

Goodman HTS Pipeline Developer Manual

version 0.1

Simon Torres

July 05, 2017

Contents

Goodman High Throughput Spectrograph Pipeline - Developer Manual	1
Version Numbering: Proposal	1
Introduction	1
Pipeline Versioning	1
Summary and Schedule	1
How to create a <code>requirements.txt</code> file	1
Documentation using Sphinx	2
Install Sphinx	2
Configure PyCharm to do it	4
Local Management - How to use github pages	4
Create Directory Structure	4
Clone the repository	4
Modifying the Makefile	5
How to include Markdown README to index	5
Convert from Markdown to reStructuredText	6
Building the <code>html</code> Documentation	6
Final Thoughts	7
Setting up VNC Servers	7
Login as root or administrative user	7
Install GNOME	7
Install Tigervnc	7
Configure the VNC user	7
Configuring the Firewall	8
Setting the VNC password	9
Enable Service	9
Log in using your new vnc account	9
User and Developer Manuals	9
Preparation	9
Get the source Files	9
Editing the Manuals	10
Building the PDF Files	10
Commit to GitHub	11

Goodman High Throughput Spectrograph Pipeline - Developer Manual

This document is the User Manual for Goodman HTS Pipeline. It will provide a quick overview of the main features of the pipeline, installation instructions, restrictions and guidelines for using it.

Version Numbering: Proposal

Introduction

Version number can be a confusing matter, specially because there are many ways to do it and at the end it is a matter of personal (or team choice). After doing some digging in python PEPs and internet in general I found this [Wikipedia site on Software Versioning](#) that to me makes a lot of sense and what I propose here is the following:

Pipeline Versioning

Let's define our goal to be the version 1.0 as the one that will be stable, portable and also must contain an automatic wavelength calibration module as well as a flux calibration module. Besides, the list of issues tagged as bug, or enhancements should be at least 70 ~ 90% closed. That is a large gap but some enhancements might be of less importance, bugs will be naturally prioritized.

According to the [Wikipedia site](#) and the current status of our development the number would be something like this:

1.0a1 which can be read as **“we are developing towards version 1.0 and currently we are working on the alpha version of our pipeline”**.

For the first beta release the version number will be 1.0b1 and as we add improvements and fix bugs we can do for instance 1.0b2 which would be considered a new beta release, or in other words *beta version 2*

Right now numbering is a mere formality and I have been experimenting in introducing them since some time ago, but they will become very important when we go public, for the first beta release.

Summary and Schedule

- 1.0a1 until the end of May 2017.
- 1.0b1 From beginning of June until end of August (last digit may vary)
- 1.0 Sometime between September and before the end of the year (ideally October).

How to create a requirements.txt file

Warning

DO THIS ALWAYS BEFORE A RELEASE

The file `requirements.txt` is necessary to facilitate other users the task of setting up the system to get your *Shiny Python Code* working for them. It is basically a list of libraries with their respective version numbers.

We came across a solution to do this automatically which is:

```
pip freeze > requirements.txt
```

This works fine but it saves ALL the libraries in the environment and not only for the project (this might be useful to save the environment though).

I found a better way to do it, also automatically:

```
sudo pip install pipreqs
```

Once installed do:

```
pipreqs /path/to/the/repository
```

For the Goodman Pipeline the list also contained `pygtk` but it fails. Simply delete it. PyGTK should come by default in most systems.

Documentation using Sphinx

We have chosen to use Google's style for writing the Python Docstrings because is easy to read in the code and also allows to generate documentation in different formats. The guidelines for writing Docstrings are stated in the [PEP257](#) But no style is defined in there.

By default, Sphinx uses *reStructuredText* for Python's Docstrings. But it is a bit hard to read, for instance:

```
def some_function(name, state):  
    """This function does something.  
  
    :param name: The name to use.  
    :type name: str.  
    :param state: Current state to be in.  
    :type state: bool.  
    :returns: int -- the return code.  
    :raises: AttributeError, KeyError  
  
    """  
    return True
```

The same function with Google Style Docstrings would be:

```
def some_function(name, state):  
    """This Function does something  
  
    Args:  
        name (str): The name to use.  
        state (bol): Current State to be in.  
  
    Returns:  
        int: The return code.  
  
    Raises:  
        AttributeError: Some description  
        KeyError: Some other description  
  
    """  
    return True
```

Everyone in our team agrees that Google style is easier to read. Sphinx's style produces better formatted output and gives you more control on formatting.

The good news is that it is possible to produce documentation using Google Style and Sphinx. There are a few necessary steps to achieve this:

Install Sphinx

1. Install sphinx.

```
pip install sphinx
```

2. Install Napoleon Plugin. Allows to use Google style with Sphinx.

```
pip install sphinxcontrib.napoleon
```

3. Go to your source folder and run `sphinx-quickstart` and adapt the

following answers to your needs. (some parts were omitted)

```
$ sphinx-quickstart
Welcome to the Sphinx 1.4.5 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
> Root path for the documentation [.] : docs

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n] : y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other static
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_] :

The project name will occur in several places in the built documentation.
> Project name: MyProject
> Author name(s): The Author
...
> Project version: 1.0
> Project release [1.0]:
...
> Create Makefile? (y/n) [y] :
```

4. A file `conf.py` has been created in the folder `docs`. In there you need to change some settings:

- Add `sphinxcontrib.napoleon`
- Insert `$PATH` variable into the system
- Add `napoleon` settings

```
# conf.py

# Add autodoc and napoleon to the extensions list
extensions = ['sphinx.ext.autodoc', 'sphinxcontrib.napoleon']

# Add source directory to PATH (where the modules are)
import sys
sys.path.append('/path/to/source')

# Napoleon settings
napoleon_google_docstring = True
napoleon_numpy_docstring = True
napoleon_include_init_with_doc = False
napoleon_include_private_with_doc = False
napoleon_include_special_with_doc = False
napoleon_use_admonition_for_examples = False
napoleon_use_admonition_for_notes = False
napoleon_use_admonition_for_references = False
napoleon_use_ivar = False
napoleon_use_param = True
napoleon_use_rtype = True
napoleon_use_keyword = True
```

5. Use `sphinx-apidoc` to build your documentation.

```
sphinx-apidoc -f -o docs/source projectdir
```

Or as I did from the home directory:

```
sphinx-apidoc -f -o ~/development/soar/goodman/docs/source \  
~/development/soar/goodman
```

6. Make your documentation: Go to `~/development/soar/goodman/docs` and run the command below. (There are more options that you can use).

```
cd ~/development/soar/goodman/docs  
make html
```

Configure PyCharm to do it

7. In PyCharm go to `Run > Edit Configurations...` and then click the green **+** symbol in the upper left corner (*Pycharm Community Edition 2016.2*) and choose `Python Docs > Sphinx Task`

8. A new dialog will open and it should look like this:

![docs-configuration](./docs/img/configurations.png)

Apply or click OK. In the upper right corner of pycharm you should find the option for `build-documentation` if you used the same name for this particular configuration.

Local Management - How to use github pages

Warning

This section is incomplete if you need assistance in this regard please contact me

In order to minimize the chance of making a mistake I suggest the following configuration.

Just to be sure we are talking the same, I want to remind you that the symbol `~` represent your home directory, for instance, if your user name is `user` its home directory will be `/home/user`. In GNU/Linux, this is by default but beware that it might change, a simple *trick* to find out your home directory, in case you are in doubt, is to do the following:

Open a terminal and type these two commands

```
cd
```

By default this will take you to your home directory

```
pwd
```

This command does *Print Working Directory*

Create Directory Structure

I suggest using separate directories for *code development* and for *building documentation*. The command sequence is as follows.

For the **code**, create the folder `~/development/soar/`

```
mkdir -p ~/development/soar/
```

For the **documentation** is similar:

```
mkdir -p ~/documentation/soar
```

Clone the repository

At this point we need to run `git clone` in both location.

Go to the code folder:

```
cd ~/development/soar/
```

Clone the repository, for this you need to setup your github account first.

```
git clone git@github.com:soar-telescope/goodman.git
```

You will have now a folder named `goodman`

Now go to the documentation folder and repeat the `clone` command:

```
cd ~/documentation/soar/
```

```
git clone git@github.com:soar-telescope/goodman.git
```

Again you will have the `goodman` directory but there is an extra step to do here.

Go into the `goodman` directory and checkout the branch `gh-pages`

```
cd goodman
```

```
git checkout gh-pages
```

This will bring all the files related to [GitHub Pages](#)

Inside your code development directory is a folder called `docs`. Here is where you run `sphinx-quickstart`. After this it will contain a file named `Makefile` and two folders `source` and `build`

Modifying the Makefile

Modify `Makefile` to build the documentation in a different location. Open the file using your favourite editor and change the `BUILDDIR` variable to:

```
BUILDDIR      = /HOME/USER/documentation/soar/goodman
```

It should look like this: (Note that I changed the name of the home directory for security reasons, change `/HOME/USER` by whatever matches your system)

```
# Minimal makefile for Sphinx documentation
#

# You can set these variables from the command line.
SPHINXOPTS    =
SPHINXBUILD   = sphinx-build
SPHINXPROJ    = GoodmanPipelines
SOURCEDIR     = source
BUILDDIR      = /HOME/USER/documentation/soar/goodman

# Put it first so that "make" without argument is like "make help".
help:
    @$(SPHINXBUILD) -M help "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)

.PHONY: help Makefile

# Catch-all target: route all unknown targets to Sphinx using the new
# "make mode" option. $(O) is meant as a shortcut for $(SPHINXOPTS).
%: Makefile
    @$(SPHINXBUILD) -M $@ "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O)
```

How to include Markdown README to index

Although I will recommend to write an `.rst` file instead, I will include the necessary steps to include a Markdown README. I didn't get the results I wanted with Markdown.

1. Install `recommonmark`

```
sudo pip2.7 install recommonmark
```

2. Edit Sphinx's `conf.py` and add the following lines.

```
from recommonmark.parser import CommonMarkParser

source_suffix = ['.rst', '.md']

source_parsers = {
    '.md': CommonMarkParser,
}
```

4. go to `docs/sources` and add the following line to `index.rst`:

```
.. include:: README.md
```

So it will look like this:

```
Welcome to Goodman Spectroscopic Tools's documentation!
=====

.. include:: README.md

Contents:

.. toctree::
    :maxdepth: 2

Indices and tables
=====

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Convert from Markdown to reStructuredText

But you can also convert your Markdown README to `.rst` using `pandoc`.

```
sudo yum install pandoc
```

And you can use it with:

```
pandoc --from=markdown --to=rst --output=README.rst README.md
```

Converting Markdown to `.rst` worked better for me. Also I would recommend some editing since you don't want all the README information to go to the web page, such as the link to the page itself for instance. Also some checks are needed to make sure the translation from *Markdown* to *reStructuredText* was correct.

Building the html Documentation

Use `make html` in your documentation's root folder (where the Makefile is located)

```
cd ~/development/soar/goodman/docs
make html
```

The new html documentation will be located at `~/documentation/soar/goodman/html`. You need to move all its content to the parent directory i.e. `~/documentation/soar/goodman`

```
cd ~/documentation/soar/goodman
cp -vr html/* .
```

6. Finally in add commit and push

Add all new files to the repository

```
git add *
```

Commit the new changes. Try to use a meaningful message.

```
git commit -m "updating html documentation"
```

Push to the repository

```
git push
```

Then you can see the page in <https://soar-telescope.github.io/goodman/>

Final Thoughts

These tools are highly customizable so expect some troubles setting this up. Keep your eyes open and read very well the debug or feedback on the problem.

Setting up VNC Servers

The SOAR Telescope Data Reduction Servers run Centos 7, below are the instructions to set it up. The original instructions were obtained from [this website](#)

Login as root or administrative user

A non-root administrative user was provided for basic server management, I would recommend not going further than installing necessary programs or system configurations such as the one described in this document. For any other aspects please contact CISS. For passwords please contact your supervisor to get the appropriate documents.

```
ssh sdevel@soardataX
```

Where x can be 1, 2 or 3

Install GNOME

For this application we have chosen GNOME which can be installed with the following command.

```
sudo yum groupinstall "GNOME Desktop"
```

Answer yes or y to approve the installation

Install Tigervnc

```
sudo yum install tigervnc-server
```

Configure the VNC user

For this example we will be using the username goodman.

```
sudo useradd goodman
```

In order to add a password, do:

```
sudo passwd goodman
```

Type the password for the goodman user and then verify it by typing it again.

Copy the default vncserver configuration file with the following command.

```
sudo cp /lib/systemd/system/vncserver@.service \  
/etc/systemd/system/vncserver@:1.service
```

And now edit it with your favorite text editor. My preferred text editor for a terminal is vim.

```
sudo vim /etc/systemd/system/vncserver@:1.service
```

replace all <USER> appearances by goodman

```
1 # The vncserver service unit file  
2 #
```

```
3 # Quick HowTo:
4 # 1. Copy this file to /etc/systemd/system/vncserver@.service
5 # 2. Edit /etc/systemd/system/vncserver@.service, replacing <USER>
6 #    with the actual user name. Leave the remaining lines of the file unmodified
7 #    (ExecStart=/usr/sbin/runuser -l <USER> -c "/usr/bin/vncserver %i"
8 #    PIDFile=/home/<USER>/.vnc/%H%i.pid)
9 # 3. Run `systemctl daemon-reload`
10 # 4. Run `systemctl enable vncserver@:<display>.service`
11 #
12 # DO NOT RUN THIS SERVICE if your local area network is
13 # untrusted! For a secure way of using VNC, you should
14 # limit connections to the local host and then tunnel from
15 # the machine you want to view VNC on (host A) to the machine
16 # whose VNC output you want to view (host B)
17 #
18 # [user@hostA ~]$ ssh -v -C -L 590N:localhost:590M hostB
19 #
20 # this will open a connection on port 590N of your hostA to hostB's port 590M
21 # (in fact, it ssh-connects to hostB and then connects to localhost (on hostB)).
22 # See the ssh man page for details on port forwarding)
23 #
24 # You can then point a VNC client on hostA at vncdisplay N of localhost and with
25 # the help of ssh, you end up seeing what hostB makes available on port 590M
26 #
27 # Use "--nolisten tcp" to prevent X connections to your VNC server via TCP.
28 #
29 # Use "--localhost" to prevent remote VNC clients connecting except when
30 # doing so through a secure tunnel. See the "--via" option in the
31 # `man vncviewer` manual page.
32 #
33 #
34 [Unit]
35 Description=Remote desktop service (VNC)
36 After=syslog.target network.target
37
38 [Service]
39 Type=forking
40 # Clean any existing files in /tmp/.X11-unix environment
41 ExecStartPre=/bin/sh -c '/usr/bin/vncserver -kill %i > /dev/null 2>&1 || :'
42 ExecStart=/usr/sbin/runuser -l <USER> -c "/usr/bin/vncserver %i"
43 PIDFile=/home/<USER>/.vnc/%H%i.pid
44 ExecStop=/bin/sh -c '/usr/bin/vncserver -kill %i > /dev/null 2>&1 || :'
45
46 [Install]
47 WantedBy=multi-user.target
```

So lines 42 and 43 will look like this now:

```
1 ExecStart=/usr/sbin/runuser -l goodman -c "/usr/bin/vncserver %i"
2 PIDFile=/home/goodman/.vnc/%H%i.pid
```

Configuring the Firewall

If you have a firewall you need to do the following commands:

```
firewall-cmd --permanent --zone=public --add-service vnc-server
firewall-cmd --reload
```

Setting the VNC password

Login as goodman

```
su goodman
```

And run vncserver, this will ask you the VNC password, don't confuse it with your *user* password.

```
vncserver
```

Type your password and verify it by retyping it when requested.

Enable Service

Exit the goodman user

```
exit
```

Now you should be logged in as the sdevel user.

```
systemctl daemon-reload systemctl enable vncserver@:1.service reboot
```

You will loose conection after this, wait for a reasonable time (1 minute or so) and then log in again as sdevel

```
ssh sdevel@soardataX
```

Remember that x can be 1, 2 or 3 and should be the same that you used at the beginning.

```
systemctl start vncserver@:1.service
```

Log in using your new vnc account

Using your vnc client of choice try to connect to the server. I will be using vncviewer for this example:

```
vncviewer soardataX:1
```

Remember that x has to be the same of the beginning (1, 2 or 3).

User and Developer Manuals

The user and developer manuals are created using Sphinx. The source files are written in *reStructuredText* a markup language with simple syntaxis that allows for rich formatting. You can start [here](#) Although the language's syntaxis is simple you need to familiarize with it, or you might get scared away, but don't worry, is really easy.

Preparation

I recommend using [PyCharm](#) I will not provide instruccions for installing it but is easy and you can find instructions in their site. Also because the installation depends on the platform.

You will also need a [GitHub](#) account and very important [Register your public ssh key](#) on GitHub.

And finally define a directory structure, here I present my suggestion but feel free to ignore this in case you already have yours. Keep in mind here that the most important here is to have *something that works* for you.

```
mkdir -p ~/documentation/soar
```

```
cd ~/documentation/soar
```

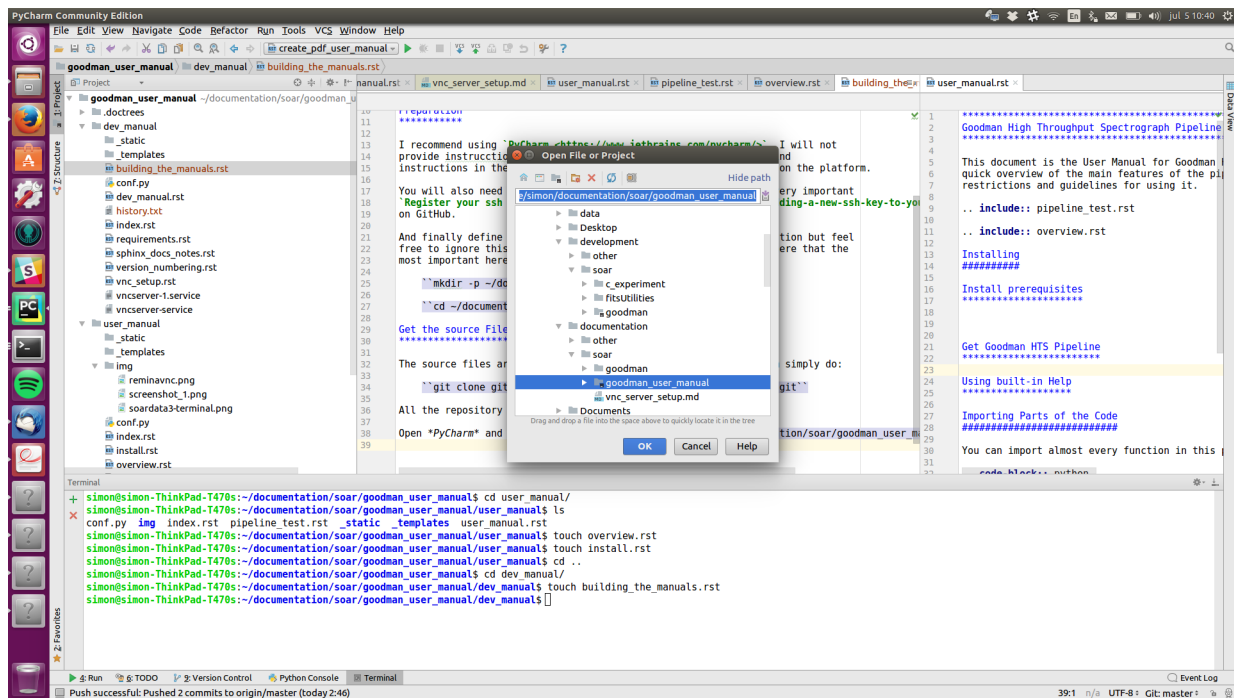
Get the source Files

The source files are stored on GitHub.com, in order to obtain them simply do:

```
git clone git@github.com:soar-telescope/goodman_user_manual.git
```

All the repository will be downloaded to goodman_user_manual

Open *PyCharm* and go to File > Open Navigate to ~/documentation/soar/goodman_user_manual and click *Open*



Choose *Open in a new window* when asked.

Editing the Manuals

This [rst cheatsheet](#) seems a very good resource to start. I will not go into further details here.

Building the PDF Files

In order to produce the pdfs, first you need to configure *PyCharm*. Go to `Run > Edit Configurations...`

- Click the green plus sign and select `Python Docs... > Sphinx` task.
- Edit the fields so they will look like this:

- For the Developers Manual

Name: `create_pdf_dev_manual`

Command: `pdf`

Input: `~/documentation/soar/goodman_user_manual/dev_manual/`

Output: `~/documentation/soar/goodman_user_manual`

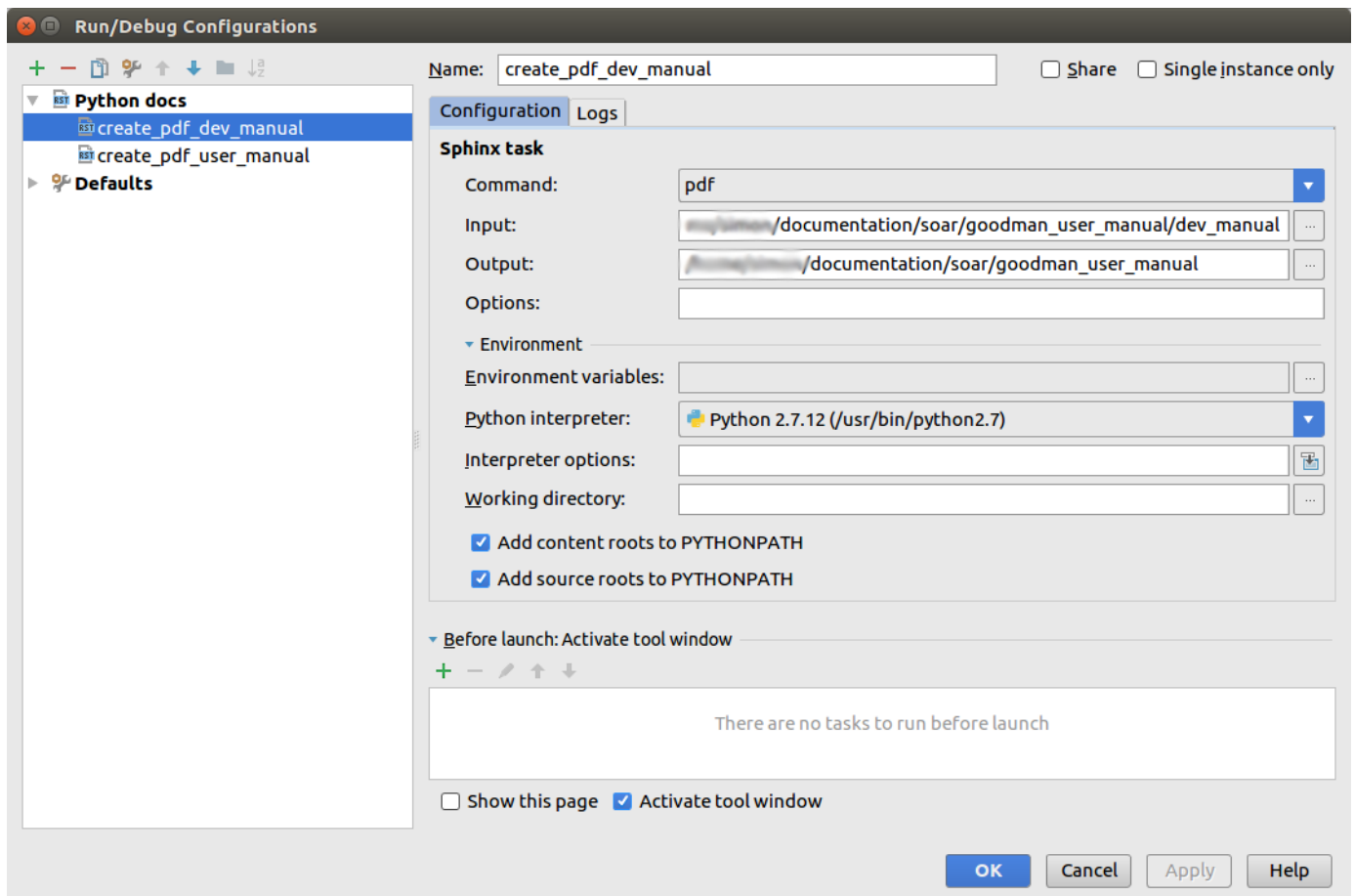
- For the Users Manual:

Name: `create_pdf_user_manual`


Command: `pdf`

Input: `~/documentation/soar/goodman_user_manual/user_manual/`

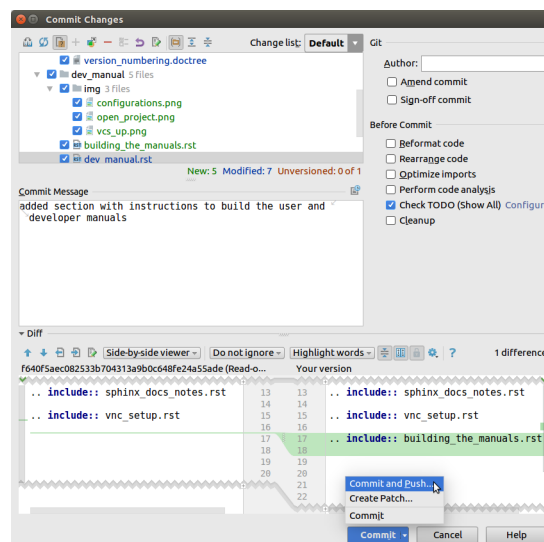
Output: `~/documentation/soar/goodman_user_manual`



Commit to GitHub

Once you have completed a feature click this button  to do a commit to your repository.

This will open a pop up window. In the upper left region you see the modified files, below this you type a *Commit Message*, this has to be precise, not too short not too long, just a reminder of what you did. At the bottom you can see the changes you are introducing in the file selected in the top left section of the window.



If you select *Commit and Push* this will update the local repository as well as the remote, i.e, the GitHub hosted version. If you choose *Commit* only the changes will not be uploaded. Later you can go to `VCS > Git > Push...`