

## Table of Contents

- Introduction
- What is here
- How-to-Use
- License
- Troubleshooting

## Introduction

Welcome to my i-ASK repository where I store all the files that I have worked on. Here you can find my module files, various shell scripts, singularity images, etc.

## What is here

- Scripts
- Modules
- Singularity Definition Files
- Installers

## Scripts

A collection of shell scripts that I use to help users. Most are written for bash, but I will look into supporting users that prefer to use non-bash shells such as tcsh. Scripts are stored in `/scripts`, and the executables are stored in `/scripts/bin`. For detailed use of each script please refer to the how-to-use section of the documentation. The list of available scripts is as follows: `### Bash`

- `collector`
- `gathero`
- `relink_work_scratch`
- `setup_comsol_symlink`
- `setup_conda_symlink`

## Modules

A collection of modules that I have created for users. Written in lua for use with Lmod. Please note that these have been configured for the specific situation of the user (i.e. don't drag and drop module files). Modules are stored in the `/modules` directory. The list of available modules is as follows:

- `Julia`
- `pandoc`
- `scripts`

## Singularity Definition Files

A collection of definition files that I have used to build containers needed by users using Singularity. I prefer to host my images on Sylabs Cloud, but there are many other ways to host singularity images. Generally, I design the containers specific to the individual user's needs, but sometimes I will use base images that I have built myself. For specific information on definition files please refer to the how-to-use section of the documentation. The definition files are stored in `/src/def`. The list of available definition files is as follows:

- Cadabra2
- calc-def
- Deeplearning Toolbox
- HiC-Pro
- LAYNII-def
- Libbi
- NLOpt
- RStudio Base

## Installers

This is a collection of scripts that I send to users to help them locally install software that they need for their research. Generally, they are written in bash, but sometimes I may use python to help with file management. These installers have dependencies, but they are usually packaged within the tar file or downloaded from the internet. The list of available installers is as follows:

- calc
- LAYNII
- netCDF-c-base
- R-4.0.2
- SQLite3

## How-to-Use

- Scripts
  - collector
  - gathero
  - relink\_work\_scratch
  - setup\_comsol\_symlink
  - setup\_conda\_symlink
- Modules
  - Julia
  - pandoc
  - scripts

- Singularity Definition Files
  - Cadabra2
  - calc-def
  - Deeplearning Toolbox
  - HiC-Pro
  - LAYNII-def
  - Libbi
  - NLOpt
  - RStudio Base
- Installers
  - calc
  - LAYNII
  - netCDF-c-base
  - R-4.0.2
  - SQLite3

## Scripts

### collector

A collector is someone who collects. Collector is a simple script that collects info on the user's home directory. Produces a tar archive containing info on the users .bashrc, .bash\_history, .bash\_profile, .bash\_aliases, directory size, etc. Pretty much helpful anytime a user is having issues with anything. With this script the user will need to use it, and then send the responding tech the tar file. Here are the commands you should send to the user:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ collector #=> Creates ${USER}_info.tar.gz
```

It prints out a help message directing the user to use Open Ondemand to download the tar file. Once they send back the file simply unzip it using tar (or 7zip if using Windows) and examine the contents:

```
$ tar -xzf ${USER}_info.tar.gz #=> Creates directory named ${USER}_info
$ less ${USER}_info/${USER}_bashrc.txt
```

All the files will be named after the user so you know who you're looking at. Now go find what's wrong!

### Update 8/4/2020

As of this update **collector** can be invoked as **collector**. This update was made because many users are used to just spelling "collector." The invocation is the same as **collector**, but here the commands below for reference:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ collector #=> Creates ${USER}_info.tar.gz
```

## gathero

Gathering is for those who want information, and that's exactly what gathero does! Too often are we as techs left wondering, "Why won't this job start? Why was this job suspended?" First we run checkjob. Then maybe we use account\_quota\_check or mam-list-accounts. Well now we no longer have to do this. Gathero does this for us! Simply get the job id from the user and use the following commands:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ gathero ${JOB_ID} #=> Creates ${JOB_ID}_info directory in ${HOME}/scratch
```

Now there is a lot that goes on here, but there are 5 big things that this script does:

1. Creates the file checkjob\_output.txt, which is generated from the output of `checkjob -v ${JOB_ID} --timeout=300`
2. Creates the file user\_info.txt, which is generated from the output of `account_quota_check ${USER}`, `qstat -u ${USER}`, and `mam-list-accounts -u ${USER}`
3. Creates the file allocation\_info.txt, which is generated from the output of `showq -w acct=${ALLOC_ID}` and `mam-list-funds -u ${USER} -h`
4. Creates the file all\_info.txt, which is generated by concatenating checkjob\_output.txt, user\_info.txt, and allocation\_info.txt together (for those of us who are busy)
5. Creates the file `${JOB_ID}_info.zip`, which can be downloaded and mailed to the inquisitive user

Simply read through the files, find what's wrong with the job, and mail of the zip file so the user knows what you're talking about:

```
$ cd ${HOME}/scratch/${JOB_ID}_info
$ less all_info.txt
```

## relink\_work\_scratch

You ever have one of those moments where you can't tell why a user's home directory is full? Well, I've been working here long enough to discover that some user's unlink their `work` and `scratch` directories and use `mkdir` to create them.

What they don't know is that this will count towards their home directory's storage limit. Generally this is fixable but it requires knowledge of how symlinks work. `relink_work_scratch` takes care of this for you. Simply have the user execute the following commands:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ relink_work_scratch
```

This will rebuild the symlinks `work` and `scratch`, plus create the directories `not_real_work` within `work` and `not_real_scratch` within `scratch`. Really want to hammer home that `work` and `scratch` are actually symlinks.

### **setup\_comsol\_symlink**

User's like to use comsol. Comsol likes to write out to its cache in the home directory. What do you get? "Error: Disk Quota Exceeded." Generally, I would have users either delete the cache or create a symlink, but I found that users will mess it up quite often. Therefore, I wrote this shell script. Have the user use the following commands to create their symlink:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ setup_comsol_symlink
```

Disk Quota Errors beware!

### **setup\_conda\_symlink**

Same deal as comsol. Users love it, and conda loves writing out to its cache in the home directory. Let users create conda environments to their hearts content by having them use the following commands:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
$ setup_conda_symlink
```

## Modules

### Julia

Since Julia is being consistently updated by its developers I like to have the most recent version of it installed on the cluster. I do this so that users can utilize the bleeding-edge of Julia, and get access to the newest features. Since there are many different versions of Julia, I have the ones I have documented installing below:

- Julia 1.4.2
- Julia 1.5.0

#### Julia 1.4.2

To load Julia 1.4.2, use the following commands:

```
$ module use /gpfs/group/dml129/default/nucci2/sw/modules
$ module load julia/1.4.2
```

#### Julia 1.5.0

To load Julia 1.5.0, use the following commands:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load julia/1.5.0
```

### pandoc

Pandoc is great because it allows you to convert to many different markup languages. It is even a dependency for some popular packages, such as Rmarkdown. I now bring users the power to use this great tool with the following commands:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load pandoc
```

To check that the module works, use the following command:

```
$ pandoc --version
```

You should see the following printed out to your command line:

```
pandoc 2.10
Compiled with pandoc-types 1.21, texmath 0.12.0.2, skylighting 0.8.5
Default user data directory: /storage/home/${USER}/.local/share/pandoc or /storage/home/${USER}/.local/share/pandoc
Copyright (C) 2006-2020 John MacFarlane
Web: https://pandoc.org
This is free software; see the source for copying conditions.
```

There is no warranty, not even for merchantability or fitness for a particular purpose.

To get more detailed information on pandoc you can visit their website here: <https://pandoc.org/MANUAL.html> #

### **scripts**

This module is how to access the scripts I have written. Simply use the following commands to load the module:

```
$ module use /gpfs/group/dml129/default/sw/modules
$ module load scripts
```

To see a list of available scripts you can use one of the following commands:

```
$ module help scripts
```

or

```
$ scriptslist
```

If you are interested in writing your own shell scripts you can refer to this guide here: [https://www.tutorialspoint.com/unix/shell\\_scripting.htm](https://www.tutorialspoint.com/unix/shell_scripting.htm)

## **Singularity Definition Files**

### **Cadabra2**

The Cadabra software is a field-theory motivated approach to computer algebra. Here it is installed inside a singularity container built upon Debian 9. I just host the definition file here but build the image is pretty easy.

Cadabra2 is available as a module on the cluster and can be loaded using the following commands:

```
$ module use /gpfs/group/dml129/default/nucci2/sw/modules
$ module load cadabra/2.2.9
```

To launch the Cadabra2 notebook you can use the following command:

```
$ cadabra-gtk
```

To launch the Cadabra2 CLI you can use the following command:

```
$ cadabra
```

### **calc-def**

Calc is an interactive calculator which provides for easy large numeric calculations, but which also can be easily programmed for difficult or long calculations. It can accept a command line argument, in which case it executes that single command and exits. Otherwise, it enters interactive mode. In this mode, it accepts commands one at a time, processes them, and displays the answers. This is an interesting piece of software that I installed for a user. It is virtually the same as the Unix utility `bc`, but it has more features. However, unfortunately, the maintainers did not include support for the `PREFIX` build option in the Makefile. Therefore, I had to install it inside a singularity container.

The container is primarily used for the calc installer. It is downloaded or copied in the script to the appropriate location, and is then setup as a loadable module.

### **Deeplearning Toolbox**

The Deeplearning Toolbox is just a collection of python programs that are used for deep learning. You can find the likes of tensorflow, keras, and OpenCV2 installed inside of it. The definition file is just hosted [here](#).

The Deeplearning Toolbox is available as a module on the cluster, and it can be loaded using the following commands:

```
$ module use /gpfs/group/dml129/default/nucci2/sw/modules
$ module load python-deeplearning-toolbox/1.1
```

To get a full list of what is installed inside the container simply use the following command:

```
$ module help python-deeplearning-toolbox/1.1
```

### **HiC-Pro**

HiC-Pro is an optimized and flexible pipeline for Hi-C data processing. Luckily, this is an easy program to install inside a container. The definition file is just hosted [here](#).

HiC-Pro is available as a module and can be loaded using the following commands:

```
$ module use /gpfs/group/dml129/default/nucci2/sw/modules
$ module load hicpro/2.11.4
```

Simply call the HiC-Pro executable to use it:



```
$ HiC-Pro <options> <arguments>
```

### **LAYNII-def**

This is a package of standalone layer functional magnetic resonance imaging (layer-fMRI) C++ programs that depends only on a C++ compiler. The purpose of this package is to provide layer-analysis software that are not (yet) included in the other major MRI analysis software. This software was built into a container since it depends on a newer version of glibc. The definition file is just hosted in this repository.

### **Libbi**

LibBi is used for state-space modelling and Bayesian inference on modern computer hardware, including multi-core CPUs, many-core GPUs (graphics processing units) and distributed-memory clusters. This is an image that I built for a user. Unfortunately it is not available as a module.

### **NLopt**

NLopt is a free/open-source library for nonlinear optimization, providing a common interface for a number of different free optimization routines available online as well as original implementations of various other algorithms. This is another container that I built for a user. Unfortunately it is also not available to be used as a module.

### **RStudio Base**

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. I use this image as a bootstrap for other images that require R and RStudio.

## Installers

### calc

I built this installer to help users locally install calc. This script does nothing fancy, but it does download my calc container stored in the cloud and move the module file to its respective directory. This file can be executed anywhere as all it depends on is the `${HOME}` bash environment variable. The one major improvement however is that this installer can use a local copy of the calc container, instead of always having to download it from the cloud. Just package the container inside of the tar archive!

You can use the following instructions to construct the installer:

```
$ tar -czvf calc_installer.tar.gz calc #=> Clone installers directory
```

Then, send the tar file to the user and have them execute the following commands:

```
$ tar -xzf calc_installer.tar.gz
$ cd calc
$ chmod +x INSTALL
$ ./INSTALL
```

The user should then be able to access the module using the following commands:

```
$ module use ${HOME}/work/sw/modules
$ module load calc/2.12.7.1
```

Now the user should have their own LAYNII module!

### LAYNII

I built this installer to help users locally install LAYNII. The script does nothing fancy, but it does download my LAYNII container stored in the cloud and move the module file to its respective directory. This file can be executed anywhere as all it depends on is the `${HOME}` bash environment variable.

You can use the following instructions to construct the installer:

```
$ tar -czvf LAYNII_installer.tar.gz LAYNII #=> Clone installers directory
```

Then, send the tar file to the user and have them execute the following commands:

```
$ tar -xzf LAYNII_installer.tar.gz
$ cd LAYNII
$ chmod +x INSTALL
$ ./INSTALL
```

The user should then be able to access the module using the following commands:

```
$ module use ${HOME}/work/sw/modules
$ module load laynii/1.5.6
```

Now the user should have their own LAYNII module!

### **netCDF-c-base**

I built this installer to help users and groups install the most basic version of netCDF4 you can. There are a bajillion ways to configure netCDF, and this installer is as simple as you can get. There is no built-in support for Fortran or Java, just beautiful, pure C. This installation is perfect for anyone trying to install the R package ncdf4. The installer will also automatically generate a module file. Not in the game of modifying users `.bashrc` files. To create the installer, use the following command:

```
$ tar -czvf netcdf-c-base_installer.tar.gz netcdf-c-base #=> Clone installers directory
```

Please note that since I don't trust the internet (RHEL 6 repositories won't be around forever!) I put the appropriate dependencies in the `src` directory. Unfortunately, this does make the installer almost too large to send through email, so it might be better to host it in a shared folder (or FTP server!). However you get the installer out to the user, have them execute the following commands to install the basic netCDF:

```
$ tar -xzvf netcdf-c-base_installer.tar.gz
$ cd netcdf-c-base
$ chmod +x INSTALL
$ ./INSTALL /path/to/desired/dir
```

Once the installer is finished, the user will be able to load the module using the following commands:

```
$ module use /path/to/desired/dir/modules
$ module load netcdf/base-4.7.4
```

Now the user can compile their programs that only require the base netCDF!

### **R-4.0.2**

I built this installer to help users and groups install the newest version of R. R has a lot of dependencies, and this installer should take care of them. This installer is a little bit more involved since R is being installed in different locations. The

neat thing that this installer does it automatically generate a module file, and it automatically detects if you have a prior R installation.

You can construct the installer using the following command:

```
$ tar -czvf R-4.0.2_installer.tar.gz R-4.0.2 #=> Clone installers directory
```

Then, send the tar file to the user and have them execute the following commands:

```
$ tar -xzf R-4.0.2_installer.tar.gz
$ cd R-4.0.2
$ chmod +x INSTALL
$ ./INSTALL /path/to/desired/dir
```

This script will create the `modules` directory under the desired dir. To load the R module, use the following commands:

```
$ module use /path/to/desired/dir/modules
$ module load r/4.0.2
```

Now users can have the newest version of R!

### Update 8/9/2020

The R installer now relies on `tar.gz` files stored in the `src` directory. I did this because the server that I would download the `.rpm` file from was very unweildy and not very dependable. Therefore, I modified the script to use files already stored on the cluster.

### SQLite3

Everyone loves databases, but sadly, databases don't love ACI. Since we don't have the allocatable resources to run databases, users can't utilize technologies like MySQL and MariaDB. Oh the humanity! Whatever will we do? Well, I know. Use SQLite3! SQLite3 is my favorite DBMS because it is so versatile. Because I love it so much, I built an installer that users can use to install SQLite3 themselves!

To create the installer, use the following commands:

```
$ tar -czvf sqlite3_installer.tar.gz R-4.0.2 #=> Clone installers directory
```

Then, send a copy to the user and have them execute the following commands:

```
$ tar -xzf sqlite3_installer.tar.gz
$ cd sqlite3
$ chmod +x INSTALL
$ ./INSTALL /path/to/desired/dir
```

This script will create the `modules` directory under the desired dir. To load the SQLite3 module, use the following commands:

```
$ module use /path/to/desired/dir/modules
$ module load sqlite/3.33
```

Now users can execute structured queries to their hearts content!

## License

This repository is licensed under the GNU General Public License v3.0. For more information on what this license entails, please feel free to visit <https://www.gnu.org/licenses/gpl-3.0.en.html>

## Troubleshooting

If you run into any issues regarding the use of anything in this repository then please contact Jason at either [jcn23@psu.edu](mailto:jcn23@psu.edu) or at the ICDS i-ASK center ([iask@ics.psu.edu](mailto:iask@ics.psu.edu)). If you do run into an issue, please be as descriptive as possible.