clavin User Guide

Release 0.1

Philip Miller, Skip Breidbach

October 31, 2013

CONTENTS

CHAPTER

ONE

INTRODUCTION

clavin defines the SRI VT software build and release workflow. The primary tools used in clavin are Artifactory, maven, cmake, and jenkins. Artifactory is used as the repository of binary artifacts. Maven is used to both manage dependencies and deploy artifacts that are consumed and generated by the build process. cmake is used to generate the back-end build files for a particular software component. jenkins is the tool providing a continuous integration (CI) build and test system.

clavin is just a code-name that does not have a particular meaning. It may help to think of *clavin* sounding like *cmake* and *maven* joined together. It is also last name of the Cliff Clavin character on the TV series *Cheers*.

1.1 Developing a clavin Based Project

Artifactory is a server-based product with the VT instance running at https://artifactory-vt. Please make sure you have read access to this server.

Typical developers do not install jenkins on their individual development computers, so we discuss that later in this document in *Configuring Jenkins*.

The rest of this section will focus on the installation of cmake and maven.

1.2 Maven Installation

1.2.1 Prerequisite Java JDK

Download the jdk from oracle.com[1]. At the time this was written, October, 2013, the latest version was 7u45.

windows Download and install the Windows x64 `jdk-7u45-windows-x64.exe`

CentOs Download and install the file `jdk-7u45-linux-x64.rpm`. Then install using:

```
sudo rpm -Uvh $HOME/Downloads/jdk-7u40-linux-x64.rpm
```

I found the above rpm command here[2]

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
[2] http://www.if-not-true-then-false.com/2010/install-sun-oracle-java-jdk-jre-7-on-fedora-centos-red-hat-rhel/

Ubuntu Download and unpack the file `jdk-7u45-linux-x64.tar.gz`. You will need to set the `JAVA_HOME` to point to the parent folder of `bin/java`

Alternatively, *Oracle Java JDK with apt-get* shows how to use **apt-get** to install the jdk. Installing with apt-get will provide automatic notifications of updates.

Note that maven requires the java *jdk*; the *jre* is not sufficient for using for maven.

1.2.2 Maven Installation Instructions

The content in this subsection was copied from maven Installation Instructions[3].

Windows

1. Unzip the distribution archive, i.e. apache-maven-3.1.1-bin.zip to the directory you wish to install Maven 3.1.1. These instructions assume you chose C:Program FilesApache Software Foundation. The subdirectory apache-maven-3.1.1 will be created from the archive.
2. Add the `M2_HOME` environment variable by opening up the system properties (WinKey + Pause), selecting the "Advanced" tab, and the "Environment Variables" button, then adding the M2_HOME variable in the user variables with the value C:Program FilesApache Software Foundationapache-maven-3.1.1. Be sure to omit any quotation marks around the path even if it contains spaces. Note: For Maven 2.0.9, also be sure that the M2_HOME doesn't have a '' as last character.
3. In the same dialog, add the `M2` environment variable in the user variables with the value %M2_HOME%bin.
4. Optional: In the same dialog, add the MAVEN_OPTS environment variable in the user variables to specify JVM properties, e.g. the value -Xms256m -Xmx512m. This environment variable can be used to supply extra options to Maven.
5. In the same dialog, update/create the `PATH` environment variable in the user variables and prepend the value %M2% to add Maven available in the command line.
6. In the same dialog, make sure that `JAVA_HOME` exists in your user variables or in the system variables and it is set to the location of your JDK, e.g. `C:Program FilesJavajdk1.5.0_02` and that %JAVA_HOME%bin is in your `PATH` environment variable.
7. Open a new command prompt (Winkey + R then type cmd) and run `mvn --version` to verify that it is correctly installed.

Unix-based Operating Systems (Linux, Solaris and Mac OS X)

1. Extract the distribution archive, i.e. apache-maven-3.1.1-bin.tar.gz to the directory you wish to install Maven 3.1.1. These instructions assume you chose /usr/local/apache-maven. The subdirectory apache-maven-3.1.1 will be created from the archive.
2. In a command terminal, add the `M2_HOME` environment variable, e.g. `export M2_HOME=/usr/local/apache-maven/apache-maven-3.1.1`.
3. Add the `M2` environment variable, e.g. `export M2=$M2_HOME/bin`.
4. Optional: Add the MAVEN_OPTS environment variable to specify JVM properties, e.g. export MAVEN_OPTS="-Xms256m -Xmx512m". This environment variable can be used to supply extra options to Maven.
5. Add M2 environment variable to your path, e.g. `export PATH=$M2:$PATH`.

[3] http://maven.apache.org/download.cgi

6. Make sure that `JAVA_HOME` is set to the location of your JDK, e.g. `export JAVA_HOME=/usr/java/jdk1.5.0_02` and that `$JAVA_HOME/bin` is in your PATH environment variable.
7. Run `mvn --version` to verify that it is correctly installed.

Note: Maven and Jenkins

The Jenkins master will automatically install/use **maven** from its configuration. So a manual installation of maven is not needed (and probably not wanted) on a slave. Developers will need to have maven (and the prerequisite jdk) on their development computers.

1.3 cmake Installation

Linux Use the source distribution in our local cmake repository on git-open. The commands below will install **cmake** to `/usr/local`:

```
git clone ssh://git-open/scm/3rdparty/cmake.git -b v2.8.12
mkdir cmake-build
cd cmake-build
../cmake/configure
make -j4 -l4
sudo make install
```

Windows Use the latest installer from cmake.org[4].

1.4 A note about this documentation

This documentation is written in RestructuredText and generated using Sphinx. This is a common toolset, particularly in the python domain, but is also widely used elsewhere; e.g., The OpenCV documentation uses this toolset. If you are unfamilar with RestructuredText and Sphinx, here are a few links to get you started:

- Installation[5]
- ReStructuredText Primer[6] describes the basics of ReStructuredText markup.
- Sphinx Markup[7] describes relevant additional constructs available when using the sphinx builder.

The above links should be adequate for anyone wishing to contribute to this documentation.

[4] http://www.cmake.org/cmake/resources/software.html
[5] http://docutils.sourceforge.net/README.html#installation
[6] http://sphinx-doc.org/rest.html
[7] http://sphinx-doc.org/markup/index.html

CHAPTER

TWO

THE BOB PARENT POM

The Bob pom file at `/cmake-maven-parent/pom.xml` should be used as the parent pom file for all pom files conforming to the Bob conventions for building cmake based C++ projects. You make this the parent pom by using the following lines in your `pom.xml`

2.1 Using the Bob Parent POM

```
<parent>
    <groupId>com.sri.vt.clavin</groupId>
    <artifactId>cmake-maven-parent</artifactId>
    <version>0.1.2-SNAPSHOT</version>
</parent>
```

For multi-module builds, the above lines would be placed into the aggregator pom and individual modules would make their parent be their primary parent.

2.2 Properties Defined in the Bob Parent Pom

All of the properties in the parent pom are shown in the snippet below. Important properties include:

package.prefix.directory the directory to which *external package*'s will be installed. In cmake terms, this is the *CMAKE_INSTALL_PREFIX*.

```
    <properties>
        <!--
        The CMake INSTALL_PATH. Usual form is something like
            <basedir>/build/[operatingsystem]/exports/foo-1.3.2-snapshot
        The last directory becomes the root directory of the archive.
        -->
        <package.root.directory>${package.file.name}</package.root.directory>
        <package.install.directory>
            ${project.build.directory}/exports/${package.root.directory}
        </package.install.directory>

        <!-- becomes the cmake prefix path - all dependencies are unpacked here -->
        <package.prefix.directory>${project.build.directory}/pkg</package.prefix.directory>

        <!-- used to determine if a package has been processed or not -->
        <package.touch.directory>${project.build.directory}/cache</package.touch.directory>

        <!-- classifier and build path are defined by OS profile -->
```

```
        <package.file.classifier>undefined-classifier</package.file.classifier>

        <!-- allow maven default build directory to be overridden
             note that ${project.build.directory} (not this parameter)
             can (should) still be referenced everywhere a full,
             project-specific path is needed. -->
        <alt.build.directory>build/${os.classifier}</alt.build.directory>

        <package.file.extension>tar.bz2</package.file.extension>

        <!-- define extension separately - used as artifact type, for example -->
        <package.file.name>${project.artifactId}-${project.version}</package.file.name>
        <package.file.path>
            ${project.build.directory}/${package.file.name}.${package.file.extension}
        </package.file.path>
        <package.file.classifier>${os.classifier}</package.file.classifier>

        <!-- allow child projects to skip dependency extraction phase.
             this is especially useful for projects with no dependencies -
             there appears to be a bug within the dependencyFilesets
             task such that if there are no deps, it will get *all* dependencies
             that just happen to be installed to the local repository -->
        <skip.dependency.extraction>false</skip.dependency.extraction>
    </properties>
```

2.3 Profiles

Profiles are used in the Bob parent pom to provide the custom build logic unique to cmake builds of C++ projects. In the future many/most of these profiles should be replaced with a more maven conventional custom plugin.

2.3.1 OS-Unix and OS-Windows

The profile used on Unix/Linux.

```
        <profile>
            <id>OS-Unix</id>
            <activation>
                <os>
                    <family>Unix</family>
                </os>
            </activation>
            <properties>
                <!-- TODO: distinguish linux distros! -->
                <os.classifier>linux</os.classifier>

                <cmake.generator>Unix Makefiles</cmake.generator>

                <build.command>make</build.command>
                <build.arguments>-j5</build.arguments>

                <install.command>make</install.command>
                <install.arguments>install</install.arguments>

                <package.unpack.command>tar</package.unpack.command>
                <package.unpack.arguments>-xvjf</package.unpack.arguments>
```

```
            <package.pack.command>tar</package.pack.command>
            <package.pack.arguments>-cvjf</package.pack.arguments>
        </properties>
    </profile>
```

The profile used on Windows

```
    <profile>
        <id>OS-Windows</id>
        <activation>
            <os>
                <family>Windows</family>
            </os>
        </activation>
        <properties>
            <os.classifier>win64</os.classifier>

            <cmake.generator>Visual Studio 10 Win64</cmake.generator>

            <!-- TODO release vs. debug -->
            <build.command>cmake</build.command>
            <build.arguments>--build . --config Release</build.arguments>

            <install.command>cmake</install.command>
            <install.arguments>--build . --target install --config Release</install.arguments>

            <package.unpack.command>cmake</package.unpack.command>
            <package.unpack.arguments>-E tar xvjf</package.unpack.arguments>

            <package.pack.command>cmake</package.pack.command>
            <package.pack.arguments>-E tar cvjf</package.pack.arguments>
        </properties>
    </profile>
```

2.3.2 Aggregator Present

The profile activated when a pom file is in the presence of an *aggregator* pom. This is simply a recognition that the parent folder of the pom file also has a pom file.

```
    <profile>
        <id>aggregator-present</id>
        <activation>
            <file>
                <exists>${basedir}/../pom.xml</exists>
            </file>
        </activation>
        <properties>
            <!-- TODO: consider using a property, always set in the aggregator pom
                 to define the location of that pom. That way, if we nest more than one
                 level deep, this will continue to work - as is, we only support one
                 level of aggregation -->
             <package.prefix.directory>
                 ${basedir}/../${alt.build.directory}/pkg
             </package.prefix.directory>
             <package.touch.directory>
                 ${basedir}/../${alt.build.directory}/cache
```

```
            </package.touch.directory>
        </properties>
    </profile>
```

2.3.3 cmake

This profile defines the default arguments used in each of the cmake configure, build, and install steps. This profile hooks cmake invocations into the process-sources, compile, and prepare-package phases of the lifecycle.

Todo

Skip, why is the cmake test step not also in this profile?

```
    <profile>
        <id>cmake</id>
        <activation>
            <file>
                <exists>${basedir}/CMakeLists.txt</exists>
            </file>
        </activation>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>exec-maven-plugin</artifactId>
                    <executions>
                        <!-- the cmake configure step -->
                        <execution>
                            <id>cmake-configure-build</id>
                            <phase>process-sources</phase>
                            <goals>
                                <goal>exec</goal>
                            </goals>
                            <configuration>
                                <executable>cmake</executable>
                                <workingDirectory>${project.build.directory}</workingDirectory>
                                <arguments>
                                    <argument>-G${cmake.generator}</argument>
                                    <argument>-DPACKAGE_PATH=${package.prefix.directory}</argumer
                                    <argument>-DINSTALL_PATH=${package.install.directory}</argume
                                    <argument>-DCPACK_PACKAGE_VERSION=${project.version}</argumer
                                    <argument>${basedir}</argument>
                                </arguments>
                            </configuration>
                        </execution>

                        <!-- the cmake build step -->
                        <execution>
                            <id>cmake-build</id>
                            <phase>compile</phase>
                            <goals>
                                <goal>exec</goal>
                            </goals>
                            <configuration>
                                <workingDirectory>${project.build.directory}</workingDirectory>
                                <executable>${build.command}</executable>
```

```
                                <commandlineArgs>${build.arguments}</commandlineArgs>
                            </configuration>
                        </execution>

                        <!-- the cmake install step -->
                        <execution>
                            <id>cmake-install-build</id>
                            <phase>prepare-package</phase>
                            <goals>
                                <goal>exec</goal>
                            </goals>
                            <configuration>
                                <workingDirectory>${project.build.directory}</workingDirectory>
                                <executable>${install.command}</executable>
                                <commandlineArgs>${install.arguments}</commandlineArgs>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
```

2.3.4 Unpack Packages

The profile that defines how to unpack package tarballs. This profile hooks into the validate phase of the lifecycle.

Todo

Skip, shouldn't this hook into the *initialize* phase that comes between validate and initialize?

```
    <profile>
        <id>unpack-packages</id>
        <activation>
            <file>
                <exists>${basedir}/CMakeLists.txt</exists>
            </file>
        </activation>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-antrun-plugin</artifactId>

                    <executions>
                        <execution>
                            <id>ant-unpack-packages</id>
                            <phase>validate</phase>
                            <configuration>
                                <skip>${skip.dependency.extraction}</skip>
                                <target>
                                    <!-- Set up a fileset. -->
                                    <dependencyfilesets scopes="compile" types="${package.file.ex

                                    <mkdir dir="${package.prefix.directory}" />
                                    <mkdir dir="${package.touch.directory}" />
```

```
                                <mkdir dir="${package.touch.directory}/tmp" />

                                <!--
                                Unpack into the touch directory as a temp, but only if the ta
                                is newer than our marker. We extract to tmp to make sure we
                                never collide with the touch file name.
                                -->
                                <apply executable="${package.unpack.command}" dir="${package
                                    <arg line="${package.unpack.arguments}" />
                                    <fileset refid="maven.project.dependencies" />
                                    <regexpmapper handledirsep="true" from="^.*/(.*)" to="\1"
                                </apply>

                                <!--
                                Move the unpacked components to their final location, removi
                                the top-level conventional tar directory and the tmp locatio
                                -->
                                <move todir="${package.prefix.directory}" includeEmptyDirs="y
                                    <fileset dir="${package.touch.directory}">
                                        <include name="**/*" />
                                    </fileset>
                                    <cutdirsmapper dirs="2" />
                                </move>

                                <!--
                                Canonicalize the touch path. On Windows, the ant touch task
                                silently removes ".."'s in the middle of a path instead of
                                backing up one level.
                                -->
                                <pathconvert property="package.touch.canonical" targetos="uni
                                    <path location="${package.touch.directory}" />
                                </pathconvert>

                                <!-- update the marker -->
                                <touch verbose="true" datetime="now">
                                    <fileset refid="maven.project.dependencies" />
                                    <regexpmapper handledirsep="true" from="^.*/(.*)" to="${p
                                </touch>
                            </target>
                        </configuration>
                        <goals>
                            <goal>run</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</profile>
```

2.3.5 Create Package Tarballs

The profile that defines how to make package tarballs. This profile hooks into the *package*[1] phase of the lifecycle.

[1] http://docs.python.org/glossary.html#term-package

```
<profile>
    <id>create-package</id>
    <activation>
        <file>
            <exists>${basedir}/CMakeLists.txt</exists>
        </file>
    </activation>
    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <id>package</id>
                        <phase>package</phase>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                        <configuration>
                            <executable>${package.pack.command}</executable>
                            <workingDirectory>${package.install.directory}/..</workingDirecto
                            <commandlineArgs>
                                ${package.pack.arguments} ${package.file.path} ${package.root
                            </commandlineArgs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</profile>
```

2.3.6 Attach Package

Todo

What does this profile do?

```
<profile>
    <id>attach-package</id>
    <activation>
        <file>
            <exists>${basedir}/CMakeLists.txt</exists>
        </file>
    </activation>
    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>build-helper-maven-plugin</artifactId>
                <configuration>
                    <artifacts>
                        <artifact>
                            <file>${package.file.path}</file>
                            <classifier>${package.file.classifier}</classifier>
```

```
                                    <type>${package.file.extension}</type>
                                </artifact>
                            </artifacts>
                        </configuration>
                        <executions>
                            <execution>
                                <goals>
                                    <goal>attach-artifact</goal>
                                </goals>
                            </execution>
                        </executions>
                    </plugin>
                </plugins>
            </build>
        </profile>
```

2.3.7 Build Element

The first thing done here is to define the maven build directory to be the value of the property alt.build.directory. The remaining major sections are for build plugin management, and the definition of the organization and description metadata values.

```
    <build>
        <!-- TODO: change this to be parallel to the project directory e.g. project-work -->
        <directory>${alt.build.directory}</directory>

        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-antrun-plugin</artifactId>
                    <version>1.7</version>
                </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>exec-maven-plugin</artifactId>
                    <version>1.2.1</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-install-plugin</artifactId>
                    <version>2.5</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>2.5</version>
                </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>build-helper-maven-plugin</artifactId>
                    <version>1.8</version>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-project-info-reports-plugin</artifactId>
                    <version>2.7</version>
```

```
36                </plugin>
37                <plugin>
38                    <groupId>org.apache.maven.plugins</groupId>
39                    <artifactId>maven-dependency-plugin</artifactId>
40                    <version>2.8</version>
41                </plugin>
42                <plugin>
43                    <!-- add support for ssh/scp -->
44                    <groupId>org.apache.maven.plugins</groupId>
45                    <artifactId>maven-site-plugin</artifactId>
46                    <version>3.3</version>
47                    <dependencies>
48                        <dependency>
49                            <groupId>org.apache.maven.wagon</groupId>
50                            <artifactId>wagon-ssh</artifactId>
51                            <version>2.5</version>
52                        </dependency>
53                        <dependency>
54                            <groupId>org.apache.maven.wagon</groupId>
55                            <artifactId>wagon-ssh-external</artifactId>
56                            <version>1.0</version>
57                        </dependency>
58                    </dependencies>
59                </plugin>
60            </plugins>
61        </pluginManagement>
62    </build>
63
64    <organization>
65        <name>SRI International</name>
66        <url>http://www.sri.com</url>
67    </organization>
68
69    <description>
70      The parent pom file to use for all maven projects following the
71      Bob conventions.
72    </description>
73</project>
```

CHAPTER

THREE

MAVEN OVERVIEW

3.1 Maven Settings File

A sample maven settings file is shown in *Sample Maven settings.xml*. If you do not have one, copy this file to `~/.m2/settings.xml`. This file is used by maven to hold user specific settings such as repository URLs, private key files, etc.

3.2 Lifecycle

The correspondence between conventional cmake steps and maven steps are:

1. **generate-sources**: is the cmake configure step; `cmake -D... pathToSource`
2. **compile**: is the build step; `make`
3. **test**: is the unit test step; `make test`
4. **prepare-package**: is the generate documentation and install artifacts; `make doc install`
5. **package**: is where the installed artifacts from the previous step are put into a tarball for maven to use in its *install* step; `make tarball`, where the *tarball* target is a clavin replacement of the conventional make package cpack step, which we want to avoid right now because it is too hard to get the cmake package cpack step correct.

The complete list of maven lifecycle phases (obtained executing **mvn** with no arguments) are listed below. The highlighted phases are for the corresponding cmake steps discussed above.

validate
initialize

generate-sources

process-sources
generate-resources
process-resources

compile

process-classes
generate-test-sources
process-test-sources
generate-test-resources
process-test-resources
test-compile

process-test-classes

test

prepare-package

package

pre-integration-test
integration-test
post-integration-test
verify
install
deploy
pre-clean
clean
post-clean
pre-site
site
post-site
site-deploy

3.3 Troubleshooting

Effective POM The "effective" pom is the pom that is actually used when building. The effective pom has all of the variables replaced with their actual value. The effective pom is analogous to the output of the pre-processor in a C/C++ compilation. To see the effective pom use:

```
mvn help:effective-pom
```

Active Profiles mvn profiles are ... To see the active profiles available use:

```
mvn help:active-profiles
```

Starting a new projcect

validate a project's pom:: mvn validate

Show environment:

```
mvn help:env
```

Todo

Skip, please put in a short description of maven profiles above.

3.4 Listings

3.4.1 Sample Maven settings.xml

The contents of the sample maven settings file from `/example-project/settings-m2home.xml` is below:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <settings
```

```
    xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0 http://maven.apache.org/xsd/settings-1
    xmlns="http://maven.apache.org/SETTINGS/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <profiles>
        <profile>
            <id>artifactory-vt</id>
            <repositories>
                <repository>
                    <id>repo-artifactory-vt</id>
                    <name>SRI VT Maven Repository</name>
                    <url>https://artifactory-vt.sarnoff.internal/artifactory/repo</url>
                </repository>
            </repositories>
        </profile>
    </profiles>

    <activeProfiles>
        <activeProfile>artifactory-vt</activeProfile>
    </activeProfiles>

    <servers>
      <!-- set up private key to use when deploying the clavin site -->
      <server>
        <id>clavin.website</id>
        <username>johndoe</username>
        <privateKey>c:/Users/jdoe/Documents/git-private.ppk</privateKey>
        <configuration>
          <sshExecutable>c:\Program Files (x86)\GitExtensions\PuTTY\plink.exe</sshExecutable>
          <scpExecutable>pscp</scpExecutable>
        </configuration>
      </server>
    </servers>
</settings>
```

CHAPTER

FOUR

GIT AND SSH

4.1 Installing git

linux On linux installing **git** is most easily done using `sudo apt-get install git` or `sudo yum install git` for either Ubuntu or CentOS.

windows On windows Git Extensions[1] is a popular packaging of git. Git Extensions comes with a **bash** command interpreter and all of the required **putty** utilities.

smartgit gui client I (pwm) cannot effectively use git via command line only. I have to have a gui client. I use smartgit[2], which has nominal (about $80) license fee but is portable across and linux and may be installed on ALL the computers you use.

4.2 Configuring ssh

Before you can use git, you must also have your ssh keys set up. Charles says you are supposed to create a new private key for every computer account. He has very good instructions on the Git-ssh[3] page on the VT Wiki[4]. I think it is easier and better to reuse mine, so I copy them from an existing `~/.ssh` folder. When you copy those keys around, be sure to remove insecure permissions to the `~/.ssh` folder and its files by using:

```
$ chmod -R g-rwx,o-rwx ~/.ssh
```

For other accounts to be able to **ssh** to this computer without using a password, you need to add your public key(s) to the `~/.ssh/authorized_keys` file.

Details of configuring jenkins and ssh are discussed in *Configuring Jenkins*.

4.3 Tips

4.3.1 git clone of boost-svn

Clone of boost-svn on github

(The content in this subsection came from here[5].)

[1] http://sourceforge.net/projects/gitextensions/files/latest/download
[2] http://www.syntevo.com/smartgithg/download
[3] https://wiki.sri.com/display/VT/Git-ssh
[4] https://wiki.sri.com/display/VT/Home
[5] http://boost.2283326.n4.nabble.com/Live-read-only-GIT-mirrors-of-Boost-trunk-SVN-td4646063.html

So I decided to try out a new Subversion <=> Git mirroring tool, SubGit v2.0 EAP (http://subgit.com/eap/) whose author very kindly gave me a free open source licence and the result is at:

https://github.com/ned14/boost-trunk

All branches and tags, as far as I can tell, are mirrored correctly.

Clone of github boost on git-open

Url: `git-open/scm/3rdparty/boost-trunk` (this has also been softlinked to `boost.git`).

This repo was made by cloning the github ned14/boost-trunk repo with the additional option of `--bare`.

4.3.2 Selecting branches when cloning with git-svn

Warning: git-svn is Fragile
My very limited experience agrees with the comments in the links reference *above* that mention **git-svn** usage is fragile and error prone. I spent over 8 hours trying to `git svn clone` the boost subversion repository. Sometime between 8 and 14 hours (I have to sleep sometimes!), the clone failed and had to be removed to restart. So I gave up git cloning the boost svn repo after I found the ned14/boost-trunk repo on git hub.
While **git svn** did not work for me with a huge svn codebase like boost, it did work when I cloned just the 1.54 release branch of boost. So **git svn** can and does work. So this section may still be helpful if you use a 3rd party project that uses subversion.

(The content in this subsection came from this gist[6] on github.)

If you want to clone an svn repository with git-svn but don't want it to push all the existing branches, here's what you should do.

- Clone with git-svn using the -T parameter to define your trunk path inside the svnrepo, at the same time instructing it to clone only the trunk:

  ```
  git svn clone -T trunk http://example.com/PROJECT
  ```

- If instead of cloning trunk you just want to clone a certain branch, do the same thing but change the path given to -T:

  ```
  git svn clone -T branches/somefeature http://example.com/PROJECT
  ```

This way, git svn will think that branch is the trunk and generate the following config on your .git/config file:

```
[svn-remote "svn"]
        url = https://example.com/
        fetch = PROJECT/branches/somefeature:refs/remotes/trunk
```

- If at any point after this you want to checkout additional branches, you first need to add it on your configuration file:

  ```
  [svn-remote "svn"]
          url = https://example.com/
          fetch = PROJECT/branches/somefeature:refs/remotes/trunk
          branches = PROJECT/branches/{anotherfeature}:refs/remotes/*
  ```

The branches config always needs a glob. In this case, we're just specifying just one branch but we could specify more, comma separating them, or all with a *.

[6] https://gist.github.com/trodrigues/1023167

- After this, issue the following command:

```
git svn fetch
```

Sit back. It's gonna take a while, and on large repos it might even fail. Sometimes just hitting CTRL+C and starting over solves it. Some dark magic here.

- After this, if you issue a git branch -r you can see your remote branch definitions:

```
git branch -r

anotherfeature
```

- Now you can add a local branch which tracks the remote svn branch:

```
git branch --track myanotherfeature remotes/anotherfeature
```

Try not to use the same branch name for the local one if you don't wanna mess it up easily.

4.3.3 git Tips

To see a git revision graph in text use:

```
git log --oneline --decorate --graph
```

which has sample output:

```
git log --oneline --decorate --graph
* 283f667 (HEAD, master) prepare for merge to aspn-super by moving everything to aspn-vfc subdirecto
* f212a14 make destructor signature nothrow exception specification match base class (for centos6 bu
* 76d3d86 corect place to count the number of samples in AirDataTrainer.
* f38601d make the HmfTsmController period_ parameter configurable. the default to 1.0 seconds.
*   5de731e Merge branch 'master' of ssh://git-open.sarnoff.com/scm/vision/aspn-vfc into debug/compas
|\
| *   8672f09 (tag: v2.1.6) Merge branch 'origin/feature/MultipleImu-Te2'
| |\
| | *   83c00a4 Merge branch 'origin/master'
| | |\
| | |/
| |/|
| | *   1b00a25 Merge branch 'origin/feature/MultipleImu-Te2'
```

To get information about a git tag use:

```
git show v2.1.6
```

which produces output:

```
tag v2.1.6
Tagger: Philip Miller <philip.miller@sri.com>
Date:   Wed Oct 9 12:38:41 2013 -0400

infrastructure is in place to support multiple IMUs

commit 8672f097ab911a354d27269ef89f2f097e1ad51b
Merge: 3708494 83c00a4
Author: Philip Miller <philip.miller@sri.com>
Date:   Wed Oct 9 12:26:32 2013 -0400

    Merge branch 'origin/feature/MultipleImu-Te2'
```

CHAPTER

FIVE

CONFIGURING JENKINS

5.1 Install Jenkins

The documentation for Jenkins says that to use stable releases use:

```
wget -q -O - http://pkg.jenkins-ci.org/debian-stable/jenkins-ci.org.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ > /etc/apt/sources.list.d/jenkin
sudo apt-get update
sudo apt-get install jenkins
```

To use more recent releases remove the `-stable` suffix from `debian-stable` in the two http URLs above.

5.2 Configure ssh Keys for the jenkins User

There is a lot of unexplained/misremembered folklore relating to proper configuration of a jenkins build account. Here is what I did for the aspn build.

I first logged into the jenkins account on the computer running the jenkins server using the following command as explained in this Stack Overflow answer[1]

```
# Login as the jenkins user and specify shell explicity,
# since the default shell is /bin/false for most
# jenkins installations.
sudo su jenkins -s /bin/bash
```

This gave me a command prompt for the jenkins user account. Then I generated an empty passphrase key using:

```
ssh-keygen
```

From another terminal session I ssh'ed to `vtbuild-open@git-open` using the credentials for the vtbuild-open account. Then I vi'ed the `~/.ssh/authorized_keys` file, and at the very end of that file, I pasted the contents from the public key from the `~jenkins/.ssh/id_rsa.pub`. To verify that it works, use:

```
ssh vtbuild-open@git-open date
```

When you are able to get this to execute with no errors *AND* no prompts, the jenkins build should be able to use the vtbuild-open account for cloning git repositories. In addition to setting up the ssh keys, you need to configure git for the jenkins account using:

```
git config --global user.email jenkins@aspn
git config --global user.name "Jenkins Build"
```

[1] http://stackoverflow.com/questions/15314760/managing-ssh-keys-within-jenkins-for-git

Verify that git works without prompts using:

```
git clone ssh://vtbuild-open@git-open/scm/vision/vtcmake.git
```

Tip:

Other repositories and build accounts Above I am describing how you get the jenkins public key into the `vtbuild-open` account on the `git-open` server. You will need to repeat these steps for each additional git server and user that you want Jenkins to use. For aspn, you must repeat this for the shared `aspn-user` account on `aspn-sri.sarnoff.com` server. The aspn account is used to clone the gtsam repository from this externally accessible server. For other projects you may also need to repeat it for `vtbuild-itar` account on the `git-itar` server.

5.3 Configure vtbuild-open user account on a slave computer

For jenkins to execute jobs on a slave, the master jenkins account needs ssh privileges to the slave. Following the current VT conventions for Jenkins, this is done by creating a user named `vtbuild-open` on the slave.

After creating the `vtbuild-open` account (I am sure there is a command line way that Doug or Skip would use, but I use the gui for this), put the public key for the jenkins user on the master into the newly created vtbuild-open `~/.ssh/authorized_keys` file following the same instructions as above. (Remember: put the contents of the `~.ssh/id_rsa.pub` from the jenkins user account on the master into the `vtbuild-open:.ssh/authorized_keys` file on the slave.)

Then repeat these steps so that the vtbuild-open account on the slave can clone from git-open without prompts. In summary there are two required special **ssh** permissions that are needed for a Jenkins master/slave relationship. The two verification steps are:

1. in a terminal window for the *jenkins* account on the *vtbuild-open* computer execute:

   ```
   jenkins@vtbuild-open$ ssh vtbuild-open@bso-cent-xyz date
   ```

   to verify that no prompts occur between that session and a remote session for the *vtbuild-open* account on the *bso-cent-xyz* slave computer. This ensures that the jenkins account on the master will be able to run commands as the build account on the slave computer.

2. in a terminal window for the *vtbuild-open* account on the *bso-cent-xyx* computer execute:

   ```
   vtbuild-open@bso-cent-xyz$ ssh vtbuild-open@git-open date
   vtbuild-open@bso-cent-xyz$ git clone ssh::://vtbuild-open@git-open/scm/vision/vtcmake.git
   ```

   and verify that no prompts occur. This ensures that the build account on the slave will be able to clone projects from git-open.

In the verification commands above, I use the notation of `user@computer$` to indicate the shell prompt *$* for the *user* on the *computer*.

5.4 Configuring/Managing the Jenkins Master

Todo

add more details about configuring the jenkins master, slave, build jobs, test jobs, etc.

5.4.1 JDK and Maven

As part of the system configuration, initialize the JDK and Maven settings for the master.

5.4.2 Plugins

5.4.3 Build Jobs

5.4.4 Test Jobs

5.4.5 Documentation Jobs

5.4.6 Slave Nodes

On the Master

Once you have a jenkins installed and running on a master, and a second computer to which the jenkins account has ssh privileges, you can then configure a jenkins node and jobs meant to run on that node.

The slave itself

5.5 Copy a Jenkins Master Configuration

There are at least two ways to copy a jenkins configuration from one computer to another.

1. You can manually copy the jobs (without the builds) and the plugins from the jenkins account and use them in a jenkins account on another computer. This requires you to know where files are.
2. You can use a thinBackup set. Do a "Backup Now" on the source computer and copy the timestamped sub-directory to the destination computer. You will need to edit the backed up file with the url location, so it uses the proper url for the jenkins installation on the destination computer. Point your browser to the jenkins server on the destination computer and go to the Manage Jenkins -> Plugins page and install the thinBackup plugin. Change the ThinBackup settings to point to the folder containing the backup set, and restore the configuration. (I successfully did this when copying a jenkins configuration from a vm on my laptop to a vm on the vtopen vcloud.)

5.6 Disable the Jenkins Service at Startup

```
sudo sh -c "echo 'manual' > /etc/init/jenkins.override"
```

CHAPTER

SIX

CONFIGURATION OF NEW UBUNTU 12.04 COMPUTER ASPN03

In this section we show various apt-get installs of packages used for ASPN development.

6.1 Generally useful packages

```
sudo apt-get install \
   emacs23 \
   gnome-system-tools \
   unrar-free \
   p7zip-full \
   synaptic \
   aptitude
```

6.2 ROS Desktop

Install ROS following the directions on the ROS Wiki[1]

```
sudo sh -c \
   'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.l
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install ros-groovy-desktop-full
```

6.3 xerces-c

```
sudo apt-get install \
   libxerces-c-dev \
   libxerces-c-doc \
   libxerces-c-samples \
   libxerces-c3.1
```

[1] http://www.ros.org/wiki/groovy/Installation/Ubuntu

6.4 gdal

The conventional Ubuntu source for gdal does not install with ROS because it has a conflicting dependency. So I download and build gdal from source instead. Current source for gdal may be obtained from osgeo[2]. Unpack and do an in-source configure and build with:

```
cd gdal-1.10.0
./configure --prefix=/home/pmiller/projects/aspn-super-pkg
make -j12
make install
echo export GDAL_ROOT=/home/pmiller/projects/aspn-super-pkg >> ~/.bashrc
```

If you are not using ROS and want a conventional install of gdal, use:

```
sudo apt-get install libgdal1-dev libgdal-doc gdal-bin python-gdal
```

6.5 Qt4

```
sudo apt-get install \
   qt4-dev-tools \
   qt4-designer \
   qtcreator \
   qtcreator-doc \
   qt4-doc \
   qt4-demos
```

6.6 Eclipse for C++:

Before installing eclipse, you may want to install an up-to-date JDK as described below. You will need a JRE, but the one that comes with Ubuntu should be okay for eclipse. Either way, you may install Eclipse using:

```
sudo apt-get install eclipse-cdt
```

Then try to run the `eclipse` executable. If/when eclipse does not run because of an error with swt, a Stack Overflow answer[3] says to:

```
ln -s /usr/lib/jni/libswt-* ~/.swt/lib/linux/x86_64/
```

(The reason you must run eclipse first is the target directory for this link will not exist until eclipse has been run.)

6.7 Oracle Java JDK with apt-get

PPA installation instructions for the Oracle JDK are found at webupd8[4]. To install:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java7-installer
echo export JAVA_HOME=/usr >> ~/.bashrc
```

[2] http://download.osgeo.org/gdal/CURRENT
[3] http://stackoverflow.com/questions/10970754/cant-open-eclipse-in-ubuntu-12-04-java-lang-unsatisfiedlinkerror-could-not-l
[4] http://www.webupd8.org/2012/01/install-oracle-java-jdk-7-in-ubuntu-via.html

The environment variable `JAVA_HOME` is a conventional way to tell java applications where to get the JRE.

6.8 hdf5

```
sudo apt-get install libhdf5-serial-dev libhdf5-doc hdf5-tools hdfview
```

6.9 Python Goodies

```
sudo apt-get install python-setuptools
sudo easy_install pip
sudo pip install docutils
```

The docutils package contains the sphinx documentation builder used to generate this documentation set.

6.10 Grub-customizer

To get a gui for customizing grub see this blog[5], which has the following instructions:

```
sudo add-apt-repository ppa:danielrichter2007/grub-customizer
sudo apt-get update
sudo apt-get install grub-customizer
```

6.11 Miscellaneous Tips

6.11.1 Customize Windows Shortcut Keys

Ubuntu has something very close to the win7 winkey-right/winkey-left behavior to snap a window to the right or left side of the screen. You may use Ctrl-Alt 1, 3, 9, or 7 to snap to the bottom left, bottom right, top right or top left quadrants of the screen. Then you may maximize vertically, by assigning Super-Up as the windows shortcut key. To create the short cut key, go to System Settings -> Hardware/Keyboard -> Shortcuts -> Windows then set the shortcut for "Maximize window vertically".

6.11.2 Mount Windows Partitions

```
sudo apt-get install ntfs-config
sudo ntfs-config
```

6.11.3 Use Same Packages as Another Computer

To configure a computer with same packages as another computer (from askubuntu[6]:

- Create a backup of what packages are currently installed:

[5] https://launchpad.net/ danielrichter2007/+archive/grub-customizer?field.series_filter=precise
[6] http://askubuntu.com/questions/17823/how-to-list-all-installed-packages)

```
sudo dpkg --get-selections > list.txt
```

- Then (on another system) restore installations from that list:

```
sudo dpkg --clear-selections
sudo dpkg --set-selections < list.txt
```

- To get rid of stale packages:

```
sudo apt-get autoremove
```

- To get installed like at backup time:

```
sudo apt-get dselect-upgrade
```

CHAPTER

SEVEN

CONFIGURING A LINUX (CENTOS/REDHAT) SYSTEM FOR DEVELOPMENT

7.1 EPEL

> $ wget http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm $ rpm -Uvh epel-release-5-4.noarch.rpm

Or simply click on this link: http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm .

7.2 RepoForge

RepoForge[1] is the next generation rpmforge. RepoForge was the only place I could find a reasonably up to date git. The system install of git was 1.7.1, but when I used the *annotate* feature in the new qtcreator, it complained that it needed at least git version 1.7.2. So I added the RepoForge EL 6 x86_64[2] repository and got 1.7.11.3.

7.3 boost

Before building install:

```
sudo yum install bzip2-devel
sudo yum install zlib-devel
```

Build using:

```
./bootstrap.sh --prefix=$HOME/projects/aspn-pkg
./b2 --build-type=complete --layout=tagged -j4 address-model=64 link=static,shared install
```

Todo

installation/building of boost become part of the general documentation of clavin and should not be mentioned in this CentOs section.

[1] http://repoforge.org/use/
[2] http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el6.rf.x86_64.rpm

7.4 7zip

Using the epel repository:

```
sudo yum install p7zip-plugins
```

7.5 xerces-c-dev

Install using:

```
sudo yum install xerces-c-devel
```

7.6 gdal

There are various versions of gdal available from several repositories. At the current time (October 14, 2013), the choices are:

EPEL 1.7.3 (Nov, 2010)

EL GIS[3] 1.9.2 (Oct, 2012), 1.9.1, 1.8.1, 1.8.0, 1.7.3, and 1.7.2

osgeo[4] **source distribution:** 1.10.1 (Aug, 2013)

7.7 doxygen (and graphviz)

The installed doxygen (1.6) is old and does not meet the requirements for generating geographiclib's docuementation. The easiest thing is to uninstall doxygen and install it locally. You may get the source from: here[5]. Then use the following commands:

```
sudo yum remove doxygen
sudo yum install graphviz

cd ~/projects/pkg-src
zcat ~/Downloads/doxygen-1.8.5.src.tar.gz | tar -xvf -
cd doxygen-1.8.5
./configure
make -j6 -l6
sudo make install
```

7.8 Packages Peculiar to Locktight

7.8.1 ACE

Copy the repo file from http://download.opensuse.org/repositories/devel:/libraries:/ACE:/bugfixonly/CentOS_CentOS-5/devel:libraries:ACE:bugfixonly.repo to `/etc/yum.repos.d/OpenSUSE-ACE.repo`

[5] http://www.stack.nl/ dimitri/doxygen/download.html

7.8.2 Add/Remove Software

From Add/Remove Software, Development, make sure the following are checked:

1. Development Libraries
   (a) Development Tools
   (b) Legacy Software Development
   (c) X Software Development

   After that is done, remove boost 1.33.
2. ace-devel
3. boost-devel
4. gdal-devel
5. gsoap-devel
6. openmotif-devel
7. xerces-c-devel

(See note below on configuring boost on CentOs 5.7.)

7.8.3 Qt

Qt is not readily available for Centos 5.7. On Centos 6.3, you can get Qt from the base repository, but it is old. On windows I could not find a 64-bit distribution for VS 2010. So we now have our own Qt source tree on git-open that is a clone of the one on gitorious. Here are the build instructions for Linux:

```
$ git clone ssh://<username>@git-open/scm/vendor/gitorious/qt.git
$ cd qt
$ ./configure

$ make -j8
$ make install

# default configuration of Qt will be install into /usr/local/Trolltech/Qt-4.8.5
#
# the cmake find scripts for Qt use the ``qmake`` that it finds
# first in your path when setting paths to Qt components, so make a
# softlink to qmake to put it early in your path.

$ ln -s /usr/local/Trolltech/Qt-4.8.5/bin/qmake \
        someDirectoryAtTheBeginningOfYourPath
```

7.9 Red Hat Software Collections for new gcc and friends

First install the scl package:

```
sudo yum install scl-utils scl-utils-build
```

Then install the collection itself (this is what includes a new gcc and is described at http://qt-project.org/wiki/Building_Qt_5_from_Git:

```
sudo wget http://people.centos.org/tru/devtools-1.1/devtools-1.1.repo -O /etc/yum.repos.d/devtools-1.
sudo yum install devtoolset-1.1
```

Enable the software collection:

```
scl enable devtoolset-1.1 bash

# Test - Expect to see gcc version 4.7.2 ( * not * gcc version 4.4.7 )
gcc -v
```

7.10 Build qt5 from git

Install missing Qt build dependencies: yum install libxcb libxcb-devel xcb-util xcb-util-devel

Install Red Hat DevTools 1.1 for CentOS-5/6 x86_64 wget http://people.centos.org/tru/devtools-1.1/devtools-1.1.repo -O /etc/yum.repos.d/devtools-1.1.repo yum install devtoolset-1.1

Open new terminal in ~/projects folder and enable devtoolset-1.1 mkdir ~/projects cd ~/projects scl enable devtoolset-1.1 bash

Test - Expect to see gcc version 4.7.2 (* not * gcc version 4.4.7) gcc -v

Git Qt source git clone git://gitorious.org/qt/qt5.git qt5 cd qt5 git checkout stable perl init-repository

Clean and configure # Optional clean is needed if re-configuring git submodule foreach –recursive “git clean -dfx” ./configure -opensource -nomake examples -nomake tests -no-gtkstyle -confirm-license -qt-libpng -no-c++11

If making on multi-core, for example a quad-core,use “make -j 4” make

make install copies to /usr/local/Qt-5.1.2/ # Run as su or using sudo make install

Build Qt Creator export QTDIR=/usr/local/Qt-5.1.2/

Git Qt Creator source cd ~/projects git clone git://gitorious.org/qt-creator/qt-creator.git cd qt-creator

${QTDIR}/bin/qmake -r make

./bin/qtcreator &

CHAPTER

EIGHT

VC10 NOTES

installed gmake (and a bunch of other stuff) using GetGnuWin32 as described at http://getgnuwin32.sourceforge.net/ . Simply

- install GetGnuWin32
- run its `download.bat`
- run its `install.bat`
- put the gnuwin32\bin folder in your path (I put it after git, so that we use git's bash program.)

CHAPTER

NINE

MISCELLANEOUS NOTES ON LINUX

9.1 gcc

9.1.1 predefined macros

This post[1] on stackoverflow provides a very handy tip for getting **gcc** to give you all of the predefined macros that is uses.

```
gcc -E -dM - < /dev/null
```

9.1.2 Version Macros

__GNUC__ major version number

__GNUC_MINOR__ minor version number

__GNUC_PATCHLEVEL__ patch level number

For example, Ubuntu 12.04 comes with gcc 4.6.3 and Centos6 comes with gcc 4.4.7. To distinguish between the two versions, you may use:

```
__GNUC__ == 4 && __GNUC_MINOR__ < 5
```

when you encounter differences between the newer gcc with Ubuntu and the older gcc with Centos6.

9.2 Add Disk to VM

Use the virtual machine software application (either VMWare Workstation or VDirector) to add a new disk to the VM. Then I followed this posting, http://blog.jiwen.info/?p=115, which mostly consists of

1. Create the partition using `fdisk`. For my system I used `fdisk /dev/sdb` since it was my second drive. Ater using `m` for help (i.e., menu), I believe I used the commands `n` for new, and `p` for parition, and `w` for writing.
2. Format the filesystem using `mkfs`. I used `mkfs -t ext3 /dev/sdb1`
3. Create a mount point: `mkdir /disk2`
4. Add this line to /etc/fstab:

   ```
   /dev/sdb1   /disk2   ext3   default 1 2
   ```

5. reboot or just remount using `mount -a`

[1] http://stackoverflow.com/questions/1936719/what-are-the-gcc-predefined-macros-for-the-compilers-version-number

9.3 IPP and MKL

Get a license, download the latest version, and install following the instructions. You will get a warning about installing on an unsupported operating system, but you may ignore that because it seems to work on CentOs 5.7, 5.8, 6.3, and OpenSuse 12.2.

9.4 Install Matlab

Matlab installers may be obtained from `\\rebel\it_installs\Matlab\Linux 64-bit`. For the `R2013a` version, make a local copy of the entire R2013a folder, unzip the glnxa64 installer in place, and then run the setup.

```
cd R2013a
unzip matlab_R2013a_glnxa64_installer.zip
cp matlab_license.dat ~/Documents
sudo ./install
echo export PATH=\$PATH:/usr/local/MATLAB/R2013a/bin >> ~/.bashrc
```

Use the file installation key `02586-43325-41039-17882-41372-47263`.

After installation is complete, be sure to add the matlab bin folder to your `PATH` environment variable.

9.5 VPN

Todo

These instructions for vpn are now out of date. Need to bug Doug Corliss to get a new set of instructions.

Use this link[2] and download and install the snx application. It will likely not run until you also install the following dependent packages:

```
sudo apt-get install libstdc++5:i386
sudo apt-get install libpam-dev:i386
```

Then you may run it with your SRI certificate using:

```
snx -g -s whitebeard.sarnoff.com -c /home/pmiller/Documents/pmiller.p12
```

[2] https://whitebeard.sarnoff.com

CHAPTER

TEN

GLOSSARY

CMAKE_INSTALL_PREFIX The prefix directory to which cmake will install a project.

external package A packages that is *external* to a project under consideration. The notion of *external* is relative to the context in which it is being used. Generally, external packages change infrequently, details of its implementation are not relevant or of interest, may be large in size, etc. The term *external* is used in contrast to *internal package*. An external package will be treated differently from an internal package when it comes to its installed location and the rules or goals used when cleaning. External packages will be installed to a common location and will be cleaned explicitly; not as part of the conventional clean goal, which is considered a *lightweight* clean step.

internal package A package that is *internal* to a project under consideration. The notion of *internal* is used to distinguish with *external package*. Internal packages are installed in a in the same location as project artifacts and are cleaned by the clean goal.

CHAPTER

ELEVEN

TODO

Todo

installation/building of boost become part of the general documentation of clavin and should not be mentioned in this CentOs section.

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\configureCentos.rst, line 37.)

Todo

add more details about configuring the jenkins master, slave, build jobs, test jobs, etc.

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\jenkinsSetup.rst, line 126.)

Todo

These instructions for vpn are now out of date. Need to bug Doug Corliss to get a new set of instructions.

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\linuxNotes.rst, line 109.)

Todo

Skip, why is the cmake test step not also in this profile?

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\bobParentPom.rst, line 90.)

Todo

Skip, shouldn't this hook into the *initialize* phase that comes between validate and initialize?

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\bobParentPom.rst, line 106.)

Todo

What does this profile do?

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\bobParentPom.rst, line 132.)

Todo

Skip, please put in a short description of maven profiles above.

(The *original entry* is located in c:\Users\pmiller\projects\clavin\doc\src\site\sphinx\maven.rst, line 97.)

Symbols

A

C

D

E

G

H

I

J

M

N

P

Q

R

S

V

W

X