# CASAL2 Contributors Manual

C. Marsh and S. Rasmussen

## Contents

## 1. Introduction

This is a comprehensive manual for users who wish to develop and contribute to CASAL2. CASAL2 is open source and the development team encourage users to consider enhancing this program through contributing to the source code. The source code is hosted on github and can be found at https://github.com/NIWAFisheriesModelling/CASAL2. For more information you can contact the development team at casal2@niwa.co.nz. This manual is a step by step guide for users who wish to setup the build system and contribute to the source code. It begins by showing how to get a copy of the repository using git commands through to being able to compile CASAL2 and finally adding changes to the master (main) repository. Once changes have been added to the master repository they will be incorporated into the latest compiled version of CASAL2 which will be available for the public to download and use.

To maintain integrity of the code base and ensure confidence in users that the available version is reliable and accurate, the development team has created this manual for contributors. For the reasons above the development team will only accept changes to the repository that abide to the guidelines set out in this manual. This process and manual is intended to be pain free to encourage your contributions.

At the beginning of each section there will be a list of points that the section will go into at great detail. If you have experience with some aspects you can skip but if you run into trouble you can always come back to check this guide. This manual covers basics such as setting up a github profile, to using github and all the way to compiling and modifying C++ code.

## 2. Creating a forked local repository

This section will cover the following points

1. Create a github profile
2. Download git software
3. Fork the master repository

### 2.1. Git Profile

The first step you will have to do is create a profile on github (if you do not already have one). Github is free and easy to setup. Go to `https://github.com` to set up a profile if you do not already have one. Once you have set up a github account you need download git software that you use to "communicate" to repositories.

### 2.2. Git Software

As mentioned earlier the source code is hosted on github so before you can get a copy of the repository you will need to install a program that can "communicate" with github. You will need to acquire Command line GIT from `https://git-scm.com/downloads` this is because it is used to build a Version.h header file so CASAL2 but this will be discussed earlier. I use tortoisegit `https://tortoisegit.org/download` on windows to pull, push and commit changes to repositories. However there are many programs to use and all have the same functionality.

### 2.3. Forking a repository

The main repository which contains code that is in the publicly available CASAL2 executable is called the 'master' repository. Only the development team have permission to add, delete and change code directly with the 'master' repository. For other contributors there is a very easy way to add, delete and change code. This is through forking the master repository. This is done by going to the CASAL2 github repository found at `https://github.com/NIWAFisheriesModelling/CASAL2` and selecting the fork button in the top right of the page circled in Figure 2.1

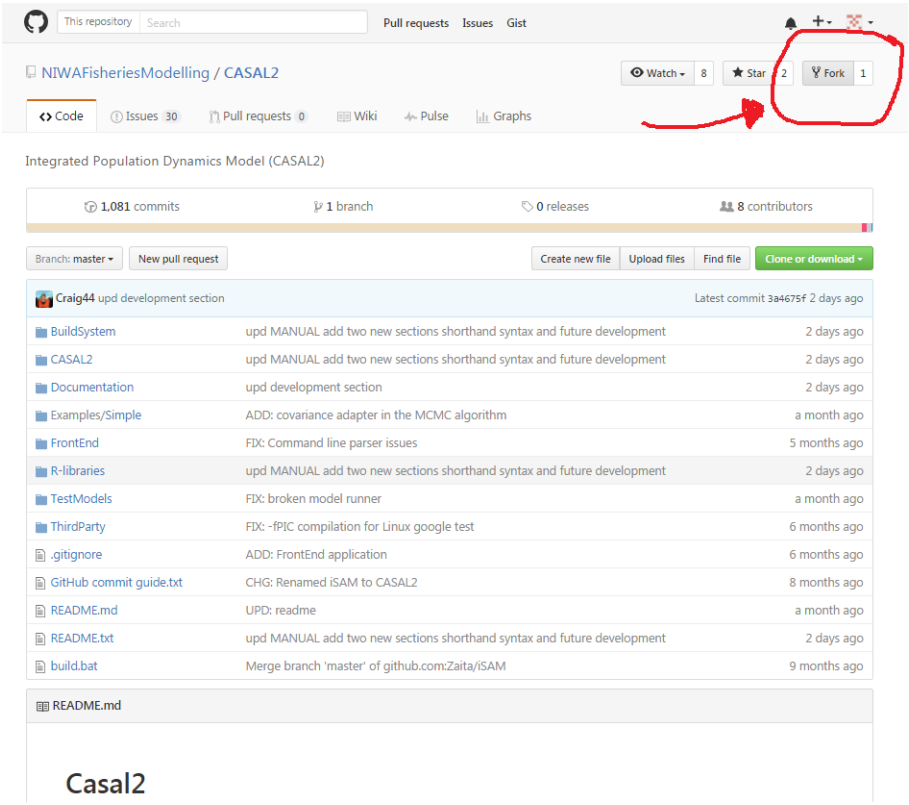


**Figure 2.1: Creating a forked repository**

This will create a copy of the CASAL2 repository under your profile at the point of the fork. To check that you have successfully forked the repository, go to your git profile and you should see a CASAL2 repository under your repositories, shown in Figure 2.2,

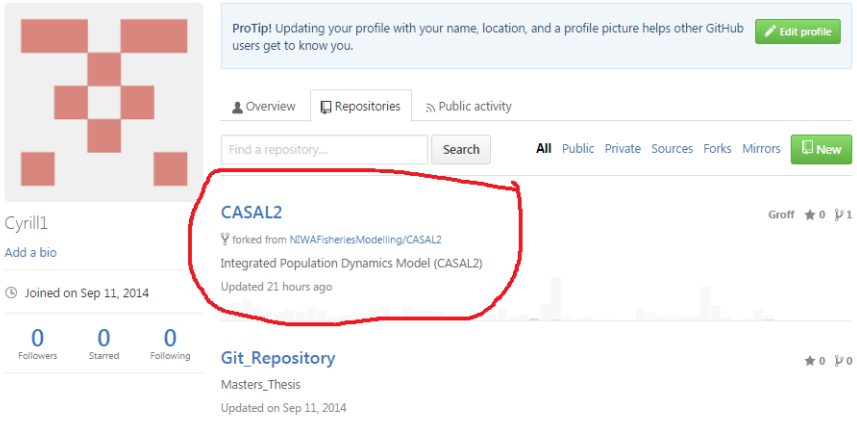


**Figure 2.2: Fork success**

An important point is that the forked repository will not automatically keep up to date with the 'master' repository. So if the development team make changes, you will want to keep your forked repository up to date. This is easily done and is explained in the next section on how to maintain and contribute to a forked repository.

## 3. Maintaining and contributing to a forked repository

This section will cover the following points

1. Check the status of the forked repository compared to the 'master' repository
2. Update forked repository
3. Make changes to your forked repository

### 3.1. Checking the status of the forked repository

To see how your forked repository is compared to the 'master' repository. There is line that is underlined in red shown in Figure 3.1.

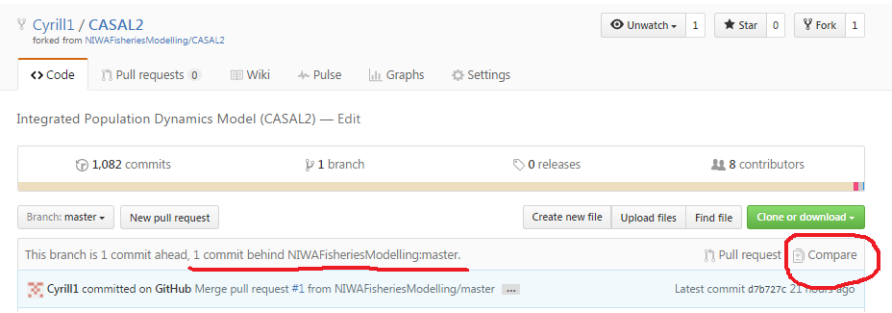


**Figure 3.1: Fork status**

This line tells us if we are up to date or as shown in Figure 3.1 one commit behind. To update this repository click the 'compare' button which is circled in red. This will bring up a page that will tell you all the changes that have occurred to the 'master' repository. There are two situations that can occur when updating a forked repository. The first and easiest is that there is no conflicts and you can merge teh 'master' changes with ease, the second is there are conflicts.

### 3.2. Updating the forked repository

## 4. Setting up CASAL2 BuildSystem

This section will cover, how to setup the build environment on you're local machine to compile CASAL2 on both windows and linux machines. At present the CASAL2 build system supports Microsoft Windows 7+ and Linux (with GCC/G++ 4.9.0+). Apple OSX is currently not supported.

### 4.1. Introduction

The build system is a collection of python scripts that do various tasks. These are located in `CASAL2/BuildSystem/buildtools/classes/`. Each python script has its own set of functionality. The top level of the build system is `CASAL2/BuildSystem/`. In this directory you can run `doBuild.bat help` or `./doBuild.sh help` on the command line to view all of the different build modes. The build system will take one or two parameters depending on what style of build youd like. Its possible to build various stand-alone binaries as well as shared libraries.
For example:
`doBuild debug` - will build a stand alone debug binary
`doBuild library debug` - will build a debug shared library
`doBuild thirdparty` - will build all of the third party libraries
`doBuild thirdparty betadiff` - will rebuild only the betadiff thirdparty library

All outputs from the build system will be placed in subfolders of `CASAL2/BuildSystem/bin/<operatingsystem>/<build_type>`
For example:
`CASAL2/BuildSystem/windows/debug`
`CASAL2/BuildSystem/windows/library_release`
`CASAL2/BuildSystem/windows/thirdparty/`
`CASAL2/BuildSystem/linux/library_release`

### 4.2. Building on Windows

#### 4.2.1. Prerequisite Software

**C++ and Fortran Compiler** Source: tdm-gcc (MingW64) from `http://tdm-gcc.tdragon.net/`. CASAL2 is designed to compile under GCC on all platforms. At present there is no support for Microsoft Visual C++. If you had all of the third-party libraries built under Visual C++ then itd be possible to compile CASAL2İts recommend you get the latest version of TDM-GCC. At the time of writing that was 5.1.0. During installation, you must check to install the GFortran components as well as the C++ components. Some of the third-party libraries required GFortran. **Note**: A common issue is having other GCC compilers in your path for example rtools has a GCC compiler. Make sure this compiler is the first in the path otherwise you may run into errors.

**GIT Version Control** Source: Command line GIT from `https://git-scm.com/downloads`. GIT is used to build a Version.h header file so CASAL2 knows its version internally and this can be used for diagnostic purposes. During installation, you must pick the Windows/DOS command line style of installation. Selecting the Linux/*nix style will cause GIT to mangle the PATH environment variable and the build system will be unable to locate GIT.

**MikTex Latex Processor** Source: Portable version from http://miktex.org/portable. The documentation for CASAL2 is a PDF manual generated from Latex. Much of the Latex has been

written by hand, but the syntax sections of the manual are generated directly from the code. In order to generate the documentation you will need the MikTex latex processor.

**Inno Setup Installer Builder (optional)** Source: Inno Setup 5 from `http://www.jrsoftware.org/isdl.php` If you wish to build a single installer for CASAL2 youll need the Inno Setup 5 application installed on the machine. The installation path must be `C:\ProgramFiles(x86)\InnoSetup5\`

#### 4.2.2. Pre-Build Requirements

Prior to building CASAL2 you will need to ensure you have both G++ and GIT in your path. You can check both of these by typing:
`g++ --version`
`git --version`
Its worth checking to ensure GFortran has been installed with the G++ compiler by typing:
`gfortran --version`
If you wish to build the documentation bibtex will also need to be in the path:
`bibtex -version`

#### 4.2.3. Building CASAL2

The build process is very simple. You can run `doBuild.bat check` to see if your build environment is ready.

1. Get a copy (clone) of the forked code on your local machine, mentioned in Section 2:
2. Navigate to the BuildSystem folder in `CASAL2/BuildSystem`
3. You need to build the third party libraries with:
   - `doBuild thirdparty`
4. You need to build the binary you want to use:
   - `doBuild release`
5. You can build the documentation if you want:
   - `doBuild documentation`

### 4.3. Building on Linux

This guide has been written against a fresh install of Ubuntu 15.10. With Ubuntu we use apt-get to install new packages. Youll need to be familiar with the package manager for your distribution to correctly install the required prerequisite software.

#### 4.3.1. Prerequisite Software

**Compiler G++**
Ubuntu 15.10 comes with G++ 15.10, gfortran is not installed though so we can install it with: `sudo apt-get install gfortran`.
**GIT Version Control**

Git isnt installed by default but we can install it with:
`sudo apt-get install git`
Source: Command line GIT from `https://git-scm.com/downloads` GIT is used to build a Version.h header file so CASAL2 knows its version internally and this can be used for diagnostic purposes. During installation, you must pick the Windows/DOS command line style of installation. Selecting the `Linux/*nix` style will cause GIT to mangle the PATH environment variable and the build system will be unable to locate GIT.
**CMake**
CMake is required to build multiple third-party libraries and the main code base. You can do this with:
`sudo apt-get install cmake`
**Python2 Modules**
There are a couple of Python2 modules that are required to build CASAL2. These can be installed with:
`sudo apt-get install python-dateutil`
You may also need to install **datetime**, re and **distutils**. **Texlive** Latex Processor. No supported latex processors are installed with Ubuntu by default. You can install a suitable latex process with:
```
sudo apt-get install texlive-binaries
sudo apt-get install texlive-latex-base
sudo apt-get install texlive-latex-recommended
sudo apt-get install texlive-latex-extra
```
Alternatively you can do:
`sudo apt-get install texlive-full`
But this will require 3GBs of storage.

### 4.3.2. Building CASAL2

The build process is very simple. You can run `./doBuild.sh check` to see if your build environment is ready.

1. Get a copy (clone) of the forked code on your local machine, mentioned in Section 2:
2. Navigate to the BuildSystem folder in `CASAL2/BuildSystem`
3. You need to build the third party libraries with:
   - `./doBuild.sh thirdparty`
4. You need to build the binary you want to use:
   - `./doBuild.sh release`
5. You can build the documentation if you want:
   - `./doBuild.sh documentation`

## 4.4. Troubleshooting

### 4.4.1. Third-party Libraries

Its possible there will be build errors or issues building the third-party libraries. If you encounter an error then its worth checking the log files. Each third-party build system stores a log of everything its doing. The files will be named

- casal2_unzip.log
- casal2_configure.log
- casal2_make.log
- casal2_build.log
- ...etc

Some of the third-party libraries require very specialised environments for compiling under GCC on Windows. These libraries are packaged with MSYS (MinGW Linux style shell system). The log files for these will be found in `ThirdParty/<libraryname>/msys/1.0/<libraryname>/`
e.g:
`ThirdParty/adolc/msys/1.0/adolc/ADOL-C-2.5.2/casal2_make.log`

### 4.4.2. Main Code Base

If the unmodified code base does not compile the most likely cause is an issue with the third-party libraries not being built. Ensure theyve been built, otherwise engage the CASAL2 development team at `casal2@niwa.co.nz` to help resolve your issue. Running `./doBuild.sh check` can also be helpful to identify any issues.

## 5. CASAL2 build rules

This section will cover, the standards contributors are expected to follow and the builds that must pass on the local machine in order to add a pull request for changes to be accepted into the 'master' repository.

### 5.1. CASAL2 coding practice and style

CASAL2 is written in C++ and follows googles C++ style guide which can be found at `https://google.github.io/styleguide/cppguide.html`. Because we are scientists and not computer programmers we don't expect you to read all that standard, but the main things the development team would like you to follow is the current code style. That is good indentations, sensible variable names and note the _ on the end of variables that are class variables defined in the .h files. **Annotate** your code, for readability we encourage you to put comments around your code. On topping of annotating your code we encourage developers to add logs (print messages) in the source code. You will see in the source code this already. The purpose for logging out is for debugging purposes. By adding them and running CASAL2 in log mode you can find out exactly where CASAL2 is crashing. You can also output equations that would normally be too detailed for end users to be interested in, this is very useful for checking equations are correctly implemented.
There are different levels of logging in CASAL2 listed below.

- LOG_MEDIUM()
- LOG_FINE()
- LOG_FINEST()
- LOG_TRACE()

To run CASAL2 in log mode piping out any LOG_FINEST and coarser logs (LOG_MEDIUM and LOG_FINE) you can use the following command,
`casal2 -r --loglevel finest > MyReports.csl2 2> log.out`
This will output all the logged information to `log.out`.

### 5.2. Unit tests

One of the key focusses in the CASAL2 development is the emphasis on software integrity. It's hugely important to ensure results coming from user models are consistent and correct. As part of this we utilise unit tests to check individual components of the software and run entire models verifying results. CASAL2 uses:

- Google testing framework
- Google mocking framework

It is advised when adding unit tests that they be validated outside of CASAL2Ḟor example in **R** or another program like CASAL2 e.g. CASAL, Stock Synthesis. An example of how to add a unit test for a process is shown in Section 6

### 5.3. Reporting (Optional)

Currently CASAL2 has reports that are **R** compatible, that is that all reports produced by CASAL2 can be read into **R** using the **CASAL2 R** package. If you create a report you most follow the

standard so that the report is **R** compatible.

All reports must start with,
`*label (type)`
and end with,
`*end`

Depending on what type of information you wish to report, will depend on the syntax you need to use. For example

#### {d} (Dataframe)

Report a dataframe

```
*estimates (estimate_value)
values {d}
process[Recrutiment_BOP].r0 process[Recrutiment_ENLD].r0
2e+006 8e+006
*end
```

#### {m} (Matrix)

Report a matrix

```
*covar (covariance_matrix)
Covariance_Matrix {m}
2.29729e+010 -742.276 -70160.5
-110126 -424507 -81300
-36283.4 955920 -52736.2
*end
```

#### {L} (List)

Report a List

```
*weight_one (partition_mean_weight)
year: 1900
ENLD.EN.notag {L}
mean_weights {L}
0.0476604 0.111575 0.199705
end {L}
age_lengths {L}
12.0314 16.2808 20.0135
end {L}
end {L}
*end
```

### 5.4. Update manual

Sections of the user manual are automatically generated from the source code. This is the Syntax section of the user manual. This means contributors need to manually update the sections of the user manual that their changes relate to. This is important if you want the end user to be able to use

you're functionality with ease. It can be very difficult is users see functionality in the syntax section but no reference in text, so please keep this in mind.

## 5.5. Builds to pass before merging changes

Once you have made changes to your local repository you must run the following builds before your changes are considered for merging with the 'master' repository.
build the unittest version see Section 4 for how to build unittest depending on you're system.
`DoBuild test`
run the unittest's check they all pass
`casal2` run this command in the `CASAL2/BuildSystem/bin/<operatingsystem>/test` folder
`DoBuild debug`
run the second phase of unitests which are complete model runs
`DoBuild modelrunner`
this is conditional on having a debug version previously built. Build the archive which contains
`DoBuild archive`

## 6. An example of adding a new process

The following shows a sequence of figures and text of an example, where we are adding a new process called survivorship.

## 7. Merging changes form a forked repository to a master repository

This section will cover, how to merge changes from a forked repository to the 'master' repository. This is under the assumptions that the contributor has followed all the rules laid out in Section 5.