The Neurospaces Workflow Automation Engine

Automatically generated draft document

Hugo Cornelis

18 December, 2023

Summary

The Neurospaces Workflow Automation Engine assists in the development and automation of project specific workflows based on system shell commands.

Its main features are:

- Simple to use through bash completion: Use Bash completion to find your way through a hierarchy of project specific command workflows.
- Modularity: Compose complex workflows from simple ones and separate commands and configuration.
- Roles: Implement workflows on different physical and virtual machines, tmux sessions and other accounts, then integrate them.
- Export: Export your sophisticated workflows as a single shell script to be used by others.
- Colors: Explore role diagrams of your workflows color coded according to your project specific settings.
- Collaboration: Use a common git repository to share project workflows between colleagues.
- Coherence: Your documented developer workflows are the workflows you use for development.

The benefits of using the Neurospaces Workflow Engine:

- Reduced ramp up time: New colleagues have immediate access to hierarchically organized workflows and their configuration.
- Persistence: Never forget about that complicated command that you used to 'flash your device'.
- Structure: Group related workflows in workflow modules and namespaces.
- Auto-exploration: The workflows of your projects become self-documenting.

Enjoy your workflows.

1 Testing of the workflow automation engine

The file *10_help-pages.t* defines the tests that are explained in this section.

This module tests the workflow automation engine.

The workflow script enables the automation of customizable modular project-specific workflows that use system shell commands.

1.1 The main help page: Do we get the main help page ?

The test is started with the system shell command: **../bin/workflow –help**

Expected application output:

```
../bin/workflow: support for workflow design for embedded software engineers.

SYNOPSIS

../bin/workflow <options> <target> <command> -- < ... command specific options and arguments ... >

EXAMPLES -- first try these with the --dry-run to understand what they do:

  $ ../bin/workflow --help-targets                                    # display the available targets that are found in the configuration file.

  $ ../bin/workflow --help-commands                                   # display the available commands that are found in the configuration file.

  $ ../bin/workflow ssp build                                         # 'build' the 'ssp' target (if it exists for your local configuration).

  $ ../bin/workflow --dry-run ssp build                               # display the shell commands that would be executed to 'build' the 'ssp' target.

options:
    --bash-completion              compute bash completion for the given command line.
                                   hint: the bash completion script implements completion for options, targets and commands.
    --branch                       git branch to work with.
    --build-server                 the build server profile to work with.
    --built-image-directory        the directory on the build server where the built images are to be found.
    --command                      commands to execute, hyphens (-) in the command will be replaced with underscores (\_).
    --dry-run                      if set, do not execute system shell commands but print them to STDOUT.
    --dump-all-interaction-roles   dump all the interaction roles found in the configuration.
    --dump-interaction-roles       dump the found interaction roles (note that they depend on the scheduled commands).
    --dump-module-interaction-roles dump all the interaction roles found in the module of the given command.
    --dump-schedule                dump the constructed schedule to standard output without executing the scheduled commands.
    --export-remote                include the remote access part of exported commands.
                                   this option takes a number: 0 means all roles are exported, any other number exports only that respective role.
    --export-sh                    export the commands to a file with the given name.
    --export-sudo                  include the sudo commands when exporting commands to a file.
    --export-times                 export the times when commands are started and ended to a file with the given name.
    --export-verbose               when exporting the commands to a file, interleave them with echo commands.
    --force-rebuild                force a rebuild regardless of the existence and build date of previously built artefacts.
    --forward-destination          the target file forward destination to copy to.
    --forward-source               the target file forward source to copy from.
```

```
    --help                           display usage information and stop execution.
    --help-build-servers             display the known build servers.
    --help-commands                  display the available commands, add a target name for restricted output.
    --help-module                    display all the available help information about the commands of the module.
    --help-field-project-name        print the field project name and exit.
    --help-options                   print the option values.
    --help-packages                  display known package and overriden package information and stop execution.
    --help-projects                  display known project information and stop execution.
    --help-targets                   display known targets and stop execution.
    --incremental                    assume an incremental build (default is yes
    --interactions                   show the interaction diagram of the commands.
    --interactions-all               show a diagram with all the commands and all the interaction roles.
    --interactions-module            show the interaction diagram of all the commands in the module.
    --interactions-module-all-roles show the interaction diagram of the commands using all the found interaction roles in the configuration.
    --packages                       packages to operate on, can be given multiple times.
    --ssh-port                       the ssh port.
    --ssh-server                     the used ssh build server.
    --ssh-user                       ssh-user on the build server (please configure your public key).
    --target                         the target to apply the given commands to.
    --tftp-directory                 the target tftp directory (eg. where your device will find its kernel and rootfs).
    --verbose                        set verbosity level.

NOTES

OVERRIDE\_SRCDIR delivered packages for Buildroot targets are recognized.
```

1.2 Default builtin commands: Do we get the builtin commands help page ?

The test is started with the system shell command: **../bin/workflow –help-commands**

Expected application output:

```
'available\_commands (copy-paste the one you would like to execute, try it with the --help or the --dry-run option, or execute it without these options)':
  - workflow builtin add\_target --help
  - workflow builtin install\_scripts --help
  - workflow builtin print\_configuration\_directory --help
  - workflow builtin start\_project --help
```

1.3 Default builtin targets: Do we get the builtin targets help page ?

The test is started with the system shell command: **../bin/workflow –help-targets**

Expected application output:

```
targets:
  builtin:
    description: the builtin target allows to start a new project and upgrade existing projects
```

1.4 Default builtin targets: Do we see the help page for starting a new project ?

The test is started with the system shell command: **../bin/workflow builtin start_project –help**

Expected application output:

```
builtin start\_project: start a new project with a given name in the current directory.

This will install a project descriptor, a configuration file and an
empty command file.

arguments:

    name: name of the new project.
```

2 Testing of the workflow automation engine

The file *40_workflow-automator/25_new-project-docker.t* defines the tests that are explained in this section.

This module tests the workflow automation engine.

The workflow script enables the automation of customizable modular project-specific workflows that use shell commands.

2.1 Showing that the container works: working directory: Are we in the correct working directory in the Docker container ?

The test is started with the system shell command: **pwd**

Expected application output:

```
/home/neurospaces
```

2.2 Showing that the container works: current directory contents: Can we list the current directory in the Docker container ?

The test is started with the system shell command: **ls -1**

Expected application output:

```
projects
```

2.3 Showing that the container works: generation of configure scripts: Can generate configure scripts in the Docker container ?

The test is started with the system shell command: **cd projects/workflow-automation-engine/source/snapshots/master/ && ./autogen.sh**

2.4 Showing that the container works: configuration of the workflow automation engine: Can we configure the workflow automation engine in the Docker container ?

The test is started with the system shell command: **cd projects/workflow-automation-engine/source/snapshots/master/ && ./configure**

Expected application output:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether we build universal binaries.... no
checking OS specifics..... Host is running .
checking for perl5... no
checking for perl... perl
checking Checking the perl module installation path...  ${prefix}/share/perl/5.36.0
./configure: line 2663: cd: perl: No such file or directory
./configure: line 2666: cd: perl: No such file or directory
checking for mtn... no
checking for monotone... no
checking for dpkg-buildpackage... dpkg-buildpackage
checking for dh... no
checking for rpmbuild... no
checking for python... no
checking for python2... no
checking for python3... /usr/bin/python3
checking for python version... 3.11
checking for python platform... linux
checking for GNU default python prefix... ${prefix}
checking for GNU default python exec\_prefix... ${exec\_prefix}
checking for python script directory (pythondir)... ${PYTHON\_PREFIX}/lib/python3.11/site-packages
checking for python extension module directory (pyexecdir)... ${PYTHON\_EXEC\_PREFIX}/lib/python3.11/site-packages
checking Python prefix is ... '${prefix}'
find: 'tests/data': No such file or directory
checking that generated files are newer than configure... done
: creating ./config.status
config.status: creating Makefile
```

2.5 Showing that the container works: build of the workflow automation engine: Can we build the workflow automation engine in the Docker container ?

The test is started with the system shell command: **cd projects/workflow-automation-engine/source/snapshots/master/ && make**

Expected application output:

```
make[1]: Entering directory '/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master'
```

2.6 Showing that the container works: installation of the workflow automation engine: Can we install the workflow automation engine in the Docker container ?

The test is started with the system shell command: **cd projects/workflow-automation-engine/source/snapshots/master/ && sudo make install**

Expected application output:

```
make[1]: Entering directory '/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master'
make[2]: Entering directory '/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master'
 /usr/bin/mkdir -p '/usr/local/bin'
 /usr/bin/install -c bin/workflow '/usr/local/bin'
======================== Installing CPAN modules
( cd cpan ; ./cpan\_install [0-9][0-9]*.gz )
Installing CPAN modules
$VAR1 = [
          '13-Data-Utilities-0.04.tar.gz'
        ];
checking for perl -e 'use Data::Utilities 0.04'
Can't locate Data/Utilities.pm in @INC (you may need to install the Data::Utilities module) (@INC contains: /etc/perl /usr/local/lib/x86\_64-linux-gnu/perl/5.36.0 /usr/local/share/perl/5.36.0 /usr/lib/x86\_64
BEGIN failed--compilation aborted at -e line 1.
Installing Data-Utilities-0.04 (13-Data-Utilities-0.04.tar.gz)
/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master/cpan
/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master/cpan/Data-Utilities-0.04
Checking if your kit is complete...
Looks good
Generating a Unix-style Makefile
Writing Makefile for Data::Utilities
Writing MYMETA.yml and MYMETA.json
make[3]: Entering directory '/home/neurospaces/projects/workflow-automation-engine/source/snapshots/master/cpan/Data-Utilities-0.04'
```

2.7 Showing that the container works: help page of the workflow engine: Can we get the help page of the workflow engine in the Docker container ?

The test is started with the system shell command: **workflow –help**

Expected application output:

```
/usr/local/bin/workflow: support for workflow design for embedded software engineers.

SYNOPSIS

/usr/local/bin/workflow <options> <target> <command> -- < ... command specific options and arguments ... >

EXAMPLES -- first try these with the --dry-run to understand what they do:

  $ /usr/local/bin/workflow --help-targets                                    # display the available targets that are found in the configuration file.

  $ /usr/local/bin/workflow --help-commands                                   # display the available commands that are found in the configuration file.

  $ /usr/local/bin/workflow ssp build                                         # 'build' the 'ssp' target (if it exists for your local configuration).
```

```
  $ /usr/local/bin/workflow --dry-run ssp build                                 # display the shell commands that would be executed to 'build' the 'ssp' target.

options:
    --bash-completion               compute bash completion for the given command line.
                                    hint: the bash completion script implements completion for options, targets and commands.
    --branch                        git branch to work with.
    --build-server                  the build server profile to work with.
    --built-image-directory         the directory on the build server where the built images are to be found.
    --command                       commands to execute, hyphens (-) in the command will be replaced with underscores (\_).
    --dry-run                       if set, do not execute system shell commands but print them to STDOUT.
    --dump-all-interaction-roles    dump all the interaction roles found in the configuration.
    --dump-interaction-roles        dump the found interaction roles (note that they depend on the scheduled commands).
    --dump-module-interaction-roles dump all the interaction roles found in the module of the given command.
    --dump-schedule                 dump the constructed schedule to standard output without executing the scheduled commands.
    --export-remote                 include the remote access part of exported commands.
                                    this option takes a number: 0 means all roles are exported, any other number exports only that respective role.
    --export-sh                     export the commands to a file with the given name.
    --export-sudo                   include the sudo commands when exporting commands to a file.
    --export-times                  export the times when commands are started and ended to a file with the given name.
    --export-verbose                when exporting the commands to a file, interleave them with echo commands.
    --force-rebuild                 force a rebuild regardless of the existence and build date of previously built artefacts.
    --forward-destination           the target file forward destination to copy to.
    --forward-source                the target file forward source to copy from.
    --help                          display usage information and stop execution.
    --help-build-servers            display the known build servers.
    --help-commands                 display the available commands, add a target name for restricted output.
    --help-module                   display all the available help information about the commands of the module.
    --help-field-project-name       print the field project name and exit.
    --help-options                  print the option values.
    --help-packages                 display known package and overriden package information and stop execution.
    --help-projects                 display known project information and stop execution.
    --help-targets                  display known targets and stop execution.
    --incremental                   assume an incremental build (default is yes
    --interactions                  show the interaction diagram of the commands.
    --interactions-all              show a diagram with all the commands and all the interaction roles.
    --interactions-module           show the interaction diagram of all the commands in the module.
    --interactions-module-all-roles show the interaction diagram of the commands using all the found interaction roles in the configuration.
    --packages                      packages to operate on, can be given multiple times.
    --ssh-port                      the ssh port.
    --ssh-server                    the used ssh build server.
    --ssh-user                      ssh-user on the build server (please configure your public key).
    --target                        the target to apply the given commands to.
    --tftp-directory                the target tftp directory (eg. where your device will find its kernel and rootfs).
    --verbose                       set verbosity level.

NOTES

OVERRIDE\_SRCDIR delivered packages for Buildroot targets are recognized.
```

2.8 Showing that the container works: start of a new project: Can we start a new project in the Docker container ?

The test is started with the system shell command: **mkdir workflow-test && cd workflow-test && workflow builtin start_project workflow-tests**

Expected application output:

```
Using 'workflow-tests' as name for your project.
Created a template configuration file for project 'workflow-tests'
Created a template workflow-project in 'workflow-project-template.pl' with contents:
---
#!/usr/bin/perl -w

use strict;

use warnings;

my $configuration
    = {
       field\_project\_name => 'workflow-tests',
      };

return $configuration;

---
If this looks good, please rename it to 'workflow-project.pl' using the command:

  mv 'workflow-project-template.pl' 'workflow-project.pl'

And test it with the command:

  workflow --help-field-project-name

Afterwards install the scripts on your system using the command:

  workflow builtin install\_scripts -- --engine --commands --git

Then check if they work by inspecting the examples they provide (with various options):

  workflow-tests-workflow examples array\_of\_commands\_remote\_execution --interactions

  workflow-tests-workflow examples sequencing\_and\_composition --interactions-module

  workflow-tests-workflow examples single\_command --dry-run

  workflow-tests-workflow examples array\_of\_commands --help
```

2.9 Showing that the container works: correct creation of the field project file: Have the project files been created inside the Docker container?

The test is started with the system shell command: **cd workflow-test && find .**

Expected application output:

```
.
./workflow-project-template.pl
./workflow-tests-bash-completion.sh
./workflow-tests-configuration-data
./workflow-tests-configuration-data/targets.yml
./workflow-tests-configuration-data/command\_filenames.yml
./workflow-tests-configuration-data/target\_servers.yml
./workflow-tests-configuration-data/build\_servers.yml
./workflow-tests-configuration-data/node\_configuration.yml
./conf.workflow-tests-workflow
./conf.workflow-tests-configuration
./workflow-tests-commands-data
./workflow-tests-commands-data/examples\_sh
./workflow-tests-commands-data/examples\_sh/sh\_array\_of\_commands.sh
./workflow-tests-commands-data/examples\_sh/sh\_single\_command.sh
./workflow-tests-commands-data/examples\_sh/sh\_remote\_execution.sh
./workflow-tests-commands-data/examples\_yml
./workflow-tests-commands-data/examples\_yml/remote\_execution.yml
./workflow-tests-commands-data/examples\_yml/array\_of\_commands.yml
./workflow-tests-commands-data/examples\_yml/single\_command.yml
./workflow-tests-configuration
./workflow-tests-commands
```

2.10 Showing that the container works: rename the field project file to its final name: Can we rename the project configuration files to activate the project inside the container ?

The test is started with the system shell command: **cd workflow-test && mv –verbose workflow-project-template.pl workflow-project.pl**

Expected application output:

```
renamed 'workflow-project-template.pl' -> 'workflow-project.pl'
```

2.11 Showing that the container works: correct creation of the field project file: Has the project been correctly initialized inside the container?

The test is started with the system shell command: **cd workflow-test && workflow –help-field-project-name**

Expected application output:

```
global\_field\_project\_configuration:
  field\_project\_configuration\_filename: workflow-project.pl
  field\_project\_name: workflow-tests
  from\_directory: .
  from\_executable: workflow on the command line
  technical\_project\_configuration\_directory: .
  true\_technical\_project\_configuration\_directory: /home/neurospaces/workflow-test
  true\_technical\_project\_configuration\_filename: /home/neurospaces/workflow-test/workflow-project.pl
  true\_technical\_project\_data\_commands\_directory: /home/neurospaces/workflow-test/workflow-tests-commands-data
  true\_technical\_project\_data\_configuration\_directory: /home/neurospaces/workflow-test/workflow-tests-configuration-data
```

2.12 Showing that the container works: correct installation of the new project files: Have the project files been correctly installed inside the container ?

The test is started with the system shell command: **cd workflow-test && workflow builtin install_scripts – –engine –commands**

Expected application output:

```
# ln -sf /usr/local/bin/workflow /home/neurospaces/bin/workflow-tests-workflow
#
# ln -sf /home/neurospaces/workflow-test/workflow-tests-configuration /home/neurospaces/bin/./workflow-tests-configuration
#
# ln -sf /home/neurospaces/workflow-test/workflow-tests-commands /home/neurospaces/bin/./workflow-tests-commands
#
# bash -c "echo '# workflow-tests-workflow

alias workflow-tests-workflow=\"grc workflow-tests-workflow\"
alias workflow-tests-configuration=\"grc workflow-tests-configuration\"
' | cat >>/home/neurospaces/.bashrc"
#
# bash -c "echo '. /home/neurospaces/workflow-test/workflow-tests-bash-completion.sh
' | cat >>/home/neurospaces/.bashrc"
#
# sudo     bash -c "echo '
# workflow-tests-workflow
(^|[/\w\.]+/)workflow-tests-workflow\s?
conf.workflow-tests-workflow

# workflow-tests-configuration
(^|[/\w\.]+/)workflow-tests-configuration\s?
conf.workflow-tests-configuration

' | cat >>/etc/grc.conf"
#
# sudo     ln -sf /home/neurospaces/workflow-test/conf.workflow-tests-configuration /usr/share/grc/conf.workflow-tests-configuration
#
# sudo     ln -sf /home/neurospaces/workflow-test/conf.workflow-tests-workflow /usr/share/grc/conf.workflow-tests-workflow
#
```

2.13 Showing that the container works: update .bashrc to make sure that the project specific workflow engine is found: Can we update .bashrc to make sure that the project specific workflow engine is found inside the container ?

The test is started with the system shell command: **echo "export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/neurospaces/bin" > /.bashrc**

2.14 Showing that the container works: correct installation of the new project commands: Have the project specific commands been correctly installed inside the container ?

The test is started with the system shell command: **cd workflow-test && workflow-tests-workflow –help-commands**

Expected application output:

```
'available\_commands (copy-paste the one you would like to execute, try it with the --help or the --dry-run option, or execute it without these options)':
  - workflow-tests-workflow builtin add\_target --help
  - workflow-tests-workflow builtin install\_scripts --help
  - workflow-tests-workflow builtin print\_configuration\_directory --help
  - workflow-tests-workflow builtin start\_project --help
  - workflow-tests-workflow examples array\_of\_commands --help
  - workflow-tests-workflow examples array\_of\_commands\_remote\_execution --help
  - workflow-tests-workflow examples sequencing\_and\_composition --help
  - workflow-tests-workflow examples single\_command --help
  - workflow-tests-workflow examples\_sh sh\_array\_of\_commands --help
  - workflow-tests-workflow examples\_sh sh\_remote\_execution --help
  - workflow-tests-workflow examples\_sh sh\_single\_command --help
  - workflow-tests-workflow examples\_yml array\_of\_commands --help
  - workflow-tests-workflow examples\_yml remote\_execution --help
  - workflow-tests-workflow examples\_yml single\_command --help
```

2.15 Showing that the container works: are the shell command templates installed and executed, –dry-run ?: Have the project specific commands been correctly installed inside the container ?

The test is started with the system shell command: **workflow-tests-workflow examples_sh sh_single_command –dry-run**

Expected application output:

```
/home/neurospaces/bin/workflow-tests-workflow: *** Running in dry\_run 1 mode, not executing: '/home/neurospaces/workflow-test/workflow-tests-commands-data/examples\_sh/sh\_single\_command.sh'
```

2.16 Showing that the container works: are the shell command templates installed and executed ?: Have the project specific commands been correctly installed inside the container ?

The test is started with the system shell command: **workflow-tests-workflow examples_sh sh_single_command**

Expected application output:

```
# /home/neurospaces/workflow-test/workflow-tests-commands-data/examples\_sh/sh\_single\_command.sh
#
an example of the invocation of a single command
```

2.17 Showing that the container works: are the shell command templates installed and executed from a different directory?: Have the project specific commands been correctly installed and are they executed when invoked from a different directory, inside the container ?

The test is started with the system shell command: **cd .. && workflow-tests-workflow examples_sh sh_single_command**

Expected application output:

```
# /home/neurospaces/workflow-test/workflow-tests-commands-data/examples\_sh/sh\_single\_command.sh
#
an example of the invocation of a single command
```

2.18 Showing that the container works: can we add new targets with a shell template file for their commands ?: Can we add a new target and a template for new shell commands for this target inside the container ?

The test is started with the system shell command: **workflow-tests-workflow builtin add_target – new_target "Add commands to this new target that do new things" –install-commands-sh**

Expected application output:

```
workflow-tests-workflow: added target new\_target to /home/neurospaces/workflow-test/workflow-tests-configuration-data/targets.yml
workflow-tests-workflow: created the shell command file for target new\_target
```

2.19 Showing that the container works: can we add new targets2 with a shell template file for their commands ?

The tests are started with the system shell command: **workflow-tests-workflow builtin add_target – new_target2 "Add commands to this new target2 that do new things2" –install-commands-sh**

2.20 Showing that the container works: can we add new targets2 with a shell template file for their commands ?: Can we list the known workflow projects using the regular workflow executable, inside the container ?

The test is started with the system shell command: **workflow –help-projects**

Expected application output:

```
available\_workflow automation projects (copy-paste the one you would like to get help for):
  - workflow-tests-workflow --help-commands
```