bashSTScI Documentation

Release 0.1.0

Sara Ogaz, Joe Hunkler

July 14, 2016

Contents

Anaconda is a distribution platform that uses the Python programming language. The Anaconda Python distribution uses the Conda package manager to keep track of a user's Python libraries and secondary software packages in a very automated way. The user is able to interact at the top level with the Conda package manager, and all installs and environment setup is controlled by Conda. Anaconda has been widely adopted by the Python community, and provides easy access to many of the most popular scientific Python packages (Numpy, Astropy, SciPy, etc).

For these reasons, among others, SSB has chosen to change the way we will be distributing STScI software packages and tools. This new STScI Conda package is called Astroconda, and in order to install these packages the user must install the Anaconda Python distribution platform. Here you will find a guide for how to install Anaconda and the Astroconda package on STScI machines. We will go over the following steps:

1. Set up your `bash` shell (not always straightforward on STScI machines)
2. Install Anaconda
3. Add the STScI Conda channel
4. Install the STScI Conda software
5. Activate/test the new installation

1 Note about bash

Anaconda requires the `bash` shell. If you have no shell preference, we highly recommend switching over to `bash` as your default shell. The institute sets up your default shell as `tcsh` (this will most likely change in the future). This means that when you start up a terminal you are automatically put into a `tcsh` terminal.

You can temporarily switch to the `bash` shell by typing in `bash` or `bash -l`. We will come back to this later in the guide. While it is an option to temporarily switch to the `bash` shell whenever you want to use Anaconda, this option is much more inefficient and error prone, so once again, we recommend switching over to `bash` as your default unless you have extenuating circumstances. You should never run `anaconda` from a `tcsh` terminal. **Anaconda is designed to run in the ‘‘bash‘‘ shell only.** You will find directions for both options in this guide.

2 Note For Linux Users

There is one change you will need to make when following this guide. Anaconda’s default install location is your home directory, but on Linux machines at STScI the home directory has a 10GB quota. To work around this we recommend installing Anaconda into your `/user/username/` folder.

To do this you can use the `-p` option when installing Anaconda. At this time you can provide whatever directory path you would like, but keep in mind the final directory in the path must not exist, as the installer will want to create this folder. For example:

```
sh Anaconda2-4.0.0-Linux-x86_64.sh -p /user/username/anaconda2
```

Here you can change the folder name `anaconda2` to correspond to whichever version you chose to install, i.e. `anaconda3`, `miniconda2`, or `miniconda3`.

Note: If you are an internal STScI user and have any questions about the content of this guide, please contact support@stsci.edu.

2.1 Bash Setup

Step 1: Removing references to Ureka

Remove all references to `Ureka` (`SSB` , `SSBX`, `SSBDEV`) in your shell files. If you have never edited your `bash` setup files `.profile`, or `.bash*`), you can ignore these for now as we are going to delete them all later on, and just focus on your `tcsh` files, these could include: `.cshrc`, `.setenv`, or `.scienv`. You may or may not have some of these files depending on when your machine was set up originally. Accounts created after 2008 may have an additional `.mysetenv` file.

If you not sure what to look for, and haven’t edited your shell files before, look for references to `ssbx`, `ssbrel` or `ssbdev` and remove those lines. You can use the following listed commands to search for `Ureka` dependencies. You may see lines from your `$HOME/.ureka/.default` and `$HOME/.bash_history` files. This is expected and can be ignored.

Linux:

```
find $HOME -maxdepth 1 -type f -name '.*' | xargs -I'{}' grep -n -H -E --color=auto 'ssbrel|ssbx|ssb
```

OSX:

Darwin’s BSD find does not support -maxdepth (users can install gnu findutils if they wish to execute the Linux-style command on OS X, however)

```
find $HOME -type f -name '.*' | xargs -I'{}' grep -n -H -E --color=auto 'ssbrel|ssbx|ssbdev' "{}"
```

Step 2: Backup your current bash files

Warning: Only follow this step if you have not added anything to your ITSD-provided `bash` setup files (`.profile`, or `.bash*`). If your `bash` files are configured already, you can skip steps 2 and 3.

Tar and zip your current bash setup files with the following command:

```
tar cfz envbackup.tar.gz .profile .bash*
```

Step 3: Rebuilding your bash startup files

Now that you have a backup of your `bash` files safely stored, wipe all `bash` startup files currently in your `~home` directory.

```
rm ~/.profile
rm ~/.bash*
```

You should also delete your `$HOME/.pydistutils.cfg` file. This file can interefere with Anaconda's Python install.

```
rm ~/.pydistutils.cfg
```

Now you can open up a new `~/.bash_profile` file. Remember this should be a blank text file, since we just deleted the previous copy if it existed. We will also set up a `~/.bashrc` file. Below is an example of a standard `~/.bash_profile` and `~/.bashrc`.

```
# File - ~/.bash_profile (or .profile)

# Common shell aliases
alias ls='ls -G'
alias ll='ls -l'
alias grep='grep --color=auto'

if [ -f $HOME/.bashrc ]; then
    source $HOME/.bashrc
fi

# EOF
```

```
# File - ~/.bashrc

# Tune your profile... these are example only
# Replace these with desired paths
export PATH="$PATH:$LOCAL_CUSTOM/bin:$PATH"
export MANPATH="$LOCAL_CUSTOM/share/man:$MANPATH"

# EOF
```

Using this line:

```
if [ -f $HOME/.bashrc ]; then
    source $HOME/.bashrc
fi
```

the `~/.bashrc` file will get sourced by `~/.bash_profile`.

Now we can start to port the environment setup information that was in the `tcsh` startup files over to your `bash` files. Most of these commands will either be `setenv` or `alias` commands. **There is a syntax difference between ''tcsh''**

and ''bash''. You can put these kinds of commands into your `.bash_profile` file. Below are some examples of how to translate `tcsh` to `bash` syntax.

tcsh syntax	bash syntax
setenv cdbs /grp/hst/cdbs/	export cdbs="/grp/hst/cdbs/"
setenv PATH $HOME/pybin:${PATH}	export PATH="~/pybin:$PATH"
alias emax 'open -a "Aquamacs"'	alias emax='open -a "Aquamacs"'
setenv EMACS editor	EDITOR=emacs; export EDITOR

Finally, you should now restart your terminal program so that these changes are applied.

Note: Regarding if statements: Many of the statements originally in the `tcsh` files that were nested in `if` statement calls were set up to test if your machine was connected to the STScI network. For example, if you set up an environment variable that links to a directory on `/grp/hst/` and try and access this directory from outside the institute network, it will fail.

For `if` statements that you have written into your `tcsh` files yourself, please see this bash guide for `if` statements in `bash`.

Step 4: Bash as default, or temporary bash sessions

Switching to bash as your default shell

Warning: You may want to wait to execute this step until after you have installed and tested Anaconda.

For Mac

To switch your default shell on Mac machines, you can change your local system by opening a terminal and using the following command, you will need to enter your password when prompted:

```
chsh -s /bin/bash
```

To verify that the change went through, restart your terminal program, and type the following:

```
echo $SHELL
```

This command should return `/bin/bash`. Remember this changes your **local** default only. To change your default on all future systems and builds you should also follow the directions below for Linux machines so that your AD default is changed. But keep in mind this immediately changes your default on all Linux machines.

For Linux

To change the default shell on Linux machines (this includes the Linux servers at STScI) you will need to contact IT to switch your AD account settings. The path to your default shell is controlled by Active Directory (AD), which can only be modified by ITSD.

Using bash from tcsh

If you plan on using `bash` from `tsch`, you can switch into `bash` using

```
bash -l
```

This call will inherit your environment setup from your `tcsh`. This means any environment variables you have set in your `tsch` will get transferred over.

Warning: If you have a call to `ssbx/dev/rel` in one of your `tsch` setup file `Anaconda` will not run properly!

2.2 Installing Anaconda and AstroConda

Note: If you already have `Anaconda` on your machine and are familiar with it, you can skip ahead to Step 3.

Step 1: Install

There are several flavors of Anaconda for download. Anaconda contains the full `anaconda` software suite, while Miniconda contains only Conda and Python. You also have the choice of Python 3 or Python 2.7 as your default Python version. Please note, you can still access Python 3 from the Anaconda Python 2.7 version, and vice versa. This just sets up your default to be one or the other. Further details and installation instructions are available on the Anaconda webpage.

As previously mentioned, Anaconda works with the `bash` shell, so please make sure you are running install commands, etc. in a `bash` environment. To do this before you have switched to `bash` as your default terminal, please use

```
bash -l
```

Step 2: Make sure Anaconda was set up correctly

To test that your `PATH` variable was correctly set for Anaconda after installation, check for the following line in your `~/.bashrc` file. It should be similar to the following:

```
Mac:
export PATH="$HOME/anaconda3/bin:$PATH"

Linux:
export PATH="/user/username/anaconda2/bin"
```

If this line is missing please add it, making sure to use your Anaconda folder name. This varies depending on which version of Anaconda/Miniconda you downloaded. The default install location of Anaconda/Miniconda is in your home directory. So if you've run the Anaconda/Miniconda install with the default settings you should now see a directory similar to this in your home directory: `~/anaconda2` (If you grabbed Anaconda2), `~/miniconda3` (if you grabbed Miniconda3), etc.

Let's talk about Conda environments...

Creating and switching Conda environments is just like switching `Ureka` environments, but *even better*! We'll cover it in a nutshell here, but for full details please see the Anaconda documentation. What so much better about it, you might ask? Well, Conda environments give you complete freedom to:

- Make as many different custom environments as you want
- Use different versions of different packages in each environment
- You decide when to update packages
- Can share your environment setup easily

Conda environments give you all the freedom to pick and choose what versions and which packages are included in that environment, without any of the hassle of having to manually install everything. It also does a great job of keeping all your different environments separate from each other. You can use Python 2.7 and an older version of `Numpy` in one environment, and another where you're using Python 3 and the cutting edge version of your favorite Python libraries. This makes creating a testing environment extremely easy. Want that new `Astropy` function but not sure if that version of `Astropy` will break other parts of your code? Just make a new environment with the new `Astropy` version to test with. You can still keep your working environment just where you left it. Another great benefit is the ability to share an exact copy of your environment with others.

So how do you make a new Conda environment? This can be done in two simple statements. To make a new environment just use the following commands:

Example 1:

This installs a base version of a new environment. Containing just the basic Python libraries. Here we're specifying to use Python 2 (2.7).

```
conda create -n myenvironment python=2
```

Example 2:

Here we're creating a new environment that will be populated with the basic Python libraries and adding in the Python `bokeh` library.

```
conda create -n bokehenv python=2 bokeh
```

Example 3:

Alternatively, I could have omitted `bokeh` as an argument, or any specific packages, and populated my environment manually.

```
conda create -n bokehenv
source activate bokehenv
conda install bokeh
```

Example 4:

It is also possible to install packages into a named environment without the need to activate it first. The `bokeh` environment must already exist for this work

```
conda install -n bokehenv bokeh
source activate bokehenv
```

You'll see these commands again as you walk through the AstroConda installation.

Step 3: Get AstroConda

AstroConda is a package repository that is built to hook into the Anaconda distribution. You can think of Anaconda (using Conda) as your environment manager, and AstroConda as an extra repository of packages and software (called a "channel" in Conda). It contains many of the tools that were built into `Ureka`. For more information on the AstroConda packages please see the AstroConda doc page. You will also want to reference this page if you need to include `iraf` in your AstroConda install.

The next step is to add the `astroconda` channel.

```
conda config --add channels http://ssb.stsci.edu/astroconda
```

Now we will create a new environment that contains AstroConda's `stsci` metapackage

```
conda create -n astroconda stsci
```

and activate this new environment.

```
source activate astroconda
```

Make sure you are installing the `stsci` metapackage into a new environment and not your root Anaconda environment. If this has happened, please see the AstroConda FAQ page for instructions.

Step 3B: Add the Astropy channel

The Astropy Project also has a Anaconda channel which contains all of the `astropy` affiliated packages. While some of these are already included in `astroconda`, if you think you might want some that aren't you can use the following command

```
conda config --add channels astropy
```

and you now have the `astropy` channel included by default.

Step 4: Quick test to see if your setup was succesful

Note: For instructions on how to keep your AstroConda package updated, please see the AstroConda docs.

Note: For a easy to use Conda reference sheet the Anaconda website has a helpful Conda cheat sheet, or you can pick up a hardcopy right outside the IT helpdesk room in Muller 330.

Were going to simulate a fresh new bash shell for testing, so open a new terminal window and use the following commands. You can start from a `tcsh` terminal.

```
/grp/hst/ssb/blackhole/interactive.sh
source ~/.bash_profile
```

If successful, the PS1 variable, responsible for controlling the look and feel of your shell prompt, will be reset to the system default; often `[user@host:  directory]$`. From here we will test Anaconda and your AstroConda install.

```
source activate astroconda
which python
```

`which python` should return `/path/to/astroconda/bin/python`, where `/path/to/` will be the path to your Anaconda installation. If this test returns unexpcted results, you can contact support@stsci.edu for assistance.

2.3 Installing Anaconda On Linux Servers

You will probably want to access Anaconda from your account on the STScI Linux servers. If you are working on a Linux machine, congratulations this is already done! When you set up Anaconda in your `/user/username` directory and set up your `bash` files, this will get propagated to any other Linux servers. If you are on a Mac machine, you can follow the Linux part of this instruction manual in an `ssh` session on any of the STScI Linux servers.