Greenplanet Software (EL7)

- Software Environment
- Python (Conda/Jupyter/Etc.)
    - Miniconda
    - Miniforge
    - JupyterHub
- X11
    - X11 for Mac
    - X11 for Windows
    - X11 for Linux
- Compiler Variants
- OpenMP Variants
- OpenMPI
- Software Catalog

Software Environment

Environment Modules (Lmod) are used to configure your environment for many pieces of installed software on greenplanet, including compilers and MPI environments.

To see a list of available packages, type: "ml spider"

```
ml spider
```

To load a specific package/version, type: `ml modulename/versionnumber`. The 2020.1 release of the Intel Compilers is: Type: "ml intel/2020.1"

```
ml intel/2020.1
```

To see which modules are currently loaded: Type: "ml"

```
ml
```

If you wish to make your settings default for all login sessions and jobs, place your ml commands in ~/.bashrc. If you want to use them just in a specific job, add them to the job script.

Some pieces of software may be installed to `/sopt` on the cluster (shared opt) and not be configured in modules. Other packages that were modules previously may now be provided by the OS.

Note: it is highly recommended to a have `ml purge` at the beginning of the job execution portion of your slurm submission script. This clears out any accidental environments picked up from your login shell.

Python (Conda/Jupyter/Etc.)

Miniconda

> The default installation of Miniconda may have restrictive licensing on some packages. The drop-in replacement "miniforge" will set up a similar minimal conda environment that defaults to the "conda-forge" channel of open packages.

As the packages in Python distributions can change rapidly, it is difficult to have a single, system-wide installation that is useful to everyone. We will continue to install some basic ones in /sopt, but they are not likely to change after initial build.

To let users customize a minimal Python environment that won't disturb others, we suggest installing the "Miniconda" version of the Anaconda Python distro. (See https://conda.io/miniconda.html for additional info.)

Example Install 1) Python3-based conda installed in the default $HOME/miniconda3/ directory:

```
cd
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod u+x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -b
```

Example Install A) Same as option1, but installed to /DFS-L/DATA/$group/$user/miniconda3/ directory, with link in home directory (allows installing very large numbers of packages that would make your home directory go over quota):

```
cd
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod u+x Miniconda3-latest-Linux-x86_64.sh
mkdir /DFS-L/DATA/$(id -gn)/$USER/miniconda3
ln -s /DFS-L/DATA/$(id -gn)/$USER/miniconda3 miniconda3
./Miniconda3-latest-Linux-x86_64.sh -bu
```

As of 23-November-2020, this installs Miniconda3 4.9.2, which uses Python 3.8.5. It also installs the minimal set of packages:

```
_libgcc_mutex-0.1-main
brotlipy-0.7.0-py38h27cfd23_1003
ca-certificates-2020.10.14-0
```

```
certifi-2020.6.20-pyhd3eb1b0_3
cffi-1.14.3-py38h261ae71_2
chardet-3.0.4-py38h06a4308_1003
conda-4.9.2-py38h06a4308_0
conda-package-handling-1.7.2-py38h03888b9_0
cryptography-3.2.1-py38h3c74f83_1
idna-2.10-py_0
ld_impl_linux-64-2.33.1-h53a641e_7
libedit-3.1.20191231-h14c3975_1
libffi-3.3-he6710b0_2
libgcc-ng-9.1.0-hdf63c60_0
libstdcxx-ng-9.1.0-hdf63c60_0
ncurses-6.2-he6710b0_1
openssl-1.1.1h-h7b6447c_0
pip-20.2.4-py38h06a4308_0
pycosat-0.6.3-py38h7b6447c_1
pycparser-2.20-py_2
pyopenssl-19.1.0-pyhd3eb1b0_1
pysocks-1.7.1-py38h06a4308_0
python-3.8.5-h7579374_1
readline-8.0-h7b6447c_0
requests-2.24.0-py_0
ruamel_yaml-0.15.87-py38h7b6447c_1
setuptools-50.3.1-py38h06a4308_1
six-1.15.0-py38h06a4308_0
sqlite-3.33.0-h62c20be_0
tk-8.6.10-hbc83047_0
tqdm-4.51.0-pyhd3eb1b0_0
urllib3-1.25.11-py_0
wheel-0.35.1-pyhd3eb1b0_0
xz-5.2.5-h7b6447c_0
yaml-0.2.5-h7b6447c_0
zlib-1.2.11-h7b6447c_3
```

This uses about 323MB of disk space.

Once installed, you can set up the environment paths to your private version (with either install option) using:

```
ml miniconda/3/own
```

Installing new packages (e.g. numpy) within your miniconda3 directory is as simple as:

```
conda install numpy
```

Miniconda2 can be installed in a similar way with:

```
cd
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
chmod u+x Miniconda2-latest-Linux-x86_64.sh
./Miniconda2-latest-Linux-x86_64.sh -b
```

There is also the corresponding miniconda/2/own module. Only one python module (miniconda/anaconda/python/Intel-python, all with versions 2 or 3) can be loaded at a time.

Miniforge

> The default installation of Miniconda may have restrictive licensing on some packages. The drop-in replacement "miniforge" will set up a similar minimal conda environment that defaults to the "conda-forge" channel of open packages.

As the packages in Python distributions can change rapidly, it is difficult to have a single, system-wide installation that is useful to everyone. We will continue to install some basic ones in /sopt, but they are not likely to change after initial build.

To let users customize a minimal Python environment that won't disturb others, we suggest installing the "Miniforge" version of the Conda Python environment. (See https://conda-forge.org/download/ for additional info.)

Example Install 1) Python3-based conda installed in the default $HOME/miniforge3/ directory:

```
cd
wget https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh
chmod u+x Miniforge3-Linux-x86_64.sh
./Miniforge3-Linux-x86_64.sh -b
```

Example Install A) Same as option1, but installed to /X2/SCRATCH/$group/$user/miniforge3/ directory, with link in home directory (allows installing very large numbers of packages that would make your home directory go over quota):

```
cd
wget https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh
chmod u+x Miniforge3-Linux-x86_64.sh
mkdir /X2/SCRATCH/$(id -gn)/$USER/miniforge3
ln -s /X2/SCRATCH/$(id -gn)/$USER/miniforge3 miniforge3
./Miniforge3-Linux-x86_64.sh -bu
```

As of 30-Marchr-2026, this installs Conda 26.1.1, which uses Python 3.13.12.

This uses about 552MB of disk space.

Once installed, you can set up the environment paths to your private version (with either install option) using:

```
ml miniforge/3/own
```

Updating will upgrade a few packages

```
conda update --all
```

Installing new packages (e.g. numpy) within your miniforge3 directory is as simple as:

```
conda install numpy
```

You can create a new environment for CPU-based PyTorch:

```
conda create --name torch-CPU pytorch torchvision torchaudio cpuonly -c pytorch
```

Or one for PyRosetta:

```
conda create --name pyrosetta -c https://conda.rosettacommons.org
```

Only one python module (miniforge/miniconda/anaconda/python/Intel-python) can be loaded at a time.

JupyterHub

While Jupyter notebooks can be run on any login node by installing the needed packages via conda or pip, running on compute nodes can be tricky. We have installed JupyterHub servers on gplogin2 and gplogin3 that can spawn JupyterLab sessions as slurm jobs.

Starting

Point your local computer's Web browser to either https://gplogin2.ps.uci.edu:8000 or https://gplogin3.ps.uci.edu:8000 and enter your Greenplanet username and password. Then, for the "simple" configuration:

1. Select which partition to run in
2. Select number of CPUs
3. Select which Jupyter environment to use (more info below)
4. If JupyterLab is desired, click box
5. Select runtime limit
6. Click Start

For "Advanced", you have much more control of each option. See https://github.com/silx-kit/jupyterhub_moss for detailed descriptions.

Jupyter environments

Default

The default Python environment (/sopt/JupyterHub/) is what is used to run the JupyterHub software. It is not user-customizable, but has a lot of general-use conda and pip packages and may just work. We can update and add more packages if needed at any time, so don't count on it having specific package versions.

Miniforge3

One of the pre-configured environments assumes you have miniforge3 set up in your home directory at ~/miniforge3 (this can also be a link to somewhere else, like /X2/SCRATCH/group/user/miniforge3). See https://knowledge.ps.uci.edu/books/greenplanet-software/page/miniforge for details on preparing that.

The miniforge/3 environment will use the base miniforge that is activated by loading the module with "module load miniforge/3". You will need to run "conda install jupyterhub jupyterlab batchspawner" to make sure the JupyterLab prerequisites are installed.

User JupyterHub

If you want to keep your conda base environment clean, you can create a custom environment at ~/jupyterhub through either conda or pip. For example, starting with miniforge3's conda, do this to install a custom environment that is stored in your directory on /X2:

```
mkdir /X2/SCRATCH/$(id -gn)/$USER/jupyterhub
ln -s /X2/SCRATCH/$(id -gn)/$USER/jupyterhub ~/jupyterhub
ml miniforge/3
conda create -y -p ~/jupyterhub batchspawner jupyterhub jupyterlab
conda activate ~/jupyterhub
conda install [list of needed packages for your notebook]
pip install [list of any needed packages not in conda]
```

Other Locations

If you need to run in another arbitrary python environment, you can enter the name, location, and any environment modules you want to load manually in the "Advanced" tab.

X11

So one can run X11 (graphical) apps on the cluster and see the output on your local display, you may need to see one of the below links if you don't already have this configured.

X11 for Mac

MacOS 10.9 and later (11.2 “Big Sur” Working as of 4 March 2021):

1. Upgrade OS to latest available (not always necessary, but usually avoids bugs)
2. Upgrade to (or install) XQuartz-2.8.5
3. Log out/Log in to your Mac (may not be needed for upgrades from earlier 2.8.0 releases)
4. starting Mac terminal (or x2go, if installed) should trigger xquartz.
5. For OpenGL graphics to work, you may need to turn on indirect GLX by typing the following into your terminal:

   ```
   defaults write org.xquartz.X11 enable_iglx -bool true
   ```

Troubleshooting:

- If `glxgears` gives the error: “couldn’t get an RGB, Double-buffered visual”, add the following to your GreenPlanet .bash_profile or equivalent:

  ```
  export __GLX_VENDOR_LIBRARY_NAME=mesa
  ```

- If Matlab windows keep blanking out, create a file "java.opts" in your GreenPlanet home folder with the following content:

  ```
  -Dsun.java2d.xrender=false
  -Dsun.java2d.pmoffscreen=false
  ```

  (If you start Matlab from a different folder, the java.opts may need to be put there instead.)

Before 10.9, use XQuartz 2.7.11

  - Warning: Some ancient apps may require an even *older* version (2.7.8) of XQuartz due to incompatibilities with “El Capitan” and “Mojave”

X11 for Windows

Here are a list of X Windows servers for Windows

X2Go

Requires an X2Go server on the target server. Creates a full remote desktop connection.

- https://wiki.x2go.org/doku.php
- https://wiki.x2go.org/doku.php/doc:installation:x2goclient

Note: X2Go is great for overcoming platform specific bugs like requiring certain versions of XQuartz for a given app on macOS

MobaXterm

Terminal / SSH chient with X11 support.

- https://mobaxterm.mobatek.net/download.html
- https://mobaxterm.mobatek.net/download-home-edition.html (select Installer edition)

Note: works great for accessing CentOS/Ubuntu, but haven't fully tested this personally to access X apps on macOS

Cygwin

- https://x.cygwin.com/

Note: mileage varies depending on the repo (i.e., if it's not up to date).

XMing X Server

- https://sourceforge.net/projects/xming/
- http://www.straightrunning.com/XmingNotes/

Windows Subsystem for Linux

- https://docs.microsoft.com/en-us/windows/wsl/install-win10

Inspired by https://statistics.berkeley.edu/computing/x11-forwarding

X11 for Linux

X2Go

Requires an X2Go server running on the target server. Creates a full remote desktop session.

Client can be installed through the system repos, eg on Ubuntu:

```
sudo apt install x2goclient
```

X11 Forwarding

As long as your current environment supports X11, X11 forwarding can be enabled for an SSH connection using the -Y flag, eg:

```
ssh -Y username@gplogin3.ps.uci.edu
```

The forwarding can be tested by running `glxgears`.

Compiler Variants

Here are the compiler options on the new side of the cluster (gplogin2/3).

gnu => GCC; intel => Intel Parallel Studio XE; mkl => Intel Math Kernel Library

For MPI Variants see this page.

Compiler/Version	To use
gnu/4.8.5 gnu/7.2.0 gnu/7.3.0 gnu/8.1.0 gnu/8.2.0 gnu/8.3.0 gnu/9.1.0	ml gnu/4.8.5 ml gnu/7.2.0 ml gnu/7.3.0 ml gnu/8.1.0 ml gnu/8.2.0 ml gnu/8.3.0 ml gnu/9.1.0
intel/2017.2 intel/2017.4 intel/2017.5 intel/2018.0 intel/2018.1 intel/2018.2 intel/2018.3	ml intel/2017.2 ml intel/2017.4 ml intel/2017.5 ml intel/2018.0 ml intel/2018.1 ml intel/2018.2 ml intel/2018.3
intel/2018.0 + mkl intel/2018.1 + mkl intel/2018.2 + mkl intel/2018.3 + mkl	ml intel/2018.0 mkl ml intel/2018.1 mkl ml intel/2018.2 mkl ml intel/2018.3 mkl

Note: If you require MKL, please use a recent version from 2018 and newer versions are supposed to be backward compatible.

OpenMP Variants

OpenMP is a threading option built into GCC/Intel compilers.

https://www.openmp.org/resources/openmp-compilers-tools/

Intel options:

- OpenMP 4.5 C/C++/Fortran supported in version 17.0 compilers
- OpenMP 4.5 C/C++/Fortran supported in version 18.0 compilers
- Compile with -Qopenmp on Windows, or just -openmp or -qopenmp on Linux or Mac OS X / macOS

OpenMPI

Here are all the possibilities for the currently installed OpenMPI versions, so you can compile any application with the desired version of OpenMPI.

OpenMPI Version	**Use GNU Compiler**	**Use Intel Compiler**
openmpi/1.10.7	ml gnu/7.2.0 openmpi/1.10.7	ml intel/2018.0 openmpi/1.10.7
openmpi/2.1.1		ml intel/2017.2 openmpi/2.1.1
openmpi/2.1.2	ml gnu/7.2.0 openmpi/2.1.2	ml intel/2017.4 openmpi/2.1.2
openmpi/3.0.1a	ml gnu/7.2.0 openmpi/3.0.1a	ml intel/2018.0 openmpi/3.0.1a
openmp/3.0.1	ml gnu/7.3.0 openmpi/3.0.1	ml intel/2018.3 openmpi/3.0.1
openmpi/3.1.1	ml gnu/8.2.0 openmpi/3.1.1	ml intel/2018.2 openmpi/3.1.1 ml intel/2018.3 openmpi/3.1.1
openmpi/3.1.2 openmpi/3.1.2-slim	ml gnu/8.2.0 openmpi/3.1.2 ml gnu/8.2.0 openmpi/3.1.2-slim	ml intel/2018.3 openmpi/3.1.2 ml intel/2018.3 openmpi/3.1.2-slim

ml spider openmpi/x.y.z to generate columns 2 and 3

Be advised that only 2.1.x, 3.0.x and 3.1.x are currently under development and prior versions are **retired**!

Software Catalog

Below is a table of all custom compiled software available on the new side of cluster (gplogin2/3) via lmod.

If compiling an app from source see Compiler Variants.

If compiling an MPI app from source see OpenMPI Variants.

This serves as a quick reference on how to load any package based on its prerequisite compiler if any.

Software	Indepedent	GNU prerequisite	Intel prerequisite
anaconda/2 anaconda/3	ml anaconda/2 ml anaconda/3		
arpack-ng/3.5.0	ml arpack-ng/3.5.0		ml intel/2018.2 arpack-ng/3.5.0
avogadro/1.2	ml avogadro/1.2		
castep/19.1		ml intel/2018.3 openmpi/4.0.1 castep/19.1	
cp2k/4.1 cp2k/5.1	ml cp2k/4.1 ml cp2k/5.1		
dftbplus/18.1			ml intel/2018.2 dftbplus/18.1
fftw/3.3.6-pl2 fftw/3.3.8		ml gnu/7.2.0 fftw/3.3.6-pl2 ml gnu/8.2.0 fftw/3.3.8 (09/06)	ml intel/2018.0 fftw/3.3.6-pl2 ml intel/2018.3 fftw/3.3.8 (09/06)
gamess/2018.R3			ml intel/2018.3 openmpi/3.1.1 gamess/2018.R3 ml intel/2018.3 openmpi/3.1.2 gamess/2018.R3 ml intel/2018.3 openmpi/3.1.2-slim gamess/2018.R3

gaussian/09 gaussian/16	ml gaussian/09 ml gaussian/16		
gerris/1.3.2		ml gnu/8.2.0 openmpi/3.1.1 gerris/1.3.2 ml gnu/8.2.0 openmpi/3.1.2 gerris/1.3.2 ml gnu/8.2.0 openmpi/3.1.2-slim gerris/1.3.2	
gromacs/2018.3		ml gnu/7.3.0 openmpi/3.0.1 gromacs/2018.3	
hdf5/1.8.21 hdf5/1.10.2		ml gnu/8.2.0 hdf5/1.8.21 ml gnu/8.2.0 openmpi/3.1.1 hdf5/1.8.21 ml gnu/8.2.0 openmpi/3.1.2 hdf5/1.8.21 ml gnu/7.3.0 hdf/1.10.2 ml gnu/7.3.0 openmpi/3.0.1 hdf5/1.10.2	ml intel/2018.3 hdf5/1.8.21 ml intel/2018.3 openmpi/3.1.1 hdf5/1.8.21 ml intel/2018.3 openmpi/3.1.2 hdf5/1.8.21 ml intel/2018.2 hdf/1.10.2 ml intel/2018.2 openmpi/3.0.1 hdf5/1.10.2
hwloc/1.11.10 hwloc/1.11.11 hwloc/2.0.1 hwloc/2.0.2 hwloc/2.0.3	ml hwloc/1.11.10 ml hwloc/1.11.11 ml howloc/1.11.12 ml hwloc/2.0.1 ml hwloc/2.0.2 ml hwloc/2.0.3 (compiled w/ gnu/4.8.5)		
idl/8.1	ml idl/8.1		
julia/0.6.4 julia/0.7.0 julia/1.0.0 julia/1.1.0	ml julia/0.6.4 ml julia/0.7.0 ml julia/1.0.0 ml julia/1.1.0		
matlab/R2017b matlab/R2018b	ml matlab/R2017b ml matlab/R2018b		
miniconda/2/own miniconda/3/own	ml miniconda/2/own ml miniconda/3/own		
molden/5.7	ml molden/5.7		

namd/2.10_REST namd/2.12 namd/2.13b1		ml gnu/7.2.0 openmpi/3.0.1a namd/2.10_REST ml gnu/7.2.0 openmpi/3.0.1a namd/2.12	ml intel/2018.3 openmpi/3.1.2 namd/2.13b1
ncl/6.3.0 ncl/6.4.0 ncl/6.5.0	ml ncl/6.3.0 ml ncl/6.4.0 ml ncl/6.5.0		
netcdf/4.4.1.1 netcdf/4.6.1 netcdf/4.7.0		ml gnu/8.2.0 netcdf/4.4.1.1 ml gnu/8.2.0 openmpi/3.1.1 netcdf/4.4.1.1 ml gnu/8.2.0 openmpi/3.1.2 netcdf/4.4.1.1 ml gnu/8.2.0 openmpi/3.1.2-slim netcdf/4.4.1.1 ml gnu/8.3.0 netcdf/4.7.0 ml gnu/8.3.0 openmpi/4.0.1 netcdf/4.7.0	ml intel/2018.3 netcdf/4.4.1.1 ml intel/2018.3 openmpi/3.1.1 netcdf/4.4.1.1 ml intel/2018.3 openmpi/3.1.2 netcdf/4.4.1.1 ml intel/2018.3 openmpi/3.1.2-slim netcdf/4.4.1.1 ml intel/2018.2 netcdf/4.6.1 ml intel/2018.2 openmpi/3.0.1 netcdf/4.6.1 ml intel/2018.3 netcdf/4.7.0 ml intel/2018.3 openmpi/4.0.1 netcdf/4.7.0
openblas/0.2.20- openmp_64 openblas/0.2.20-openmp openblas/0.2.20-single_64 openblas/0.2.20-single openblas/0.2.20- pthreads_64 openblas/0.2.20-pthreads		ml gnu/7.2.0 openblas/0.2.20- openmp_64 ml gnu/7.2.0 openblas/0.2.20-openmp ml gnu/7.2.0 openblas/0.2.20-single_64 ml gnu/7.2.0 openblas/0.2.20-single ml gnu/7.2.0 openblas/0.2.20- pthreads_64 ml gnu/7.2.0 openblas/0.2.20-pthreads	

openmpi/1.10.7 openmpi/2.1.1 openmpi/2.1.2 openmpi/3.0.1a openmpi/3.0.1 openmpi/3.1.1 openmpi/3.1.2 openmpi/3.1.2-slim		ml gnu/7.2.0 openmpi/1.10.7 ml gnu/7.2.0 openmpi/2.1.2 ml gnu/7.2.0 openmpi/3.0.1a ml gnu/7.3.0 openmpi/3.0.1 ml gnu/8.2.0 openmpi/3.1.1 ml gnu/8.2.0 openmpi/3.1.2 ml gnu/8.2.0 openmpi/3.1.2-slim	ml intel/2018.0 openmpi/1.10.7 ml intel/2017.2 openmpi/2.1.1 ml intel/2017.2 openmpi/2.1.2 ml intel/2018.0 openmpi/3.0.1a ml intel/2018.2 openmpi/3.0.1 ml intel/2018.2 openmpi/3.1.1, ml intel/2018.3 openmpi/3.1.1 ml intel/2018.3 openmpi/3.1.2 ml intel/2018.3 openmpi/3.1.2-slim
pio/2.3.1		ml gnu/8.2.0 openmpi/3.1.1 pio/2.3.1 ml gnu/8.2.0 openmpi/3.1.2 pio/2.3.1	ml intel/2018.2 openmpi/3.0.1 pio/2.3.1 ml intel/2018.3 openmpi/3.1.1 pio/2.3.1 ml intel/2018.3 openmpi/3.1.2 pio/2.3.1
pnetcdf/1.9.0 pnetcdf/1.10.0		ml gnu/7.3.0 openmpi/3.0.1 pnetcdf/1.9.0 ml gnu/8.2.0 openmpi/3.1.1 pnetcdf/1.10.0 ml gnu/8.2.0 openmpi/3.1.2 pnetcdf/1.10.0	ml intel/2018.2 openmpi/3.0.1 pnetcdf/1.9.0 ml intel/2018.3 openmpi/3.1.1 pnetcdf/1.10.0 ml intel/2018.3 openmpi/3.1.2 pnetcdf/1.10.0
python/2.7.14 python/3.6.4	ml python/2.7.14 ml python/3.6.4		
qchem/5.0		ml gnu/7.2.0 openmpi/1.10.7 qchem/5.0	
qespresso/6.3			ml intel/2018.3 openmpi/3.1.1 qespresso/6.3 ml intel/2018.3 openmpi/3.1.2 qespresso/6.3
turbomole/7.2-official turbomole/7.2.1-official turbomole/7.3-beta turbomole/7.3-official		ml turbomole/7.2-official ml turbomole/7.2.1-official ml turbomole/7.3-beta ml turbomole/7.3-official	

udunits/2.2.26	ml udunits/2.2.26 (compiled w/ gnu/4.8.5)		
vmd/1.9.3	ml vmd/1.9.3		

One can generate a list of all available custom installed software via `ml spider`
Then `ml spider <package>` to get details on all variants of a given package.
The later is how the above table was generated!