COMET LiCSAR

CEMAC

cemac-support@leeds.ac.uk

CONTENTS

Notes regarding the COMET LiCSAR system

CHAPTER

ONE

SOFTWARE VERSION CONTROL

The GitLab service is currently used for hosting Git repositories related to the operation of the COMET LiCSAR system.

1.1 Current GitLab Usage

The COMET LiCSAR related material is stored within a GitLab *group* at:

> https://gitlab.com/comet_licsar

There are currently 11 repositories, 5 of which are publicly visible.

Several of the repositories have not been updated in some time (one year or more), but this may be due to the content not requiring regular updates.

The licsar_proc, licsar_framebatch and licsar_documentation repositories appear to be the most active and frequently updated.

The *licsar_documentation* repository makes use of the GitLab Wiki functionality. This allows the documentation to be created in markdown format, and rendered by the GitLab web interface.

1.2 Git Hosting Options

There are various options available for hosting Git repositories, with GitHub and GitLab being the most popular services available, and the two which we will consider.

Both services have been around for several years, and can be considered stable and reliable. GitHub is the most well known and popular of the two services, and is owned by Microsoft.

1.3 Feature Comparison

The features offered at the non charged level of hosting by GitHub and GitLab are both very similar.

- GitLab Pricing information
- Github Pricing information

In the past GitLab allowed private repositories at the non charged level, where as GitHub did not, and this could have been one of the main reasons why GitLab was originally chosen for the COMET LiCSAR repositories.

Where GitLab allows repositories to be organised in GitLab Groups, GitHub uses GitHub Organisations, for example, the Met Office GitHub Organisation.

GitLab and Github both have educational programs:

- https://about.gitlab.com/solutions/education/
- https://docs.github.com/en/education/

These services allow an individual to apply to join the educational program (GitLab Education, GitHub Education), to access features which usually incur a charge.

We do not have any experience with the GitLab offering, but there are several members of the School of Earth and Environment in Leeds who have applied successfully for the GitHub education program. The main benefit is that this allows the individual to upgrade an organisation to the *Team* level, which adds some additional features, mainly to the functionality of private repositories.

1.4 Hosting Considerations

As there is not a huge number of repositories currently located within the COMET LiCSAR GitLab group, and these would be fairly simple to migrate to GitHub, I believe it would be worth seriously consider migrating to GitHub.

1.4.1 Familiarity

As GitHub is the most popular Git hosting service, this the service which is most likely to be familiar to new members of the group, so there is a lower barrier to entry.

Git training (for example using the Software Carpentry Git Lesson) may also include specific sections of working with GitHub.

GitLab works in a reasonably similar way to GitHub, and it is even possible to log in to GitLab with a GitHub account, but the requirement to use an 'unfamiliar' system can often prove a significant barrier to engagement.

1.4.2 Working With Existing Projects

Some projects of significant interest to the COMET LiCSAR group, such as LiCSBAS and ISCE are hosted on GitHub.

Working on GitHub would make it easier to work with the code hosted in these repositories.

For example, there is a fork of the LiCSBAS repository hosted in the COMET LiCSAR GitLab group. If this was hosted on GitLab, it would be possible to track the differences between this fork and the original repository via the GitHub web interface, as well as easily contribute changes back to the upstream source.

1.4.3 Integration With Third Party Tools

Many third party tools which integrate with Git repositories will integrate with GitLab repositories just as easily as as GitHub repositories, such as Binder and Read The Docs.

One tool which could be of significant interest to the group, and integrates easily with GitHub is Zenodo, which allows creation a copy of the content of a repository at a particular release, with a DOI, making the code easily citable.

There is some more information at the GitHub Zenodo Documentation, and a simple example of a repository which has been archived with Zenodo can be found at:

> https://doi.org/10.5281/zenodo.5997119

Each time a new release is created in a repository which is linked to Zenodo, a new DOI is created, where there is also an overarching DOI which always links to the most recent release.

1.4.4 Restricted GitLab Functionality Without Verification

Possibly the largest issue with GitLab is the inability to use any of the CI / CD (continuous integration / continuous deployment) features without first verifying your identity by providing a credit card.

This means it is not possible to use features such as the Pages (GitLab Pages, GitHub Pages) feature to easily host web content, which can be extremely useful.

It also means it is not possible to enable functionality such as automated testing of code, when changes are pushed to a repository.

The Pages feature on GitHub is much more simple to use and can be enabled with a couple of clicks in the repository settings page. Enabling the Pages feature is more complicated on GitLab, which may also be off putting to individuals who want a simple way to host web content.

1.5 Future Use Of Hosting Services

There are opportunities for increased use of features offered by services such as GitLab and GitHub which could benefit the COMET LiCSAR code and its users.

This may include automated documentation building and deployment, or automated testing of code when changes are pushed to a Git repository.

Some of these things may have an initial time cost, for example creating tests for the software, setting up the tasks to run the tests, and so on, but he benefits may be significant in the long run.

It may also be worth considering how Git tags, software versioning and releases might be used effectively by the COMET LiCSAR project. For example, should there be a version of the software which directly corresponds to a version of the data set?

Any additional software used as part of the processing, and scripts used to build these tools could also be stored in Git repositories, and this may allow easier duplication of the processing environment of different systems.

CHAPTER

TWO

DOCUMENTATION

Currently, most COMET LiCSAR documentation is stored in a GitLab wiki at:

> https://gitlab.com/comet_licsar/licsar_documentation/-/wikis/home.

This wiki contains quite a lot of useful information, and the GitLab wiki renders the content in an easily readable format.

The markdown format used by the wiki makes creating documentation fairly quick and simple.

There is already interest in the group to further improve the documentation of the COMET LiCSAR activities and interest in the benefits which may be gained by making use of Sphinx and Read The Docs.

2.1 Sphinx And Read The Docs

Read The Docs is a documentation hosting platform which generates documentation written with the Sphinx documentation generator

A set of documentation hosted using the Read The Docs service is know as a *Project*.

A Read The Docs account can easily be linked with a repository hosted on GitHub or GitLab, which allows the documentation to automatically be rebuilt and updated when changes are made.

2.2 Sphinx

Sphinx is a documentation generator, which is able to generate formatted documentation in a range of formats (HTML, PDF, etc.) from plain text source files.

Sphinx was originally created to produce the Python documentation, and has become a widely used tool for generating documentation.

2.2.1 Examples

Some example of Sphinx generated documentation

- Matplotlib Documentation
- QGIS Documentation
- NCAS UM Training

Many more listed at the Sphinx Examples

2.2.2 Getting Started

Sphinx documentation can be created on a local machine using the `sphinx` Python library.

This library can be installed in an anaconda environment using the `conda` command:

```
conda install sphinx
```

The library can also be installed using `pip`:

```
pip install sphinx
```

The Read The Docs theme is also available for install using either `conda` or `pip` and can be useful for generating HTML documentation:

```
conda install sphinx_rtd_theme
```

or:

```
pip install sphinx_rtd_theme
```

A skeleton Sphinx project can be created with the command:

```
sphinx-quickstart
```

This will prompt for some information, such as the name of the project, and create the required files to get started, most notably the files:

- `source/conf.py`
- `source/index.rst`
- `Makefile`

The first is the configuration file where various project parameters are set, such as the theme for HTML documentation:

```
html_theme = 'sphinx_rtd_theme'
```

The second is a sample index file for the documentation, and the third is the `Makefile` which can be used to build the documentation, for example to build HTML output:

```
make html
```

2.3 RST Files

By default, Sphinx uses the *rst* (reStructuredText) format for generating documentation.

The rst format was developed for writing documentation. There are numerous markup options available such as using asterisks to emphasise text:

```
*emphasised text*
```

will be rendered as:

emphasised text

The Sphinx web pages include a useful rst primer:

- reStructuredText Primer

The rst web pages include detailed information regarding markup specification and available directives:

- RST Markup Specification
- RST Directives

2.4 Autodoc

The *autodoc* feature of Sphinx can be used to automatically generate documentation from docstrings in Python code.

To enable this feature the autodoc extension first needs to be enabled in the `conf.py` file:

```
extensions = ['sphinx.ext.autodoc']
```

Sphinx also needs to be able to import the Python code from which the documentation should be generated, so it will likely also be necessary to make sure the directory containing the Python code is in the Python path.

This can be done in `conf.py` file, and the default file contains a (commented) example:

```
# If extensions (or modules to document with autodoc) are in another directory,
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown here.
#
# import os
# import sys
# sys.path.insert(0, os.path.abspath('.'))
```

In the above example, the current directory (`.`), relative to the documentation root is added to the path.

2.4.1 Documenting The Python Code

Python code documentation can be included in docstrings, for example this function includes a docstring describing it can be used:

```
def function_a(arg_a, arg_b):
    """
    This is function a, which adds two values

    :param arg_a: first argument is a ``float``
    :param arg_b: second argument in an ``int``
    :return: arg_a + arg_b

    Example usage::

      >>> from python_library import function_a
      >>> function_a(2, 3)
      5
    """
    return arg_a + arg_b
```

This will produce documentation which renders as:

`python_library.function_a(arg_a, arg_b)`

This is function a, which adds two values

Parameters

- **arg_a** – first argument is a `float`
- **arg_b** – second argument in an `int`

Returns arg_a + arg_b

Example usage:

```
>>> from python_library import function_a
>>> function_a(2, 3)
5
```

To include autogenerated documentation, there are various methods available (see the Sphinx Autodoc Documentation), for example, to include documentation for all members of the Python library `python_library`:

```
.. automodule:: python_library
    :members:
```

2.5 Building The Documentation

The documentation can be built by running the `make` command in the documentation source directory.

By default, `make help` will be run, which will display information about how to build various targets:

```
$ make
Sphinx v4.4.0
Please use `make target' where target is one of
  html        to make standalone HTML files
  dirhtml     to make HTML files named index.html in directories
  singlehtml  to make a single large HTML file
...
```

For example, to build HTML documentation, run:

```
make html
```

This will produce create HTML files in the `build` directory:

```
$ make html
Running Sphinx v4.4.0
loading pickled environment... done
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 1 source files that are out of date
...
build succeeded.

The HTML pages are in build/html.
```

2.6 Hosting HTML Documentation On The Web

Once HTML documentation has been generated it can be hosted on the web in various ways.

The content of the HTML directory could be copied to any web server and requires no special server capabilities.

Documentation can also be hosted using popular services such as GitHub, GitLab and Read The Docs.

2.6.1 Read The Docs

The Read The Docs service provides a service for building and hosting Sphinx documentation.

Once an account has been created, this can be linked with a GitHub or GitLab account, which then allows a Read The Docs project to be created from a repository hosted on one of these services.

A Read The Docs project requires a unique name, with the documentation being published at *https://project-name.readthedocs.io/*.

When integrated with a GitHub or GitLab repository, each time the repository is updated, the documentation will be rebuilt and updated.

The Read The Docs service will search the repository for the `conf.py` and use this to build the documentation.

There are various options available via the Read The Docs interface, such the branch in the repository from which the documentation will be built.

Git tags will be recognised, and by default the most recent tag will be labelled as *stable* and the most recent commit labelled as *latest*.

Documentation builds can be triggered for additional versions (from Git tags), and so it is possible to have multiple versions of the documentation available, such as:

- https://project-name.readthedocs.io/en/latest/
- https://project-name.readthedocs.io/en/stable/
- https://project-name.readthedocs.io/en/v0.1/

2.6.2 GitHub

The GitHub Pages allows web content to easily be created and served from a GitHub repository.

The Pages service can be enabled in the repository settings, for example to make the content of:

> https://github.com/username/repo-name

available at:

> https://username.github.io/repo-name.

The simplest way to host HTML documentation with GitHub pages would be to build the files on a local machine, and then commit and push to a GitHub repository.

A more automated method is possible using GitHub workflows. If a repository contains the Sphinx source files, a workflow can be created to automatically build and publish the documentation when the repository is updated.

The source of this documentation is stored on GitHub at:

> https://github.com/cemacrr/sphinx-rtd

This repository contains a `.github/workflows` directory, where the GitHub pages.yml file contains the instructions for building the documentation, which includes building the documentation with Python and then pushing the built HTML files to the `gh-pages` branch, which is then available at:

> https://cemacrr.github.io/sphinx-rtd/

2.6.3 GitLab

GitLab also offers a GitLab Pages service, very similar to the GitHub service.

The GitLab service is not quite so simple to use, and requires creating a `.gitlab-ci.yml` file, which works in a similar way to the GitHub workflows, containing instructions on how to build the web content.

At present, it seems it is not currently possible to to use the GitLab CI service without first registering a credit card with GitLab …

Similar to GitHub, it would be possible to either build the HTML files locally and the commit and push to GitLab, or to have the HTML documentation built automatically when changes are made to the repository. An example of how to automate the Sphinx build with GitLab can be found at the GitLab Pages Sphinx documentation.

CHAPTER

THREE

THIRD PARTY SOFTWARE

On the JASMIN systems third party software used for the LiCSAR processing is stored within:

```
/gws/smf/j04/nceo_geohazards
```

This is SSD storage (JASMIN Storage Types), which is more suitable for storing software than the slower larger volume storage areas, though the space is much more limited.

At present this area is 100G in size:

```
Size  Used Avail Use% Mounted on
100G   55G   46G  55% /gws/smf/j04/nceo_geohazards
```

Software includes GAMMA, LiCSBAS, SNAPHU, an Anaconda environment and various other tools.

There is some use of Environment Modules for loading and unloading of software, and various versions of some packages are available.

3.1 CEMAC ARC Software

CEMAC manage various software installations on the ARC systems in Leeds, and could be a useful example to show some ideas which might help with managing the LiCSAR software.

The files are all within:

```
/nobackup/cemac/software
```

The software is built via build scripts within:

```
/nobackup/cemac/software/build
```

This is also where the required installation sources are stored.

The software is installed within the directories:

```
/nobackup/cemac/software/apps
/nobackup/cemac/software/compilers
/nobackup/cemac/software/libraries
```

This directory structure was set up to match the structure used by the system software on the ARC systems, and may not be required for the LiCSAR software, where a single installation directory may be more suitable.

The Environment Modules files are stored within:

```
/nobackup/cemac/software/modulefiles
```

The build scripts and module files are all stored in a GitHub repository at:

https://github.com/cemac/arc/

An individual can add to their login files (~/.bashrc):

```
if [ -r /nobackup/cemac/cemac.sh ] ; then
  . /nobackup/cemac/cemac.sh
fi
```

The software is then available for use via the `module` command.

3.2 JASMIN Considerations

Unlike the ARC systems at Leeds, where all machines related to the system (e.g. `arc4`) are all more or less the same and run the same version of the operating system, the systems available to users at JASMIN differ.

In the past, the machines have been mixture of different versions of the operating system and this could happen again in the future. It may therefore be necessary to include the operating system version in installation directory / module loading paths, for example, if compiling GAMMA from source on a CentOS / RedHat 7 machine, it may be sensible to install to a path similar to:

```
/gws/smf/j04/nceo_geohazards/software/apps/el7/gamma/20220101
```

If JASMIN machines were upgraded machines to a newer version of the operating system, or a different operating system altogether, the software could also be built for that operating system, and the process which sets up the `MODULEPATH` for a user could automatically adjust settings based on the machine in use.

Another thing to be aware of at JASMIN is that some machines don't have access to the network storage areas (such as the login nodes) and may have to be treated differently. Again, this could be handled by the process which sets up the Environment Modules variables, by checking if things are readable or checking the system host name.

3.3 Build Scripts

Keeping track of the installation process for each version of each piece of software can be incredibly valuable, not only for keeping a useful record of how the software was built and saving time when installing an updated version of a piece of software, it may also make it easier to deploy the software on a new system in the future if required.

Storing the source files with the build scripts helps to keep all required files in the same place, and can also be very useful if rebuilding software is required in the future, as older installation / source files can sometimes become unavailable.

The build process for some software may be as simple as extracting some files and moving them in to place, or it may be a more complex process of building from source and building various dependencies, but it is usually possible to script the process in some way.

3.4 Installation Directories

Ideally, once a particular version of a piece of software has been installed, it should be made read only, never changed, and kept indefinitely.

If a particular version of a piece of software needed to be rebuilt or changed for any reason, a new *build version* could be created.

For example, if GAMMA/20220101 was already installed, but a rebuild of this version was required, for example to apply a patch, the updated version could be built and installed in separate directories, so you could end up with something like:

```
/gws/smf/j04/nceo_geohazards/software/apps/el7/gamma/20220101/1
/gws/smf/j04/nceo_geohazards/software/apps/el7/gamma/20220101/2
```

Where `1` and `2` are the build versions.

There are different ways in which loading software via Environment Modules could be handled, and it would be possible for a user to `module load gamma` and always get the latest version, while also being able to load previous versions by specifying the version of the software.

As the space in the software area is limited, older versions of different pieces of software could be moved to the slower / larger volume storage and replaced with symlinks, so the older versions are still accessible if and when required.

3.5 Environment Modules

Keeping Environment Module files tidy and easy to understand and use isn't always simple, taking a look at the modules provided by default on the JASMIN systems will show how things can become messy and difficult to use.

For any software which has multiple versions, it is possible to set the default version by placing a `.version` file in the directory.

For example, a directory containing module files for GAMMA may contain:

```
20210101
20220101
.version
```

Where the `.version` file contains:

```
#%Module1.0
set ModulesVersion "20210101"
```

So when `module load gamma` is run, the `20210101` version is loaded by default.

It is also possible to create Environment Modules named `old`, `testing`, or similar, and when these modules are loaded, the `MODULEPATH` variable is expanded to include older or testing versions of software. This can help keep the default list of software tidy, while allowing access to additional versions of software.

3.6 Anaconda Considerations

Creating a list of installed packages in an Anaconda environment is quite straight forward, and is a useful way of keeping a record of installed packages.

In theory, it should be possible to use this list to create an identical environment at a later date, but in practice this is rarely simple and creation of a new environment fails.

When creating an environment, it is often best to specify names of required packages, create the environment and then test everything works as required.

The Mamba tool is a “faster” version of the `conda` command, and can greatly reduce the time required to set up a new environment.

Creation of a new Anaconda installation may be similar to:

```
wget -O ${MINICONDA_FILE} \
  "https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh"
chmod 755 ${MINICONDA_FILE}
./${MINICONDA_FILE} -b -p ${INSTALL_DIR}
\rm ${MINICONDA_FILE}
. ${INSTALL_DIR}/etc/profile.d/conda.sh
conda update -y -n base -c defaults conda
conda install -y -n base -c conda-forge mamba
mamba install matplotlib notebook h5py gdal==2.1.0
```

In this example, `gdal` version `2.1.0` is explicitly required, whereas other requested packages should be the most recent available that don’t conflict with any other requested packages.

3.7 Linking Software Versions With Data Versions

At present, I don’t believe the LiCSAR outputs have any kind of version information.

One step towards this may be creating versions of the LiCSAR software tools.

For example, the LiCSAR tools version 1.0.0 may include:

```
condalics/20220101
gamma/20220101
licsbas/1.5.11
```

These packages could be loaded by creating a `LiCSAR` (or similarly named) Environment Modules file.

It could also be possible that in the future the version of the LiCSAR tools is directly linked to the version of the output data, if desired. This could mean that version `x` of the LiCSAR outputs were created with version `x` of the LiCSAR tools.

A good, and hopefully simple, first step would be to record the versions of the software used to create data when generating new LiCSAR output files, for example in the output directory for a processed pair of interferograms, some kind of meta data file in plain text, JSON, both or something else, could be created to include software details, along with any other required metai data.