miniconda-introduction.md

References

- Miniconda
- 30 min test drive

Description

The Miniconda installer installs a smaller version of the conda package manager to your home directory as will as well as it's own version of Python. Once Miniconda is installed, you can use the conda command to create additional Python environments which you can customize as you like. You can create additional environments that run "any" version of Python and any other Python programs you install to the virtual environments.

Benefits

- You can easily create, modify, activate and deactivate mulitiple custom Python environments.
- You can create and run virtual Python environments using almost "any" version of Python.
- Each Python environement can be customized as you wish.

Keep in mind

It is easy to accidentally customize the Python environment that the miniconda installer installs for you and you probably do not want to do this.

Create a new, hosted, Python (virtual) environment for each of your project(s). This way your host Python installation, the one that supports the `conda` program, remains 'clean'. This allows you to freely experiment and easily discard or duplicate and or exchange virtual environement contained Python projects without disturbing host Python version.

Requirements

sudo access

Users need sudo access to:

- `apt-get update` or `apt update`
- `apt-get install` or `apt install`

Options

The checksums comparison in this document makes use of `html2text`

```
sudo apt update
sudo apt install html2text
```

Install miniconda

Is miniconda alr eady installed?

```
conda --version
```

Target Workstation

```
mkdir -p ~/sw/ubuntu/16.04/miniconda
cd ~/sw/ubuntu/16.04/miniconda
```

Download and verify

Latest

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Make executable

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
```

checksums

Download

```
wget https://repo.continuum.io/miniconda/  -O checksums.html
```

Convert to text

```
html2text -width 500 checksums.html > md5sums
```

Compare

```
md5sum Miniconda3-latest-Linux-x86_64.sh |  awk '{ print $1 }'
```

Compare to checksum file entry:

```
cat md5sums | grep Miniconda3-latest-Linux-x86_64.sh |  awk '{ print $5 }'
```

Installation

```
bash Miniconda3-latest-Linux-x86_64.sh  -b
```

Add Miniconda3 to your path (~/.bashr c or ~/.pr ofile)

```
cat ~/.bashrc | grep miniconda
```

To add to your path:

```
nano ~/.bashrc
```

Content example:

```
# set PATH to include Miniconda3's bin directory
PATH="$HOME/Miniconda3/bin:$PATH"
```

Confirmation

Start a new terminal session or run `source ~/.bashrc`

Version check

```
conda --version
```

List current conda stuff

```
conda list
```

Update conda

```
conda update -y conda
```

IMPORTANT: Host vs. Client Virtual Envir onments

OK, so you just installed miniconda and created a **host** virtual environment. It is important to keep in mind that this **host** virtual environment is going to be used to create **client** virtual environments. You want to always **create and activate** a **client** virtual environment before you install anything else or you are likely end up with with a broken **host** virtual environment. Using a pristine **host** environment will allow you to create many, many **client** virtual environments, each with it's own version of Python or R and each with it's own packages isolated from the host virtual environment and the system's default Python environment.

Create and activate a virtual p ython envir onment

Use the conda create command, followed by any name you wish to call it. Here we create two virtual environments:

```
conda create --name deep python=2
conda create --name rminc python=2
```

output example from the first command:

```
#
# To activate this environment, use:
# > source activate deep
#
# To deactivate this environment, use:
# > source deactivate deep
#
```

Activate the environment

If you run `which python` before activating your new Python environment you will see that the **host** Python environment that ships with miniconda is the active Python at the moment:

```
which python
```

Output example:

```
/home/users/mary/Miniconda3/bin/python
```

Activate one of your new Python environment:

```
source activate deep
```

Notice your prompt

When you activate a conda created Python environement the name you gave the environment prepended to your command prompt. In this example you will see that the prompt now begins with **(deep)** telling us that the custom python 2 environment we created earlier called **deep** is now the **active** Python for our user.

```
(deep) suser@ace-ws-30:~/sys/sw/linux/miniconda $
```

Running `which python` again will show you that the Python for the **deep** virtual environment is the active Python:

```
~/Miniconda3/envs/deep/bin/python
```

Install some stuff

Once you have activated your **client** virtual environment anything you install using Anaconda cloud or using the pip command will be installed into that environment.

Now we are going to install our pip packages to our custom virtual python environment. Notice that we do not need to use **sudo** or the **root** user as this installation takes place in a virtual environment that is in our home directory.

Deep project learning example:

```
pip install numpy matplotlib tensorflow
```

End of output example:

```
...
Successfully installed absl-py-0.1.12 astor-0.6.2 backports.functools-lru-cache-1.5
backports.weakref-1.0.post1 bleach-1.5.0 cycler-0.10.0 enum34-1.1.6 funcsigs-1.0.2
futures-3.2.0 gast-0.2.0 grpcio-1.10.0 html5lib-0.9999999 kiwisolver-1.0.1 markdown-
2.6.11 matplotlib-2.2.2 mock-2.0.0 numpy-1.14.2 pbr-3.1.1 protobuf-3.5.2.post1
pyparsing-2.2.0 python-dateutil-2.7.2 pytz-2018.3 six-1.11.0 subprocess32-3.2.7
tensorboard-1.6.0 tensorflow-1.6.0 termcolor-1.1.0 werkzeug-0.14.1
You are using pip version 9.0.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Lets go ahead and upgrade pip in our virtual environment:

```
pip install --upgrade pip
```

Create another custom Python envir onment

This time we will create a virtual environment called **data** running Python 3.

```
conda create --name data python=3
```

IMPORTANT: Even though we have created a new virtual environment, the previous virtual environment called **deep** is still the active virtual environment. If you install anything Pythonic now it would be installed to the **deep** Python environment and not the **data** virtual environment you just created. You will want to deactivate **deep** and then activate **data** before you can install to your Python 3 environment:

```
source deactivate deep
```

you will notice that the **(deep)** command prompt prefix is no longer visible. This means that the Miniconda **host** virtual environment is the active virtual environment right now. We want to keep it pristine so we will activate our new Python 3 environment called **data** before continuing:

```
source activate data
```

Now you are ready to install to the **deep** Python 3 environment. Note that you can switch environments without deactivating as activating a new environment automatically deactivates the currently active environment. In other words we could have just run `source activate data` .

Confirm that your prompt prefix is now **data** and install your stuff:

```
pip install ipython notebook
```

and upgrade pip in your **data** virtual environment

```
pip install --upgrade pip
```

Which pythons are available

You set up Python environments with (almost?) any version of Python available. Have a look:

```
conda search python
```

That's a who lotta Pythons...

List your current conda environments:

```
conda info --envs
```

output example:

```
# conda environments:
#
data                  *  /home/vagrant/Miniconda3/envs/data
deep                     /home/vagrant/Miniconda3/envs/deep
root                     /home/vagrant/Miniconda3
```

Hint:

Notice the environment named **base** or **root** in some cases. Again, this is the **host** Python environment which hosts the `conda` command. You will probably don't want to install to your root Python so ensure that you make one of your custom Pythons is active before installing programs using `pip` or the Anaconda cloud.

Anaconda cloud example

Update your host environment

First we will update our conda **host** environment

```
source deactivate deep
conda update conda
```

Create a new virtual environment client

Here we create a python environment using an Ansible conda package from Anaconda Cloud using the following values:

```
environment name  == ansible-2.0.0.2
python version  ==2.7
Anaconda cloud user kbroughton
Anaconda cloud package to install to environment   == ansible
```

Create our a new client environment called ansible-2.0.0.2:

(You can easily delete this new environment once you are finished with it.)

```
conda create --name ansible-2.0.0.2 python=2.7
```

Activate the ne w envir onment

```
source activate ansible-2.0.0.2
```

Notice that your command prompt has changed telling you which virtual environment you are in. When you are finished using an virtual environment, deactivate as follows, but **do not deactivated it just yet** as we are going to install an Anaconda Cloud package into it next:

```
# source deactivate ansible-2.0.0.2
```

Make sure the environment is the active one. Look for the environments name `(ansible-2.0.0.2)` command prompt preface and then install your Anaconda Cloud package.

```
conda install -c kbroughton ansible=2.0.0.2
```

Confirm that things are working as they should:

```
which ansible
```

Output example:

```
/home/users/csteel/miniconda/envs/ansible-2.0.0.2/bin/ansible
```

Notice that all of your conda's hosted virtual environments are in the `/home/users/csteel/miniconda/envs/` directory.

Check your Ansible version:

```
ansible --version
```

Output example:

```
ansible 2.0.0.2
  config file = /etc/ansible/ansible.cfg
  configured module search path  = Default w/o overrides
```

Removing environments

You can delete any non-root Python environments you have created using the `conda` command like this:

```
rm -R /home/vagrant/Miniconda3/envs/data
```

... but you might want to check out the manual first eh?

Completely removing miniconda

If you accidentally destroy your Miniconda installation you can back up any environments in the `envs` directory and then delete the `~/miniconda` directory. Keep in mind that this will also delete any host environments you left in the miniconda `envs` directory.

```
rm -rf ~/miniconda
rm -rf ~/.condarc ~/.conda ~/.continuum
```

remove PATH environment variable

```
edit ~/.bash_profile
```