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 the lines below
        # 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 the lines below
        # 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

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
```

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>/<doc-file>.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: 256683072fb4c5b0bba36ddfd273b9bab0213945

Build date:  Tue Nov 10 18:42:38 UTC 2015
```

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.

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: 256683072fb4c5b0bba36ddfd273b9bab0213945

Build date: Tue Nov 10 18:42:38 UTC 2015