

VERA Installation Guide

Author: Roscoe A. Bartlett (bartlettra@ornl.gov)
Author: Mark Baird (bairdml@ornl.gov)
Author: Joel A. Kulesza (kuleszj@westinghouse.com)

Contents

1	Introduction	1
2	Standard VERA Dev Env Directory Structure	1
3	Initial Setup	3
3.1	Requesting Access to VERA Repositories	3
3.2	System Configuration Considerations	3
3.3	SSH Setup For Accessing casl-dev	4
3.4	Minimal System Package Setup	4
3.5	Create Unix Groups	5
3.6	Local Directory Structure	5
3.7	Skeleton Directory Setup	5
3.8	Clone Minimal Bootstrap Repositories	6
4	Install Standard Basic VERA Dev Env	7
4.1	Standard Dev Env Script	7
4.2	Install eg, egdist And Perform git Setup	8
4.3	Install CMake From Source	9
4.4	Install git From Source	9
4.5	Install GCC From Source	10
4.6	Install OpenMPI From Source	10
4.7	Summary of VERA Dev Env Setup	11
5	Install Standard VERA TPLs	11
5.1	Set Up Base TPL Directory And Basic Variables	11
5.2	Install TPLs One At A Time	12
5.2.1	Install LAPACK	12
5.2.2	Install Boost	12
5.2.3	Install Zlib	12
5.2.4	Install MOAB	12
5.2.5	Install HDF5	13
5.2.6	Install Hypre	13
5.2.7	Install PETSC	13
5.2.8	Install SILO	14

5.2.9	Install QT	14
5.3	Install TPLs All At Once	14
6	VERA Component Build, Test, and Installation	15
6.1	Load VERA Dev Env	15
6.2	Clone Remaining VERA Components	15
6.3	Checking Out a Specific Version of VERA	16
6.4	Set Up Build Directory	18
6.5	Set Up do-configure Script	19
6.6	Run Basic Build And Test	19
7	Finalizing VERA Dev Env Installation	20
8	Installing VERA	21
8.1	Get Source For VERA Components To Install	21
8.2	Configure, Build, And Test VERA Components To Install	22
8.3	Install Built VERA Components	23
8.4	Documentation For Installed VERA Components	23
9	Appendix	23
9.1	Set Up Remote SSH Tunnel	23

1 Introduction

This guide describes the structure and setup of the standard VERA development environment (VERA Dev Env) and standard Third Party Libraries (TPLs) that need to be in place before installing many of the VERA simulation components. It describes everything from the initial setup on a new machine to the final builds and testing of VERA components. The goal of this document is to describe how to create the directories and contents outlined in [Standard VERA Dev Env Directory Structure](#), which once finished, allows one to clone the VERA git source repositories and build any of the necessary VERA components on a given system.

WARNING: This guide only describes the installation of the VERA Dev Env and TPLs and does not contain specific information about specific VERA simulation components. That information is found in other sources. Please consult with a CASL representative about what VERA components are available to install from source and what capabilities they provide. Mention of other VERA repositories in only used as examples and may not even be up to date.

2 Standard VERA Dev Env Directory Structure

The standard directory structure for the installation of the VERA Development Environment (VERA Dev Env) is given below:

```

${VERA_DEV_ENV_BASE}/
  common_tools/
    autoconf-2.69/
    cmake-2.8.5/

```

```

git-1.7.0.4/
eg
egidst
gcc-4.6.1/
  toolset/
    gcc-4.6.1/
    openmpi-1.4.3/
  tpls/
    opt/
      common/
        lapack-3.3.1/
        boost-1.49.0/
        zlib-1.2.5-patched/
        moab-4.5.0/
        hypre-2.8.0b/
        petsc-3.3-p4/
      vera_cs/
        hdf5-1.8.7/
        silo-4.8/
        qt-4.8.2/
      ...
    dbg/
      common/
      vera_cs/
      ...
    dbg-checkedstl/
      common/
      vera_cs/
      ...
  ...
intel-13.x/
  toolset/
    openmpi-1.4.3/
  tpls/
    opt/
      common/
      vera_cs/

```

(WARNING: The exact list of TPLs and versions are defined in the file VERA/cmake/std/vera-tpls-std.cmake. The primary purpose of the above list is to describe the standard format, not be the definitive definition per say.)

For the example in this guide, we will set:

```
VERA_DEV_ENV_BASE=/tools/vera
```

but note that any base directory can be used. This directory is where the prerequisite TPLs and VERA tools are deployed for use by end users.

For the standard GCC 4.6.1 VERA Dev Env, we set:

```
VERA_DEV_ENV_COMPILER_BASE=${VERA_DEV_ENV_BASE}/gcc-4.6.1
```

For other supported compilers (e.g. intel-13.x), other directories can be used. This directory structure keeps the compatible tools and TPLs together

to maintain consistency (i.e., so we don't mix TPL builds of one compiler with TPL builds for another compiler which can otherwise cause problems in some cases).

For a given compiler set, TPLs can be installed for different configurations, such as debug (dbg), optimized (opt), or other variations (e.g. dbg-checkedstl). The standard TPL install, and the one used in this guide as an example, is:

```
TPL_INSTALL_DIR=${VERA_DEV_ENV_COMPILER_BASE}/tpls/opt
```

All of the VERA Dev Env and TPL related scripts, etc. will use the variables:

- VERA_DEV_ENV_BASE
- VERA_DEV_ENV_COMPILER_BASE
- TPL_INSTALL_DIR

to determine a particular installation of the VERA Dev Env to use for building/testing/installing VERA components.

3 Initial Setup

Before any of the VERA-specific prerequisites can be installed, a some initial setup is required. This section describes some of the tasks that need to be performed in order to set up a machine so that it can be used to install the VERA Dev Env and then clone the VERA repositories and build the various VERA components from source.

3.1 Requesting Access to VERA Repositories

Before one can access the various VERA git repositories on casl-dev.ornl.gov, one must first be given an ORNL UCAMS account, given an account on casl-dev, and be given explicit access to the different git repositories (in accordance to the CASL Technology Control Plan (TCP)). Contact your CASL representative to get started getting this access setup. Please provide them with a list of the specific VERA git repositories that one needs access to.

3.2 System Configuration Considerations

Before work can begin, an accounting of the system resources should be made. For example:

- How many processors are available for parallel compilation?
- What is the fastest storage system to use for holding source code, compiling, and running test cases?
- Avoid NFS-mounted directories for compilation and testing.

Meanwhile, other considerations include:

- Can the machine create SSH tunnels to download VERA and TPL repositories?

- Can the machine access the Internet? If not, ensure that all packages needed are available in the VERA and/or TPL repositories.
- Can remote users access the machine if troubleshooting assistance is needed?

3.3 SSH Setup For Accessing casl-dev

In order to access the VERA repositories on `casl-dev.ornl.gov`, one must set up public/private SSH keys and then the public SSH must be registered with the gitolite system that is used to manage the VERA git repositories. In this guide, we use `<ucams-id>` to signify the user's 3-char ORNL UCAMS ID.

If the machine `casl-dev` (`casl-dev.ornl.gov`) is not directly reachable from your machine (referred to as `<your-machine>` in this document), you will first need to set up a remote SSH tunnel to `casl-dev` as described in [Set Up Remote SSH Tunnel](#).

First, on the given machine, one sets up public/private SSH keys (if not already existing) as:

```
$ cd ~/.ssh && /usr/bin/ssh-keygen -t rsa -b 1024
```

Several prompts will appear. The defaults should be accepted with three strikes of the `<ENTER>` key.

The public key just created, `~/.ssh/id_rsa.pub`, must then be sent to casl-vri-infrastructure@casl.gov in order to be registered with your account in the gitolite system on `casl-dev`.

After one's public SSH key has been registered with gitolite on `casl-dev`, one can test to see if one has access by performing:

```
$ ssh git@casl-dev info
```

This command should return the list of git repositories that one has access to. At the very minimum, this should return:

```
$ ssh git@casl-dev info

hello <userid>, this is git@casl-dev running ...

...
R    Trilinos
...

```

Access to other repositories requires being explicitly added to the appropriate gitolite permission groups.

If this command does not work without a password challenge, then something is wrong and no repositories will be able to be cloned.

3.4 Minimal System Package Setup

Before proceeding to start installing the various VERA prerequisites, the following software packages must be installed on the system:

- GCC compilers (does not have to be the most recent version): Needed to build the official version of GCC built from source

- Git: Needed to clone several git repositories
- X11 (development libraries, not just the runtime libraries): Needed to build and install the required VERA TPL QT
- Perl (version v5.10.1 is what VERA is tested with but newer should work as well): Needed for building libmesh/MOOSE/Peregrine and to run the react2xml.pl perl parser.

There are many other software packages that one needs but the rest should already be installed on any reasonable Linux/Unix system.

3.5 Create Unix Groups

In order to properly protect VERA installs, it is recommended to set up the following groups on the build and installation machines:

- **vera-admin**: List of Unix users that are charged with maintaining the installs of VERA
- **vera-users**: List of Unix users that have access permission and need-to-know for running the VERA components.

NOTE: The list of users added to **vera-users** must have been given explicit permission to access **all** of the installed VERA components. If one is not sure who can be in this list, please contact casl-vera-users@casl.gov or casl-vri-infrastructure@casl.gov or some other responsible CASL representative for guidance.

3.6 Local Directory Structure

The installation of the VERA development environment requires setting up some local directories, including scratch space, and then cloning some git repositories. In this guide, we will assume the following local directory structure:

```

<SOME-BASE-DIR>/
  VERA.base/          # VERA repos and local builds
  VERA/              # Base VERA git repos
  BUILDS/            # Base builds directory
    GCC-4.6.1/      # Standard GCC 4.6.1 builds
      MPI_DEBUG/
        do-configure
  scratch/          # Contains temp build files, log files, etc.
  vera_tpls/        # VERA TPLs and install scripts

```

For example, <SOME-BASE-DIR> could just be the home directory ~/ for the person who is doing the install of the VERA Dev Env.

If the VERA source

3.7 Skeleton Directory Setup

In order to create the skeleton for the base directory setup, first, one must create a local **scratch** directory:

```
$ mkdir <SOME-BASE-DIR>/scratch
```

Then, one must create the skeleton for the base VERA Dev Env install directory `$VERA_DEV_ENV_BASE`. For example, one could use:

```
/tools/vera
```

To set up `/tools/vera`, for example, (owned by group `<some-group>`, e.g. `vera-admin`) one would do:

```
$ sudo mkdir /tools
$ sudo chmod 775 /tools
$ sudo chgrp vera-admin /tools
$ sudo chmod g+s /tools
```

Now that `/tools` is owned by the `<some-group>` group (e.g. `vera-admin`), anyone in that group can maintain these files, reinstall, etc. without needing `sudo` on the system. Note that if one's `umask` is set to `0022` when installing the tools (the default on many Linux systems), then as files and directories are installed under `/tools/vera`, the the permissions will be set correctly for usage by anyone on the system. NOTE: There are no sensitive or export controlled files or data that get installed under `$VERA_DEV_ENV_BASE/` described in this document. If sensitive files get installed, they will need to be protected with an appropriate Unix group.

The rest of the skeleton of the directory structure for the VERA Dev Env `$VERA_DEV_ENV_BASE` is done with:

```
$ mkdir /tools/vera
$ mkdir /tools/vera/common_tools
$ mkdir /tools/vera/gcc-4.6.1
```

All of the other directories should be created automatically during the various installs processes as described below.

3.8 Clone Minimal Bootstrap Repositories

Before the basic VERA Dev Env can be built and installed, a few of the VERA git repos must be cloned (if building from the version controlled source) and the source and install scripts for the official VERA TPLs `vera_tpls` must be obtained.

When cloning the VERA git repos, consistent with the directory structure in [Local directory structure](#), one first performs:

```
$ cd <SOME-BASE-DIR>
$ mkdir VERA.base
$ chgrp vera-admin VERA.base
$ chmod 750 VERA.base
$ chmod g+s VERA.base
$ cd VERA.base/
$ git clone git@casl-dev:/VERA
$ cd VERA/
$ git clone git@casl-dev:/Trilinos
```

When the sources are obtained from a (release) tarball of VERA, one must just untar the single tarball `vera-X.Y-Source.tar.gz` (where `X.Y = 3.2, 3.3, or 3.4` for example) as:

```
$ cd <SOME-BASE-DIR>
$ mkdir VERA.base
$ cd VERA.base/
$ tar -xzf ~/vera-X.Y-Source.tar.gz
$ ln -s vera-X.Y-Source VERA
```

By creating the symbolic link `VERA` for the untarred sources, the rest of the instructions remain the same as for the cloned git repos. However, one can choose to not create the symbolic link and just explicitly use `vera-X.Y-Source` instead of `VERA` in these instructions but other adjustments may be needed as well.

The various open-source third party libraries are stored in a few git repos that are have clones under both `git@casl-dev:/prerequisites` and on `github.com/CASL`. Only those with UCAMS accounts can access the `casl-dev` repos using an SSH tunnelling but anyone can access the `github.com/CASL` repos if their system allows https downloads from `github.com` (and some systems don't). When referring to these open-source git repos, this document will assume the following variable is set in the shell env:

```
VERA_PREREQUISITES_GIT_URL_BASE
```

to either:

```
VERA_PREREQUISITES_GIT_URL_BASE=git@casl-dev:/prerequisites
```

or:

```
VERA_PREREQUISITES_GIT_URL_BASE=https://github.com/CASL
```

Now that the discussion of the different locations for the open-source git repos is out of the way, one must clone the `vera_tpls` git repo containing the source code and install scripts for all of the official VERA open-source TPLs:

```
$ cd <SOME-BASE-DIR>
$ git clone $VERA_PREREQUISITES_GIT_URL_BASE/vera_tpls
```

With this, one should have access to all the scripts and source code need to get started installing the VERA Dev Env as well as the TPLs. A few other git repos will be cloned for GCC, OpenMPI, etc. as part of the following process.

4 Install Standard Basic VERA Dev Env

In this section, the installation of the standard basic VERA Dev Env is described which includes the GCC compilers, OpenMPI, CMake, and some supporting scripts (but not the TPLs yet). This includes everything shown in [Standard VERA Dev Env Directory Structure](#) except the `tpl` subdirectory.

4.1 Standard Dev Env Script

Before starting the install of the basic VERA Dev Env tools, one should set up and source a standard `load_dev_env.sh` script. A standard script for the standard VERA Dev Env looks like:

```
# Set the base dir to anything you want but default is given
if [ "$VERA_DEV_ENV_BASE" == "" ] ; then
    export VERA_DEV_ENV_BASE=/projects/vera
fi

# A) Common tools independent of the compiler
export PATH=${VERA_DEV_ENV_BASE}/common_tools:$PATH
export PATH=${VERA_DEV_ENV_BASE}/common_tools/cmake-2.8.5/bin:$PATH
export PATH=${VERA_DEV_ENV_BASE}/common_tools/git-1.7.0.4/bin:$PATH

# B) The GCC-4.6.1 compiler stack
export VERA_DEV_ENV_COMPILER_BASE=${VERA_DEV_ENV_BASE}/gcc-4.6.1

# B.1) GCC 4.6.1
export VERA_GCC_DIR=${VERA_DEV_ENV_COMPILER_BASE}/toolset/gcc-4.6.1
export PATH=${VERA_GCC_DIR}/bin:$PATH
export LD_LIBRARY_PATH=${VERA_GCC_DIR}/lib64:$LD_LIBRARY_PATH

# B.2) OpenMPI 1.4.3
export VERA_OPENMPI_DIR=${VERA_DEV_ENV_COMPILER_BASE}/toolset/openmpi-1.4.3
export PATH=${VERA_OPENMPI_DIR}/bin:$PATH
export LD_LIBRARY_PATH=${VERA_OPENMPI_DIR}/lib:$LD_LIBRARY_PATH
```

For the standard VERA Dev Env, a standard `load_dev_env.sh` is found at:

```
<SOME-BASE-DIR>/VERA.base/VERA/cmake/std/gcc-4.6.1/load_dev_env.sh
```

To use this standard `load_dev_env.sh` script, one just needs to set, for example:

```
$ export VERA_DEV_ENV_BASE=/tools/vera
```

and then do:

```
$ . <SOME-BASE-DIR>/VERA.base/VERA/cmake/std/gcc-4.6.1/load_dev_env.sh
```

With these paths set, as tools are installed, they will automatically show up and be used!

To make this happen automatically, one could add the lines:

```
export VERA_DEV_ENV_BASE=/tools/vera # Or whatever it is on the system
. <SOME-BASE-DIR>/VERA.base/VERA/cmake/std/gcc-4.6.1/load_dev_env.sh
```

to the end of their `.bash_profile` file so the VERA Dev Env will be set on login.

NOTE: The remaining instructions assume that the above `load_dev_env.sh` script has been sourced before installing any of the VERA prerequisites.

4.2 Install eg, egdist And Perform git Setup

To install eg and git, do:

```
$ cd <SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits/common_tools/git/
$ cp eg egdist ${VERA_DEV_ENV_BASE}/common_tools/

$ which eg
.../common_tools/eg

$ which egdist
.../common_tools/egdist

$ eg --version
eg version 1.7.0.4
git version xxxxxx
```

(where `.../` above is meant to signify the actual directory base path).

If one will be making commits or using git for anything interesting, they should set up their `~/.gitconfig` file. An example of how to do this, for Trilinos developer `bartlettra`, is:

```
$ <SOME-BASE-DIR>/VERA.base/VERA/Trilinos
./sampleScripts/git-profiles/git-config-bartlettra.sh
```

(NOTE: the above script requires `eg`). One can just copy the file `git-config-bartlettra.sh` as a template and modify it for one's own use.

4.3 Install CMake From Source

To install an optimized version of CMake from source do:

```
$ cd <SOME-BASE-DIR>/scratch

$ env CXXFLAGS=-O3 CFLAGS=-O3 \
  <SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits/python/install-cmake.py \
  --checkout-cmnd="git clone $VERA_PREREQUISITES_GIT_URL_BASE/cmake.BASE" \
  --parallel=8 \
  --install-dir=${VERA_DEV_ENV_BASE}/common_tools/cmake-2.8.5 \
  --do-all \
  &> install-cmake.log
```

After this, `cmake` should up automatically be in the path (see [Standard Dev Env Script](#)) as seen by:

```
$ which cmake
.../common_tools/cmake-2.8.5/bin/cmake
```

4.4 Install git From Source

While one needs some version of git to clone the initial VERA repositories, it is best to install the official version of git so as to ensure that the various git-related code in TriBITS will work correctly.

To install Git from source do:

```

$ cd <SOME-BASE-DIR>/scratch

$ <SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits/python/install-git.py \
  --checkout-cmd="git clone $VERA_PREREQUISITES_GIT_URL_BASE/git.BASE" \
  --make-options=-j8 \
  --install-dir=${VERA_DEV_ENV_BASE}/common_tools/git-1.7.0.4 \
  --do-all \
  &> install-git.log

```

After this, this version of `git` should automatically be in the path (see [Standard Dev Env Script](#)) as seen by:

```

$ which git
.../common_tools/git-1.7.0.4/bin/cmake

$ eg --version
eg version 1.7.0.4
git version 1.7.0.4

```

4.5 Install GCC From Source

To install GCC 4.6.1 from source, do:

```

$ cd <SOME-BASE-DIR>/scratch

$ <SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits/python/install-gcc.py \
  --checkout-cmd="git clone $VERA_PREREQUISITES_GIT_URL_BASE/gcc.BASE" \
  --install-dir=${VERA_DEV_ENV_COMPILER_BASE}/toolset/gcc-4.6.1 \
  --make-options=-j8 \
  --do-all \
  &> install-gcc.log

```

Once the `install-gcc.py` script finishes (without error), the new GCC should automatically show up (see [Standard Dev Env Script](#)) as seen by:

```

$ which gcc
.../gcc-4.6.1/toolset/gcc-4.6.1/bin/gcc

$ gcc --version
gcc (GCC) 4.6.1
...

```

4.6 Install OpenMPI From Source

To install OpenMPI 1.4.3 from source do:

```

$ cd <SOME-BASE-DIR>/scratch

$ <SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits/python/install-openmpi.py \
  --checkout-cmd="git clone $VERA_PREREQUISITES_GIT_URL_BASE/openmpi.BASE" \
  --make-options=-j8 \
  --install-dir=${VERA_DEV_ENV_COMPILER_BASE}/toolset/openmpi-1.4.3 \

```

```
--do-all \  
&> install-openmpi.log
```

(NOTE: This will use the GCC 4.6.1 compiler just installed above. This is critical!)

Once the `install-openmpi.py` script finishes (without error), the new Open-MPI executables should automatically show up (see [Standard Dev Env Script](#)) as seen by:

```
$ which mpicc  
.../gcc-4.6.1/toolset/openmpi-1.4.3/bin/mpicc
```

4.7 Summary of VERA Dev Env Setup

After the above installation steps are complete, one should log off then log back in (sourcing `.bash_profile` which will source the `load_dev_env.sh` script) where then one should see:

```
$ which eg  
.../common_tools/eg  
$ which egdist  
.../common_tools/egdist  
$ which git  
.../common_tools/git-1.7.0.4/bin/cmake  
$ which cmake  
.../common_tools/cmake-2.8.5/bin/cmake  
$ which gcc  
.../gcc-4.6.1/toolset/gcc-4.6.1/bin/gcc  
$ which mpicc  
.../gcc-4.6.1/toolset/openmpi-1.4.3/bin/mpicc
```

which confirms that the basic VERA Dev Env (minus the TPLs) is fully set up and ready to install TPLs.

5 Install Standard VERA TPLs

After finishing [Install Standard Basic VERA Dev Env](#), one is ready to start building and installing the standard VERA TPLs from source.

WARNING: It is critical that the TPLs be built using the compilers and MPI implementation installed in [Install Standard Basic VERA Dev Env](#) and that will also be used to build the VERA components.

5.1 Set Up Base TPL Directory And Basic Variables

Before installing VERA TPLs using the scripts in the cloned `vera_tpls` repo, one needs to set up a few variables that are used in the TPL install scripts. For a standard install of VERA, for example, one would set:

```
export TPL_INSTALL_DIR=${VERA_DEV_ENV_BASE}/gcc-4.6.1/tpls/opt  
export MAKE_FLAGS=-j8  
export METHOD=opt
```

NOTE: One could install a 'dbg' version of the libs as well by setting METHOD=dbg.

5.2 Install TPLs One At A Time

In this section, the installation of the individual TPLs one at a time is described. This mode is to be preferred if one wants to carefully validate the correct install and inspect the results along the way.

NOTE: When building individual TPLs, be mindful of the TPL dependencies as the TPLs must be built in the right order. To see what these dependencies are, just inspect the scripts. Also note that the size of the tpl sub-directories will vary slightly.:

```
<SOME-BASE-DIR>/vera_tpls/scripts/std/*.sh
```

5.2.1 Install LAPACK

```
cd <SOME-BASE-DIR>/scratch
env BUILD_LAPACK=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> lapack.log
```

```
ls ${TPL_INSTALL_DIR}/common/lapack-3.3.1/lib/
libblas.a liblapack.a libtmplib.a
```

5.2.2 Install Boost

```
cd <SOME-BASE-DIR>/scratch
env BUILD_BOOST=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> boost.log
```

```
du -sh ${TPL_INSTALL_DIR}/common/boost-1.49.0/*
103M  ../common/boost-1.49.0/include
34M   ../common/boost-1.49.0/lib
```

5.2.3 Install Zlib

```
env BUILD_ZLIB=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> zlib.log
```

```
du -sh ${TPL_INSTALL_DIR}/common/zlib-1.2.5-patched/*
96K   ../common/zlib-1.2.5-patched/include
156K  ../common/zlib-1.2.5-patched/lib
8.0K  ../common/zlib-1.2.5-patched/share
```

5.2.4 Install MOAB

```
env BUILD_MOAB=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
```

```
&> moab.log
```

```
du -sh ${TPL_INSTALL_DIR}/common/moab-4.5.0/*
476K  .../common/moab-4.5.0/bin
968K  .../common/moab-4.5.0/include
6.5M  .../common/moab-4.5.0/lib
856K  .../common/moab-4.5.0/share
```

5.2.5 Install HDF5

```
env BUILD_HDF5=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> hdf5.log
```

```
du -sh ${TPL_INSTALL_DIR}/vera_cs/hdf5-1.8.7/*
1.5M  .../vera_cs/hdf5-1.8.7/bin
8.1M  .../vera_cs/hdf5-1.8.7/include
11M   .../vera_cs/hdf5-1.8.7/lib
1.2M  .../vera_cs/hdf5-1.8.7/share
```

5.2.6 Install Hypre

```
env BUILD_HYPRE=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> hypre.log
```

```
du -sh ${TPL_INSTALL_DIR}/common/hypre-2.8.0b/*
1.3M  .../common/hypre-2.8.0b/include
16M   .../common/hypre-2.8.0b/lib
```

5.2.7 Install PETSC

```
env BUILD_PETSC=1 \
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \
  &> petsc.log
```

```
du -sh ${TPL_INSTALL_DIR}/common/petsc-3.3-p4/*
3.6M  .../common/petsc-3.3-p4/bin
3.3M  .../common/petsc-3.3-p4/conf
33M   .../common/petsc-3.3-p4/include
17M   .../common/petsc-3.3-p4/lib
220K  .../common/petsc-3.3-p4/share
```

NOTE: To reinstall PETSC, one must first delete (or move) the existing PETSC install using:

```
rm -rf ${TPL_INSTALL_DIR}/common/petsc-3.3-p4
```

The PETSC install target will not copy over an existing PETSC install.

5.2.8 Install SILO

```
env BUILD_SILO=1 \  
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \  
  &> silo.log  
  
du -sh ${TPL_INSTALL_DIR}/vera_cs/silo-4.8/*  
2.1M  .../vera_cs/silo-4.8/bin  
164K  .../vera_cs/silo-4.8/include  
2.0M  .../vera_cs/silo-4.8/lib
```

5.2.9 Install QT

```
env BUILD_QT=1 \  
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \  
  &> qt.log  
  
du -sh ${TPL_INSTALL_DIR}/vera_cs/qt-4.8.2/*  
230M  .../vera_cs/qt-4.8.2/bin  
345M  .../vera_cs/qt-4.8.2/demos  
549M  .../vera_cs/qt-4.8.2/doc  
3.6G  .../vera_cs/qt-4.8.2/examples  
448K  .../vera_cs/qt-4.8.2/imports  
17M   .../vera_cs/qt-4.8.2/include  
106M  .../vera_cs/qt-4.8.2/lib  
5.1M  .../vera_cs/qt-4.8.2/mkspecs  
320K  .../vera_cs/qt-4.8.2/phrasebooks  
4.0M  .../vera_cs/qt-4.8.2/plugins  
328K  .../vera_cs/qt-4.8.2/q3porting.xml  
7.2M  .../vera_cs/qt-4.8.2/translations
```

5.3 Install TPLs All At Once

The standard VERA TPLs can be installed all at once by doing:

```
$ cd <SOME-BASE-DIR>/scratch  
  
$ env BUILD_ALL=1 \  
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \  
  &> all.log
```

or in groups of TPLs as:

```
$ cd <SOME-BASE-DIR>/scratch  
  
$ env BUILD_LAPACK=1 BUILD_BOOST=1 ... BUILD_PETSC=1 \  
  <SOME-BASE-DIR>/vera_tpls/scripts/std/install_tpls.sh \  
  &> some.log
```

WARNING: When installing multiple TPLs at the same time, if a TPL install fails, the script should stop but one will have to carefully look at the output to see what TPL install failed and then to address the issue for that TPL before installing the other TPLs, in the right order.

6 VERA Component Build, Test, and Installation

Once the VERA prerequisites (i.e., compilers, TPLs, other tools) have been installed, if working from the version-controlled source, one needs to clone the remaining VERA git repositories for the desired components, set up a build configuration, build, test, and finally install. If working from an untarred (release) tarball, no extra git clones are needed. All of the required source will already be in place.

6.1 Load VERA Dev Env

Before one can configure, build, test, and install any VERA component software, the installed VERA Dev Env must first be loaded into the users shell. If the VERA Dev Env was loaded in the current shell as part of installing the VERA Dev Env as described in [Standard Dev Env Script](#), then the environment should already be loaded. Otherwise, if the VERA Dev Env installation was already finalized, as described in [Finalizing VERA Dev Env Installation](#), then the `load_dev_env.sh` script has already been installed and one just needs to source it, for example, as:

```
. /tools/vera/gcc-4.6.1/load_dev_env.sh
```

(or whatever base `VERA_DEV_ENV_BASE` directory was used for the install).

IMPORTANT: One should check that the correct VERA Dev Env is loaded as described in [Summary of VERA Dev Env Setup](#) to make sure the right tools are in the default path.

6.2 Clone Remaining VERA Components

If working from the version-controlled sources, the remaining VERA git repositories can be cloned, for example, as:

```
cd <SOME-BASE-DIR>/VERA.base/VERA
git clone git@casl-dev:/Dakota Trilinos/packages/TriKota/Dakota
git clone git@casl-dev:/TeuchosWrappersExt
git clone git@casl-dev:/VERAInExt
git clone git@casl-dev:/DataTransferKit
git clone git@casl-dev:/COBRA-TF
git clone git@casl-dev:/SCALE
git clone git@casl-dev:/Exnihilo SCALE/Exnihilo
git clone git@casl-dev:/MPACT
git clone git@casl-dev:/LIMEExt
git clone git@casl-dev:/PSSDriversExt
```

To see the exact repo names and the repo URLs, see:

```
VERA/cmake/ExtraRepositoriesList.cmake
```

NOTE: The exact list of repositories that one needs to clone greatly depends on what VERA components with what functionality one desires or needs from VERA. Such information is not provided in this document.

NOTE: One must be part of the gitolite group that protects a repository or when one clones one will get an error message like:

```
$ git clone git@casl-dev:MPACT
Initialized empty Git repository in <some-base-dir>/MPACT/.git/
FATAL: R any MPACT <userid> DENIED by fallthru
(or you mis-spelled the reponame)
fatal: The remote end hung up unexpectedly
```

where `<some-base-dir>` and `<userid>` are replaced with the local base directory and the gitolite user account name on `casl-dev`. To see if one has misspelled the repo or if one just does not have permission, one should run:

```
ssh git@casl-dev info
```

If the git repository that one is trying to clone is not listed in the output from this command, then one don't have permissions to clone the given git repository.

After cloning the git repositories, a `.egdist` file should be created to facilitate updating the VERA sources for future installs. For the above set of VERA git repos, this would be done by creating the `.egdist` file:

```
$ cd <SOME-BASE-DIR>/VERA.base/VERA
$ cat .egdist
Trilinos/packages/TriKota/Dakota
TeuchosWrappersExt
VERAInExt
DataTransferKit
COBRA-TF
SCALE
SCALE/Exnihilo
MPACT
LIMEExt
PSSDriversExt
```

See `egdist --help` for more details. (Note that `egidst` should have been installed as described in [Install eg, egdist And Perform git Setup](#)).

6.3 Checking Out a Specific Version of VERA

The most recent version of VERA can be pulled just using:

```
$ cd <SOME-BASE-DIR>/VERA.base/VERA
$ egdist pull
```

This will pull the most recent version of VERA (at that moment) from the official development `casl-dev/master` branches. While a rigorous almost continuous integration process ensures that all of the basic automated tests pass before anything is pushed into the `casl-dev/master` branches, mistakes do occur and there may be some more detailed acceptance tests that may not run successfully on any particular version of VERA at any moment in `casl-dev/master`. To reduce the probability of the customer pulling a defective version of VERA components for their usage, it is recommended that more specific versions of VERA be pulled that have undergone more testing (both more expensive automated acceptance tests run nightly and weekly as well as some larger manually run tests in some cases). This is accomplished using the `egdist` tool and a

VERARepoVersion.txt file. A VERARepoVersion.txt file is created whenever VERA is configured from local git repositories and that file is written to the VERA build tree and it gets installed in the base install tree. Every automated build (which includes the tests) posted to the VERA CDash server includes the information in a VERARepoVersion.txt file. For a subset of VERA repos, a VERARepoVersion.txt file looks like:

```
*** Base Git Repo: VERA
f773368 [Wed Sep 25 10:52:06 2013 -0400] <briadam@sandia.gov>
Final cleanup of Dakota sync script for now.
** Git Repo: Trilinos
209ac0f [Wed Sep 25 12:36:06 2013 -0400] <bartlettra@ornl.gov>
Added add support for --dist-version-file with unit tests (VRI Kanban #3006)
** Git Repo: TeuchosWrappersExt
4b18529 [Fri Aug 30 09:55:07 2013 -0400] <bartlettra@ornl.gov>
Added read and broadcast of PL from XML file (VRI Kanban #3062)
** Git Repo: Trilinos/packages/TriKota/Dakota
0b8d9ca [Thu Aug 8 06:04:53 2013 -0600] <dakota-developers@development.sandia.gov>
Sync from DAKOTA stable dated 20130808
** Git Repo: VERAInExt
b4f8c16 [Tue Sep 24 21:04:07 2013 -0400] <rks171@gmail.com>
Reblessing the xml gold file for the input file I added CTF convergence criteria
** Git Repo: DataTransferKit
9e80d98 [Mon Jul 15 10:51:55 2013 -0400] <uy7@ornl.gov>
Merge branch 'github-dev'
** Git Repo: COBRA-TF
ead9db0 [Wed Sep 25 20:07:35 2013 -0400] <sankaranr@ornl.gov>
Adding a clad_tmp_sect variable for transferring sector wise clad temperatures
** Git Repo: SCALE
6dab14f [Tue Sep 24 20:57:25 2013 -0400] <clarnokt@ornl.gov>
changeset: 9677:7cb0277be232 tag: tip user: Doro Wiarda <dw8@or
** Git Repo: SCALE/Exnihilo
02a43bf [Tue Sep 24 20:50:04 2013 -0400] <clarnokt@ornl.gov>
Merge remote branch 'angmar/master'
** Git Repo: MPACT
3676968 [Mon Sep 23 00:18:27 2013 -0400] <sgstim@umich.edu>
updated the two-node NEM/SANM sweeper
** Git Repo: MOOSEExt
dfa90ec [Fri Sep 13 13:47:38 2013 -0400] <bartlettra@ornl.gov>
Protecting test correctly with if statement.
** Git Repo: MOOSEExt/MOOSE
f3d3b42 [Wed Jul 31 17:22:49 2013 -0400] <bartlettra@ornl.gov>
Adding missing std:: (Trilinos #5945)
** Git Repo: LIMEExt
f9d8bcc [Thu Sep 19 00:11:42 2013 -0400] <rhoope@sandia.gov>
Revert "Update LIME dependence on TriKota. This is an odd way to do it, but cur
** Git Repo: PSSDriversExt
35d59b0 [Wed Sep 25 20:12:35 2013 -0400] <sankaranr@ornl.gov>
Use the clad temperature instead of the coolant temperature as the BC for Mamba
```

Before updating to a specific version of VERA, a CASL representative will provide the customer a specific VERARepoVersion.txt file, for example VERARepoVersion.<newdate>.txt, which the customer can use to checkout that specific version with:

```
$ cd <SOME-BASE-DIR>/VERA.base/VERA
$ egdist fetch
$ egdist --dist-version-file=~/.VERARepoVersion.<newdate>.txt \
  checkout _VERSION_
```

(see `egdist --help` for more details.)

This will create a “detached head” state for the local VERA git repos where each repo will be at the exact commit listed in the VERARepoVersion.<newdate>.txt file. Here is the message that you might get from each of the repos:

```
Note: checking out '00149f1'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
```

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b new_branch_name
```

```
HEAD is now at 00149f1... Merge remote branch 'origin/master' into rsicc_2013_tarball
```

This is not a troublesome state for the purposes of just building the source.

Also, using the VERARepoVersion.txt file for a previous install, one can see what repos have been changed and what commits have been added. For example, to compare to an older install in:

```
${VERA_DEV_ENV_BASE}/vera/<olddate>/
```

one could compare to the new recommended version by running:

```
$ egdist fetch
$ egdist \
  --dist-version-file=~/.VERARepoVersion.<newdate>.txt \
  --dist-version-file2=${VERA_DEV_ENV_BASE}/vera/<olddate>/VERARepoVersion.txt \
  diff _VERSION_ ^_VERSION2_
```

Many other types of git commands are possible where one or two of the repo versions can be supplied through a VERARepoVersion.txt file.

NOTE: If working from an untarred (release) tarball, no extra clones are necessary.

6.4 Set Up Build Directory

Once all of the VERA git repositories have been cloned (or are already there in the untarred source), one can set up a build directory to configure and build the code in. According to [Local directory structure](#), the build directory can be set up as:

```

$ cd <SOME-BASE-DIR>/VERA.base
$ mkdir BUILDS
$ cd BUILDS
$ mkdir GCC-4.6.1
$ cd GCC-4.6.1
$ mkdir MPI_RELEASE
$ cd MPI_RELEASE

```

6.5 Set Up do-configure Script

To get started setting up a configure script, one can copy a default configure script for the standard gcc-4.6.1 configuration. For example, one can grab a standard configuration using:

```

$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE
$ ln -s ../../../../VERA/cmake/std/gcc-4.6.1/do-configure.MPI_RELEASE_SHARED \
    do-configure

```

6.6 Run Basic Build And Test

Using the do-configure script created above, configure, build, and test with, for example:

```

$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE
$ ./do-configure \
  -DVERA_ENABLE_CTeuchos=ON \
  -DVERA_ENABLE_ForTeuchos=ON \
  -DVERA_ENABLE_VERAIn=ON \
  -DVERA_ENABLE_DataTransferKit=ON \
  -DVERA_ENABLE_MPACT_libs=ON \
  -DVERA_ENABLE_Insilico=ON \
  -DVERA_ENABLE_COBRA_TF=ON \
  -DVERA_ENABLE_VRIPSS=ON \
  &> configure.out
$ make -j8 &> make.out
$ ctest -j8 &> ctest.out

```

NOTE: This will enabled many VERA packages but the tests and examples for only the explicitly enabled packages shown above will be turned on. For more details, see the [VERA Build Quick Reference](#).

NOTE: If not configuring from scratch, it is advisable to first delete the existing CMakeCache.txt file and CMakeFiles/ directory first as:

```

$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE
$ rm -r CMake*
$ ./do-configure [other options]

```

This will take care if problems with changing cache variables that are set to the default and other issues like updates to the TPL locations, compilers, and

about anything else. However, during a repeated cycle of having to reconfigure for simple updates to CMakeLists.txt files, one does not necessarily want to always want to completely configure from scratch to speed up the configuration process.

7 Finalizing VERA Dev Env Installation

Once one has finished installing the VERA prerequisites for the compilers, Open-MPI, and other tools and a complete set of TPLs shown in [Standard VERA Dev Env Directory Structure](#) and one is finished testing the installs by building and testing needed VERA components, one just needs to install the VERA Dev Env `load_dev_env.sh` script and do some final cleanup.

First, one can copy the default `load_dev_env.sh` script to `${VERA_DEV_ENV_BASE}/` and hard-code the default path as:

```
$ cd <SOME-BASE-DIR>/VERA.base/VERA/cmake/std/gcc-4.6.1/
$ cp load_dev_env.sh /tools/vera/gcc-4.6.1/
$ emacs -nw /tools/vera/gcc-4.6.1/load_dev_env.sh
```

and replace:

```
if [ "$VERA_DEV_ENV_BASE" == "" ] ; then
    export VERA_DEV_ENV_BASE=<something>
fi
```

with just:

```
export VERA_DEV_ENV_BASE=/tools/vera
```

(or whatever directory was used for `${VERA_DEV_ENV_BASE}` for this install).

Next, after all of the installs under `${VERA_DEV_ENV_BASE}` are finished, then the directory permissions should be opened up for maintenance by `<some-group>` (e.g. `vera-admin`) and opened up for all to use the generic tools:

```
$ export TRIBITS_DIR=<SOME-BASE-DIR>/VERA.base/VERA/Trilinos/cmake/tribits
$ ${TRIBITS_DIR}/common_tools/setup/setup-shared-dir.sh \
  ${VERA_DEV_ENV_BASE}/common_tools <some-group>
$ ${TRIBITS_DIR}/common_tools/setup/setup-shared-dir.sh \
  ${VERA_DEV_ENV_BASE}/gcc-4.6.1 <some-group>
```

That should allow anyone in `<some-group>` to read/write any of the installed files and directories but let anyone on the system read the installed files (but not modify them and mess them up).

WARNING: This will only change the owning group. This will *not* change the owning user. Most Linux systems will not allow a regular user to change the owning user. To change the owning user to avoid accidental modifications by the original installer, run the following command:

```
$ sudo chown -R <some-other-user> <some-dir>
```

for example:

```
$ sudo chown -R root ${VERA_DEV_ENV_BASE}/common_tools
$ sudo chown -R root ${VERA_DEV_ENV_BASE}/gcc-4.6.1
```

This will avoid problems with accidental modifications to the installed directories without sudo, for example.

As an optional final step, to clean up disk space, one can delete the scratch space and TPLs source repo by doing:

```
$ cd <SOME-BASE-DIR>
$ rm -rf vera_tpls
$ rm -rf scratch
```

WARNING: One should only remove these directories after one is sure that the VERA dev env is correctly installed and that the necessary dependent VERA components are building and running correctly (at least related to the installed VERA dev env).

All that should be left locally would be the local VERA source and build tree:

```
<SOME-BASE-DIR>/VERA.base
```

which can be used to clone and build VERA components using the installed VERA Dev Env. However, if VERA will no longer be built under this directory, then it can be removed as well with:

```
$ cd <SOME-BASE-DIR>
$ rm -rf VERA.base
```

After this, all that would left would be the install of the VERA dev env under `${VERA_DEV_ENV_BASE}/`.

After all of the the VERA Dev Env would be considered successfully installed and now users of the dev env just need to source the script, for example:

```
. /tools/vera/gcc-4.6.1/load_dev_env.sh
```

in their `.bash_profile` file and they will not have to have the VERA git repo cloned to access the standard version under version control.

8 Installing VERA

Once the VERA Dev Env is installed on a system (see [Install Standard Basic VERA Dev Env](#)) and loaded in the the user's shell environment (see [Load VERA Dev Env](#)) then anyone with access to the VERA git repositories for the needed VERA components can clone the VERA repositories, configure, build, and install the VERA components.

Information on the installation directory layout is found at:

```
VERA/doc/install/README.VERA
```

8.1 Get Source For VERA Components To Install

Getting the sources for the VERA components to install is identical to getting them them to install and test the VERA Dev Env. See [Clone minimal bootstrap repositories](#) and [Clone remaining VERA components](#). Again, exactly what VERA git repositories that need to be cloned for a given piece of VERA functionality is not specified in this document.

8.2 Configure, Build, And Test VERA Components To Install

To set up to build, test, and install various VERA components by end users, one must first select a configuration setting. The following `do-configure` script is recommended for doing an install of VERA for usage by end users:

```
#!/bin/bash

EXTRA_ARGS=$@

if [ "$VERA_DIR" == "" ] ; then
  VERA_DIR=../../../VERA
fi

VERA_DIR_ABS=$(readlink -f $VERA_DIR)
echo "VERA_DIR_ABS = $VERA_DIR_ABS"

VERA_STD_GCC461=$VERA_DIR_ABS/cmake/std/gcc-4.6.1

cmake \
-D VERA_CONFIGURE_OPTIONS_FILE:FILEPATH="$VERA_STD_GCC461/mpi-release-static-options.
-D VERA_INSTALL_LIBRARIES_AND_HEADERS:BOOL=OFF \
-D CMAKE_INSTALL_PREFIX="$${VERA_DEV_ENV_BASE}/vera/'date +%Y-%m-%d'" \
-D VERA_ENABLE_TESTS:BOOL=ON \
-D DART_TESTING_TIMEOUT:STRING=600.00 \
$EXTRA_ARGS \
$${VERA_DIR_ABS}
```

Here, we assume this is put in the directory:

```
$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE_INSTALL
```

The above configure script uses static libraries and executables and turns off the install of the static libraries (since they are not needed by the statically build executables). This script uses the install directory:

```
<install-base-dir> = $${VERA_DEV_ENV_BASE}/vera/<YYYY-MM-DD>
```

but any `<install-base-dir>` can be used.

The configure, build, and test can then be performed with, for example:

```
$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE_INSTALL
$ ./do-configure \
-DVERA_ENABLE_CTeuchos=ON \
-DVERA_ENABLE_ForTeuchos=ON \
-DVERA_ENABLE_VERAIn=ON \
-DVERA_ENABLE_DataTransferKit=ON \
-DVERA_ENABLE_MPACT_libs=ON \
-DVERA_ENABLE_Insilico=ON \
-DVERA_ENABLE_COBRA_TF=ON \
-DVERA_ENABLE_VRIPSS=ON \
```

```
&> configure.out
$ make -j8 &> make.out
$ ctest -j8 &> ctest.out
```

All of the tests should pass. If they do not, please send email to casl-vera-users@casl.gov giving the tail of `ctest.out` and a copy of your `do-configure`

8.3 Install Built VERA Components

Before running `make install`, in order to protect VERA appropriately, one may need to set up the base directory for the install as:

```
$ cd ${VERA_DEV_ENV_BASE}
$ mkdir vera
$ chgrp -R vera-users vera
$ chmod 750 vera
$ chmod g+s vera
```

After a successful configure, build, and test has been performed, an install is simply performed as:

```
$ cd <SOME-BASE-DIR>/VERA.base/BUILDS/GCC-4.6.1/MPI_RELEASE_INSTALL
$ umask 0007
$ make -j8 install &> make.install.out
```

(setting `umask 0007` will ensure that only the `vera-users` group (or whatever it is called on the given system

This should set up an installation directory that looks like:

```
<install-base-dir>/
  bin/
  example/
  README.VERA
  README.react2xml
  ...
```

8.4 Documentation For Installed VERA Components

Once installed, information on how to access the installed VERA components along with their documentation and examples is found in:

```
<install-base-dir>/
  README.VERA
```

9 Appendix

9.1 Set Up Remote SSH Tunnel

In order to access the Git repositories on `casl-dev.ornl.gov` when outside of the ORNL network, a SSH tunnel must be set up through `login1.ornl.gov`. This requires the user to have an active UCAMS account with the 3-char `<ucams-id>`.

Once established, this SSH tunnel will set up a machine name called `casl-dev` on the local machine that can then be used in Git commands.

In one's home directory, create the file:

```
~/ssh/config
```

which contains:

```
host tunnelinit
  Hostname login1.ornl.gov
  User <ucams-id>
  LocalForward 28881 casl-dev.ornl.gov:22
  LocalForward 28882 u233.ornl.gov:22

Host casl-dev
  HostKeyAlias casl-dev.ornl.gov
  Hostname localhost
  Port 28881
  User <ucams-id>

Host u233
  HostKeyAlias u233.ornl.gov
  Hostname localhost
  Port 28882
  User <ucams-id>
```

Please make sure to change the above port numbers to not conflict with other ports being used on the system or other SSH tunnels. If multiple users use the same port numbers there will be collisions, or the host machine will disallow the connection altogether.

To set up the SSH tunnel, in any terminal, type:

```
$ ssh -fN tunnelinit
```

You will be prompted for a PASSCODE, this is your pin+6digits from RSA token). This will return to the command-line prompt and then one can then open a SSH connection to `casl-dev` as:

```
$ ssh casl-dev
```

(which will require the user to type in their UCAMS password).

After the tunnel is established, once can set up their SSH public/private keys and copy the public key over to `casl-dev` as described in [SSH setup for accessing casl-dev](#). Once the public SSH key is copied over and one can SSH to `casl-dev` without a password challenge, then this is confirmation that the SSH tunnel has been correctly established.

The SSH tunnel will stay open for some amount of time, longer if it is being actively used. However, it may be important in some cases to ensure that the tunnel is closed before logging off or doing other tasks. If a user creates a SSH tunnel, the user should be able to close the SSH tunnel. Since the SSH tunnel is in the background the user should use the following command to find the ssh tunnel process.:

```
$ ps aux | egrep ssh -fN tunnelinit
```

then issue the kill command to end the process and close the tunnel. This will return something along the lines of:

```
<ucams-id> 10535  0.3  0.0  66032  3804 ?   Ss   11:08   0:19 ssh -fN tunnelinit
```

The user can then use the kill command to end the tunnel process:

```
$ kill -9 <process number>
```

in this case 10535, the process number will always be the second item in the returned fields from the ps aux command.

This will close the SSH tunnel.