Example Documentation table of contents

Release arno.2015.1.0 (2e05184)

OPNFV

February 07, 2016

CONTENTS

Contents:

CHAPTER

ONE

HOW TO SETUP THE WORKFLOW OF AUTOMATIC DOCUMENTATION BUILD FOR YOUR PROJECT

Setup you repository and then clone locally:

```
ssh-add your-ssh.key

git clone ssh://<username>@gerrit.opnfv.org:29418/<project>

cd <project>
```

Inside the repository create the following structure::

```
gerrit.opnfv.org/<project>
                      |-- docs/
                      |     |-- some-project-description.rst
                      |     |-- other-doc-1.rst
                      |     |-- images/
                      |           |-- *.png|*.jpg
                      |-- release/
                      |     |-- some-release-doc.rst
                      |     |-- images/
                      |           |-- *.png|*.jpg
                      |-- requirements/
                      |     |-- requirements.rst
                      |     |-- images/
                      |           |-- *.png|*.jpg
                      |-- design_docs/
                      |     |-- some-design-doc.rst
                      |     |-- images/
                      |           |-- *.png|*.jpg
                      |-- some_project_file.py
                      |-- some_shell_script.sh
                      |-- INFO
                      `-- README
```

More details about the default structure you can find here at paragraph “How and where to store the document content files in your repository”.

In order to obtain a nice .html & .pdf at then end you must write you documentation using reSt markup

quick guides:

- http://docutils.sourceforge.net/docs/user/rst/quickref.html
- http://rest-sphinx-memo.readthedocs.org/en/latest/ReST.html

- http://www.math.uiuc.edu/~gfrancis/illimath/windows/aszgard_mini/movpy-2.0.0-py2.4.4/manuals/docutils/ref/rst/directives.html

An nice online editor that will help you write reSt and see your changes live. After done editing you can copy the source document in the repository and follow the workflow.

Clone the releng repository so you can created jobs for JJB:

```
git clone ssh://<username>@gerrit.opnfv.org:29418/releng
```

Enter the project settings:

```
cd releng/jjb/<project>/
```

Create the verify & build scripts

The scripts are the same for most projects and if you need customizations copy them under your project in releng/jjb/<project>/:

```
cp releng/jjb/opnfvdocs/build-docu.sh releng/jjb/<your-project>/
```

and change according to you needs.

If standard will suffice for you skip this step and jump to **Edit <your-project>.yml, Variant 1 - standard**

docu-build.sh:

```
#!/bin/bash
set -e
set -o pipefail

project="$(git remote -v | head -n1 | awk '{{print $2}}' | sed -e 's,.*:\(.*/\)\?,,' -e 's/\.git$//')"
export PATH=$PATH:/usr/local/bin/

git_sha1="$(git rev-parse HEAD)"
docu_build_date="$(date)"

files=()
while read -r -d ''; do
        files+=("$REPLY")
done < <(find * -type f -iname '*.rst' -print0)

for file in "${{files[@]}}"; do

        file_cut="${{file%.*}}"
        gs_cp_folder="${{file_cut}}"

        # sed part
        # add one '_' at the end of each trigger variable; ex: _sha1 +'_' & _date + '_' on both of th
        # they were added here without the '_'suffix to avoid sed replacement
        sed -i "s/_sha1/$git_sha1/g" $file
        sed -i "s/_date/$docu_build_date/g" $file

        # rst2html part
        echo "rst2html $file"
        rst2html --halt=2 $file | gsutil cp -L gsoutput.txt - \
        gs://artifacts.opnfv.org/"$project"/"$gs_cp_folder".html
        gsutil setmeta -h "Content-Type:text/html" \
                       -h "Cache-Control:private, max-age=0, no-transform" \
                       gs://artifacts.opnfv.org/"$project"/"$gs_cp_folder".html
        cat gsoutput.txt
```

```
        rm -f gsoutput.txt

        echo "rst2pdf $file"
        rst2pdf $file -o - | gsutil cp -L gsoutput.txt - \
        gs://artifacts.opnfv.org/"$project"/"$gs_cp_folder".pdf
        gsutil setmeta -h "Content-Type:application/pdf" \
                       -h "Cache-Control:private, max-age=0, no-transform" \
                       gs://artifacts.opnfv.org/"$project"/"$gs_cp_folder".pdf
        cat gsoutput.txt
        rm -f gsoutput.txt

done

images=()
while read -r -d '' ; do
        images+=("$REPLY")
done < <(find * -type f \( -iname \*.jpg -o -iname \*.png \) -print0)

for img in "${{images[@]}}"; do

        # uploading found images
        echo "uploading $img"
        cat "$img" | gsutil cp -L gsoutput.txt - \
        gs://artifacts.opnfv.org/"$project"/"$img"
        gsutil setmeta -h "Content-Type:image/jpeg" \
                        -h "Cache-Control:private, max-age=0, no-transform" \
                        gs://artifacts.opnfv.org/"$project"/"$img"
        cat gsoutput.txt
        rm -f gsoutput.txt

done

#the double {{ in file_cut="${{file%.*}}" is to escape jjb's yaml
```

docu-verify.sh:

```
#!/bin/bash
set -e
set -o pipefail

project="$(git remote -v | head -n1 | awk '{{print $2}}' | sed -e 's,.*:\(.*/\)\?,,' -e 's/\.git$//')
export PATH=$PATH:/usr/local/bin/

git_sha1="$(git rev-parse HEAD)"
docu_build_date="$(date)"

files=()
while read -r -d '' ; do
        files+=("$REPLY")
done < <(find * -type f -iname '*.rst' -print0)

for file in "${{files[@]}}"; do

        file_cut="${{file%.*}}"
        gs_cp_folder="${{file_cut}}"

        # sed part
        # add one '_' at the end of each trigger variable; ex: _sha1 +'_' & _date + '_' on both of t
```

```
        # they were added here without the '_'suffix to avoid sed replacement
        sed -i "s/_sha1/$git_sha1/g" $file
        sed -i "s/_date/$docu_build_date/g" $file

        # rst2html part
        echo "rst2html $file"
        rst2html --exit-status=2 $file > $file_cut".html"

        echo "rst2pdf $file"
        rst2pdf $file -o $file_cut".pdf"

done

#the double {{ in file_cut="${{file%.*}}" is to escape jjb's yaml
```

Edit <your-project>.yml:

```
vi releng/jjb/<your-project>/<your-project>.yml
```

Make sure you have the job-templates set correctly as below.

example:: vi releng/jjb/opnfvdocs/opnfvdocs.yml # make sure you are using one of the variants below and that !include-raw directive is present

CHAPTER

TWO

VARIANT 1 - STANDARD

By chosing **Variant 1** you will use the scripts from opnfvdocs project.

<your-project>.yml:

```
- job-template:
   name: 'opnfvdocs-daily-{stream}'

   node: master
   ...
   builders:
       - shell:
           !include-raw ../opnfvdocs/docu-build.sh

- job-template:
   name: 'opnfvdocs-verify'

   node: master
   ...
   builders:
       - shell:
           !include-raw ../opnfvdocs/docu-verify.sh

- job-template:
   name: 'opnfvdocs-merge'

   node: master
   ...
   builders:
       - shell:
           !include-raw ../opnfvdocs/docu-build.sh
```

CHAPTER

THREE

VARIANT 2 - CUSTOM

<your-project>.yml:

```
- job-template:
   name: 'opnfvdocs-daily-{stream}'

   node: master
   ...
   builders:
       - shell:
           !include-raw docu-build.sh

- job-template:
   name: 'opnfvdocs-verify'

   node: master
   ...
   builders:
       - shell:
           !include-raw docu-verify.sh

- job-template:
   name: 'opnfvdocs-merge'

   node: master
   ...
   builders:
       - shell:
           !include-raw docu-build.sh
```

“node: master” is important here as all documentations are built on Jenkins master node for now.

Please reffer to the releng repository for the correct indentation as JJB is very picky with those and also for the rest of the code that is missing in the example code and replaced by ”...”. Also you must have your documentation under docs/ in the repository or gsutil will fail to copy them; for customizations you might need to addapt build-docu.sh as we did for genesis project as different documents need to go into different places.

Stage files example:

```
git add docu-build.sh docu-verify.sh <project>.yml
```

Commit change with –signoff:

```
git commit --signoff
```

Send code for review in Gerrit:

```
git review -v
```

Create the documentation using the recommended structure in your repository and submit to gerrit for review

Jenkins will take over and produce artifacts in the form of .html & .pdf

Jenkins has the proper packages installed in order to produce the artifacts.

Artifacts are stored on Google Storage (still to decide where, structure and how to present them)

http://artifacts.opnfv.org/

Here you can download the PDF version of this guide.

Scrape content from html artifacts on wiki

This section describes how the html build artifacts can be made visible on Wiki using he scrape method. DokuWiki speeds up browsing through the wiki by caching parsed files1). If a currently cached version of a document exists, this cached copy is delivered instead of parsing the data again. On editing and previewing no cache is used.

To prevent a page from ever being cached, use the NOCACHE tag anywhere in the document. This is useful if the page contains dynamic content, e.g. PHP code that pulls in outside information, where the caching would prevent the most recent information from being displayed. Same applies if documentation artifacts are rebuilt the cached version is shown if the NOCACHE tag is not used.

https://www.dokuwiki.org/caching

In order to have you documentation on Wiki you need to create a wiki page and include an adaption of the code below:

example:

```
~~NOCACHE~~

{{scrape>http://artifacts.opnfv.org/opnfvdocs/docs/enable_docu_gen.html}}
```

Please try to write documentation as accurate and clear as possible as once reviewed and merged it will be automatically built and displayed on Wiki and everyone would apreciate a good written/nice looking guide.

If you want to see on wiki what code is scraped from the built artifacts click “Show pagesource” in the right (it will appear if you hover over the magnifier icon); this way you know what is written straight on wiki and what is embedded with “scrape”. By knowing these details you will be able to prevent damages by manually updating wiki.

Wiki update - how it works

Edit Wiki page https://wiki.opnfv.org/<page> and look for {{scrape>http://artifacts.opnfv.org/<project>/<folder>/<docfile>.html}} Click “Preview” and see if the change you submitted to Git is present; add a short description in “Edit summary” field, then click “Save” to update the page. This extra step is needed as Wiki does not auto update content for now.

How to track documentation

You must include at the bottom of every document that you want to track the following:

```
**Documentation tracking**

Revision:

Build date:  _date_
```

Image inclusion for artifacts

Create a folder called images in the same folder where you documentation resides and copy .jpg or .png files there, according to the guide here: https://wiki.opnfv.org/documentation

Here is an example of what you need to include in the .rst files to include an image:

```
.. image:: images/smiley.png
   :height: 200
   :width: 200
   :alt: Just a smiley face!
   :align: left
```

The image will be shown in both .html and .pdf resulting artifacts.

CHAPTER

FOUR

NOTE:

In order to generate html & pdf documentation the needed packages are rst2pdf & python-docutils if the Jenkins is CentOS/RHEL; many variants have been tested but this is the cleanest solution found. For html generation it also supports css styles if needed.

Documentation tracking

Revision:

Build date: _date_

CHAPTER

FIVE

PROJECT NAME: DOCUMENTATION

- Proposed name for the project: ‘’opnfv documentation’‘
- Proposed name for the repository: ‘’opnfvdocs’‘
- Project Categories: Documentation

5.1 Project description:

- Produce documentation for OPNFV releases including but not limited to:
  - Release notes
  - Installation guide
  - User guide
    - Any relevant references and interface specifications for OPNFV projects or components.
  - Include any architecture diagrams or specifications, reference to OPNFV requirements list.
  - Provide guidelines and tooling for documentation handling across all OPNFV projects

5.2 Scope:

- Set up a structure, and a template, for document development with source control (same as source code). Leveraging upstream documentation structure and tools.
- Following as close as possible the same contribution process & tools as our source code
- Structure OPNFV documentation logically
- Develop initial set of release documents:
  - Release note
  - Install guide
  - User Guide
  - API reference (if there is content in release 1)
  - Interface specification (if there is content in release 1)
- Provide language options for documentation where applicable: In first release English only, Wiki (via HTML scraping from Gerrit), and PDF.
- Provide tooling and processes for OPNFV projects to implement and follow for consistency

5.3 Dependencies:

- All OPNFV projects participating in a release.
- Upstream project documentation to be referenced
- Where there are external fora or standard development organization dependencies, list informative and normative references & specifications.

5.4 Committers and Contributors:

- Name of and affiliation of the project leader :
  - Christopher Price: christopher.price@ericsson.com
- Names and affiliations of the committers
  - Christopher Price: christopher.price@ericsson.com
  - Wenjing Chu (Dell): wenjing_chu@dell.com
  - Ashiq Khan (NTTdocomo): khan@nttdocomo.com
  - Fatih Degirmenci: fatih.degirmenci@ericsson.com
  - Rodriguez, Iben: Iben.Rodriguez@spirent.com
  - Malla Reddy Sama: sama@docomolab-euro.com
- Any other contributors
  - Bryan Sullivan (AT&T)
  - Trevor Cooper: trevor.cooper@intel.com

Description of roles in the documentation project:

- Committers (Editors): has overall responsibility of document structure, editing, style and toolchains
- opnfvdocs contributors: individual section will have contributors who are domain experts in those areas, other contributors may simply help out working on the documentation and tools as needed.
- other projects: Committers will be responsible for maintaining documentation artifacts in project repositories.

5.5 Planned deliverables

- Project release documentation for OPNFV
  - Including collation of all release relevant project documentations
- Establishment and maintenance of the OPNFV documentation processes and toolchains

5.6 Proposed Release Schedule:

- opnfvdocs will follow each OPNFV release and produce needed documentation
  - Release 1 will provide basic documentation including revision control.
  - By release 2 a multi-project toolchain will be in place with processes and version control

- Iterative improvements to the processes and toolchains are expected on a release by release basis.

Documentation tracking

Revision:

Build date: _date_

CHAPTER

SIX

CREATING/CONFIGURING/VERIFYING JENKINS JOBS

Clone the repo:

```
git clone ssh://YOU@gerrit.opnfv.org:29418/releng
```

make changes:

```
git commit -sv
git review
remote: Resolving deltas: 100% (3/3)
remote: Processing changes: new: 1, refs: 1, done
remote:
remote: New Changes:
remote:   https://gerrit.opnfv.org/gerrit/51
remote:
To ssh://agardner@gerrit.opnfv.org:29418/releng.git
 * [new branch]      HEAD -> refs/publish/master
```

Follow the link to gerrit https://gerrit.opnfv.org/gerrit/51 in a few moments the verify job will have completed and you will see Verified +1 jenkins-ci in the gerrit ui.

If the changes pass the verify job https://build.opnfv.org/ci/view/builder/job/builder-verify-jjb/ The patch can be submitited by a committer.

Job Types

- Verify Job
- Trigger: **recheck** or **reverify**
- Merge Job
- Trigger: **remerge**

The verify and merge jobs are retriggerable in Gerrit by simply leaving a comment with one of the keywords listed above. This is useful in case you need to re-run one of those jobs in case if build issues or something changed with the environment.

You can add below persons as reviewers to your patch in order to get it reviewed and submitted.

- Ulrich Kleber (Ulrich.Kleber@huawei.com)
- Fatih Degirmenci (fatih.degirmenci@ericsson.com)
- Xinyu Zhao(Jerry) (zhaoxinyu@huawei.com)

Or just email a request for submission to opnfv-helpdesk@rt.linuxfoundation.org

The Current merge and verify jobs for jenkins job builder as pulled from the repo:

```
- project:
   name: builder-jobs
   jobs:
       - 'builder-verify-jjb'
       - 'builder-merge'

   project: 'releng'

- job-template:
   name: builder-verify-jjb

   project-type: freestyle

   logrotate:
       daysToKeep: 30
       numToKeep: 10
       artifactDaysToKeep: -1
       artifactNumToKeep: -1

   parameters:
       - project-parameter:
           project: '{project}'
       - gerrit-parameter:
           branch: 'master'
   scm:
       - gerrit-trigger-scm:
           credentials-id: '{ssh-credentials}'
           refspec: '$GERRIT_REFSPEC'
           choosing-strategy: 'gerrit'

   wrappers:
       - ssh-agent-credentials:
           user: '{ssh-credentials}'

   triggers:
       - gerrit:
           trigger-on:
               - patchset-created-event:
                   exclude-drafts: 'false'
                   exclude-trivial-rebase: 'false'
                   exclude-no-code-change: 'false'
               - draft-published-event
               - comment-added-contains-event:
                   comment-contains-value: 'recheck'
               - comment-added-contains-event:
                   comment-contains-value: 'reverify'
           projects:
             - project-compare-type: 'ANT'
               project-pattern: 'releng'
               branches:
                 - branch-compare-type: 'ANT'
                   branch-pattern: '**/master'
               file-paths:
                   - compare-type: ANT
                     pattern: jjb/**
                   - compare-type: ANT
                     pattern: jjb-templates/**
```

```
  builders:
      - shell: |
          source /opt/virtualenv/jenkins-job-builder/bin/activate
          jenkins-jobs test /opt/jenkins-ci/builder/

- job-template:
  name: 'builder-merge'

  # builder-merge job to run JJB update
  #
  # This job's purpose is to update all the JJB

  project-type: freestyle

  logrotate:
      daysToKeep: 30
      numToKeep: 40
      artifactDaysToKeep: -1
      artifactNumToKeep: 5

  parameters:
      - project-parameter:
          project: '{project}'
      - gerrit-parameter:
          branch: 'master'

  scm:
      - gerrit-trigger-scm:
          credentials-id: '{ssh-credentials}'
          refspec: ''
          choosing-strategy: 'default'

  wrappers:
      - ssh-agent-credentials:
          user: '{ssh-credentials}'

  triggers:
      - gerrit:
          trigger-on:
              - change-merged-event
              - comment-added-contains-event:
                  comment-contains-value: 'remerge'
          projects:
            - project-compare-type: 'ANT'
              project-pattern: 'releng'
              branches:
                  - branch-compare-type: 'ANT'
                    branch-pattern: '**/master'
              file-paths:
                  - compare-type: ANT
                    pattern: jjb/**

  builders:
      - shell: |
              source /opt/virtualenv/jenkins-job-builder/bin/activate
              cd /opt/jenkins-ci/releng
              git pull
              jenkins-jobs update --delete-old jjb/
```

Documentation tracking

Revision:

Build date: _date_

CHAPTER

SEVEN

OTHER OPTIONS TO GENERATE DOCUMENTATION THAT WE TESTED

Doxygen plugin -> HTML published plugin (html)/ LaTeX (pdf)

Description: This was the first discovered method

- html: using Doxygen plugin + HTML publisher It involves some customization at doxygen level + custom html header/footer
- pdf: it generates a .pdf using latex
- Input files: .md , .rst
- Output: .html & .pdf
- Pros:
  - standard tools: doxygen, html publisher, LaTeX suite
  - doxygen plugin available in Jenkins, you just need to install it; html publisher plugin available in Jenkins, you just need to install it
  - destination files are generated fast
  - standard reStructuredText or Markdown
- Cons:
  - takes some time to customize the output in matters of template, requires custom html header/footer
  - latex suite is quite substantial in amount of packages and consumed space (around 1.2 GB)
- Tested: roughly, functional tests only

Maven & clouddocs-maven-plugin (actually used to generate openstack-manuals)

Description: It represents the standard tool to generate Openstack documentation manuals, uses maven, maven plugins, clouddocs-maven-plugins; location of finally generated files is the object of a small Bash script that will reside as Post-actions

- Input files: .xml
- Output: .html & .pdf
- Pros:
  - quite easy for initial setup
  - uses openstack documentation generation flows as for openstack-manuals (clouddocs-maven-plugin), maven installs all you need generate the documentation
- Cons:

  - could be tricky to generate a custom layout, knowledge about Maven plugins required, .pom editing
  - dependent of multiple maven plugins
  - input files are .xml and xml editing knowledge is required
- Tested: roughly, functional tests only

Sphinx & LaTeX suite

Description: The easiest to install, the cleanest in matter of folder & files structure, uses standard tools available in repositories; location of finally generated files is the object of a small Bash script that will reside as Post-actions

- Input files: .rst as default
- Output: .html & .pdf
- Pros:
  - standard tools: Python Sphinx, LaTeX suite
  - destination files are generated fast
  - standard reStructuredText as default; other inputs can be configured
  - Sphinx's installation is very clean in matters of folder structure; the cleanest from all tested variants
  - latex suite is also easy to install via yum/apt and available in general repos
  - everyone is migration from other tools to Spinx lately; it provides more control and better looking documentation
  - can be used also for source-code documentation, specially if you use Python
- Cons:
  - takes some time to customize the output in matters of template, requires custom html header/footer
  - latex suite is quite substantial in amount of packages and consumed space (around 1.2 GB)
- Tested: roughly, functional tests only

Documentation tracking

Revision:

Build date: _date_

CHAPTER

EIGHT

INDICES AND TABLES

- search

Revision:

Build date: February 07, 2016