Environment Modules

Gadi uses Environment Modules to customise user shell environment. The Environment Modules package is a tool that simplifies shell initialisation and allow users to modify their environments using provided commands like `module load` and `module unload` during the session with modulefiles.

On Gadi, we prepare modulefiles for all software applications available on /apps and keep the recommended shell configurations to run the application version in the corresponding modulefile. To look up all the modules available on Gadi, run `module avail`. To look up for modules with name start with string <str> run `module avail <str>`.

Typically modulefiles instruct the module command to alter or set shell environment variables such as PATH. To look up the instructions in the modulefile, run `module show`. For example, the modulefile for application version python3/3.7.4 contains the following instructions.

```
$ module show python3/3.7.4
-------------------------------------------------------------------
/apps/Modules/modulefiles/python3/3.7.4:

prepend-path    PATH /apps/python3/3.7.4/bin
prepend-path    C_INCLUDE_PATH /apps/python3/3.7.4/include/python3.7m
prepend-path    CPLUS_INCLUDE_PATH /apps/python3/3.7.4/include/python3.7m
prepend-path    CPATH /apps/python3/3.7.4/include/python3.7m
prepend-path    FPATH /apps/python3/3.7.4/include/python3.7m
prepend-path    LIBRARY_PATH /apps/python3/3.7.4/lib
prepend-path    LD_LIBRARY_PATH /apps/python3/3.7.4/lib
prepend-path    LD_RUN_PATH /apps/python3/3.7.4/lib
prepend-path    MANPATH /apps/python3/3.7.4/share/man
prepend-path    PKG_CONFIG_PATH /apps/python3/3.7.4/lib/pkgconfig
module          load intel-mkl/2019.3.199
conflict        python3
setenv          PYTHON3_BASE /apps/python3/3.7.4
setenv          PYTHON3_ROOT /apps/python3/3.7.4
setenv          PYTHON3_VERSION 3.7.4
module-whatis   {python3, version 3.7.4}
-------------------------------------------------------------------
```

It tells the command `module load` to set the environment variables PYTHON3_BASE, PYTHON3_ROOT, and PYTHON3_VERSION to the corresponding values.

It also tells `module load` to prepend the `bin` folder to the environment variable PATH in order to let the shell environment know where to find the executables provided by the python3/3.7.4 installation. There are also instructions for adding the path to the folder that contains the header files, `/apps/python3/3.7.4/include/python3.7m`, to the environment variables C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, CPATH, and FPATH, and the path to the folder that contains libraries, `/apps/python3/3.7.4/lib`, to LIBRARY_PATH, LD_LIBRARY_PATH, and LD_RUN_PATH in order to let the shell environment know where to find the header files and libraries, respectively, that are expected to be part of python version 3.7.4. There are also paths added to MANPATH and PKG_CONFIG_PATH to tell the command `man` and `pkg-config` where to search for python3/3.7.4 related man pages and .pc files, respectively.

In the modulefile, it also says `module load intel-mkl/2019.3.199`. This instruction tells when loading the module python3/3.7.4, load the module intel-mkl/2019.3.199 together with it. We recommend this configuration because python3/3.7.4 is built with intel-mkl libraries and it is necessary to set environment variables for it too.

Since there are going to be multiple versions of the same application installed on Gadi and, in a certain shell, only one can be used, it is important to tell that all other module versions conflict with the current one. In the above example, `conflict python3` instructs the `module load` command not to load any other python3 modules if the module python3/3.7.4 has been already loaded to the current shell.

Load Modulefiles of a Specific Version of Applications

Users are encouraged to load modulefiles with their fullnames, <app>/<version>, to remove ambiguity and improve the reproducibility of works on Gadi, especially in job submission scripts and their private modulefiles. Because most of software applications Gadi keeps centrally on /apps are expected to have multiple versions available and we keep installing newly released versions, it is the best to load modulefiles with the specific version to avoid using a wrong version when resuming/replicating previous works on Gadi. For example,

```
$ module load openmpi
$ module list
Currently Loaded Modulefiles:
 1) openmpi/3.1.4
...
###################
# 3 months later  #
###################
...
$ module load openmpi
$ module list
Currently Loaded Modulefiles:
 1) openmpi/4.0.2
```

We always set the default version of the application to the most recent or the most reliable version, therefore, it is expected that `module load openmpi` loads different versions of the openmpi application overtime. If your binary is compiled with openmpi/3.1.4 and the job submission script has `module load openmpi` in it, it may run at the time version 3.1.4 is still the default version, but it could fail 3 months later when the newer version is installed and set to the default version for openmpi applications.

Module Sub-commands

Command	Notes
module list	List all loaded modules in the current shell environment
module purge	Remove all modulefiles from the current shell environment.
module load <app>/<version>	Load modulefile <app>/<version> into the current shell environment.
module unload <app>	Remove modulefile for application <app> in the current shell
module avail <str>	List all available modulefiles in the current MODULEPATH whose pathname starts with <str>. All directories in the MODULEPATH are recursively searched for the target files that contain the modulefile magic cookie. If no argument <str> is given, all the modulefiles are displayed.
module whatis <app>/<version>	Display the information set up by the module-whatis commands inside the modulefile <app>/<version>.

User Defined Modules

Users can edit their own modulefiles for different versions of applications in their home folder ~/privatemodules. Once the modulefile <modulefile> is created in the `privatemodules` folder, first run

```
module load use.own
```

to include `~/privatemodules` in the search path MODULEPATH and prepare the reference counter to start tracking it, then run `module load <modulefile>` as you would load other modulefiles for applications installed on /apps.

For example, after installing python packages `neural-structured-learning`, with the module `python3/3.7.4` and `tensorflow/2.0.0` loaded, to a site-packages folder in your home folder, like

```
$ module purge
$ unset PYTHONPATH
$ module load python3/3.7.4
$ module load tensorflow/2.0.0
$ pip3 install -v --no-binary :all: --prefix=$HOME/envs/nsl/1.1.0 neural-structured-learning==1.1.0
```

edit the modulefile ~/privatemodules/nsl/1.1.0 as

```
#%Module1.0
prereq     python3/3.7.4
prereq     tensorflow/2.0.0
prepend-path    PYTHONPATH [getenv HOME]/envs/nsl/1.1.0/lib/python3.7/site-packages
```

so that loading the modulefile `nsl/1.1.0` gets the environment prepared for jobs need neural-structured-learning version 1.1.0 imported. For example,

```
$ module load use.own
$ module load python3/3.7.4
$ module load tensorflow/2.0.0
$ module load nsl/1.1.0
$ python3
>> import tensorflow as tf
>> import neural-structured-learning as nsl
>> nsl.__version__
1.1.0
```