Infrastructure Guide

Release 9.21

Diamond Light Source

Jul 16, 2021

CONTENTS

This manual describes the common infrastructure used to check out, develop and build various Eclipse-based products and features.

It is applicable to GDA and Dawn, and related components.

It does not cover the code internals of any specific product (for that, see the applicable Developer Guide).

CHAPTER

ONE

SOFTWARE REQUIRED FOR DEVELOPMENT

This chapter describes the software you need to have installed in order to get started developing a GDA or DAWN product.

1.1 Java JDK

Install Java SE **8** (also known as Java SE 1.8) **JDK** (Java Development Kit).

Note: Be sure to install the JDK, not the JRE (Java Runtime Environment).

Note: We use the Java 8 JVM, but:

- for DAWN 1.x and GDA 8.x, **ALL** code must be Java *7* compliant - do not use constructs that are new in Java 8.
- for DAWN 2.x and GDA 9.you may use constructs that are new in Java 8.

The JDK can be downloaded from Oracle. Alternatively, you could use OpenJDK, but be aware that DLS (Diamond Light Source) only test using the Oracle JDK.

1.2 Eclipse IDE (Eclipse 4.5.2 "Mars SR2") - pre-configured version

This section describes how to use a previously-configured Eclipse IDE. See *Installing and configuring* if you want to do install and configure from scratch.

Instructions are provided for Eclipse 4.5.2 ("Mars SR2").

1.2.1 For general developers

If you use Windows, or if are *not* on the Diamond Light Source (DLS) network, we suggest you download and install a pre-configured Eclipse.

1. Download the most recent Eclipse from alfred.diamond.ac.uk.
2. `untar/unzip` as appropriate for your platform.
3. If you need to specify a network proxy, the best way on Linux is to set environment variables `$http_proxy/$https_proxy/$no_proxy` before starting Eclipse.
   Alternatively, or if you are on Windows, specify the network proxy within Eclipse, at *Window* → *Preferences* → *General* → *Network Connections*
4. At *Window* → *Preferences* → *General* → *Startup and Shutdown*, disable any plugins you do not require.
5. Run *Help* → *Check for Updates* and restart if necessary

1.2.2 For Diamond Light Source developers (Linux)

At DLS, a shared Eclipse install is available via the `module load` system. You can use this as follows:

```
module load java/<version>          # examples: module load java/gdamaster; module load java/gda
module load eclipse/<version>       # example: module load eclipse/442
eclipse &
```

Note that `module load eclipse` script looks to see whether you have a local copy of the requested eclipse version on `/scratch`. If it finds one, it will use that, otherwise it will use the shared install in `/dls_sw/apps/<etc>`.

To find out how to install a local copy of Eclipse, simply issue `module load eclipse/<version>`.

For further information, and a list of what versions are available, see Confluence.

1.3 Eclipse IDE (Eclipse 4.5.2 "Mars SR2") - install and configure

This section describes how to install and configure the Eclipse IDE. See *Installing a previously-configured Eclipse* if you want to use a pre-configured version.

Instructions are provided for Eclipse 4.5.2 ("Mars SR2").

1.3.1 Install and configure Eclipse

Install Eclipse for RCP and RAP Developers

1. Download Eclipse 4.5.2 ("Mars SR2") "Eclipse for RCP and RAP Developers" from http://www.eclipse.org/downloads/.

   (Note: At this stage, use of the Eclipse installer is incompatible with the way we use Buckminster, so don't use it).

   Use of a 64-bit OS (and 64-bit Java and Eclipse) is strongly recommended. We only support DAWN and GDA on 64-bit, so it makes sense that you develop on that.

   The description below assumes Linux, but the same procedure can be used for Windows.

2. Unzip the downloaded file:

```
download=/dls_sw/dasc/eclipse_IDE_downloads/eclipse-rcp-mars-2-linux-gtk-x86_64.tar.gz
arch=64
eversion=452
unpacked=/scratch/eclipse${eversion}
mkdir -v ${unpacked}
tar -zxf ${download} -C ${unpacked}
today=$(date +"%Y%m%d")
mv -v ${unpacked}/eclipse ${unpacked}/eclipse${eversion}_linux${arch}_${today}
```

3. In the unzipped directory, edit `eclipse.ini` (use `gedit ${unpacked}/eclipse${eversion}_linux${arch}_${ & or equivalent):

   Add the following line immediately *above* `-vmargs`:

```
-showLocation
```

   Add the following line immediately *after* `-vmargs`:

```
-Dfile.encoding=UTF-8
```

Change the minimum Java version to 1.8:

```
-Dosgi.requiredJavaVersion=1.8
```

Delete the backup file:

```
rm ${unpacked}/eclipse${eversion}_linux${arch}_${today}/eclipse.ini~
```

4. Start the Eclipse IDE using the following command line options (use a new workspace):

```
# Ensure that the correct Java (Java 8) is on the path
module load java/gdamaster  # at Diamond Light Source
java -version
# If you use a network proxy, and are on Linux, ensure environment variables $http_proxy/$https_
module load global/http_proxy  # at Diamond Light Source
printenv | grep "_proxy="
${unpacked}/eclipse${eversion}_linux${arch}_${today}/eclipse -refresh &
```

5. At *Project*, **de**-select `Build automatically`
6. If you need to specify a network proxy, the best way on Linux is to set environment variables `$http_proxy/$https_proxy/$no_proxy` before starting Eclipse.
   Alternatively, or if you are on Windows, specify the network proxy within Eclipse, at *Window* → *Preferences* → *General* → *Network Connections*
7. Run *Help* → *Check for Updates* and restart if necessary

Once you have installed Eclipse, you will need to install a number of additional features. Be sure to restart Eclipse when prompted after each install.

EGit/JGit (Eclipse IDE Git adapter) update

The version of EGit/JGit bundled with Eclipse Mars SR2 is 4.1.1. Update it to 4.2.0 (or whatever the latest at https://www.eclipse.org/egit/download/ is).

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
http://download.eclipse.org/egit/updates
```

2. At *Help* → *Install New software*, select the site you just added, and install:

```
Eclipse Git Team Provider
   Eclipse Git Team Provider
   Eclipse Git Team Provider - Task focused interface
JGit
   Java implementation of Git
```

Subclipse (Eclipse IDE Subversion adapter)

Note: This addition to the Eclipse IDE is optional, but recommended for developers located at Diamond Light Source (external developers do not require this)

If you need access to source code stored in a Subversion repository, you need to install an Eclipse Subversion adapter. Subclipse (recommended) and Subversive are mutually exclusive Eclipse Subversion adapters; choose exactly one (this manual assumes you selected `Subclipse`).

1. We will install Subclipse 1.10.x, which includes and requires Subversion 1.8.x client (not server) features and working copy format:

```
module load subversion/1.8.13  # at Diamond Light Source, if you need a subversion command line
```

2. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
http://subclipse.tigris.org/update_1.10.x
```

3. At *Help* → *Install New software*, select the site you just added, and install:

```
Subclipse
   Subclipse (Required)
   Subclipse Client Adapter (Required)
   Subversion JavaHL Native Library Adapter
SVNKit
   JNA Library
   SVNKit Client Adapter
   SVNKit Library
```

4. At *Window* → *Preferences* → *Team* → *SVN*, and under *SVN Interface: Client*, select an available client.
   `SVNKit` is the most portable, so if you a creating a shared install, that's probably the most appropriate. It can be changed later if required.
   Whichever client you select, be sure to read the notes on *Subversion security*.

Buckminster

Note: This addition to the Eclipse IDE is REQUIRED for Dawn and GDA development

Buckminster is required for materializing and building, so you need to install it, and its Git and (optionally) Subversion support.
Install at most one of Buckminster `Subclipse` and `Subversive` support, matching the Eclipse Subversion adapter you installed previously.

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
http://download.eclipse.org/tools/buckminster/updates-4.5
```

2. At *Help* → *Install New software*, select the site you just enabled, and install:

```
Buckminster
   Buckminster - Core
   Buckminster - Git Support
   Buckminster - Maven Support
   Buckminster - PDE support
Buckminster SVN Support (Subclipse)
   Buckminster - Subclipse support  # optional, only select if you installed Subclipse in the pr
```

PyDev

Note: This addition to the Eclipse IDE is optional, but recommended for Dawn and GDA development

PyDev is recommended for development work with Python and Jython.

1. At *Help* → *Eclipse Marketplace*, find "PyDev" and install these items:

```
PyDev - Python IDE for Eclipse
   PyDev for Eclipse
```

Spring IDE

Note: This addition to the Eclipse IDE is optional, but recommended for GDA development

Spring IDE is recommended for development work with Spring configurations. The Spring IDE is a a subset of the larger Spring Tool Suite.

1. At *Help* → *Eclipse Marketplace*, find "Spring IDE Eclipse 3.7.3.RELEASE" and install the required components

FindBugs

Note: This addition to the Eclipse IDE is optional, but recommended for Dawn and GDA development

FindBugs uses static analysis to look for bugs in Java code.

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
http://findbugs.cs.umd.edu/eclipse/
```

2. At *Help* → *Install New software*, select the site you just enabled, and install:

```
FindBugs
   FindBugs feature
```

ObjectAid UML Explorer

Note: This addition to the Eclipse IDE is optional, but recommended for Dawn and GDA development

ObjectAid UML Explorer is a code visualization tool for the Eclipse IDE.

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
http://www.objectaid.net/update
```

2. At *Help* → *Install New software*, select the site you just enabled, and install:

```
ObjectAid UML Explorer
   ObjectAid Class Diagram
```

vogella e4 tools

Note: This addition to the Eclipse IDE is optional, but is being trialled to see if it is useful for Dawn and GDA development

vogella e4 tools provides e4 tools for Eclipse RCP development.

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
https://dl.bintray.com/vogellacompany/e4tools-mars
```

2. At *Help* → *Install New software*, select the site you just enabled, and install all available items

Gonsole

Note: This addition to the Eclipse IDE is optional, but is being trialled to see if it is useful for Dawn and GDA development

Gonsole provides a Git Console for the Eclipse IDE

1. At *Window* → *Preferences* → *Install/Update* → *Available Software Sites*, enable (adding first if necessary):

```
https://rherrmann.github.io/gonsole/repository/
```

2. At *Help* → *Install New software*, select the site you just enabled, and install:

```
Gonsole - Git Console for Eclipse
   Gonsole - EGit Integration
   Gonsole - Git Console for Eclipse
```

1.3.2 Publish the Eclipse IDE

Note: This step only needs to be done by the build engineer.

You should install and configure the Eclipse IDE for both Linux (64-bit) and Windows (64-bit).

Once you have installed and configured the Eclipse IDE, zip it and copy to a shared network drive, so that developers can take a local copy.

1. Restart the IDE with the *-clean* option:

```
${unpacked}/eclipse${eversion}_linux${arch}_${today}/eclipse -clean -refresh &
```

2. At *Window* → *Preferences* → *General* → *Startup and Shutdown* → *Workspaces*

   Remove all recent workspaces
   Change *Number of recent workspaces to remember* to 10

3. `tar/zip` the distribution:

```
# environment variables that are probably already defined
arch=64
eversion=452
unpacked=/scratch/eclipse${eversion}/
today=$(date +"%Y%m%d")
# check environment and publish
echo "eversion=${eversion} unpacked=${unpacked} arch=${arch} today=${today}"
packed=/scratch/eclipse${eversion}_linux${arch}_${today}.tar.gz
cd ${unpacked}/
find eclipse${eversion}_linux${arch}_${today}/ -name ".keyring" -print -delete
tar -zcf ${packed} eclipse${eversion}_linux${arch}_${today}/
cp -pvi ${packed} /dls_sw/dasc/eclipse_IDE_Diamond/
ls -la /dls_sw/dasc/eclipse_IDE_Diamond
```

4. install the distribution into the `modules` system:

```
tarball=/dls_sw/dasc/eclipse_IDE_Diamond/eclipse${eversion}_linux${arch}_${today}.tar.gz
install=/dls_sw/apps/eclipse
tar -zxf ${tarball} -C ${install}
java -version
${install}/eclipse${eversion}_linux${arch}_${today}/eclipse -initialize
chmod -R a-w ${install}/eclipse${eversion}_linux${arch}_${today}
gedit ${install}/HISTORY.txt
gedit /dls_sw/apps/Modules/modulefiles/eclipse/<whatever>
```

1.4 Buckminster Headless

A headless version of Buckminster (*i.e.* a version that can be run from the command line) is available. This is separate from the version that you install into your Eclipse IDE.

Buckminster Headless is optional, but having it available means that you can use Buckminster from scripts (for example, in automated testing).

1.4.1 Using a previously-configured headless Buckminster

At DLS headless Buckminster has been installed and can be accessed using one of the following commands:

```
module load java/gdamaster          # or java/gda95 etc
module load buckminster/gdamaster   # or buckminster/gda95 etc
```

ALTERNATIVELY

1.4.2 Install and configure headless Buckminster

Install Buckminster Headless with the extra features required for development.

1. install Buckminster Headless by using the script buckminster_headless_install.sh (Linux) or buckminster_headless_install.bat (Windows)

Note that you *will* need to modify these scripts before using them at your site.

1.5 pewma.py

`pewma.py` is a Python script that let you run some or all of the **setup**, **materialize**, **build** and **create product** steps from the command line.

pre-requisites: Python 2.7+, and Buckminster Headless.

1.5.1 Install pewma.py

Simply download pewma.py. If you need to do this using the Linux command line, try:

```
wget --tries=1 --timeout=5 --no-cache "https://alfred.diamond.ac.uk/buckminster/software/pewma.p
```

If a network proxy is required, make sure it is defined first (i.e. `$http_proxy` is set).

1.5.2 Run pewma.py

Help text is available by specifying the `--help` option. For usage examples, see the Reference Guide.

```
pewma.py --help
Usage: pewma.py [options] action [arguments ...]

For more information, see the Infrastructure guide at https://alfred.diamond.ac.uk/documentation

Options:
  Workspace options:
    -w <dir>, --workspace=<dir>               Workspace location (default: (None))
    --delete                                  First completely delete current workspace/ and wor
    --recreate                                First completely delete current workspace/, but ke

  Materialize options:
    --directories.groupname=<groupname>       Linux group to set on directories that are created
    -l <location>, --location=<location>      Download location ("diamond" or "public")
    -k <path>, --keyring=<path>               Keyring file (for subversion authentication)
    --materialize.properties.file=<path>      Properties file, relative to workspace if not absc
                                              materialize-properties.txt)
    --maxParallelMaterializations=<value>     Override Buckminster default
    --maxParallelResolutions=<value>          Override Buckminster default

  Get-branches-expected options:
    --cquery.branches.file=<path>             Report file, relative to current directory if not
                                              cquery-branches-file.txt)

  Build/Product options:
    --scw, --suppress-compile-warnings        Don't print compiler warnings
    --assume-build                            Skip explicit build when running "site.p2" or "pro
    --recreate-symlink                        Create or update the "client" symlink to the built
    --buckminster.properties.file=<path>      Properties file, relative to site project if not a
                                              filenames looked for in order: buckminster.propert
                                              buckminster.beamline.properties)
    --buckminster.root.prefix=<path>          Prefix for buckminster.output.root and buckminster

  Test/Corba options:
    --include=<pattern>,<pattern>,...         Only process plugin names matching one or more of
    --exclude=<pattern>,<pattern>,...         Do not process plugin names matching any of the gl
    --GDALargeTestFilesLocation=              Default: /dls_sw/dasc/GDALargeTestFiles/
```

```
  General options:
    -D key=value                              Pass a system property to Buckminster or Ant
    -J key=value                              Pass an additional JVM argument
    -h, --help                                Show help information and exit
    -n, --dry-run                             Log the actions to be done, but don't actually do
    -s <path>, --script-file=<path>           Script file, relative to workspace if not absolute
    -q, --quiet                               Be less verbose
    --log-level=<level>                       Logging level (default: INFO)
    --skip-proxy-setup                        Don't define any proxy settings

  Git options:
    -p, --prefix                              Prefix first line of git command output with the r

Actions and Arguments:
    setup [<category> [<version>] | <cquery>]
      Set up a new workspace, with the target platform defined, but otherwise empty
      (parameters are the same as for the "materialize" action)
    materialize <component> [<category> [<version>] | <cquery>]
      Materialize a component and its dependencies into a new or existing workspace
      Component can be abbreviated in many cases (eg just the beamline name is sufficient)
      Category can be one of "gda/ida/dawn/none/training/gda-training"
      Version defaults to master
      CQuery is only required if you need to override the computed value
    get-branches-expected <component> [<category> [<version>] | <cquery>]
      Determine the CQuery to use, and return from it a list of repositories and branches
    gerrit-config
      Switch applicable repositories to origin Gerrit and configure for Eclipse
    git <command>
      Issue "git <command>" for all git clones
    clean
      Clean the workspace
    bmclean <site>
      Clean previous buckminster output
    build
      [alias for buildthorough]
    buildthorough
      Build the workspace (do full build if first incremental build fails)
    buildinc
      Build the workspace (incremental build)
    target
      List target definitions known in the workspace
    target path/to/name.target
      Set the target platform for the workspace
    sites
      List the available site projects in the workspace
    site.p2 <site>
      Build the workspace and an Eclipse p2 site
      Site can be omitted if there is just one site project, and abbreviated in most other cases
    site.p2.zip <site>
      Build the workspace and an Eclipse p2 site, then zip the p2 site
      Site can be omitted if there is just one site project, and abbreviated in most other cases
    product <site> [ <platform> ... ]
      Build the workspace and an Eclipse product
      Site can be omitted if there is just one site project, and abbreviated in most other cases
      Platform can be something like linux64/mac64/win64/all (defaults to current platform)
    product.zip <site> [ <platform> ... ]
      Build the workspace and an Eclipse product, then zip the product
    tests-clean
```

```
    Delete test output and results files from JUnit/JyUnit tests
junit-tests
    Run Java JUnit tests for all (or selected) projects
jyunit-tests
    Runs JyUnit tests for all (or selected) projects
all-tests
    Runs both Java and JyUnit tests for all (or selected) projects
corba-make-jar
    (Re)generate the corba .jar(s) in all or selected projects
corba-validate-jar
    Check that the corba .jar(s) in all or selected plugins match the source
corba-clean
    Remove temporary files from workspace left over from corba-make-jar
```

1.6 Python

A number of scripts used for development are written in Python, and require that Python 2.6 or Python 2.7 (but not Python 3.0+) be installed, and able to be invoked from the command line via the **`python`** command.

- **On Linux, a suitable version of Python is already installed with most recent distributions.** Distributions that do *not* include a suitable version include Red Hat Enterprise Linux 5, and CentOS 5. On these distributions, you will need to download and install a more recent version of Python. Source tarballs are available from the Python website. Alternatively, a suitable RPM may be available.
- **On Windows, you will need to download and install Python.** Windows installers are available from the Python website.

1.6.1 Additional Python packages required

CHAPTER

TWO

ACCESSING SOURCE CODE REPOSITORIES AND P2 SITES

This chapter describes how to set up access to the various source code repositories and p2 sites that you require.

2.1 Locations and Access

2.1.1 SSH tunnels

If you have a DLS FedID, but are not connected directly to the DLS network, you should set up a tunnel to the DLS network using an SSH client (PuTTY for Windows, SSH for Linux).

Tunnelling using PuTTY (Windows)

On Windows, we recommend PuTTY, a Free Telnet/SSH Client

- Set *Session* → *Host Name* to `nx-staff.diamond.ac.uk` (or as advised)
- Set *Session* → *Port* to `22`
- Set *Connection* → *Seconds between keepalives* to `300` or some suitable value
- Set *Connection* → *Data* → *Auto-logon username* to your FedID
- Go to *Connection* → *SSH* → *Tunnels* and define two tunnels (using whatever local port numbers you want):
  - Set *Source port* to `5022`, *Destination* to `dasc-git.diamond.ac.uk:22` and click *Add* (`5022` is user-selected)
  - Set *Source port* to `5080`, *Destination* to `dawn.diamond.ac.uk:80` and click *Add* (`5080` is user-selected)
- Go to *Session*, enter a suitable name in the *Saved Sessions* box, and click *Save*
- Click *Open* to actually run the session and create the tunnels. You will be prompted for the password to your FedID.

Tunnelling using SSH (Linux)

On Linux, the following command will establish two tunnels (using whatever local port numbers you want):

- `ssh -L 5022:dasc-git.diamond.ac.uk:22 -L 5080:dawn.diamond.ac.uk:80 <FedID>@nx-staff.diamond.ac.uk` (`5022` and `5080` are user-selected)
- You will be prompted for the password to your FedID

2.1.2 Locations

Repositories and p2 sites are hosted and/or mirrored in a number of locations:

DLS Subversion server

- can be accessed from anywhere
- access requires authorisation (authentication is with a DLS FedID and password)
- is not mirrored externally

DLS Git server (Gitolite)

- can only be accessed when connected to the DLS network, either directly or using SSH tunnels (PuTTY for Windows, SSH for Linux)
- requires authorisation (authentication is with a public/private keypair)
- some components are mirrored on GitHub

GitHub Git server - http://github.com/

- can be accessed from anywhere
- access can be anonymous, or optionally authenticated (is with a public/private keypair)

DLS p2 server - https://alfred.diamond.ac.uk/

- can be accessed from anywhere

2.2 Subversion

Note: Recent versions of Eclipse implement something called "Secure Storage". We have not yet investigated this.

2.2.1 Diamond Light Source Subversion server

The DLS Subversion server (https://svn.diamond.ac.uk/subversion/gda/) can be accessed from outside the Diamond network, but requires authorisation. Anonymous access is not supported.

Once you have been are authorised, you will be issued a Diamond FedID (username) and password, which is required to access the repository.

2.2.2 Subversion security

You can check which SVN interface client is used by Eclipse at *Window* → *Preferences* → *Team* → *SVN*, under *SVN Interface: Client*.

There are (at least) two possible locations where subversion credentials can be cached.

1. The user's private runtime configuration area. This location is used by the subversion command line client `svn`, and by Eclipse if its SVN interface client is `JavaHL`. The default location (usually unchanged) is:

```
~/.subversion/auth/svn.simple/              on Unix-like systems
%APPDATA%\Subversion\auth\svn.simple\       on Windows
```

> **Warning:** You should ensure that the `auth` directory and its contents can only be read by you

```
chmod go-rwx -R ~/.subversion/auth/          # make all files readable by the owner only
```

2. In the Eclipse keyring. This location is used by Eclipse if its SVN interface client is `SVNKit`. The default location is:

```
${ECLIPSE_HOME}/configuration/org.eclipse.core.runtime/.keyring
```

> **Warning:** If you are using `SVNKit`, do not use the default location for the Eclipse keyring, since it is likely to be readable by others. You can change the location of the Eclipse keyring by starting Eclipse with the *-keyring* option.

```
/path/to/eclipse -keyring /path/to/.keyring  # the .keyring file will be created if it does not
chmod go-rwx /path/to/.keyring               # do immediately the .keyring file is created, befo
```

If you need to delete cached credentials, simply delete the `auth/svn.simple/` directory, or the `.keyring` file, as appropriate.

See also:

Subversion Subversion project home page

The Subversion Book Free, comprehensive book, viewable online or as a PDF to download

2.3 Git

2.3.1 Diamond Light Source Git server

The DLS Git server cannot be accessed from outside the Diamond network, and requires authorisation. Anonymous access is not supported. Access is controlled by SSH keys (using Gitolite).

Once you have been are authorised, you will need to send your public SSH key(s) to the DLS Git repository administrator (there are no security problems with publishing your public key(s); that's why they're called *public*).

Once you have been registered with the DLS Git server, you can see what repositories you have access to, by issuing the following:

```
ssh dascgitolite@dasc-git.diamond.ac.uk info  # issue command exactly as it appears, don't change the
                                              # if you are prompted for a password, there is an erro
```

2.3.2 GitHub server

Github can be accessed from anywhere, and authentication is optional.

Access is controlled by SSH keys.

2.3.3 Creating an SSH keypair

The simplest cross-platform way to generate an SSH keypair is from within Ellipse itself, at *Window → Preferences → General → Network Connections → SSH2 → Key Management*. Be careful not to overwrite any existing keys that you are using.

There are many instructions on the web for setting up an SSH key pair from the command line:

- One set is Github's instructions. Follow the *Set Up SSH Keys* section, stopping when you come to the step *Add your SSH key to GitHub* (you don't actually need to set up a GitHub account, unless you want to).
- DLS users might also want to look at the relevant Diamond Intranet page, which includes instructions on using a key pair to enable SSH between Diamond machines without entering your password.

Security Notes

When you generate a new key pair, use RSA encryption in preference to DSA. It is possible to have multiple key pairs, each one dedicated for a specific service or role.

You must **keep your private key(s) absolutely secure**. The private key file (default name: `~/.ssh/id_rsa`) must be readable only by you.

If need to access the Git repositories from more than one location:

- You don't need to copy your private key from one machine to another; if you do this, make sure the copy is done in a secure manner. alternatively, generate a new key pair at each location.
- Provide a copy of each public key to the DLS Git repository administrator. Multiple keys for one user are ok.

It is normally recommended that you protect your private key file with a **passphrase**. However, note that command line Buckminster cannot materialize from Git repositories if our private key has a passphrase.

- Without a passphrase, your private key file on disk will be all an attacker needs to gain access to any machine configured to accept that key.
- *Do not forget your passphrase.* There is no way to recover it.
- (Linux) This command will list all your private key files that do *not* have a passphrase:

```
find ~/.ssh -type f -exec grep -l "BEGIN RSA PRIVATE KEY" {} \; | xargs -i grep -L "Proc-Type" {
```

- (Linux) You can add/change/remove the passphrase for existing private key file (this does not change your public key):

```
ssh-keygen -p -f ~/.ssh/id_rsa  # the -f option specifies the name of your private key file
```

See also:

Eclipse EGit User Guide The official guide to EGit (Eclipse integration with git), including a section on github

2.4 Cross-platform considerations

Developers can use a variety of platforms such as Linux and Windows. This section describes how differences between the platforms are handled.

2.4.1 Repository rules

Our rules for end-of-line character sequences, and text file encoding, are as follows:

Warning: Text files, when checked in to the repository, MUST use LF line endings only.

Warning: Text files, when checked in to the repository, MUST be Unicode with UTF-8 encoding.

A daily build job checks that all checked-in text files conform to these rules.

Note: Properties files (*.properties) are ISO-8859-1 by definition (see the docs for the java.util.Properties class). We use Apache Commons Configuration to read our property files (rather than java.util.Properties), but it uses the same default encoding.

2.4.2 How to ensure compliance

1. Text files checked out from the repository.
   If you only edit the file with Eclipse, the existing (correct) EOL and encoding will be observed, so no specific action is required.
   If you edit the file with some other editor, you need to check what that editor does.
2. New text files created by Eclipse, and then checked in.
   Before creating any new text files, make sure that Eclipse is set to create text files with the correct attributes.
   This needs to be set separately for each workspace - if you use the template workspace, this is already set up for you.
3. Text files created outside Eclipse, and then checked in.
   You need to manage this yourself.

git users (all platforms) should read http://help.github.com/line-endings/, and set:

```
git config --global core.autocrlf input
```

Linux users can check line ending characters and encoding by using the **`file`** command (it's correct most of the time), and change it using the **`dos2unix`** command:

```
$ file Version.java
Version.java: UTF-8 Unicode English text, with CRLF line terminators
$ dos2unix Version.java
dos2unix: converting file Version.java to UNIX format ...
$ file Version.java
Version.java: UTF-8 Unicode English text
```

With some care, you can easily fix up all the invalid line endings within a directory:

```
bad_text_pattern="text, with\( very long lines, with\)\? CRLF"
# list the files with incorrect file endings
find . -type d -name ".svn" -prune -false -o -type f -exec file {} \; | grep "${bad_text_pattern}" |
# fix the files with incorrect file endings
find . -type d -name ".svn" -prune -false -o -type f -exec file {} \; | grep "${bad_text_pattern}" |
# validate the result before checking in, obviously!
```

CHAPTER

THREE

CHECKING OUT SOURCE CODE

This chapter describes how to check out source code for a GDA OR Dawn product. Buckminster is used to resolve dependencies.

3.1 Set up a new Eclipse workspace

This section describes how to create and configure an empty Eclipse *workspace* (the next section describes how to download the source code into the workspace).

3.1.1 Option 1 of 3: Download and use a template workspace (script method)

ALTERNATIVELY

3.1.2 Option 2 of 3: Download and use a template workspace (manual method)

1. Browse https://alfred.diamond.ac.uk/buckminster/templates/ and download the applicable `template_workspace_`*`version`*`.zip`
   If you are developing on DAWN 1.x or GDA 8.x, use the highest *2.x* version (this specifies Java 1.7 compliance)
   If you are developing on DAWN 2.x or GDA 9.x, use the highest *3.x* version (this specifies Java 1.8 compliance)
2. Unzip the downloaded `template_workspace_`*`version`*`.zip` into a new, empty directory
3. Start your Eclipse IDE, specifying the new directory as your workspace

ALTERNATIVELY

3.1.3 Option 3 of 3: Configure the workspace manually

This is described in Confluence at Creating the template workspace for GDA or Dawn developers

3.2 Materializing a workspace using the Eclipse IDE

This section describes how to download the source code into a workspace, using the Eclipse IDE (the previous section describes how to create and configure the workspace).

Warning: You **must** set the target platform correctly, as described in the previous section. The default target (the running platform) must not be used.

3.2.1 Materialize the target platform and code base

Set-up for users not directly connected to the DLS network.

- Define SSH tunnels to the DLS hosts (instructions *here*), and make a note of the local port numbers.
- Before materializing the CQUERY (Component Query), go to the *Properties* tab and add two new keys (using the same port numbers as when you created the SSH tunnels):
  - `diamond.dascgit.host.port=localhost:5022`
  - `diamond.p2.host.port=localhost:5080`

Check out the target platform and code base

1. ***File*** → ***Open a Component Query*** **and enter the appropriate URL (see the Reference Card)**

   Set the *Component Name* field to the value for the parent component (see *Buckminster components* for a list of possible values)

   Set the *Component Type* field to the value for the parent component, if you know it

   Change any other values on the *Properties* tab as required (see *Buckminster properties*). You need to add property key *github.authentication* and its value to *anonymous* for org.dawb in dawn-common.git.

2. **Click *Resolve and Materialize*. You can close the CQuery after materialization completes.**

   This downloads the specified component and all its dependencies to the Eclipse workspace, and imports them as Eclipse projects.

   The target platform components will be materialized in the `tp` project in the workspace.

Notes

If Eclipse prompts you to *Enter Username and Password* for `https://svn.diamond.ac.uk`, you must select the *Save Password* option; otherwise you will be prompted with repeated requests for your credentials. For more information on where your password is saved, and how to keep it secure, see *Subversion security*.

3.3 Materializing a workspace using Headless Buckminster

This section describes how to create and configure a single workspace, using the command line. Prerequisite: you need install and configure *Buckminster Headless*.

Coming soon!

3.4 Effective Buckminster

See also:

Eclipse Buckminster, The Definitive Guide The *BuckyBook* (PDF download, 286 pages)

3.4.1 Buckminster components

When you run a Buckminster CQUERY, you specify the name of the component you wish to materialize. Buckminster will materialize the specified component, *and* all the components which it depends on, and then all the components they depend on, recursively, until all dependencies can be located.

Valid component names in a CQuery include (but are not limited to):

documentation.gda, documentation.diamond ***(component type=buckminster)***

Eclipse projects containing the source (markup) for the documentation.
You need this if you want to make changes to the manuals - see *Writing and Publishing Documentation*.

diffcalc ***(component type=buckminster)*** Eclipse project containing the Diffcalc software, and source (markup) for the Diffcalc documentation.

example-config ***(component type=buckminster)*** Parent component for Eclipse projects containing the configuration and all required source for the example GDA beamline.

GDALargeTestFiles ***(component type=buckminster)*** Eclipse project containing large files used in GDA testing.

thirdparty ***(component type=buckminster)*** Eclipse project containing the static target platform.

uk.ac.diamond.sda.site ***(component type=eclipse.feature)*** Parent component for Eclipse projects containing the SDA application and all its dependencies.

xx*-config** ***(component type=buckminster)

Parent component for Eclipse projects containing the configuration and all required source for a GDA beamline.
Examples: `example-config`, `scisoft-config`, `i20-config`

Any component name can be specified, provided there is an entry for it in the RMAP (Resource Map) which the CQuery points to.

Overriding what is materialized

By default, Buckminster will not materialize anything unless *all* the dependencies of the requested component can be materialized.

How to materialize just one component

Occasionally, you may want to materialize a single component, but not its dependencies. This can be done by editing the CQuery as follows:

1. On the *Advisor Nodes* tab, add a new entry, with a *Name Pattern* matching the single component you require. Move the new entry to be the first (top) entry.
2. On the *Advisor Nodes* tab, add a second entry, with a *Name Pattern* of .*, and select the *Skip Component* option. Move the new entry to be the second entry.
3. Click *Resolve and Materialize*

How to handle missing components

If dependencies are missing, either remove the dependencies (probably from a `feature.xml` or `MANIFEST.MF`), or update the RMap so that the dependency can be found.

As an work-around, you can specify that missing dependencies be skipped, and Buckminster will then materialize the available components. Of course, since the missing dependencies will not be present in the workspace, you may get compilation or runtime errors.

Skipping specific dependencies can be done by editing the CQuery as follows:

1. On the *Advisor Nodes* tab, add a new entry, with a *Name Pattern* matching one or more of the missing dependencies, and select the *Skip Component* option.
   Move the new entry to be the first (top) entry.
2. Repeat for any other missing dependencies.
3. Click *Resolve and Materialize*

Continuing materializing when there are errors

Select the CQuery option *Continue on error* (located at the bottom of the editor window).

3.4.2 Buckminster properties

When you open a Buckminster CQuery, the *Properties* tab contains a number of settings you can add or change. The default values are normally appropriate.

Property	Valid values	Meaning
Repository_branch		Specifies the version control branch for which components are downloaded
include_source	yes	Retained for backward compatibility DO NOT CHANGE
option_Diamond_components_origin	diamond opengda	Download components from DLS internal p2 server DEFAULT Download components from openGDA p2 server
option_Diamond_subversion_access	true false	Check out source from DLS repository DEFAULT Download source from openGDA server (not all items available)
option_Eclipse_origin	diamond eclipse	Download Eclipse components from DLS internal p2 server DEFAULT Download Eclipse components from Eclipse Foundation website
option_Eclipse_release	helios indigo	Target platform uses Eclipse Helios (3.6.2) DEFAULT Target platform uses Eclipse Indigo (3.7) UNSUPPORTED

3.5 Troubleshooting Buckminster problems

3.5.1 Errors and Warnings

WARNING [0072] : Component request org.junit:osgi.bundle is in conflict ...

This error message is displayed during workspace materialization, and can be ignored:

```
WARNING [nnnn] : Component request org.junit:osgi.bundle/[4.8.1.v4_8_1_v20100427-1100,4.8.1.v4_8_1_v2
                 request org.junit:osgi.bundle/[3.8.2.v3_8_2_v20100427-1100,3.8.2.v3_8_2_v20100427-11
```

It occurs because the `org.eclipse.jdt` feature depends on two different versions of `org.junit`, namely JUnit 3 and JUnit 4.

Resource '/.buckminster/tp' is not local

This error message appears at the end of materialization (during the refresh). It is benign and can be ignored (see https://bugs.eclipse.org/bugs/show_bug.cgi?id=310850):

```
Problems during metadata refresh
E Resource '/.buckminster/tp' is not local
```

3.5.2 Troubleshooting techniques

These techniques may assist in resolving Buckminster problems:

Problems in the Eclipse IDE

1. **Before re-executing a Buckminster action, clear any previous error messages to avoid confusion**

   Right-click in the *Error Log* view, and select *Delete Log*
   Right-click in the *Console* view, and select *Clear*
   Right-click in the *Console* view, and select *Remove All Terminated*

2. **Refresh all Buckminster metadata (sometimes this becomes stale)** Go to *Window* → *Preferences* → *Buckminster*, and click *Clear URL cache* and *Refresh Meta-data*

3. Restart Eclipse (certain workspace changes are not recognised by Buckminster until Eclipse is restarted)

4. **Increase the logging level**

   Go to *Window* → *Preferences* → *Buckminster*, and set *Console logger level:* to *DEBUG*
   Go to *Window* → *Preferences* → *Buckminster* → *Console*, and deselect *Limit console output*

5. **Disable parallel materialization**

   By default materialization is performed in multiple threads, which makes the logging output difficult to read
   Go to *Window* → *Preferences* → *Buckminster*, and set *Max number of parallel materializations* to *1*
   Go to *Window* → *Preferences* → *Buckminster*, and set *Maximum number of resolver threads* to *1*

Problems with headless Buckminster

1. Increase the logging level

CHAPTER

FOUR

SOFTWARE DEVELOPMENT

This chapter describes software development tasks for a GDA or Dawn product.

4.1 Running JUnit tests

Good software development practice suggests that the test suite be run before committing code. This section describes how you can do that.

4.1.1 Prerequisite: get large test files

Some JUnit tests reference a directory of large test files (`GDALargeTestFiles`) that are not saved in any of the Git repositories. Note that, despite the name, these files are used in both Dawn and GDA testing.

Developers running JUnit tests at DLS will do not need to do anything to get these files. External developers will need to get a copy of the files (method to be described).

4.1.2 Running JUnit test(s) from the IDE using Run Configurations

Run a single JUnit test with an on-the-fly Run Configuration

Note: This method is NOT RECOMMENDED! It does not work for tests that use native libraries or `GDALargeTestFiles`.

- In the Package or Navigator view, right-click on the class name.
- Select *Run As → JUnit Test*

Run all the JUnit tests in a project, using a pre-defined Run Configuration

- In the Package or Navigator view, right-click on the class name.
- Select *Run As → JUnit-<project-name>*

Run a single JUnit test in a project, using a pre-defined Run Configuration

sss

4.1.3 Run JUnit tests from the IDE using a pre-defined Run Configuration

Use the Eclipse IDE, running the tests using a JUnit launcher (not available for all projects):

> **Warning:** This method currently doesn't work for tests that use `GDALargeTestFiles`. To be fixed soon.

- Right-click on `JUnit-<project-name>.launch` at the root of the project directory
- Select *Run As → JUnit-<project-name>*

Use the Eclipse IDE, running the tests using an Ant script:

> **Warning:** This method currently doesn't work for tests that use `GDALargeTestFiles`. To be fixed soon.

- Right-click on `releng.ant` at the root of the project directory
- Select *Run As → Ant Build...*
- On the *Targets* tab, select `junit-tests` and *Run*

OR

Use the Eclipse IDE, running the tests using a JUnit launcher (not available for all projects):

> **Warning:** This method currently doesn't work for tests that use `GDALargeTestFiles`. To be fixed soon.

- Right-click on `JUnit-<project-name>.launch` at the root of the project directory
- Select *Run As → JUnit-<project-name>*

OR

Use the command line and `pewma.py` (requires Ant version 1.8+):

```
if [[ "$(uname -n)" == *diamond.ac.uk ]]; then
    module load ant
fi
ant -version

# optionally use tee and grep to save the full console log in a file, but only display a relevant sub
pewma.py --workspace=/path/to/workspace --include=name.of.project1[,name.of.project2, ...] junit-test
    | tee ~/junit-test-console-log.txt | grep "\(Running\|Tests run\)"
```

OR

Use the command line and native `ant` (requires Ant version 1.8+):

```
if [[ "$(uname -n)" == *diamond.ac.uk ]]; then
    module load ant
fi
ant -version

# optionally use tee and grep to save the full console log in a file, but only display a relevant sub
ant -logger org.apache.tools.ant.NoBannerLogger -buildfile /path/to/workspace_git/project_name/releng
    | tee ~/junit-test-console-log.txt | grep "\(Running\|Tests run\)"
```

> **Warning:** This is provisional and **will** change.

CHAPTER

FIVE

WORKFLOW - BRANCHING, MERGING, RELEASING

This chapter describes the development workflow, and how we use Git to support it.

There are 2 different environments you can use: the command line, using native `git`; and the Eclipse IDE, using EGit. You should probably have at least some familiarity with both. To enable this, Git repositories for training/testing purposes are available (1 for the command line, and 3 for the IDE).

To experiment with the training repositories, you start off by creating your own private copy of them on a remote Git server. You can then make changes in your local clone, check them in, and push them to the remote server, without affecting anyone else.

5.1 Command line workflow

The purpose of this section is not to teach you the `git` command line (there are already many good `git` tutorials available on the web). Instead, this section illustrates the specific workflow that we use.

A training repository called, appropriately, `training/workflow.small`, is available for you to practice with, following the script below.

Confirm that you have access to the training repositories

```
$ ssh dascgitolite@dasc-git.diamond.ac.uk info       # issue command exactly as it appears, don't char
                                                     # if you are prompted for a password, there is a
  @C @R   W      training/${USER}/workflow\..*       # you can create and update the training reposit
     @R_         training/template/workflow\..*      # you can read the training repository templates
```

Check out the `training/workflow.small` repository

Start off by cloning a remote repository that you can practice with.

You only have read access to the remote repository, so having cloned it, change your clone to point to your own remote repository (which you *can* change).

```
$ rm -rf /scratch/training.small/                                                    # remove u
$ mkdir /scratch/training.small/
$ cd /scratch/training.small/
$ git clone dascgitolite@dasc-git.diamond.ac.uk:training/template/workflow.small.git      # clone the
Initialized empty Git repository in /scratch/training.small/workflow.small/.git/
$ cd workflow.small                                                                  # cd into
$ git remote set-url origin dascgitolite@dasc-git.diamond.ac.uk:training/${USER}/workflow.small.git
$ git push --mirror --verbose                                                        # (re-)crea
```

```
Pushing to dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
Initialized empty Git repository in /var/www/dascgitolite_repositories/training/bmn54829/workflow.sma
To dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
 * [new branch]      master -> master
 * [new branch]      origin/8.14 -> origin/8.14
 * [new branch]      origin/8.16 -> origin/8.16
 * [new branch]      origin/HEAD -> origin/HEAD
 * [new branch]      origin/master -> origin/master
$ gitk --all &                                                                    # take a lo
```

Make a simple change (no feature branch required)

Make a simple change to the repository (add a new file). Since the change is small, make it directly on the `master` branch.

```
  ### make sure we are on the master branch
$ git branch -a                                                                   # display a
* master
  remotes/origin/8.14
  remotes/origin/8.16
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$ git checkout master                                                             # switch to
Already on 'master'

  ### create a new file
$ echo "first line" > simple-file-add.txt

 ### commit the change to our local repository
$ git status                                                                      # display a
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   simple-file-add.txt
nothing added to commit but untracked files present (use "git add" to track)

$ git add --verbose -A                                                            # tell git t
add 'simple-file-add.txt'

$ git status                                                                      # the new fi
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   simple-file-add.txt
#

$ git commit -m "TRAINING: add new test file"                                     # this commi
[master d80b5ac] TRAINING: add new test file
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 simple-file-add.txt

$ git status                                                                      # no pending
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
```

```
nothing to commit (working directory clean)

  ### push current branch to the remote repository
$ git push --verbose                                                                    # update the
Pushing to dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
To dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
   8d4269f..d80b5ac  master -> master
```

Make a complex change (new feature branch used)

Make a more complex change to the repository. Since the change is large, create a new feature branch, and develop the feature there. Once coding (and testing!) is complete, merge the feature branch into the master branch. Branches for features under development tend to live in developer repositories only, not on `origin`.

New feature branches should have a meaningful name that starts with *feature-*.

```
  ### create a new branch and switch to it
$ git branch -a                                                                         # display a
* master
  remotes/origin/8.14
  remotes/origin/8.16
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$ git checkout master                                                                   # switch to
Already on 'master'
$ git pull                                                                              # do a fetch
Already up-to-date.

$ git checkout -b feature-test1 master                                                  # create a
Switched to a new branch 'feature-test1'

  ### create a new directory and file, and commit
$ mkdir newdir                                                                          # create a
$ echo "first line" > newdir/file1.txt                                                  # and a new
$ git status                                                                            # display a
# On branch feature-test1
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   newdir/
nothing added to commit but untracked files present (use "git add" to track)
$ git add --verbose newdir/*                                                            # tell git
add 'newdir/file1.txt'
$ git status                                                                            # changes n
# On branch feature-test1
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   newdir/file1.txt
#
$ git commit -m "TRAINING: add new feature dir and file"                                # this comm
[feature-test1 e5a0112] TRAINING: add new feature dir and file
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 newdir/file1.txt
$ git status                                                                            # no pendin
# On branch feature-test1
nothing to commit (working directory clean)
```

```
  ### now modify an existing file, create a new file, and commit
$ echo "some text" > newdir/file2.txt                                          # create a n
$ sed --in-place 's/line/text/' newdir/file1.txt                               # modify exi
$ git status                                                                   # display a
# On branch feature-test1
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   newdir/file1.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   newdir/file2.txt
no changes added to commit (use "git add" and/or "git commit -a")
$ git add --verbose newdir/*                                                   # tell git t
add 'newdir/file1.txt'
add 'newdir/file2.txt'
$ git status                                                                   # changes no
# On branch feature-test1
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   newdir/file1.txt
#   new file:   newdir/file2.txt
#
$ git commit -m "TRAINING: add new feature dir and file"                       # this commi
[feature-test1 767f33e] TRAINING: add new feature dir and file
 2 files changed, 2 insertions(+), 1 deletions(-)
 create mode 100644 newdir/file2.txt
$ git status                                                                   # no pending
# On branch feature-test1
nothing to commit (working directory clean)

  ### optionally push the new feature branch to the remote repository
# you might want to do this as a way of backing up the new feature from your local repository
# you might want to do this so another user can see your "in-progress" code (alternatively, they cou
$ git push --verbose -u origin feature-test1
Pushing to dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
To dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
 * [new branch]      feature-test1 -> feature-test1
Branch feature-test1 set up to track remote branch feature-test1 from origin.

  ### merge your new feature from the feature branch into the master branch
$ git checkout master                                                          # switch to t
Switched to branch 'master'
$ git pull --verbose                                                           # we should p
From dasc-git.diamond.ac.uk:training/bmn54829/workflow.small
 = [up to date]      feature-test1 -> origin/feature-test1
 = [up to date]      master     -> origin/master
Already up-to-date.
$ git merge --no-ff feature-test1                                              # merge our f
Merge made by recursive.
 newdir/file1.txt |    1 +
 newdir/file2.txt |    1 +
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 newdir/file1.txt
```

```
 create mode 100644 newdir/file2.txt
$ git push origin master                                                    # update the
To dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
   d80b5ac..e133d92  master -> master

  ### delete your feature branch
$ git branch -a
  feature-test1
* master
  remotes/origin/8.14
  remotes/origin/8.16
  remotes/origin/HEAD -> origin/master
  remotes/origin/feature-test1
  remotes/origin/master
$ git branch -d feature-test1                                               # delete the
Deleted branch feature-test1 (was 767f33e).
$ git branch -a
* master
  remotes/origin/8.14
  remotes/origin/8.16
  remotes/origin/HEAD -> origin/master
  remotes/origin/feature-test1
  remotes/origin/master
$ git push origin :feature-test1                                            # delete the
To dascgitolite@dasc-git.diamond.ac.uk:training/bmn54829/workflow.small.git
 - [deleted]         feature-test1
$ git branch -a
* master
  remotes/origin/8.14
  remotes/origin/8.16
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

5.2 IDE workflow

This section introduces you to using Git from within the Eclipse IDE.

There are 3 training repositories available, set up so that they can mimic our development workflow.

- Each repository contains multiple projects.
- Each repository has branches called `master`, `8.16`, `8.14`
- The contents themselves are minimal, just a few files.
- The repositories and their contents are:

```
    workflow.git.master
        workflow.master.feature

    workflow.a.git
        workflow.a.group.feature
        workflow.a.plugin1
        workflow.a.plugin2

    workflow.b.git
        workflow.b.group.feature
```

```
        workflow.b.plugin3
        workflow.b.plugin4
```

Confirm that you have access to the training repositories

```
$ ssh dascgitolite@dasc-git.diamond.ac.uk info      # issue command exactly as it appears, don't cha
                                                    # if you are prompted for a password, there is a
  @C @R   W      training/${USER}/workflow\..*      # you can create and update the training reposit
     @R_         training/template/workflow\..*     # you can read the training repository templates
```

5.2.1 Copy the training repositories

First make a copy, on the remote Git server, of the remote training repositories, using this command line script (download):

```
work_dir=/scratch/training_temp/
rm -rf ${work_dir}
mkdir ${work_dir}

for repository in "a" "b" "master"; do
  cd ${work_dir}
  git clone dascgitolite@dasc-git.diamond.ac.uk:training/template/workflow.${repository}.git
  cd workflow.${repository}
  git remote set-url origin dascgitolite@dasc-git.diamond.ac.uk:training/${USER}/workflow.${reposito
  for branch in "8.14" "8.16" "master"; do
    git checkout ${branch}
    git push origin ${branch}:refs/heads/${branch} --verbose  # (re-)create our private remote
  done
done

rm -rf ${work_dir}                                  # having created a remote copy of the remote repo
```

If you have previously copied the repositories, and wish to re-initialise them, simply run the above script again.

5.2.2 Check out the training repositories

Start your Eclipse IDE, pointing to a new workspace.

1. At *Project*, **de**-select `Build automatically`
2. Open the console view at *Window* → *Show View*
3. Open the **Git Repositories** and **Git Staging** views.
4. Open Buckminster CQUERY `https://alfred.diamond.ac.uk/buckminster/base/training-trunk.cquery`
5. On the Main tab, specify a component name of `workflow.master.feature` and a component type of `eclipse.feature`
6. On the Properties tab, change `training.repository.user` to your FedID
7. Resolve and Materialize, then enjoy!

Git repositories are cloned outside the workspace, in a directory called `${workspace_loc}_git`, and imported into your workspace as existing projects.

Other possible checkout options:

- For the `training/workflow.small` repository, specify a component name of `workflow.small` (and leave component type unspecified).
- For the IDE training repositories v8.16, specify CQuery `https://alfred.diamond.ac.uk/buckminster/base/tra`

Didn't work?

Did you:

Issue the `ssh dascgitolite@dasc-git.diamond.ac.uk info` command to check your access?
Create the remote copy of the training repositories?
Set the `training.repository.user` property?

Once you have fixed the problem, you need to:

Delete the `${workspace_loc}_git` directory
Restart Eclipse

5.2.3 Use EGit

more to come

5.3 Creating the training repositories

The training repositories are set up just one, and then copied by students.

5.3.1 Creating the small training repository

To create the `training/workflow.small` repository (for command line training), use the script (`./make_training_repo_small.sh`):

```
#!/bin/bash -ev

# create new repository
work_dir=/scratch/training.small.setup/
rm -rf ${work_dir}
mkdir ${work_dir}
cd  ${work_dir}

git init
git config user.name "Diamond Light Source"
git config user.email "gda@diamond.ac.uk"
git remote add origin dascgitolite@dasc-git.diamond.ac.uk:training/template/workflow.small.git

# populate the master branch and commit
cat <<END_OF_FILE > .project
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>workflow.small</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
```

```
    </buildSpec>
    <natures>
    </natures>
</projectDescription>
END_OF_FILE

cat <<END_OF_FILE > buckminster.cspec
<?xml version="1.0" encoding="UTF-8"?>
<cs:cspec xmlns:cs="http://www.eclipse.org/buckminster/CSpec-1.0" name="workflow.small" componentType
END_OF_FILE

echo "Let's call this version trunk" > version.txt              # create new file
git add -A
git commit -m "Initial \"trunk\" commit"                        # commit to local repository

# create a new 8.14 branch and commit
git checkout -b 8.14
sed --in-place 's/trunk/8.14/' version.txt                      # modify existing file
git add -A
git commit -m "Make 8.14 branch"                                # commit to local repository

# make a change on the 8.14 branch
git checkout 8.14
echo "A file added to 8.14" > added-to-8.14.txt                 # create new file
git add -A
git commit -m "Add new file in 8.14"                            # commit to local repository

# make a change on trunk
git checkout master
echo "Added post-8.14 file" > added-to-trunk-after-8.14.txt     # create new file
git add -A
git commit -m "Add new file after 8.14"                         # commit to local repository

# create a new 8.16 branch and commit
git checkout -b 8.16
sed --in-place 's/trunk/8.16/' version.txt                      # modify existing file
git add -A
git commit -m "Make 8.16 branch"                                # commit to local repository

# make a change on the 8.16 branch
git checkout 8.16
echo "A file added to 8.16" > added-to-8.16.txt                 # create new file
git add -A
git commit -m "Add new file in 8.16"                            # commit to local repository

# make a change on trunk
git checkout master
echo "Added post-8.16 file" > added-to-trunk-after-8.16.txt     # create new file
git add -A
git commit -m "Add new file after 8.16"                         # commit to local repository

# push all branches to remote repository, overwriting existing contents
git push --mirror --verbose
```

5.3.2 Creating the large training repositories

To create the `training/workflow.master`, `training/workflow.a`, `training/workflow.b` repositories (for IDE training), use the script (`./make_training_repo_large.sh`):

```
#!/bin/bash -ev

# create new repositories
work_dir=/scratch/training.large.setup/
rm -rf ${work_dir}
svn export /scratch/workspace/documentation.gda/training_template/ ${work_dir}

for repository in $(ls -1 ${work_dir}); do
    echo -e "\n\n********** processing ${repository} **********\n"

    cd ${work_dir}${repository}
    git init
    git config user.name "Diamond Light Source"
    git config user.email "gda@diamond.ac.uk"
    git remote add origin dascgitolite@dasc-git.diamond.ac.uk:training/template/${repository}

    # populate the master branch and commit
    for item in $(ls -1); do
        echo "Let's call this version trunk" > $item/version.txt     # create new file
    done
    git add -A
    git commit -m "Initial \"trunk\" commit"                        # commit to local repository
    git push origin master

    # create a new 8.14 branch and commit
    git checkout -b 8.14
    for item in $(ls -1); do
        sed --in-place 's/trunk/8.14/' $item/version.txt            # modify existing file
    done
    git add -A
    git commit -m "Make 8.14 branch"                                # commit to local repository

    # make a change on the 8.14 branch
    git checkout 8.14
    for item in $(ls -1); do
        echo "A file added to 8.14" > $item/added-to-8.14.txt       # create new file
    done
    git add -A
    git commit -m "Add new files in 8.14"                           # commit to local repository
    git branch
    git push origin 8.14
    git status

    # make a change on trunk
    git checkout master
    for item in $(ls -1); do
        echo "Added post-8.14 file" > $item/added-to-trunk-after-8.14.txt     # create new file
    done
    git add -A
    git commit -m "Add new files after 8.14"                        # commit to local repository

    # create a new 8.16 branch and commit
    git checkout -b 8.16
    for item in $(ls -1); do
```

```
        sed --in-place 's/trunk/8.16/' $item/version.txt            # modify existing file
    done
    git add -A
    git commit -m "Make 8.16 branch"                                # commit to local repository

    # make a change on the 8.16 branch
    git checkout 8.16
    for item in $(ls -1); do
        echo "A file added to 8.16" > $item/added-to-8.16.txt       # create new file
    done
    git add -A
    git commit -m "Add new files in 8.16"                           # commit to local repository
    git branch
    git push origin 8.16
    git status

    # make a change on trunk
    git checkout master
    for item in $(ls -1); do
        echo "Added post-8.16 file" > $item/added-to-trunk-after-8.16.txt     # create new file
    done
    git add -A
    git commit -m "Add new files after 8.16"                        # commit to local repository
    git branch
    git push origin master
    git status

done
```

5.4 Git branching model

5.4.1 Background theory

If you don't want to read all this theory, you can skip straight to the next section.

Our branching model is based on Vincent Driessen's article A successful Git branching model (published Jan 2010).

The original article did not adequately handle concurrent support for multiple releases, but the following extracts from comments to the article do:

- *Rory Fitzpatrick*:

```
    Not sure what software you're releasing, but I'm wondering how you would handle hotfixing a prev
    i.e. you've just released 2.0 but have to hotfix 1.0 for a customer, where does the hotfix-1.0.1
```

Vincent Driessen in reply to Rory Fitzpatrick:

```
    @Rory: That's an excellent follow-up question. To be honest, I haven't thought about that scenar

    But say you have the following situation:
    * You have tagged your releases 1.0, 1.1 and 2.0.
    * 1.x and 2.x are sold under different licenses.
    * Some critical bug must be fixed in both the 1.x and 2.x versions, and you do not want your cus

    My solution to that would be the following:
    * Create a "support branch" called support-1.x, based on tag 1.1 (since that is the latest 1.x c
      This branch should be long-lived and can be safely removed if you drop support on 1.x.
```

```
    * Create a branch hotfix-1.1.1, based on support-1.x
    * Create a branch hotfix-2.0.1, based on 2.0
    * Fix the bug on each (git cherry-pick may help here)

    Then, finish hotfix-1.1.1 by merging back into support-1.x (don't forget --no-ff), so not into m
    Technically you could just merge back those changes into develop, but the commit graph would qui
    Rather leave these situations to git cherry-pick (personal taste).

    You can finish hotfix-2.0.1 as you would regularly (merge back into master or develop).
    Merging the fix into develop causes the bug be fixed in all future releases.

    Effectively, what you're doing then, is having a master branch per supported product release.

    Maybe I'm forgetting something, haven't thought about it too long yet. Very interesting addition
```

- *Joe*:

```
    Acutally I think a more classical way of supporting the "in the wild" releases is to keep the re
    (instead of deleting after it is merged back to develop). You would have (for example) release 1
    with subsequent releases 1.2.1, 1.2.2, etc. delivered from this branch and with bug fixes merged
    You would delete the release branch when you stop supporting a release (but this is optional (yo
```

The following discussions of the branching model may be of interest:

- Are There Any Flaws With This Git Branching Model? (programmers.stackexchange.com, Apr 2011)
- What git branching models actually work - the final question has lots of useful discussion and links (stackoverflow, Apr 2010)

5.4.2 Supporting software

A collection of Git extensions (`git-flow`) to provide the high-level repository operations for the branching model has been developed, and is available at github. The scripts can handle so-called *support branches* as described in the previous section.

Although use of `git-flow` is not compulsory (you can always issue the native `git` commands), it simplifies the workflow and reduces the chance of error.

5.4.3 Other Git considerations

What is the git equivalent for revision number? (stackoverflow, Nov 2010)

CHAPTER

SIX

BUILDING THE PRODUCT

This chapter describes how to get compile a GDA or Dawn product

6.1 Building and Starting the GDA Server and Client

This section should be read in conjunction with the GDA QuickStart Guide.

Building GDA means compiling the Java code into .class files. The GDA distribution does not include these files, so you need to run a build before you can start using GDA.

6.1.1 Building the distribution

The GDA distribution includes, in addition to the base GDA, an example plugin, and an example configuration to run it. The GDA Developer Guide uses this to illustrate how to extend the GDA.

The command to build GDA is:

```
$ cd $GDA_WORKSPACE
$ python builder/gda-build.py --nowarn --product=example clean build
```

This results in the Java code in each plugin being compiled into the `bin` directory for the plugin (except for the `uk.ac.gda.core` plugin which is compiled into the `classes` directory). The GDA server runs using these .class files.

Additionally, a new directory called `client` is created, which holds the GDA client (an Eclipse RCP application).

6.1.2 Starting GDA

Once you have built GDA, start the GDA servers (there are several, a Name Server, an Event server, and an Object Server, etc), and then the GDA client. Although the servers and client would normally run on separate machines, the example configuration that comes with this release expects them on the same machine.

To start the servers:

```
$ cd $GDA_WORKSPACE
$ ./config/bin/gdaservers
```

You can add the `--verbose` if you wish. Ignore `Motor Position File` error messages, and wait for the `Server initialisation complete` message:

```
ERROR gda.device.motor.MotorBase - Motor Position File <path>/JT_RRMotor not found - setting JT_RRMot
   ...
INFO  gda.util.ObjectServer - Server initialisation complete. xmlFile = <$GDA_WORKSPACE>/config/xml/c
```

To start the client:

```
$ cd $GDA_WORKSPACE
$ ./config/bin/gdaclient
```

Various configuration related errors can be ignored. Once the GUI has launched and you have logged in, switch to the "Scripting" perspective. The GDA Developer Guide has a number of commands that you can try out.

6.1.3 Stopping GDA

To stop the client, `Exit` from the `File` menu usually works!

To stop the servers:

```
$ cd $GDA_WORKSPACE
$ ./config/bin/gdaservers --stop
```

On shutdown, errors such as `Retries exceeded, couldn't reconnect to 127.0.0.1` can be ignored.

CHAPTER

SEVEN

COMPONENTS USED IN THE PRODUCT

This chapter describes how various components that form part of an end product are maintained and published. For normal development work, you can simply use the published version of the component.

Note: These components become part of some products. For developer tools that do not form part of any product, see the section on *Tools*.

7.1 Target Platform Components

Below are listed the (non-Eclipse) components that can (if required) be materialized into your development environment's target platform (`tp/`). Only the components and versions that your source project(s) depend on will actually be materialized.

A plugin dependency (declared in either a plugin's `MANIFEST.MF` or in a feature's `feature.xml`) will materialize the dependent component. A package import (declared in a plugin's `MANIFEST.MF`) will *not* result in anything being materialized. If you use package imports, then the supplying plugin must be referenced somewhere (typically in a feature).

Whenever a plugin depends on a bundle or a package, the manifest should always declare a dependency on a specific version, or version range. Features should not normally specify any version dependency (i.e. they should depend on version `0.0.0`).

Usually, you specify a version range something like `[2.1.0,2.2.0)` or `[2.1.0,3.0.0)`, where the [indicates inclusive, and the) indicates exclusive. Otherwise a fresh materialize will get the latest available version, which can change at any time, and which your code might not even compile against.

Note: This list of available components is not yet complete. It is in the process of being updated.

If the bundle or version you want is not available, please contact `matthew.webber (at) diamond.ac.uk` to have it added. Although you could just add the bundle `.jar` to your own plugin, it's preferable that common bundles be provided once via the target platform, rather than everyone having their own copy.

Antlr (http://www.antlr.org/) (downloads) - A powerful parser generator

- `3.5.2_1` `org.apache.servicemix.bundles.antlr` - added 2014/06/03
- `3.5.2_1` `org.apache.servicemix.bundles.antlr-runtime` - added 2014/06/03

Apache Camel (http://camel.apache.org/) - A messaging technology glue with routing.

- `2.13.2` `org.apache.camel.camel-core` - added 2014/09/03
- `2.13.2` `org.apache.camel.camel-core-osgi` - added 2014/09/03

2.13.2 org.apache.camel.camel-guava-eventbus - added 2014/09/03
2.13.2 org.apache.camel.camel-jms - added 2014/09/03

Apache Commons Beanutils (http://commons.apache.org/proper/commons-beanutils/) - Easy-to-use wrappers around the Java

1.9.1 org.apache.commons.beanutils - added 2014/01/21
1.8.3 org.apache.commons.beanutils - added 2012/07/30
1.8.0 com.springsource.org.apache.commons.beanutils - also exported from uk.ac.gda.libs (removed 2014/01/22)

Apache Commons Codec (http://commons.apache.org/proper/commons-codec/) - General encoding/decoding algorithms (for ex

1.9.0 org.apache.commons.codec - added 2014/01/21
1.7.0 org.apache.commons.codec - added 2013/03/05
1.6.0 org.apache.commons.codec - added 2012/07/30
1.3.0 com.springsource.org.apache.commons.codec - not available to tp/, but exported from uk.ac.gda.libs (removed 2014/01/22)

Apache Commons Collections (http://commons.apache.org/proper/commons-collections/) - Extends or augments the Java Colle

note There are two different major versions available, which can be used together since the package names do not overlap
4.0.0 org.apache.commons.collections4 - added 2012/12/11 - package names start with org.apache.commons.collections4
3.2.1 org.apache.commons.collections - added 2012/07/30 - package names start with org.apache.commons.collections
3.2.1 com.springsource.org.apache.commons.collections - also exported from uk.ac.gda.libs

Apache Commons Configuration (http://commons.apache.org/proper/commons-configuration/) - Reading of configuration/pref

1.10.0 org.apache.commons.configuration - added 2013/12/02
1.8.0 org.apache.commons.configuration - added 2012/07/30
1.8.0 com.springsource.org.apache.commons.configuration - not available to tp/, but exported from uk.ac.gda.libs - removed 2013/12/02

Apache Commons CSV (http://commons.apache.org/proper/commons-csv/) - Reads and writes files in variations of the Comma

1.1.0 org.apache.commons.csv - added 2014/11/03

Apache Commons DBCP (http://commons.apache.org/proper/commons-dbcp/) - Database connection pooling services

1.4.0 org.apache.commons.dbcp - added 2012/07/30
1.2.2 org.apache.commons.dbcp - materialized from old dawb p2 site

Apache Commons Digester (http://commons.apache.org/proper/commons-digester/) - XML-to-Java-object mapping utility

note There are two different major versions available, which can be used together since the package names do not overlap
3.2.0 org.apache.commons.digester - added 2012/07/30 - package names start with org.apache.commons.digester3

`1.8.1` `com.springsource.org.apache.commons.digester` - also exported from `uk.ac.gda.libs` (removed 2014/02/12)

Apache Commons IO (http://commons.apache.org/proper/commons-io/) - Collection of I/O utilities

`2.4.0` `org.apache.commons.io` - added 2012/07/30

`1.4.0` `com.springsource.org.apache.commons.io` - also exported from `uk.ac.gda.libs` (removed 2014/01/21)

Apache Commons JEXL (http://commons.apache.org/proper/commons-jexl/) - Java Expression Language

`2.1.1` `org.apache.commons.jexl` - added 2013/03/05

Apache Commons JXPath (http://commons.apache.org/proper/commons-jxpath/) - Utilities for manipulating Java Beans using

`1.3` `org.apache.commons.jxpath` - added 2013/12/02

Apache Commons Lang (http://commons.apache.org/proper/commons-lang/) - Provides extra functionality for classes in java.la

note There are two different major versions available, which can be used together since the package names do not overlap

`3.2.1` `org.apache.commons.lang3` - added 2014/01/21

`3.1.0` `org.apache.commons.lang3` - package names start with `org.apache.commons.lang3`

`2.6.0` `org.apache.commons.lang` - package names start with `org.apache.commons.lang`

`2.5.0` `org.apache.commons.lang` -

`2.4.0` `org.apache.commons.lang` - also exported from `uk.ac.gda.libs` (removed 2014/01/23)

Apache Commons Math (http://commons.apache.org/proper/commons-math/) - Lightweight, self-contained mathematics and st

note Major versions 2 and 3 available, which can be used together since the package names do not overlap (`org.apache.commons.math` and `org.apache.commons.math3`)

`3.6.0` `org.apache.commons.math3` - added 2016/01/11

`3.2.0` `org.apache.commons.math3` - added 2013/04/23

`3.1.1` `org.apache.commons.math3` - added 2013/01/14

`3.1.0` `org.apache.commons.math3` - added 2013/01/07 - removed 2013/01/09 due to a major bug (use 3.1.1 instead)

`3.1.0-SNAPSHOT` `org.apache.commons.math3` - added 2012/10/09

`3.0.0` `org.apache.commons.math` - package names start with `org.apache.commons.math3`

`2.2.0` `org.apache.commons.math` - package names start with `org.apache.commons.math`

Apache Commons Net (http://commons.apache.org/proper/commons-net/) - Collection of network utilities and protocol implem

`3.3.0` `org.apache.commons.net` - added 2013/12/11

`3.2.0` `org.apache.commons.net` - added 2013/03/05

`3.1.0` `org.apache.commons.net` - added 2012/07/30

`1.4.1` `org.apache.commons.net` - materialized from old dawb p2 site

Apache Commons Pool (http://commons.apache.org/proper/commons-pool/) - Generic object pooling component

note There are two different major versions available, which can be used together since the package names do not overlap

`2.1.0` `org.apache.commons.pool2` - added 2014/01/21

`1.6.0` `org.apache.commons.pool` - added 2012/07/30

1.3.0 org.apache.commons.pool - materialized from old dawb p2 site

Apache Commons Validator (http://commons.apache.org/proper/commons-validator/) - Framework to define validators and val

1.4.0 org.apache.commons.validator - added 2012/07/30

Apache Commons VFS (http://commons.apache.org/proper/commons-vfs/) - Provides a single API for accessing various differe

1.0 com.springsource.org.apache.commons.vfs - added 2014/01/07

Apache HttpComponents HttpClient (http://hc.apache.org/httpcomponents-client-ga/index.html)

4.3.2 org.apache.httpcomponents.httpclient - added 2014/01/21
4.2.3 org.apache.httpcomponents.httpclient - added 2013/03/05
4.2.2 org.apache.httpcomponents.httpclient - added 2012/10/26
4.2.1 org.apache.httpcomponents.httpclient - added 2012/07/30

Apache HttpComponents HttpCore (http://hc.apache.org/httpcomponents-core-ga/index.html)

4.3.1 org.apache.httpcomponents.httpcore - added 2014/01/21
4.2.3 org.apache.httpcomponents.httpcore - added 2013/03/05
4.2.2 org.apache.httpcomponents.httpcore - added 2012/10/09
4.2.1 org.apache.httpcomponents.httpcore - added 2012/07/30

Apache Mina Core (http://mina.apache.org/mina-project/index.html) - A network application framework

2.0.9 org.apache.mina.core - added 2014/11/24
2.0.9 org.apache.mina.statemachine - added 2014/11/24
2.0.7 org.apache.mina.core - added 2014/02/03 (removed 2014/11/24)

Apache Mina SSHD (http://mina.apache.org/sshd-project/index.html) - Pure java library to support the SSH protocols on both

0.13.0 org.apache.sshd.core - added 2014/11/24
0.9.0 org.apache.sshd.core - added 2014/02/03 (removed 2014/11/24)

Apache XML-RPC (http://ws.apache.org/) - A Java implementation of XML-RPC

note this project is dead (has been archived by Apache), so consider using an alternative for new code.
3.1.3 uk.ac.diamond.org.apache.xmlrpc.client
3.1.3 uk.ac.diamond.org.apache.xmlrpc.common
3.1.3 uk.ac.diamond.org.apache.xmlrpc.server

Apache POI (http://poi.apache.org/) - Java API for Microsoft Documents

note The original jar has been repackaged as an OSGI bundle
3.1.0 uk.ac.diamond.org.apache.poi - your MANIFEST.MF should depend on version [3.1.0,3.2.0)

BoofCV (http://boofcv.org and https://github.com/lessthanoptimal/BoofCV) - fast computer vision library written entirely in Jav

note use the version exported by the org.dawnsci.boofcv rather than the tp/ version
0.16.0 uk.ac.diamond.boofcv - added 2014/06/18

Ddogleg (http://ddogleg.org/ and https://github.com/lessthanoptimal/ddogleg) - numerical optimization, polynomial root finding

0.6.0 org.ddogleg - added 2014/10/23
0.5.0 uk.ac.diamond.ddogleg - added 2014/07/02

`0.4.0` `uk.ac.diamond.ddogleg` - added 2014/06/18

Eclipse Graphiti (http://www.eclipse.org/graphiti/) - a Graphical Tooling Infrastructure

`0.10.0` `org.eclipse.graphiti(.*)` - added 2013/09/25

Eclipse NatTable (http://eclipse.org/nattable/) - a powerful and flexible SWT table/grid widget

`1.1.0` `org.eclipse.nebula.widgets.nattable.{core|extension.glazedlists|extension.poi}` - added 2014/07/07

`1.0.1` `org.eclipse.nebula.widgets.nattable.{core|extension.glazedlists|extension.poi}` - added 2013/07/29

`1.0.0` `org.eclipse.nebula.widgets.nattable.{core|extension.glazedlists|extension.poi}` - added 2013/05/31

`0.9.0` `org.eclipse.nebula.widgets.nattable.core` - added 2012/10/09

Eclipse Nebula Release (http://www.eclipse.org/nebula/) - Custom SWT widgets and reuseable UI-Components

note This is a pre-release version, if you use items from this you should probably depend on an exact version number in case of changes

`x.x.x.201401291008` `org.eclipse.nebula(.*)` - added 2014/02/05, taken from the project's "snapshot" build (published 2014/01/29 10:08)

`<various>` `org.eclipse.nebula(.*)` - added 2013/07/30, taken from the project's "snapshot" build (published 2013/07/09 21:47)

`<various>` `org.eclipse.nebula(.*)` - added 2013/05/31, taken from the project's "snapshot" build (published 2013/05/10 16:02)

Eclipse Nebula Incubation (http://www.eclipse.org/nebula/) - Custom SWT widgets and reuseable UI-Components

note This is a pre-release version, if you use items from this you should probably depend on an exact version number in case of changes

`x.x.x.201401291007` `org.eclipse.nebula(.*)` - added 2014/02/05, taken from the project's "snapshot" build (published 2014/01/29 10:07)

`<various>` `org.eclipse.nebula(.*)` - added 2013/07/30, taken from the project's "incubation/snapshot" build (published 2013/07/09 21:48)

EJML (http://code.google.com/p/efficient-java-matrix-library/ and https://github.com/lessthanoptimal/ejml) - A fast and easy to

`0.26.0` `com.googlecode.efficient-java-matrix-library.core` - published 2014/10/23

`0.26.0` `com.googlecode.efficient-java-matrix-library.equation` - published 2014/10/23

`0.26.0` `com.googlecode.efficient-java-matrix-library.experimental` - published 2014/10/23

`0.25.0` `uk.ac.diamond.ejml` - republished 2014/07/02 with changes

`0.25.0` `uk.ac.diamond.ejml` - added 2014/06/18

GeoRegression (http://georegression.org and https://github.com/lessthanoptimal/GeoRegression) - Geometry library for transfo

`0.7.0` `org.georegression` - added 2014/10/23

`0.7.0` `org.georegression.experimental` - added 2014/10/23

`0.6.0` `uk.ac.diamond.georegression` - added 2014/07/02

0.5.0 uk.ac.diamond.georegression - added 2014/06/18

Google Gson (http://code.google.com/p/google-gson/) - Google's Gson converts Java Objects into their JSON representation, and

2.2.4 com.google.gson - added 2013/12/11
2.2.2 com.google.gson - added 2013/02/11

Google Guava (http://code.google.com/p/guava-libraries/) - Google's core libraries for their Java-based projects

18.0.0 com.google.guava - added 2014/11/05
17.0.0 com.google.guava - added 2014/11/05
16.0.1 com.google.guava - added 2014/11/05
15.0.0 com.google.guava - added 2013/12/11
14.0.0 com.google.guava - added 2013/03/05
12.0.1 com.google.guava - added 2012/07/30
11.0.1 uk.ac.diamond.guava

Jackson JSON Processor (http://wiki.fasterxml.com/JacksonHome) - Jackson is a multi-purpose Java library for processing JSO

2.2.0 com.fasterxml.jackson.core.jackson-annotations - added 2013/05/10
2.2.0 com.fasterxml.jackson.core.jackson-core - added 2013/05/10
2.2.0 com.fasterxml.jackson.core.jackson-databind - added 2013/05/10

JAI Codec - Java Advanced Image API Codec

1.1.3 com.springsource.javax.media.jai.codec - added 2013/03/18

JAI Core - Java Advanced Image API Core

1.1.3 com.springsource.javax.media.jai.core - added 2013/03/18

Jaret Timebars (http://jaret.de/timebars/index.html) - SWT widget showing intervals in a Gantt chart

1.48.0 uk.ac.diamond.de.jaret.timebars - added 2013/09/03

Jaret Util (http://jaret.de/jaretutil/index.html) - helper classes for the Jaret project

0.32.0 uk.ac.diamond.de.jaret.util - added 2013/09/03

Javassist (http://www.javassist.org/) - makes Java bytecode manipulation simple

note This is used by Powermock
3.20.0.GA javassist - added 2015/07/01
3.18.0.GA javassist - added 2013/09/26

Javamail (https://java.net/projects/javamail/pages/Home) - a set of abstract APIs that model a mail system. Several service prov

1.5.2 com.sun.mail.javax.mail - added 2014/06/02
1.4.1 com.springsource.javax.mail - also exported from uk.ac.gda.libs (removed 2014/06/02)

JLargeArrays (https://github.com/IcmVis/JLargeArrays) - pure-java library of one-dimensional arrays

note This is used by JTransforms
1.2.0 uk.ac.diamond.jlargearrays - added 2015/02/06

JScience (http://jscience.org/) - Java library for the scientific community

note There are two different major versions available, with overlapping package names

4.3.1 uk.ac.diamond.org.jscience4 - added 2013/07/29

4.3.1 uk.ac.diamond.org.jscience - deprecated, since it's been republished with a new bundle name

2.0.2 uk.ac.diamond.org.jscience

JTransforms (https://github.com/wendykierp/JTransforms) - multithreaded FFT library written in pure Java

3.0.0 uk.ac.diamond.jtransforms - added 2015/02/06

Logback (http://logback.qos.ch/) - a logging library for Java

see also SLF4J

1.1.6 ch.qos.logback.classic - added 2016/03/11

1.1.6 ch.qos.logback.core - added 2016/03/11

1.1.1 ch.qos.logback.classic - added 2014/02/07

1.1.1 ch.qos.logback.core - added 2014/02/31

1.1.0 ch.qos.logback.classic - added 2014/01/07

1.1.0 ch.qos.logback.core - added 2014/01/31

Logback Gelf (https://github.com/Moocar/logback-gelf) - a Logback appender that encodes logs to GELF and transports them t

0.3.0 me.moocar.logback-gelf - added 2016/04/26

Mockito (http://code.google.com/p/mockito/) - Simpler & better mocking

1.9.5 org.mockito.mockito-core - added 2012/11/30

Objenesis (http://objenesis.org/) - a small Java library to instantiate a new object of a particular class

note This is used by Powermock

2.1.0 org.objenesis - added 2015/07/01

1.0.0 com.springsource.org.objenesis - added 2013/09/26

Opal (http://code.google.com/a/eclipselabs.org/p/opal/) - New widgets for the SWT API

note The original jar has been repackaged as an OSGI bundle

0.9.5.2 uk.ac.diamond.org.mihalis.opal - added 2013/07/05

Powermock (https://code.google.com/p/powermock/) - a Java framework that allows you to unit test code normally regarded as

note The original jar has been repackaged as an OSGI bundle

1.5.1 uk.ac.diamond.org.powermock - your MANIFEST.MF should depend on version [1.5.1,1.5.2) - added 2013/09/26

Proxy Vole (http://code.google.com/p/proxy-vole/) - A Java library to auto detect the platform network proxy settings

note The original jar has been repackaged as an OSGI bundle (note the change in version number format)

1.0.20120727 uk.ac.diamond.proxyvole - added 2012/08/07

Sherpabeans Commons (http://code.google.com/a/eclipselabs.org/p/passerelle/)

6.2.5 com.isencia.sherpa.commons.reduced - added 2013/01/14 (from svn r1027)

SLF4J (http://www.slf4j.org/) - Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various log

see also Logback

1.7.18 jcl.over.slf4j - added 2016/03/11

1.7.18 log4j.over.slf4j - added 2016/03/11
1.7.18 slf4j.api - added 2016/03/11
1.7.18 slf4j.ext - added 2016/03/11
1.7.6 jcl.over.slf4j - added 2014/02/06
1.7.6 log4j.over.slf4j - added 2014/02/06
1.7.6 slf4j.api - added 2014/02/06
1.7.5 jcl.over.slf4j - added 2014/01/31
1.7.5 log4j.over.slf4j - added 2014/01/31
1.7.5 slf4j.api - added 2014/01/31

Uncommons Maths (http://maths.uncommons.org/) - Random number generators, probability distributions, combinatorics and

1.2.3 org.uncommons.maths - added 2013/01/08
1.2.2 uk.ac.diamond.org.uncommons.maths - added 2012/08/07

Vecmath (http://java3d.java.net/) - 3D Vector Math Package

1.5.2 javax.vecmath - added 2012/11/06 (vecmath.jar comes from the zip binaries from http://java3d.java.net/binary-builds.html)

xraylib (https://github.com/tschoonj/xraylib) - a library for X-ray matter interactions cross sections for X-ray fluorescence appli

3.2.0 com.github.tschoonj.xraylib.jar - added 2016/03/02
3.1.0 com.github.tschoonj.xraylib.jar - added 2015/12/10

XStream (http://xstream.codehaus.org/) - XStream is a simple library to serialize objects to XML and back again

1.4.7 xstream - added 2014/06/11

7.2 Pydev

GDA and Scisoft use a modified version on PyDev.

This version of Pydev needs to be kept up to date with the main releases of Pydev, and an example of that follows here. This example assumes that you have a Github account, as well as push access to the opengda repository.

7.2.1 Update Process

Load up git

```
module load git
```

Generate a ssh key for Github

```
ssh-keygen -t rsa -C "mark.basham@diamond.ac.uk"
```

Look at the public key and add it to Github

```
gedit id_rsa.pub
```

Ssh to Github to check that this is working

```
ssh git@github.com
```

Configure git to work with Github, the token is obtained from the Github web-site for your login

```
git config --global user.name "Mark Basham"
git config --global user.email "mark.basham@diamond.ac.uk"
git config --global github.user markbasham
git config --global github.token 4906####################d87e
```

Go to where you want the git clone

```
cd ~/pydev/
```

Then clone it in to the directory using the URL from the Github web-site

```
git clone git://github.com/openGDA/Pydev.git
```

Go into that directory

```
cd Pydev/
```

You can use the gitk GUI to look at the current state

```
gitk &
```

Now we have the diamond Pydev checked out we want to add the Aptana one as a remote, so we can get the changes

```
git remote add pydev git://github.com/aptana/Pydev.git
```

Then fetch all the changes to our repository so we can make the merge

```
git fetch pydev -t
```

Now this is done go the the opengdatrunk, and then make a branch of this into which we shall make the merge

```
git checkout opengdatrunk
git checkout -b upgradeto2_1
```

Now check which tag we want to look at

```
git tag
```

Then choose the tag we want to merge to

```
git merge pydev_2_1_0
```

The merge is now complete, check the status to see how things went, or there is a nice GUI for looking at all the merges and conflicts

```
git status
git gui
```

However at this point it can often be useful to return to Eclipse to make the merge

Once happy that everything is done, add all the changes and commit to your local branch

```
git add -A
git status
git commit
git status
```

You can then check the diff between the new branch and the tag if you wish to check that sensible changes have been made

```
git diff pydev_2_1_0
```

Now the merge is complete on the branch, you need to merge back to the trunk and update your changes toGithub.

First set up the remote site for writing, make sure you have write access to the openGDA

```
git remote set-url origin git@github.com:openGDA/Pydev.git
```

Go back to the trunk

```
git checkouk opengdatrunk
```

Then merege back in the branch we just made, this should take no time, as it should just be a 'fast forward' and so no work to do

```
git merge upgradeto2_1
```

Have a quick look to make sure things are all as they should be

```
git status
gitk
```

Then push the changes back into the man repository on Github

```
git push origin opengdatrunk
```

Finally there is a master branch, which should b e brought up to date

```
git checkout -b bringmasterup2date origin/master
git merge opengdatrunk
git push origin bringmasterup2date:master
```

And thats it, all done.

CHAPTER

EIGHT

DEVELOPER TOOLS

This chapter describes how various tools that are useful during development are maintained and published.
For normal development work, you can simply use the published version of the tool.

Note: These tools do *not* become part of any product. For components that do form part of some products, see the section on *Components*.

8.1 Developer Tools Plugin

This is a plugin for the Eclipse IDE you use for software development.

8.1.1 Installing the tool

See this page on the wiki https://trac.diamond.ac.uk/gda/wiki/DeveloperToolsPlugin

8.1.2 Building the tool

CHAPTER

NINE

WRITING AND PUBLISHING DOCUMENTATION

This section describes how to write and publish documentation for GDA and Scisoft.

If you just want to view the manuals, they are available at the alfred.diamond.ac.uk/documentation/ website.

See also:

Sphinx documentation generator

reStructuredText markup language used by Sphinx

CheatSheet Restructured Text (reST) and Sphinx CheatSheet

sphinx-dev Google Group

9.1 Checking out the documentation source

9.2 Structure of a manual

9.3 Building and Publishing

Make your changes by editing the markup (in `.rst` files) using any suitable editor (eg Eclipse text editor). Unfortunately, there is no suitable WSYWIG editor available.

To build a manual using the Eclipse IDE:

Use the External Tools icon  on the Launch toolbar, or go to *Run* → *External Tools* and select the manual

(the output from the build process, including any error messages, appears in the *Console* view)

To build a manual from the command line:

```
cd documentation.gda/Infrastructure_Guide/            # specify the appropriate directory for yo
make clean all CONTEXT=diamond                        # or, e.g., "make html", "make clean pdf",
```

The built manual can be found in:

- The built manual `HTML` is written to *project_directory*/*manual_name*/build/html/contents.html and can be viewed within Eclipse, or in an external web browser
- The built manual `.pdf` is written to *project_directory*/*manual_name*/build/pdf-*papersize*/*manual_name*.

To publish your changes, simply check them back in to the repository

- Please make sure that your changes build without errors (for both `HTML` and `.pdf`) *before* checking them in to the repository
- Do *not* check in the `build` directory, only your changes in the `source` directory
- Once checked in, new versions of the manuals will be automatically built and published to the web sites (this normally takes around 10-15 minutes)
- You will receive an email once the build/publish job completes or fails, provided the build server knows the email address associated with your FedID

9.3.1 Notes on the automated build

The build and publish process is performed by the various Jenkins jobs named `*.documentation.manuals`. To see how this works, look at:

- The *Configure* page for the specific job (this points to the script which does the actual build and publish)

CHAPTER

TEN

SUPPORTING INFRASTRUCTURE

This chapter describes how additional infrastructure is set up and used. For normal development work, you can ignore this.

10.1 Gitolite

Gitolite is used to control access to the Diamond Light Source internal Git repository.

10.1.1 Installing Gitolite

It's important to refer to the comprehensive gitolite documentation when installing, follow the instructions, and not take shortcuts. Doing so will bring you grief. Here's what I did:

Define environment variables:

```
gitolite_server=dasc-git.diamond.ac.uk                                  # the server that hosts gito
gitolite_server_dev=dasc-git-dev.diamond.ac.uk                          # the test server that hosts
gitolite_hosting_user=dascgitolite                                      # the userid that gitolite ru
gitolite_admin=bmn54829                                                 # a userid that administers
```

As ${gitolite_admin}:

Generate a public/private keypair (you can use an existing key pair, or generate a new one):

```
ssh-keygen -t rsa                                                       # generate a keypair
cp -vi ~/.ssh/id_rsa.pub /dls_sw/dasc/pub/${gitolite_admin}.pub         # copy public key somewhere ac
                  OR
ssh-keygen -t rsa -f ~/.ssh/id_rsa-gitolite.pub                         # generate a keypair with a no
cp -vi ~/id_rsa-gitolite.pub /dls_sw/dasc/pub/${gitolite_admin}.pub     # copy public key somewhere ac
```

As ${gitolite_hosting_user}:

The install needs to be done on the hosting server:

```
ssh ${gitolite_server} OR ${gitolite_server_dev}
sudo su - ${gitolite_hosting_user}                                      # switch user via whatever me
```

Ensure the following lines are in `/localhome/dascgitolite/.bashrc` (put there by `cfengine`):

```
PATH=${PATH}:~/bin
test -f /etc/bashrc && . /etc/bashrc && module load git
```

Install gitolite:

```
git --version                                                    # check that git is loaded
echo $PATH                                                       # check that /localhome/dascg

cd ~
git clone git://github.com/sitaramc/gitolite.git gitolite_source
cd gitolite_source/
git tag -l
git checkout v2.1                                                # or whatever tag you want
src/gl-system-install                                            # installs into ~/bin ~/share/
```

Configure gitolite. This writes into `~/.gitolite.rc` and `~/.gitolite/` (locations *cannot* be changed):

```
gl-setup /dls_sw/dasc/pub/${gitolite_admin}.pub                  # configures gitolite with ${g
```

At this point, you are using the `vi` editor to edit the configuration file. Referring to the vi reference as required, set:

```
$REPO_BASE="/var/www/dascgitolite_repositories";                 # this is backed up
$GL_WILDREPOS = 1;                                               # allow wildcard in repository
```

Save your changes and exit (**`:wq`**). Gitolite will then be set up.

Set up access:

```
gl-tool add-shell-user /dls_sw/dasc/pub/${gitolite_admin}.pub              # allows ${gitolite_admin
```

10.1.2 Administering Gitolite

To administer Gitolite, the normal basic idea is to check out a copy of the `gitolite-admin` repository from the gitolite server, modify the configuration information, commit locally, then push back. See administering and running gitolite for details.

> **Warning:** Unless you know what you're doing, do not do anything manually on the server (except when the documentation says you should, for example to add custom hooks). In particular, adding new repositories or users or changing the access control rules should not be done directly on the server.

If you enter the following in your `~/.ssh/config`, you can reduce the amount of typing you need to do:

```
Host gitolite-dev
    User dascgitolite
    HostName dasc-git-dev.diamond.ac.uk

Host gitolite
    User dascgitolite
    HostName dasc-git.diamond.ac.uk
```

To clone the `gitolite-admin` repository:

```
# if you haven't previously cloned gitolite-admin, do so now:
git clone ${gitolite_hosting_user}@${gitolite_server}:gitolite-admin ~/gitolite-admin
git clone ${gitolite_hosting_user}@${gitolite_server_dev}:gitolite-admin ~/gitolite-admin-dev

# if you have previously cloned gitolite-admin, make sure it's up to date:
git pull
```

Add a new user

To add new user `someoneid`, they need to provide their public key:

```
# clone gitolite-admin as described above
# add key to gitolite-admin/keydir/
cp -iv /tmp/someoneid.pub ~/gitolite-admin/keydir/someoneid@diamond.pub
# push change back to server
cd ~/gitolite-admin/
git add -A
git commit -m "add new user someoneid firstname lastname"
git push
```

Add a new repository

If you have an existing repository that you want to add to gitolite:

```
# clone gitolite-admin as described above
# add a rule for the new repository to gitolite-admin/conf/<group>.conf
cd ~/gitolite-admin/
gedit conf/<group>.conf
# push change back to server
git add -A
git commit -m "add new repository"
git push
# this creates an empty, clonable repo on the gitolite server

# now push the new contents to the gitolite server
cd your-copy-of-the-new-repository
# make sure all the branches are correct and no extra stuff, "temp" branches, etc., are present
git remote add origin dascgitolite@dasc-git.diamond.ac.uk:${reponame}.git
git push origin master
```

Other tasks

At some point you will need to refer to *Creating the training repositories*.

If the Git server dies

(to be written)

See also:

Gitolite Gitolite project home page on Github

The access control file gitolite.conf Look here for information on the syntax of configuration file entries

CHAPTER

ELEVEN

GLOSSARY

workspace Every Eclipse application requires a workspace, which manages projects. The workspace contains (in a sub-directory called `.metadata`) information about the projects. The workspace can also contain the projects themselves.

B

S

W