数。

## Mugen 工作原理和实例

- 网络连通性测试用例实例

这种测试用例通常用于测试网络相关的软件，如防火墙。这里以 `firewalld` 测试套的 `oe_test_firewalld_block_icmp` 用例为例

```
28 function run_test() {
29     LOG_INFO "Start executing testcase."
38     SSH_CMD "ping $NODE1_IPV4 -c 1" "${NODE2_IPV4}" "${NODE2_PASSWORD}" "${NODE2_USER}"
39     CHECK_RESULT $?
40     sudo firewall-cmd --add-icmp-block=echo-request
41     SSH_CMD "ping $NODE1_IPV4 -c 1" "${NODE2_IPV4}" "${NODE2_PASSWORD}" "${NODE2_USER}"
42     CHECK_RESULT $? 0 1
43     LOG_INFO "Finish testcase execution."
44 }
```

这种用例通常需要在多测试机环境下执行测试，其中 `NODE2_IPV4` 等变量值为 `conf/env.json` 中定义的值。`SSH_CMD` 函数为 `mugen` 框架定义的函数，使用 `ssh` 登陆到指定测试机后远程执行指定的命令。

## 测试使用场景

- mugen 的使用

最简单的方式使用 mugen ，由于测试套数量众多，最简单的方法就是在待测环境跑一个循环。

```
for t in $(cat mylist); do
    bash mugen.sh -f $t -x
done
```

Problem -> 这样运行 mugen 测试没有考虑两个问题：

- 一些 mugen 测试会对测试环境进行一系列操作，且不见得能还原
- 不同 mugen 测试要求的磁盘数量、测试机数量、网卡数量并不相同

尽管可以简单搭建一个最大化的环境，但使用上面的脚本无法实现测试套之间测试环境的统一，而实际上它也是无法完成所有测试的。

## 测试使用场景

- 基于 QEMU 的自动化测试 [mugen-riscv](https://github.com/brsfl1/mugen-riscv)

### 概述：

在 2309 发版的测试中，向 openEuler 上游索要了它们的测试日志。从日志中看，上游支持的 x86\_64 和 aarch64 两个架构也是使用 QEMU 作为测试环境的。

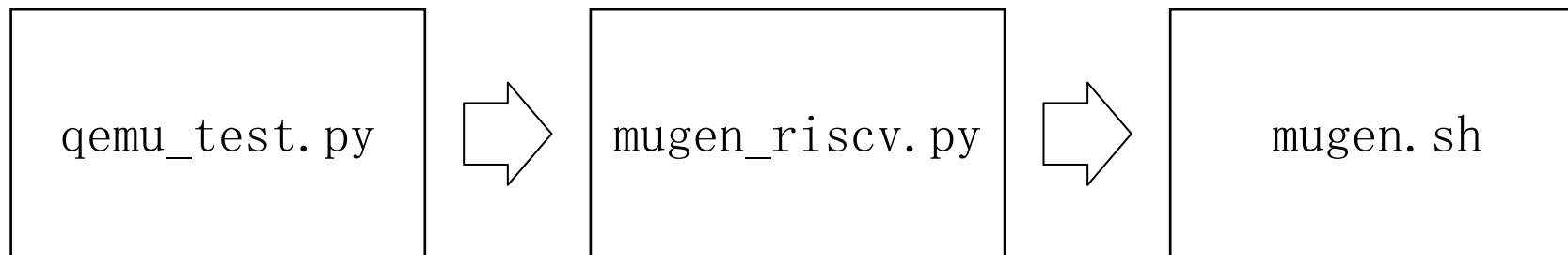
自动化的 mugen 测试需要实现测试环境的自动建立和销毁、测试结果的自动判断和测试日志的自动整理。mugen 本身不具有其中任何一项功能。

第三测试小队自行开发了一套自动化测试工具，以执行上述过程，实现自动化的测试流程。

<https://github.com/brsfl1/mugen-riscv>

## 测试使用场景

- 基于 QEMU 的自动化测试流程



qemu\_test.py 将读取测试套配置（ `suite2cases/*.json` ） 并根据该配置利用 **QEMU 虚拟机** 建立符合该测试套要求的测试环境。

mugen\_riscv.py 在每个测试套完成后， **自动对测试结果进行判断，归类失败日志**，并获取测试机的 `journalctl` 日志和 `systemctl` 日志，保留更多失败信息。

所有测试套完成后由 `result_parser.py` 对用例进行统计，并基于字符串匹配的方法分析日志输出 **csv 表格**。

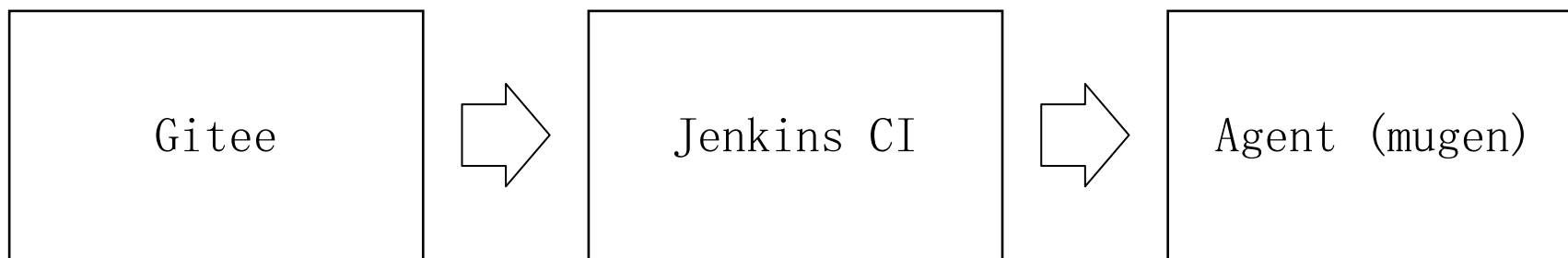


## 测试使用场景

- 基于容器的自动化测试

由于 mugen 很多测试用例事实上不能很好地完成测试环境的还原，在严重时导致系统无法正常进行测试。**直接在实体机**进行测试可能导致大量的刷机操作，进行自动化测试会有一定的困难。

部分在 QEMU 测试缓慢，又不需要内核模块相关操作的测试套可以在容器环境运行。



现有容器测试环境使用了 systemd-nspawn 和 LXC 容器，主要应用在单机测试套的 mugen 自动化测试。由 Jenkins CI 维护自动化测试过程。 <https://jenkins.inuyasha.love/>



## 测试使用场景

- 测试的必要性

软件包自带测试套通过的情况下，是否还需要 mugen 测试

无法互相替代，这两个测试的测试环节不同，相应的测试目的也不一样。从测试环节来说，软件包本身的测试侧重于单元测试，而 mugen 测试则侧重于集成测试和系统测试。软件包自身的测试至多只能保证软件本身的质量，其测试环境与实际运行的环境也有差异。而 mugen 测试不仅保证软件本身的质量，还试图保证二进制构建的质量，软件包打包的质量和系统镜像的质量。在过去的 mugen 测试实践中，发现的问题就有比如软件包打包时少打 .service 文件，又比如镜像制作过程有问题导致预装软件包需要重新安装才能正常使用。

## 测试使用场景

- 测试的必要性

在 QEMU 镜像运行通过的 mugen 测试，是否还有必要在实体机镜像复测

这是有必要的。仅以 RISC-V 架构的测试来说，一些实际处理器具有的特性并不被 QEMU 支持，相关的测试无法在 QEMU 环境进行，这些测试路径并没有被覆盖。在过去的 mugen 测试实践中，有一些无法在 QEMU 环境进行测试，而在实体开发板上测试后发现了软件包质量问题。另外不同的 RISC-V 开发板厂商对 Linux 内核的支持有实际差异，导致不同开发板镜像的内核版本是不一致的。内核不同源，显然就有单独测试的必要；当然在实体机开展测试又需要额外的基础设施支持，这便是另一个问题了。

## 挑战和尝试

- 上游测试套质量问题

在 openEuler 的测试实践中，发现了绝大部多数在 RISC-V 失败的测试套，在 x86\_64 也失败了。

这些测试用例数量大，需要由上游进行修复，而需要持续跟进

## 挑战和尝试

- 实现 RISC-V 适配

在 openEuler 的测试实践中，发现了极少数测试套的失败是架构相关的，即测试用例不兼容 RISC-V 。

对于其中已经明确原因的，我们向上游提交了十几个 pr 。希望在 2403 发版周期能够解决这些架构相关问题。

## 挑战和尝试

- 实验实体机测试

由于支持 H 扩展（RISC-V H-extension）的 RISC-V 处理器尚不存在，而在 x86\_64 上使用 QEMU 虚拟机进行测试有难以解决的性能问题，实体机测试是一个待考虑的解决方案。

```
--privileged \  
-v /dev:/dev \  

```

为了保证每个测试套的初始环境相同，使用实体机在测试前也需要还原系统环境，使用物理磁盘刷机并不现实。一个可行的方案是使用 PXE 网络启动，实际应用可行性还待实验。

## 挑战和尝试

- 突破软件包版本兼容性

软件包版本兼容性问题体现在两个方面

1. 框架对不同版本软件包的兼容性问题
2. 测试套对不同版本软件包的兼容性问题

它们都体现在软件命令行的输出在不同版本有所差别。框架本身对不同版本的 `systemd` 支持不完善导致大量测试失败；大部分测试套的编写也没有考虑被测软件包的版本问题

这个问题与指令集架构无关，故需要推动上游给出明确的修复方法。

## 挑战和尝试

- 突破发行版局限

第三测试小队正在将 mugen 测试应用到 openEuler 以外的测试环境中去，而上游 mugen 是只支持 openEuler 的。

在多种发行版上实现测试，并不是不可能的。实际上进行少量更改（几百行）就可以扩展框架支持范围，当然测试套编写也需要适应这些更改。

在 RUYISDK 的测试中，我们就实现了相同测试流程在 Ubuntu 、 RevyOS 、 Fedora 、 openEuler 的自动化测试

## 今后的工作

今后的工作依然围绕在使用 QEMU 虚拟机进行 mugen 测试

1. 改进自动化测试工具，提升测试效率
2. 向上游提交 issue 和 pr ，解决测试用例不兼容 RISC-V 的问题
3. 向上游提交 issue ， push 上游解决测试用例中架构无关的问题



# 目 录

01

openEuler RISC-V测试

02

Mugen

03

openQA

04

未来的构想

# 03

## openQA

### 3.1

openQA 简介

### 3.2

工作原理和实例

### 3.3

测试场景


### 3.4

挑战 and 我们的尝试

### 3.5

今后的工作

# openQA 简介


 All Tests Job Groups ▾

Login

Welcome to openQA

Life is too short for manual testing!

Learn more »



SUSE

Sponsor

openSUSE Tumbleweed

Build20240110 (about 8 hours ago) 🌟

Build20240109 (a day ago) 🌟

Build20240108 (2 days ago) 🌟

251 passed33 softfail25 failed

252 passed35 softfail24 failed

248 passed32 softfail28 failed

openSUSE Tumbleweed AArch64

Build20240110 (about 10 hours ago) 🗨

Build20240109 (a day ago) 🗨

Build20240108 (2 days ago) 🗨

153 passed29 softfail45 failed

164 passed31 softfail32 failed

166 passed30 softfail30 failed

openSUSE Tumbleweed Kernel

Build20240110 (about 9 hours ago) 🗨

Build20240109 (2 days ago) 🗨

Build20240108 (2 days ago) 🗨

145 passed69 softfail37 failed

144 passed7 softfail46 failed

147 passed9 softfail41 failed

openSUSE Tumbleweed Legacy x86

Build20240110 (about 5 hours ago) 🗨

Build20240109 (2 days ago) 🌟

Build20240108 (2 days ago) 🌟

18 passed4 failed

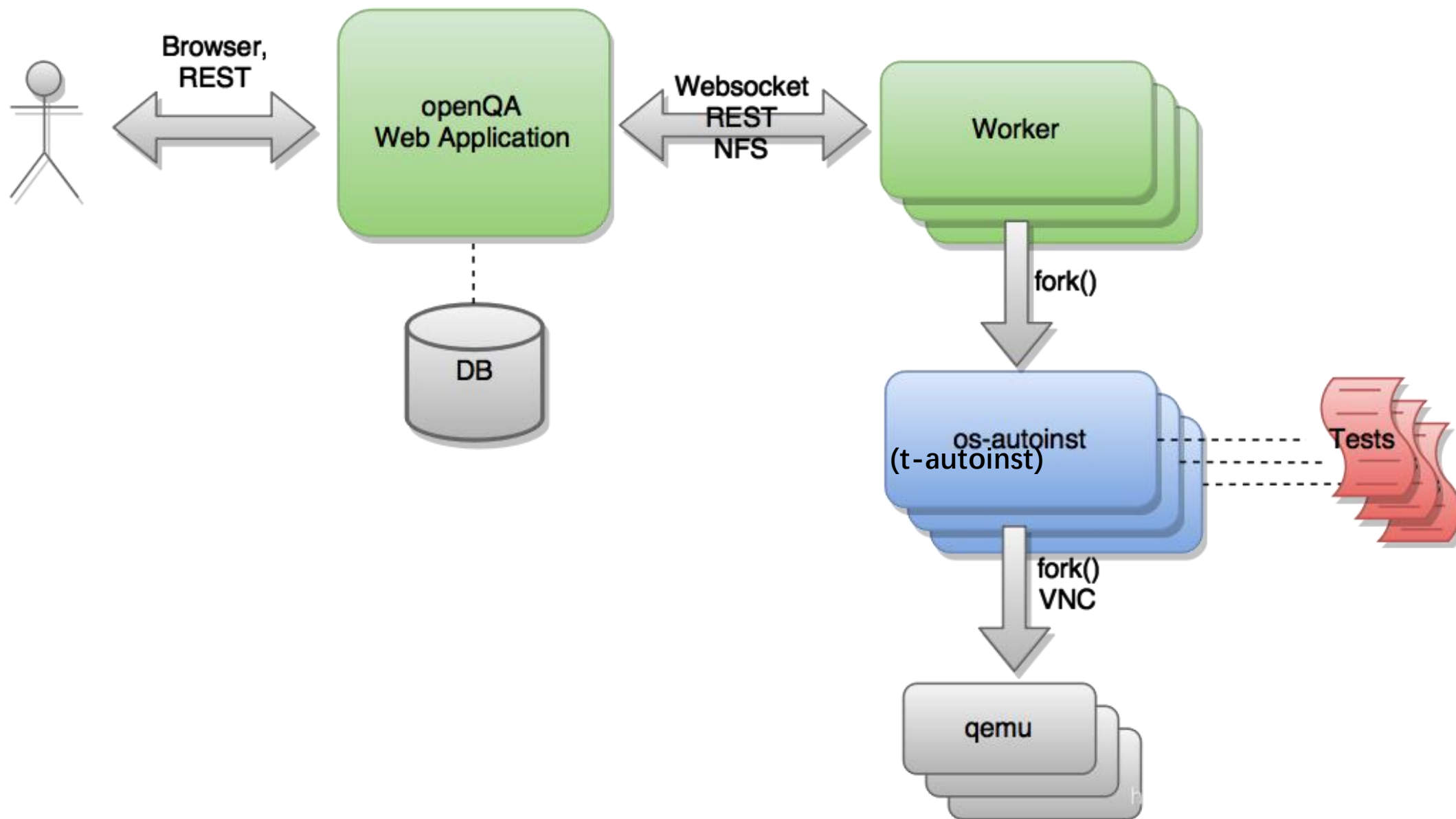
20 passed2 failed

20 passed2 failed

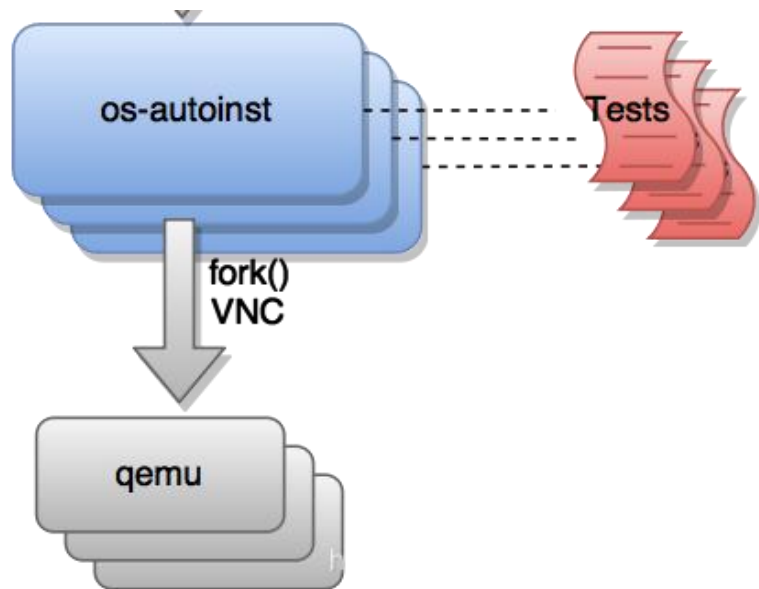
openQA 是一个自动化测试框架，允许一方面测试 GUI 应用程序，另一方面测试引导加载程序和内核。在这两种情况下，都很难编写测试脚本和验证输出。输出可以是弹出窗口，也可以是早期引导中的错误，这个早期甚至在执行 `init` 之前。

与部分软件自带的单元测试相比，openQA 可以模拟真实用户操作自动化进行测试，属于 E2E / 端到端测试，此外，由于用户系统环境与构建环境存在不同，采用模拟用户操作的方式进行测试更贴近最终用户的使用场景，可以排查其他测试中难以发现的错误，包括一些不存在于测试目标软件本身，而是系统环境中的问题。

## openQA 工作原理



## openQA 工作原理



os-autoinst 定义了一系列 console 终端  
大致分为以下两类：

1. serial

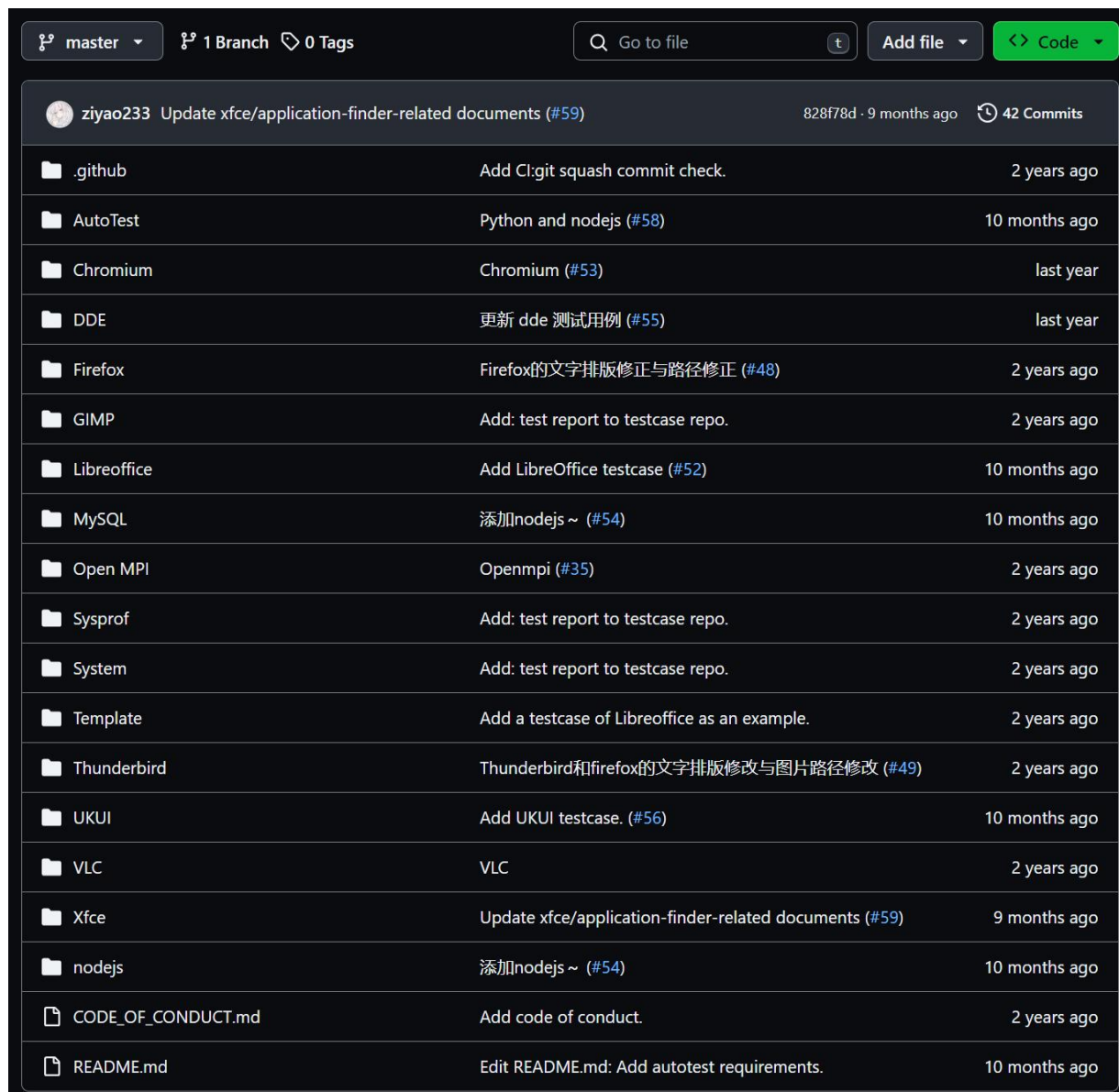
- 串口
- ssh 终端

2. video\_base

- vnc
- ffmpeg 录屏

os-autoinst 通过 console 与测试机器交互，例如读写串口，向 qemu（运行有 vnc server）发送符合 RFB 协议请求模拟鼠标键盘操作

# 测试用例库建设



The screenshot shows a GitHub repository interface. At the top, it says 'master', '1 Branch', and '0 Tags'. There is a search bar 'Go to file', an 'Add file' button, and a 'Code' button. Below this, the repository name 'ziyao233' is followed by the description 'Update xfce/application-finder-related documents (#59)', the commit hash '828f78d', the time '9 months ago', and the number of commits '42 Commits'. The main content is a list of files and folders with their commit messages and timestamps.

File/Folder	Commit Message	Time
.github	Add CI:git squash commit check.	2 years ago
AutoTest	Python and nodejs (#58)	10 months ago
Chromium	Chromium (#53)	last year
DDE	更新 dde 测试用例 (#55)	last year
Firefox	Firefox的文字排版修正与路径修正 (#48)	2 years ago
GIMP	Add: test report to testcase repo.	2 years ago
Libreoffice	Add LibreOffice testcase (#52)	10 months ago
MySQL	添加nodejs ~ (#54)	10 months ago
Open MPI	Openmpi (#35)	2 years ago
Sysprof	Add: test report to testcase repo.	2 years ago
System	Add: test report to testcase repo.	2 years ago
Template	Add a testcase of Libreoffice as an example.	2 years ago
Thunderbird	Thunderbird和firefox的文字排版修改与图片路径修改 (#49)	2 years ago
UKUI	Add UKUI testcase. (#56)	10 months ago
VLC	VLC	2 years ago
Xfce	Update xfce/application-finder-related documents (#59)	9 months ago
nodejs	添加nodejs ~ (#54)	10 months ago
CODE_OF_CONDUCT.md	Add code of conduct.	2 years ago
README.md	Edit README.md: Add autotest requirements.	10 months ago

- 测试用例库：  
<https://github.com/ArielHel eneto/RISCV-testcase>
- 覆盖常用桌面软件
  - Chromium
  - Firefox
  - GIMP
  - LibreOffice
  - . . .


# 测试实例

trdthg fix: trinity 6ceccbb 2个月前		
← ...		
📁 compiler	fix: set compiler timeout	
📁 function	feat: add post_fail_hook to LTP	
📁 kernel	fix: trinity	
📁 long_stress	fix: trinity	
📁 mugen	feat: move barrier to init.pm	
📁 performance	fix 20231009	
📄 init.pm	fix: install git make at init.pm	
📄 reboot.pm	refactor: use oerv tests	
📄 success.pm	refactor: use oerv tests	
📄 application_finder_Remember_last_sele...	add xfce case	
📄 application_finder_Text_Beside_Icons.pm	add xfce case	
📄 application_finder_centre_window.pm	add xfce case	
📄 application_finder_clear_custom_comma...	add xfce case	
📄 application_finder_fold_unfold.pm	Add xfce tests	
📄 application_finder_hide_category_pane.pm	Add xfce tests	
📄 application_finder_icon_display_mode.pm	Add xfce tests	
📄 application_finder_icon_size.pm	Add xfce tests	
📄 xfce_applications.pm	add xfce testcases	
📄 xfce_arrange_desktop_icons.pm	add xfce testcases	
📄 xfce_create_document.pm	add xfce testcases	
📄 xfce_create_folder.pm	add xfce testcases	
📄 xfce_create_launcher.pm	add xfce testcases	
📄 xfce_create_url_link.pm	add xfce testcases	

- os-autoinst 自动化测试用例库：<https://gitee.com/yunxiangluo/os-autoinst-distri-openeuler>
- 覆盖大量常用软件以及系统基础组件



## openQA 测试实例

 trdthg feat: add login prompt b6cad86 3个月前	📄 51 次提交
⚠️ 文件数量太多, 仅显示 1000 文件	
📄 alignmenttest-output-pdf-20230718.j...	
📄 alignmenttest-output-pdf-20230718.p...	
📄 alignmenttest-output.pdf-icon-20230...	
📄 alignmenttest-output.pdf-icon-20230...	
📄 application-finder-Education-category...	
📄 application-finder-Education-category...	
📄 application-finder-Education-category...	
📄 application-finder-Education-category...	
📄 application-finder-Text-Beside-Icons-...	

- os-autoinst needles: <https://gitee.com/yunxiangluo/os-autoinst-needles-openeuler>
- 测试用例 570 个

## 测试使用场景1——图形化应用

- 创建 needles

使用 Needle Editor （图形化界面）创建，可通过 Flatpak 安装

- 创建测试用例

在 `/var/lib/openqa/tests/openeuler/tests` 目录下新建 perl 模块，在 `run` 函数中按照测试步骤编写

- 创建测试套

编写对应的测试套 `schedule` 文件，存入对应目录，之后在 WebUI 中创建测试套即可

# 测试使用场景1——图形化应用



- 绘制选取
- 添加标签
- 添加选区
- 保存 needle 配置文件
- 部署 needle

# 测试使用场景1——图形化应用

```
---
name: firefox_bookmark
schedule:
  - installation/oerv_first_boot
  - wrapper_testsuite/wrapper_package_install
  - x11/firefox/firefox_bookmark_new
  - x11/firefox/firefox_bookmark_edit
  - x11/firefox/firefox_bookmark_tags
  -
x11/firefox/firefox_bookmark_toolbar_only_show_on
_new_tab
  -
x11/firefox/firefox_bookmark_toolbar_always_show
  -
x11/firefox/firefox_bookmark_toolbar_never_show
  - x11/firefox/firefox_bookmark_suggest
  - x11/firefox/firefox_bookmark_add_to_toolbar
  - x11/firefox/firefox_bookmark_separator
  - x11/firefox/firefox_bookmark_tabs
  - x11/firefox/firefox_bookmark_delete_all
  - x11/firefox/firefox_bookmark_backup
  - x11/firefox/firefox_bookmark_import
  - shutdown/shutdown
```

```
use base "x11test";
use strict;
use warnings;
use Test::API;
sub run() {
    my ($self) = shift;
    #start firefox to baidu
    $self->start_firefox;
    wait_still_screen;

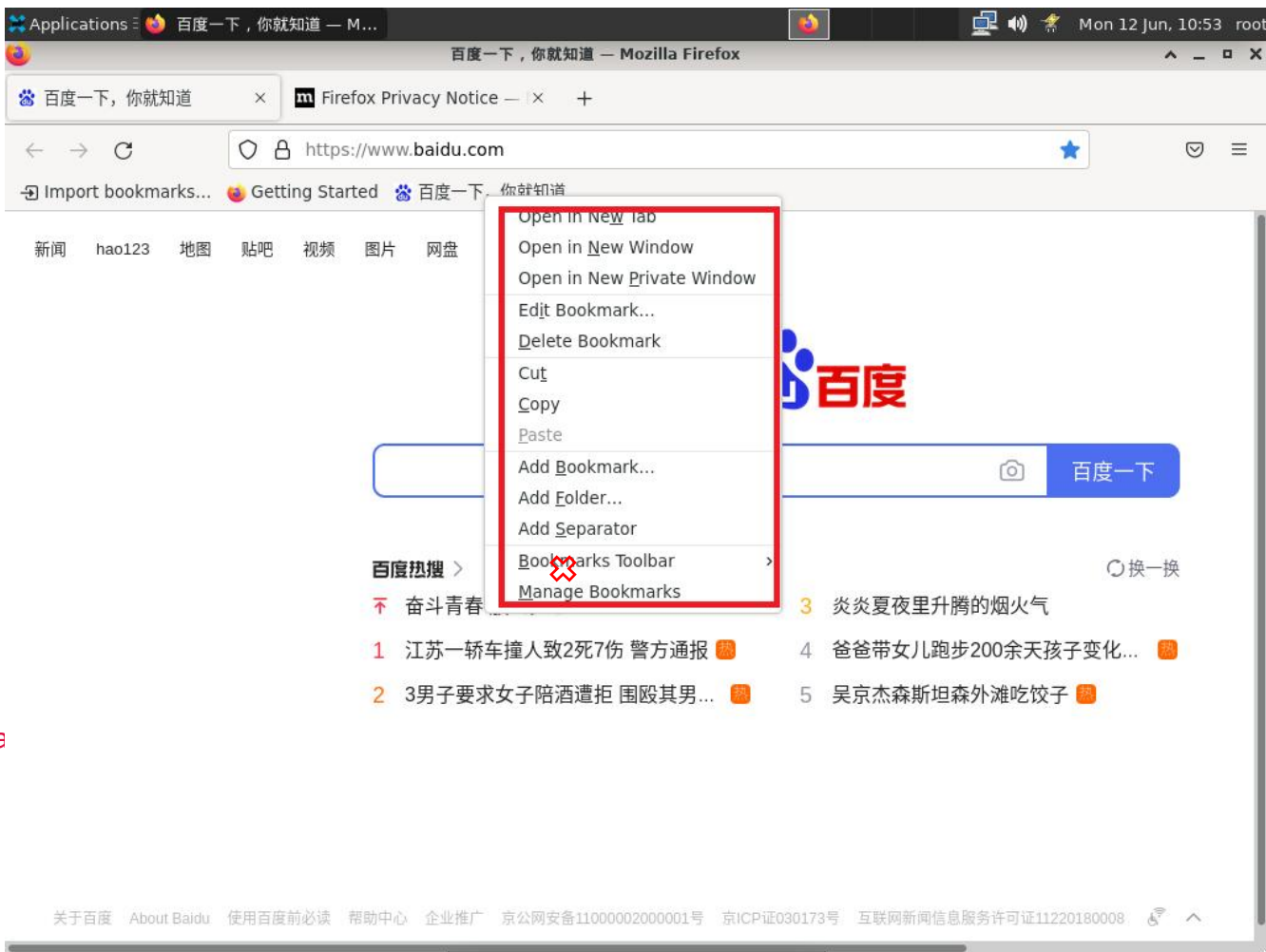
    #open two page and bookmark all tabs
    assert_and_click 'firefox-open-new-tab';
    $self->firefox_open_url("https://www.openeuler.org/zh/");
    assert_and_click ('firefox-tab-baidu', button=>'right');
    assert_screen 'firefox-tab-menu';
    send_key "ctrl-s";
    assert_and_click ('firefox-tab-baidu', button=>'right');
    assert_screen 'firefox-tab-menu';
    send_key "ctrl-b";
    assert_screen 'firefox-new-bookmarks';
    type_string 'test folder';
    wait_still_screen(2);
    assert_and_click 'firefox-new-bookmarks-select-location';
    assert_and_click 'firefox-new-bookmarks-location-other';
    assert_and_click 'firefox-new-bookmarks-save';

    #delete folder and bookmarks
    $self->firefox_manage_bookmarks;
    assert_and_click ('firefox-manage-bookmarks-library-test-folder');
    assert_screen 'firefox-manage-bookmarks-library-test-content';
    assert_and_click('firefox-library-organise');
    send_key "ctrl-d";
    assert_screen("firefox-library-no-bookmarks");
    if (check_screen 'firefox-manage-bookmarks-library-test-folder') {
        die 'delete folder failed';
    }
    send_key "alt-f4";
    wait_still_screen;

    #close firefox
    send_key "alt-f4";
    assert_screen "generic-desktop-oerv";
```

# 测试使用场景1——图形化应用

```
{
  "area": [
    {
      "type": "match",
      "ypos": 163,
      "height": 321,
      "xpos": 402,
      "width": 216,
      "click_point": {
        "xpos": 54,
        "ypos": 286.5
      }
    }
  ],
  "properties": [],
  "tags": [
    "firefox-bookmark-toolbar-menu",
    "firefox-bookmark-toolbar-menu-bookmarks_toolba
  ]
}
```



## 测试使用场景2——开发板启动和功能

- 目前依赖 SSH 和 VNC 接入
- 前置条件
  - 设置好防火墙
  - 配置 SSH 访问
  - 安装开启 VNC （对于图形化应用）
- 硬件开发板串口接入：开发中（t-autotest）

## 测试使用场景2——开发板启动和功能

修改 worker.ini 配置

开发板编写部分脚本

- 开发板重启控制脚本
- 编写 serial 抓取脚本
- 编写自动烧录脚本
- 配置 VNC 相关配置

后续和图形化测试相同

```
[global]
GENERAL_HW_CMD_DIR =
/var/lib/openqa/share/tests/openeuler/data/generalhw_scripts
[21]
```

```
# 重启脚本
GENERAL_HW_REBOOT_CMD = reboot_SUT_via_SSH.sh
GENERAL_HW_REBOOT_ARGS = 192.168.0.44
```

```
# 串口连接
GENERAL_HW_SOL_ARGS = ttyUSB1
GENERAL_HW_SOL_CMD = get_sol_dev.sh
```

```
# 烧录脚本
GENERAL_HW_FLASH_ARGS = 192.168.0.44:~/
GENERAL_HW_FLASH_CMD = flash_ssd.sh
```

```
# VNC 连接
GENERAL_HW_VNC_IP=192.168.0.44      #桌面测试时需要配置
GENERAL_HW_VNC_PASSWORD=123456     #桌面测试时需要配置
GENERAL_HW_VNC_PORT=5901           #桌面测试时需要配置
SUT_IP = 192.168.0.44
```

```
WORKER_CLASS = generalhw_unmatched # 修改测试对象类型
```

## 挑战和尝试

- os-autoinst 存在的问题:
- 对硬件开发板适配不好
- 十年老库，代码库复杂，基于多进程架构
- 基于 Perl 语言及其生态，可维护性差
- .....
- 基于 Rust 重写的 os-autoinst:

<https://github.com/trdthg/t-autotest>

- 解决开发板串口接入问题



## 今后的工作

- 继续改进测试用例库，丰富测试用例
- 改进 `os-autoinst`
- 改善对 RISC-V 开发板的支持，解决串口接入问题

# 目 录

01

openEuler RISC-V测试

02

Mugen

03

openQA

04

未来的构想

# 04

## 未来构想

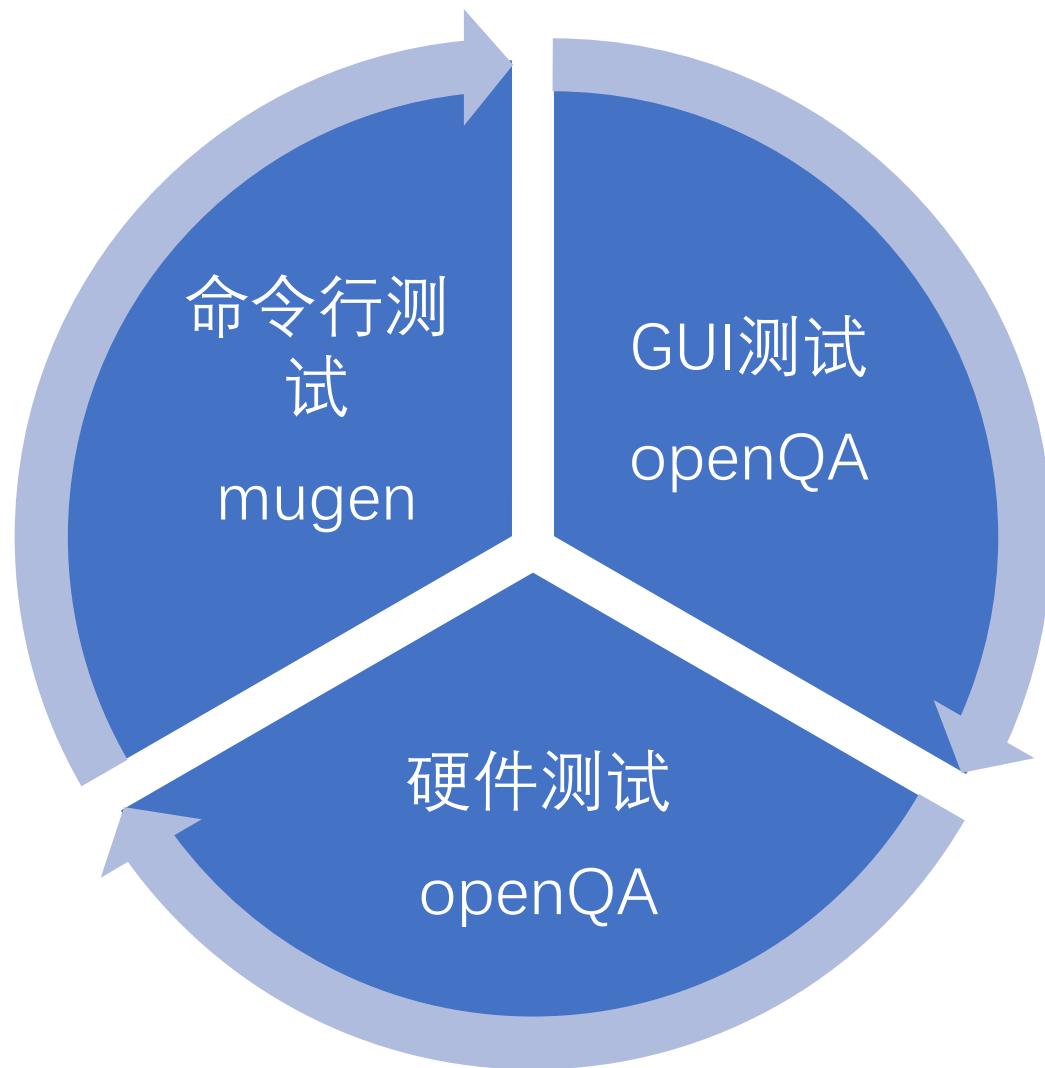
### 4.1

测试工具开发

### 4.2

测试环境的基础设施建设

## 测试工具的开发



# 测试环境的基础设施建设



PLCT云测试中心

命令行

GUI

硬件  
(启动和内核)

谢谢&提问