University of Manitoba

Digital Research Alliance of Canada

High Performance Computing and software environments:

Install and/or use existing environments and modules

UofM-Spring-Workshop 2022

May 4th-5th, 2022

Ali Kerrache

University of Manitoba

Outline

- Software distribution on HPC clusters
- Why modules? How to find modules?
- Software stacks on Grex
- How to build software from sources
  - R packages
  - Perl modules
  - configure/make
  - cmake/make
- Singularity

Software distribution

Operating system package managers / repos:

- ★ **Ubuntu:** ~$ ***sudo apt-get install <package>***
- ★ **CentOS:** ~$ ***sudo yum install <package>***
- ★ **On HPC:** users do not have **sudo**! (**NO NEED TO ASK FOR IT**)

Local installation: usually to $HOME or $PROJECT

- ★ **Get the code:** download the sources/binaries: wget, git clone, … etc.
- ★ **Settings:** load dependencies, set environment variables, … etc.
- ★ **Build:** ./configure {***cmake*** ..} +opts; make; make test {***check***}; make install

Using a centralized HPC software stack:

- ★ **Software distributed via CVMFS:** CC software stack (CC clusters), …
- ★ **Local software:** modules, legally restricted software (VASP, Gaussian, …)

Software layers

User layer: Python packages, Perl and R modules, home made codes, …

Easybuild layer: modules for Intel, PGI, OpenMPI, CUDA, MKL, high-level applications. Multiple architectures (sse3, avx, avx2, avx512)

Nix or gentoo layers: GNU libc, autotools, make, bash, cat, ls, awk, grep, etc.

Gray area: Slurm, Lustre client libraries, IB/OmniPath/InfiniPath client libraries (all dependencies of OpenMPI) in Nix {or gentoo} layer, but can be overridden using PATH & LD_LIBRARY_PATH.

OS: kernel, daemons, drivers, libcuda, anything privileged (e.g. the sudo command): always local. Some legally restricted software too (VASP).

Why modules?

- **Why modules?**
  - Control different versions of the same program.
  - Avoid conflicts between different versions and libraries.
  - Set the right path to each program or library.
- **Useful commands for working with modules:**
  - module **list**; module **avail**
  - module **spider** <soft>/<version>
  - module **load** soft/version; module **unload {rm}** <soft>/<version>
  - module **show** soft/version; module **help** <soft>/<version>
  - module **purge**; module --force **purge**
  - module **use** ~/modulefiles; module **unuse** ~/modulefiles

Software stacks on Grex

- ★ Grex environment [default]: GrexEnv
  - no module loaded by default.
  - use module spider <name of the software> to search for modules
  - Compilers: {GCC, Intel}, MKL, PETSc, … etc.
  - Gaussian, ANSYS, MATLAB, … etc.
- ★ Compute Canada environment [optional]: CCEnv
  - Switch to CCEnv; load a standard environment; choose the architecture[sse3, avx2, avx512], use module spider <soft>

```
module load CCEnv
module load StdEnv/2020
module load arch/avx512
module load StdEnv/2020  gcc/9.3.0 geant4/10.7.3
```

```
module load StdEnv/2016.4
module load arch/sse3
module load nixpkgs/16.09  gcc/5.4.0 geant4/10.05.p01
```

Modules on Grex

- About 500 modules:
  - GCC [5,7,9, 11]; Intel [2014 - 2020].
  - Libraries: HDF5, PETSc, GSL, MKL, Libxc, Boost, ...
  - Gaussian, ANSYS, MATLAB, VASP, MCR, Java, … etc.
  - LAMMPS, GROMACS, ABINIT, QE, VMD, Molden, … etc.
- Software maintenance on Grex:
  - We install programs and update modules on request from users.
  - Search for a program using "module spider <name of your program>"
  - If not installed, ask for support "support@computecanada.ca"
  - We will install the module or update the version.
  - For commercial software, contact us before you purchase the code:
    - to check license type.
    - see if it will run under Linux environment, … etc.

Building software

- Local installation (user's directory):
  - R packages; Julia packages
  - Python packages: virtual environment, conda
  - Perl modules
- Installation with:
  - make; make test {check}; make install
  - configure; make test {check}; make install
  - cmake; make test {check}; make install
- Java applications: jar files
- Singularity and/or Aptainer:
  - build the image and run the program from the container

Local installation

- R packages: minimal installation
  - R as modules: users can install the packages in their home directory.
- Python as modules: python and scipy-stack
  - users can install the packages needed in their home directory.
- Perl and bioperl as modules:
  - users can install the packages needed in their home directory.
- Other software installed locally:
  - Home made programs
  - Restricted and licensed software that can not be distributed via CVMFS.
  - Custom software: patch from a user, changing parts of the code, … etc.

https://docs.computecanada.ca/wiki/Installing_software_in_your_home_directory

Local installation: R packages

R packages: rgdal, adegenet, stats, rjags, dplyr, … etc.

Choose a module version: module spider r

Load R and dependencies (gdal, geos, jags, gsl, udunits… etc):

```
module load gcc/7.3.0 r/3.6.0 gdal udunits…
```

Launch R and install the packages:

```
~$ R
> install.packages(”sp")
'lib =/cvmfs/soft.computecanada.ca/easybuild/{..}/R/library'" is not writable
Would you like to use a personal library instead? (yes/No/cancel) yes
Would you like to create a personal library ‘~/R/{…}’ to install packages into? (yes/No/cancel) yes
--- Please select a CRAN mirror for use in this session ---
> install.packages(”dplyr")
```

Local installation: perl

Example: Hash::Merge; Logger::Simple; MCE::Mutex; threads …

Load Perl module: module load perl

Install the the first package using cpan or cpanm:

```
~$ cpan install YAML
Would you like to configure as much as possible automatically? [yes] yes
What approach do you want?  (Choose 'local::lib', 'sudo' or 'manual')
[local::lib] local::lib
Would you like me to append that to /home/$USER/.bashrc now? [yes] yes
```

Install the rest of the packages:

```
~$ cpan install Hash::Merge
~$ cpan install Logger::Simple
~$ cpan install MCE::Mutex
```

Installation with make: STAR

- ★ Download the code:

  wget https://github.com/alexdobin/STAR/archive/refs/tags/2.7.8a.tar.gz

- ★ Unpack the code: tar -xvf 2.7.8a.tar.gz
- ★ Load GCC compiler: module load gcc
- ★ Compile the code:

  cd STAR-2.7.8a/source

  make

- ★ Copy the binaries and set the path:

  mkdir -p ~/software/star/2.7.8a/bin

  cp STAR ~/software/star/2.7.8a/bin

  export PATH=$PATH:~/software/star/2.7.8a/bin

```
~/.bash_profile
PATH=$PATH:$HOME/bin:$HOME/software/star/2.7.8a/bin
export PATH
```

Installation with configure/make

- ★ Download and unpack the code: wget, … gunzip, … etc.
- ★ Load the modules and dependencies: module load gcc ompi fftw
- ★ Configure the program
  - If configure not included, run: autoreconf -fvi [to generate it].
  - ./configure --help [to see the different options].
  - ./configure --prefix=installdir {+other options}
- ★ Compile and test:
  - make
  - make check; make test
- ★ Install the program:
  - make install

Example: PETSc

```
./configure --with-blas-lapack-dir=$MKLROOT/lib/intel64  --prefix=${instdir}   --with-cxx-dialect=C++11
--download-scalapack=yes --download-blacs=yes   --download-superlu_dist=yes
--download-mumps=yes --download-parmetis=yes --download-metis=yes --download-spooles=yes
--download-cproto=yes --download-prometheus=yes --with-mkl_pardiso=1
--with-mkl_pardiso-dir=$MKLROOT --with-mkl-sparse-optimize=1   --with-scalar-type=complex
--with-debugging=0  --with-hdf5=yes --with-hdf5-dir=$HDF5HOME  --download-suitesparse=yes
--download-fftw=${fftsrc} --download-amd=yes --download-adifor=yes  --download-superlu=yes
--download-triangle=yes --download-generator=yes   --with-64-bit-pointers=no  --with-cc=mpicc
--CFLAGS='-O2  -msse4.2 -xSSE4.2 -mp1 -I$MKLROOT/include -mkl -fPIC '   --with-cxx='mpicxx'
--CXXFLAGS='-O2 -msse4.2 -xSSE4.2 -mp1 -I$MKLROOT/include -mkl -std=c++11 -fPIC '
--with-fc='mpif90'  --FFLAGS='-O2 -msse4.2 -xSSE4.2 -mp1 -I$MKLROOT/include -mkl -fPIC '
--with-single-library=yes --with-shared-libraries=yes  --with-shared-ld=mpicc
--sharedLibraryFlags="-fpic -mkl -fPIC"  --with-mpi=yes  --with-mpi-shared=yes --with-mpirun=mpiexec
--with-mpi-compilers=yes  --with-x=yes
```

Example: ABINIT

```
module load intel ompi gsl netcdf
instdir=<path to the installation directory>

../configure --prefix=${instdir} --enable-mpi --enable-mpi-io --with-fft-flavor=fftw3-mkl
--with-linalg-flavor=mkl --with-math-flavor=gsl --enable-debug="no"
--enable-optim="standard" --enable-64bit-flags --with-linalg-libs="-L$MKLROOT/lib/intel64
-lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64  -lmkl_intel_lp64 -lmkl_sequential
-lmkl_core -lm"  --with-fft-incs="-I$MKLROOT/include/fftw -I$MKLROOT/interfaces/fftw3xf"
--with-fft-libs="-L$MKLROOT/interfaces/fftw3xf -lfftw3xf_intel_lp64"
--with-dft-flavor="atompaw+libxc+wannier90" --with-trio-flavor="netcdf"  --enable-lotf
--enable-macroave   --enable-gw-dpc  CC=mpicc CXX=mpic++ FC=mpif90 F77=mpif77
F90=mpif90
```

Example with cmake/make

- ★ Download and unpack the code: wget, … gunzip, … etc.
- ★ Load the modules and dependencies: module load gcc ompi fftw
- ★ Configure the program: you may need to load cmake module
  - mkdir build && cd build
  - cmake .. --help [to see the different options].
  - cmake .. -DCMAKE_INSTALL_PREFIX=installdir {+other options}
- ★ Compile and test:
  - make
  - make check; make test
- ★ Install the program:
  - make install

Using the GUI:

ccmake .. {+ options}

Cmake options for GROMACS

```
module load intel/15.0
module load ompi/3.1.4 fftw
module load cmake

cd gromacs-5.1.4; mkdir build; cd build
cmake -DCMAKE_INSTALL_PREFIX=<path to install dir>  -DBUILD_SHARED_LIBS=off
-DBUILD_TESTING=off  -DREGRESSIONTEST_DOWNLOAD=off
-DCMAKE_C_COMPILER=`which mpicc` -DCMAKE_CXX_COMPILER=`which mpicxx`
-DGMX_BUILD_OWN_FFTW=on -DGMX_SIMD=SSE4.1 -DGMX_DOUBLE=off
-DGMX_EXTERNAL_BLAS=on -DGMX_EXTERNAL_LAPACK=on
-DGMX_FFT_LIBRARY=fftw3 -DGMX_GPU=off -DGMX_MPI=on -DGMX_OPENMP=off
-DGMX_X11=on ../gromacs-5.1.4
make -j4
```

Java applications

- ★ Download and unpack the code
- ★ Load java module: module load java
- ★ Run the code

- ★ Example: Trimmomatic
  - wget http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.39.zip
  - unzip Trimmomatic-0.39.zip

- ★ Run the code

```
module load java
java -jar <path to>/trimmomatic-0.39.jar {+options if any}
```

Singularity

Resources: Github, DockerHub, SingularityHub, Aptainer.

Singularity examples: https://github.com/singularityware/singularity/tree/master/examples

- ★ **Documentation:** http://singularity.lbl.gov/user-guide
- ★ **DockerHub:** https://hub.docker.com/explore/
- ★ **SingularityHub:** https://www.singularity-hub.org/
- ★ **Aptainer:** https://apptainer.org/

https://docs.computecanada.ca/wiki/Singularity
https://um-grex.github.io/grex-docs/

Access to Singularity:

- ★ **Connect to cluster:** Grex, cedar, graham, beluga or narval:
- ★ **Load a module:** module load singularity
- ★ **Build the image:** convert the image from Docker to Singularity
- ★ **Note:** You may need to use your own Linux machine or VM to build the image

Singularity

- ★ Alternative for running software: difficult to build from source
- ★ Possibilite to convert Docker images to singularity.
- ★ Singularity installed on all clusters {no Docker for security reasons}
- ★ Build the image:

```
module load singularity
singularity build qiime2-2019.10.sif docker://qiime2/core:2019.10
```

- ★ Run the code via singularity:

```
singularity exec -B $PWD:/home -B /global/scratch/someuser:/outputs  \
-B /global/scratch/someuser/path/to/inputs:/inputs qiime2-2019.10.sif   \
qiime feature-classifier fit-classifier-naive-bayes \
--i-reference-reads  /outputs/some_output_feature.qza \
--i-reference-taxonomy /outputs/some_output_ref-taxonomy.qza \
--o-classifier /outputs/some_output_classifier.qza
```

The Alliance [Compute Canada] wiki

- Systems and services
- Guides

- Links to specific documentation by disciplines
- Links to the documentation from regional partners

Systems and services [edit]

- List of current Compute Canada systems
- Cedar, Graham and Béluga, general-purpose clusters
  - System status and **upcoming outages**
  - Known issues
- Niagara, a cluster designed for large parallel jobs
- Hélios, a GPU cluster
- Available software
- National Data Cyberinfrastructure, long-term and tape storage services (limited availability)
- Cloud computing service
- Globus file transfer service
- Policy table of contents
- FAQ, Frequently Asked Questions
- Using a resource allocation, a guide for Principal Investigators
  - RAC 2019 transition FAQ, notes on the implementation of 2019 RAC awards

How-to guides [edit]

- Getting started
  - Getting started with the new national systems (mini-webinar series)
  - Niagara Quick Start Guide
  - SSH - How to connect to our servers
  - Linux introduction
- Storage and file management
  - Transferring data
  - Scratch purging policy
- Best practices for data migration
- Using modules to access software
- Running jobs
- Installing software yourself
- Programming guide
- Visualization
- How to get technical support

Discipline guides [edit]

- AI and Machine Learning
- Bioinformatics
- Biomolecular simulation
- Computational chemistry
- Computational fluid dynamics (CFD)
- Geographic information systems (GIS)
- Humanities
- Subatomic physics

Regional partners and services [edit]

- WestGrid
- SHARCNET
- SciNet
- Centre for Advanced Computing
- Calcul Québec
- ACENET
- ownCloud storage service

Grex documentation

University of Manitoba Unofficial Grex User Guide

Search

Notes of Grex Changes

Accessing Compute Canada resources

Grex HPC Documentation

Access and Usage conditions

Connecting / Transferring data

Storage and Data

Running Jobs

Software

Frequently Asked Questions

Local IT Resources

Support and Training

Disclaimer

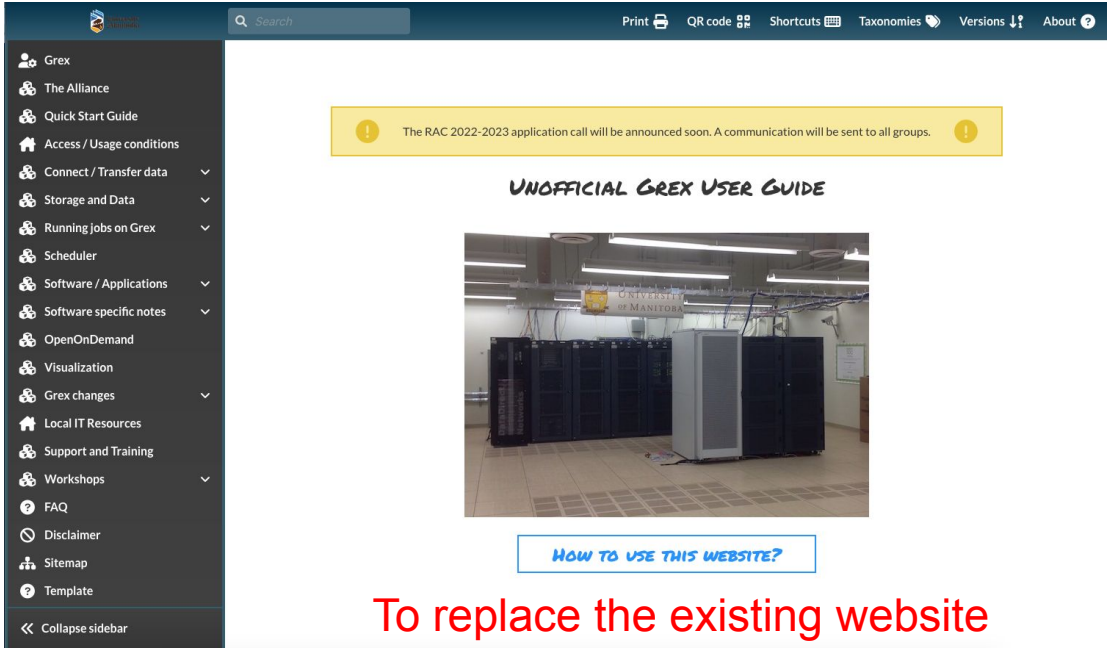
User documentation for HPC resources at University of Manitoba

Since you have found this Website, you may be interested in Grex documentation. Grex is the University of Manitoba's High-Performance Computing system.

- Updating the documentation after adding the new hardware.
- Hosted on GitHub
- The link is available as MOTD when login to Grex.

https://um-grex.github.io/grex-docs/

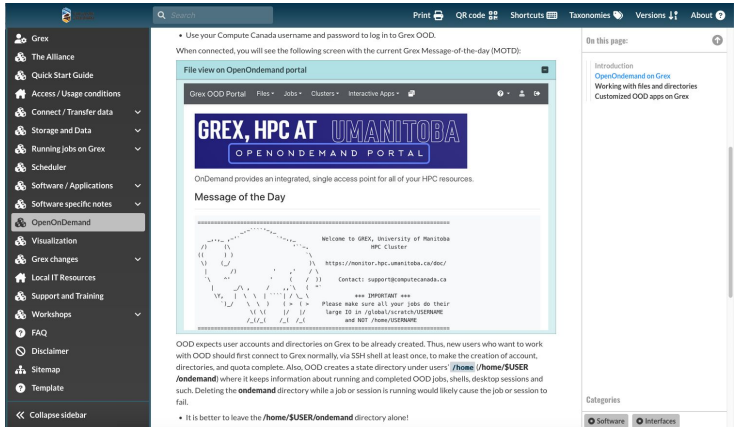
Grex documentation: new website

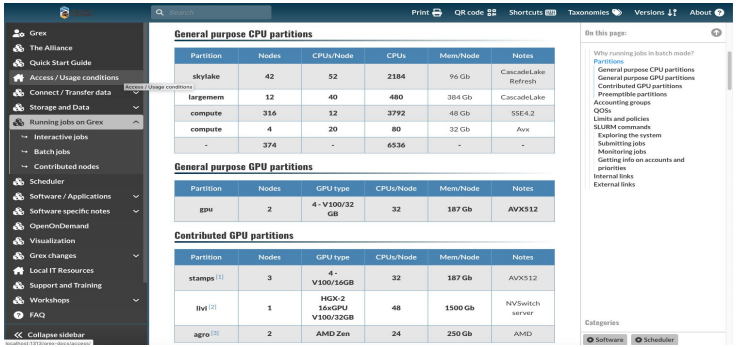


To replace the existing website

https://um-grex.github.io/grex-docs/

Online soon !





Useful links

- ★ FAQ: https://docs.computecanada.ca/wiki/Frequently_Asked_Questions
- ★ Jobs:
  - https://docs.computecanada.ca/wiki/Running_jobs
  - https://docs.computecanada.ca/wiki/Job_scheduling_policies#Percentage_of_the_nodes_you_have_access_to
  - https://docs.computecanada.ca/wiki/Advanced_MPI_scheduling#Whole_nodes
  - https://docs.computecanada.ca/wiki/Using_GPUs_with_Slurm
- ★ Storage:
  - https://docs.computecanada.ca/wiki/Storage_and_file_management#Filesystem_Quotas_and_Policies
  - https://docs.computecanada.ca/wiki/Project_layout?
  - https://docs.computecanada.ca/wiki/Transferring_data
- ★ Software:
  - https://docs.computecanada.ca/wiki/Available_software
  - https://docs.computecanada.ca/wiki/Utiliser_des_modules/en
  - https://docs.computecanada.ca/wiki/Installing_software_in_your_home_directory
  - https://docs.computecanada.ca/wiki/Python
  - https://docs.computecanada.ca/wiki/R

Upcoming WestGrid trainings

https://westgrid.github.io/trainingMaterials

https://rcmodules22.netlify.app/