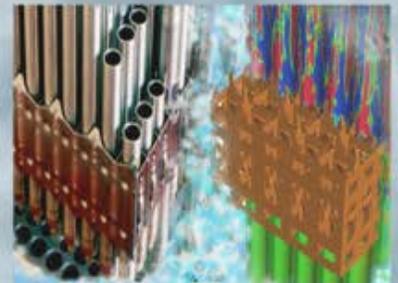
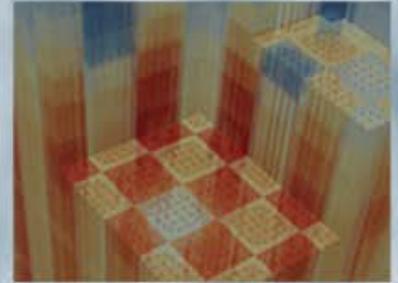


VERA Installation Guide

Roscoe A. Bartlett
Mark Baird
Mark Berrill
Joel A. Kulesza
Brenden T. Mervin

Oak Ridge National Laboratory

April 17, 2015



VERA Installation Guide

Author: Roscoe A. Bartlett (bartlettra@ornl.gov)
Author: Mark Baird (bairdml@ornl.gov)
Author: Mark Berrill (berrillma@ornl.gov)
Author: Joel A. Kulesza (jkulesza@umich.edu)
Author: Brenden T. Mervin (bmervin@epri.com)
Contact: support@casl.gov
Date: 2015-04-17
Version: vera-3.3.0

Contents

1 Introduction	1
2 Standard VERA Dev Env Directory Structure	1
3 Installation Process	3
3.1 Make sure the basic prerequisites are satisfied	3
3.2 Determine source, scratch and install directories	4
3.3 Get the base VERA and TriBITS source directories	4
3.4 Install the base development environment	4
3.5 Install the VERA TPLs	5
3.6 Build and build/install VERA Components	6
3.7 Final setup of installed VERA dev env and final cleanup	7
4 Details on Initial Setup	7
4.1 Requesting Access to VERA Repositories	7
4.2 System Configuration Considerations	7
4.3 Minimal System Package Setup	8
4.4 SSH Setup For Accessing casl-dev	8
4.5 Create Unix User and Group	9
4.6 Setup Base Directories for VERA	9
5 Details on TPL Installation	10
6 Details VERA Component Build, Test, and Installation	11
6.1 Load VERA Dev Env	11
6.2 Clone Remaining VERA Components	12
6.3 Checking Out a Specific Version of VERA	12
7 Details on Finalizing VERA Dev Env Installation	14

8	Details on Installing VERA	15
8.1	Get Source For VERA Components To Install	15
8.2	Configure, Build, And Test VERA Components To Install	15
8.3	Install Built VERA Components	16
8.4	Documentation For Installed VERA Components	16
9	Appendix	17
9.1	Set Up Remote SSH Tunnel	17
9.2	Minimal System Package Setup on Various Systems	18
9.3	Official VERA TPL Versions	19
9.4	Shared verses Static Libraries	19

1 Introduction

This guide describes the structure and setup of the standard VERA development environment (VERA Dev Env) and standard VERA 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 build, testing, and installation 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](#) and then obtain the remaining VERA source and build, test, and install any of the necessary VERA components on a given system. This document describes the process both for a development version of VERA and for a released tarball of the VERA sources.

One should start by getting acquainted with [Standard VERA Dev Env Directory Structure](#). Then one should work through the [Installation Process](#) to see the major steps needed. If everything goes well, the [Installation Process](#) contains all of the information needed to perform the full install of the VERA Dev Env as well as the install of VERA components themselves. The remaining sections contain more information and details for variations and tips for how to solve problems when things go wrong.

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-2.8.11/
    git-1.7.0.4/
    gitdist
  gcc-4.8.3/
  load_dev_env.sh
  toolset/
    gcc-4.8.3/
    mpich-3.1.3/
  tpls/
    opt/
      lapack-3.3.1/
      boost-1.55.0/
      zlib-1.2.7/
      hdf5-1.8.10/

```

```

    moab-4.5.0/
    hypre-2.8.0b/
    petsc-3.3-p4/
    silo-4.10.2/
    qt-4.8.2/
    ...
opt_static/
    ...
dbg/
    ...
dbg-checkedstl/
    ...
    ...
intel-13.x/
    load_dev_env.sh
    toolset/
        mpich-3.1.3/
    tpls/
        opt/
        ...

```

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 (see [Official VERA TPL Versions](#)) and VERA tools are deployed that are used for configuring, building, testing, and installing VERA.

For the standard GCC 4.8.3 VERA Dev Env, we set:

```
VERA_DEV_ENV_COMPILER_BASE=$VERA_DEV_ENV_BASE/gcc-4.8.3
```

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:

```
VERA_TPL_INSTALL_DIR=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt
```

For the install of static TPLs, it should be:

```
VERA_TPL_INSTALL_DIR=$VERA_DEV_ENV_COMPILER_BASE/tpls/opt_static
```

(But shared libs should be the default, see [Shared versus Static Libraries](#).)

All of the VERA Dev Env and TPL related install tools, etc. will use the variables (shown with the typical value):

VERA Directory Env Vars:

Variable	Common/Example Value
VERA_DEV_ENV_BASE	/tools/vera
VERA_DEV_ENV_COMPILER_BASE	\$VERA_DEV_ENV_BASE/gcc-4.8.3
VERA_TPL_INSTALL_DIR	\$VERA_DEV_ENV_COMPILER_BASE/tpls/opt
VERA_BASE_DIR	\$HOME/VERA.base
VERA_SCRATCH_DIR	\$VERA_BASE_DIR/scratch
VERA_DIR	\$VERA_DEV_ENV_BASE/vera-Source.X.Y.Z
VERA_BUILD_DIR	\$HOME/VERA_BUILD
VERA_INSTALL_DIR	/tools/vera_installs/2015-02-06

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

3 Installation Process

This section gives the set of commands to run in order to install the VERA development environment and do a test build of VERA. This assumes that the entire VERA development environment will be installed starting with GCC, MPI, CMake and on up. Customizing this can be done as needed but that is not covered here. Links to more detail are given for each step below or in the install tools themselves.

The steps are:

- [Make sure the basic prerequisites are satisfied](#)
- [Determine source, scratch and install directories](#)
- [Get the base VERA and TriBITS source directories](#)
- [Install the base development environment](#)
- [Install the VERA TPLs](#)
- [Build and build/install VERA Components](#)
- [Final setup of installed VERA dev env and final cleanup](#)

The steps in detail are given below.

3.1 Make sure the basic prerequisites are satisfied

Before one can install the VERA development/install environment and VERA itself, one must first:

1. Create a `vera-admin` Unix user, a `vera-users` Unix group, and the required base directories for building, testing, installing, and maintaining VERA (see [Create Unix User and Group](#) and [Setup Base Directories for VERA](#)).
2. Make sure the system can handle an installation of VERA (see [System Configuration Considerations](#) and [Minimal System Package Setup](#)).
3. If one is getting source from the VERA git repositories:
 1. Set up SSH access to the ORNL machine `casl-dev.ornl.gov` (`casl-dev` for short, see [SSH Setup For Accessing casl-dev](#) and [Set Up Remote SSH Tunnel](#)).
 2. Be approved to access the required protected VERA git repositories (see [Requesting Access to VERA Repositories](#) and [Clone Remaining VERA Components](#)).

3.2 Determine source, scratch and install directories

Set environment variables for the location of the installed/shared VERA development/install environment `VERA_DEV_ENV_BASE` and the base location of the VERA sources `VERA_BASE_DIR` to drive the install and other specified in [VERA Directory Env Vars](#). For example:

```
# Set the base directories (you can pick any paths you want)
export VERA_DEV_ENV_BASE=/tools/vera
export VERA_BASE_DIR=${VERA_DEV_ENV_BASE}/VERA.base
export VERA_SCRATCH_DIR=${VERA_BASE_DIR}/scratch
export VERA_TPL_INSTALL_DIR=${VERA_DEV_ENV_BASE}/gcc-4.8.3/tpls/opt
export VERA_BUILD_DIR=$HOME/VERA_BUILD
export VERA_INSTALL_DIR=/tools/vera/installs/`date +%Y-%m-%d`

# Create some base directories (if they don't already exist)
mkdir ${VERA_BASE_DIR}
mkdir ${VERA_SCRATCH_DIR}
```

All of the other directories will be created below or automatically by the various install tools that are run.

3.3 Get the base VERA and TriBITS source directories

If using a VERA tarball, do:

```
cd ${VERA_BASE_DIR}/
tar -xzf ~/vera-Source.X.Y.Z.tar.gz
```

then set:

```
export VERA_DIR=${VERA_BASE_DIR}/vera-Source.X.Y.Z
```

If cloning the sources from `casl-dev` (if they are not already cloned), first set you the SSH tunnel (see [Set Up Remote SSH Tunnel](#)) if to `casl-dev` if needed with:

```
ssh -fN tunnelinit
```

then do the clones with:

```
cd ${VERA_BASE_DIR}/
git clone git@casl-dev:VERA
cd VERA/
git clone git@casl-dev:TriBITS
```

then set:

```
export VERA_DIR=${VERA_BASE_DIR}/VERA
```

3.4 Install the base development environment

To install all of the genetic tools (does not include the TPLs) one can run the single command:

```
cd ${VERA_SCRATCH_DIR}/
${VERA_DIR}/cmake/tribits/devtools_install/install_devtools.py \
  --install-dir=${VERA_DEV_ENV_BASE} --parallel=8 --do-all \
  &> install_devtools.out
```

TIPS:

- Running `install_devtools.py` with `--no-op` will show what the install tool will do without actually doing anything (see `--help` for details).
- Adjust `--parallel=8` to the appropriate number of processes for the current machine.
- If any error occurs, look at the log file `install_devtools.out` to see what failed and then look at the log file it points to to see the actual errors.

This should install GCC 4.8.3, MPICH 3.1.3, CMake 2.8.11, git 1.7.0.4, `gitdist`, and `load_dev_env.sh` as shown in [Standard VERA Dev Env Directory Structure](#). For details on what can go wrong and how to deal with problems with the install, use `--help`.

After a successful install, one needs to source the installed `load_dev_env.sh` script as:

```
source ${VERA_DEV_ENV_BASE}/gcc-4.8.3/load_dev_env.sh
```

then `PATH` is prepended to find the installed programs `cmake`, `git`, `gitdist`, `gcc`, `mpicc`, ..., in the locations shown in [Standard VERA Dev Env Directory Structure](#)). Make sure you have sourced the new dev env, for example, with:

```
which cmake
which gcc
which mpicc
```

which should return:

```
.../common_tools/cmake-2.8.11/bin/cmake
.../gcc-4.8.3/toolset/gcc-4.8.3/bin/gcc
.../gcc-4.8.3/toolset/mpich-3.1.3/bin/mpicc
```

3.5 Install the VERA TPLs

First, get the `vera_tpls` source repository and the matching version of the TPLs:

```
cd ${VERA_BASE_DIR}/
git clone https://github.com/CASL/vera_tpls
cd vera_tpls/
git checkout vera-tpls-2.3
```

If getting the `vera_tpls` repo from `casl-dev`, instead use:

```
git clone git@casl-dev:prerequisites/vera_tpls
```

Then configure and build the TPLs with:

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh -DPROCS_INSTALL=8 \
&> install_tpls.out
```

NOTE: Adjust `-DPROCS_INSTALL=8` to the appropriate number of processes for the current machine.

By default, this installs a shared library version of the TPLs (see [Shared verses Static Libraries](#)). One can install a static library version passing in `-DENABLE_SHARED=OFF` and pass in:

```
-DCMAKE_INSTALL_PREFIX=${VERA_DEV_ENV_COMPILER_BASE}/tpls/opt_static
```

For more details, see [Details on TPL Installation](#).

3.6 Build and build/install VERA Components

Finally, one must test the installed VERA Dev Env by configuring, building, testing, and installing VERA from source (installation not required).

If the VERA source was obtained from a source tarball file (see above), the all of the sources needed to build the VERA components should be in place (pointed to by `${VERA_DIR}`).

If the VERA source is obtained from the git repos on `casl-dev`, then the remaining VERA git repos must be cloned. First, if an SSH tunnel is necessary, first initial the tunnel (see [Set Up Remote SSH Tunnel](#)) with:

```
ssh -fN tunnelinit
```

Then the remaining repos can be cloned using:

```
cd ${VERA_DIR}/
./clone_vera_repos.py
```

Once the source is obtained, one can configure, build, test, and install with:

```
# Set up the build dir
mkdir -p ${VERA_BUILD_DIR}
cd ${VERA_BUILD_DIR}/
ln -s ${VERA_DIR}/cmake/std/gcc-4.8.3/do-configure.MPI_RELEASE_SHARED \
do-configure

# Configure, build, and test
./do-configure \
-D CMAKE_INSTALL_PREFIX=${VERA_INSTALL_DIR} \
-D CASL_MOOSE_PARALLEL_BUILD_LEVEL=8 \
-D VERA_ENABLE_ALL_PACKAGES=ON &> configure.out
make -j8 &> make.out
ctest -j8 &> ctest.out
```

NOTE: Above shows the usage of 8 processes to build the code and run the tests. Change 8 to whatever value is appropriate for the current machine.

If all of the tests passed, i.e. the output from `ctest` looks something like:

```
100% tests passed, 0 tests failed out of 697
```

```
Label Time Summary:
CASL_MOOSE           = 78.30 sec
COBRA_TF             = 70.01 sec
CTeuchos            = 0.40 sec
DataTransferKit     = 16.06 sec
ForTeuchos          = 0.73 sec
Insilico             = 281.07 sec
MPACT_Drivers       = 303.89 sec
MPACT_libs          = 86.12 sec
TriBITS             = 469.13 sec
VERAIn              = 199.05 sec
VRIPSS              = 2283.48 sec
```

```
Total Test time (real) = 690.22 sec
```

then all is well.

If one wants to install VERA for usage by others, this can be done with:

```
make -j8 install &> make.install.out
```

Permissions on the installed version of VERA should be set using:

```
chmod chgrp -R vera-users $VERA_INSTALL_DIR
```

(see [Create Unix User and Group](#).)

After the install, see the instructions using VERA in the file:

```
$VERA_INSTALL_DIR/README
```

In particular, one should source the script:

```
$VERA_INSTALL_DIR/load_env.sh
```

in order to set up one's environment (i.e. `PATH` and other variables) so as to run the installed executables and scripts. See the online copy [README.VERA](#) (becomes the installed file `README`).

NOTE: The above commands build and install a shared library version of VERA (see [Shared verses Static Libraries](#)). A static library version can be installed by replacing `MPI_RELEASE_SHARED` with `MPI_RELEASE_STATIC` above. One would want to do this if installing static TPLs using `-DENABLE_SHARED=OFF` in [Install the VERA TPLs](#).

3.7 Final setup of installed VERA dev env and final cleanup

After the VERA dev env (basic tools and TPLs) has been installed, then one must open up the directories for others to access the installed tools and libraries. Since there are no export controlled or other sensitive data contained in the installed VERA dev env, one can open it up to everyone by doing:

```
chmod -R a+rX ${VERA_DEV_ENV_BASE}
```

When one is finished installing and testing the VERA dev env, then one can delete the scratch directory `${VERA_SCRATCH_DIR}` if desired. However, one should save the generated `*.out` and `*.log` files to archive details about the VERA dev env install for later reference before deleting the scratch directory.

Now that the installation process has been described, more details about the installation of the VERA dev env and VERA itself are given below.

4 Details on Initial Setup

Before any of the VERA-specific prerequisites can be installed, 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 (or just untar the VERA sources from a tarball) and build the various VERA components from source.

4.1 Requesting Access to VERA Repositories

If installing VERA from a release tarball, then one already has the sources. However, if installing VERA by pulling from the git repositories on `casl-dev.ornl.gov`, one must first be given access. Before one can access the various VERA git repositories on `casl-dev.ornl.gov`, one must first be given an ORNL UCAMS account, be added to the gitolite site 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 start getting this access setup. Please provide them with a list of the specific VERA git repositories (or VERA components) that one needs access.

4.2 System Configuration Considerations

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

- Make sure there are enough processors available for parallel compilation. It is assumed in this document that there at least 8 cores available with `witch` to run parallel builds, tests, etc. If that is not the case, one can reduce the parallel level as will be obvious with each command.
- Use the fastest file storage system available for holding source code, compiling, and running test cases (e.g. Try to avoid NFS-mounted directories for compilation and testing).

Meanwhile, other considerations include:

- Can the machine create SSH tunnels to download VERA and TPL repositories (if one is not building from a tarball)?
- Can the machine access the Internet (e.g. `github.com`)? If not, ensure that all packages needed are available in the VERA and/or TPL repositories from `casl-dev` using an SSH tunnel.
- Can a remote users access the machine if troubleshooting assistance is needed?

4.3 Minimal System Package Setup

Before proceeding to start installing the various VERA prerequisites, the following software packages must be installed on the system (i.e. using the systems native package manager):

- **GCC gcc C compiler** (does not have to be a very recent version): Needed to build the official version of GCC from source. The newer GCC version will be used to build everything else.
- **GCC g++ C++ compiler** (does not have to be a very recent version): Needed to build CMake from source. The newer GCC g++ version will be used to build everything else.
- **patch**: Needed to support CMake.
- **texinfo**: The `makeinfo` program is needed to install GCC from source.
- **GNU M4**: Needed for the build of `autoconf`.
- **Git** (version 1.6.0 or newer): Needed to clone several git repositories. A newer version of git will be installed as part of the TriBITS/VERA development tools.
- **python** (version 2.6.6 or newer but not 3.x): Needed for the basic install scripts and some helper scripts used in VERA.
- **bash**: Needed for some of the shell scripts that refer explicitly to `bash`.

- **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 VERA input parser `react2xml.pl`.
- **X11** (development libraries, not just the runtime libraries): Needed to build and install the required VERA TPL QT.
- **ZLIB** (development libraries, not just the runtime libraries): Needed to build and install the required VERA TPL HDF5.

For the exact packages that one needs to install on different systems, see [Minimal System Package Setup on Various Systems](#).

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

4.4 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 by email to casl-vri-infrastructure@casl.gov in order to be registered with one's account in the gitolite system on `casl-dev`. In addition, you must be approved the VERA git repositories before one's key can be added.

If one is not on the ORNL network with direct access to the machine `casl-dev`, then one will need to open the SSH tunnel to `casl-dev` using:

```
ssh -fN tunnelinit
```

(See [Set Up Remote SSH Tunnel](#).)

After one's public SSH key has been registered with gitolite on `casl-dev` (and the SSH tunnel to `casl-dev` has been established if needed), then one can test to see if one has repo access by running the command:

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

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

```
ssh git@casl-dev info

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

...
R    TriBITS
...
R    Trilinos
...
R    VERA
...
```

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

Access to other repositories requires being explicitly added to the appropriate gitolite groups (again, contact your CASL representative).

4.5 Create Unix User and Group

In order to properly administer and protect VERA installations, it is recommended to set up the following:

- A `vera-admin` user: Unix user account specifically for maintaining VERA installations.
- A `vera-users` group: List of Unix users that have access permission and need-to-know for running the installed 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 support@casl.gov or some other responsible CASL representative for guidance.

4.6 Setup Base Directories for VERA

Once the `vera-admin` user account has been created, the `${VERA_DEV_ENV_BASE}` and `${VERA_INSTALL_DIR}` directories need to be created by the `root` user or a user with `sudo` privileges. Then ownership of these directories needs to be transferred over to the `vera-admin` user. This can be performed using the following commands:

```
export VERA_DEV_ENV_BASE=<some-dir>
mkdir ${VERA_DEV_ENV_BASE}
chown -R vera-admin ${VERA_DEV_ENV_BASE}

export VERA_INSTALL_DIR=<some-dir>
mkdir ${VERA_INSTALL_DIR}
chown -R vera-admin ${VERA_INSTALL_DIR}
```

To reduce the need for `root` or `sudo` access during the installation process, a `${VERA_ROOT_DIR}` can be used which will contain the following directories:

```
${VERA_ROOT_DIR}/
  ${VERA_DEV_ENV_BASE}/
  ${VERA_SCRATCH_DIR}/
  ${VERA_BASE_DIR}/
  ${VERA_BUILD_DIR}/
  ${VERA_INSTALL_DIR}/
```

By using this directory structure, someone with `root` or `sudo` access only needs to perform the following operations:

- Make sure all of the required system packages have been installed.
- Create the `vera-admin` user.
- Create the `vera-users` group.
- Create the `${VERA_ROOT_DIR}`.
- Transfer ownership of `${VERA_ROOT_DIR}` over to `vera-admin`.

Once these actions have been performed, the `vera-admin` user will have all of the necessary permissions to build, test, install, and maintain the VERA installation.

5 Details on TPL Installation

A standard installation of the VERA TPLs can be performed using the command:

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh -DPROCS_INSTALL=8 \
  &> install_tpls.out
```

as described in [Install the VERA TPLs](#). However, this is just a thin little shell script that uses a CMake ExternalProject build of the TPLs.

The TPL build system can be pulled by using:

```
cd ${VERA_BASE_DIR}/
git clone https://github.com/CASL/vera_tpls
```

when pulling from github. When pulling from casl-dev using SSH, use:

```
cd ${VERA_BASE_DIR}/
git clone git@casl-dev:prerequisites/vera_tpls
```

The shell script `install_tpls.sh` just creates a CMake binary directory:

```
${VERA_SCRATCH_DIR}/TPL_build/
```

and then runs CMake pointing to the CMake ExternalProject should directory:

```
${VERA_BASE_DIR}/vera_tpls/TPL_build/
```

This TPL_build CMake project accepts a number of CMake cache variables which include:

- **CMAKE_BUILD_TYPE**: Release or Debug
- **ENABLE_SHARED**: ON means that only shared *.so libs will be installed while OFF means that only static *.a libs will be installed.
- **CMAKE_INSTALL_PREFIX**: Base path to TPL install location. The TPLs are installed under this directory, one subdirectory for each TPL.
- **TPL_LIST**: List of TPLs to install. This can be a common separated list. The full list of TPLs (and the default value for this variable) is:

```
ZLIB;HDF5;LAPACK;HYPRE;QT;MOAB;PETSC;SILO;BOOST
```

Any subset of TPLs can be listed and will be installed instead.

- **PROCS_INSTALL**: Number of processors to use to build each TPL. At least 4 is recommended due to the lengthy BOOST and QT build times. Last Line Path to “vera_tpls/TPL_build repository sub-directory

The install tool:

```
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh
```

is a bash script that sets all of the defaults needed to do the basic build, including compiler options, etc. But as shown in [Install the VERA TPLs](#), one can pass in any CMake cache variable and it will override the defaults set in `install_tpls.sh`. For example, to install static libraries in addition to shared libraries and use 16 processes, one can use:

```
cd ${VERA_SCRATCH_DIR}/
${VERA_BASE_DIR}/vera_tpls/TPL_build/install_tpls.sh \
-DPROCS_INSTALL=16 \
-DENABLE_SHARED=OFF \
-DCMAKE_INSTALL_PREFIX=${VERA_DEV_ENV_COMPILER_BASE}/tpls/opt_static \
&> install_tpls.out
```

After the above install script finishes calling `cmake` to configure the TPL build, it just calls:

```
make
```

to drive the build and install process.

One can just examine the script `install_tpls.sh` to see what it does and can therefore make needed modifications to the install process and run the commands manually if needed.

6 Details VERA Component Build, Test, and Installation

Once the VERA prerequisites (i.e., compilers, TPLs, other tools) have been installed, one needs to clone the remaining VERA git repositories for the desired components (if working from the version-controlled source), 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 not already loaded, then run:

```
source ${VERA_DEV_ENV_COMPILER_BASE}/load_dev_env.sh
```

6.2 Clone Remaining VERA Components

When working from a tarball distribution of VERA, there is nothing left to clone so one can skip this section.

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

```
cd ${VERA_DIR}/  
./clone_vera_repos.py
```

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. Ask your CASL VERA contact about what repositories you need to clone for your needs.

NOTE: One must be part of the gitolite group for `git@casl-dev` 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 under `git@casl-dev`. To see if one has misspelled the repo or if one just does not have permission, 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.

6.3 Checking Out a Specific Version of VERA

When working from a tarball distribution of VERA, the version is fixed in which case, one can skip this section.

However, when working from the git repositories, various versions of VERA can be accessed.

The most recent version of VERA can be pulled using:

```
cd ${VERA_DIR}/  
gitdist 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 for what is in `casl-dev/master`.

Therefore, 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). Checking out a specific version of the VERA repositories is accomplished using the `gitdist` 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 VERA build/test posted to the VERA CDash server includes exact git version information in the generated `VERARepoVersion.txt` file. For a subset of VERA repos, a `VERARepoVersion.txt` file looks like:

```
*** Base Git Repo: VERA
2d2d797 [Fri Feb 13 15:09:59 2015 -0500] <bartlettra@ornl.gov>
SQUASH AGAINST 'Create gcc-4.8.3 env TPL list, upgrade to hdf5' and 'Remove com
** Git Repo: TriBITS
cccb63b [Wed Feb 11 07:54:42 2015 -0500] <bartlettra@ornl.gov>
Minimal work to get InstallDriver unit test to work (PHI Kanban #3217)
** Git Repo: Trilinos
f91ed19 [Wed Jan 28 16:15:55 2015 -0500] <bartlettra@ornl.gov>
Fix setting TRILINOS_BUILD_SHARED_LIBS (CMake 3.1 policy)
** Git Repo: TeuchosWrappersExt
e895fcf [Mon Oct 20 16:54:13 2014 -0400] <bartlettra@ornl.gov>
Switching from deprecated DEPLIBS to TESTONLYLIBS
** Git Repo: COBRA-TF
c376794 [Mon Feb 9 09:45:25 2015 -0500] <rks171@gmail.com>
Merge branch 'master' of casl-dev:COBRA-TF
** Git Repo: VERAInExt
c5cb1a7 [Mon Feb 9 09:51:25 2015 -0500] <rks171@gmail.com>
Merge branch 'master' of casl-dev:VERAInExt
** Git Repo: DataTransferKit
4ed4c31 [Tue Nov 11 17:01:06 2014 -0500] <uy7@ornl.gov>
Fixing std::vector error in CommIndexer test
** Git Repo: MOOSEExt
e713009 [Fri Nov 14 16:14:57 2014 -0500] <rppawlo@sandia.gov>
Removed .exe suffix from xml2moose executable.
** Git Repo: MOOSEExt/MOOSE
d2881f5 [Tue Nov 11 15:27:54 2014 -0500] <rppawlo@sandia.gov>
Merge branch 'inl_clean_svn'
** Git Repo: SCALE
f92392f [Tue Feb 10 23:40:26 2015 -0500] <clarnokt@ornl.gov>
changeset: 14402:5a86dc26cb7b tag: tip user: Kevin Clarno <kto@
** Git Repo: SCALE/Exnihilo
52aae91 [Tue Feb 10 23:33:05 2015 -0500] <clarnokt@ornl.gov>
Merge remote branch 'angband/master'
** Git Repo: MPACT
bd7123b [Tue Feb 10 13:25:31 2015 -0500] <bkochuna@umich.edu>
Merge remote branch 'arc-05/master'
** Git Repo: LIMEExt
8961330 [Mon Feb 2 10:08:37 2015 -0500] <mervinbt@ornl.gov>
Fix CRLF with LF
** Git Repo: Mamba
92ab580 [Tue Dec 16 19:18:54 2014 -0500] <bartlettra@ornl.gov>
Remove call to deprecated function
** Git Repo: PSSDriversExt
bc63af1 [Wed Feb 11 09:39:04 2015 -0500] <bartlettra@ornl.gov>
Fix help_only test for MPICH 3.1.3 (PHI Kanban #3217)
```

Before updating to a specific version of VERA, a CASL representative will provide the customer a specific

VERARepoVersion.txt file, typically named VERARepoVersion.<newdate>.txt (for some specific date <newdate>=YYYY-MM-DD), which the customer can use to checkout that specific version with:

```
cd ${VERA_DIR}/
gitdist fetch
gitdist --dist-version-file=~/.VERARepoVersion.<newdate>.txt \
  checkout _VERSION_
```

(see `gitdist --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 VERARepoFileVersion.<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_refactor_3104
```

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:

```
cd ${VERA_DIR}/
gitdist fetch
gitdist \
  --dist-mod-only \
  --dist-version-file=~/.VERARepoVersion.<newdate>.txt \
  --dist-version-file2=${VERA_DEV_ENV_BASE}/vera/<olddate>/VERARepoVersion.txt \
  log --name-status _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.

7 Details on Finalizing VERA Dev Env Installation

Once one has finished installing the VERA prerequisites consisting of the compilers, OpenMPI, 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 fix up permissions on the installed files and directories.

The directory permissions for the Unix tools, compilers, and TPLs can be opened up for all to use. This can be accomplished with the following command:

```
chmod -R a+rX <some-dir>
```

For example, world readable permissions can be assigned using the following commands:

```
chmod -R a+rX ${VERA_DEV_ENV_BASE}/common_tools
chmod -R a+rX ${VERA_DEV_ENV_BASE}/gcc-4.8.3
```

That should allow anyone to read any of the installed files and directories (but not modify them and mess them up).

If the owning user is not `vera-admin`, change the owning user to avoid accidental modifications by the original installer using the following command:

```
chown -R vera-admin <some-dir>
```

for example, if the original installer is not the current owning using, then the following commands can be run to transfer ownership over to the `vera-admin` user:

```
chown -R vera-admin ${VERA_DEV_ENV_BASE}/common_tools
chown -R vera-admin ${VERA_DEV_ENV_BASE}/gcc-4.8.3
```

This will avoid problems with accidental modifications to the installed directories by the original installer.

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

```
rm -rf ${VERA_SCRATCH_DIR}
rm -rf ${VERA_BASE_DIR}/vera_tpls
```

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:

```
${VERA_BASE_DIR}/
```

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:

```
rm -rf ${VERA_BASE_DIR}
```

All that would left would be the install of the VERA dev env under `${VERA_DEV_ENV_BASE}/.`

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

```
source ${VERA_DEV_ENV_BASE}/gcc-4.8.3/load_dev_env.sh
```

(in their `.bash_profile` file for instance).

8 Details on Installing VERA

Once the VERA Dev Env is installed on a system and loaded in the user's shell environment, then anyone with access to the VERA sources (either through a tarball or through the cloned git repositories) can configure, build, test, and install the VERA components.

Information on the installation directory layout is found at:

```
${VERA_DIR}/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 to install and test the VERA Dev Env. This can be done simply with:

```
cd ${VERA_BASE_DIR}/
git clone git@casl-dev:VERA
cd VERA/
./clone_vera_repos.py
```

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. For most systems, shared libraries should be preferred since they typically use up less disk space.

When installing VERA, it is important to build in a *very* shallow build directory. For example, using local scratch space to set up a build directory:

```
cd /scratch/<user-id>/
mkdir VERA_BUILD
cd VERA_BUILD/
ln -s $VERA_DIR/cmake/std/gcc-4.8.3/do-configure.MPI_RELEASE_SHARED \
    do-configure
```

If you don't use a shallow build directory, then CMake may fail to replace the RPATHs in the executables due to strings being too long.

Once a do-configure script is set up, one just needs to configure pointing to the final install location. Assuming one will install into:

```
export VERA_INSTALL_DIR=/tools/vera_installs/`date +%Y-%m-%d`
```

one would configure with:

```
cd /scratch/<user-id>/VERA_BUILD/
./do-configure \
  -D CMAKE_INSTALL_PREFIX=$VERA_INSTALL_DIR \
  -D CASL_MOOSE_PARALLEL_BUILD_LEVEL=8 \
  -D VERA_ENABLE_ALL_PACKAGES=ON &> configure.out
```

Assuming the configure passed, one would build and test using:

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

(see [Build and build/install VERA Components](#)).

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

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 /tools/
mkdir vera_installs
chgrp -R vera-users vera_installs
chmod 750 vera_installs
chmod g+s vera_installs
```

(see the `vera-users` group described in [Create Unix User and Group](#).)

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

```
cd /scratch/<user-id>/VERA_BUILD/
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) will have access to the installed VERA software.

This should set up an installation directory that looks like:

```
{VERA_INSTALL_DIR}/
bin/
lib/
share/
README
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 the file:

```
{VERA_INSTALL_DIR}/
README
```

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

Host casl-dev
  HostKeyAlias casl-dev.ornl.gov
  Hostname localhost
  Port 28881
  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 the command:

```
ssh -fN tunnelinit
```

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

```
ping casl-dev
```

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
```

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.

9.2 Minimal System Package Setup on Various Systems

Package managers on different systems have different names for the packages they install that are needed described in [Minimal System Package Setup](#).

The following table gives the names of the packages that need to be installed on three different machines where VERA has been installed in the past. In addition, the last column specifies the executable that should be in the system's path provided that the package is installed. Testing for the existence of these executables can be accomplished with the `which` command. For example:

```
which gcc
```

may return:

```
/usr/bin/gcc
```

Note that if a path to the executable is not returned by the `which` command, then the package is not installed.

Package	CentOS 6.6/Redhat 6.4	Ubuntu 14.04.1	Executables
GCC C Compiler	gcc	gcc	gcc
GCC C++ Compiler	gcc-c++	g++	g++
GNU M4	m4	M4	M4
GNU texinfo	texinfo	texinfo	makeinfo
Git	git	git	git
Python	python	Python-minimal	python
Bash	bash	bash	bash
Perl	perl	perl-base	perl
X11 (dev)	libX11-devel	libx11-dev	
Xorg (dev)	xorg-x11-server-devel	xorg-dev	
Zlib (dev)	zlib-devel	zlib-dev	

Note that there are no executables for the X11, Xorg, or Zlib development libraries. To verify that these packages are installed, one can try and look for a required library (in the user's `LD_LIBRARY_PATH`) or a required include file or directory in `/usr/include`. Suggested files/folders to check for to verify installation of the X11, Xorg, and Zlib development libraries are provided in the following table.

Package	usr/lib or /usr/lib64	/usr/include
X11 (dev)	libX11.so	X11

... continued on next page

Package	usr/lib or /usr/lib64	/usr/include
Xorg (dev)		xorg
Zlib (dev)	libz.so	zlib.h

On CentOS and Redhat systems, the command `yum install <package>` installs a package, for example:

```
yum install g++
```

On Ubuntu systems, the command `apt-get install <package>` installs a package, for example:

```
apt-get install g++
```

Note that someone with `sudo` must install these packages since they get installed into the base system directories. However, it is possible to use some package managers to install missing packages to a different location that does not require root or sudo access but that is beyond the scope of this document (consult the documentation for the package manager on your system).

9.3 Official VERA TPL Versions

The source code for the official TPL versions that are installed using the [install_tpls.sh](#) tool are stored in the `vera_tpls` git repository and the current official versions are shown in the below table.

Third Partly Library (TPL)	Version
LAPACK	3.3.1
Boost	1.55.0
ZLib	1.2.7
HDF5	1.8.10
MOAB	4.5.0
HYPRE	2.8.0b
PETsc	3.3-p4
Silo	4.10.2
QT	4.8.2

The exact version of TPLs matching this particular version of VERA is always given by the git tag for the `vera_tpls` repo given in the section [Install the VERA TPLs](#).

9.4 Shared verses Static Libraries

The default install of the VERA TPLs in [Install the VERA TPLs](#) and VERA itself in [Build and build/install VERA Components](#) use shared libraries. However, these instructions also describe how to installed static libraries. In general, one should prefer shared libraries over static libraries on most platforms as shared libraries use less disk space, link faster, and allow for quick bug fixes and simpler upgrades. But if `RPATH` is not written into the executables and shared libraries, then one will have to set `LD_LIBRARY_PATH` pointing to the location of the correct shared libraries.

However, some systems (especially some HPC machines) either require or strongly recommend the usage of static libraries. In addition, static libraries are more self-contained and are less sensitive to the machine environment from which they run and they never have `RPATH` issues. But executables built using static libraries take up far more disk space and take longer to link.