Megatest/Logpro Training

Using Megatest and Logpro for creating flows and Automation.

Matt Welland

Training Overview

- Background
- Getting started
  - Running an existing flow
    - Launching runs, debugging
  - Creating a flow
    - configs: megatest, runconfig
    - tests/tasks: testconfig, logpro
  - Getting information about runs and tests
- Roadmap

What does Megatest do?

- Run tests or tasks under different contexts with
  - one or many sequential steps per task
  - dynamic test dependency calculation
  - on multiple hosts
  - easy to use but powerful iteration
- Comprehensive meta data capture
  - task state: RUNNING, COMPLETED
  - task status: PASS, FAIL, WARN, CHECK
  - host, test run time etc.

Megatest Goals

Unix based, decentralized, easy and sustainable automation

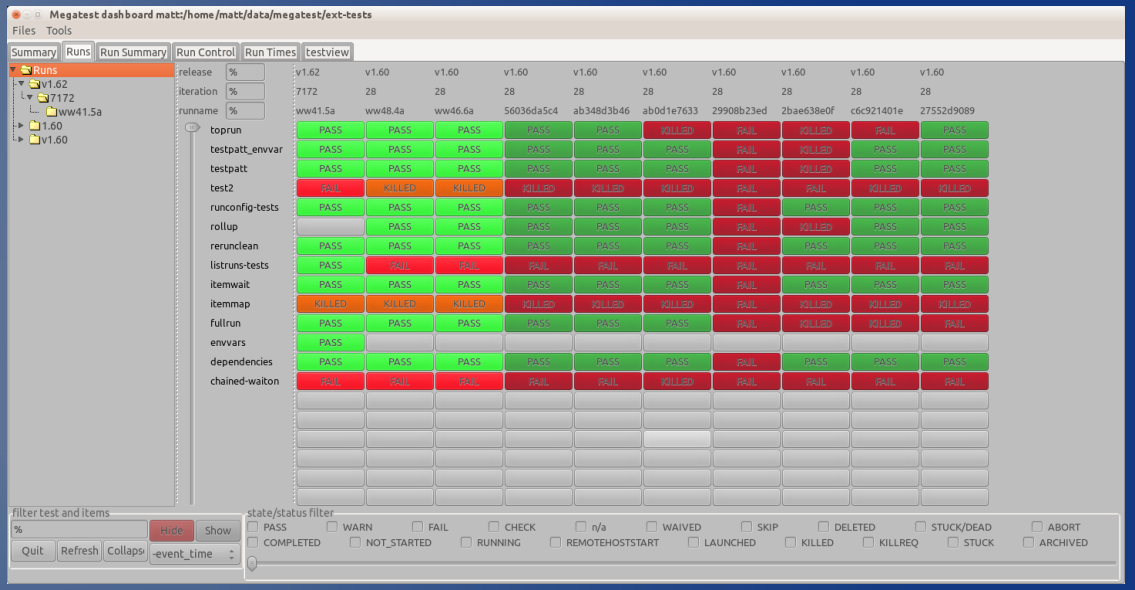
Self-checking	Easy to write self-checking tests
Traceable	environment variables, host OS, etc. captured and recorded.
Immutable	do not modify or overwrite previously run tests.
Repeatable	results can be easily recreated by running same target again
Relocatable	the test area can be checked out and the tests run anywhere
Encapsulated	test run area is self-contained with all inputs and outputs kept
Deployable	Source files can be checked out and run by anyone

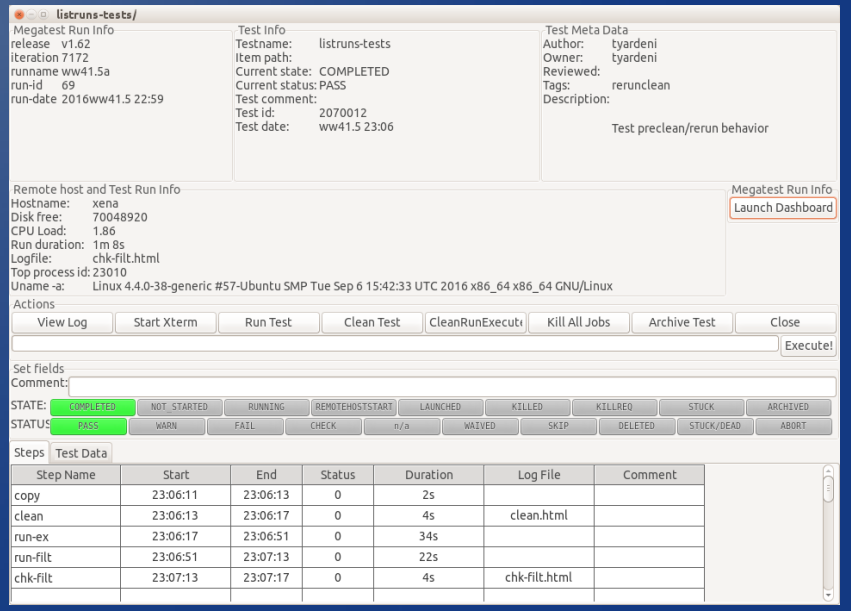
Wisdom is knowing when it is ok to bend or break the rules.

Megatest strives to make it straightforward to do things right but still possible to get the job done when the rules must be bent. i.e. Megatest is not opinionated.

Dashboard/Test Control Panel

- dashboard
  - browse runs
  - filtering
    - target
    - runname
    - test pattern
    - state/status
  - launch, remove runs
- test control panel
  - Xterm for debug
  - view logs
  - re-run tests





Terminology

target	one or more keys separated by /, used to set context; e.g. OS, release, architecture, stage (e.g. development, final QA, alpha, beta) and so forth.
run name	unique name (within a single target grouping) for a run, a common idiom is to use week and day numbers: e.g w41.6 (use unix command: date +w%V.%u)
run	a group of tests run under a single target and run name
task/test	a self-contained area with scripts and data to achieve some testing or automation goal
iterated task	a single task run multiple times with one or more variables iterated over a range of values
state	the state of a test; NOT_STARTED, RUNNING, COMPLETED etc.
status	the current status of this test; PASS, FAIL, n/a

Megatest System Overview

- Config files (the source code)
  - megatest.config (setup for given area)
  - runconfigs.config (context table, targets)
  - tests/<testname>/testconfig (test spec)
- database, dynamic state of runs
  - megatest.db
- Tools
  - **megatest** (command line), **dashboard** (gui), **logpro** (log file analysis via rules), and **refdb** (text based data tables)

Getting Help

- Command line help:

  megatest -h

  or try: megatest -h |& less

- The user manual:

  megatest -manual

- Support contact: mattrwelland@gmail.com
- Web Site:
  - https://chiselapp.com/user/kiatoa/repository/megatest
  - https://www.kiatoa.com/fossils/megatest (mirror)

dashboard

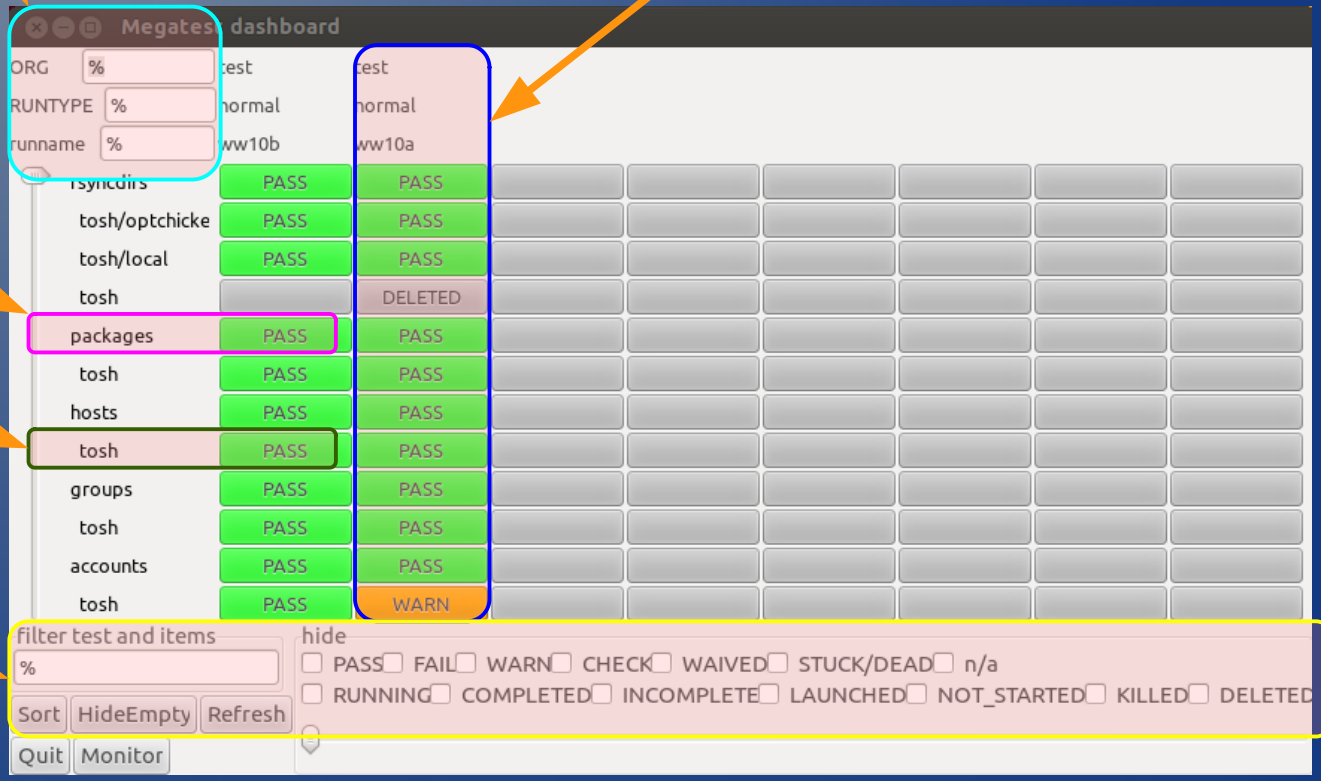
Target and runs filter

a “run”

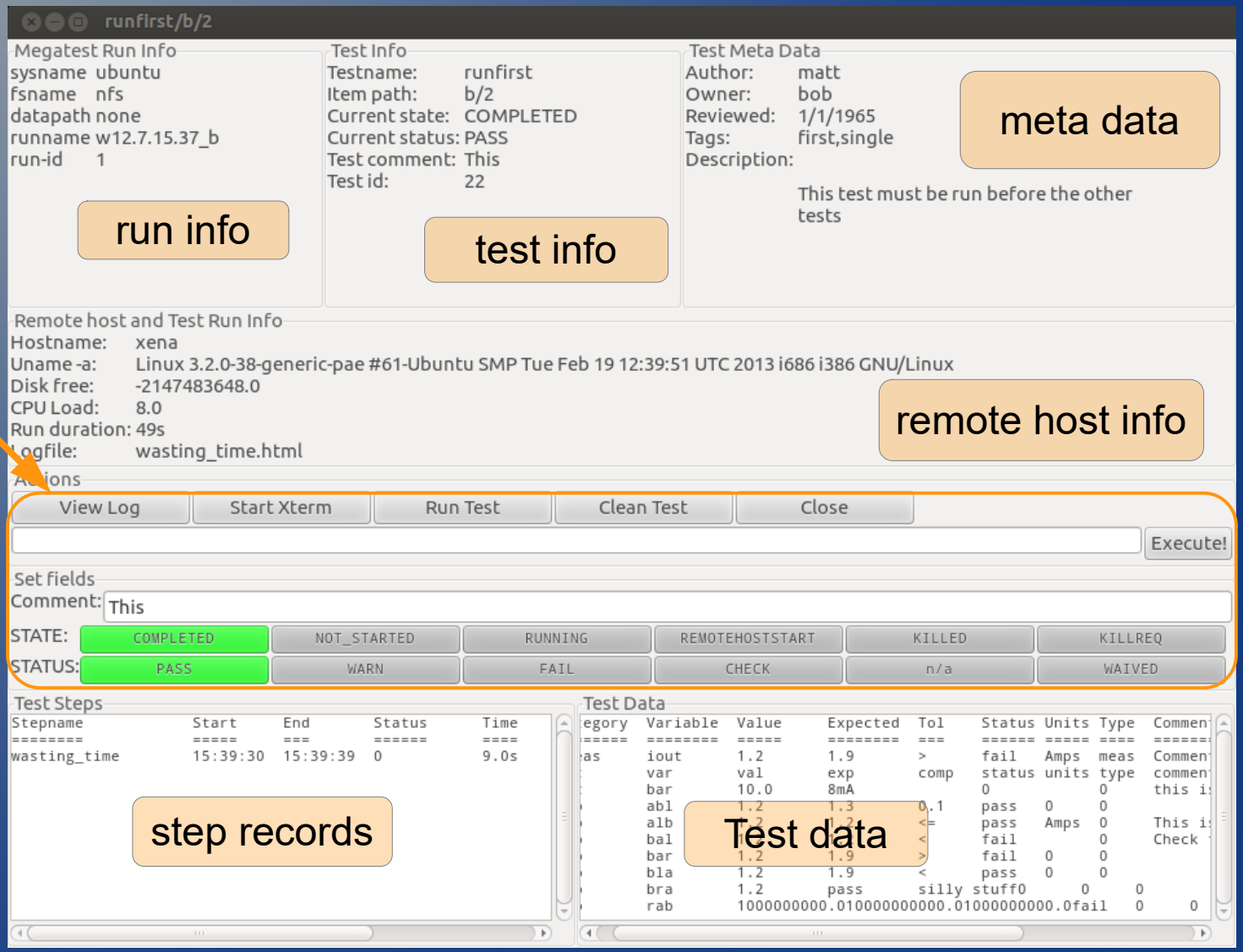
a “test”

a “test item”

tests filter



test control panel



Run Management

- Launching runs
  - dashboard – from Run Control tab.
  - command line - megatest -run …
  - test control panel – [run]->[execute]
- Removing runs
  - command line: megatest -remove-runs ...

note: all these commands require the use of additional selector parameters such as -target, -runname and -testpatt

Task/Test Management

- Killing jobs
  - In the gui set status to “KILLREQ” and the job will be killed.
  - Command line using -set-state-status
- Changing state and status of tests
  - Use test control panel. E.g. set a test to PASS after debugging to enable downstream tests to be run.
- Use -rerun to re-run jobs with given status
  - … -rerun ABORT

Test Selectors

- -testpatt testpattern/itempattern
  - wild card is “%”

    % synonymous with %/%

    %/ toplevel tests (no items)
- comma separate multiple patterns (OR)

  %/,%/a/b All toplevel + any items matching a/b

Getting information

- -list-runs pattern
  - lists runs with runname matching pattern.
- -extract-ods
  - creates an open-document spreadsheet
- Miscellaneous queries

  -list-disks

  -list-targets

  -list-db-targets

Creating Flows/Automation

- Try the area and test helpers to get a basic start

  > megatest -create-megatest-area

  > megatest -create-test TESTNAME

Config File Syntax

The config file syntax was designed to be:

- familiar, simple and forgiving to syntax mistakes
- easy to understand
- easy to trace where values originated
- expressive enough for complex needs.

	Example	description of the example
Sections	[setup]	Variables defined on subsequent lines will be in the “setup” section
Variables	ABC 1	Variable “ABC” will have the value “1”
[ ] directives	[include a.txt]	include file “a.txt”, see manual for all directives
#{ } text substitutions	#{shell ls $PWD}	replace the #{ … } with the output of the ls $PWD command. Note that newlines are replaced with spaces.

Config File Text Substitutions

NOTE: [ ] substitutions can be deferred by megatest and executed just before launching a test but #{ } substitutions are done as each line is read.

[include filename]	Includes filename. Ignores if filename does not exist
[system command]	replaced with output from command
#{shell command}	replaced with output from command
#{system command}	replaced with the exit code of command
#{scheme (schemecode)}	replaced with the result of evaluating (schemecode)
#{getenv VAR}	replaced with the value of environment variable VAR
#{get section var}	replaced with the value of var from section
#{rget var}	use runconfig rules to get a variable
#{mtrah some/path/or/file}	Insert the path or file as based at the run area home

Creating a Megatest Area

- Required Config files
  - megatest.config
  - runconfigs.config
- Tests
  - tests/<testname>/testconfig
- Can use the helper “wizards”

  megatest -create-megatest-area

  megatest -create-test <testname>

Required Config Files

megatest.config

```
[fields]
PLATFORM TEXT
OS       TEXT

[setup]
# Adjust max_concurrent_jobs to limit parallel jobs
max_concurrent_jobs 50

# This is your link path, best to set it and then not change it
linktree #{mtrah linktree}

# Job tools control how your jobs are launched
[jobtools]
useshell yes
launcher nbfake

# You can override environment variables for all your tests here
[env-override]
EXAMPLE_VAR example value

# As you run more tests you may need to add additional disks
# the names are arbitrary but must be unique
[disks]
disk0 #{mtrah runs}
```

runconfigs.config

```
[default]
ALLTESTS see this variable

# Your variables here are grouped by targets [SYSTEM/RELEASE]
[SYSTEM_val/RELEASE_val]
ANOTHERVAR only defined if target is SYSTEM_val/RELEASE_val
```

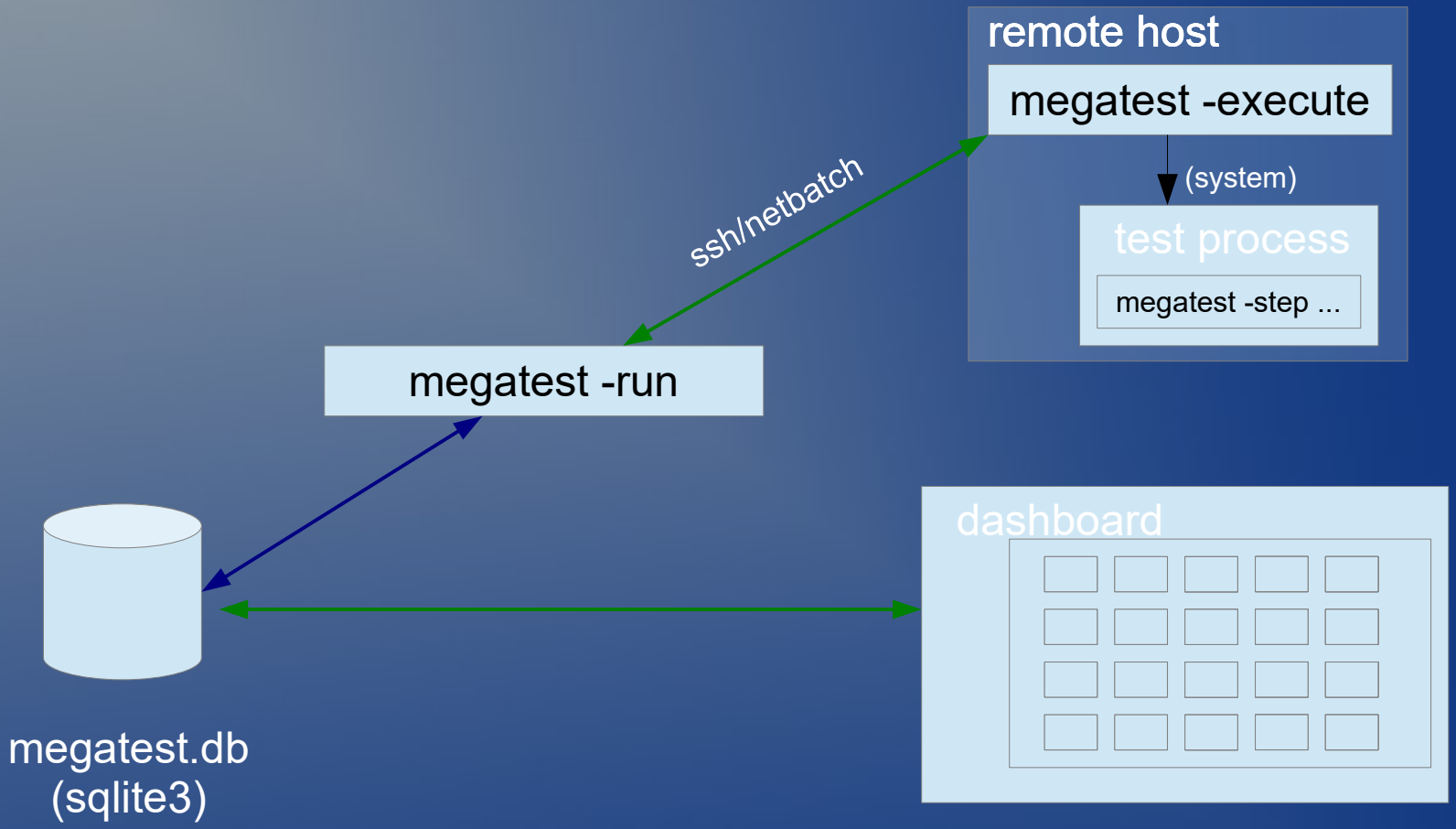
Example testconfig

testconfig

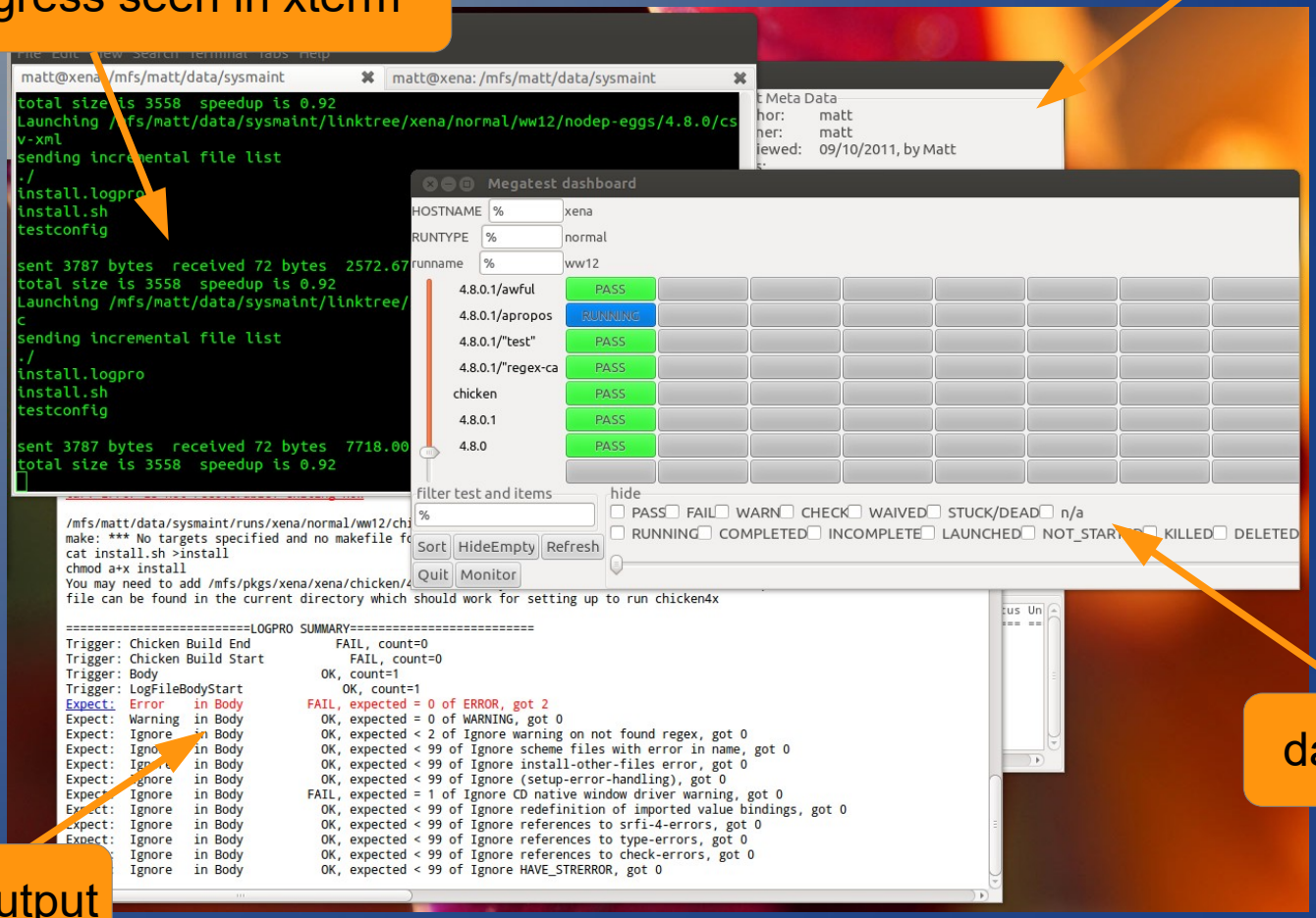
```
# Add additional steps here. Format is "stepname script"
[ezsteps]
step1 step1.sh
step2 step2.sh

# Test requirements are specified here
[requirements]
waiton setup
priority 0

# Iteration for your tests are controlled by the items section
[items]
COMPONENT parser datastore transport analyzer

[scripts]
step1.sh #! /bin/bash
  do-stuff-here

[logpro]
step1 ;;
   (expect:error in “LogFileBody” = 0 “No errors” #/err/i)

# test_meta is a section for storing additional data
# on your test
[test_meta]
author matt
owner  matt
description An example test
tags tagone,tagtwo
reviewed never
```

Backup

How it Works


remote host
megatest -execute
(system)
test process
megatest -step ...
ssh/netbatch
megatest -run
dashboard
megatest.db
(sqlite3)

A Day in The Life ...



Writing a Test “checkspace”

- Write a test that checks for available space
  - tests can “waiton” this test before running.
- Our test will use this simple script, checkspace.sh:

```
#!/bin/bash -e
freespace=`df -k $DIRECTORY | grep $DIRECTORY | awk '{print $4}'`
if [[ $freespace -lt $REQUIRED ]];then
  echo "ERROR: insufficient space on $DIRECTORY"
  exit 1
else
  echo "There is adequate space on $DIRECTORY"
fi
```

Note: Files for this example can be found

Writing a Test “checkspace”

- Commands to create test “checkspace”
  - mkdir -p linktree runs tests/checkspace
  - cd tests/checkspace
  - vi checkspace.sh
  - chmod a+x checkspace.sh
  - vi testconfig

```
# Add steps here. Format is "stepname script"
[ezsteps]
checkspace checkspace.sh

# Iteration for your tests are controlled by the items section
[itemstable]
DIRECTORY    /tmp   /opt
REQUIRED  1000000 100000
```

Writing a test “checkspace”

- Write a logpro file to analyze your results

```
(expect:error in    "LogFileBody" = 0 "Any error"         #/err/i)
(expect:required in "LogFileBody" = 1 "Sucess signature" #/adequate space/)
```

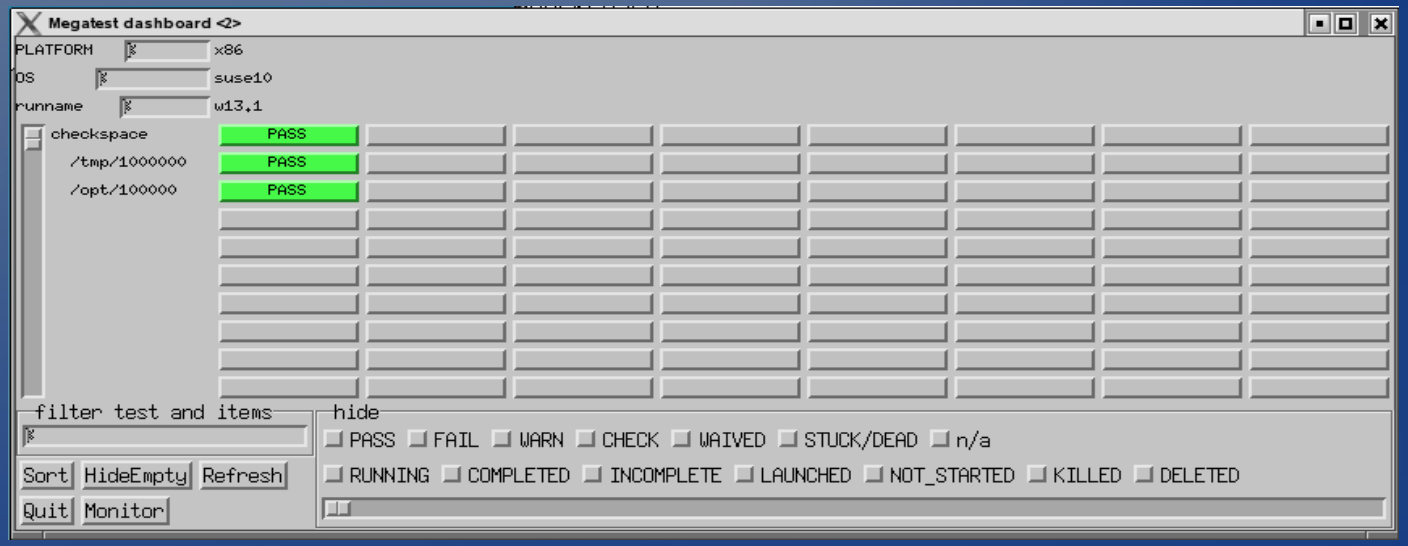
```
.
|-- megatest.config
|-- megatest.db
|-- monitor.db
|-- runconfigs.config
`-- tests
    `-- checkspace
        |-- checkspace.logpro
        |-- checkspace.sh
        `-- testconfig
```

Runing the “checkspace” Test

Run your test

From the directory where “megatest.config” exists run these commands:

dashboard &
megatest -runtests % -target x86/suse10 :runname w`date +%V.%u`



The “checkspace” Test Directories

```
|-- linktree
|   `-- x86
|       `-- suse10
|           `-- w13.1
|               `-- checkspace
|                   |-- opt
|                   |   `-- 100000 -> /nfs/ch/disks/ch_unienv_disk005/qa_mrwellan/interim/src/megatest/example/runs/x86/suse10/w13.1/checkspace//opt/100000
|                   |-- testdat.db
|                   `-- tmp
|                       `-- 1000000 -> /nfs/ch/disks/ch_unienv_disk005/qa_mrwellan/interim/src/megatest/example/runs/x86/suse10/w13.1/checkspace//tmp/1000000
|-- runs
|   `-- x86
|       `-- suse10
|           `-- w13.1
|               `-- checkspace
|                   |-- opt
|                   |   `-- 100000
|                   |       |-- NBFAKE-2013WW13.1_09:57:48
|                   |       |-- checkspace.html
|                   |       |-- checkspace.log
|                   |       |-- checkspace.logpro
|                   |       |-- checkspace.sh
|                   |       |-- megatest.csh
|                   |       |-- megatest.sh
|                   |       |-- mt_launch.log
|                   |       |-- testconfig
|                   |       `-- testdat.db
|                   `-- tmp
|                       `-- 1000000
|                           |-- NBFAKE-2013WW13.1_09:57:49
|                           |-- checkspace.html
|                           |-- checkspace.log
|                           |-- checkspace.logpro
|                           |-- checkspace.sh
|                           |-- megatest.csh
|                           |-- megatest.sh
|                           |-- mt_launch.log
|                           |-- testconfig
|                           `-- testdat.db
```

Setup for Run “Flavors”

- runconfigs.config

  [default]

  VARS here are inherited by all runs

  [some/target]

  VARS here inherited in some/target runs

- NB// the last specified definition overrides prior definitions.

Setup Tests/Tasks

- A test or task is a set of scripts and data designed to do something or test something.
- Create in tests directory
- Test name limitations
  - No spaces or special characters
  - [a-zA-Z0-9_] and “-” are ok.

The testconfig file [setup]

- [setup]

  runscript scriptname.sh

  - The script must exist in the testconfig directory and be executable
  - Output from the script is NOT captured by Megatest directly
  - The script can be an executable or written in any scripting language

The testconfig file [ezsteps]

- [ezsteps]

  step1 script1.sh

  - The script “script1.sh” will be executed and its output redirected to the file step1.log.
  - If a logpro file step1.logpro exists it will be used to process the logfile step1name.log and generate the PASS/FAIL/WARN status.

The testconfig file [items]

[items]

VAR1 value11 value12 value13 …

VAR2 value21 value22 value23 ...

- This will iterate this test with all possible combinations of VAR1 and VAR2 values.

- Results:
  - value11/value21, value11/value22, value11/value23, value12/value21, value12/value22, value12/value23 ...

The testconfig file [itemstable]

[itemstable]

VAR1 value11 value12 …

VAR2 value21 value22 …

- This will iterate over the test with only aligned value combinations.

- Result:
  - value11/value21, value12/value22 …

NOTE: You can combine items and itemstable but they work independently and the result may not be what you expect.

The testconfig file [requirements]

[requirements]

waiton <testname … >

- this test will not be launched until the listed tests are COMPLETED and PASS, WAIVE or SKIP.

jobgroup <groupname>

- this test will be added to the named job group and the relevant max concurrent jobs will apply

mode toplevel

- this test will proceed once all it waiton tests are completed with any status.

The testconfig file[test_meta]

- author matt
- owner bob
- description The description can run to multiple lines but subsequent lines must be indented with spaces.
- tags first,single
- reviewed 09/10/2011, by Matt

Megatest Calls in Tests

- -step stepname
  - mark the start or end of a step
- -test-status

  set the state and status of a test
- -setlog logfname

  set the path/filename to the final log relative to the test directory.
- -set-toplog logfname

  set the log for a series of iterated tests

Other Megatest calls

- -summarize-items

  for an itemized test create a summary html (usually called automatically)

- -m comment

  insert a comment for this test, can be used with any of the above calls

- -test-files or -test-paths

  Use the database to search for files or paths in the test run area

Example Megatest in-test calls

- -step

```
$MT_MEGATEST -step step1 :state start :status
running -setlog step1.html
```

- -test-status

(Mark a test as completed and trigger a rollup to the parent test of overall status)

```
$MT_MEGATEST -test-status :state
COMPLETED :status AUTO
```

- -test-path

```
export EZFAILPATH2=`$MT_MEGATEST -test-paths -target
  $MT_TARGET :runname $MT_RUNNAME -testpatt
  runfirst/a%`
```

Environment Variables

MT_TARGET	Contains the target for this run
MT_RUNNAME	The run name
MT_MEGATEST	Full path to megatest executable
MT_TEST_RUN_DIR	The area where the test itself runs
MT_TEST_NAME	The name of the current test
MT_ITEM_INFO	Data on the iteration
MT_RUN_AREA_HOME	The base area for this regression
MT_CMDINFO	Used internally by megatest
MT_DEBUG_MODE	Used to propogate debug mode to underlying megatest calls.
MT_LINKTREE	Full path to the link tree, use to find tests

Additional Features

- Run locking
  - Prevents removing or adding tests to a run

    -lock

    -unlock

Logpro

- Logpro syntax

  Logpro uses scheme calls directly and the full power of scheme is available. However 99% of logpro rule files will not need anything other than the base logpro rules.

- Documentation at: http://www.kiatoa.com/fossils/logpro

Rule	Example	Purpose
expect:error	(expect:error in “Logf” = 0 “Err desc” #/err1/i)	Flags errors matching the pattern err1
expect:ignore	(expect:ignore in “Logf” < 10 “Err desc” #/err2/i)	Ignore errors matching the pattern err2
expect:warning	(expect:warning in “Logf” = 0 “Desc” #/warn1/i)	Lines matching pattern warn1 flagged as warning
expect:required	(expect:required in “Logf” = 1 “Desc” #/reqrd/i)	Line matching pattern reqrd must exit in log file
expect:waive	(expect:waive in “Logf” = 0 “Err desc” #/err3/i)	Waive error matching pattern err3
expect:value	(expect:value in “Logf” 10 1 “Err desc” #/(\d+)/i)	The number matched must be 10 +/- 1
trigger	(trigger “start” #/Start logfile/)	Set trigger “**start**” on line with “Start logfile” string.
section	(section “Logf” “start” “end”)	Section **Logf** starts at trigger **start**, ends at **end**
hook:add	(hook:add “err1” “err1.pl #{msg}”)	On err1 call the err1.pl script with msg as param

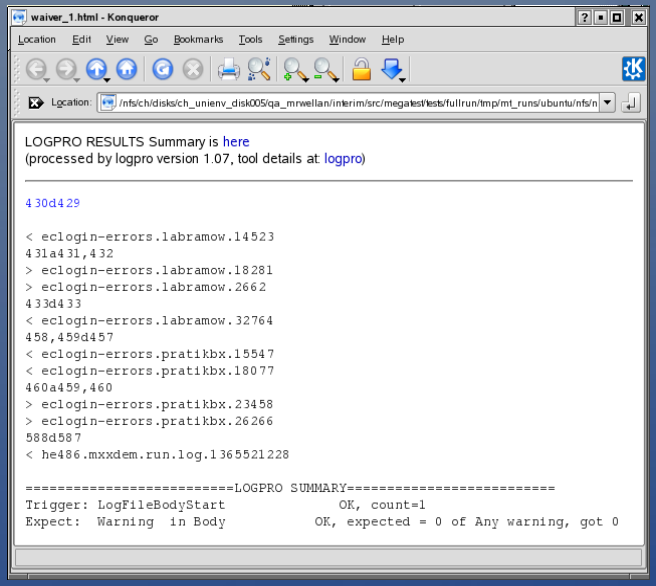
Advance Logpro Usage

- Data collection
  - Capturing with logpro
  - Rolling up with Megatest

Waiver Propagation

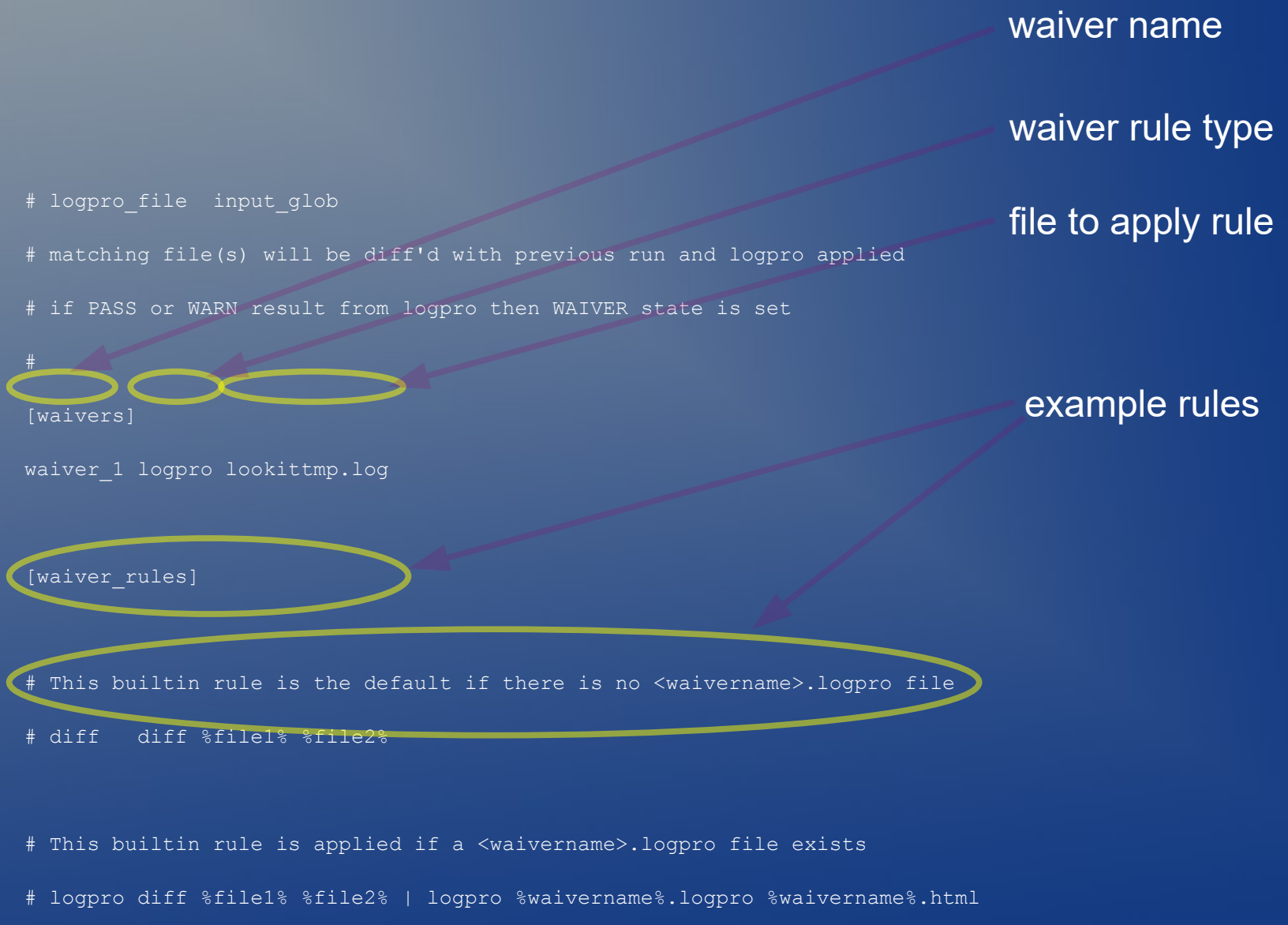
This test failed and was manually set to WAIVED in the next run

This test uses diff and logpro to determine if ok to propagate WAIVED



The WAIVED status was propagated because the criteria set in testconfig were all met

Waiver Propagation



Direct Access to Megatest Functions

- -repl
  - This will start a read-eval-print loop allowing you to directly call Megatest calls.
- -load test.scm
  - This will load the scheme source code and exectute it in the Megatest context.

New Features in v1.55

- Task/Test search path
  - organize your tests in different directories
  - reuse tests from other flows
- Automatic SKIP handling
  - Crontab friendly runs (can overlap)
- “itemmatch” mode
  - iterated tests block only on previous same-named iteration