



User Documentation for FishEye 4.0

Contents

Getting started	4
Supported platforms	5
Native support for SVN	9
Installing FishEye on Windows	14
Installing FishEye on Linux and Mac	17
Starting to use FishEye	20
Configuring JIRA integration in the Setup Wizard	25
Using FishEye	31
Using the FishEye screens	32
Browsing through a repository	33
Viewing file content	35
Viewing a file history	36
Using side by side diff view	37
Viewing the changelog	38
Searching FishEye	40
FishEye charts	44
Using favourites in FishEye	47
Changeset discussions	48
Viewing the commit graph for a repository	48
Viewing people's statistics	53
Using Smart Commits	55
Changing your user profile	60
Re-setting your password	63
Pattern matching guide	63
Date expressions reference guide	64
EyeQL reference guide	65
Integrating FishEye with Atlassian applications	71
JIRA Integration in FishEye	72
Integrating FishEye with Bitbucket Server	76
Transitioning issues in JIRA	76
FishEye FAQ	78
General FAQ	79
About database encoding	80
About the Lines of Code Metric	81
Cannot View Lines of Code Information in FishEye	83
Finding your Server ID	83
How Do I Archive a Branch within Perforce	83
How do I Avoid Long Reindex Times When I Upgrade	84
Mercurial Known Issues	86
Ordering of Branches Important When Visualizing Git Changesets	86
Permanent authentication for Git repositories over HTTP(S)	87
Perforce Changesets and Branches	88
What SCM systems are supported by FishEye?	88
Automating Administrative Actions in Fisheye	88
How are indexing requests handled when they are triggered via commit hook	89
Installation & Configuration FAQ	89
Can I deploy FishEye or Crucible as a WAR?	90
Does Fisheye support SSL (HTTPS)?	90
Improve FishEye scan performance	90
Migrating FishEye Between Servers	91
Setting up a CVS mirror with rsync	92
What are the FishEye System Requirements?	93
How to reset the Administration Page password in FishEye or Crucible	94
How Do I Configure an Outbound Proxy Server for FishEye	94
How to remove Crucible from FishEye 2.x or later	94
How to run Fisheye or Crucible on startup on Mac OS X	95

Licensing FAQ	96
Are anonymous users counted towards FishEye's license limits?	96
Restrictions on FishEye Starter Licenses	97
Updating your FishEye license	99
Git or Hg Repository exceeds number of allowed committers	100
Example EyeQL Queries	102
How do I find changes made to a branch after a given tag?	102
How do I filter results?	102
How do I find changes between two versions, showing separate histories?	103
How do I find changes made between two version numbers?	103
How do I find commits without comments?	103
How do I find files on a branch, excluding deleted files?	103
How do I find files removed from a given branch?	103
How do I find revisions made by one author between versions?	103
How do I select the most recent revisions in a given branch?	104
How do I show all changesets which do not have reviews?	104
Integration FAQ	104
How do I disable the Source (FishEye) tab panel for non-code projects?	104
How do I enable debug logging for the JIRA FishEye plugin?	104
How do I uninstall the JIRA FishEye plugin?	105
How is the Reviews (Crucible) tab panel for the JIRA FishEye Plugin populated?	105
What do I do if I discover a bug with the JIRA FishEye plugin?	105
Subversion FAQ	105
Configuring Start Revision based on date	106
Errors 'SEVERE assert' or 'Checksum mismatch'	106
FishEye fails to connect to the Subversion repository after a short time of successful operation.	106
How can FishEye help with merging of branches in Subversion?	106
Subversion Changeset Parents and Branches	107
SVN Authentication Issues	107
What are Subversion root and tag branches?	108
Why do I need to describe the branch and tag structure for Subversion repositories? ...	108
Why don't all my tags show up in FishEye?	109
About merges in Subversion	111
CVS FAQ	114
How does FishEye calculate CVS changesets?	114
How is changeset ancestry implemented for CVS?	114
Support Policies	114
Bug Fixing Policy	114
New Features Policy	115
Security Bugfix Policy	116
Troubleshooting	116
After I commit a change to my CVS repository, it takes a long time before it appears in FishEye.	117
FishEye freezes unexpectedly	117
I have installed FishEye, and the initial scan is taking a long time. Is this normal?	118
I have installed FishEye, but there is no data in the Changelog.	118
Initial scan and page loads are slow on Subversion	119
JIRA Integration Issues	119
Manually Generating a Thread Dump	119
Message 'org.tigris.subversion.javahl.ClientException svn Java heap space'	122
Problems with very long comments and MySQL migration	122
URLs with encoded slashes don't work, especially in Author constraints	122
FishEye Developer FAQ	123
Contributing to the FishEye Documentation	123
FishEye Documentation in Other Languages	124
FishEye Resources	125
Glossary	125
Collecting analytics for FishEye	127

Getting started

Atlassian's FishEye is the on-premise source code repository browser for enterprise teams. It provides your developers with advanced browsing and search for SVN, Git, Mercurial, Perforce and CVS code repositories, from any web browser.

Browse repos from your browser!

- SVN, Hg, Git, CVS, P4
- files, changesets, revisions, branches, tags, diffs, annotations
- side-by-side diffs
- unified diffs for word-level changes
- see details at the repository, branch, directory, or file levels
- filter commits by log message, path, author, date, branch

Search everything

- File names, commit messages, authors, text, JIRA issue keys, partial paths, wild cards, camel case
- QuickNav
- Simple search
- Advanced search - SQL-like syntax (EyeQL)
- Bookmarks any query
- Download search results
- Link to any revision, changeset, diff view, line of code, search

Monitor and visualize

- Understand what, when, where, who, how changes were introduced
- Commit graph for changesets
- Activity streams
- Charts and reports
- Notifications for code activity byemail, RSS, OpenSocial dashboards

Integrate with JIRA and other Atlassian applications

- See related JIRA issues
- Use smart commits to transition issues
- Crucible reviews
- Bamboo builds

Enterprise ready

- User management and authentication
- Database support and migration
- Security
- Performance
- Backup and restore

Ecosystem

- Marketplace for add-ons
- REST API to write your own custom add-ons e.g. to integrate with other applications







Get started!

1. Install and start FishEye on either [Windows](#), or [Linux and Mac](#).
2. Work through [Starting to use FishEye](#).
3. Tell FishEye about your [repositories](#).
4. Set up [users and groups](#).
























Supported platforms

This page lists the supported platforms for **FishEye 4.0.x** and its minor releases.

Key:  = Supported  = Deprecated  = Not Supported

<i>Java</i>		
Oracle JRE / JDK	 1.8  1.7	<p>FishEye requires the Java Runtime (JDK or JRE), version as noted. Pre-release/Early access versions of Java are <i>not supported</i>.</p> <p>We highly recommend that you use the Oracle JVM (or OpenJDK for Linux only). Other Java implementations have not been tested.</p> <p>You can download an Oracle Java Runtime.</p> <p>For the OpenJDK, download and install instructions for Linux flavors are at http://openjdk.java.net/install/.</p> <p>Please note:</p> <ul style="list-style-type: none"> • Once you have installed Java, you must set the JAVA_HOME environment variable. See Installing FishEye on Windows or Installing FishEye on Linux and Mac. • If you are using a 64-bit JVM, please ensure that you've set your max heap size (<code>--Xmx</code>) to a reasonable value, considering the RAM requirements of your system. • If you intend to run FishEye as a Windows Service, using the Java Service Wrapper, you should use the Java JDK rather than the JRE so as to take advantage of the <code>-server</code> parameter. If you're using the Windows Installer distribution, which handles the FishEye service differently, using the Java JRE should suffice. • You'll need the JDK for the JSP source download. • For OpenJDK, you'll need the DejaVu font package installed. Installation instructions can be found here: http://dejavu-fonts.org/wiki/Download. <p> Support for Java 7 was removed in FishEye 3.9, as previously announced.</p>
OpenJDK	 1.8 (Linux only)	
<i>Operating systems</i>		
Microsoft Windows		<ul style="list-style-type: none"> • FishEye is a pure Java application and should run on any platform provided the requirements for the JRE or JDK are
Linux		

Apple Mac OS X		<div>satisfied.</div> <div>Although FishEye can be run in virtualized environments, Atlassian is not yet able to provide technical support for performance-related problems in a virtualized environment. If you do choose to run FishEye in a VM, please ensure that you choose a VM with good IO throughput.</div>
<div>Databases</div>		
HSQLDB	Bundled; for evaluation use only	<div>The FishEye built-in database, running HSQLDB, is somewhat susceptible to data loss during system crashes. We recommend that you do not use HSQLDB for production systems.</div> <div>External databases (such as MySQL) are generally more resistant to data loss during a system crash.</div> <div>See the FishEye Database documentation for further details.</div>
MySQL	MySQL Enterprise Server 5.1+ MySQL Community Server 5.1+ MySQL 5.0 MariaDB, Percona	<div>For MySQL:</div> <ul style="list-style-type: none"> For 5.1, versions earlier than 5.1.10 are not supported For 5.6, versions earlier than 5.6.11 are not supported For 5.7, versions earlier than 5.7.5 are not supported Support for MySQL 5.0 was removed in FishEye 3.3. See End of Support Announcements for FishEye. MariaDB and Percona variants of MySQL are not supported, and are known to cause issues when used with FishEye. <div> Support for PostgreSQL 8.2 was removed in FishEye 3.3. See End of Support Announcements for FishEye.</div> <div> Support for SQL Server 2005 was removed in FishEye 3.3. See End of Support Announcements for FishEye.</div>
PostgreSQL	9.0, 9.1, 9.2, 9.3, 9.4 8.3, 8.4 8.2	
Oracle	12c 11g	
SQL Server	2012 2008, 2008 R2 2005	
<div>Web browsers</div>		
Microsoft Internet Explorer	10.0, 11.0 9.0	Support for Internet Explorer 9 was removed in FishEye 3.9. See End of Support Announcements for FishEye .
Mozilla Firefox	Latest stable version supported 3.6, 4.0	Support for Firefox 3.6 and 4.0 was removed in FishEye 3.7.
Safari	Latest stable version supported 4, 5	Support for Safari 4 and 5 was removed in FishEye 3.7.

Chrome	 Latest stable version supported	
<i>Version control systems</i>		
Subversion	Server:  1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8 Client:  SVNKit (bundled & the default)  Native JavaHL 1.8  Native JavaHL 1.7  Native JavaHL 1.6	 FishEye 3.1, and later, do not support the native JavaHL 1.6 client. See Native support for SVN for discussion.
CVS (and CVSNT)	 All versions	
Perforce	 Client version 2007.3 or later	The Server must support the ztag tagged protocol.  Perforce Streams, introduced in 2011.1, is not currently supported. See <div> FE-3886 - support for Streams in p4 OPEN </div>
Git	 2.8.2  2.7.3  2.6.5  2.5.4  2.4.10  2.3.10  2.2.3  2.1.4  2.0.5  1.9.5  1.8.0.3, 1.8.1.5, 1.8.2.3, 1.8.3.4, 1.8.4.5, 1.8.5.6  1.7.1.1, 1.7.2.5, 1.7.3.5, 1.7.4.5, 1.7.5.4, 1.7.6.6, 1.7.7.7, 1.7.8.6, 1.7.9.7, 1.7.10.5, 1.7.11.7, 1.7.12.4	These are the versions of Git that we currently test FishEye against.  Git 1.8.4.3 is not supported. See <div> BSERV-4101 - Clone and fetch fail with "protocol error: impossibly long line" CLOSED </div> Externally-hosted repositories only are supported. <div> [Security vulnerability CVE-2014-9390] affects multiple Git versions. FishEye itself is not affected, however you should update your <i>clients</i> to a patched maintenance version: v1.8.5.6, v1.9.5, v2.0.5, v2.1.4 and v2.2.1 or newer. </div> <div> Git for Windows is the <i>only supported distribution</i> when running FishEye on Windows. Cygwin Git is <i>not supported</i> and has known issues. </div>

Mercurial	<p>✔ 3.0.2, 3.1.2, 3.2.4, 3.3.3, 3.4.2, 3.5.2, 3.6.3, 3.7.2</p> <p>✔ 2.0.2, 2.1.2, 2.2.3, 2.3.2, 2.4, 2.5.2, 2.6.3, 2.7.2, 2.8.2, 2.9.1</p> <p>✔ 1.5.1, 1.5.4, 1.6.4, 1.7.5, 1.8.4, 1.9.3</p>	<p>These are the versions of Mercurial that we currently test FishEye against.</p> <p>As of version 3.6.3, FishEye supports Mercurial 3.</p> <p>✖ Mercurial 2.1 has a bug that makes it incompatible with FishEye. Please use Mercurial 2.1.1 or later.</p> <p>You should restart FishEye after upgrading Mercurial.</p>
Atlassian applications		
Crowd	<p>✔ Crowd 2.4.x+</p> <p>✔ Crowd client library: 2.4.1</p>	<p>From version 2.8.x, FishEye bundles the Crowd 2.4.1 client library, and supports the Crowd 2.4.x server, and later versions.</p>
JIRA	<p>✔ JIRA 5.0+</p>	<p>FishEye to JIRA communication requires JIRA 5.0.x or later. Communication the other way, from JIRA to FishEye, depends on the JIRA FishEye Plugin.</p> <p>Note that the JIRA FishEye Plugin is bundled with JIRA. If you are using a version of JIRA earlier than JIRA 5.0 you can upgrade the plugin in JIRA to get support for FishEye.</p>

Hardware requirements

FishEye should ideally run on a dedicated server. The most important aspect for a large-repository deployment will be I/O speed. You definitely want a fast local HDD for FishEye's cache. Note that NFS and SAN are not supported.


Component	Specifications
CPU	1.8GHz or higher, a single core is sufficient. More cores or higher GHz will result in better load-handling ability.
RAM	1GB minimum, 2GB will provide performance "headroom". Your Java heap should be sized at 512MB with the <code>FISHEYE_OPTS</code> environment variable, adjustable up to 1024MB depending on performance.
I/O	FishEye's input/output is an important element of its overall performance. If FishEye accesses your repository remotely, make sure that the throughput is maximum and the latency minimum (ideally the servers are located in the same LAN, running at 100Mbps or faster).
Monitor	Minimum screen resolution of 1024x768. Recommended screen resolution of 1280x768 or above.

While some of our customers run FishEye on SPARC-based hardware, Atlassian only officially supports FishEye running on x86 hardware and 64-bit derivatives of x86 hardware.

Disk space requirement estimates

Disk space requirements for FishEye may vary due to a number of variables such as the repository implementation, file sizes, content types, the size of diffs and comments being stored. The following table contains some real-world examples of FishEye disk space consumption.

Repository technology	Commits	Codebase size (HEAD of trunk)	FishEye index size
Subversion	14386	466 MB in 12151 files	647 MB
CVS	8210	115 MB in 11433 files	220 MB

 These disk space estimates are to be used as a guideline only. We recommend you monitor the disk space that your FishEye instance uses over time, as needs for your specific environment may vary. It may be necessary to allocate more space than indicated here. Additionally, you can reduce disk space consumption by [turning off diff storage](#) in FishEye.

Deployment notes for version control systems

Subversion (server)	FishEye can communicate with any repository running Subversion 1.1 or later.
Subversion (client)	FishEye now bundles the SVNKit client, which becomes the default Subversion interface. An alternative is to use the native subversion client, using JavaHL bindings. Please see Subversion Client Setup for more information.
Perforce (client)	FishEye needs access to the p4 client executable. Due to some problems with earlier versions of the client, we recommend version 2007.3 or later.
CVS	If you are using CVS, FishEye needs read-access to your CVS repository via the file system . It does not support protocols such as <code>pserver</code> at the moment.

Support for other version control systems is planned for future releases. Let us know what SCM system you would like to see supported by creating a [JIRA issue](#) or adding your vote to an issue, if the request already exists.

WAR deployment

FishEye/Crucible is a standalone Java program. It cannot be deployed to web application servers such as WebSphere, Weblogic or Tomcat.

Single sign on with Atlassian Crowd

From version 2.8.x, FishEye bundles the Crowd 2.4.1 client library, and supports the Crowd 2.4.x server, and later versions.

Native support for SVN

This page describes an advanced feature of FishEye's Subversion support. It explores the technical background, and some of the issues you may encounter, if you wish to use the native JavaHL access feature.

For most users we recommend that you use the default SVNKit Subversion access client that is bundled with FishEye. You are only likely to need the native JavaHL access described on this page for certain edge case repositories.

FishEye Subversion access

FishEye interacts with Subversion repositories through a layer, defined by the Apache Subversion project, known as JavaHL. This is the high-level Java language binding for Subversion. There are two implementations of the JavaHL interface available:

Bundled SVNKit

The SVNKit implementation is a largely Java-based implementation provided by the SVNKit project. It is bundled with FishEye and is the default JavaHL implementation used. As a Java implementation it operates on all of FishEye's supported platforms.

Native JavaHL

The JNI-based implementation, coupled with a shared dynamic library, is referred to here as native JavaHL. As a native library, native JavaHL is platform-dependent. The shared library is C-based and must be compatible with the remaining Subversion client components installed on the platform. It varies across each platform and distribution.

On this page:

- [FishEye Subversion access](#)
- [Native vs. bundled](#)
- [Native JavaHL support](#)
- [Installing JavaHL for your platform](#)
 - [Windows 7](#)
 - [Ubuntu 12](#)
 - [CentOS 6.4](#)
- [JavaHL considerations when upgrading FishEye](#)

Native vs. bundled

Given that FishEye bundles the SVNKit implementation, why might you want to use the native implementation of JavaHL? In general our recommendation is to stick with the bundled SVNKit implementation. It is the simplest to use and works in the widest variety of scenarios. Nevertheless, there are some scenarios where it may be desirable to use the native implementation, if it is available.

The two implementations have quite different characteristics – these can affect the decision about which to use. Here is a high-level list of some of the considerations we have encountered over the years:

Aspect	SVNKit - bundled with FishEye	Native JavaHL - platform dependent
Memory Usage	SVNKit uses the Java Heap. It therefore shares the heap that is being used for all FishEye's operations. It does benefit, however, from Java's garbage collection mechanism and we have not seen any memory leaks.	The native JavaHL implementation uses the native process heap and not the Java heap. It can increase the overall process memory usage but does not interfere with the Java heap usage. In some rare instances, we have seen memory leaks in the C-based JavaHL code. As FishEye is a long running service, these can cause problems over the life of the FishEye process.
Speed	In general, when using any of the Subversion network protocols, the JavaHL implementation speed is not a significant factor in the overall speed of Subversion operations as they are dominated by the network latency. Even for file:// access SVNKit is rarely the bottleneck.	If you are using file:// access to talk to a Subversion repository on the same server, then native JavaHL will most likely give the highest performance.
Compatibility	SVNKit has proven to be highly compatible with Subversion across all releases. The project is very responsive to bug reports when any differences become apparent. As an alternative implementation of JavaHL there will be differences between the SVNKit and the native Subversion JavaHL. This may affect some edge case repositories.	JavaHL uses predominantly the same code as Subversion itself so it is virtually 100% compatible.

Availability	SVNKit works on all of FishEye's supported platforms	It can be difficult to get an install of the JavaHL jar and shared library that is compatible with the version of Subversion installed on your platform.
--------------	--	--

Native JavaHL support

The native JavaHL interface and implementation naturally change with every release of Subversion. Normally these changes are incremental and backward compatible.

The compatibility matrix for recent FishEye versions is:

	SVNKit	Native JavaHL 1.6 client	Native JavaHL 1.7 client	Native JavaHL 1.8 client
FishEye 3.3+				
1.8 Subversion Server	✓	✗	✓	✓
1.7 Subversion Server	✓	✗	✓	✓
1.6 Subversion Server	✓	✗	✓	✓
FishEye 3.1 – 3.2				
1.8 Subversion Server (not file:// access)	✓	✗	✓	Unsupported
1.8 Subversion Server (file:// access)	✗	✗	✗	Unsupported
1.7 Subversion Server	✓	✗	✓	Unsupported
1.6 Subversion Server	✓	✗	✓	Unsupported
FishEye 3.0 and earlier				
1.7 Subversion Server (not file:// access)	✓	✓	✗	✗
1.7 Subversion Server (file:// access)	✓	✗	✗	✗
1.6 Subversion Server any access	✓	✓	✗	✗

Note that:

- FishEye 3.1, and later, is not compatible with the JavaHL 1.6 client (or older versions) – it uses the org.apache.subversion package which is not provided in 1.6 JavaHL builds
 - From FishEye 3.1, you must use a 1.7 or later JavaHL library if you want native JavaHL access.
- ✓ [Click here to read about compatibility changes from SVN 1.6 to 1.7...](#)

The change from Subversion 1.6 to Subversion 1.7 was much more significant and for a number of FishEye's usages of the interface, it broke compatibility.

In Subversion 1.7, the JavaHL interfaces were updated and moved from the org.tigris.subversion package to the org.apache.subversion package. This coincided with the move of the Subversion project to the Apache Software Foundation. In addition to moving the package, the interface was modernized in a number of ways:

- Extended use of callbacks.
- Use of Java collections rather than native arrays.
- Properties were clarified as byte arrays rather than Java Strings.
- Use of typed Enums rather than primitive integer and char fields.

The existing org.tigris package was retained in most 1.7 distributions and was implemented as an adapter

layer over the new org.apache package classes. Unfortunately a number of incompatibilities in the adapter layer meant that FishEye could not use the 1.7 native implementation:

1. Property returning methods were wrapped in a String constructor to convert from the byte[] type of the new interface to the String type used in the old interface. This meant that any null returns would throw NullPointerExceptions rather than returning null Strings.
2. Some of the callbacks were changed from plain interfaces to being interface extensions of the corresponding callback in the new package. This changed the type definition of the callback from an untyped Map to a typed Map. This caused ClassCastExceptions because the code is expecting a map containing byte[] but the underlying code was passing in a map containing strings.

For this reason, FishEye did not support native access using the 1.7 native library prior to FishEye 3.1.

Installing JavaHL for your platform

Atlassian FishEye bundles the SVNKit library to make connecting to your Subversion repository a painless process out of the box. If you do wish to use native JavaHL, it is your responsibility to install it onto your platform. Different organizations have different operating procedures and policies regarding how and what packages they are able to install on production servers.

In some cases the distribution you use will not provide a compatible JavaHL from an official package. In this case you will either need to build everything from source yourself (hard) or use a package from a Subversion vendor. We have used packages from two vendors over time, CollabNet and Wandisco. More recently, we have found it easier to use the Wandisco packages for JavaHL support.

The following sections detail our experiences when we investigated deploying JavaHL 1.7 on a variety of platforms. This is not a definitive list or guide. It is to give you an idea of some of the issues you are likely to encounter getting a compatible JavaHL install working on a range of platforms and distributions.

32-bit / 64-bit: The DLL file is platform dependent so it needs to match the architecture of your VM. The JAR file however, contains only metadata for the JVM about how to load the DLL so the same JAR will work across 32-bit and 64-bit operating systems as long as the subversion binaries match.

Windows 7

Windows does not include a Subversion client by default so you will need to install a Subversion package. We installed the 1.7.11 "client only" install from Wandisco. This installs Subversion, including the javahl components, in C:\Program Files\WANDisco\Subversion. It is interesting to note that the JavaHL package in this install does not include the org.tigris package adapter layer.

Ubuntu 12

Ubuntu provides packages for both core subversion and the JavaHL library for Subversion. We installed these for Subversion 1.7.5 as follows:

- `sudo apt-get install subversion`
- `sudo apt-get install libsvn-java`

Unfortunately the version installed seems to have a consistent assertion failure:

```
java: /build/buildd/subversion-1.7.5/subversion/libsvn_subr/dirent_uri.c:1519:
uri_skip_ancestor: Assertion `svn_uri_is_canonical(child_uri, ((void *)0))' failed.
```

We then removed the two Subversion packages from Ubuntu itself:

- `sudo apt-get remove libsvn-java`
- `sudo apt-get remove subversion`

We installed the Wandisco packages by downloading and running the Wandisco installer:

```
svn1.7_ubuntu_wandisco-precise.sh
```

This configures the Wandisco servers as a source of packages and installs the core Subversion install. At the time of writing this installed 1.7.11. Once installed, reinstall the javahl package:

- `sudo apt-get install libsvn-java`

This will now come from the Wandisco package repository. The location of the shared library and JavaHL jar is:

```
/usr/lib/jni/libsvnjavahl-1.so  
/usr/share/java/svn-javahl.jar
```

CentOS 6.4

If you install the Subversion packages (subversion and subversion-javahl) using yum, you will have a 1.6.11 install of Subversion which is not compatible with FishEye if you wish to use JavaHL as described above.

If you have previously upgraded to a version of SVN 1.7 before 1.7.11 you may see the message below in your logs. If you do, please upgrade to the Wandisco SVN 1.7 as described below:

```
java: /build/builddd/subversion-1.7.5/subversion/libsvn_subr/dirent_uri.c:1519:  
uri_skip_ancestor: Assertion `svn_uri_is_canonical(child_uri, ((void *)0))' failed.
```

You will need to remove the standard yum packages and use a Wandisco install, `svn1.7_centos6_wandisco.sh`. This installs plain subversion and configures the Wandisco servers as a source of packages. You can then use yum to install subversion-javahl. The following files are installed:

```
$ repoquery --list subversion-javahl  
/usr/lib/libsvnjavahl-1.so  
/usr/lib/svn-javahl/svn-javahl.jar  
/usr/lib64/libsvnjavahl-1.so  
/usr/lib64/svn-javahl/svn-javahl.jar
```

If you are using a 64bit JVM, use the `/usr/lib64` library, otherwise use the 32bit library in `/usr/lib`.

JavaHL considerations when upgrading FishEye

If you are currently using SVNKit with FishEye (the default), then you do not have to do anything when upgrading to FishEye 3.1 and later. FishEye will continue to use the bundled SVNKit library.

FishEye's Admin UI now displays information about the Subversion client in use – click **Server**, under 'Global Settings'. With no native client, configured, the display would look like:

Subversion client

The bundled Subversion client, SVNKit, is being used for Subversion operations.
The JavaHL client version is SVNKit v1.8.3.10190.

JAR *not set*

Dynamic library *not set*

Edit Details

If you have been using native JavaHL prior to FishEye 3.1, FishEye will detect that you have configured a pre-1.7 version of JavaHL and fallback to the bundled SVNKit client and start up normally. You will see the following in the **Server** section of the admin UI:

Subversion client

The configured Subversion client, a pre 1.7 native JavaHL client, is no longer supported.
The bundled Subversion client, SVNKit, is being used for Subversion operations.
Please update the configuration to a 1.7 or later version to continue using the native JavaHL Subversion client.

JAR `/opt/subversion/lib/svn-javahl/svn-javahl.jar`

Dynamic library `/opt/subversion/lib/libsvnjavahl-1.0.dylib`

[Edit Details](#)

You can use the FishEye admin UI to update the JavaHL client information to point FishEye to a 1.7 or later JavaHL jar and shared library. FishEye will perform some checks that the configured library supplies the correct classes. You will need to restart for the changes to take effect. If there are problems with the JavaHL library on restart, FishEye will again fallback to SVNKit. Once you have updated the configuration, FishEye will show a message that the configuration has been changed and a restart is required:

Subversion client

The Subversion client configuration has been changed and will take effect after restart.

JAR `not set`

Dynamic library `not set`

[Edit Details](#)

Upon restart, the display will show the operation of the native library and its version:

Subversion client

The configured Subversion client, native JavaHL, is being used for Subversion operations.
The JavaHL client version is 1.8.0 (r1490375).

JAR `/opt/subversion/lib/svn-javahl/svn-javahl.jar`

Dynamic library `/opt/subversion/lib/libsvnjavahl-1.0.dylib`

[Edit Details](#)

Installing FishEye on Windows

This page...

... describes how to perform a clean install of FishEye on Windows.

Upgrading?

If you're upgrading your FishEye installation, read the [FishEye upgrade guide](#) first.

Using Linux or Mac OS X?

If you're on one of these platforms read [Installing FishEye on Linux and Mac ins](#)

tead.

1. Check supported platforms

Better check the [Supported platforms](#) page first; it lists the application servers, databases, operating systems, web browsers and JDKs that we have tested FishEye with, and that we recommend.

Atlassian only officially supports FishEye running on x86 hardware and 64-bit derivatives of x86 hardware.

2. Create a dedicated FishEye user (recommended)

For production installations, we recommend that you create a new dedicated Windows user that will run FishEye on your system. This user:

- Should not have admin privileges.
- Should be a non-privileged user with read, write and execute access on the FishEye install directory and instance (data) directory. These directories are described below.
- Should only have read access to your repositories.

If you have created a dedicated FishEye user, ensure you are logged in as this user to complete the remaining instructions.

3. Check your version of Java

In a command prompt, run this:


```
java -version
```

The version of Java should be **1.8.x**.

The recommended way to install FishEye is to use the installer, which installs FishEye as a Windows service – see step 5 below.

▼ [If you don't see a supported version of Java, then get Java...](#)

Download and install the Java Platform JDK from [Oracle's website](#).

 *We recommend that the Java install path should not contain spaces, so don't install into C:\Program Files\Java\. Instead, use a path like C:\Java.*

Now try running 'java -version' again to check the installation. The version of Java should be **1.8.x**.

4. Check that Windows can find Java

Windows uses the JAVA_HOME environment variable to find Java. To check that, in a new command prompt, run:

```
echo %JAVA_HOME%
```

You should see a path to the Java install location. We recommend that this path does *not* contain spaces, and that JAVA_HOME should point to the JDK home path.

▼ [If you don't see a path without spaces...](#)

- If you see a path with spaces, like `C:\Program Files\Java\`, then sorry, but go back to 3. and reinstall Java to a location that doesn't have spaces.
- If you don't see a path at all, or if you just see `%JAVA_HOME%`, then set `JAVA_HOME` as follows:

For Windows 7:

1. Go to **Start**, search for "sys env" and choose **Edit the system environment variables**.
2. Click **Environment Variables**, and then **New** under 'System variables'.
3. Enter "JAVA_HOME" as the **Variable name**, and the absolute path to where you installed the Java JDK as the **Variable value**, that is, something like `C:\Java\jdk1.7.0_51`. Don't use a trailing backslash. We recommend that `JAVA_HOME` should point to the JDK home path.
4. Now, in a new command prompt, try running '`java -version`'. You should see the same version of Java as you saw above.

5. Now it's time to get FishEye

Download the FishEye installer from the Atlassian download site.

There are 32-bit and 64-bit installers for FishEye on Windows. Each installer adds FishEye as a Windows service, and starts the service, automatically. The express install creates, by default, a `Data` directory and a separate install directory in `C:\Atlassian`. The custom install mode allows you to choose different locations for the install and `Data` directories, with the restriction that the `Data` directory must not be contained in the install directory.

- The installer creates the `FISHEYE_INST` system environment variable. This points to the location of the instance (data) directory.
- The path to the installation location is referred to as the `<FishEye install directory>` in these instructions.
- You need separate FishEye instance (data) directories if you want to run multiple copies of FishEye.
- If you expect to have a large number of users for this FishEye installation, and FishEye will be [connected to an external database](#), consider installing FishEye on a different server from the one running the external database, for improved performance.
- If you have a large number of repositories, we recommend you increase the default number of files that FishEye is allowed to open. See the following knowledge base article for more info: [Subversion Indexer Paused with "Too many open files" Error](#).
- For FishEye 3.4.4 and later, you can edit JVM parameters for the Windows service by going to **Start > All Programs > FishEye > Configure FishEye**. Ensure that you restart the FishEye service when finished. Do *not* reference any environment variables in the settings (e.g. `%FISHEYE_INST%`). Instead, set the actual path.

6. Visit FishEye!

Give the FishEye service a minute to launch. Then, in a web browser on the same machine, go to `http://localhost:8060/` (or, from another machine, type `http://hostname:8060/`, where `hostname` is the name of the machine where you installed FishEye).

Enter your license, then an admin password, to finish the setup. Note that this password is for the 'built-in' FishEye admin user. You can log in as this user, if necessary, by clicking the **Administration** link in the page footer. See also [How to reset the Administration Page password in FishEye or Crucible](#).

You can postpone setting up JIRA integration until later if you wish; see [Configuring JIRA integration in the Setup Wizard](#).

7. Add repositories

Now you can tell FishEye about any existing repositories you have. Please read [Starting to use FishEye](#) for the details.

FishEye will perform an initial index of your repositories, during which it accesses, indexes and organizes a view of your repositories (including all historical items) back to the earliest commits. If you are evaluating

FishEye, we suggest that you index a single project, so you can use FishEye as soon as possible. If you choose to index your entire repository, be aware that this can take a long time (possibly days) for massive or complex repositories and can be more complex to set up (especially for Subversion). The basic process is slightly different for each SCM type.

8. Add users and groups

You will want to set up your users and groups in FishEye. You can [add users directly](#) to FishEye, or connect to an [external user directory](#). Please read [Starting to use FishEye](#) for an introduction.

9. Set up your mail server

Configure the FishEye email server so that users can get notifications from FishEye. See [Configuring SMTP](#).

10. Connect to an external database (recommended)

If you intend to use this FishEye installation in a production environment, it is highly recommended that you use one of the [supported](#) external databases. See [Migrating to an external database](#).

If you are evaluating FishEye, or don't wish to do this now, FishEye will happily use its embedded HSQL database, and you can easily migrate later.

11. Stop FishEye (optional)

Control the FishEye service from the Windows administration console. Alternatively, in a command prompt, change directory to <FishEye install directory> and run this:

```
bin\stop.bat
```

12. Tuning FishEye performance

To get the best performance from your new FishEye installation, please consult [Tuning FishEye performance](#).

Installing FishEye on Linux and Mac

Hey! We're going to install FishEye on a Linux box, or a Mac. There are a few steps involved, but we think you'll find it easy to follow along. If you are upgrading an existing installation, please refer to the [FishEye upgrade guide](#) instead.

1. Check supported platforms

Better check the [Supported platforms](#) page first; it lists the application servers, databases, operating systems, web browsers and JDKs that we have tested FishEye with, and that we recommend.

Atlassian only officially supports FishEye running on x86 hardware and 64-bit derivatives of x86 hardware.

Related pages:

- [Installing FishEye on Windows](#)
- [Starting to use FishEye](#)
- [Supported platforms](#)
- [FishEye upgrade guide](#)

2. Create a dedicated FishEye user (recommended)

For production installations, we recommend that you create a new dedicated user that will run FishEye on your system. This user:

- Should not have admin privileges.
- Should be a non-privileged user with read, write and execute access on the FishEye install directory and instance (data) directory. These directories are described below.
- Should only have read access to your repositories.

If you created a dedicated FishEye user, ensure you are logged in as this user to complete the remaining instructions.

3. Check your version of Java

In a terminal, run this:

```
java -version
```

The version of Java should be **1.8.x**.

▼ If you don't see a supported version of Java, then get Java...

Download and install the [Oracle JVM](#) (JDK or JRE), or [OpenJDK](#).

Now try running 'java -version' again to check the installation. The version of Java should be **1.8.x**.

4. Check that the system can find Java

In a terminal, run this:

```
echo $JAVA_HOME
```

You should see a path like `/System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Home/`.

▼ If you don't see a path to the Java location, then set JAVA_HOME...

Linux

Do either of the following:

- If `JAVA_HOME` is not set, log in with 'root' level permissions and run:

```
echo JAVA_HOME="path/to/JAVA_HOME" >> /etc/environment
```

where `path/to/JAVA_HOME` may be like: `/System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Home/`

- If `JAVA_HOME` needs to be changed, open the `/etc/environment` file in a text editor and modify the value for `JAVA_HOME` to:

```
JAVA_HOME="path/to/JAVA_HOME"
```

It should look like:

```
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Home/
```


5. Now it's time to get FishEye

1. Download FishEye from the Atlassian download site.

2. Please check your unzip program before extracting the downloaded zip file. Some archive-extract programs cause errors when unzipping the FishEye zip file:
 - Windows users must avoid the Windows built-in unzip utility, as it doesn't extract all the files. Use a third-party unzip program like [7-Zip](#) or [Winzip](#).
 - Solaris users will need to use [GNU tar](#) to handle the long file names.
3. Extract the downloaded file to an install location:
 - Folder names in the path to your FishEye executable should not have spaces in them. The path to the extracted directory is referred to as the `<FishEye install directory>` in these instructions.
 - If you expect to have a large number of users for this FishEye installation, and FishEye will be [connected to an external database](#), consider installing FishEye on a different server from the one running the external database, for improved performance.

6. Tell FishEye where to store your data

The FishEye instance directory is where your FishEye data is stored.

 You *should not* locate your FishEye instance directory inside the `<FishEye install directory>` — they should be entirely separate locations. If you do put the instance directory in the `<FishEye install directory>` it will be overwritten, and lost, when FishEye gets upgraded. And by the way, you'll need separate FishEye instance directories if you want to run multiple copies of FishEye.

For production installations, we recommend that the FishEye instance directory be [secured against unauthorized access](#).

Create your FishEye instance directory, and then tell FishEye where you created it by adding a `FISHEYE_INST` environment variable as follows:

Linux	Mac
Open the <code>/etc/environment</code> file in a text editor and insert: <pre>FISHEYE_INST="path/to/<FishEye instance directory>"</pre>	Open the <code>~/.profile</code> file for the current user in a text editor and insert: <pre>FISHEYE_INST="path/to/<FishEye instance directory>" export FISHEYE_INST</pre>
You need to log out and log in again so the new environment variable is set.	
<div> Alternatively you can run <pre>export FISHEYE_INST="path/to/<FishEye instance directory>"</pre> to avoid logging out and in. </div>	

Now, copy the `<FishEye install directory> /config.xml` to the root of the `FISHEYE_INST` directory, so that FishEye can start properly.

Also, if you have a large number of repositories, we recommend you increase the default number of files that FishEye is allowed to open. See the following knowledge base article for more info: [Subversion Indexer Paused with "Too many open files" Error](#).

7. Start FishEye!

In a terminal, change directory to `<FishEye install directory>` and run this:

```
bin/start.sh
```

After a few moments, in a web browser on the same machine, go to `http://localhost:8060/` (or, from another machine, type `http://hostname:8060/`, where `hostname` is the name of the machine where you installed FishEye).

Enter your license, then an admin password, to finish the setup. Note that this password is for the 'built-in' FishEye admin user. You can log in as this user, if necessary, by clicking the **Administration** link in the page footer.

You can postpone setting up JIRA integration until later if you wish; see [Configuring JIRA integration in the Setup Wizard](#).

8. Add repositories

Now you can tell FishEye about any existing repositories you have. Please read [Starting to use FishEye](#) for the details.

FishEye will perform an initial index of your repositories, during which it accesses, indexes and organizes a view of your repositories (including all historical items) back to the earliest commits. If you are evaluating FishEye, we suggest that you index a single project, so you can use FishEye as soon as possible. If you choose to index your entire repository, be aware that this can take a long time (possibly days) for massive or complex repositories and can be more complex to set up (especially for Subversion). The basic process is slightly different for each SCM type.

9. Add users and groups

You will want to set up your users and groups in FishEye. You can [add users directly](#) to FishEye, or connect to an [external user directory](#). Please read [Starting to use FishEye](#) for an introduction.

10. Set up your mail server

Configure the FishEye email server so that users can get notifications from FishEye. See [Configuring SMTP](#).

11. Connect to an external database (recommended)

If you intend to use this FishEye installation in a production environment, it is highly recommended that you use one of the [supported](#) external databases. See [Migrating to an external database](#).

If you are evaluating FishEye, or don't wish to do this now, FishEye will happily use its embedded HSQL database, and you can easily migrate later.

12. Stop FishEye (optional)

In a terminal, change directory to `<FishEye install directory>` and run this:

```
bin/stop.sh
```

13. Tuning FishEye performance

To get the best performance from your new FishEye installation, please consult [Tuning FishEye performance](#).

Starting to use FishEye

This page will guide you through the basics of using FishEye. By the end of it you should be able to:

- Create accounts for your collaborators, and organize them into groups.
- Add repositories that need to be indexed and setup permissions.
- Use the Commit Graph to trace the history of your code

This page assumes that:

- You have installed and started the latest version of FishEye. See [Installing FishEye on Linux and Mac](#) or [Installing FishEye on Windows](#) for details.
- You are using a [supported browser](#).

On this page:

- [Create users in FishEye](#)
- [Add a repository](#)
- [Move forward](#)

Related pages:

- [Installing FishEye on Windows](#)
- [Installing FishEye on Linux and Mac](#)
- [Supported platforms](#)
- [Managing your repositories](#)
- [Setting up your Users and Security](#)

Create users in FishEye

FishEye doesn't have any user accounts after you have installed it for the first time. You need to go to the Administration interface to add the first users of the system.

Click on the **Administration** link in the footer:

Atlassian FishEye analysis with Crucible code review. (Version:3.10.0 Build:r62dc31d 2015-08-07) - Administration - Page generated 2015-08-11 04:30 +0000

In the **Users** listing page click **Add user** to go to the user creation form:

Users ?

The User Browser allows you to browse all the users in the system. Filters allow you to limit the users that you see.

[Reset filter](#)

Filter In group Any Filter Add user

Fill in the form and create your user:

in

Settings
es

Groups
Users
Repositories

URL Mappings
Customization
Links
Logs

Add New User

Information about the new user

Username

Display name

Email

Auth Type

Password

Confirm Password

From the **User** page you can click on **Create another user** to repeat this operation:

User

The user John Doe has been created.
[Create another user.](#)

User Details

Username **jdoe**

Display name **John Doe**

Email **john@doe.com**

Type **built-in**

Admin **false**

[Edit](#) | [Change Password](#) | [Rename](#) | [Delete](#)

Group Details

Groups **No groups.**

[Edit Groups](#)

List of users

[Return to the list of users](#)

Add a repository

In this section we're going to add a repository to FishEye.

Click on **Add repository** in the **Repositories** listing of the Administration.

Now choose the repository type and fill in the name and description:

Add Repository - Page 1 of 3

Basic Details

Repository Type: Mercurial

Name: * ?

Description: ?

Back Next Add Cancel

In the repository configuration, add the location of your repository. Fill in the authentication details if necessary.

Add Repository - Page 2 of 3

Mercurial Connection Details

Repository Location: * ?

Mercurial Authentication

Authentication Style: No authentication ✓

FishEye will not add any authentication to requests. Authentication will be handled transparently by the SCM.

☐ Show advanced settings

Test Connection Back Next Add Cancel

Finally indicate whether or not you would like diff indexing should be turned on and if the repository should not be indexed right away.

Click **Add** to finish the process.

The screenshot shows the 'Add Repository - Page 3 of 3' window. Under the 'Final Settings' section, there are two options: 'Store Diff Info:' with an unchecked checkbox and a help icon, and 'Enable Repository After Adding:' with a checked checkbox and a help icon. At the bottom of the window, there are five buttons: 'Test Connection', 'Back', 'Next', 'Add', and 'Cancel'.

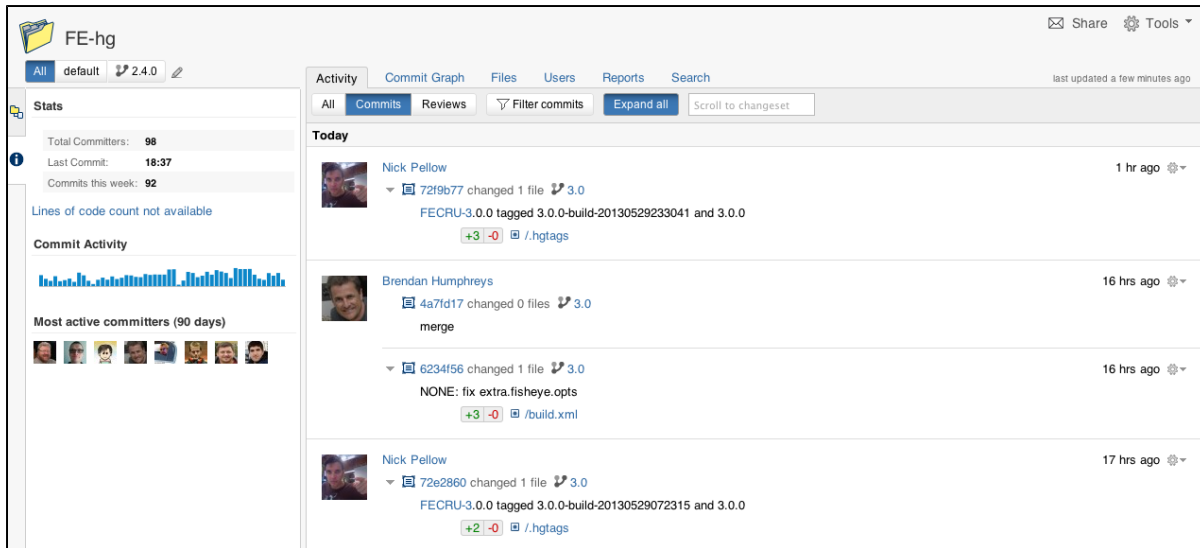
Move forward

Once it's created you can click **Browse**, in the repository options menu, to access your repository.

The screenshot shows the 'Repositories' page. It has a tab for 'Native Repository Access (1)'. Below the tab, it says 'FishEye natively supports Subversion, CVS, Perforce, Git and Mercurial repositories' with a green checkmark. There are buttons for 'Add Existing...', 'Create New...', 'Refresh', and a 'Filter repositories' input field. A pagination bar shows 'Page 1 of 1'. Below this is a table with the following columns: Name, Type, Description, Location, State, Last Update, and Actions.

Name	Type	Description	Location	State	Last Update	Actions
jqplot	Mercurial	JS charts library	https://bitbucket.org/cleonello/jqplot/	Running		 View Browse Stop Restart

You can now browse your files in FishEye, search through your code or track modifications via the commit graph.



Configuring JIRA integration in the Setup Wizard

This page describes the **Connect to JIRA** tab of the FishEye setup wizard.

Connecting FishEye to a JIRA application allows you to manage your users with JIRA. See [Connecting to JIRA for user management](#) for more information.

On this page:

- [Connecting to a JIRA application in the Setup Wizard](#)
- [Troubleshooting](#)

Related pages:

- [Starting to use FishEye](#)
- [Linking FishEye to JIRA applications](#)
- [JIRA Integration in FishEye](#)

Connecting to a JIRA application in the Setup Wizard

To configure JIRA integration while running the FishEye setup wizard:

1. Configure the following setting in the JIRA application: [Allow remote API access](#).
2. Enter the following information on the 'Connect to JIRA' step of the setup wizard:

JIRA Base URL	The base URL set for your JIRA application. Examples are: <code>http://www.example.com:8080/jira/</code> <code>http://jira.example.com</code>
Admin Username	The credentials for a user with the 'JIRA System Administrators' global permission in the JIRA application.
Admin Password	
FishEye Base URL	Click Advanced Options to see this field. The JIRA application will use this URL to access your FishEye server. The URL you give here will override the base URL specified in your FishEye administration console, for the purposes of the JIRA connection.
Groups to Synchronize	Click Advanced Options to see this field. Select at least one JIRA application group to synchronize. The default group is <code>jira-users</code> . The JIRA application will synchronize all changes in the user information on a regular basis. The default synchronization interval is 1 hour.

Admin Groups

Click **Advanced Options** to see this field. Specify a JIRA group whose members should have administrative access to FishEye/Crucible. The default group is `jira-administrators`.

3. Click **Connect to JIRA**.
4. Finish the setup process.

Troubleshooting

▼ [Click to see troubleshooting information...](#)

This section describes the possible problems that may occur when integrating your application with JIRA via the setup wizard, and the solutions for each problem.

Symptom	Cause	Solution
<p>The setup wizard displays one of the following error messages:</p> <ul style="list-style-type: none"> Failed to create application link from JIRA server at <URL> to this <application> server at <URL>. Failed to create application link from this <application> server at <URL> to JIRA server at <URL>. Failed to authenticate application link from JIRA server at <URL> to this <application> server at <URL>. Failed to authenticate application link from <application> server at <URL> to this JIRA server at <URL>. 	<p>The setup wizard failed to complete registration of the peer-to-peer application link with JIRA. JIRA integration is only partially configured.</p>	<p>Remove the partial configuration if it exists, try the 'Connect to JIRA' step again, and then continue with the setup. Detailed instructions are below.</p>

<p>The setup wizard displays one of the following error messages:</p> <ul style="list-style-type: none"> Failed to register <application> configuration in JIRA for shared user management. Received invalid response from JIRA: <response> Failed to register <application> configuration in JIRA for shared user management. Received: <response> 	<p>The setup wizard failed to complete registration of the client-server link with JIRA for user management. The peer-to-peer link was successfully created, but integration is only partially configured.</p>	<p>Remove the partial configuration if it exists, try the 'Connect to JIRA' step again, and then continue with the setup. Detailed instructions are below.</p>
<p>The setup wizard displays the following error message:</p> <ul style="list-style-type: none"> Error setting Crowd authentication 	<p>The setup wizard successfully established the peer-to-peer link with JIRA, but could not persist the client-server link for user management in your <code>config.xml</code> file. This may be caused by a problem in your environment, such as a full disk.</p>	<p>Please investigate and fix the problem that prevented the application from saving the configuration file to disk. Then remove the partial configuration if it exists, try the 'Connect to JIRA' step again, and then continue with the setup. Detailed instructions are below.</p>
<p>The setup wizard displays the following error message:</p> <ul style="list-style-type: none"> Error reloading Crowd authentication 	<p>The setup wizard has completed the integration of your application with JIRA, but is unable to start synchronizing the JIRA users with your application.</p>	<p>Restart your application. You should then be able to continue with the setup wizard. If this solution does not work, please contact Atlassian Support.</p>
<p>The setup wizard displays the following error message:</p> <ul style="list-style-type: none"> An error occurred: <code>java.lang.IllegalStateException: Could not create the application in JIRA/Crowd (code: 500)</code>. Please refer to the logs for details. 	<p>The setup wizard has not completed the integration of your application with JIRA. The links are only partially configured. The problem occurred because there is already a user management configuration in JIRA for this <application> URL.</p>	<p>Remove the partial configuration if it exists, try the 'Connect to JIRA' step again, and then continue with the setup. Detailed instructions are below.</p>
<p>No users can log in after you have set up the application with JIRA integration.</p>	<p>Possible causes:</p> <ul style="list-style-type: none"> There are no users in the group that you specified on the 'Connect to JIRA' screen. For FishEye: There are no groups specified in the 'groups to synchronize' section of your administration console. For Stash: You may not have granted any JIRA groups or users permissions to log in to Stash. 	<p>Go to JIRA and add some usernames to the group.</p> <ul style="list-style-type: none"> For FishEye: Go to the FishEye administration screens and specify at least one group to synchronize. The default is 'jira-users'. For Stash: Grant the Stash User permission to the relevant JIRA groups on the Stash Global permissions page. <p>If this solution does not work, please contact Atlassian Support.</p>

Solution 1: Removing a Partial Configuration – The Easiest Way

If the application's setup wizard fails part-way through setting up the JIRA integration, you may need to remove the partial configuration from JIRA before continuing with your application setup. Please follow the

steps below.

Remove the partial configuration if it exists, try the 'Connect to JIRA' step again, and then continue with the setup wizard:

1. Log in to JIRA as a user with the '**JIRA System Administrators**' global permission.
2. Click the '**Administration**' link on the JIRA top navigation bar.
3. Remove the application link from JIRA, if it exists:
 - a. Click **Application Links** in the JIRA administration menu. The 'Configure Application Links' page will appear, showing the application links that have been set up.
 - b. Look for a link to your application. It will have a base URL of the application linked to JIRA. For example:
 - If you want to remove a link between JIRA and FishEye, look for the one where the **Application URL** matches the base URL of your FishEye server.
 - If you want to remove a link between JIRA and Confluence, look for the one where the **Application URL** matches the base URL of your Confluence server.
 - If you want to remove a link between JIRA and Stash, look for the one where the **Application URL** matches the base URL of your Stash server.
 - c. Click **Delete** next to the application link that you want to delete.
 - d. A confirmation screen will appear. Click **Confirm** to delete the application link.
4. Remove the user management configuration from JIRA, if it exists:
 - a. Go to the JIRA administration screen for configuring the applications that have been set up to use JIRA for user management:
 - In JIRA 4.3: Click '**Other Applications**' in the '**Users, Groups & Roles**' section of the JIRA administration screen.
 - In JIRA 4.4: Select '**Administration**' > '**Users**' > '**JIRA User Server**'.
 - b. Look for a link to your application. It will have a name matching this format:

```
<Type> - <HostName> - <Application ID>
```

For example:

```
FishEye / Crucible - localhost -  
92004b08-5657-3048-b5dc-f886e662ba15
```

Or:

```
Confluence - localhost -  
92004b08-5657-3048-b5dc-f886e662ba15
```

If you have multiple servers of the same type running on the same host, you will need to match the application ID of your application with the one shown in JIRA. To find the application ID:

- Go to the following URL in your browser:

```
<baseUrl>/rest/applinks/1.0/manifest
```

Replace `<baseUrl>` with the base URL of your application.

For example:

```
http://localhost:8060/rest/applinks/1.0/manifest
```

- The application links manifest will appear. Check the application ID in the `<id>` element.
 - c. In JIRA, click '**Delete**' next to the application that you want to remove.
5. Go back to the setup wizard and try the 'Connect to JIRA' step again.

Solution 2: Removing a Partial Configuration – The Longer Way

If solution 1 above does not work, you may need to remove the partial configuration and then add the full integration manually. Please follow these steps:

1. Skip the 'Connect to JIRA' step and continue with the setup wizard, to complete the initial configuration of the application.
2. Log in to JIRA as a user with the '**JIRA System Administrators**' global permission.
3. Click the '**Administration**' link on the JIRA top navigation bar.
4. Remove the application link from JIRA, if it exists:
 - a. Click **Application Links** in the JIRA administration menu. The 'Configure Application Links' page will appear, showing the application links that have been set up.
 - b. Look for a link to your application. It will have a base URL of the application linked to JIRA. For example:
 - If you want to remove a link between JIRA and FishEye, look for the one where the **Application URL** matches the base URL of your FishEye server.
 - If you want to remove a link between JIRA and Confluence, look for the one where the **Application URL** matches the base URL of your Confluence server.
 - If you want to remove a link between JIRA and Stash, look for the one where the **Application URL** matches the base URL of your Stash server.
 - c. Click **Delete** next to the application link that you want to delete.
 - d. A confirmation screen will appear. Click **Confirm** to delete the application link.
5. Remove the user management configuration from JIRA, if it exists:
 - a. Go to the JIRA administration screen for configuring the applications that have been set up to use JIRA for user management:
 - In JIRA 4.3: Click '**Other Applications**' in the '**Users, Groups & Roles**' section of the JIRA administration screen.
 - In JIRA 4.4: Select '**Administration**' > '**Users**' > '**JIRA User Server**'.
 - b. Look for a link to your application. It will have a name matching this format:

```
<Type> - <HostName> - <Application ID>
```

For example:

```
FishEye / Crucible - localhost -  
92004b08-5657-3048-b5dc-f886e662ba15
```

Or:

```
Confluence - localhost -  
92004b08-5657-3048-b5dc-f886e662ba15
```

If you have multiple servers of the same type running on the same host, you will need to match the application ID of your application with the one shown in JIRA. To find the application ID:

- Go to the following URL in your browser:

```
<baseUrl>/rest/applinks/1.0/manifest
```

Replace `<baseUrl>` with the base URL of your application.

For example:

```
http://localhost:8060/rest/applinks/1.0/manifest
```

- The application links manifest will appear. Check the application ID in the `<id>` element.
- c. In JIRA, click '**Delete**' next to the application that you want to remove.

6. Add the application link in JIRA again, so that you now have a two-way trusted link between JIRA and your application:
 - a. Click **Add Application Link**. Step 1 of the link wizard will appear.
 - b. Enter the **server URL** of the application that you want to link to (the 'remote application').
 - c. Click **Next**.
 - d. Enter the following information:
 - **Create a link back to this server** – Check to add a two-way link between the two applications.
 - **Username and Password** – Enter the credentials for a username that has administrator access to the remote application.
Note: These credentials are only used to authenticate you to the remote application, so that Application Links can make the changes required for the new link. The credentials are not saved.
 - **Reciprocal Link URL** – The URL you give here will override the base URL specified in your remote application's administration console, for the purposes of the application links connection. Application Links will use this URL to access the remote application.
 - e. Click **Next**.
 - f. Enter the information required to configure authentication for your application link:
 - **The servers have the same set of users** – Check this box, because the users are the same in both applications.
 - **These servers fully trust each other** – Check this box, because you trust the code in both applications and are sure both applications will maintain the security of their private keys.
For more information about configuring authentication, see [Configuring authentication for an application link](#).
 - g. Click **Create**.
7. Configure a new connection for user management in JIRA:
 - a. Go to the JIRA administration screen for configuring the applications that have been set up to use JIRA for user management:
 - In JIRA 4.3: Click '**Other Applications**' in the '**Users, Groups & Roles**' section of the JIRA administration screen.
 - In JIRA 4.4: Select '**Administration**' > '**Users**' > '**JIRA User Server**'.
 - b. **Add** an application.
 - c. Enter the **application name** and **password** that your application will use when accessing JIRA.
 - d. Enter the **IP address** or addresses of your application. Valid values are:
 - A full IP address, e.g. 192.168.10.12.
 - A wildcard IP range, using CIDR notation, e.g. 192.168.10.1/16. For more information, see the introduction to [CIDR notation on Wikipedia](#) and [RFC 4632](#).
 - **Save** the new application.
8. Set up the JIRA user directory in the application.
 - For Confluence:
 - a. Go to the **Confluence Administration Console**.
 - b. Click '**User Directories**' in the left-hand panel.
 - c. **Add** a directory and select type '**Atlassian JIRA**'.
 - d. Enter the following information:
 - **Name** – Enter the name of your JIRA server.
 - **Server URL** – Enter web address of your JIRA server. Examples:


```
http://www.example.com:8080/jira/
http://jira.example.com
```
 - **Application name** and **Application password** – Enter the values that you defined for Confluence in the settings on JIRA.
 - e. Save the directory settings.
 - f. Define the **directory order** by clicking the blue up- and down-arrows next to each directory on the '**User Directories**' screen.
For details see [Connecting to Crowd or JIRA for User Management](#).
 - For FishEye/Crucible:
 - a. Click **Authentication** (under 'Security Settings').
 - b. Click **Setup JIRA/Crowd authentication**. Note, if LDAP authentication has already been

set up, you will need to remove that before connecting to JIRA for user management.

c. Make the following settings:

Authenticate against	Select a JIRA instance
Application name and password	Enter the values that you defined for your application in the settings on JIRA.
JIRA URL	The web address of your JIRA server. Examples: <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"><code>http://www.example.com:8080/jira/</code> <code>http://jira.example.com</code></div>
Auto-add	Select Create a FishEye user on successful login so that your JIRA users will be automatically added as a FishEye user when they first log in.
Periodically synchronise users with JIRA	Select Yes to ensure that JIRA will synchronize all changes in the user information on a regular basis. Change the value for Synchronise Period if required.
When Synchronisation Happens	Select an option depending on whether you want to allow changes to user attributes from within FishEye.
Single Sign On	Select Disabled . SSO is not available when using JIRA for user management and if enabled will make the integration fail.

d. Click **Next** and select at least one user group to be synchronised from JIRA. If necessary, you could create a new group in JIRA, such as 'fisheye-users', and select this group here.

e. Click **Save**.

- For Stash:

- a. Go to the Stash administration area.

- b. Click **User Directories** in the left-hand panel.

- c. **Add** a directory and select type **Atlassian JIRA**.

- d. Enter the following information:

- **Name** – Enter the name of your JIRA server.

- **Server URL**– Enter web address of your JIRA server. Examples:

`http://www.example.com:8080/jira/`
`http://jira.example.com`

- **Application name and Application password** – Enter the values that you defined for Stash in the settings on JIRA.

- e. Save the directory settings.

- f. Define the directory order by clicking the blue up- and down-arrows next to each directory on the 'User Directories' screen.

For details see [Connecting Stash to JIRA for user management](#).

Using FishEye

- [Using the FishEye screens](#)
 - [Browsing through a repository](#)

- Viewing file content
- Viewing a file history
- Using side by side diff view
- Viewing the changelog
- Searching FishEye
- FishEye charts
- Using favourites in FishEye
- Changeset discussions
- Viewing the commit graph for a repository
- Viewing people's statistics
- Using Smart Commits
- Changing your user profile
 - Re-setting your password
- Pattern matching guide
- Date expressions reference guide
- EyeQL reference guide
- Integrating FishEye with Atlassian applications
 - JIRA Integration in FishEye
 - Integrating FishEye with Bitbucket Server
- Transitioning issues in JIRA

Using the FishEye screens

The sections below describe the different screens in FishEye and the information you can retrieve from them. Each page (tab) has a number of panes, each of which is described separately below.

Header



The header along the top of each FishEye page provides the following:

- The application navigator, at the left of the header, connects you directly to your other applications, such as JIRA and Bamboo. Admins can configure which apps appear in the navigator – just click **Application navigator** in the admin area.
- FishEye logo (with the logo for Crucible if you are using that) – click to go to the dashboard, to see your personal code commits, your reviews (if you are using [Crucible](#)) and your activity stream.
- **Repositories** — the list of all FishEye repositories. Click a repository name to [browse the repository](#). A number of sub-tabs become available, as described below (see 'Repository sub-tabs' below).
- **Projects** (*when used with Crucible*) – a link to all projects (see the [Crucible documentation](#)). Logged-in users can see links to recently visited projects.
- **People** – tab to view statistics about committers to your FishEye repositories (see [Viewing People's Statistics](#)). Logged in users can see links to users they have recently visited.
- **Reviews** (*when used with Crucible*) – go to your code reviews (see the [Crucible documentation](#)). Logged-in users can see links to recently visited reviews as well as to the Crucible 'Inbox' and 'Outbox'.
- **Create review** (*when used with Crucible*) – click the down arrow to choose **Create snippet**.
- **+** – add a new repository. See [Adding an external repository](#).
- **Search** – use FishEye's powerful search engine to find changesets, committers and files. See [Searching FishEye](#).
- Click your avatar to change your user settings (see [Changing your user profile](#)).

Repository tabs

Once you have selected a repository, you can navigate through it by selecting files and folders in the navigation tree on the left. The available tabs change according to whether you are viewing a repository or a file:

Tab	Repo	File	Description
-----	------	------	-------------

Files	✓		Provides details about the files in the repository. See Browsing through a repository .
Revisions		✓	Shows the latest revisions of the file. See Viewing a file history .
Activity	✓	✓	Shows recent activity on the item. There are a number of sub-options here (see Viewing the changelog): <ul style="list-style-type: none"> • All — The default view, showing commits, reviews (when used with Crucible) and JIRA issues (when used with JIRA). • Commits — Shows commits in the activity stream. • Reviews — Shows review activity in the activity stream (when used with Crucible). • Filter — Applies constraints to the current activity stream. • Scroll to Changeset — Opens the changeset ID specified in the text field (press Enter to carry out the action).
Commit graph	✓		Provides a visual representation of commits to, and branches in, the repository. See Viewing the commit graph for a repository .
Users	✓	✓	Shows the commit history of the different users that have committed changes on the item.
Reports	✓	✓	Shows activity charts for the item. Various chart options can be selected in the left navigation bar (see FishEye charts).
Source		✓	Shows the contents of the file. See Viewing file content .
Search	✓		Gives access to the advanced search capabilities in FishEye.

Browsing through a repository

Browse your repositories in FishEye to see information about the files in the repository and related activity, including commits to the repository. You can also generate charts, and search for specific file revisions in the repository.

On this page:

- [Browse a repository](#)
- [View information about a repository or directory](#)
- [View information about a file](#)
- [Hide empty directories and deleted files](#)
- [Watch a repository](#)

Browse a repository


1. Click **Repositories** in the header and choose either a recently viewed repository, or **All repositories**.
2. Click the name of a repository to view its contents.
3. If required, use the branch/tag selector (just above the file tree) to change context.

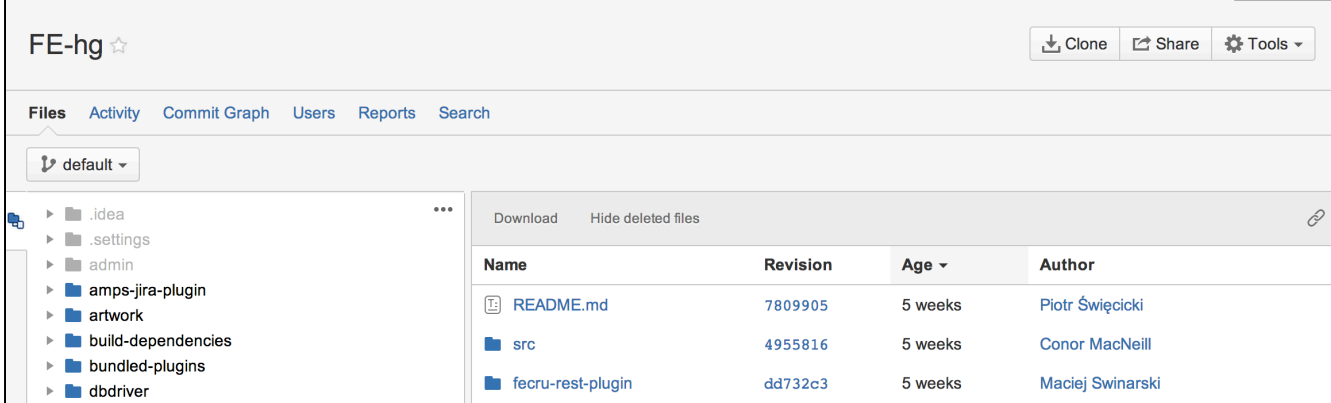
View information about a repository or directory

Click a folder/directory name to see information about that on these tabs:

Files	View the contents of the repository/folder.
Activity	View the commit, review and issue (requires JIRA) activity related to the repository/folder. The activity stream is similar to the changelog activity stream. See Viewing the changelog .
Commit Graph	Visualize the repository, using the commit graph. See Viewing the commit graph for a repository .

Users	View the commit history of the users that have committed changes to files in the repository/folder. See Viewing People's Statistics .
Reports	View activity charts for the repository/folder. See FishEye charts .
Search	Search the repository/folder. See Searching FishEye .

 A greyed out item is either deleted or empty.



FE-hg ☆

Files Activity Commit Graph Users Reports Search

default

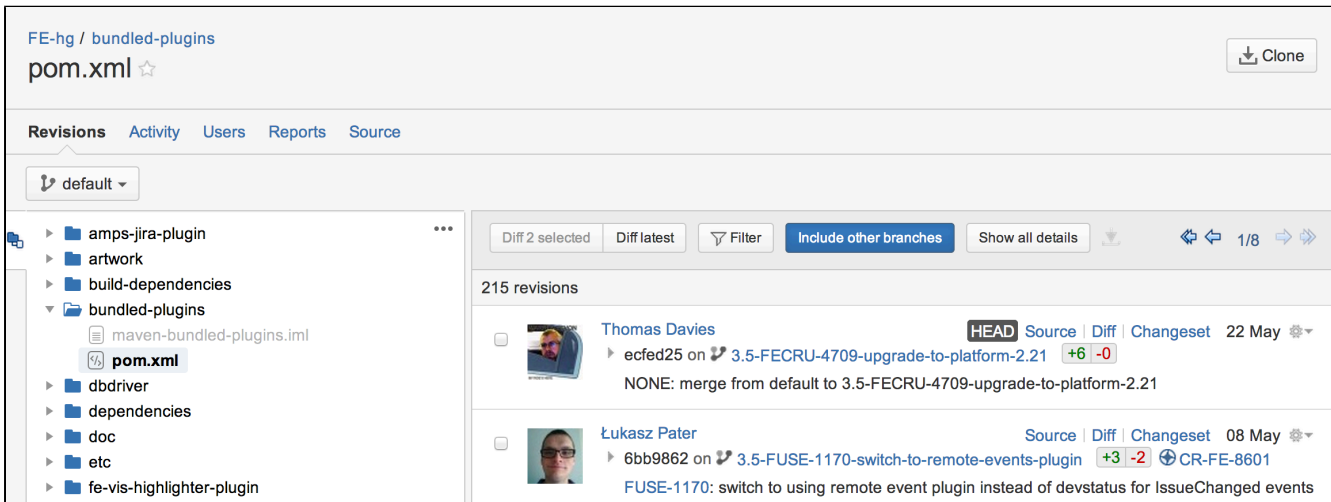
Download Hide deleted files

Name	Revision	Age	Author
README.md	7809905	5 weeks	Piotr Święcicki
src	4955816	5 weeks	Conor MacNeill
fecru-rest-plugin	dd732c3	5 weeks	Maciej Swinarski

View information about a file

Click a file name to see information about the file on these tabs:

Revisions	The history of revisions for the file. See Viewing a file history .
Activity	The commits and reviews activity related to the file. See Viewing the changelog .
Users	Commit histories for users who have committed changes to the file. See Viewing people's statistics .
Reports	Charts for the file activity. See FishEye charts .
Source	The raw file can be downloaded from this tab. See Viewing file content .



FE-hg / bundled-plugins

pom.xml ☆

Revisions Activity Users Reports Source

default

Diff 2 selected Diff latest Filter Include other branches Show all details

215 revisions

Thomas Davies HEAD Source Diff Changeset 22 May

ecfed25 on 3.5-FECRU-4709-upgrade-to-platform-2.21 +6 -0

NONE: merge from default to 3.5-FECRU-4709-upgrade-to-platform-2.21

Łukasz Pater Source Diff Changeset 08 May

6bb9862 on 3.5-FUSE-1170-switch-to-remote-events-plugin +3 -2 CR-FE-8601

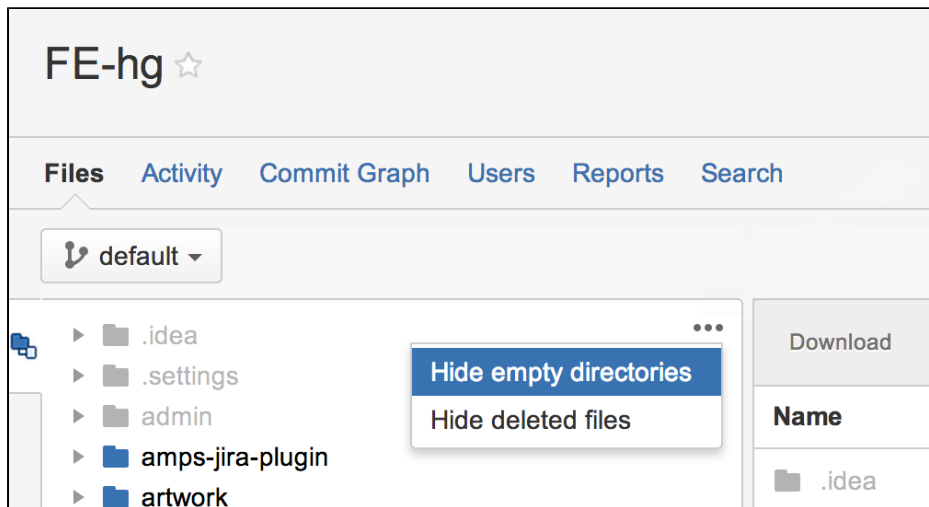
FUSE-1170: switch to using remote event plugin instead of devstatus for IssueChanged events

Hide empty directories and deleted files

FishEye tracks deleted files for your repository. Deleted files will be greyed out in the left-hand navigation tree. If all of the files in a directory are deleted, the directory will also be greyed out.

You can hide deleted files and empty directories in the left navigation tree:

1. Click **Repositories** in the header and browse to a repository.
2. In the left hand navigation panel, click the 'Actions' menu and choose either:
 - **Hide empty directories** – hide all empty (greyed out) directories and their contents (i.e. deleted files and empty sub-directories).
 - **Hide deleted files** – hide all deleted (i.e. greyed out) files. This does not affect directories.



i If you hide both empty directories and deleted files, you will only see files and directories that exist on the [Head](#) of that path. In repositories other than Subversion repositories, this could mean files/directories on any branch.

Watch a repository

You can "watch" a repository in FishEye/Crucible. Watching the repository allows you to receive email notifications when changes are made to the repository.

1. Navigate to the repository that you want to watch.
2. Choose **Tools > Watch**.

You can view all of your watches and configure the frequency of your watch emails in your user profile. See [Changing your user profile](#).

To remove the watch, choose **Tools > Unwatch**. You can also remove watches from within your user profile.

The option to add a watch will only be available if the administrator has [enabled watches](#) for the repository.

Viewing file content

You can [search](#) or [browse](#) your repositories in FishEye to view a specific file. Once you find the file, you can view the diffs between different revisions of the file, and see any annotations.

You can also download the source code for the file.

View the source code for a file

1. [Search](#), or [browse](#) through a repository, to find the file.
2. Click the file name to see the revision history for the file.
3. On the **Source** tab:

Displaying a diff	Click the arrow buttons to see the previous or next revisions of the file. Pick revision numbers (e.g. 'b3ebfaf') from the two lists to display the diff for those two revisions.
Changeset	View the changeset that the revision was a part of.
Raw	Download the raw source code for the file.

Reviews	Select Create Review to create a Crucible review from the file, or go to a review that relates to the file. <i>(Requires Crucible)</i>
Blame	Press to display authors and revision numbers next to each line of code in the file



Viewing a file history

You can view a specific file when [browsing a repository](#). This allows you to see information about the file, including the history of file revisions.

View the history of revisions for a file

- 1. Log into FishEye/Crucible.
- 2. [Search](#), or [browse](#) through a repository, to find the file.
- 3. Click the **Revisions** tab. The history of revisions for the file will be displayed. See the '[File Revisions](#)' screenshot below.

Diff 2 selected	Check boxes for two file revisions and then click to view the diff for those revisions.
Diff latest	View the diff of the two most recent file revisions.
Filter	View the file filter. Enter the desired fields to filter the file history results on.
Include other branches	Include revisions of the same file from other branches. ⓘ <i>A file can have many physical paths, all of which relate to the same filename in your project structure, or repository's logical structure. This applies to Subversion's branching and tagging directory structure in particular.</i>

Show all details

Toggle expand or collapse of all file revisions to show additional information including the revision ID, parents and the branch where it is head, denoted with this graphic:



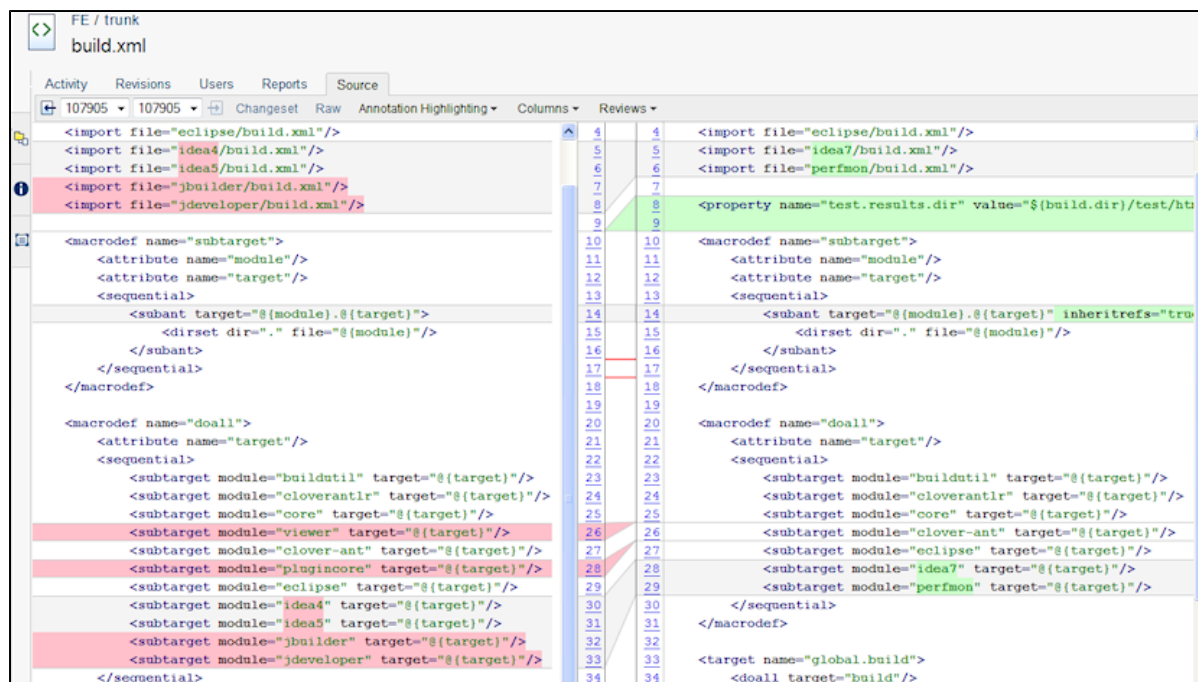
See the '[Overview of a File Revision](#)' diagram below for an explanation of the information provided about individual revisions.

Screenshot: File Revisions

The screenshot shows the FishEye interface for a file named `pom.xml` located in the `bundled-plugins` directory. The interface includes a sidebar with a file tree, a main area with revision details, and a top navigation bar. The revision details section shows a list of revisions, with the most recent one being a merge from default to 3.5-FECRU-4709-upgrade-to-platform-2.21, authored by Thomas Davies on 22 May. The revision details also include a diff view, a source link, and a changeset link.

Diagram: Overview of a file revision**Using side by side diff view**

FishEye's 'side-by-side diff' view allows you to see how a file's content has changed. Different versions of the file are displayed together to make line-by-line comparisons easy.



Show the side-by-side diff view

1. Open the source code view for the file. See [Viewing file content](#).
2. Use the elect a range of revisions to compare.
3. Choose **View > Side-by-side Diff**.

Note that:

- Green highlighting indicates where lines have been added; red shows where they've been removed.
- Grey highlights indicate that a line's internal content has changed.
- Each addition or deletion is linked to the adjacent pane by a colored triangle to show the location of that change in the comparison file.
- Line numbers in the margin are permanent links ("permalinks") that can be sent to colleagues. When they open those links, the view will automatically open in side by side diff mode.

Alternative ways to open side-by-side diffs

From the FishEye Dashboard

You can also open side by side diffs from the Dashboard screens, by clicking the 'Delta' triangle icon next to an item when it appears in the stream of events. This will open the file in the diff view. If you have currently selected side by side diff as the viewing mode, then the diff will automatically be displayed in that mode. If not, you can select side by side diff from the **View** menu.

From the Revisions History view

When in the revisions view, you can show a diff by checking the boxes next to two revisions, then clicking the **Diff** in the top control bar. If you have currently selected side by side diff as the viewing mode, then the diff will automatically be displayed in that mode. If not, you can select side by side diff from the **View** menu.

You can also launch into a diff of the latest revision and the second most recent by clicking **Latest Diff** in the top control bar.

Viewing the changelog

The changelog is a record of the commits and reviews for a repository, branch, directory or file. You can view the recent activity in the changelog when [browsing a repository/branch/directory](#).

On this page:

- [View changelog activity](#)
- [Filter commit activity in the changelog](#)
- [Watch the changelog activity](#)

View changelog activity

1. [Browse to the desired repository, branch, or directory.](#)
2. Use the branch/tag picker (under the repository or file name) to change context, if required.
3. Click the **Activity** tab.
4. Use the following functions of the **Activity** tab to see different aspects of the changelog activity:

All	Click to show commits, reviews (requires integration with Crucible) and JIRA issues (requires integration with JIRA) activity in the activity stream.
Commits	Click to show only commits in the activity stream.
Reviews	Show only review activity. (Requires integration with Crucible)
Filter commits	See Filter commit activity in the changelog (below) for more information.
Expand all	Show all modified files related to each changeset.
Scroll to changeset	Enter a changeset ID (e.g. 107856) to scroll to that changeset in the activity stream.

The screenshot displays the FishEye web interface for the 'FE-hg' repository. The top navigation bar includes 'Files', 'Activity' (selected), 'Commit Graph', 'Users', 'Reports', and 'Search'. Below the navigation bar, there's a 'default' branch selector. The left sidebar shows a file tree with folders like 'amps-jira-plugin', 'artwork', 'build-dependencies', etc. The main area shows the 'Activity' tab with sub-tabs 'All', 'Commits' (selected), 'Reviews', 'Filter commits', 'Expand all', and 'Scroll to changeset'. The activity stream lists commits:

- Wednesday 04 Jun:** Piotr Świącicki (7809905) changed 1 file (README.md) at 10:48 am.
- Tuesday 03 Jun:** Cezary Zawadka (c5d1005) changed 2 files at 1:19 pm.
- Tuesday 03 Jun:** Piotr Świącicki (97f9f03) changed 0 files (MERGE) at 12:50 pm.

 Each commit entry includes a user profile picture, commit ID, file changes, and a brief description.

Filter commit activity in the changelog

You can filter the commits that are displayed in the activity streams on the **All** and **Commits** tabs of the **Activity** tab. Note that you cannot use the commits filter to filter reviews.

1. Go to the **Activity** tab, as described above.
2. When viewing either the **All** or **Commits** tab, click **Filter commits**.
3. Enter filtering criteria:

Committer	Changesets checked in by the given committer/author.
Log Comment	Changesets where the commit comment matches the given text.

File Extension	Changesets that contain files with the specified file extension.
File Name	Changesets that contain a given file.
Start Date	Changesets created on or after that date. Must be of the form YYYY-MM-DDTHH:mm:ss, YYYY-MM-DD, YYYY-MM or YYYY (you can use '/' instead of '-').
End Date	Changesets created on or before that date. Must be of the form YYYY-MM-DDTHH:mm:ss, YYYY-MM-DD, YYYY-MM or YYYY (you can use '/' instead of '-').

4. Click **Apply**.

- Use **Clear** to clear the filter.
- Click **Filter commits** again to turn off the filter.

The screenshot shows the FishEye filter interface. At the top, there are tabs: 'All', 'Commits', 'Reviews', and 'Filter commits' (which is selected). To the right of the tabs are buttons for 'Expand all' and 'Scroll to changeset'. Below the tabs, there are input fields for 'Committer:', 'File Extension:', 'Start Date:', 'Log Comment:', 'File Name:', and 'End Date:'. An 'Apply' button is located at the bottom right of the filter section.

Watch the changelog activity

You can "watch" a changelog's activity stream in FishEye and Crucible. Watching the activity stream means that you receive emails when updates occur in the activity stream. You can even add filtering criteria for the watch, so for example you can restrict watches for commits by a particular user.

Note, the option to add a watch will only be available if the administrator has [enabled watches](#) for the repository.

1. Navigate to the activity stream that you want to watch, as described above.
2. If required, add filtering criteria, as described in the [Filter commit activity in the changelog](#) section above, and click **Apply**.
3. Choose **Tools > Watch**.

To remove the watch from the activity stream, choose **Tools > Unwatch**. The watch will be removed.

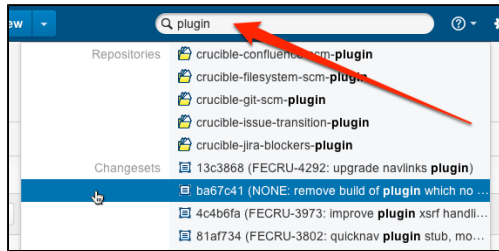
You can also view all of your watches, configure the frequency of your watch emails, and remove watches, in your user profile. See [Changing your user profile](#) for more information.

Searching FishEye

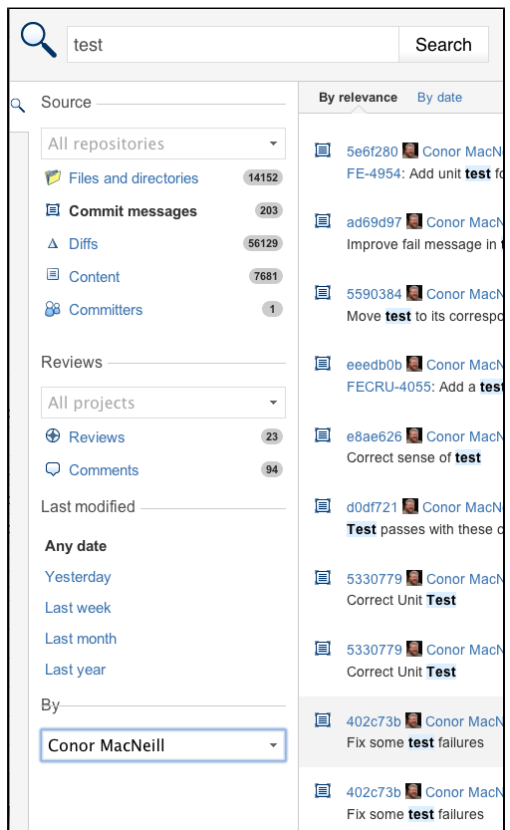
FishEye has a powerful search engine that allows you to find changesets, committers and files. There are two methods for searching in FishEye:

Quick Search

- Searches across all repositories connected to FishEye.
- Suggests "quick nav" results as you type.



- Has extra search tools, such as 'handles'.
- Press <Enter> to see filters:



Advanced Search

- Click the **Search** tab when viewing a repository.
- Searches a **single repository**.
- Access the **EyeQL** query language, if required.
- Enter search criteria for a range of attributes:

Search Criteria

☐ Search HEAD only

Comments:

Contents:

Added Text:

Removed Text:

Author(s):

File name:

(Ant-globs):

Branch:

Tag:

After:

Before:

(YYYY-MM-DD):

Order by:

Group by:

Display data:

☒ Path ☒ Revision ☒ Author ☒ Date

☒ Changeset ID ☐ Comment ☐ Total Lines

☐ Lines Added ☐ Lines Removed ☐ Tags

☐ Reviews

[Switch to EyeQL Search](#)

Use Quick Search

Simply enter your search criteria in the search box in the FishEye header to search across file, repository, committer and user names, as well as commit messages and reviews (if you are using **Crucible** with FishEye).

- Use search tools and filters to refine your results. See [Refining your Quick Search Criteria](#) and [Filtering Quick Search Results](#) below.
- Results are weighted by edit date; files edited within the last twelve months are given greater weighting.
- There is a 100-repository limit on searches, to prevent FishEye becoming unresponsive when there

are large numbers of repositories. FishEye will also limit the search to the specific repository that you are looking at, if you are already navigating within a specific repository

- If FishEye is integrated with JIRA you can see a summary of a JIRA issue in your search results by hovering over the issue key. See [JIRA Integration in FishEye](#).
- See also [Searching Crucible](#) in the Crucible documentation.

Refine your Quick Search criteria

The FishEye Quick Search has a number of powerful tools that you can use to refine your search criteria before executing the search.

Tool	Description	Example
CamelCase pattern matching	Enter a CamelCase pattern to find matching files and directories.	<code>BooQCTest</code> returns results like <code>BooleanQueryCoordTest</code> and <code>BooleanQueryClassTest</code> .
Path/File pattern matching	Enter a path/file pattern to find matching files and directories.	<code>common/final/Actions</code> returns results like <code>/src/common/eu/systems/central/specialprojects/final/Actions.java</code> .
Field handles	Use a field handle to restrict your search to a particular field: <ul style="list-style-type: none"> • <code>file</code> — file/directory names • <code>commit</code> — commit messages • <code>diff</code> — lines added/removed • <code>content</code> — file contents • <code>committer</code> — committer names 	<code>file:build.xml</code> returns files with names matching <code>build.xml</code> .
Search within a directory	Use AntGlobs to search a specific directory.	Search for <code>/src/**/*.gwt/*.xml</code> and FishEye will return all files with a .xml extension that are below both a src and a gwt directory. e.g. <code>src/java/com/atlassian/fecru/gwt/FecruCore.gwt.xml</code> but not <code>src/java/com/atlassian/fecru/ApplicationContext.xml</code>
Search for discrete phrases	Enter a phrase in quotation marks. Not case-sensitive.	<code>"update version in build"</code> returns matches for the exact string, i.e. return <code>"update build version"</code> or <code>"update version in file"</code> .

Filter Quick Search results

Once you have a set of search results on the Quick Search page, you can filter them to a subset of the original results. The filter controls are in the left panel of the Quick Search page in the 'Source' section.

Filter	Description
All repositories	By default, searches across all repositories. Choose a repo to restrict the search to just that one. For example, if you enter 'FE' then the search results page will refresh to display only files and directories in the 'FE' repository.
Files and directories	Filter your results to files and directories that have names that match the search criteria, see details below .
Commit messages	Filter your results to changeset comments that match the search criteria.
Diffs	Filter your results to diffs (lines added/removed) that match the search criteria.
Content	Filter your results to files that have content that match the search criteria, see details below .
Committers	Filter your results to committers that match the search criteria.
All projects	When using Crucible with FishEye, there is a 'Reviews' section. The All projects dropdown allows you to filter reviews returned in the search results. See Searching Crucible .
Reviews	Search in reviews
Comments	Search for review comments
Last modified	Filter by the date of the last change.
Author	Filter by author name.

Use Advanced Search

The Advanced Search can only be run against a specific repository, however you can specify more precise criteria against a number of fields for that repository.

1. Navigate to the repository that you want to search, as described in [Browsing through a repository](#).
2. Click the **Search** tab.
3. Enter your search criteria:
 - **Standard Search** — Enter criteria in the 'Search Criteria' panel. See [Specify criteria using the search interface](#) below for details.
 - **EyeQL Search** — Enter your "EyeQL" query. See [Specify criteria using EyeQL](#) section below for more details.

Use **Switch to EyeQL Search** and **Switch to Standard Search** at the bottom of the 'Search Criteria' panel to toggle between the two search methods.

Specify criteria using the search interface

The Advanced Search interface allows you to specify search criteria for multiple fields, order the results, group the results and choose the display fields in the results. You can link to the search results, as well as save the results to a CSV file.

Please note the following:

- **Contents** — Files must be non-binary, less than 5MB, and for SVN repositories, on trunk only. Only HEAD/tip revisions are searched. For older revisions, use the added/removed text search criteria.
- **File names** — [Antglobs](#) can be used to specify the criteria for this field.

Specify criteria using EyeQL

Advanced Search also allows you to run searches using FishEye's powerful EyeQL query language.

Click **Switch to EyeQL Search** at the bottom of the 'Search Criteria' panel.

See the [EyeQL reference guide](#) for more information. If you're unfamiliar with EyeQL, consider using the [Standard Search Interface](#) to build your initial query, then switch to EyeQL to modify that.

FishEye charts

When [browsing a repository](#), the **Reports** tab displays a chart of the lines of code (LoC) committed to the repository over time. The following options are available:


- [Charts](#)
- [Code Metrics](#)
- [Notes](#)

Charts

Click **Reports** and then **Charts** when browsing a repository to see charts of activity in the repository.

You can control the chart type that is displayed and various chart options. Click the cog icon on the left, select the required options, and click **Apply**. The available options include:

Setting	Explanation	Values	Default
Branch/Tag selector	Limits the chart to the selected branch/tag.	Any branch/tag from the current repository.	Displays the default/trunk.
Chart type	Changes the chart's presentation.	Area, line, pie or change* chart.	Area
Author	Limits the chart to show specific author(s).	Any author configured in the system.	All
Extension	Limits the chart to show specific file type(s).	Any file extension; e.g. '.java'.	All
Subdirectory	Limit the chart to a folder under the current branch. Files in the current directory are represented by an element labeled '.(this dir)'.	A single folder.	None (show all)
Start Date	Date of the earliest data to show.	Date in format YYYY-MM-DD.	None (show all)
End Date	Date of the latest data to show.	Date in format YYYY-MM-DD.	None (show all)

 *The 'Change' chart displays the change in lines of code, for a specific date range, expressed as a line graph. For example, if the lines of code at the start date is 100, the start point will be zero and the rest of the graph shifted by 100 lines.

Screenshot: FishEye custom chart settings

All **trunk**

Chart types:

☐ Area ☒ Line ☐ Pie ☐ Change

Chart options:

Break down by:

None ☒ Subdirectory ☐
Author ☐ User ☐
Extension ☐

Date Range:

Start Date:
:

Author(s):

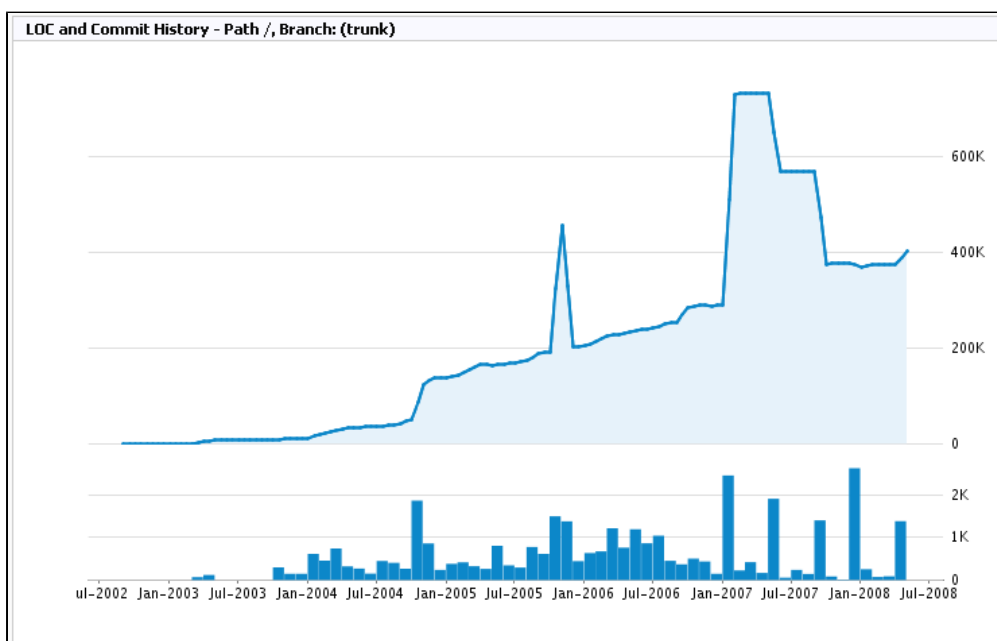
(any)
Adam Ahmed (aahmed)
Anna Buttfield (abuttfield)
Ben Speakmon (bspeakmon)
Brendan Humphreys (bhumphreys)

File Extension:

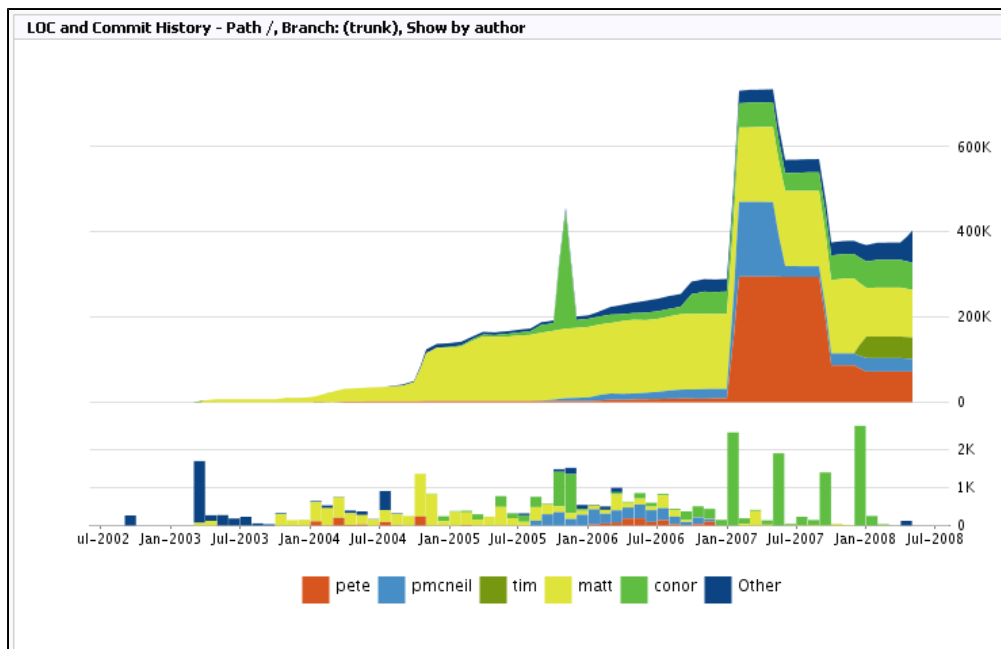
(any)
None
.MF
.TXT
.cfs

Apply

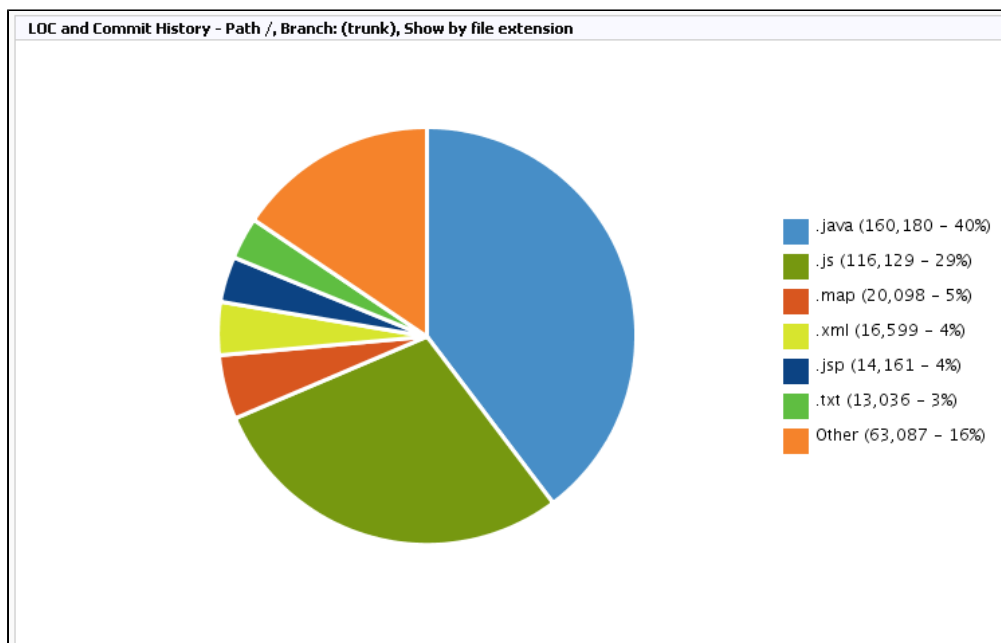
Screenshot: FishEye per-author LOC chart



Screenshot: FishEye per-author LOC chart showing multiple authors



Screenshot: FishEye LOC chart by file extension



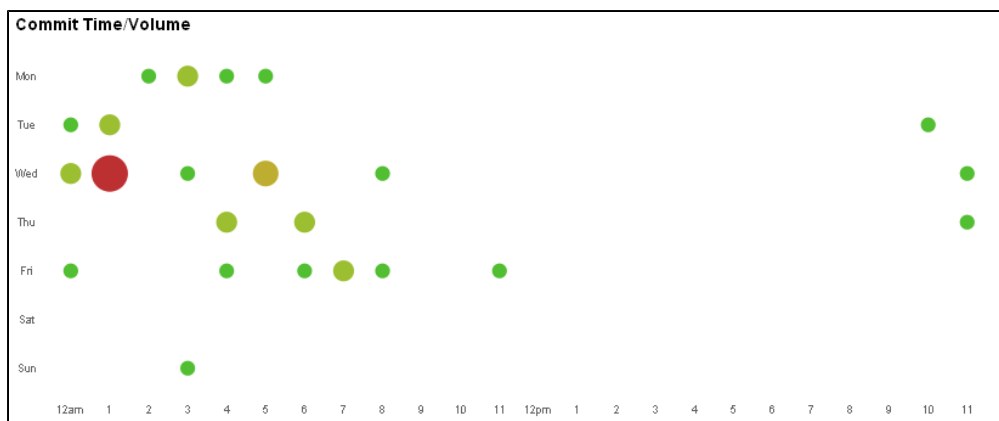
Per-Author Lines of Code Statistics

You can view per-author statistics for lines of code as a chart. This allows you to see how many lines of code were contributed to your project by each author, over time. You can easily view this information on the charts page. Note, if you are upgrading from a previous version of FishEye, you will need to re-index the repository in order to show the per-author information.

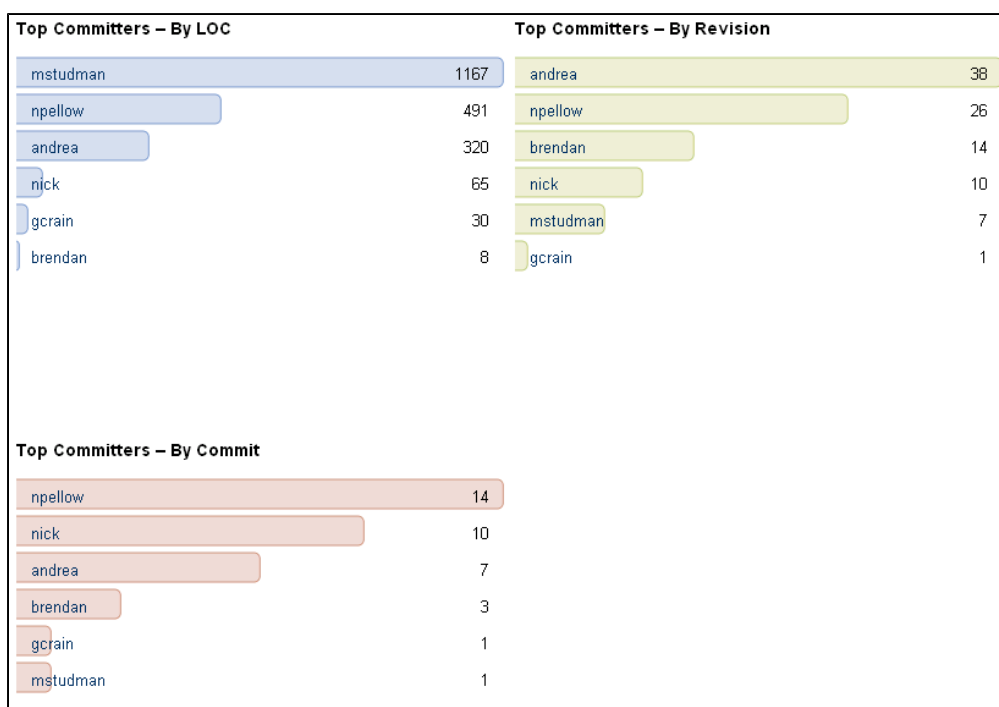
Code Metrics

A number of built-in reports are also provided:

Screenshot: Commit Time/Volume



Screenshot: Top Committers



Notes

Related Topics

[Browsing through a repository](#)

Using favourites in FishEye

FishEye allows you to add changesets, files, people and repositories as favorites. You can view your favorites, or see a stream of all activity relating to your favorites. We suggest that you select items that you are currently working on as favorites, to create a more relevant personalized view.

You can always view your favorites from the menu at the top of the screen, next to your username.

If you are using Crucible, you can also add [code reviews](#) to your favorites.

On this page:

- [Adding favorites](#)
- [Managing favorites](#)

Adding favorites

To add an item to your favorites, follow one of the options below:

People	Hover the mouse cursor over their avatar or username. In the context menu, click Follow .
Changesets	Open the changeset and click the grey star icon next to its name, near the top of the screen.
Files or folders	Open the file or folder and click the grey star icon that appears next to its name. The name appears in the breadcrumb links at the top of the screen.
Repository	Click the Source tab and then the grey star icon that appears next to the name of the desired repository.

Managing favorites

To change the display name for, or remove, a favorite:

1. Choose **Favorites** from your user menu at the top right.
2. Click the yellow star beside a favorite to edit the display name or remove the favorite.

Due to [FE-2348](#) you cannot currently rename favorite directories, users or committers

Changeset discussions

Please see the [Crucible documentation](#) for instructions on this feature.

Viewing the commit graph for a repository

The commit graph shows changesets in their respective branches, using configurable "swimlanes". This allows you to see key information such as branching and merging (and if you are using Git or Mercurial, you will be able to see anonymous branches as well).

The Highlight feature of the commit graph allows you to highlight different types of information in the swimlanes or changeset list:

- ancestors and descendants for a changeset
- commits with JIRA issues
- reviewed and unreviewed changesets.

For example, if you have the **JIRA issues** highlight active, clicking a changeset with a JIRA issue in the commit comment will show all other changesets with the same JIRA issue.

Before you begin:

- Subversion repositories currently do not show lines between branch swimlanes (i.e. merging). But in some cases, FishEye might pick up associations based on SVN branch points.
- Some features of the commit graph are only available if you are using [Crucible](#) with FishEye. For details, see the description below.
- Some features of the commit graph are only available if you are using [JIRA](#) with FishEye. For details, see the description below.

On this page:

- [Viewing the commit graph for a repository](#)
- [Highlighting the lineage of a changeset](#)
- [Highlighting JIRA issues](#)
- [Highlighting reviewed changesets](#)
- [Highlighting commits by an author](#)
- [Highlighting search results](#)
- [Viewing changesets across all branches](#)
- [Reordering swimlanes for Git repositories](#)


Related pages:

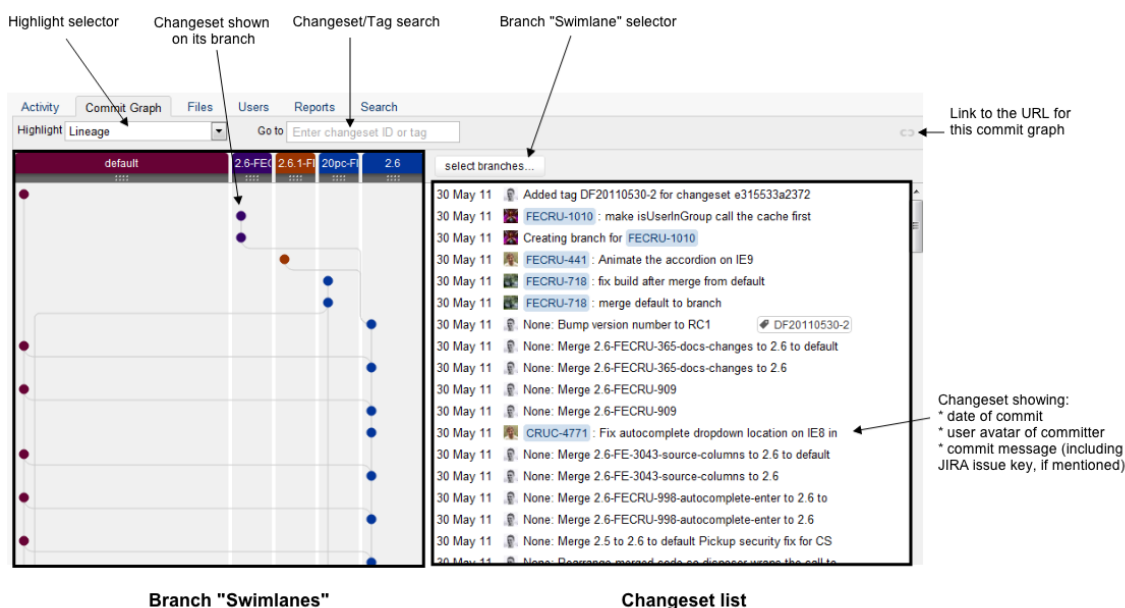
- [Subversion Changeset Parents and Branches](#)
- [What are Subversion root and tag branches?](#)
- [Perforce Changesets and Branches](#)
- [Using the FishEye screens](#)
- [Browsing through a repository](#)
- [JIRA Integration in FishEye](#)

Viewing the commit graph for a repository**To view the commit graph for a repository:**

1. Navigate to the desired repository, as described on [Browsing through a repository](#).
2. Click the **Commit Graph** tab.

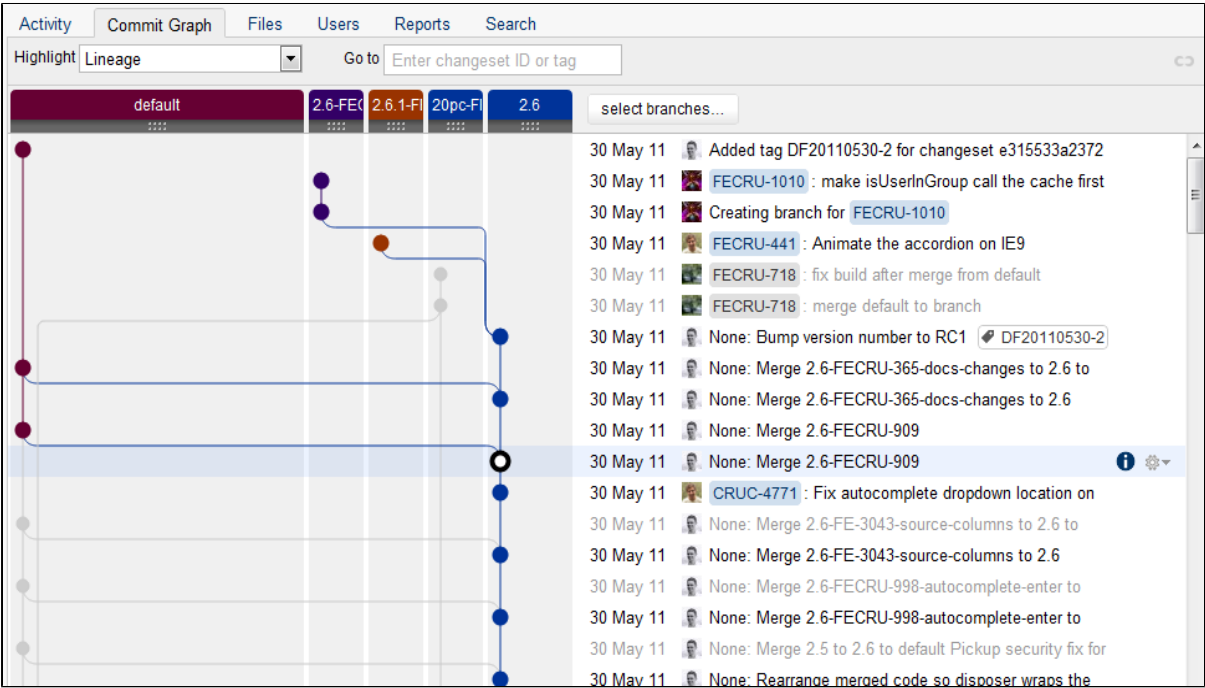
Selecting, or hovering on, a changeset (regardless of highlight) will display the following in the row for the changeset:

- an  icon. Click this icon to see details for the changeset.
- a cog icon with a menu that allows you to see the changeset ID, view the full changeset, view the changeset in the activity stream, or to create a review for the changeset.

**Highlighting the lineage of a changeset**

Choose **Highlight > Lineage** to show the ancestor and descendant changesets for a selected changeset.

Action	Behavior
Click on a changeset in the changeset list	Highlights where a changeset comes from and where it propagates to, i.e. its ancestors and descendants.
Hover over a changeset in a swimlane	Displays the changeset number and all the branches that the changeset is referenced in. This will include branches that you may not have swimlanes displayed for.

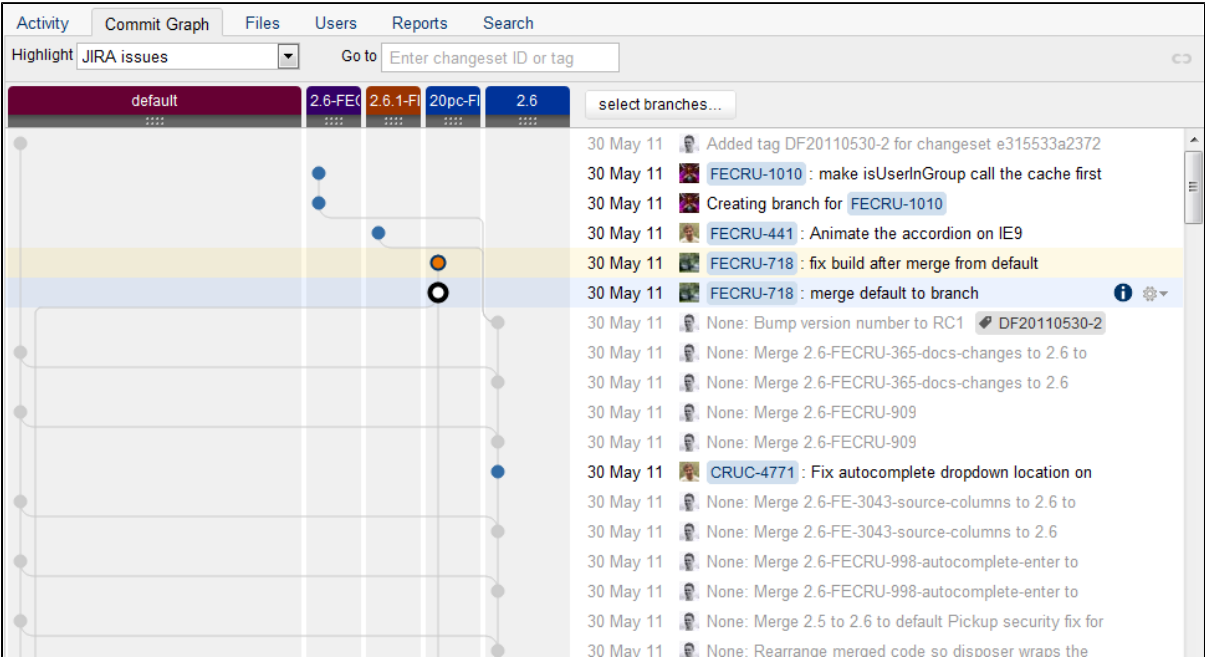


Highlighting JIRA issues

Choose **Highlight > JIRA issues** to highlight all the changesets that have a JIRA issue key in the commit message.

This highlight type is only available if you have [integrated FishEye with JIRA](#) and linked your repository to a JIRA project.

Action	Behavior
Click on a changeset in the changeset list	Highlights all other changesets that have the same JIRA issue key in the commit message.
Hover over a changeset in a swimlane	Displays all branches that the changeset is referenced in, and all referenced JIRA issues.



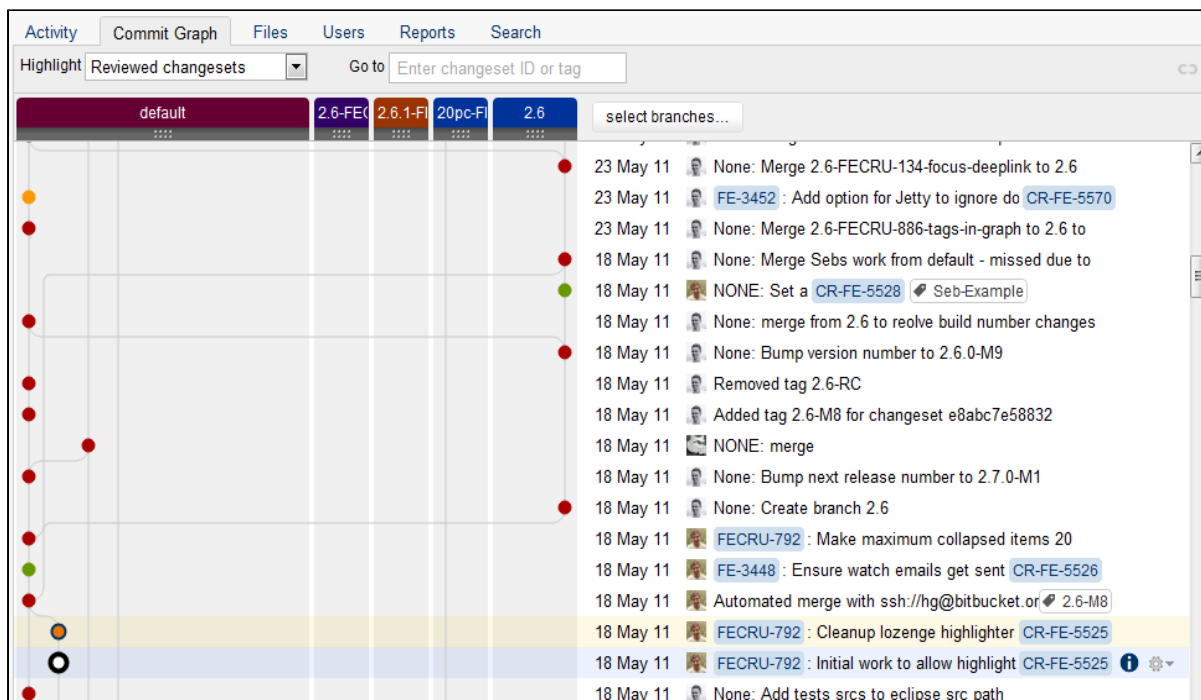
Highlighting reviewed changesets

Choose **Highlight > Reviewed changesets** to highlight the changesets that have been reviewed (i.e. included in a Crucible review):

- **Red**: unreviewed, i.e. the changeset is associated with a review in the 'Dead' or 'Rejected' state, or no review is associated.
- **Yellow**: under review, i.e. the changeset is associated with a review not in the 'Dead', 'Rejected' or 'Closed' state.
- **Green**: reviewed, i.e. the changeset is associated with a review in 'Closed' state.

This highlight type is only available if you are using FishEye with Crucible.

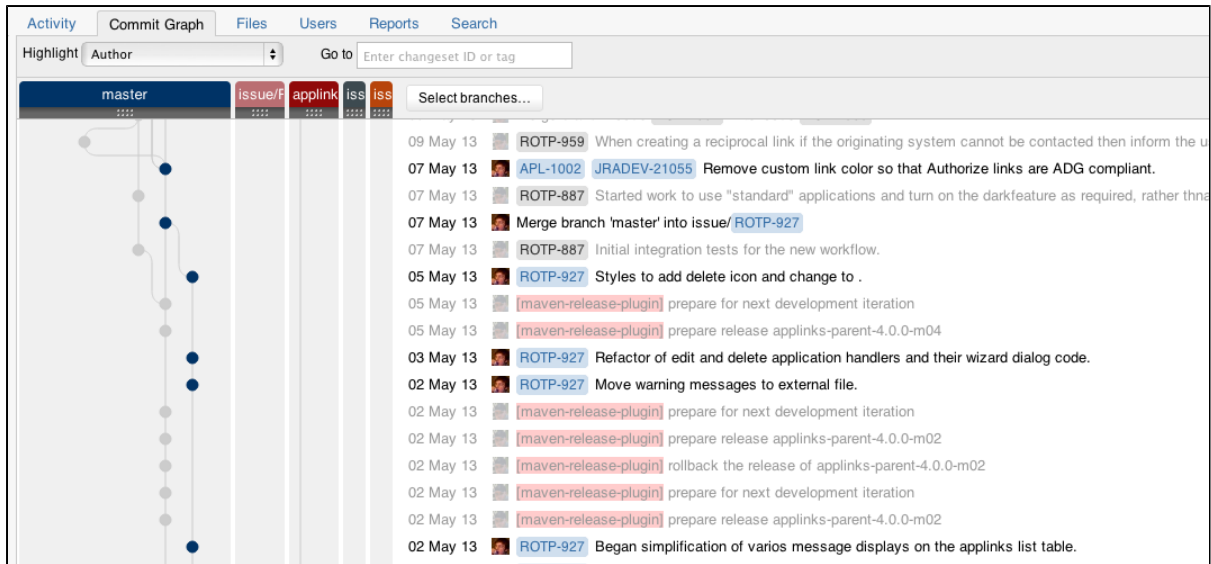
Action	Behavior
Click on a changeset in the changeset list	Highlights the changesets that are part of the same review as the selected changeset.
Hover over a changeset in a swimlane	Displays all branches that the changeset is referenced in, and the Crucible review key.



Highlighting commits by an author

Choose **Highlight > Author** to highlight all the changesets submitted by a particular author.

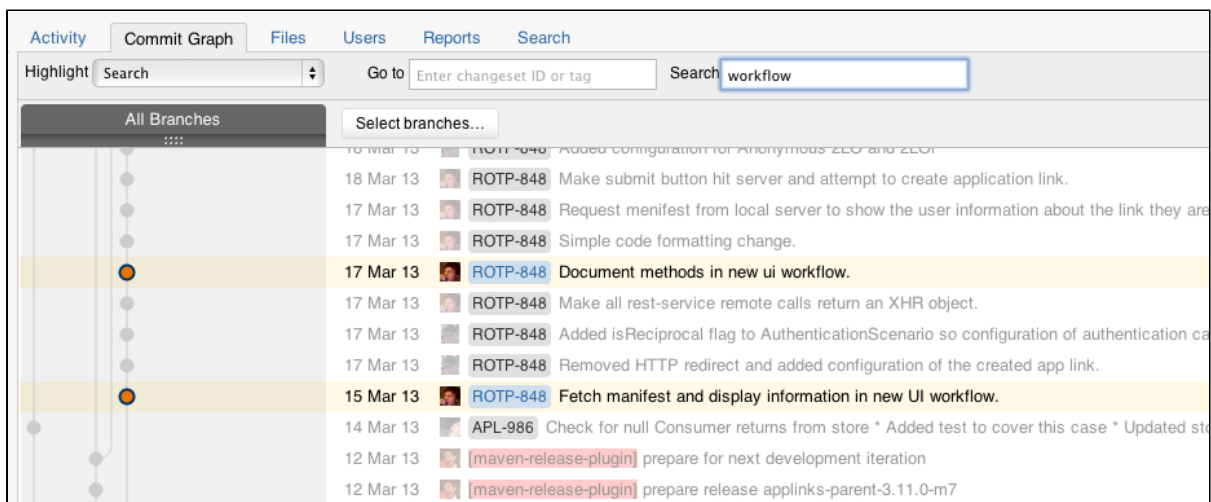
Action	Behavior
Click on a changeset in the changeset list	Highlights the changesets that were submitted by the same author.
Hover over a changeset in a swimlane	Displays the changeset number and all the branches that the changeset is referenced in.



Highlighting search results

Choose **Highlight > Search** to highlight all the changesets where the commit message contains the search term.

Action	Behavior
Click on a changeset in the changeset list	Highlights the changesets that match the search term.
Hover over a changeset in a swimlane	Displays the changeset number and all the branches that the changeset is referenced in.

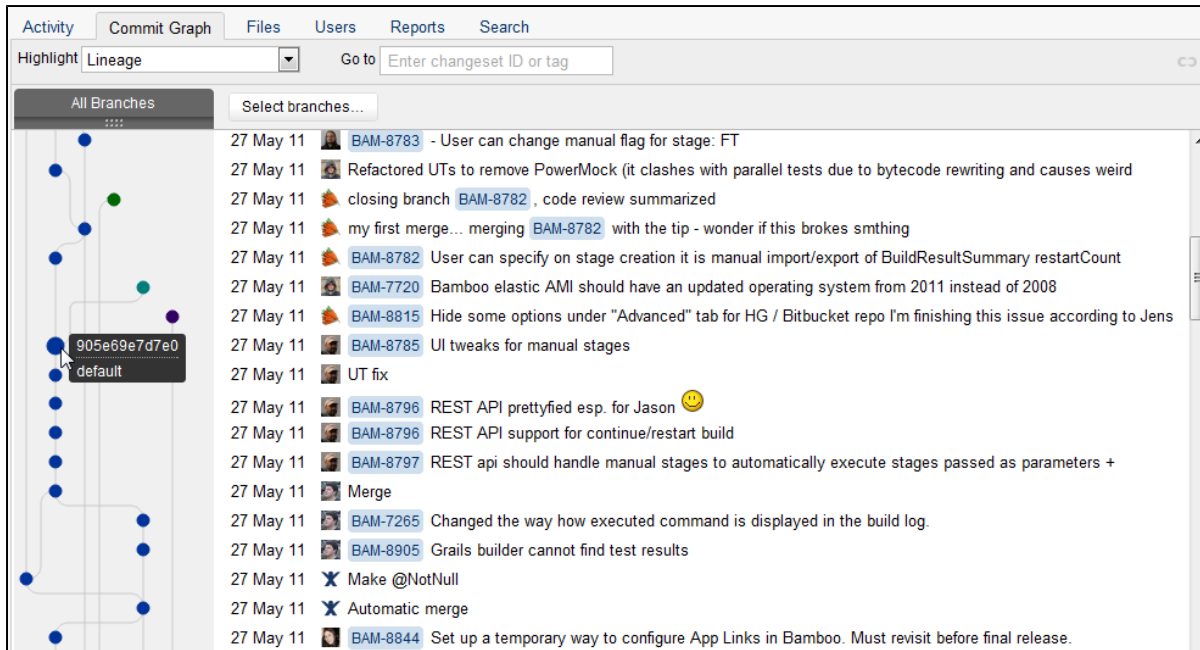


Viewing changesets across all branches

The 'All Branches' mode allows you to view commit activity across all branches of a repository. In this mode, the swimlane headers are not displayed. However, you can hover over any changeset to display information about the changeset, as described in the 'Highlighting Information in the Commit Graph' section above.

To see all the repository's branches in the commit graph:

1. Click **Select branches...** when viewing the commit graph.
2. In the 'Select Branches' dialog, click **Switch to all branches mode**.



Reordering swimlanes for Git repositories

Reordering swimlanes is useful if you just want to show branches in a certain order. However, ordering swimlanes is vital for Git repositories, as it is the only way to determine which branch a commit is from.

When you view the commit graph for a Git repository, FishEye works from the leftmost swimlane to the right and, for each swimlane, checks if the commit is in that branch:

- If the commit is in the branch, a dot is shown representing the commit.
- If the commit is not in the branch, the dot for the commit is moved to the next column on the right.

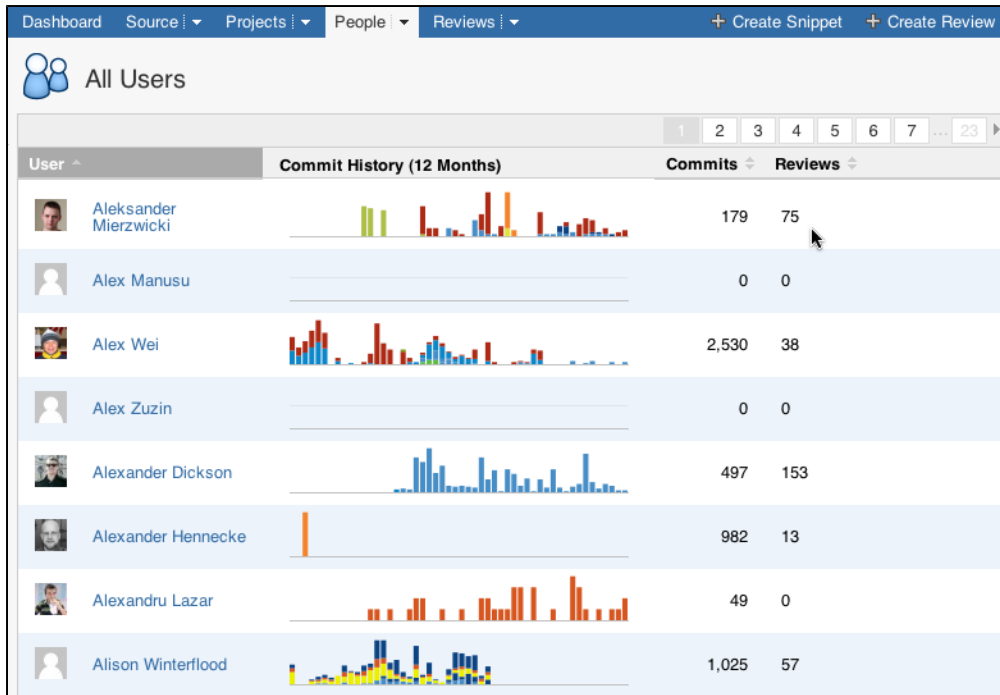
For example, if the 'master' swimlane is to the left of another swimlane, e.g. 'fisheye-2.6' branch, there will be no changesets shown in the 'fisheye-2.6' swimlane, as all the commits will be picked up in the 'master' swimlane. However, if you move the 'fisheye-2.6' swimlane to the left of the 'master' swimlane, it will pick up all of the FishEye 2.6 commits.

For more information, read this Knowledge Base article: [Ordering of Branches Important When Visualizing Git Changeset](#)

Viewing people's statistics

To see charts and activity of everyone who commits code to your FishEye repositories, click the **People** tab at the top of the screen.

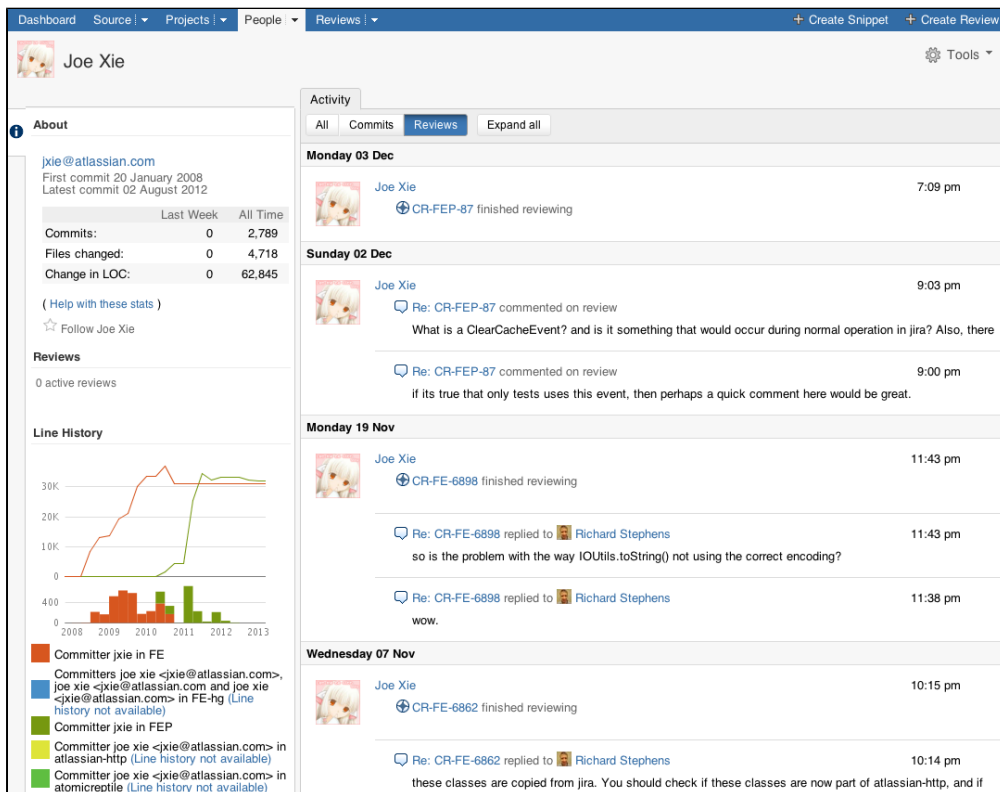
Screenshot: List of all committers in FishEye



The All Users screen shows all those with accounts on the system. You can see their commit history (expressed as a bar graph) and their total number of commits.

Click on a person's name to see detailed information about their additions to the repository, and issue updates. If you are using FishEye with Crucible and have JIRA integration set up, you can see their review activity.

Screenshot: Statistics on a Person in FishEye



Some users may not appear to have the correct number of Files Changed, despite regularly committing. In this situation, if they have committed to a directory which is not covered by the regexes in your

symbolic definition (i.e. they have committed to a directory that is neither trunk, branches or tags) then that directory will be counted as part of trunk. Also note that creating tags and branches themselves does not count toward the totals in FishEye.

Avatars

By default, each user has a unique avatar that is randomly formed from the text in their email address. You can add your own avatar by uploading an image to an external service such as Gravatar, which Crucible supports. See [changing your user profile](#).

If you are using [Crucible](#), statistics for each person's code reviews are also available.

Using Smart Commits

Smart Commits allow repository committers to perform actions such as transitioning JIRA Software issues or creating Crucible code reviews by embedding specific commands into their commit messages.

You can:

- comment on issues
- record time tracking information against issues
- transition issues to any status defined in the JIRA Software project's workflow.

There are other actions available if you use Crucible. See [below](#) for more information.

Each Smart Commit message must not span more than one line (i.e. you cannot use a carriage return in the command), but you can add multiple commands to the same line. See [this example](#) below.

On this page:

- [Smart Commit commands](#)
- [Smart Commit commands for Crucible reviews](#)
- [Advanced examples](#)
- [Get Smart Commits working](#)

Related pages:

- [Enabling Smart Commits](#)
- [Transitioning JIRA issues](#)
- [Write your own Smart Commit](#)

Smart Commit commands

The basic command line syntax for a Smart Commit message is:

```
<ignored text> <ISSUE_KEY> <ignored text> #<COMMAND> <optional  
COMMAND_ARGUMENTS>
```

Any text between the issue key and the Smart Commit command is ignored.

There are three Smart Commit commands you can use in your commit messages:

- [comment](#)
- [time](#)
- [transition](#)

Comment

Description	Adds a comment to a JIRA Software issue.
Syntax	<ignored text> ISSUE_KEY <ignored text> #comment <comment_string>
Example	JRA-34 #comment corrected indent issue
Notes	<ul style="list-style-type: none"> The committer's email address must match the email address of a single JIRA Software user with permission to comment on issues in that particular project.

Time

Description	Records time tracking information against an issue.
Syntax	<ignored text> ISSUE_KEY <ignored text> #time <value>w <value>d <value>h <value>m <comment_string>
Example	JRA-34 #time 1w 2d 4h 30m Total work logged
Notes	<p>This example records 1 week, 2 days, 4 hours and 30 minutes against the issue, and adds the comment 'Total work logged' in the Work Log tab of the issue.</p> <ul style="list-style-type: none"> Each value for w, d, h and m can be a decimal number. The committer's email address must match the email address of a single JIRA Software user with permission to log work on an issue. Your system administrator must have enabled time tracking on your JIRA Software instance.

Workflow transitions

Description	Transitions a JIRA Software issue to a particular workflow state.
Syntax	<ignored text> ISSUE_KEY <ignored text> #<transition_name> <comment_string>
Example	JRA-090 #close Fixed this today

Notes	<p>This example executes the close issue workflow transition for the issue and adds the comment 'Fixed this today' to the issue. Note that the comment is added automatically without needing to use the <code>#comment</code> command.</p> <p>You can see the custom commands available for use with Smart Commits by visiting the JIRA Software issue and seeing its available workflow transitions:</p> <ol style="list-style-type: none"> 1. Open an issue in the project. 2. Click View Workflow (near the issue's Status). <p>The Smart Commit only considers the part of a transition name before the first space. So, for a transition name such as <code>finish work</code>, then specifying <code>#finish</code> is sufficient. You must use hyphens to replace spaces when ambiguity can arise over transition names, for example: <code>#finish-work</code>.</p> <p>If a workflow has two valid transitions, such as:</p> <ul style="list-style-type: none"> • <code>Start Progress</code> • <code>Start Review</code> <p>A Smart Commit with the action <code>#start</code> is ambiguous because it could mean either of the two transitions. To specify one of these two transitions, fully qualify the transition you want by using either <code>#start-review</code> or <code>#start-progress</code>.</p> <ul style="list-style-type: none"> • When you resolve an issue with the <code>#resolve</code> command, you cannot set the Resolution field with Smart Commits. • If you want to add a comment during the transition, the transition must have a screen associated with it. • The committer's email address must match the email address of a single JIRA Software user with the appropriate project permissions to transition issues.
--------------	---

Smart Commit commands for Crucible reviews

When creating a new review using a Smart Commit the [default project objectives](#) are added to the review, and since Fisheye/Crucible 2.10.2, the commit message is also copied to the review objectives.

Note that you cannot add arbitrary objectives to the review using a Smart Commit.

Create a review

Description	Create a review in Crucible.
Syntax	<code><commit message> +review <project key></code>
Example	<code>Fix a bug +review CR-TEST</code>
Notes	The <code>+review</code> command creates a new review in the project CR-TEST with the content of the changeset. The review will be in a draft state unless the project has default reviewers or reviewers are explicitly mentioned. If you only have one project in Crucible, or a repository is a project's default repository, it is not necessary to mention the project key.

Add reviewers

Description	Add reviewers when creating a new review in Crucible.
Syntax	<code><commit message> +review <project key> <reviewer_1> <reviewer_2>... <reviewer_n></code>
Example	<code>Fix a bug +review CR-TEST @jcage @skhan</code>

Notes	<p>The command will create a new review in <i>PROJ</i> and add the users <i>jcage</i> and <i>skhan</i> to the review. The review will be automatically started if reviewers are specified.</p> <p>Note that you cannot add reviewers to existing reviews using Smart Commits.</p>
--------------	---

Update a review

Description	Iteratively add new code changes to a review
Syntax	<code><commit message> +review <review key></code>
Example	<code>Implement rework on past work +review CR-TEST-123</code>
Notes	<p>Often, reviews require rework and changes in response to comments left by the team. When committing these changes, adding the review key will iteratively add these new changes to the review.</p> <p>Note that:</p> <ul style="list-style-type: none"> Each commit command in the Smart Commit must not span more than one line (i.e. you cannot use carriage returns). However, you can use multiple commands in the same commit message, and these can be on the same line. Creating a review in Crucible using a Smart Commit requires that the author of the changeset has already been mapped to a Crucible username. See 'Author mapping' on Changing your user profile.

Advanced examples

Multiple commands on a single issue

Syntax	<code><ISSUE_KEY> #<COMMAND_1> <optional COMMAND_1_ARGUMENTS> #<COMMAND_2> <optional COMMAND_2_ARGUMENTS> ... #<COMMAND_n> <optional COMMAND_n_ARGUMENTS></code>
Commit message	<code>JRA-123 #time 2d 5h #comment Task completed ahead of schedule #resolve</code>
Result	Logs 2 days and 5 hours of work against issue JRA-123, adds the comment 'Task completed ahead of schedule', and resolves the issue.

Multiple commands over multiple lines on a single issue

Syntax	<code><ISSUE_KEY> #<COMMAND_1> <optional COMMAND_1_ARGUMENTS> #<COMMAND_2> <optional COMMAND_2_ARGUMENTS> ... #<COMMAND_n> <optional COMMAND_n_ARGUMENTS></code>
Commit message	<code>JRA-123 #comment Imagine that this is a really, and I mean really, long comment #time 2d 5h</code>

Result	<p>Adds the comment 'Imagine that this is a really, and I', but drops the rest of the comment. The work time of 2 days and 5 hours is <i>not</i> logged against the issue because there is no issue key for the <code>#time</code> command in the second line. That is, each line in the commit message must conform to the Smart Commit syntax.</p> <p>This example would work as expected if set out as:</p> <pre>JRA-123 #comment Imagine that this is a really, and I mean really, long comment JRA-123 #time 2d 5h</pre>
---------------	--

A single command on multiple issues

Syntax	<code><ISSUE_KEY1> <ISSUE_KEY2> <ISSUE_KEY3> #<COMMAND> <optional COMMAND_ARGUMENTS> etc</code>
Commit message	<code>JRA-123 JRA-234 JRA-345 #resolve</code>
Result	<p>Resolves issues JRA-123, JRA-234 and JRA-345.</p> <p>Multiple issue keys must be separated by whitespace or commas.</p>

Multiple commands on multiple issues

Syntax	<code><ISSUE_KEY1> <ISSUE_KEY2> ... <ISSUE_KEYn> #<COMMAND_1> <optional COMMAND_1_ARGUMENTS> #<COMMAND_2> <optional COMMAND_2_ARGUMENTS> ... #<COMMAND_n> <optional COMMAND_n_ARGUMENTS></code>
Commit message	<code>JRA-123 JRA-234 JRA-345 #resolve #time 2d 5h #comment Task completed ahead of schedule</code>
Result	<p>Logs 2 days and 5 hours of work against issues JRA-123, JRA-234 and JRA-345, adds the comment 'Task completed ahead of schedule' to all three issues, and resolves all three issues.</p> <p>Multiple issue keys must be separated by whitespace or commas.</p>

Get Smart Commits working

Your FishEye administrator must have:

- Enabled Smart Commits in FishEye. See [Enabling Smart Commits](#).
- Configured an application link between FishEye/Crucible and JIRA Software. See [Linking to another application](#).

Note that:

- Smart Commits only support the default JIRA Software issue key format (that is, two or more uppercase letters, followed by a hyphen and the issue number, for example BAM-123).
- Smart Commits don't provide for field-level updates in JIRA Software issues.
- When using Smart Commits you can use linkers that create a hyperlink to the JIRA Software issue. See [Linkers](#) for more information.
- If there are any errors during the processing of Smart Commits, they will be logged to FishEye's error console, as well as emailed to the actioning users. Speak to your FishEye administrator about [Configuring SMTP](#).

If the application link is configured as OAuth

If the application link to JIRA Software is configured to use OAuth, the committing user must authenticate with JIRA Software before any Smart Commit will work with JIRA Software.

▼ [Click to see how to authenticate with JIRA Software...](#)

1. Create a test review in FishEye.
2. Log in as the committing user.
3. Open the review and click **Edit Details**.
4. Enter a JIRA Software issue key for 'Linked Issue' and click **Link**. You'll be prompted to authenticate.
5. Do this for every committing user (there's no need to create a new review, just link the review to any JIRA Software issue).

Changing your user profile

You can change FishEye (and Crucible) settings such as password, notifications, profile image and display settings.

To change your FishEye settings:

1. Log into FishEye.
2. Choose **Profile settings** from the User Menu (with your avatar) at the top of the screen.
3. Update your user settings as required. Each tab is described in more detail below.
4. Click **Close**.

On this page:

- [Display settings](#)
- [Profile and email](#)
- [Change password](#)
- [OAuth authentication](#)
- [Author mapping](#)
- [Watches](#)
- [Reviews](#)

Display settings

Display Settings	File history view mode	Default is Logical . In Subversion repositories, FishEye is able to show all operations on a single logical file spread across a number of physical paths - i.e. operations in different branches. When this is set to Logical , FishEye will show all the operations across all branches . In Physical mode, only the operations related to the physical path whose history is being viewed are shown.
	Timezone	Default is the timezone of the FishEye server.
Changelog	Changesets per page	The default is 30 per page.
	Always expand changesets in stream	Default is Yes .
Diff view	Diff mode	Default is Unified .
	Line wrapping	Default is None i.e. long lines will never word-wrap. Soft is when long lines will word-wrap.
	Context lines	Default is 3. The number of lines to show (for context), if the diff contains more than three lines of code.


Source view	Tab width	Default is 8. Can be changed to a number between 1 and 10.
--------------------	------------------	--

Profile and email

Email settings	Display Name	Name displayed for the user currently logged in.
	Email Address	The address to which all email notifications will be sent.
	Email Format	Default is Text . Can be sent as HTML .
Email watches	Send Watch Emails	The frequency at which emails will be sent for watch notifications. Immediately is the default value. Daily sends a summary of changes.
Profile Picture	Choose picture	<p>Upload an avatar image of your choice. This image will be displayed next to your username throughout FishEye/Crucible.</p> <p>Accepted formats are JPG, GIF and PNG. Image file size limit is 2Mb. Images will be automatically cropped on upload.</p> <p>This is disabled if avatars are served from an external server – see Configuring avatar settings.</p>

Change password

Change your password from this tab, if required. Please note that passwords are case-sensitive.

 This tab is not displayed if your FishEye instance is connected to an external LDAP authentication source, such as LDAP. You will need to contact your administrator for assistance.

OAuth authentication

Configure your OAuth settings on this page. You can choose to allow gadgets/applications to access FishEye data using your account.

Read more about [Linking to another application](#).

Author mapping

The **Author Mapping** tab allows you to make an association between you (as a logged-in user) and a committer, for each repository.

This is only necessary if the name or email of the user within FishEye is different from the committer name or email within the repository. By default, FishEye will automatically match users to committers where it can.

Note that FishEye/Crucible administrators can control whether author mapping is available. See [Configuring user managed mappings](#).

Watches

By adding a 'watch', you can ask to receive emails about changes made to the repository. Any watches that you

have set up in FishEye/Crucible will be displayed on this tab. You can watch the dashboard activity stream, [change logs](#) and [repositories](#). Watching an activity stream/repository allows you to receive emails when updates occur. Note, the option to add a watch may only be available if the administrator has [enabled watches](#) for the repository.

You can delete any of your watches by clicking **Delete** next to the watch.

Reviews

This functionality is used by [Crucible](#).

If the SMTP server is set up, then you will receive emails when different actions occur within Crucible.

You can change the options described below, to specify the stages at which emails will be sent.

Auto-mark files as 'read'	Default is Yes .	
Review Notifications Events	State change	Default is Immediate . A Crucible review moves through different states e.g: 'Draft', 'Under Review'. An email is sent when the state changes.
	Comment added	Default is Immediate . An email is sent when a comment is added to a review.
	Comment reply added	Default is Immediate . An email is sent (to the Moderator only) when any reviewer has completed their review .
	Participant finished	Default is Immediate . An email is sent when a reviewer is added or removed from a review, after it has gone into the 'Under Review' state .
	General message	Default is Immediate . An email is sent when a reviewer is added or removed from a review, after it has gone into the 'Under Review' state .
	File revision added	Default is Immediate .
	Uncomplete review if defect is raised:	Default is Yes . This allows reviews to be resurrected automatically to deal with new code or defects.
	Uncomplete review if revision is added:	Default is Yes . This allows reviews to be resurrected automatically to deal with new code or defects.
	My actions	Default is No . If set to Yes , an email is sent every time you perform an action on a review.

Batch Notifications will be sent out by Crucible every 30 minutes. All notifications will be rolled up into a single digest e-mail.

Profile settings

Display Settings

File history view mode: ☐ Physical ☒ Logical

Timezone:

Changelog

Changesets per page:

Always expand changesets in streams: ☐ Yes ☒ No

Diff view

Diff mode: ☒ Unified ☐ Side-by-side

Line wrapping: ☒ None ☐ Soft

Context lines:

Source view

Tab width:

IDE Connector

Enable IDE icons: ☒ Yes ☐ No

Port number:

JIRA Servers

Ignored Servers: None

[Close](#)

Re-setting your password

If you need to reset your password, FishEye has a mechanism to generate a new password and send it to the email address in your profile.

To reset your password:

1. On the log in screen, click **Unable to access your account?**
2. Enter your username or email address and complete the [Captcha](#) step. An email will be sent to the email address specified in your profile.
3. Click the link in the email.

On the resulting web page, you will see the message 'A new password has been sent to your account.' An email will arrive in your inbox, containing your new password.

i If you receive a password-reset email that you did not request, simply disregard it to continue using your current password.

Pattern matching guide

FishEye supports a powerful type of regular expression for matching files and directories (same as the pattern matching in Apache Ant).

These expressions use the following wild cards:

?	Matches one character (any character except path separators)
*	Matches zero or more characters (not including path separators)
**	Matches zero or more <i>path segments</i> .

Remember that Ant globs match *paths*, not just simple filenames.

- If the pattern does not start with a path separator i.e. / or \, then the pattern is considered to start with / ** /.
- If the pattern ends with / then ** is automatically appended.

- A pattern can contain any number of wild cards.

Also see the [Ant documentation](#).

Examples

<code>*.txt</code>	Matches <code>/foo.txt</code> and <code>/bar/foo.txt</code> but not <code>/foo.txtty</code> or <code>/bar/foo.txtty/</code>
<code>/*.txt</code>	Matches <code>/foo.txt</code> but not <code>/bar/foo.txt</code>
<code>dir1/file.txt</code>	Matches <code>/dir1/file.txt</code> , <code>/dir3/dir1/file.txt</code> and <code>/dir3/dir2/dir1/file.txt</code>
<code>**/dir1/file.txt</code>	Same as above.
<code>/**/dir1/file.txt</code>	Same as above.
<code>/dir3/**/dir1/file.txt</code>	Matches <code>/dir3/dir1/file.txt</code> and <code>/dir3/dir2/dir1/file.txt</code> but not <code>/dir3/file.txt</code> , <code>/dir1/file.txt</code>
<code>/dir1/**</code>	Matches all files under <code>/dir1/</code>
<code>/dir1*</code>	Matches all files as <code>/dir11</code> , <code>/dir12</code> , <code>/dir12345</code> and so on.
<code>/dir??</code>	Matches all files as <code>/dir11</code> , <code>/dir22</code> and so on, replacing just 2 characters.

Date expressions reference guide

FishEye supports a wide variety of date expressions. A date has the two possible general forms:

- `DATE[+-]TIMEZONE[+-]DURATION`, or
- `DATECONSTANT[+-]DURATION`.

The `TIMEZONE` and `DURATION` parts are both optional.

`TIMEZONE` can be an offset from GMT `HHMM` or `HH:MM`, or simply the letter `Z` to denote GMT. If no timezone is given, the FishEye server's configured timezone is used.

`DATE` can be either of the following:

<code>YYYY-MM-DDThh:mm:ss</code>	Specifies a time and date (separated by a <code>T</code>). The seconds part may contain a fractional component. A <code>/</code> can be used instead of <code>-</code> as a separator.
<code>YYYY-MM-DD</code>	Specifies 00:00:00 on the given date. A <code>/</code> can be used instead of <code>-</code> as a separator.

`DATECONSTANT` can be any of:

<code>now</code>	This very instant (at the time the expression was evaluated).
<code>today</code> <code>todaygmt</code>	The instant at 00:00:00 today. (server-time* or GMT)
<code>thisweek</code> <code>thisweekgmt</code>	The instant at 00:00:00 on the first day of this week. Sunday is considered the first day. (server-time* or GMT)
<code>thismonth</code> <code>thismonthgmt</code>	The instant at 00:00:00 on the first day of this month. (server-time* or GMT)

<code>thisyear</code> <code>thisyeargmt</code>	The instant at 00:00:00 on the first day of this year. (server-time* or GMT)
---	--

* The timezone used for server-time is part of the FishEye configuration

The syntax for DURATION is similar to the XML Schema duration type. It has the general form PnYnMnDTnHnMnS. See Section 3.2.6 of the XML Schema Datatypes document for more details.

Examples

<code>2005-01-02</code>	The start of the day on January 1, 2005 (server's timezone)
<code>2005-01-02-0500</code>	The start of the day on January 1, 2005 at GMT offset -0500 (New York)
<code>2005-01-02T12:00:00Z</code>	Midday, January 1, 2005 GMT
<code>today-P1D</code>	Yesterday (start of day)
<code>today+P1D</code>	Start of tomorrow
<code>thismonth-P1M</code>	Start of last month
<code>thisyear+P1Y</code>	Start of next year
<code>now-PT1H</code>	One hour ago
<code>now+PT1H2M3S</code>	One hour, two minutes and three seconds from now

EyeQL reference guide

FishEye contains a powerful query language called **EyeQL**. EyeQL is an intuitive SQL-like language that allows you to write your own specific queries. [See examples](#).

EyeQL allows you to perform complex searches either within the [Advanced Search](#) or incorporated in scripts when programming the FishEye API.

<i>query:</i>	select revisions (from (dir directory) word)? (where clauses)? (order by date (asc desc)?)? Notes: <i>asc</i> produces 'ascending order'. <i>desc</i> produces 'descending order'. (group by (file dir directory csid changeset))? (return return-clauses)? (limit limit-args)?
<i>clauses:</i>	<i>clause ((or and ,) clause)*</i> Notes: <i>and</i> binds more tightly than <i>or</i> . ',' (comma) means 'and'.
<i>clause:</i>	(clauses) not clause path (not)? like word Notes: <i>word</i> is an Antglob.

path = word

Notes:

Defines an exact path without wildcards or variables. **path** must represent a complete (hard-coded) path.

path != word

Notes:

Defines an exact path exclusion without wildcards or variables. **path** must represent a complete (hard-coded) path.

date in ((| [) dateExp, dateExp () |])

Notes: The edges are

inclusive if [or] is used.

exclusive if (or) is used.

date dateop dateExp

Notes:

dateop can be <, >, <=, >=, =, == or !=.

author = word

author in (word-list)

comment matches word

Notes:

Does a full-text search.

comment = string

Notes:

Matches *string* exactly.

Most comments end in a new line, so remember to add \n at the end of your string.

comment =~ string

Notes:

string is a regular expression.

content matches word

Notes:

Does a full-text search.

At this time searches are restricted to HEAD revisions.

(modified|added|deleted)? on branch word

Notes:

Selects all revisions on a branch.

modified excludes the branch-point of a branch.

added selects all revisions on the branch if any revision was added on the branch.

deleted selects all revisions on the branch if any revision was deleted on the branch.

tagged op? word

Notes:

op can be <, >, <=, >=, =, == or !=.

op defaults to == if omitted.

These operators are 'positional' and select revisions that appear on, after, and/or before the given tag.

between tags tag-range

after tag word

before tag word

is head (on word)?

Notes:

This selects the top-most revision on any branch, if no branch is specified.

is (dead | deleted)

Notes:

Means the revision was removed/deleted.

	<p>is added Notes: Means the revision was added (or re-added).</p> <p>csid = word Notes: Selects all revisions for the given changeset ID.</p> <p>p4:jobid = word Notes: finds revisions whose Perforce jobid is <i>word</i>.</p> <p>p4:jobid =~ word Notes: finds revisions whose Perforce jobid matches regex <i>word</i>.</p> <p>reviewed Notes: (<i>applies to Crucible reviews</i>) alias for in or before any closed review.</p> <p>(in before in or before) review word</p> <p>(in before in or before) any (review states)? review</p> <p>Notes: <i>word</i> is a review key. in selects reviewed revisions. If a review contains a diff, then only the most recent revision is considered in the review. before selects all revisions in a file prior to the revision in the review. <i>review states</i> is a comma-separated list of open, closed, draft.</p>
<i>tag-range:</i>	<p>(([) T1:word, T2:word ()]) Notes: A range of revisions between those tagged T1 and T2. The edges are: inclusive if [or] is used. exclusive if (or) is used. You can mix edge types. These are all valid: (T1,T2), [T1,T2], (T1,T2] and [T1,T2). Having trouble with Subversion tags? See How tags work in Subversion for more information.</p>
<i>word:</i>	Any <i>string</i> , or any non-quoted word that does not contain white space or any other separators.
<i>string:</i>	<p>A sequence enclosed in either " (double quotes) or ' (single quotes). The following escapes work: \ ' \ " \n \r \t \b \f. Unicode characters can be escaped with \uXXXX. You can also specify strings in 'raw' mode like r"foo". (Similar to Python's raw strings. See Python's own documentation).</p>
<i>dateExp:</i>	See our Date expressions reference guide for more information on date formats.
<i>return-clauses:</i>	<p><i>return-clause</i> (, <i>return-clause</i>)* A return clause signifies that you want control over what data is returned/displayed.</p>
<i>return-clause:</i>	<p>(path dir directory revision author date comment csid isBinary totalLines linesAdded linesRemoved isAdded isDeleted isCopied isMoved tags reviews aggregate) (as word)? The attribute to return, optionally followed by a name to use for the column. Notes: reviews applies to Crucible reviews.</p>

<p><i>aggregate-return-field:</i></p>	<p>(count(revisions) count(<i>binary-field</i>) count(distinct <i>other-field</i>) sum(<i>numeric-field</i>) average(<i>numeric-field</i>) max(<i>numeric-field</i>) min(<i>numeric-field</i>))</p> <p>The aggregate field to return.</p> <p>Notes:</p> <p><i>binary-fields</i> are isBinary, isAdded, isDeleted, isCopied, isMoved. e.g. count(isAdded) will return the number of added files.</p> <p><i>numeric-fields</i> are totalLines, linesAdded, linesRemoved.</p> <p><i>other-field</i> can be path, dir, author, date, csid, tags or reviews. e.g. count(distinct path) will return the number of unique paths. count(distinct tags) will return the number of unique tags.</p> <p>If a group by is given, give sub-totals for each group.</p> <p>With no group by clause, you can have:</p> <ul style="list-style-type: none"> • return <i>normal columns</i> • return <i>aggregates</i> <p>With a group by changeset csid clause:</p> <ul style="list-style-type: none"> • return <i>normal columns</i> • return csid, comment, date, author, <i>aggregates</i> <p>With a group by file path clause:</p> <ul style="list-style-type: none"> • return <i>normal columns</i> • return path, <i>aggregates</i> <p>With a group by dir directory clause:</p> <ul style="list-style-type: none"> • return <i>normal columns</i> • return dir, <i>aggregates</i> <p>i.e. The EyeQL can contain a returns clause that contains all non-aggregate columns, or all aggregate columns. Non-aggregate and aggregate columns can only be mixed if the columns are unique for the grouping.</p>
<p><i>limit-clause:</i></p>	<p>(<i>length</i> <i>offset</i>, <i>length</i> <i>length</i> offset <i>offset</i>)</p> <p>Notes: Limits the number of results to return. <i>offset</i> specifies the starting point of the truncated result set and <i>length</i> specifies the set length. <i>offset</i> is zero-based.</p>

Examples

The following examples demonstrate using EyeQL to extract information from your repository.

Find files removed on the Ant 1.5 branch:

```
select revisions where modified on branch ANT_15_BRANCH and is dead group by changeset
```

As above, but just return the person and time the files were deleted:

```
select revisions where modified on branch ANT_15_BRANCH and is dead return path, author, date
```

Find files on branch and exclude delete files:

```
select revisions where modified on branch ANT_15_BRANCH and not is deleted group by changeset
```

Find changes made to Ant 1.5.x after 1.5FINAL:

```
select revisions where on branch ANT_15_BRANCH and after tag ANT_MAIN_15FINAL group by changeset
```

Find changes made between Ant 1.5 and 1.5.1:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_151_FINAL] group by changeset
```

As above, but show the history of each file separately:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_151_FINAL] group by file
```

Find Java files that are tagged ANT_151_FINAL and are head on the ANT_15_BRANCH: (i.e. files that haven't changed in 1.5.x since 1.5.1)

```
select revisions from dir /src/main where is head and tagged ANT_151_FINAL and on branch ANT_15_BRANCH and path like *.java group by changeset
```

Find changes made by conor to Ant 1.5.x since 1.5.0:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_154] and author = conor group by changeset
```

Find commits that do not have comments:

```
select revisions from dir / where comment = "" group by changeset
```

Find the 10 most recent revisions:

```
select revisions order by date desc limit 10
```

Find the 5th, 6th & 7th revisions:

```
select revisions order by date limit 4, 3
```

Find commits between two dates:

```
select revisions where date in [2008-03-08, 2008-04-08]
```

Find revisions that do not have any associated review:

```
select revisions where (not in any review)
```

Return number of matched revisions, the number of files modified, authors who modified code, changesets, tags, and reviews:

```
select revisions
where date in [ 2003-10-10, 2004-12-12 ]
return count(revisions), count(distinct path), count(distinct author),
count(distinct csid), count(distinct tags), count(distinct reviews)
```

As Sub-totals for each distinct changeset, Return csid, the author, date, comment, number of matched revisions, the number of files modified, the lines added/removed:

```
select revisions
where date in [ 2003-10-10, 2004-12-12 ]
group by changeset
return csid, author, date, comment, count(revisions), count(distinct path), sum(linesAdded), sum(linesRemoved)
```

For each matched file, return the file name, number of matched revisions, the lines added/removed:

```
select revisions
where date in [ 2003-10-10, 2004-12-12 ]
group by file
return path, count(revisions), sum(linesAdded), sum(linesRemoved)
```

Show all the changesets that are not in or before any closed review:

```
select revisions
from dir /
where not reviewed
group by changeset
return csid, author, count(revisions), comment
```




Includes changesets that were explicitly in a review

```
select revisions
where not in any closed review and not in any open review
group by changeset
return path, revision, author, date, csid, reviews
```

Integrating FishEye with Atlassian applications

You can integrate FishEye with the following Atlassian applications:

	<p>When FishEye is integrated with a JIRA application you can:</p> <ul style="list-style-type: none"> • Use smart commits to transition JIRA issues • Delegate user and group management to the JIRA application <p>When Crucible is integrated with JIRA, you can:</p> <ul style="list-style-type: none"> • Transition JIRA application issues <p>When a JIRA application is integrated with FishEye, you can:</p> <ul style="list-style-type: none"> • View an issue's FishEye changesets • Add the FishEye Charts Gadget to your JIRA application dashboard • Add the FishEye Recent Changesets Gadget to your JIRA application dashboard • View an issue's Crucible reviews • Add the Crucible Charts Gadget <p>See Configuring development tools for detailed information.</p>
	<p>When FishEye is used with Crucible, you can:</p> <p><i>In FishEye:</i></p> <ul style="list-style-type: none"> • Use smart commits to create Crucible reviews, add reviews to new reviews and update an existing review. <p><i>In Crucible:</i></p> <ul style="list-style-type: none"> • When using iterative reviews in Crucible, you will be prompted when a new version of a file is available. • Files and changesets displayed in activity streams (e.g. the dashboard activity stream) render as links to the relevant files/changesets. • See your content roots and repositories associated with projects. • See repository lists and browse repositories using the Files tab • View charts or code metrics. <p>See Crucible and FishEye.</p>

	<p>When FishEye is integrated with Bitbucket Server:</p> <ul style="list-style-type: none"> • You can easily add Bitbucket Server repositories to FishEye with a single click. Once added, the repository behaves just like a native repository in FishEye, so your team gets all the benefits of FishEye indexing, browsing and searching. Furthermore, the repository becomes available to Crucible, so you can perform in-depth code reviews for changes in the repository. • A push to a Bitbucket Server repository that has been added to FishEye automatically triggers FishEye to run an incremental index. You don't have to configure polling for new commits, or set up dedicated FishEye web hooks in your Bitbucket Server instance.
	<p>See:</p> <ul style="list-style-type: none"> • Integrating Crowd with Atlassian FishEye • Integrating Crowd with Atlassian Crucible
	<p>When FishEye is integrated with your Bamboo continuous integration server, you can view the code changes that triggered a build. When a build fails due to a compilation error or failed test, you can explore the failed build in FishEye and jump directly into the changeset that broke the build. You can view the history of that changeset to see what the author was trying to fix, take advantage of the the side-by-side diff view to analyze the change and then open the correct files in your IDE.</p> <p>For more details see Viewing the Code Changes that Triggered a Build.</p>

JIRA Integration in FishEye

When FishEye is integrated with JIRA Software, you and your team get all the benefits described on this page. Go straight to [Linking FishEye to JIRA Software](#) if you want to connect FishEye to a JIRA Software server.

You can also use JIRA Software for delegated management of your FishEye users. See [JIRA and Crowd authentication](#).

See [Configuring development tools](#) for the full story of how Atlassian tools work together to give you a fast and guided software development process.

Starting in JIRA 6.2.2 the Source and Reviews tabs are only displayed if JIRA Software is unable to display the associated information in the Development Tools panel.

Related pages:

- [Linking FishEye to JIRA Software](#)
- [Enabling Smart Commits](#)

Your user tiers don't need to match between JIRA Software and FishEye/Crucible in order to integrate them. JIRA Software users that are not FishEye users will see the same view as FishEye users within JIRA Software, but will not be able to log in to FishEye to view the source/reviews.

Check development progress of a version in JIRA Software

FISHEYE 3.3+

JIRA 6.4+

The Release Hub in JIRA Software shows the relevant issues and development information for a version – so you can determine which issues are likely to ship at a glance. With JIRA Software and FishEye connected, the release page can warn you about potential development issues that could cause problems for your release.

Version 2.0 UNRELEASED

Start: 03/Nov/14 Release: 15/Feb/15 [Release Notes](#)

48 Warnings
 273 Issues in version
 262 Issues done
 7 Issues in progress
 4 Issues to do

Warnings indicate when the status of a JIRA issue doesn't reflect related development activity. For example: an issue marked complete that has an open pull request should be marked as still being in progress.

Unreviewed Code
 These issues have been marked complete but the commits are not part of a pull request or review.

1–4 of 4

P	T	Key	Summary	Assignee
		SSP-1663	UI is not loading in IE8	Andrew Swar
		SSP-1979	Incorrect permissions for new report	Andrew Swar
		SSP-1555	As a developer, I want to view the build status for an issue	Bruce Temple
		SSP-1660	Missing error message when Bamboo is unavailable	Eduardo Soai

1–4 of 4

From the Release Hub you can also:

- Release a version
- Mark a version as complete
- Move incomplete issues to other versions
- Trigger release builds (if JIRA is connected to Bamboo)
- Warnings that help you reconcile what is happening in development with JIRA Software data.



To view the Release Hub (with the project sidebar enabled), navigate to a project, click on **Releases**, then select a version listed. See [Checking the progress of a version](#) for more detailed information about using the Release Hub in JIRA Software.

See in JIRA Software the FishEye repository branches related to an issue

FISHEYE 3.3+

JIRA 6.2+

For FishEye 3.3 and later, the FishEye repository branches related to a JIRA Software issue are summarized in the Development panel for the issue, when JIRA Software and FishEye are connected with an [application link](#). To see details of the branches, simply click the **branches** link. You can see which repository each branch is in and when the last commit was made. As long as the issue key is included in the branch name the branch is automatically linked to the JIRA Software issue.









FUSE-601: 2 branches		
<div> <div>Bitbucket 1</div> <div>FishEye / Crucible 1</div> </div>		
Repository	Branch	Last commit
 FE-hg	 3.3-FUSE-601-more-commit-details	10f3418

See in JIRA Software the commits related to an issue

FISHEYE 3.3+

JIRA 6.2+

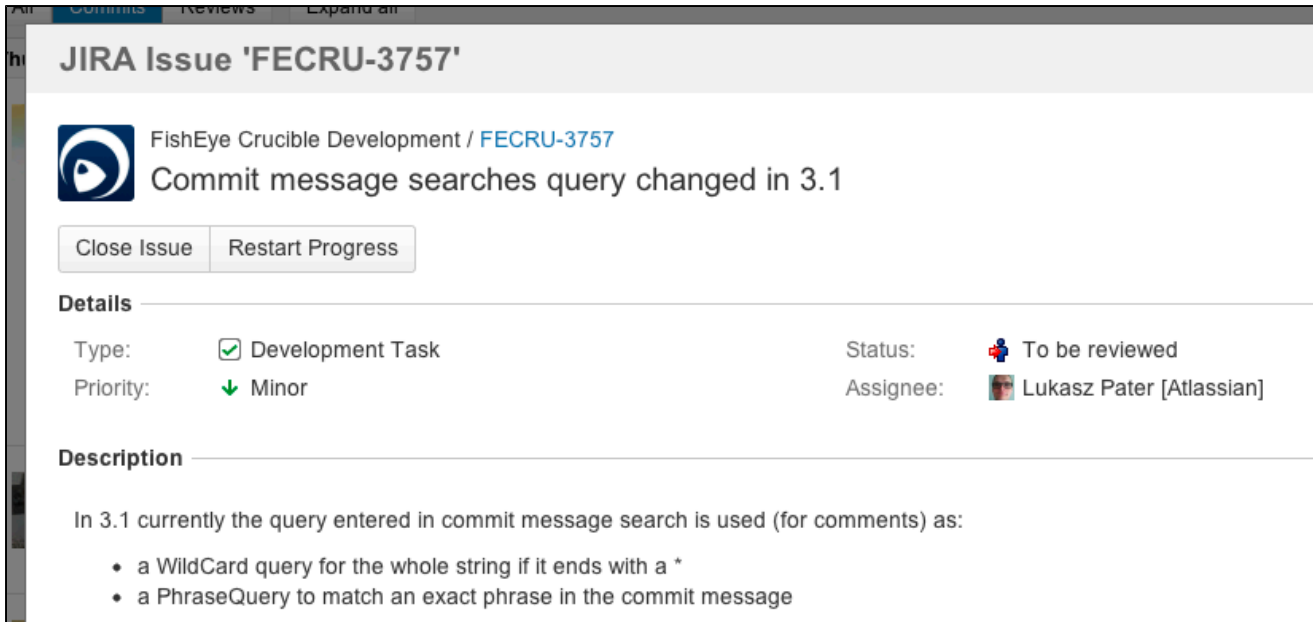
For FishEye 3.3 and later, the FishEye repository commits related to a JIRA Software issue are summarized in the Development panel for the issue, when JIRA Software and FishEye are connected with an [application link](#). You can click the **commits** link to see detailed information such as who made each commit, when they committed, and how many files were changed. Click through to see a particular commit in the FishEye instance where the commit was made. A developer only needs to add the issue key to the commit message for that commit to be automatically linked to the JIRA Software issue.

FUSE-113: 62 unique commits (and 56 duplicates)						
<div> <div>Bitbucket 40</div> <div>FishEye / Crucible 56</div> <div>Stash 22</div> </div>						
<div> <div> bamboo-git (atlaseye)</div> <div>Create</div> </div>						
Author	Commit	Message	Reviews	Date	File	
	9b9b93b	FUSE-113 Functional tests	 55	05/Dec/13 11:24 AM	8 fil	
	cfba99b	FUSE-113 FT for anonymous access	 81	18/Nov/13 1:27 PM	3 fil	
	3dea4f4	FUSE-113 post review	 81	18/Nov/13 11:00 AM	1 fil	
	2b69f9e	FUSE-113 post review	 2	18/Nov/13 11:00 AM	1 fil	
	563fa3b	FUSE-113 post review	 81	18/Nov/13 10:59 AM	1 fil	

See the JIRA Software issues related to commits

FISHEYE 3.1+

FishEye recognizes JIRA Software issue keys, and displays those as links in places such as the activity stream, side-by-side diffs, and commit messages:



JIRA Issue 'FECRU-3757'

FishEye Crucible Development / [FECRU-3757](#)

Commit message searches query changed in 3.1

Close Issue Restart Progress

Details

Type: ☒ Development Task Status: To be reviewed

Priority: Minor Assignee: Lukasz Pater [Atlassian]

Description

In 3.1 currently the query entered in commit message search is used (for comments) as:

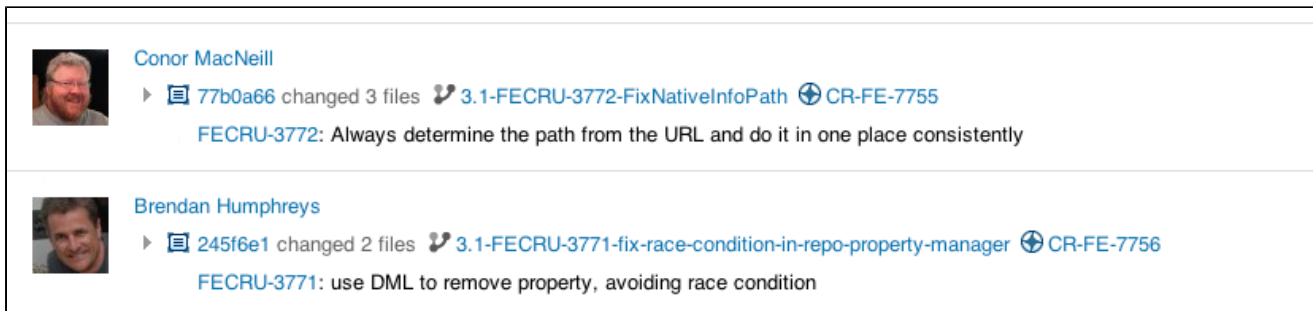
- a WildCard query for the whole string if it ends with a *
- a PhraseQuery to match an exact phrase in the commit message

Click on the linked issue key to see details for the issue, as described next.

See the details for JIRA Software issues

FISHEYE 3.1+

Click a linked issue key anywhere in FishEye to see the details of that issue in a dialog. And you can click the issue key at the top of the dialog to go straight to the issue in JIRA Software:



Conor MacNeill

77b0a66 changed 3 files 3.1-FECRU-3772-FixNativeInfoPath CR-FE-7755

FECRU-3772: Always determine the path from the URL and do it in one place consistently

Brendan Humphreys

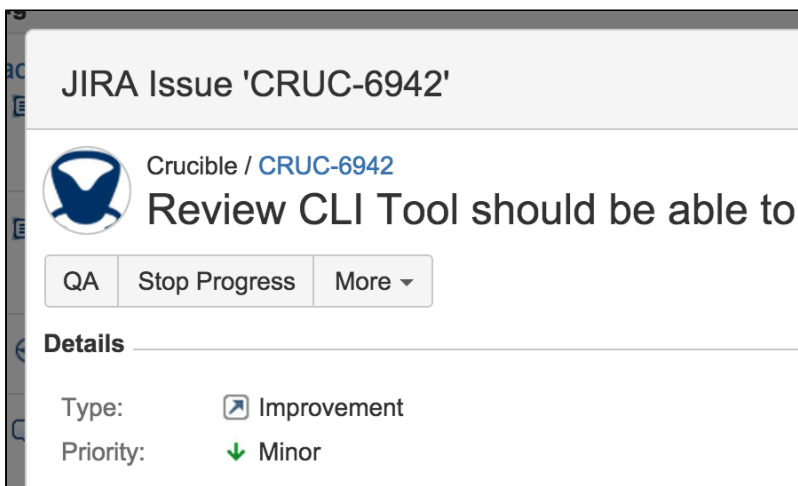
245f6e1 changed 2 files 3.1-FECRU-3771-fix-race-condition-in-repo-property-manager CR-FE-7756

FECRU-3771: use DML to remove property, avoiding race condition

Transition JIRA Software issues from within FishEye

FISHEYE 3.1+

You can easily [transition](#) a JIRA issue from within FishEye. For example, when viewing a commit, you may want to transition the related JIRA Software issue into QA. Click on a linked JIRA issue anywhere in FishEye to see a dialog with the available workflow steps:



JIRA Issue 'CRUC-6942'

Crucible / [CRUC-6942](#)

Review CLI Tool should be able to

QA Stop Progress More ▾

Details

Type: Improvement

Priority: Minor

Click on a step in the dialog, and complete any displayed fields as required. If there are custom required fields that are unsupported by FishEye, just click **Edit this field in JIRA** to transition the issue directly in JIRA.

See issues from multiple instances of JIRA Software

FISHEYE 3.1+

FishEye can link to more than one JIRA server at a time, so different teams can work with their own projects in different JIRA Software instances, or a single team can link to issues across multiple JIRA Software servers.

Integrating FishEye with Bitbucket Server

This page...

... describes the benefits for you and your team, when FishEye is integrated with [Bitbucket Server](#) (formerly known as Stash).

Set it up...

... with our short guide to help FishEye admins [connect FishEye to Bitbucket Server](#).

Bitbucket Server is...

... Atlassian's on-premise Git repository management solution for enterprise teams. Read about [getting started with Git and Bitbucket Server](#).

Easily keep FishEye repositories synced with Bitbucket Server

FISHEYE 3.5 +

STASH 3.1 +

Once you add a Bitbucket Server Git repository to FishEye, a push to the repository automatically triggers FishEye to run an incremental index. You don't have to configure polling to detect new commits, or set up dedicated FishEye web hooks in your Bitbucket Server instance.

A FishEye administrator only needs to set up an [application link](#) with Bitbucket Server for FishEye to be ready to respond to 'refs changed' notifications published by Bitbucket Server.

Note that FishEye may not receive the notifications published by Bitbucket Server under some circumstances, for example if FishEye is being restarted, or if there are intermittent network issues. Therefore, we don't recommend disabling polling completely, but that the polling period be extended to 2 hours or even 24 hours, so FishEye can catch up with any missed events. See [Updater](#).

Note also that Bitbucket Server was formerly known as Atlassian Stash.

Easily add Bitbucket Server repositories to FishEye

FISHEYE 3.4 +

STASH 2.11.4 +

When FishEye is integrated with Bitbucket Server, a FishEye administrator can easily add Bitbucket Server repositories to FishEye with a single click. Once added, the repository behaves just like a native repository in FishEye, so your team gets all the benefits of FishEye indexing, browsing and searching. Furthermore, the repository becomes available to Crucible (when integrated), so you can perform in-depth code reviews for changes in the repository.

For detailed instructions for adding a Bitbucket Server repository to FishEye, see [Adding an external repository](#).

Transitioning issues in JIRA

Transition a JIRA Software issue from within FishEye in any of the following ways:

- [Transition an issue automatically](#)
- [Transition an issue manually from FishEye](#)
- [Transition an issue with your commit message](#)

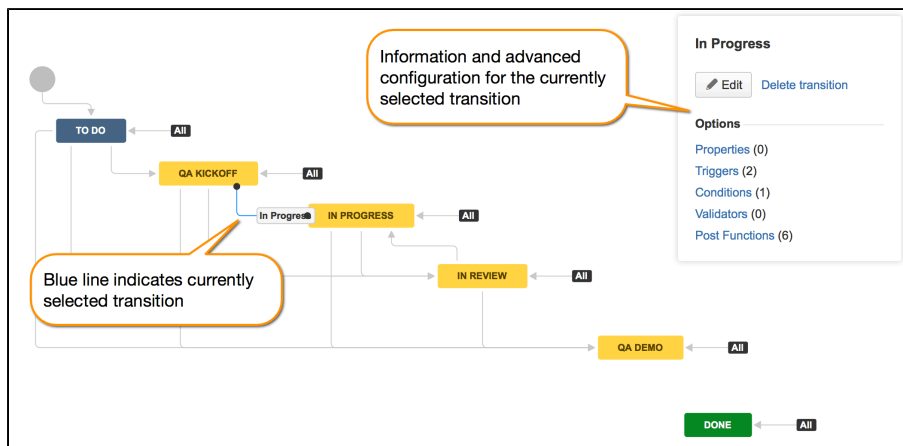
Transition an issue automatically

FISHEYE 3.5.2+

JIRA 6.3.3+

Your JIRA Software workflow can now respond to events in your linked

development tools. For example, when a commit is pushed, your JIRA Software workflow can be configured to automatically transition the related issue. Configure this using triggers for transitions within the JIRA Software workflow editor.



The events published by FishEye are:

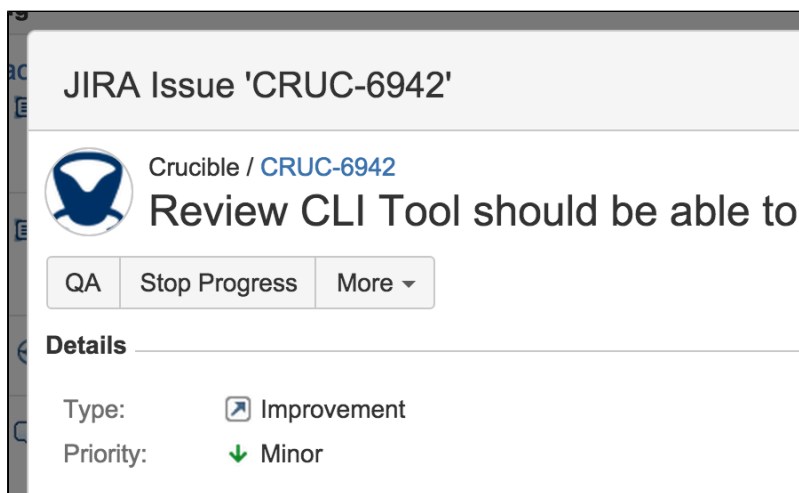
- Commit created
- Branch created

A FishEye administrator can disable event publishing if required. See [JIRA workflow triggers](#).

Transition an issue manually from FishEye

FISHEYE 3.1+

You can easily [transition](#) a JIRA Software issue from within FishEye. For example, when viewing a commit, you may want to transition the related JIRA Software issue to the QA status. Click on a linked JIRA Software issue key in FishEye to see a dialog with the available workflow steps:



Click on a step in the dialog, and complete any displayed fields as required. If there are custom required fields that are unsupported by FishEye, just click **Edit this field in JIRA Software** to go to JIRA Software to transition the issue.

Transition an issue with your commit message

FISHEYE 2.7+

Use Smart Commits when you are committing to a repository hosted in FishEye to transition JIRA Software issues. Smart commits allow you to embed commands into your commit messages, which FishEye detects and actions.

For example, if you include the following text in your commit message, FishEye will add the comment 'Task completed ahead of schedule' and resolve the issue, when you perform your commit:

```
JRA-123 #comment Task completed ahead of schedule
#resolve
```

Read more about [Using Smart Commits](#).

FishEye FAQ

FishEye FAQ

Answers to frequently asked questions about configuring and using FishEye.

- **Top Evaluator Questions**
 - [How do I fix problems with indexing my repository?](#)
 - [How do I setup JIRA integration?](#)
 - [How do I setup LDAP or external user management?](#)
 - [How do I speed up slow CVS updates?](#)
 - [How do I start FishEye as a Windows service?](#)
 - [How do I view changesets and diffs?](#)
 - [How is FishEye licensed?](#)
 - [What kind of search capabilities does FishEye have?](#)
 - [What programming languages are supported?](#)
- **General FAQ**
 - [About database encoding](#)
 - [About the Lines of Code Metric](#)
 - [Cannot View Lines of Code Information in FishEye](#)
 - [Finding your Server ID](#)
 - [How Do I Archive a Branch within Perforce](#)
 - [How do I Avoid Long Reindex Times When I Upgrade](#)
 - [Mercurial Known Issues](#)
 - [Ordering of Branches Important When Visualizing Git Changesets](#)
 - [Permanent authentication for Git repositories over HTTP\(S\)](#)
 - [Perforce Changesets and Branches](#)
 - [What SCM systems are supported by FishEye?](#)
 - [Automating Administrative Actions in Fisheye](#)
 - [How are indexing requests handled when they are triggered via commit hook](#)
- **Installation & Configuration FAQ**
 - [Can I deploy FishEye or Crucible as a WAR?](#)
 - [Does Fisheye support SSL \(HTTPS\)?](#)
 - [Improve FishEye scan performance](#)
 - [Migrating FishEye Between Servers](#)
 - [Setting up a CVS mirror with rsync](#)
 - [What are the FishEye System Requirements?](#)
 - [How to reset the Administration Page password in FishEye or Crucible](#)
 - [How Do I Configure an Outbound Proxy Server for FishEye](#)
 - [How to remove Crucible from FishEye 2.x or later](#)
 - [How to run Fisheye or Crucible on startup on Mac OS X](#)
- **Licensing FAQ**
 - [Are anonymous users counted towards FishEye's license limits?](#)
 - [Restrictions on FishEye Starter Licenses](#)
 - [Updating your FishEye license](#)
 - [Git or Hg Repository exceeds number of allowed committers](#)

- [Example EyeQL Queries](#)
 - [How do I find changes made to a branch after a given tag?](#)
 - [How do I filter results?](#)
 - [How do I find changes between two versions, showing separate histories?](#)
 - [How do I find changes made between two version numbers?](#)
 - [How do I find commits without comments?](#)
 - [How do I find files on a branch, excluding deleted files?](#)
 - [How do I find files removed from a given branch?](#)
 - [How do I find revisions made by one author between versions?](#)
 - [How do I select the most recent revisions in a given branch?](#)
 - [How do I show all changesets which do not have reviews?](#)
- [Integration FAQ](#)
 - [How do I disable the Source \(FishEye\) tab panel for non-code projects?](#)
 - [How do I enable debug logging for the JIRA FishEye plugin?](#)
 - [How do I uninstall the JIRA FishEye plugin?](#)
 - [How is the Reviews \(Crucible\) tab panel for the JIRA FishEye Plugin populated?](#)
 - [What do I do if I discover a bug with the JIRA FishEye plugin?](#)
- [Subversion FAQ](#)
 - [Configuring Start Revision based on date](#)
 - [Errors 'SEVERE assert' or 'Checksum mismatch'](#)
 - [FishEye fails to connect to the Subversion repository after a short time of successful operation.](#)
 - [How can FishEye help with merging of branches in Subversion?](#)
 - [Subversion Changeset Parents and Branches](#)
 - [SVN Authentication Issues](#)
 - [What are Subversion root and tag branches?](#)
 - [Why do I need to describe the branch and tag structure for Subversion repositories?](#)
 - [Why don't all my tags show up in FishEye?](#)
 - [About merges in Subversion](#)
- [CVS FAQ](#)
 - [How does FishEye calculate CVS changesets?](#)
 - [How is changeset ancestry implemented for CVS?](#)
- [Support Policies](#)
 - [Bug Fixing Policy](#)
 - [New Features Policy](#)
 - [Security Bugfix Policy](#)
- [Troubleshooting](#)
 - [After I commit a change to my CVS repository, it takes a long time before it appears in FishEye.](#)
 - [FishEye freezes unexpectedly](#)
 - [I have installed FishEye, and the initial scan is taking a long time. Is this normal?](#)
 - [I have installed FishEye, but there is no data in the Changelog.](#)
 - [Initial scan and page loads are slow on Subversion](#)
 - [JIRA Integration Issues](#)
 - [Manually Generating a Thread Dump](#)
 - [Message 'org.tigris.subversion.javahl.ClientException svn Java heap space'](#)
 - [Problems with very long comments and MySQL migration](#)
 - [URLs with encoded slashes don't work, especially in Author constraints](#)
 - [Manually Generating a Thread Dump \(Draft\)](#)
 - [All Repositories fail - Could not execute query \(MySQL database\)](#)
- [FishEye Developer FAQ](#)
- [Contributing to the FishEye Documentation](#)
 - [FishEye Documentation in Other Languages](#)
- [FishEye Resources](#)
- [Glossary](#)
- [Collecting analytics for FishEye](#)

Do you have a question, or need help with FishEye? Please create a support request.

General FAQ

FishEye General FAQ

- [About database encoding](#)
- [About the Lines of Code Metric](#)
- [Cannot View Lines of Code Information in FishEye](#)
- [Finding your Server ID](#)
- [How Do I Archive a Branch within Perforce](#)
- [How do I Avoid Long Reindex Times When I Upgrade](#) — If reindexing your repository takes longer than you can allow, you can use a temporary copy of your repository and FishEye instance to reduce downtime during the reindexing process.
- [Mercurial Known Issues](#)
- [Ordering of Branches Important When Visualizing Git Changesets](#)
- [Permanent authentication for Git repositories over HTTP\(S\)](#)
- [Perforce Changesets and Branches](#)
- [What SCM systems are supported by FishEye?](#)
- [Automating Administrative Actions in FishEye](#)
- [How are indexing requests handled when they are triggered via commit hook](#)

About database encoding

It is possible to have files in your repository whose names differ only in case, e.g. `Foo.java` and `foo.java`. Hence, your database will need to use rules for comparing string values which recognize that upper and lower case letters are different, that is, the database should use 'case sensitive collation'.

If your database was originally configured to use case-insensitive and/or non-UTF8 collation, FishEye will display the following message at the bottom of your screen:

"Your database is not using a case sensitive UTF8 encoding for character fields."

The following sections provide instructions for changing your database collation for each database type supported by FishEye and Crucible.

On this page:

- [MySQL](#)
- [Oracle](#)
- [PostgreSQL](#)
- [SQL Server](#)

Related pages:

- [Migrating to an external database](#)

MySQL

Please take a backup of your database before changing its collation.

To change your collation to `utf8_bin` you need to change your database's default collation, but as this only affects newly created tables you will also need to change the collation on the table for which case sensitivity is critical.

Change your database's collation

Use the `ALTER DATABASE` command, as follows:

```
alter database character set utf8 collate utf8_bin;
```

Change collation for the `CRU_STORED_PATH` table

Use the **ALTER TABLE** command, as follows:

```
alter table cru_stored_path convert to character set utf8 collate  
utf8_bin;
```

Oracle

Oracle collation encoding must be configured when installing the database server. It cannot be configured on a per database level. When installing Oracle, you should select the **AL32UTF8** encoding.

PostgreSQL

Please take a backup of your database before changing its collation.

If you have created your PostgreSQL database with the incorrect encoding, you will need to dump your database, drop it, create a new database with the correct encoding and reload your data.

You can do this using the [standard database migration procedure](#) – instead of migrating from HSQLDB to PostgreSQL, you migrate from a PostgreSQL database with the incorrect encoding to one created with the correct encoding.

SQL Server

Please take a backup of your database before changing its collation.

Unfortunately, changing the database collation for an existing SQL Server database (even using the **ALTER DATABASE ... COLLATE...** statement) does not change the collation for existing objects stored in the database. See <http://blogs.msdn.com/b/qingsongyao/archive/2011/04/04/do-not-alter-database-collation-in-your-server.aspx> for an explanation of this.

The recommended route for changing the collation for SQL Server is to migrate to a new database that has the correct collation configuration. You can do this using the [standard database migration procedure](#) – instead of migrating from HSQLDB to SQL Server, you migrate from a SQL Server database with the incorrect collation to one created with the correct collation.

The correct collation to use when you create the new SQL Server database is **Latin1_General_CS_AS**.

About the Lines of Code Metric

This page contains information about the Lines of Code metric and how it is processed and represented by FishEye.

On this page:

- [Definition](#)
- [Disadvantages](#)
- [LOC in FishEye](#)
- [User-Specific LOC](#)

Definition

Lines of Code or LOC (also known as Source Lines of Code - SLOC) is a quantitative measurement in computer programming for files that contains code from a computer programming language, in text form. The number of lines indicates the size of a given file and gives some indication of the work involved.

LOC is literally the count of the number of lines of text in a file or directory. In FishEye, blank lines and comment

lines are counted toward the total lines of code.

LOC for a file/directory is the total number of lines in the relevant files, while LOC for an author is the number of lines **blamed on that author**. Neither of these should ever be less than zero. However, the **change** in LOC over a period of time can be negative if there was a net reduction in the LOC over the period.

Disadvantages

While it can be useful, LOC has some [well documented disadvantages](#). Keep these disadvantages and limitations in mind when using LOC in your work environment.

In addition, the nature of branching in [SCM](#) applications means that calculating a LOC value for a whole project is not possible. A naive summation of the LOC of all the branches will give a meaningless number that jumps every time a branch is copied to create a new branch. Thus, in FishEye we usually look at the LOC of the trunk, unless we can infer from the context that another branch is more appropriate.

LOC in FishEye

FishEye calculates the LOC for the trunk only. For SVN repositories it can calculate LOC for a branch if it is "[tricked](#)" to see the branch as part of the trunk. FishEye also calculates the LOC for each user, unless that facility is turned off in the repository (see [Store Diff Info](#)). The LOC count will include all files except those identified by the SCM as binary.

FishEye presents LOC data as charts of the change in LOC over time, and as informational statistics in various places:

- **Chart pages**
The best way to explore the evolution of LOC in your project is the [LOC chart report](#) where you can easily filter the LOC by branch, author, file extension and date range. Here you can investigate what caused a particular spike in the LOC charts, or find the user whom has the most lines of code blamed on them and how this has changed over time.
- **Repository-specific activity pages**
These show trunk LOC statistics for the repository, limited to the directory being viewed and its subdirectories. The LOC charts show the LOC for the directory, using trunk LOC unless the directory can be identified as a branch.
- **User pages**
Here, the statistics pane in the sidebar shows the trunk LOC blamed on the user for the all repositories that have user-specific LOC enabled. The chart shows the trunk LOC from all the repositories that the user has contributed to.
- **The global User List page**
This shows the trunk LOC for all users from the repositories that have user-specific LOC enabled. Repository-specific user lists (in repositories that have user-specific LOC enabled) show the trunk LOC for the users and committers, limited to the directory being viewed and its subdirectories.
- **Project pages**
This shows a chart of the LOC for all associated repository paths, and statistics include the trunk LOC for those paths.

User-Specific LOC

The evolution of user-specific LOC over subsequent commits can appear at first glance to be counter-intuitive. It is important to keep in mind that the LOC for a given user is the number of lines in the repository that were last changed by them (as calculated by FishEye).

A couple of simple examples:

- Alice adds a files with 30 lines to the SCM. Her LOC for this file is now 30. She then edits the file, deletes 10 lines and adds 20 (+20 -10). Her LOC is now 40, as is the LOC of the file.
- Alice adds a files with 30 lines to the SCM. Her LOC for this file is now 30. Now Bob edits the file, deletes 10 lines and adds 20 (+20 -10). Alice now has LOC of 20, because Bob deleted 10 lines that were blamed on her, and Bob has LOC of 20, from the 20 lines he added. The total LOC is still 40.


A user can have LOC on a branch that they have never committed on, *if something that has been blamed on them is copied*. For example, a developer may have never committed to a particular branch, but FishEye may still report a lot of LOC for them in that area.

One current limitation of FishEye's user-specific LOC calculation is the handling of merging. For example, if a file has been changed on both trunk and branch, and the changes made on the branch are merged to trunk, the changes made on branch will generally be blamed on the person who did the merge; **not** the person who made the change.

Cannot View Lines of Code Information in FishEye

Symptoms

The LOC (Lines of Code) information in FishEye cannot be seen, for example in [charts](#) or when [viewing the statistics for a user](#).

 See [About the Lines of Code Metric](#) for more information about the usage of the LOC (Lines of Code) metric in FishEye.

Cause

There are four possible causes:

- LOC data will not be shown for users if the [Store diff info](#) setting is disabled. If a page is being viewed in FishEye that relates to a particular user or committer, and the Store Diff Info setting is disabled, no LOC information for the user will be visible.
- LOC data is currently not supported for [Mercurial](#) repositories.
- LOC data is currently not supported for [Git](#) repositories.
- The SVN repository is indexing branches only.

Resolution

Cannot view LOC information for specific users or committers:

- Enable the [Store Diff Info](#) setting for the repository in Administration > Repository Settings > Repositories, click on the repository name, and then "SCM Details". A full re-index needs to be performed on the repository after enabling this setting for FishEye to collect the diff information for all revisions in the repository. Please note that the Store Diff Info setting is always enabled for CVS repositories.

The SVN repository is indexing branches only:

- FishEye can calculate LOC for a branch if it is ["tricked" to see the branch as part of the trunk](#).

Finding your Server ID

Your Server ID can be found in the FishEye administration area.

To find your Server ID:

1. Navigate to FishEye's administration area.
2. Click **System Info** (under 'System Settings').

The Server ID for your FishEye server is displayed in the 'License' section.

The Server ID should match the one set for your license. You can check this at <http://my.atlassian.com>.

How Do I Archive a Branch within Perforce

In SVN, a branch exists as a separate directory. However in Perforce, files are given a label to identify them as belonging to the branch. Thus it may not be possible to download the branch as a tarball via FishEye.

You may be able to download the branch as a tarball, depending on your structure:

If it is not a single folder, then it is not possible to download the tarball in your perforce repository.

1. In FishEye, navigate to your perforce repository.
2. In the Constraint section on the left, select the branch. This will return the directories that belong to that branch.
3. If it is one single folder, download the tarball of it. Under `constraint` and `sub directories`, there is a panel tarball giving options on how to download the directory.

How do I Avoid Long Reindex Times When I Upgrade

Mitigating lengthy reindex times

If reindexing your repository takes longer than you can allow, you can use a temporary copy of your repository and FishEye instance to reduce downtime during the reindexing process.

Most upgrades (even major ones) do not require a reindex. If a reindex is required, this will always be explicitly mentioned in the Upgrade Guide for that release.


On this page:

- [Mitigating lengthy reindex times](#)
- [Reindexing with a temporary copy of your FishEye instance](#)
 - [How to make a temporary copy of your FishEye instance](#)
 - [How to make a temporary copy of your repository](#)
 - [How to reindex a single repository on a test server](#)
- [Upgrading your cross-repository index using a temporary staging server](#)

Reindexing with a temporary copy of your FishEye instance

This section describes how to perform a full reindex of a particular repository. Note that, depending on the repository size, the reindex could take up to several days.

To reindex a temporary copy of your FishEye instance:

1. Make a copy of your FishEye instance to another server. See 'How to make a temporary copy of your FishEye instance' below for instructions.
2. Upgrade the temporary FishEye, then start it up, connected to your repository. It will automatically begin the scanning process.
 -  If you are concerned about the repository being overloaded by the scanning process, you can make a copy of that as well. See 'How to make a temporary copy of your repository' below for instructions. If you do that, you must edit the `config.xml` of your temporary FishEye instance to point to your temporary repository.
3. The copied instance will run its course without affecting your production instance.
 - a. Shutdown both your servers completely.
 - b. Make a backup of your `FISHEYE_INST` directory.
 - c. Replace the `FISHEYE_INST/var/cache` directory on live FishEye with the `FISHEYE_INST/var/cache` from your test server.
 - d. Download the latest FishEye/Crucible from [Atlassian downloads](#).
 - e. Follow the instructions in the [Upgrade Guide](#) to upgrade to the new version.
4. The scan of the temporary FishEye instance (and repository, if you copied that also) is complete. You're now free to delete the temporary copy(s).

How to make a temporary copy of your FishEye instance

To make a copy of your FishEye instance, follow the instructions for [Migrating FishEye Between Servers](#).


How to make a temporary copy of your repository

To make a copy of your repository use [rsync](#) (for CVS repositories in the Linux environment) or [svnsync](#) documentation (for Subversion only).

How to reindex a single repository on a test server

If you need to reindex your repository on your production system but don't want to burden your production server, carry out the following steps:

1. Install another instance of FishEye on a test server (the same FishEye version as the one you are using).
2. Add a repository to FishEye with the exact same name and details as that referenced by the production server.
3. Let it finish indexing. Go to **Administration > View Repository List > Stop** (shown next to the name of your repository) and disable on both production and test.
4. Copy the `FISHEYE_INST/var/cache/REPO` directory from the test server over the `FISHEYE_INST/var/cache/REPO` directory on the production FishEye.
5. Trigger a review revision data re-index: **Administration > Repository > Maintenance > Review-Revision Data Index**.

 For this procedure, neither server needs to be shut down.

Upgrading your cross-repository index using a temporary staging server

This section describes how to upgrade the cross-repository index for selected repositories. Note that, depending on the repository size, the reindex will typically finish in a few hours, but should never take longer than a few days.

In this procedure it is assumed that you have a production server (referred to as *PROD* in these instructions) that is running a FishEye version earlier than 3.1, and a separate staging server (*STAGING*) that will be used to perform the cross-repository index upgrade offline.

1. Make a live backup of the *PROD* server with the following options:
 - a. Repository and application caches
 - b. SQL database

You can do this either from the FishEye Admin area (go to **Administration > System Settings > Backup**), or from a command line, for example:

```
$ ./bin/fisheyectl.sh backup -f ~/Documents/backup.zip --no-uploads
--no-templates --no-plugins --cache --no-ao
```

2. Install FishEye 3.1, or a later version, on the *STAGING* server.
3. Restore the backup of *PROD* to the *STAGING* server.
4. Start FishEye on the *STAGING* server. Note that:
 - The cross-repository index upgrade will start automatically on the *STAGING* server. If you want to perform the cross-repository index upgrade for selected repositories only, it is safe to remove unwanted repositories from the *STAGING* server now, either by going to **Administration > Repositories**, or by using REST endpoints (see below).
 - The *STAGING* server doesn't need to have access to configured SCM's as the cross-repository upgrade task does not interact with them.
 - You may want to disable polling on the *STAGING* server. You can either go to **Administration > Repository Settings > Defaults > Updater** to disable polling for all repositories (although this will not affect particular repositories that have been configured to ignore default settings), or go to **Administration > Repository Settings > Repositories > Repository X > Updates** to disable polling for just *Repository X*. Disabling polling is not required, but will avoid logging errors to the FishEye log file if the SCMs are not accessible from the *STAGING* server.
5. Wait for the cross-repository index upgrade to finish on the *STAGING* server. Check by going to **Administration > Repositories**.
6. Stop the *STAGING* server.
7. Make a full backup of the *PROD* server and then stop it.

8. Install the same version of FishEye on the *PROD* server as used on the *STAGING* server (as in step 2 above).
9. Delete the following FishEye indexes on the *PROD* server and replace them with the equivalent caches from the *STAGING* server. You can choose your preferred option to copy files between machines using ssh/scp/rsync, possibly combined with tar/zip tools. The example below shows how the scp command could be used:

```
ssh PROD
cd FISHEYE_INST # replace FISHEYE_INST with the location of your
FISHEYE_INST folder
rm -rf cache/globalfe
rm -rf var/cache/repoX # repeat for each repository repoX that was
upgraded on STAGING server
scp -r STAGING:STAGING_FISHEYE_INST/cache/globalfe cache/
scp -r STAGING:STAGING_FISHEYE_INST/var/cache/repoX var/cache/ #
repeat for each repository repoX that was upgraded on STAGING
server
```

10. Start the *PROD* server.
11. All the changesets that were added to SCMs after backing up the *PROD* server in step 1 will now be indexed on the *PROD* server.

The only drawback with this procedure is that changeset comments added for changesets in *PROD* after step 1 will not get indexed, so they will not appear in the activity stream. There is no easy way to reindex them, apart from fully reindexing each affected repository, which is what this procedure is intended to avoid. A new REST endpoint could be implemented to address this (see

✓ [FECRU-3764](#) - new REST endpoint to reindex changeset comments for a repo CLOSED).

Note: the following REST endpoint could be used to force a cross-repository index upgrade for a selected repository: [/rest-service-fecru/admin/repositories-v1/repoX/reindex-search](#). There should be no need to use this, but it may be useful if something goes wrong.

Mercurial Known Issues

- [CRUC-3474](#): If a file is removed and then another file is copied or moved over the same file within one commit, the ancestor revision is miscalculated and can result in errors in "diff to previous".
- [CRUC-3470](#): Permission changes (and prop changes in repos converted from svn) may result in revisions that have no ancestors - subsequent changes will consider it's parent revision to be their parent revision.
- [CRUC-3468](#): Scanning repositories converted from svn (especially using hgsubversion) can result in commits that take a long time to scan (due to the changes produced by merges from other branches).

Ordering of Branches Important When Visualizing Git Changesets

FishEye 2.6 introduced the [repository commit graph](#). The commit graph allows you to visualize changesets in their branches by showing them in configurable branch "swimlanes". One of the ways in which you can configure the commit graph is by reordering the swimlanes. Reordering swimlanes is useful for non-Git repositories, if you want to show branches in a certain order. However, ordering swimlanes is vital for Git repositories, as it is the only way of determining which branch a commit is displayed in, when a commit belongs to multiple branches.

Git Branches and Changesets in FishEye

Before considering how Git repositories are visualized in the commit graph, it is important to understand how FishEye relates Git changesets to branches.

In FishEye 2.6 and later, FishEye considers the ancestry of a Git changeset when determining which branch it is a part of. Branches can effectively be considered as pointers to changesets. Hence, merging and branching can change the branches that a changeset is considered part of.

For example, if a branch 'fisheye-2.6' is merged back to the 'master' branch, then all changesets that were seen as part of the 'fisheye-2.6' branch only will also be considered to be part of the 'master' (e.g. the changeset will be seen as part of 'master' and 'fisheye-2.6' in the activity stream).

Viewing Git Changesets and Branches in the Commit Graph

The previous section describes how a changeset can be associated with multiple branches, due to its ancestry. Instead of showing the changeset in every branch swimlane on the commit graph, FishEye represents these changesets as described below.

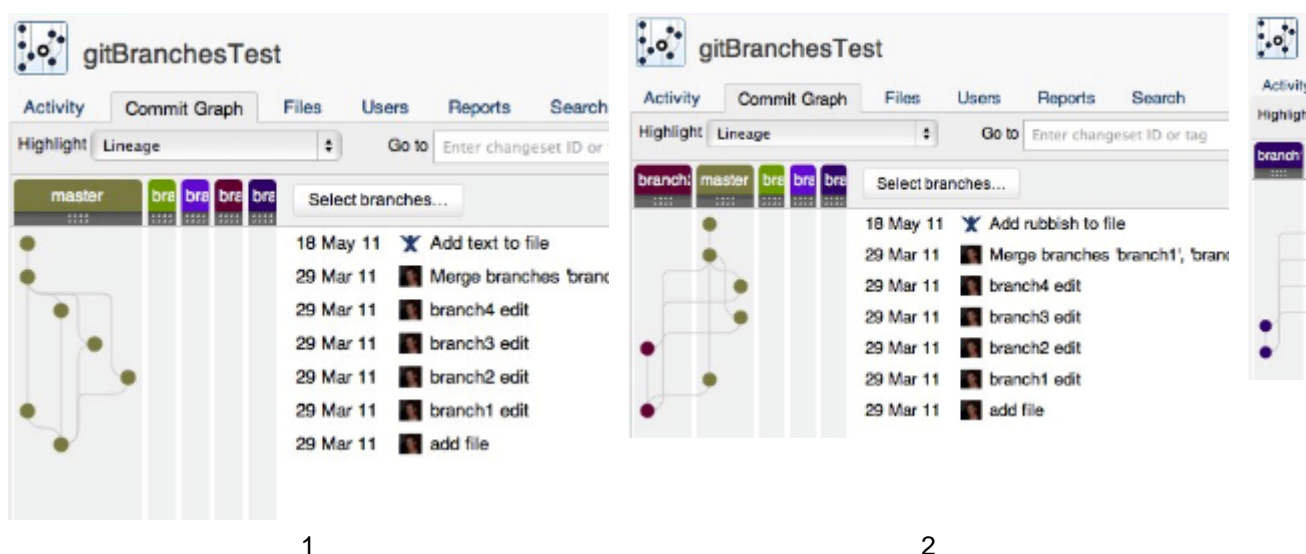
When you view the commit graph for a Git repository, FishEye works from the leftmost swimlane to the right doing the following:

- For each swimlane, FishEye checks if the commit is in that branch. If the commit is in the branch, a dot is shown representing the commit.
- If the commit is not in the branch, the dot for the commit is moved to the next column on the right.

For example, if the 'master' swimlane is to the left of another swimlane, e.g. 'fisheye-2.6' branch, there will be no changesets shown in the 'fisheye-2.6' swimlane, as all the commits will be picked up in the 'master' swimlane. However, if you move the 'fisheye-2.6' swimlane to the left of the 'master' swimlane, it will pick up all of the FishEye 2.6 commits.

This allows you to visually isolate changesets in the desired branches by reordering the swimlanes. For example, if you want to see the lineage of a branch, 'fisheye-2.6', but not 'fisheye-2.5' after both branches have previously been merged to 'master', you could arrange your swimlanes to 'fisheye-2.6', 'master', 'fisheye-2.5' from left to right. You will be able to see the 'fisheye-2.6' changesets and where the merge back to 'master' was made. The 'fisheye-2.5' changesets will just be seen as part of the 'master' branch.

Screenshots below: Example of how ordering swimlanes affects the branches that changesets are displayed on (click to view full-size images)



Permanent authentication for Git repositories over HTTP(S)

Currently, FishEye only supports HTTP or HTTPS for pushing and pulling from Git repositories. Git has no method of caching the user's credentials, so you need to re-enter them each time you perform a clone, push or pull.

Fortunately, there is a mechanism that allows you to specify which credentials to use for which server: the `.netrc` file.

Warning!

Git uses a utility called `cURL` under the covers, which respects the use of the `.netrc` file. Be aware that other applications that use `curl` to make requests to servers defined in your `.netrc` file will also now be authenticated using these credentials. Also, this method of authentication is potentially unsuitable if you are accessing your FishEye server via a proxy, as all `curl` requests that target a path on that proxy server will be authenticated using your `.netrc` credentials.

Warning!

cURL will not match the machine name in your `.netrc` if it has a username in it, so make sure you edit your `.git/config` file in the root of your clone of the repository and remove the user and '@' part from any clone url's (url fields) that look like `https://user@machine.domain.com/...` so instead they look like `http://machine.domain.com/...`

Linux or OSX

1. Create a file called `.netrc` in your home directory (`~/.netrc`). Unfortunately, the syntax requires you to store your passwords in plain text - so make sure you modify the file permissions to make it readable only to you.
2. Add credentials to the file for the server or servers you want to store credentials for, using the format below. You may use either IP addresses or hostnames, and you **do not** need to specify a port number, even if you're running FishEye on a non-standard port.

```
machine fisheye1.mycompany.com
login myusername
password mypassword
machine fisheye2.mycompany.com
login myotherusername
password myotherpassword
```

3. And that's it! Subsequent `git clone`, `git pull` and `git push` requests will be authenticated using the credentials specified in this file.

Windows

1. Create a text file called `_netrc` in your home directory (e.g. `c:\users\kannonboy_netrc`). Curl has problems resolving your home directory if it contains spaces in its path (e.g. `c:\Documents and Settings\kannonboy`). However, you can update your `%HOME%` environment variable to point to any old directory, so create your `_netrc` in a directory with no spaces in it (for example `c:\curl-auth\`) then set your `%HOME%` environment variable to point to the newly created directory.
2. Add credentials to the file for the server or servers you want to store credentials for, using the format from the **Linux or OSX** section above.

Perforce Changesets and Branches

Why does FishEye say this changeset is on more than one branch? Why does that changeset have multiple parents?

Perforce allows a single changeset to include files on multiple branches, so FishEye marks those changesets as being on all of the branches involved.

When a changeset is on multiple branches FishEye may consider it to have multiple parents from the different branches.

FishEye does not track merges in Perforce, so merges are not shown in the graph.

Changeset branches and parents are only annotations at the changeset level --- the individual file revisions will each only have a single branch and at most one parent.

What SCM systems are supported by FishEye?

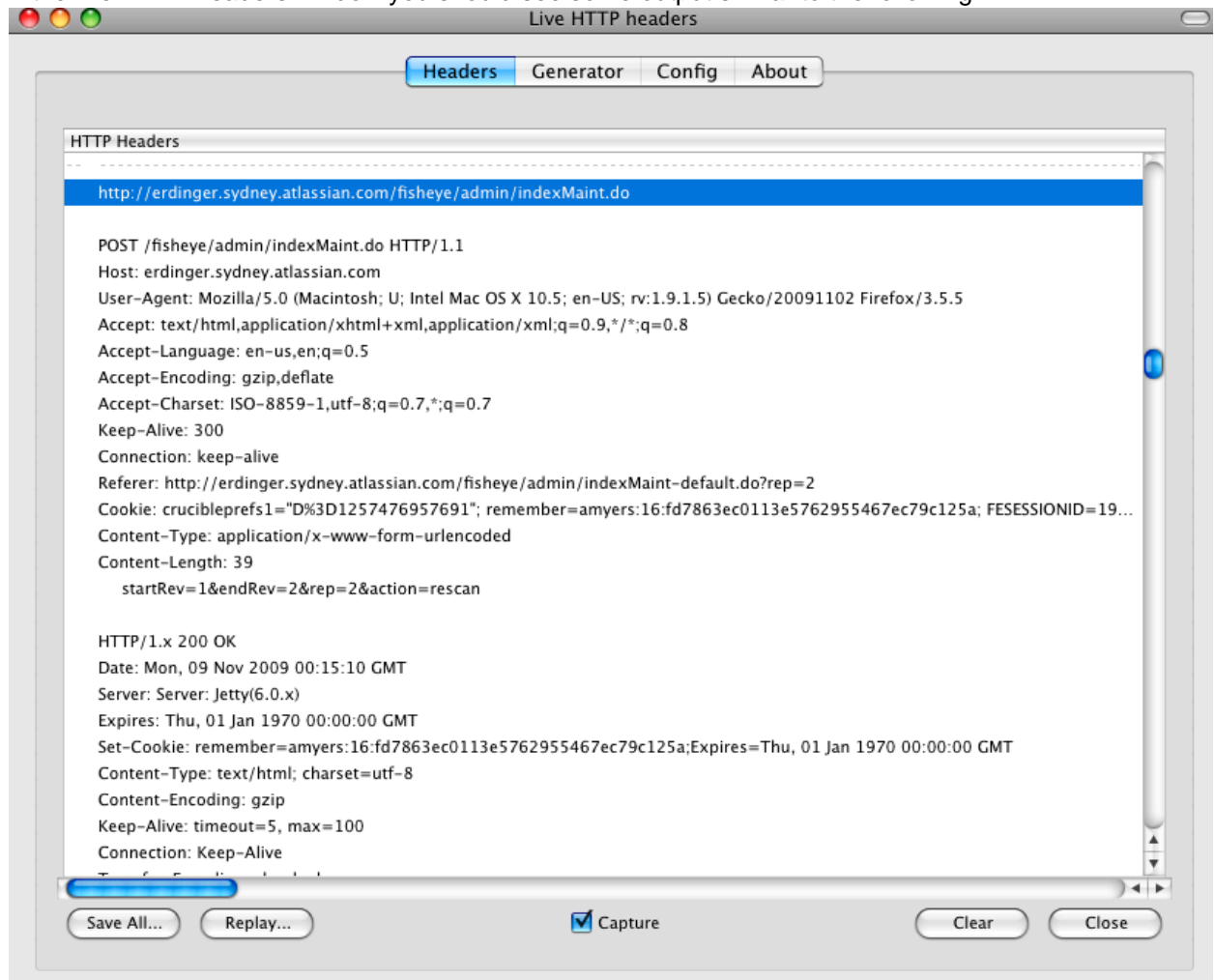
To see the list of SCM systems that is supported by FishEye, see [Supported platforms](#).

Automating Administrative Actions in Fisheye

With some command line scripting and a tool like [wget](#), and [Live HTTP Headers](#) for firefox you can automate actions. In this example, Fisheye will automatically rescan revision properties of an SVN when the commit message is updated to reference a new JIRA issue.

1. Enable live HTTP headers in firefox, then perform the action you want to perform automatically via the Fisheye Administration UI.

2. In the live HTTP headers window you should see some output similar to the following:



3. The important parts are the URL I've highlighted above (<http://erdinger.sydney.atlassian.com/fisheye/admin/indexMaint.do>) and any GET/POST parameters (`startRev=0&endRev=58&rep=2&action=rescan`)
4. Now we can construct a script with `wget` to automate this:

```
wget --keep-session-cookies --save-cookies cookie.txt
http://erdinger.sydney.atlassian.com/fisheye/admin/login.do
--post-data="origUrl=&adminPassword=admin"
wget --load-cookies cookie.txt
--post-data="startRev=0&endRev=58&rep=2&action=rescan"
http://erdinger.sydney.atlassian.com/fisheye/admin/indexMaint.do
```

With that you could generate a `post-revprop-change` hook in svn that will update the repositories automatically.

How are indexing requests handled when they are triggered via commit hook

The indexing requests are performed in order of receipt by the usual incremental indexing threads. If there is already a pending request for indexing a particular repo, a subsequent request will be ignored.

Installation & Configuration FAQ

FishEye Installation & Configuration FAQ

- [Can I deploy FishEye or Crucible as a WAR?](#) — Unfortunately FishEye and Crucible can not be deployed as a WAR http://en.wikipedia.org/wiki/WAR_%28Sun_file_format%29.
- [Does Fisheye support SSL \(HTTPS\)?](#)
- [Improve FishEye scan performance](#) — Background information
- [Migrating FishEye Between Servers](#) — This page describes the process for migrating FishEye between servers.
- [Setting up a CVS mirror with rsync](#) — In situations where running FishEye on the same server as your CVS repository is not practical or possible, you can use the Linux utility 'rsync' to mirror the CVS repository contents onto the FishEye server.
- [What are the FishEye System Requirements?](#)
- [How to reset the Administration Page password in FishEye or Crucible](#)
- [How Do I Configure an Outbound Proxy Server for FishEye](#)
- [How to remove Crucible from FishEye 2.x or later](#)
- [How to run Fisheye or Crucible on startup on Mac OS X](#)

Can I deploy FishEye or Crucible as a WAR?

Unfortunately FishEye and Crucible can not be deployed as a [WAR](#). FishEye has some special needs for performance reasons that are not easily supported on third-party containers. While this is an often requested feature, there are no immediate plans to provide a WAR version of FishEye or FishEye+Crucible. However the upcoming separate edition of Crucible (i.e. without FishEye) may at some stage be available as a WAR.

Does Fisheye support SSL (HTTPS)?

FishEye has built-in SSL support from FishEye 2.4 onwards. Read [FishEye SSL configuration](#) for more information.

Improve FishEye scan performance

This page contains information about improving the performance of FishEye repository scans.

Background information

When you add a repository, FishEye needs to perform a once-off scan through the repository to build up its initial index and cache. This scan can take some time. Until this scan is complete, you may find that some data is not displayed. As a guide, FishEye should be able to process about 100KB-200KB per second on an averaged-size PC. If FishEye is accessing the repository over the network (e.g. over a NFS mount), then you should expect the initial scan to take longer.

General improvements

You can increase the speed of your scans using the following options:

- If your repository is non-local, set up a local repository mirror on the FishEye server. This will provide a major speed boost for anyone scanning a repository across a network.
- [Exclude](#) unused file types, unused directories and specific large files from FishEye.

Improve update performance during initial scan

One option is break large repositories into multiple smaller repositories. While this technique will not improve the overall initial scan time, it allows for all fully scanned repositories to be updated while the initial scan is still being performed on those remaining.

In FishEye 1.3.4 and later, the initial and incremental scans happen in separate, single threads. So, splitting the repositories will allow incremental scans to run concurrently alongside the initial scans. You may also wish to split projects into separate repositories, since permissions in FishEye are applied on a per-repository basis.

Improving initial scan performance for an SVN repository

svnsync is not an atomic protocol. It generally performs the commit as the svn sync user and then updates the revprops to match the original commit, under some circumstances, this can cause the

wrong user to be attributed as the author of a changeset. More information available at [Wrong User Reported When Indexing SVN Mirror Repository](#).

The http/s protocol has the slowest performance during the initial scan. The svn protocol (svn://) is faster and the file protocol (file://) is the fastest.

Therefore if you find your initial scan takes an extended amount of time (more than a day or two), you should consider switching over from the http/s protocol to the svn or file protocol to define the location of your SVN repository. (Use [svnsync](#) to mirror the repository onto the fisheye server, so that you can access it with the file protocol.)

```
E.g. Switch from
https://example.com/svn/project/
to
svn://example.com/svn/project/
or
file:///home/user/some/location/svn/project
```

In order for SVN protocol to work you need to have set up an [svnserve based server](#).

More information on how to troubleshoot SVN indexing related issues can be found [here](#).

Performance support

If you have implemented at least one of the suggestions above but are still experiencing slow performance, ask us for help:

1. First read the [Tuning FishEye performance](#) document.
2. Turn on debug logging using the [command line debug flag](#).
3. Allow FishEye to continue its initial scan overnight.
4. Create a new [support request](#) in the FishEye project, including your server environment and log files with the problem description.

Migrating FishEye Between Servers

This page describes the process for migrating FishEye between servers.

If you have defined the `FISHEYE_INST` [environment variable](#), then upgrades and migrations of your FishEye instance will be relatively simpler.

If you have defined `FISHEYE_INST`

1. Shut down your current FishEye server completely.
2. Copy the `FISHEYE_INST` directory to your destination server.
3. Copy and set up all of your [environment variables](#) from your source server to your destination server (remembering to set up your `FISHEYE_INST` directory to point to the location where you copied the data to in Step 2).
4. Install FishEye on your destination server.
5. Read the note about [external databases](#) below.
6. Start FishEye. It should pick up your environment variables, and from that access your `FISHEYE_INST` directory, which contains your configuration.

If you have not defined `FISHEYE_INST` but would like to set it up

- 1) Shut down your current FishEye server completely.
- 2) Copy the following three items into to a new folder on your destination server (for example, `fisheye_inst`):

- `<FishEye installation directory>/config.xml`

- <FishEye installation directory>/var
- <FishEye installation directory>/cache

3) Copy and set up all of your [environment variables](#) from your source server to your destination server. In addition to this, set up the FISHEYE_INST environment variable as follows, replacing the /path/to/fish_eye_inst with the fully qualified path to the fisheye_inst folder you set up in Step 2:

```
export FISHEYE_INST=/path/to/fisheye_inst
```

4) Install FishEye on your destination server.

5) Read the note about [external databases](#) below.

6) Start FishEye. It should pick up your environment variables, and from that access your FISHEYE_INST directory, which contains your configuration. If you are using [JIRA for User Management](#) you might need to update the whitelist of the corresponding application connector in JIRA with the IP address of the new FishEye server to retain user authentication.

External Databases

If you are using an external database it must be accessible from the new server since the steps above only include installing a new application and relocating your configuration data and repository indexes.

If you wish to migrate both the application, data, *and* your external database to a different server (e.g. to create a development or test server) then you will need to make a copy of your external database and update the config.xml file (depending on the setup, located in either <FishEye/Crucible installation directory>/config.xml or FISHEYE_INST/config.xml) with the new database location **prior** to starting up the instance. See [Backing up and restoring FishEye data](#) for more information.

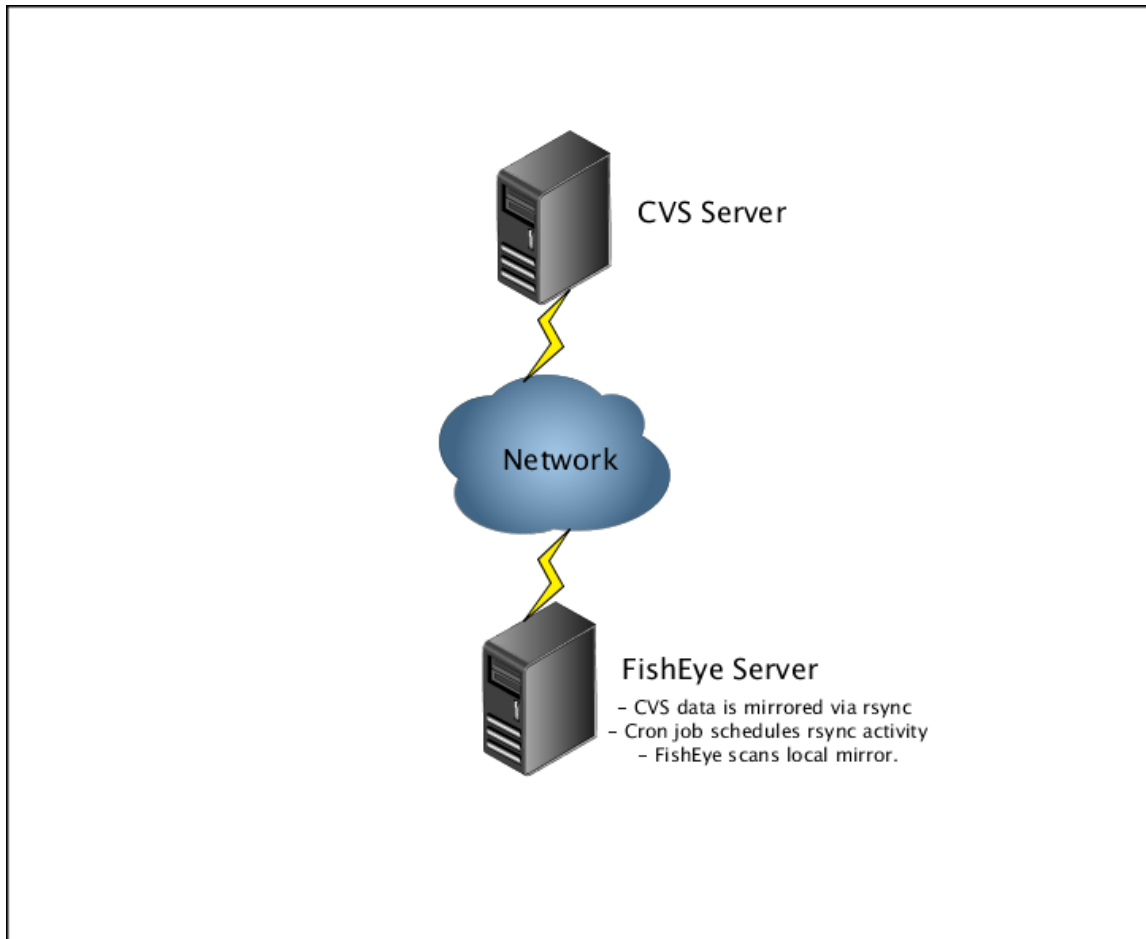
Setting up a CVS mirror with rsync

In situations where running FishEye on the same server as your CVS repository is not practical or possible, you can use the Linux utility 'rsync' to mirror the CVS repository contents onto the FishEye server. This is possible because CVS data is stored in a reasonably simple form in the file system.

We recommend this to achieve best performance when FishEye and CVS cannot be hosted on the same machine.

i This workaround requires the ability to SSH into both machines. Linux and Mac OS X operating systems have rsync built in. For Windows, you will need to [install rsync](#).

Diagram: A scenario where rsync is required



To set up a CVS mirror with rsync:

1. You will need to set up a local directory on the FishEye server for the mirrored CVS content, ensuring that this server has ample disk space to store the current CVS database and any future space requirements.
2. We will refer to the CVS instance (on your CVS Server,) as <CVS_HOME> and the new 'mirror directory' (on your FishEye server) as <MIRROR_HOME>.
3. Type the following `rsync` command on the console at the command-line of the FishEye server:

```
rsync --backup <CVS_HOME> <MIRROR_HOME>
```

A real-world example would look something like this:

```
rsync --backup user@cvs_server:/CVS_server/path/to/instance  
/datastore/FishEye/cvs-mirror
```

4. Schedule the rsync command to run regularly with a [cron](#) job. Running hourly is a good default interval. Under Windows, use a native task scheduler.
5. With the cron job active, you will have established rsync to run an hourly comparison of the two directories and copy any changes across to the mirror directory as they occur. Note that running the rsync process will impact the FishEye server's performance (and also the CVS server's) to a certain degree.
6. In the FishEye admin interface, add the local 'mirror directory' as a new CVS repository and run the initial scan. As this is local data on the same file system, FishEye's scanning of this data will be optimal.
7. Adjust the [FishEye Updater](#) Full Scan period to one hour (the default is 15 minutes).
8. The rsync configuration is now complete. Monitor the disk space on both servers to ensure there is adequate headroom for the mirroring process.

For more information on the syntax for rsync, visit the [rsync home page](#).

What are the FishEye System Requirements?

Visit the FishEye [Supported platforms](#).

How to reset the Administration Page password in FishEye or Crucible

If you have forgotten or misplaced the password for the Admin page (`http://<FECRU_URL>:<FECRU_PORT>/admin/admin.do`), you can reset it manually.

To manually reset the admin password, please edit your `FISHEYE_INST/config.xml` file (make a backup as well before editing).

You will see something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<config control-bind="127.0.0.1:8059" version="1.0"
admin-hash="352353256326369233A801FC3">
```

To reset the password to **"admin"**, please change the `admin-hash` value so that it appears as `admin-hash="21232F297A57A5A743894A0E4A801FC3"`

```
<?xml version="1.0" encoding="UTF-8"?>
<config control-bind="127.0.0.1:8059" version="1.0"
admin-hash="21232F297A57A5A743894A0E4A801FC3">
```

Restart Fisheye for this to take effect. You should now be able to access the FishEye Admin page (`http://<FECRU_URL>:<FECRU_PORT>/admin/admin.do`) with the password **"admin"**. Please change this password immediately from the Admin area: click **Change Admin password** (under "Security Settings").

See also [Best practices for FishEye configuration](#).

How Do I Configure an Outbound Proxy Server for FishEye

The Java Virtual Machine provides support for outbound proxy servers. To take advantage of this some additional parameters need to be passed to the JVM via the `FISHEYE_OPTS` environment variable:

```
export FISHEYE_OPTS="-Dhttp.proxyHost=proxy.example.org
-Dhttp.proxyPort=8080 -Dhttp.nonProxyHosts=*.foo.com|localhost
-Dhttp.proxyUser=username -Dhttp.proxyPassword=password"
```

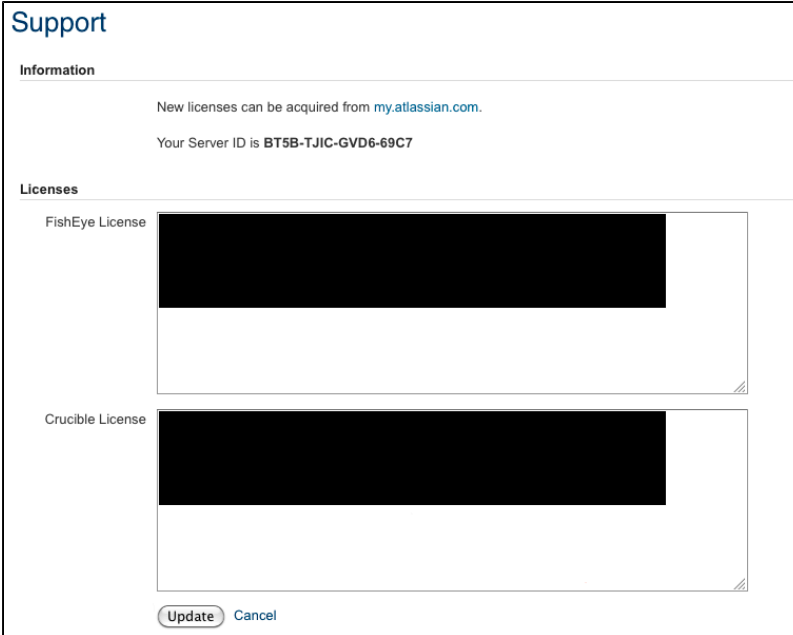
See [Environment variables](#) for instructions on how to set these parameters.

How to remove Crucible from FishEye 2.x or later

Use the same process to remove FishEye and keep Crucible; just remove the FishEye license instead.
Want to know [What happens if I decide to stop using FishEye with Crucible?](#)

Remove Crucible from a FishEye installation

1. Go to the Admin area (choose **Administration** from your user menu) and click **System Info** (under "System Settings").
2. Click **Edit License** (in the "License" section).
3. Remove the Crucible license from this screen:



The screenshot shows a 'Support' dialog box. Under the 'Information' tab, it states: 'New licenses can be acquired from my.atlassian.com. Your Server ID is BT5B-TJIC-GVD6-69C7'. Under the 'Licenses' tab, there are two sections: 'FishEye License' and 'Crucible License'. Each section contains a large black rectangular redaction box. At the bottom of the dialog, there are two buttons: 'Update' and 'Cancel'.

4. Click **Update** to save. Crucible will now be disabled and you will no longer see any reference to it in the application.

How to run Fisheye or Crucible on startup on Mac OS X

This article is only provided as a guide and has only been tested on Mac OS X 10.5

launchd does not provide support for service dependencies so if you are using an external database, this may not work for you. Fisheye assumes the database is available when it starts, and the startup scripts will not wait for the database to become available.

You need to create a .plist file to create items that will start at boot time. Please refer to <http://developer.apple.com/MacOsX/launchd.html> for details.

Here is an example .plist that should work for Fisheye/Crucible:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.atlassian.fisheye</string>
  <key>ProgramArguments</key>
  <array>
    <string>/path/to/<FishEye install directory>/bin/run.sh</string>
  </array>
  <key>OnDemand</key>
  <false/>
  <key>RunAtLoad</key>
  <true/>
  <key>StandardOutPath</key>
  <string>/path/to/FISHEYE_INST/var/log/fisheye.out</string>
  <key>StandardErrorPath</key>
  <string>/path/to/FISHEYE_INST/var/log/fisheye.out</string>
  <key>EnvironmentVariables</key>
  <dict>
    <key>FISHEYE_INST</key>  <string>/path/to/FISHEYE_INST/</string>
    <key>FISHEYE_OPTS</key>  <string>-Xms512m -Xmx1024m
-XX:MaxPermSize=128m</string>
  </dict>
</dict>
</plist>

```

Customize the `/path/to/FISHEYE_INST/` and `/path/to/<FishEye install directory>/` with the `FISHEYE_INST` and `FishEye` install directories respectively and make any required modifications to `FISHEYE_OPTS`. Save the file as `com.atlassian.fisheye.plist` and then try to load it with:

```

[amyers@erdinger:fecru-2.1.3]$ launchctl load
com.atlassian.fisheye.plist
[amyers@erdinger:fecru-2.1.3]$ launchctl start com.atlassian.fisheye

```

FishEye should now start up and you will be able to access it in your web browser.

Licensing FAQ

FishEye Licensing FAQ

- [Are anonymous users counted towards FishEye's license limits?](#) — Users accessing FishEye anonymously are, for all intents and purposes, unlimited users.
- [Restrictions on FishEye Starter Licenses](#)
- [Updating your FishEye license](#)
- [Git or Hg Repository exceeds number of allowed committers](#)

Are anonymous users counted towards FishEye's license limits?

The short answer is no. If you are using FishEye in your organization but most users require only anonymous access (that is, you have not set access restrictions on the content in your repositories), then an unlimited number of anonymous users can be accommodated regardless of the FishEye license you are using.

Users accessing FishEye anonymously are, for all intents and purposes, *unlimited* users.

However, if your users require permissions and controlled access to specific content in your repositories, then they will need to log in to FishEye. Hence, these users will need to create accounts and will be factored into the license limit.

Restrictions on FishEye Starter Licenses

This page explains the limitations of the FishEye Starter license and provides general information about using this license in production.

On this page:

- [What is a Starter License?](#)
- [What are the Starter License restrictions?](#)
- [What happens if I exceed the Starter License limits?](#)
 - [Evaluate](#)
 - [Upgrade](#)
 - [Reconfigure your repository](#)
- [Frequently Asked Questions](#)

What is a Starter License?

Starter licenses are low-cost licenses that allow small teams to make use of Atlassian products (see [more information](#)). FishEye Starter licenses were introduced with the release of FishEye 2.0.5 (October 2009).

What are the Starter License restrictions?

FishEye Starter Licenses are restricted to no more than:

- 5 FishEye users
- 5 repositories
- 10 historic [committers](#) per [repository](#)

See [FishEye Licensing and Purchasing](#) for more details.

Note that all other FishEye licenses support unlimited repositories and unlimited committers per repository.

What happens if I exceed the Starter License limits?

If you have more than 5 user accounts in FishEye, your users will not be able to log in to FishEye until an administrator reduces the number of user accounts.

If you have more than five repositories, FishEye will prevent more than five repositories from being enabled at any given time. Administrators can control which five repositories are enabled.

If you exceed more than ten [committers](#) in a [repository](#), the following warning will appear at the top of pages for the entire system:

NOTE: This repository, <repository-name> has more than 10 committers which exceeds the limits for your Starter license. Indexing has stopped. To fix this, you can [Evaluate](#), [Upgrade](#) or [Reconfigure your repository](#).

The links in this warning will lead you to the following solutions:

Evaluate

30-day evaluation licenses are available to allow you to [try out FishEye](#) and other Atlassian products. You can select a license that allows more users than your current license.

Upgrade

You can upgrade your license at any time (via the [Atlassian online store](#)), which will remove the [committer](#) and [repository](#) limits which apply to the Starter License. Please ensure to restart your repository, after the license upgrade, to ensure the changes are picked up for the new committer limit.

Reconfigure your repository

This option lets you configure your repository to remain within the limits of the Starter License. You can take the following actions to reduce the scope of FishEye's indexing:

1. **Change the repository definition to look at a subset of your repository**
Typically this involves setting the path within your repository that you wish FishEye to index. [Read more.](#)
2. **Exclude parts of the repository**
You can exclude portions of your repository that you are not interested in. [Committers](#) that are active in only these areas will not appear in FishEye and not be included in the committer count. [Read more.](#)
3. **Set a starting point**
Some of the FishEye SCM integrations allow you to configure a starting revision from which to start indexing. All [commits](#) prior to this starting point are not included in FishEye and do not contribute to the committer count. [Read more.](#)
4. **Map Committers**
If your developers have not correctly configured their committer names for Git or Mercurial, they may have committed with multiple identities. It is then possible to remap these to correct the problem. See the knowledge base article, [Git or Hg Repository exceeds number of allowed Committers](#)

Once you have reconfigured your repository, you will need to re-index the repository, allowing you to remain under the limits of the Starter license.

Frequently Asked Questions

Question	Answer
What happens when the 11th unique committer is encountered during indexing?	For all SCMs other than CVS FishEye will index all revisions up to but not including the revision that introduced the 11th committer. Since CVS is indexing is file-by-file based, FishEye will index files until it reaches the committer limit. Remaining files in the repository are not indexed. This means only files which have been indexed will be displayed in changesets and changesets may be incomplete.
What happens when a FishEye instance with a Starter license is started, using existing indexes with more than five repositories?	FishEye will only start indexing on the first five repositories. An administrator can use admin UI to adjust which repositories are enabled and hence control the five repositories that are started. FishEye should then be restarted.
What happens when a FishEye instance with a Starter license is started, using existing indexes with one or more repositories with more than ten committers?	FishEye will display all information currently indexed but for each repository that has reached the ten committer limit, no further revisions will be indexed.
What happens on upgrade from a Starter license, if indexing has been paused due to the committer limit being reached?	On restart of FishEye, indexing will resume for all repositories. Each repository can be restarted individually to avoid restarting FishEye. Due to the nature of CVS indexing, we recommend reindexing any CVS repositories which have reached the committer limit prior to the license upgrade.
What happens when upgrading from a Starter license, when repositories have not started due to the repository limit being reached?	On restart of FishEye, all enabled repositories will start. Each repository can be restarted individually to avoid restarting FishEye.

What happens if my evaluation license has expired and I upgrade to a Starter license, however I have exceeded the Starter license limitations?	As described above, a maximum of five repositories will start and for any repository with more than 10 committers, no further indexing will occur. All existing indexed content is retained and can be viewed.
What happens when downgrading to a Starter license, where the repository limit has been exceeded?	A maximum of five of your configured repositories will start running. The remainder will not start but will continue to be available.
What happens when downgrading to a Starter license, where the committer limit has been exceeded for one or more repositories?	No further indexing will occur for the repositories where the committer limit has been exceeded.

Updating your FishEye license

When you upgrade or renew your FishEye license, you will receive a new license key – you'll need to update your FishEye server with the new license.

Note that you can access your current license, or obtain a new license, by going to my.atlassian.com.

Related pages:

- [Upgrading from FishEye to Crucible](#)

To update your FishEye license key:

1. Log in to FishEye [Admin area](#).
2. Click **System Info** (under 'System Settings').
3. Click **Edit License** and paste your new license key into the appropriate text box.
4. Paste your new license into this box. Obtain a new license by clicking my.atlassian.com in the 'Information' section.
5. Click **Update**.

Support

Information

New licenses can be acquired from my.atlassian.com.

Your Server ID is **BMRI-3AS3-HMMD-PGXX**

Licenses

FishEye License

Evaluation

Crucible License

AAAA9g0ODAoPeNptkE9rwzAMxe/+FladPWa7hzXgQ5eELRAnpcmgH15Uo62G1A3+E9Zvv3Yt
dBS9C
IT03k8PPfS7RJvDRAWnfJ7NeMZnNO96Kp64JAUG4+0Y7cGp3CdjtwNuMlpOMCQ4T0nu8acpl
Kl6e
xjnTAhirurH2hp0AfvjiA3siOnAGy6/R+uMvqPgDbdJ+i779eA/og
2KctP4TnA0X2CIEIIFdzOAiXbCy5kO/YS+KtSLXIVMLjrJ3rQu2PJ1vSZd2ahTsVqKZzGXklzz
n
+R1Vdzb3l+6TN7sIOD/v3wDPH51STAtAhRYkbBDq2j1NpXxLp5ws8aCihAgBwIVAJceVTPzGB
f4w
g5FWvNY7c5adD+EX02co

Update

Cancel

Git or Hg Repository exceeds number of allowed committers

Symptoms

Adding a Git repository to FishEye with a Starter license (10 users) installed will sometimes produce an error stating that there are more than 10 committers, even though there are less than 10 real people who have committed:

```
Repository index failed due to error - class
com.atlassian.fisheye.dvcs.handler.DvcsProcessRuntimeException:
com.cenqua.fisheye.LicensePolicyException: Exceeded number of allowed
committers
```

Cause

You have exceeded the number of committers allowed by the Starter License. This is either a legitimate occurrence (you have more than 10 actual committers) or this problem can also affect git and Mercurial Starter License users over time when their committers use different ids to commit to the repository.

Due to this change ([FE-2496 - User Mapping across multiple repositories needs to be easier](#) **CLOSED**) in FishEye 2.6, when we scan a git repository, we now combine the author and email fields from the commit into a single field that FishEye uses to determine who the committer is. This allows automatic email mapping to map the commit by that committer to a FishEye User with the same email address.

Unfortunately, if your committers use different email addresses for their commits, they will now count as more than one committer, where previously they did not.

Resolution

For Mercurial, the only solution is to rewrite the history of your repository to map these committers to a single consistent committer id for each real committer you have (and to encourage them to be consistent in future).

This means getting all developers to delete their clones of the repository and re-clone once the history re-write is complete. Otherwise, they will probably end up pushing the old commits back into the repository being scanned by FishEye and the problem will reoccur.

Warning

Be very sure you know what you are doing during this process: you are re-writing history and the wrong command could lose all of your repositories history and your source code. Keep a backup clone of your repository in case something goes wrong.

Rewriting history is also valid for git (detailed below), but git has an easier alternative, using the `.mailmap` file. This allows the git repository to report to FishEye different names and email addresses than were actually used by the author - this will not affect any other repositories, unless they also get the `.mailmap` file in their local repositories. See the [git shortlog](#) man page for more details.

FishEye will use the version of the `.mailmap` file that is checked in at `HEAD`. This is normally the latest commit on `master` (or your default branch, if you have set it to other than `master`).

Once you set this up, you will need to go to the administration console in FishEye for your repository and under the 'Maintenance' tab, select 'Re-clone and Re-index'. Any changes you make to the `.mailmap` file will not take affect until you 'Re-clone and Re-index'.

Note that if you want to map multiple email addresses to one author, you need to specify them on multiple lines. E.g.

```
John Doe <john.doe@newcompany.com> <john@oldcompany.com>
John Doe <john.doe@newcompany.com> <johndoe@oldcompany.com>
John Doe <john.doe@newcompany.com> <jdoe@oldcompany.com>
```

Mercurial

In Mercurial, we use the [Convert Extension](#) to rewrite the committer names, using an `authormap` file. In the repository that FishEye is scanning, you will need to use the [Mercurial Queues Strip command](#) to remove all the current commits once the converted repository is ready, otherwise you will simply add a new set of commits next to the originals.

1. Get a copy of the repository and use:

```
hg log --template "{author}\n" | sort | uniq
```

to list the current committer names.

2. Identify those you want to rewrite and add an entry into a [authormap file](#) to rewrite them to another address.
3. Run:

```
hg convert my-clone rewritten-clone
```

4. [Strip](#) the repository that FishEye is indexing and push the `rewritten-clone` into it. Make sure all your developers use clones of the new copy of the repository from now on, otherwise they will possibly push back all the old commits and the problem will reoccur).
5. Go to the administration console in FishEye for your repository and under the '**Maintenance**' tab, select '**R-e-clone and Re-index**'.

Git

This method will rewrite your commits with undesired author/emails in them, changing history. An easier approach (see above) is to use the .mailmap file as mentioned in the [git shortlog](#) man page. You will need to reclone and re-index after setting that up.

FishEye uses the git author name and email fields on a commit to identify who made the changes (which in FishEye is referred to as the "committer", but in git is the author; in git, the committer is who committed, possibly on behalf of the author), so we need to remap them.

1. To rewrite the committer names in git, use the [git filter-branch](#). You need to create a script to check for each committer you want to remap, and replace the name and email fields. The command needs to be run for each branch you have that you wish to modify:

```
git filter-branch --commit-filter '
    if [ "$GIT_AUTHOR_NAME" = "<Old Name>" ];
    then
        GIT_AUTHOR_NAME="<New Name>";
        GIT_AUTHOR_EMAIL="<New Email>";
    elif [ "$GIT_AUTHOR_NAME" = "<Other Old Name>" ];
    then
        GIT_AUTHOR_NAME="<Other New Name>";
        GIT_AUTHOR_EMAIL="<Other New Email>";
    fi;
git commit-tree "$@" Branch
```

Replace Branch with each branch you need to modify.

Note the warnings on the [git filter-branch](#) page.

2. Once the process is complete and your repository only contains the rewritten author names and email addresses, go to the administration console in FishEye for your repository and under the '**Maintenance**' tab, select '**Re-clone and Re-index**'.

Example EyeQL Queries

EyeQL

- How do I find changes made to a branch after a given tag?
- How do I filter results?
- How do I find changes between two versions, showing separate histories?
- How do I find changes made between two version numbers?
- How do I find commits without comments?
- How do I find files on a branch, excluding deleted files?
- How do I find files removed from a given branch?
- How do I find revisions made by one author between versions?
- How do I select the most recent revisions in a given branch?
- How do I show all changesets which do not have reviews?

For more information on using EyeQL, see the [Reference guide](#).

How do I find changes made to a branch after a given tag?

Find changes made to Ant 1.5.x after 1.5 FINAL:

```
select revisions where on branch ANT_15_BRANCH and after tag
ANT_MAIN_15FINAL group by changeset
```

How do I filter results?

This query, finds files removed on the Ant 1.5 branch, but just returns the person and time the files were deleted:

```
select revisions where modified on branch ANT_15_BRANCH and is dead  
return path, author, date
```

How do I find changes between two versions, showing separate histories?

As above, but show the history of each file separately:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_151_FINAL)  
group by file
```

How do I find changes made between two version numbers?

Find changes made between Ant 1.5 and 1.5.1:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_151_FINAL)  
group by changeset
```

How do I find commits without comments?

Using the Advanced Search and EyeQL you can find commits that do not have comments using the following query:

```
select revisions from dir / where comment = "" group by changeset
```

How do I find files on a branch, excluding deleted files?

Find files on branch and exclude delete files:

```
select revisions where modified on branch ANT_15_BRANCH and not is  
deleted group by changeset
```

How do I find files removed from a given branch?

Find files removed on the Ant 1.5 branch:

```
select revisions where modified on branch ANT_15_BRANCH and is dead  
group by changeset
```

How do I find revisions made by one author between versions?

Find changes made by conor to Ant 1.5.x since 1.5.0:

```
select revisions where between tags (ANT_MAIN_15FINAL, ANT_154] and  
author = conor group by changeset
```

How do I select the most recent revisions in a given branch?

Find Java files that are tagged ANT_151_FINAL and are head on the ANT_15_BRANCH: (i.e. files that haven't changed in 1.5.x since 1.5.1)

```
select revisions from dir /src/main where is head and tagged  
ANT_151_FINAL and on branch ANT_15_BRANCH and path like *.java group by  
changeset
```

How do I show all changesets which do not have reviews?

The following query will return any changesets that have not been reviewed.

```
select revisions where (not in any review)
```

Integration FAQ

The following FAQs relate to integrating FishEye with other Atlassian applications. You may also wish to refer to [Troubleshooting JIRA FishEye Plugin](#) in the FishEye Knowledge Base.

- [How do I disable the Source \(FishEye\) tab panel for non-code projects?](#)
- [How do I enable debug logging for the JIRA FishEye plugin?](#)
- [How do I uninstall the JIRA FishEye plugin?](#)
- [How is the Reviews \(Crucible\) tab panel for the JIRA FishEye Plugin populated?](#)
- [What do I do if I discover a bug with the JIRA FishEye plugin?](#)

How do I disable the Source (FishEye) tab panel for non-code projects?

By removing all users and groups from the 'View Development Tools' permission (called the 'View Issue Source Tab' permission prior to JIRA 6.1) in your project's permission scheme, the Source and Reviews tabs will be effectively disabled for that project.

How do I enable debug logging for the JIRA FishEye plugin?

For Plugin versions prior to 3.0

You can enable DEBUG logging for the JIRA FishEye plugin temporarily (until JIRA is shutdown) by accessing the following URL in a web browser when logged in as an administrator:

```
http://YOUR-JIRA-BASE-URL/secure/admin/ConfigureLogging.jspa?loggerName=  
com.atlassian.jira.ext.fisheye&levelName=DEBUG
```

To enable debug logging across JIRA restarts, update the **CLASS-SPECIFIC LOGGING LEVELS** section of your `WEB-INF/classes/log4j.properties` file by inserting the following:


```
log4j.category.com.atlassian.jira.ext.fisheye = DEBUG, console, filelog
log4j.additivity.com.atlassian.jira.ext.fisheye = false
```

and restarting JIRA.

For Plugin versions 3.0+

You can enable DEBUG logging for the JIRA FishEye plugin temporarily (until JIRA is shutdown) by accessing the following URL in a web browser when logged in as an administrator:

```
http://YOUR-JIRA-BASE-URL/secure/admin/ConfigureLogging.jsps?loggerName=
com.atlassian.jirafisheyeplugin&levelName=DEBUG
```

To enable debug logging across JIRA restarts, update the **CLASS-SPECIFIC LOGGING LEVELS** section of your `WEB-INF/classes/log4j.properties` file by inserting the following:

```
log4j.category.com.atlassian.jirafisheyeplugin = DEBUG, console, filelog
log4j.additivity.com.atlassian.jirafisheyeplugin = false
```

and restarting JIRA.

How do I uninstall the JIRA FishEye plugin?

Plugin versions prior to 3.0

To uninstall, simply remove the `jira-fisheye-plugin-*.jar` from your `WEB-INF/lib` directory.

Plugin version 3.0+

To uninstall, simply remove any `jira-fisheye-plugin-*.jar` jars from your `WEB-INF/classes/com/atlassian/jira/plugin/atlassian-bundled-plugins.zip` and `%JIRA_HOME%/plugins/installed-plugins` directory.

How is the Reviews (Crucible) tab panel for the JIRA FishEye Plugin populated?

The Crucible 'Reviews' tab is populated differently depending on the version of the FishEye JIRA plugin you are using:

- **Prior to Version 2.3** of the plugin, a review is displayed on the Crucible tab panel if (and only if) it contains a revision that is displayed on the FishEye tab panel. Having the issue key in the description *will not* automatically link it to the issue.
- In **Version 2.3+**, you can configure the Crucible tab panel to use the method described above and/or *search for issue keys in the title/description of reviews*.

If you associate reviews with JIRA issues from the Crucible user interface, they will not necessarily show up in the issue tab panel. We are working on this issue (see [FISH-316](#)), but in the meantime, a workaround is to include the related issue key in the title of the review.

What do I do if I discover a bug with the JIRA FishEye plugin?

If you need assistance or think you've encountered a bug with the JIRA FishEye plugin, please raise an issue in the [JIRA FishEye Plugin project](#).

Subversion FAQ

FishEye Subversion FAQ

- [Configuring Start Revision based on date](#) — For Subversion repositories Fisheye has the ability to configure a Start Revision parameter to allow you to only index content from a given point in your repository.
- [Errors 'SEVERE assert' or 'Checksum mismatch'](#) — SVNKit may have problems with older version Subversion servers - versions 1.1.x and prior.
- [FishEye fails to connect to the Subversion repository after a short time of successful operation.](#) — On Unix systems, the svn:// protocol is usually handled by inetd or xinetd. These daemons apply, by default, a connection per second limit to incoming connections. Any connections above this rate are rejected by the server.
- [How can FishEye help with merging of branches in Subversion?](#) — In merge management, the main advantages of FishEye come from its search functionality. If you record the revisions merged when you check in a merge result, you can find this information in FishEye easily for the next merge operation.
- [Subversion Changeset Parents and Branches](#)
- [SVN Authentication Issues](#) — If multiple repositories have been defined in FishEye for the same SVN Server and those repositories use different credentials, FishEye may not use the correct credentials.
- [What are Subversion root and tag branches?](#)
- [Why do I need to describe the branch and tag structure for Subversion repositories?](#) — In Subversion, branches and tags are defined by convention, based on their path within a repository, and not directly defined by the repository. A few different layout alternatives are commonly used. It is also possible that you are using your own custom layout. As a result you need to describe to FishEye which paths in your repository are used as branches and tags.
- [Why don't all my tags show up in FishEye?](#)
- [About merges in Subversion](#)

Configuring Start Revision based on date

For Subversion repositories Fisheye has the ability to configure a **Start Revision** parameter to allow you to only index content from a given point in your repository. Quite often users will find it helpful to index from a revision on a given date. For example, you may want to only index SVN data in the past year. To determine the revision based on date, you can use the following command:

```
svn log -r {YYYY-MM-DD}:HEAD <SVN_URL> -l 1
```

The output of this command will reveal the revision number closest to the date that you provide.

Errors 'SEVERE assert' or 'Checksum mismatch'

When using SVNKit, you may see errors in the FishEye log such as 'SEVERE: assert #B' or 'Checksum mismatch'.

SVNKit may have problems with older version Subversion servers - versions 1.1.x and prior. If this is the case you should either use the native JavaHL layer or upgrade your Subversion server to a more recent version.

FishEye fails to connect to the Subversion repository after a short time of successful operation.

If you use the `svn://` protocol to access a Subversion repository, you may notice that FishEye fails to connect to the repository after a short time of successful operation.

On Unix systems, the `svn://` protocol is usually handled by `inetd` or `xinetd`. These daemons apply, by default, a connection per second limit to incoming connections. Any connections above this rate are rejected by the server.

Two options for fixing this problem:

- Ask your system administrator increase the connection rate allowed for the svn connection by updating the `xinetd` configuration, or
- Specify a connection per second limit in your FishEye repository definition, to prevent FishEye from exceeding the `xinetd` limits.

How can FishEye help with merging of branches in Subversion?

FishEye gives you a logical view of your branched files so you can see activity on a single file across multiple branches/trunk.

In merge management, the main advantages of FishEye come from its search functionality. If you record the revisions merged when you check in a merge result, you can find this information in FishEye easily for the next merge operation.

As an example, let's say you have a branch `dev` created at revision 1300 from `trunk`. Development has proceeded on both `trunk` and `dev`. At some point you wish to add the latest `trunk` changes into the `dev` branch. Let's say that is at revision 1400. When you check in the results of this merge, you would use some standard format checkin comment such as:

```
merge from trunk to dev 1300:1400
```

When you come to do the next merge, say at revision 1500, you can use FishEye search to find this checkin comment and know what the starting point for the merge should be. You can then check this in as:

```
merge from trunk to dev 1400:1500
```

Merges back to `trunk` from the `dev` branch are managed in the same way.

Subversion Changeset Parents and Branches

Why do some changesets have more than one branch? Why do these changesets have more than one parent?

In Subversion, a single changeset can have files and directories that are on different branches, as defined by the [SVN tag and branch structure](#). In this situation, the changeset is considered to be on all of the branches of its constituent file revisions. If a changeset is on more than one branch it can have a parent changeset of each of its branches, giving the changeset multiple parents.

FishEye does not track SVN merges, so merges are not indicated on the graph.

When I create a complex branch, how does FishEye determine which is its parent changeset? When I create a complex tag, how does FishEye decide which changeset to tag?

In Subversion, a *simple* branch or tag is created by copying a source directory, e.g. copying `/trunk` to `/branches/branch1` or `/tags/tag1`. The tag or branch is considered *complex* if a part of the copied directory is replaced with another version, e.g. `/trunk` is copied to `/tags/tag1`, and then `branches/branch1/dir1` is copied to `/tags/tag1/dir1`.

For the purpose of the [commit graph](#), FishEye looks at where the root directory was copied from, to determine where the branch or tag originated. In the example above, the label `tag1` would be applied to the latest changeset on trunk when the tag was created, even though part of the tag was copied from branch1. This only affects the annotation of the changeset, not the file revisions that are tagged — the tagged file revisions are still those on `trunk` or `branch1` as appropriate.

SVN Authentication Issues

If multiple repositories have been defined in FishEye for the same SVN Server and those repositories use different credentials, FishEye may not use the correct credentials.

FishEye does not control directly when authentication information is used to access Subversion repositories. It delegates this operation to the JavaHL layer in use. JavaHL will then ask FishEye to supply credentials when required, using a callback. The default JavaHL layer shipped with FishEye, SVNKit, can cache credentials at the server level rather than at the repository level.

There are different approaches to solving this issue:

Specify the username as part of the URL when defining your repository location (Recommended)

When using FishEye/Crucible 2.10, or later, we recommend that you simply include the username in the URL when defining the repository in FishEye. For example, use `http://username@host.com`.

Use the native JavaHL implementation

FishEye can be configured to [use the native JavaHL implementation](#), which will correctly apply the appropriate credentials.

Use the same authentication for the SVN server

The simplest solution is to have the same credentials for accessing the Subversion Server.

Use mock hostnames in the hosts file

Alternatively, SVNKit can be tricked into thinking that different servers are being used. For each connection to a repository a hostname in the *hosts* file can be defined.

All these entries then point to the same IP address of the SVN Server, but to SVNKit they look like different servers, thus bypassing the problem.

Example *hosts* entries (replace the IP address with the address of the SVN Server):

```
123.45.6.78 account1
123.45.6.78 account2
```

Replace these new server names in the SVN URLs:

```
http://account1/svn/project-a/
http://account2/svn/project-b/
```

What are Subversion root and tag branches?

FishEye identifies branches and tags in your Subversion repository by applying your specified [SVN tag and branch structure](#).

The "root:" branch

Any files or directories that fall outside the tag and branch structure are identified as being on the special branch, "root:". Some directories will almost always fall outside this structure. In general, root directories of branches are considered to be on the "root:" branch. This means that any changeset in which a branch is created is considered to be on branch, "root:". Additionally, any files or directories that fall outside the defined structure will be assigned branch, "root:". If you're seeing a lot of files and changesets on "root:", you may need to update your branch and tag structure in FishEye and rescan your repository, or exclude parts of your repository that don't follow your defined structure.

"tag:" branches

When FishEye detects that a tag has been created, it looks at the files that were tagged and adds the tag as an annotation to those file revisions. No file revisions are created at this point.

If a tag is modified after it has been created and committed, FishEye promotes the tag to a branch to preserve the history of the modification. For example, a user may create the tag "build1" by copying "trunk" to "tags/build1". If they then modify contents of `tags/build1`, a new branch "tag:build1" will be created for the modification.

Why do I need to describe the branch and tag structure for Subversion repositories?

In Subversion, branches and tags are defined by convention, based on their path within a repository, and not directly defined by the repository. A few different layout alternatives are commonly used. It is also possible that you are using your own custom layout. As a result you need to describe to FishEye which paths in your repository are used as [branches and tags](#).

It is very important that you correctly define in FishEye the layout you are using. If you do not, FishEye will not know which paths represent tags and branches. This will prevent FishEye from relating different versions of the same logical file across separate paths within your repository. It will also mean that FishEye's cache will be much larger as each tagged path will be indexed separately. This will result in an increase in the initial scan time and may reduce runtime performance.

If you are having trouble using Subversion tags, see [How tags work in Subversion](#).

Why don't all my tags show up in FishEye?

This page gives a detailed technical explanation of why certain issues affect Subversion users.

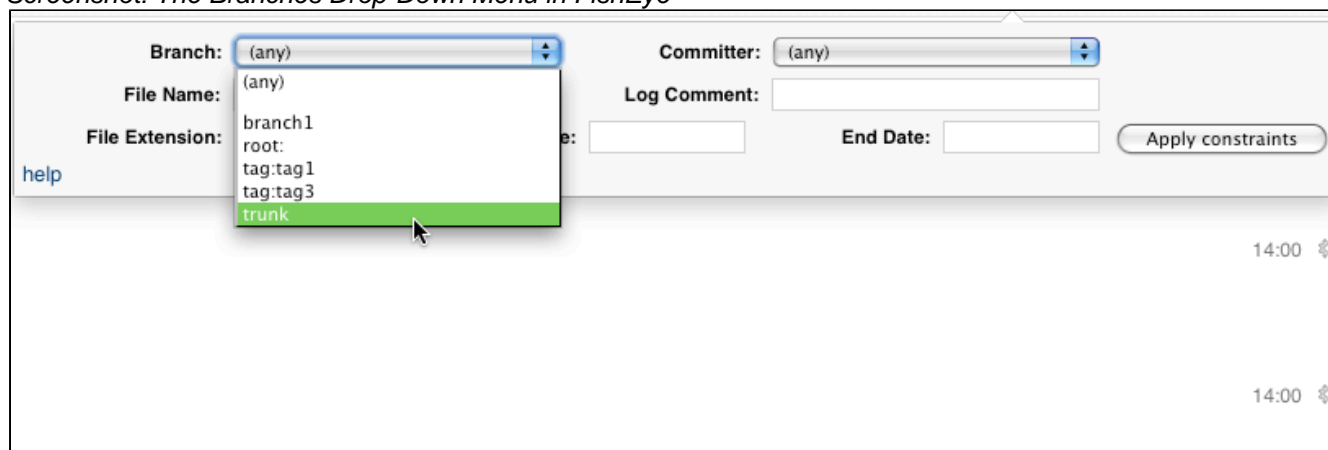
On this page:

- [Introduction](#)
- [How Subversion processes tags and branches](#)
- [An example from an Subversion repository](#)
- [Avoid modifications in the tag area](#)
- [Conclusion](#)

Introduction

When accessing Subversion from FishEye, you may see references to tags in the branches drop-down menu. In the example below, we can see **tag1** and **tag3** in the drop-down menu but not **tag2**:

Screenshot: The Branches Drop-Down Menu in FishEye



In actual fact, the branches drop-down menu shows only branch names. It does not show tags, but in some instances FishEye will synthesize a branch name to record certain operations. To understand how this occurs, you will need some background knowledge on Subversion tagging (introduced in the following segments of this page).

How Subversion processes tags and branches

In Subversion, tags are only a convention and are typically the result of a copy operation from the trunk to a tag area in the tags directory. When FishEye processes this copy operation, it recognizes that the destination is a tag directory and tags the source file on trunk with the name of the tag.

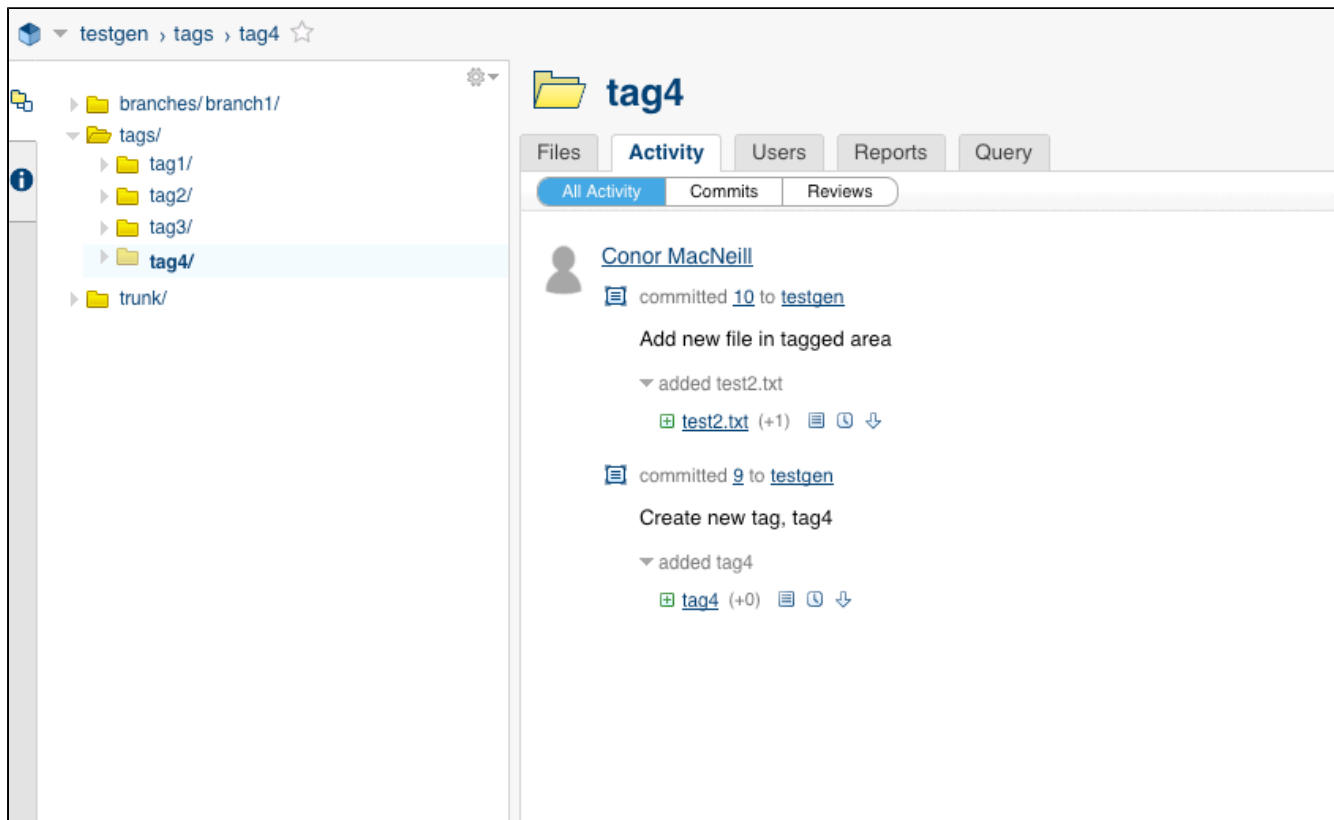
That is, FishEye is interpreting the Subversion copy to a tag directory as a tagging operation on the trunk files.

For regular changes in your Subversion repository, FishEye records each change against a branch where the change took place. If, however, after tagging, you make a change to a file in the tagged area, you are making a change outside trunk or a recognized branch. FishEye records such changes by creating an artificial branch name and associating that branch name with the change. The branch name is derived from the tag name by prepending "tag:" (in other words, the characters "tag:" appear as the first part of the name). The same thing will occur if you create a new file in the tagged area which does not come from an existing branch or trunk.

This is the reason you see some of your tags in the branch drop down. It means that for those tags, you have made a modification after the tagging operation.

An example from an Subversion repository

For example, consider **tag4** in this screenshot:



There are two changes here. The first creates the tag and the second adds a new file in the tagged area. This will result in the creation of an artificial branch, called "**tag:tag4**" within FishEye.

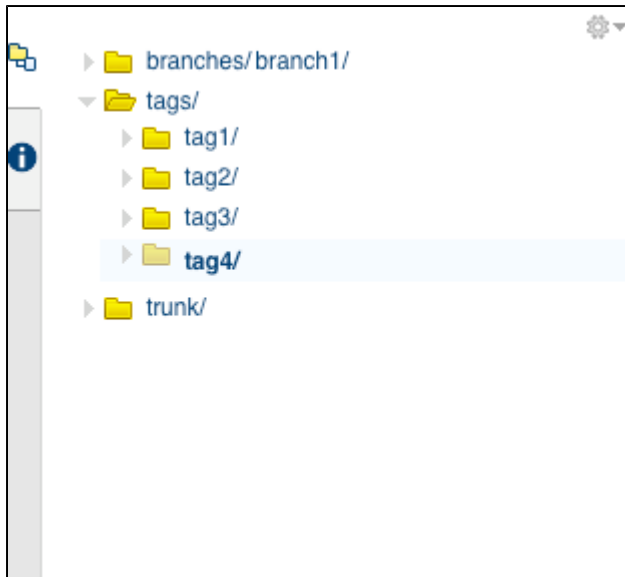
Avoid modifications in the tag area

In general, it's not good practice to make changes in the tag areas of a Subversion repository. Such changes can easily get lost if they are not applied to trunk or a current branch. It is preferable to make the change in trunk or a branch and then create a new tag to capture the update. Nevertheless, since Subversion tagging is merely a convention, this is sometimes convenient. FishEye handles this situation as described above.

Conclusion

In general a lot of systems have a large number of tags which would make the drop-down unworkable. This is the reason the tag field is a text-entry box below the branch drop-down menu in FishEye.

Since tags and branches are based on location convention in Subversion, the constraint is less effective than on other SCMs. You can always see the tag or branch you are interested in, based on its location in the repository. For example, the subdirectory list here shows all tags:



i If you want to constrain to a tag, enter the tag name in the tag field of the constraint filter.

About merges in Subversion

DO NOT PUBLISH BEFORE 4.1 release

This page explains how FishEye manages merge operations in Subversion, and provides examples of supported and unsupported merge formats.

- [Merge definition](#)
- [How FishEye shows merge information](#)
 - [Commit graph](#)
 - [Changeset view](#)
- [Acceptable path property format](#)
- [Merge targets and sources](#)
- [Unsupported operations](#)
 - [Advanced merging](#)

Merge definition

Subversion allows you to reapply changes made on other branches to the current working branch using the [merge](#) operation. For example, merging allows you to reintegrate a feature branch with trunk, or to reapply changes made on trunk to the current working branch. Subversion tracks this information in the dedicated `svn:mergeinfo` [path property](#). FishEye detects changes in those properties and uses those to mark parents of a changeset accordingly, effectively exposing merges between branches.

Notes:

- Although the path property might be set manually by using the [propset](#) command, FishEye trusts this information blindly, which means that if you add a required property in a correct format, FishEye will resolve parents for a changeset accordingly.
- The versions of the Subversion client and server used to perform merges both need to be version 1.5 or higher, since earlier versions do not have `svn:mergeinfo`. FishEye supports only branch and trunk path roots merges.

How FishEye shows merge information

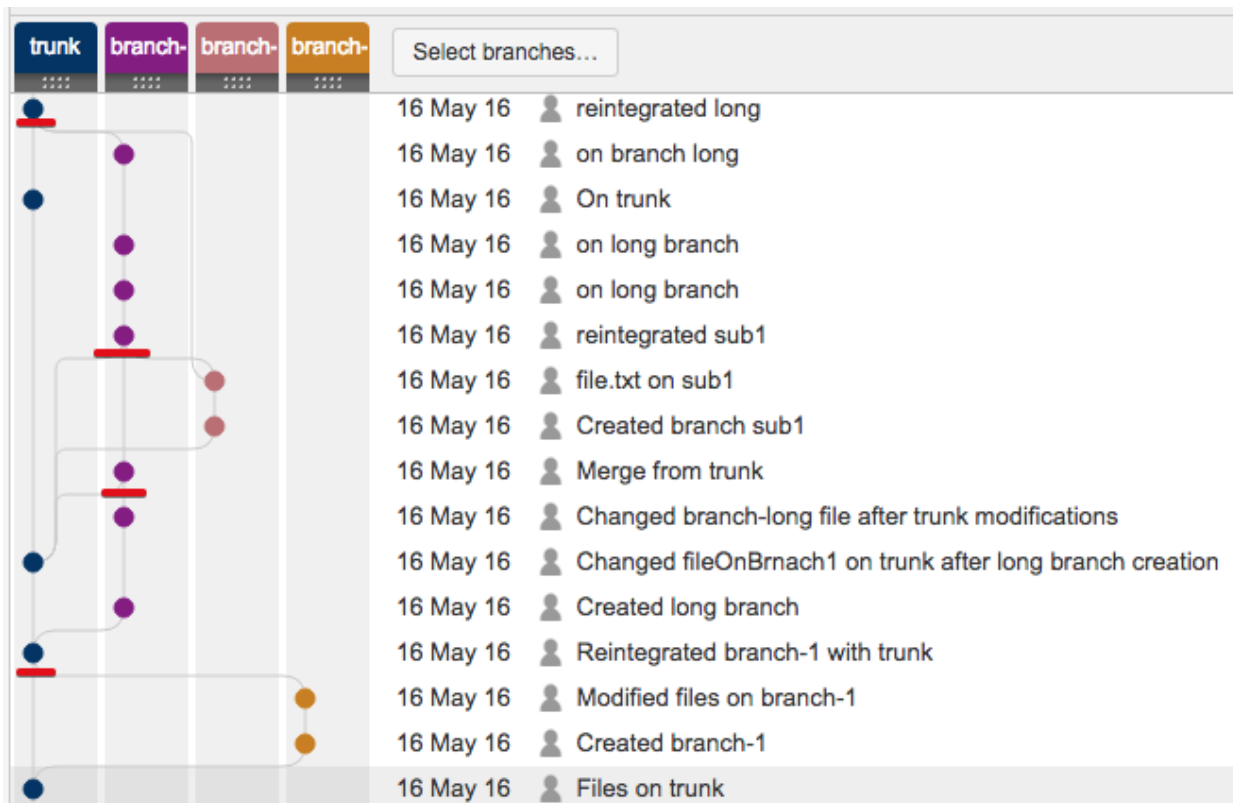
Changeset merges are exposed in three places:

- [commit graph](#)

- changeset view
- REST and Java API

Note: Although merge information is processed, changes on different branches are not reflected in the file ancestry history.

Commit graph



Changeset view

glewandowski committed 11 3 days ago
merge branch 2 to trunk

8 + 10
trunk

/trunk
file0.txt

View as Patch View in Activity Stream View in graph
Create Review

Blame j : next, k : prev

Acceptable path property format

FishEye uses the `svn:mergeinfo` property. It is set on a target path to which revisions has been merged and contains information about the revisions or revision range that has been merged, and the source of the merge. FishEye supports only base branches merge, which means that only the following format is acceptable:

```
(Source branch):(Range revision start)-(Range revision end)
```

For example:

```
/branches/feature-branch-1:10-15
```

However if none of the changesets in the range <10,15> made actual changes to `feature-branch`, (or those changes are not recognised as such by FishEye due to, for example, the included/excluded paths), the merge information won't be reflected.

Merge targets and sources

FishEye supports only branch and trunk path merges, which means that proper configuration of the [repository structure](#) is crucial. If the `mergeinfo` property is applied on a path that isn't identified as a branch or trunk root by SVN rules, it won't be processed. The same applies to `mergeinfo` value which describes the source of a merge. If it's not a branch/trunk root, or it's excluded by [include/exclude paths](#), it won't be processed or reflected in FishEye.

Considering the basic(built-in) rules and [recommended](#) repository structure, the following merge properties will be recognized:

```
Property changes on: branches/example-branch
```

```
Added: svn:mergeinfo  
Merged /trunk:r2-3
```

But the following **will not be recognized**:

```
Property changes on: trunk/directory
```

```
Added: svn:mergeinfo  
Merged /branches/example-branch/directory:r2-5
```

Unsupported operations

Advanced merging

None of the operations described [on this site](#) are supported by FishEye. Cherry picking revisions may result in `mergeinfo` formats similar to the following, which **won't be recognized**:

```
/branches/example-branch:5703  
/trunk:4622,5065,5300,5395,5403,5410-5414,5419,5423-5425
```

However, if you cherry pick revisions to create a valid revision range, it will be reflected in FishEye. For example, doing a couple of consecutive merge operations might result in a ranged `svn:mergeinfo` property format. Such a situation happens on the second merge command, which will transform `svn:mergeinfo` to contain the value `/example-branch:8-9`. In that case, FishEye will recognize it as a valid merge in the latest commit and mark it accordingly.

```
$ svn merge -c 8 ^/branches/example-branch && svn commit
$ svn merge -c 9 ^/branches/example-branch && svn commit
```

CVS FAQ

FishEye CVS FAQ

- [How does FishEye calculate CVS changesets?](#) — FishEye's goal is to allow changesets to be seen as a consistent stream of atomic commits. Revisions are collated into the same changeset provided that:
- [How is changeset ancestry implemented for CVS?](#)

How does FishEye calculate CVS changesets?

FishEye's goal is to allow changesets to be seen as a consistent stream of atomic commits. Revisions are collated into the same changeset provided that:

- They have the same commit comment.
- They are by the same author.
- They are on the same branch.
- The changeset does not span more than 10 minutes.
- The same file does not appear in a changeset more than once.

How is changeset ancestry implemented for CVS?

About Changeset Ancestry in FishEye

When FishEye indexes a CVS repository, it synthesizes a changeset identifier to group file-level changes into a single consistent changeset. The grouping is described in this FAQ: [How does FishEye calculate CVS changesets?](#).

Changeset ancestry was added in FishEye 2.6. Changeset ancestry refers to the linking of a changeset to a preceding/parent changeset(s). This allows you to view the development progress of your repository using the Commit Graph (see [Viewing the Commit Graph for a Repository](#)).

Changeset Ancestry for CVS

For CVS repositories, changeset ancestry is implemented, as follows:

- For all but the first change on a branch, FishEye chooses the most recent change on that branch as the parent changeset.
- For the first change on a branch, FishEye examines the branchpoints of all files in the branch and chooses the latest changeset that affected any such files as the parent changeset.

This approach ensures that a branch, whose first change is to a file which is very old, is not considered to have been branched at the time that file was last changed. It is considered to be branched at the last change to the repository instead.

Support Policies

Welcome to the support policies index page. Here, you'll find information about how Atlassian Support can help you and how to get in touch with our helpful support engineers. Please choose the relevant page below to find out more.

- [Bug Fixing Policy](#)
- [New Features Policy](#)
- [Security Bugfix Policy](#)

To request support from Atlassian, please raise a support issue in our online support system. To do this, visit support.atlassian.com, log in (creating an account if need be) and create an issue under FishEye. Our friendly support engineers will get right back to you with an answer.

Bug Fixing Policy

Summary

- Our Support team will help with workarounds and bug reporting
- We'll generally fix critical bugs in the next maintenance release
- We schedule non-critical bugs according to a variety of considerations

Report a bug

Building an add-on

Are you developing an add-on for an Atlassian product or using one of our APIs? Report any related bugs [here](#).

Bug reports

Atlassian Support is eager and happy to help verify bugs—we take pride in it! Create an issue in our [support system](#), providing as much information as you can about how to replicate the problem you're experiencing. We'll replicate the bug to verify, then lodge the report for you. We'll also try to construct workarounds if possible.

Search existing bug reports

Use our [issue tracker](#) to search for existing bugs, and [watch](#) the ones that are important to you. When you watch an issue, we'll send you an e-mail notification when the issue's updated.

How we approach bug fixing

Maintenance (bug fix) releases come out more frequently than major releases, and attempt to target the most critical bugs affecting our customers. The notation for a maintenance release is the final number in the version (the 1 in 6.0.1, for example).

If a bug is critical (production application down or major malfunction causing business revenue loss or high numbers of staff unable to perform their normal functions) we'll fix it in the next maintenance release, provided that:

- The fix is technically feasible (it doesn't require a major architectural change)
- It doesn't impact the quality or integrity of a product

For non-critical bugs, the developer assigned to fixing bugs prioritises the bug according to these factors:

- How many of our supported configurations are affected by the problem
- Whether there is an effective workaround or patch
- How difficult the issue is to fix
- Whether many bugs in one area can be fixed at one time

Developers responsible for fixing bugs also monitor comments on existing and new bugs, so you can comment to provide feedback if you need to. We give high priority to [security issues](#).

When considering the priority of a non-critical bug, we try to determine a *value* score for a bug. The score takes into account the severity of the bug from our customers' perspective, how prevalent the bug is, and whether new features on our roadmap may render the bug obsolete. Our developers combine the value score with a *complexity* score (how difficult the bug is) when selecting issues to work on.

Further reading

See [Atlassian Support Offerings](#) for more support-related information.

New Features Policy

Summary

- We encourage and display customer comments and votes openly in our issue tracking system, <http://jira.atlassian.com>.

- We do not publish roadmaps.
- Product Managers review our most popular voted issues on a regular basis.
- We schedule features based on a variety of factors.
- Our [Atlassian Bug Fixing Policy](#) is distinct from this process.
- Atlassian provides consistent updates on the top 20 issues.

How to track what features are being implemented

When a new feature or improvement is scheduled, the 'fix-for' version will be indicated in the JIRA issue. This happens for the upcoming release only. We maintain roadmaps for more distant releases internally, but because these roadmaps are often pre-empted by changing customer demands, we do not publish them.

How Atlassian chooses what to implement

In every [major release](#) we *aim* to implement highly requested features, but it is not the only determining factor. Other factors include:

- **Customer contact:** We get the chance to meet customers and hear their successes and challenges at Atlassian Summit, Atlassian Unite, developer conferences, and road shows.
- **Customer interviews:** All product managers at Atlassian do customer interviews. Our interviews are not simply to capture a list of features, but to understand our customers' goals and plans.
- **Community forums:** There are large volumes of posts on [answers](#), of votes and comments on [jira.atlassian.com](#), and of conversations on community forums like groups on LinkedIn.
- **Customer Support:** Our support team provides clear insights into the issues that are challenging for customers, and which are generating the most calls to support
- **Atlassian Experts:** Our [Experts](#) provide insights into real-world customer deployments, especially for customers at scale.
- **Evaluator Feedback:** When someone new tries our products, we want to know what they liked and disliked and often reach out to them for more detail.
- **In product feedback:** The [JIRA Issue Collectors](#) that we embed our products for evaluators and our Early Access Program give us a constant pulse on how users are experiencing our product.
- **Usage data:** Are customers using the features we have developed?
- **Product strategy:** Our long-term strategic vision for the product.
- Please read our [post on Atlassian Answers](#) for a more detailed explanation.

How to contribute to feature development

Influencing Atlassian's release cycle

We encourage our customers to vote on issues that have been raised in our public JIRA instance, <http://jira.atlassian.com>. Please find out if your request already exists - if it does, vote for it. If you do not find it you may wish to create a new one.

Extending Atlassian products

Atlassian products have powerful and flexible extension APIs. If you would like to see a particular feature implemented, it may be possible to develop the feature as a plugin. Documentation regarding the [plugin APIs](#) is available. Advice on extending either product may be available on the user mailing-lists, or at [Atlassian Answers](#).

If you require significant customisations, you may wish to get in touch with our [partners](#). They specialise in extending Atlassian products and can do this work for you. If you are interested, please [contact us](#).

Further reading

See [Atlassian Support Offerings](#) for more support-related information.

Security Bugfix Policy

See [Security @ Atlassian](#) for more information on our security bugfix policy.

Troubleshooting

FishEye Troubleshooting

- **After I commit a change to my CVS repository, it takes a long time before it appears in FishEye.** — If you do not have a CVSROOT/history file, then a commit will not appear in FishEye until after FishEye has done a periodic full scan of your repository. You can configure the period of this scan in the Admin pages.
- **FishEye freezes unexpectedly** — If your FishEye 2.0 or 2.0.1 instance freezes unexpectedly, this could be caused by a known issue with FishEye and MySQL database technology.
- **I have installed FishEye, and the initial scan is taking a long time. Is this normal?** — As a guide, FishEye should be able to process about 100KB-200KB per second on an averaged-size PC. If FishEye is accessing the repository over the network (e.g. over a NFS mount), then you should expect the initial scan to take longer.
- **I have installed FishEye, but there is no data in the Changelog.** — When you add a repository, FishEye needs to scan through the repository to build up its index and cache. This scan can take some time. Until this scan is complete, you may find that some data is not displayed.
- **Initial scan and page loads are slow on Subversion** — It's possible that you've mis-configured your tag and branch structure and caused FishEye to process some or all files as trunk files. You should recheck your tag configuration.
- **JIRA Integration Issues**
- **Manually Generating a Thread Dump**
- **Message 'org.tigris.subversion.javahl.ClientException svn Java heap space'** — The Java heap space needs to be increased to an acceptable size. See the FishEye Tuning documentation for more information.
- **Problems with very long comments and MySQL migration** — There is a known issue with FishEye 2.0.x and very long comments when migrating your database to MySQL.
- **URLs with encoded slashes don't work, especially in Author constraints** — If the author names in your repository contain slashes or back-slashes, and you are using Apache, you may run into a problem where these characters are incorrectly escaped.
- **Manually Generating a Thread Dump (Draft)**
 - All Repositories fail - Could not execute query (MySQL database)

FishEye Knowledge Base

See the troubleshooting guides and technical announcements in the [FishEye knowledge base](#).

After I commit a change to my CVS repository, it takes a long time before it appears in FishEye.

If possible, FishEye will monitor and parse the CVSROOT/history file in your repository to quickly work out what has changed. You may want to check with your CVS administrator to ensure this feature of CVS is turned on.

If you do not have a CVSROOT/history file, then a commit will not appear in FishEye until after FishEye has done a periodic full scan of your repository. You can configure the period of this scan in the Admin pages.

FishEye freezes unexpectedly

Issue Symptoms

If your FishEye 2.0 or 2.0.1 instance freezes unexpectedly, this could be caused by a known issue with FishEye and MySQL database technology.

This issue manifests itself in some FishEye pages returning a server timeout error. To identify the issue, check the FishEye error log. For this issue, the following output will appear in the error log:

```
2009-07-15 15:34:45,555 ERROR [btpool0-519] fisheye.app
HibernateUtil-commitTransaction - Commit fail msg-0:Could not execute
JDBC batch update
2009-07-15 15:34:45,556 ERROR [btpool0-519] fisheye.app
HibernateUtil-commitTransaction - Commit fail msg-1:Lock wait timeout
exceeded; try restarting transaction
2009-07-15 15:34:45,557 ERROR [btpool0-519] fisheye.app
HibernateUtil-commitTransaction - Commit failed rolling back.
...
...
Caused by: java.sql.BatchUpdateException: Lock wait timeout exceeded;
try restarting transaction
    at
com.mysql.jdbc.ServerPreparedStatement.executeBatch(ServerPreparedStatement
ent.java:647)
    at
com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.executeBatch(NewProxy
PreparedStatement.java:1723)
    at
org.hibernate.jdbc.BatchingBatcher.doExecuteBatch(BatchingBatcher.java:4
8)
    at
org.hibernate.jdbc.AbstractBatcher.executeBatch(AbstractBatcher.java:246
)
... 163 more
```

The FishEye error log can be found under `FISHEYE_INST/var/log/fisheye-error.log.YYYY-MM-DD`.

See the [JIRA issue](#) for more information.

Workaround

Until the issue is solved, the suggested course of action is to restart your FishEye instance. This will return FishEye to normal operation.

The FishEye development team is actively working on a solution and this be part of an upcoming point release of FishEye.

Requesting Support

If you require assistance in resolving the problem, please [raise a support request](#) under the FishEye project.

I have installed FishEye, and the initial scan is taking a long time. Is this normal?

When you add a repository, FishEye needs to scan through the repository to build up its index and cache. This scan can take some time. Until this scan is complete, you may find that some data is not displayed.

As a guide, FishEye should be able to process about 100KB-200KB per second on an averaged-size PC. If FishEye is accessing the repository over the network (e.g. over a NFS mount), then you should expect the initial scan to take longer.

For more details, see [Improve FishEye scan performance](#).

I have installed FishEye, but there is no data in the Changelog.

When you add a repository, FishEye needs to scan through the repository to build up its index and cache. This scan can take some time. Until this scan is complete, you may find that some data is not displayed.

As a guide, FishEye should be able to process about 100KB-200KB per second on an averaged-size PC. If

FishEye is accessing the repository over the network (e.g. over a NFS mount), then you should expect the initial scan to take longer.

Initial scan and page loads are slow on Subversion

Background Information

When you add a repository, FishEye needs to perform a once-off scan through the repository to build up its initial index and cache. This scan can take some time. Until this scan is complete, you may find that some data is not displayed. As a guide, FishEye should be able to process about 100KB-200KB per second on an averaged-size PC. If FishEye is accessing the repository over the network (e.g. over a NFS mount), then you should expect the initial scan to take longer. Read on if your scan appears to be considerably slower than expected.

Solutions

It's possible that you've mis-configured your tag and branch structure and caused FishEye to process some or all files as trunk files. You should [recheck your tag configuration](#).

If that fails, then the Atlassian support team will be happy to help you with this issue. Please [sign up for a support account](#) if you don't have one already, then login and [create a FishEye support request](#).

Users with very large or non-local repositories may be able to [improve their FishEye scan performance](#).

JIRA Integration Issues

*Users are mapped to their own accounts when using **Trusted Applications**.*

If you (or the general account used for JIRA access, if not using Trusted Applications) do not have the permissions to carry out the JIRA actions linked from FishEye, an error will occur. Depending on the error returned from JIRA, FishEye may not display the error correctly or display it at all, simply reporting that "An error has occurred". To investigate what the error was, you can access the FishEye debug log, named `fisheye-debug.log.YYYY-MM-DD` under the `dist.inst/var/log` folder of your FishEye installation and look for the date and time when your error took place. Here, you will be able to follow the links and see what error the JIRA instance was producing by clicking through to JIRA.

Manually Generating a Thread Dump

If FishEye/Crucible stops responding or is showing poor performance, providing thread dumps to Support can help diagnose the problem.

Note: If you were asked by Atlassian technical support to create thread dumps please take at least six thread dumps – one every ten seconds for one minute so we can identify what the application is doing. Attach all generated files to the support ticket in addition to a Support Zip (the default Support Zip options includes `fisheye.out` which is the file to which thread dumps are sometimes written).

Generating a Thread Dump for Windows

Method 1: Windows scripts

We now have scripts for generating thread dumps externally on Windows. Download them from this [Bitbucket Repository](#).

Method 2: CTRL+BREAK (only if FishEye/Crucible is running in a console window)

1. Right click on the titlebar for the command console window where FishEye/Crucible is running to open the Properties dialog box.
2. Select the **Layout** tab.
3. Under Screen Buffer Size, set the **Height** to 3000 and click OK.
4. Press CTRL-BREAK on your keyboard.
5. This will output the thread dump to the command console.
6. Scroll back in the command console until you reach the line containing "Full thread dump".

7. Right click the title bar and select **Edit -> Mark**.
8. Highlight the entire text of the thread dump.
9. Right click the title bar and select **Edit -> Copy**.
10. Open a text editor and paste the thread dump and save the file.

Method 3 - VisualVM:

VisualVM is only provided as part of the Java Development Kit (JDK) distribution of Java. The JDK is not packaged with FishEye/Crucible, and will need to be downloaded separately. Your **JAVA_HOME** environment variable specifies which version of the Java Virtual Machine (JVM) FishEye/Crucible uses. If no **JAVA_HOME** environment variable is set, FishEye/Crucible uses the Java Runtime Environment (JRE) version packaged with the application.

Scenario 1: FishEye/Crucible is using the JVM provided by the JDK - AND - FishEye/Crucible is running as a console application:

1. Start **VisualVM** (<JDK installation directory>\bin\jvisualvm.exe)
2. Select **com.cenqua.fisheye.FishEyeCtl** application in the left-hand pane under **Local**.
3. Select the **Threads** tab in the right pane.
4. Click **Thread Dump**.
5. Select the **Threads** tab to return.

Scenario 2: FishEye/Crucible is not using the JVM provided by the JDK - AND - FishEye/Crucible is running as a console application:

1. Stop FishEye/Crucible.
2. Edit <FishEye/Crucible installation directory>\bin\fisheyectl.bat
3. Add the following JMX parameters to the **%_EXEC CMD% %FISHEYE_OPTS%** property (within the existing quotes)

```
-Dcom.sun.management.jmxremote.port=6080  
-Dcom.sun.management.jmxremote.authenticate=false  
-Dcom.sun.management.jmxremote.ssl=false
```

You must choose a free port for JMX to bind to – this example uses port 6080

4. Restart FishEye/Crucible.
5. Start VisualVM: <JDK installation directory>\bin\jvisualvm.exe
6. Select **File > Add JMX Connection**.
7. Type: **localhost:<your JMX port>**
8. Select **com.cenqua.fisheye.FishEyeCtl** application in the left-hand pane under **Local**.
9. Select the **Threads** tab in the right pane.
10. Click **Thread Dump**.
11. Select the **Threads** tab to return.

Scenario 3: If FishEye/Crucible is running as a service:

1. Stop FishEye.
2. Go to **Start > Apps** (in Windows 8.1/Server 2012) or **All Apps** (in Windows 8) > **Configure FishEye**.
3. Add the following JMX parameters to the service configuration on the **Java** tab under **Java**:

```
-Dcom.sun.management.jmxremote.port=6080  
-Dcom.sun.management.jmxremote.authenticate=false  
-Dcom.sun.management.jmxremote.ssl=false
```

You must choose a free port for JMX to bind to – this example uses port 608

4. Click **Apply**, then **OK**
5. Restart FishEye.
6. Open Task Manager.
7. Look for the Process Name **Common Daemon Services Runner**.
8. Open the tree for the process and look for the sub-process "Atlassian FishEye". The PID for the process will be located under the PID column.

If you are running multiple Atlassian products on the same server, each will have a separate Process Name Common Daemon Services Runner.

9. Start VisualVM: <JDK installation directory>\bin\jvisualvm.exe
10. Select **File > Add JMX Connection**.
11. Type: **localhost**:<your JMX port> and click **OK**.
12. Select **localhost**:<your JMX port> in the left-hand pane under "Local".
13. Select the **Threads** tab in the right pane.
14. Click **Thread Dump**.
15. Select the **Threads** tab.

Generating a Thread Dump on Linux, including Solaris and other Unixes

Method 1 - jstack:

1. Find the process ID of FishEye/Crucible the JVM using the ps command.

```
FC_PID=`ps aux | grep -i fisheye | grep -i java | awk -F '[ ]*' '{print $2}'`;
```

2. Run the following command 6 times with a 10 second interval:

```
top -b -H -p $FC_PID -n 1 > fisheye_cpu_usage.`date +%s`.txt;
jstack -l $FC_PID > fisheye_threads.`date +%s`.txt
```

3. Look in the resulting CPU usage files (fisheye_cpu_usage.<timestamp>.txt) to identify which threads are consistently using a lot of CPU time.
4. Take the PID of the top 10 threads which are using CPU time and convert them to Hexadecimal (Eg: 11159 becomes 0x2b97).
5. Find the Hex values in the thread dump (fisheye_threads.<timestamp>.txt) files to figure out which threads are occupying CPU.

Method 2 - Kill Signal:

1. Find the process ID of FishEye/Crucible the JVM using the ps command:

```
ps aux | grep -i fisheye | grep -i java | awk -F '[ ]*' '{print $2}'
```

2. Run the following command 6 times with a 10 second interval:

The **kill -3** command will **NOT** terminate your FishEye/Crucible process.

```
kill -3 <pid>
```

The thread dump will be printed to FishEye/Crucible's standard output (fisheye.out).

Tools for analyzing thread dump files:

- Samurai - <http://samuraism.jp/samurai/en/index.html>
- Thread Dump Analyzer (TDA) - <https://java.net/projects/tda>

Message 'org.tigris.subversion.javahl.ClientException svn Java heap space'

When adding a new repository and on the initial scan, you may receive messages similar to this in the logs: `org.tigris.subversion.javahl.ClientException: svn: Java heap space`. The Java heap space needs to be increased to an acceptable size. See the [FishEye Tuning](#) documentation for more information.


Problems with very long comments and MySQL migration

Issue Symptoms

There is a known issue with FishEye 2.0.x and very long comments when migrating your database to MySQL. In some circumstances, this might result in truncation of very long comments, causing data loss.

Depending on your configuration, you may see an error message like this while migrating to MySQL, causing the migration to fail:

```
2009-07-16 16:56:12,390 ERROR [ThreadPool1] fisheye.app
com.cenqua.crucible.actions.admin.database.DBEditHelper-doGet -
Database migration failed:
java.sql.BatchUpdateException: Data truncation: Data too long for column
'cru_message' at row 1
java.sql.BatchUpdateException: Data truncation: Data too long for column
'cru_message' at row 1
```

 You may not see the message if you are running MySQL with default settings.

For more information, see the [JIRA issue](#).

Workaround

If your data contains very long comments or review descriptions (longer than 21,845 multibyte unicode characters), consider avoiding use of MySQL until this issue is resolved fully. Alternatively, use PostgreSQL or the default (built-in) HSQLDB database.

The FishEye developers are actively working on a solution to this problem and it will be addressed in an upcoming FishEye point release.

Requesting Support

If you require assistance in resolving the problem, please [raise a support request](#) under the FishEye project.

URLs with encoded slashes don't work, especially in Author constraints

If the author names in your repository contain slashes or back-slashes, and you are using Apache, you may run

into a problem where these characters are incorrectly escaped. By default Apache explicitly forbids encoded slashes or back-slashes in URLs. You can change this behavior with the following httpd.conf directive:

```
AllowEncodedSlashes On
```

This directive is documented [here](#).

FishEye Developer FAQ

This page contains answers to frequently asked questions posed by FishEye developers. For detailed information about developing in FishEye, see the [FishEye Developer documentation](#).

Feel free to comment, make submissions, or pose your own question on FishEye Development here.

- **Q:** *I'm getting the error "API access is disabled" as a response from <http://fisheye/api/rest/repositories> on my installation. How do I enable the API as a FishEye administrator?*
 - ▼ [Click here to expand...](#)
 - A:** A toggle to enable the API under "Server Settings" in the web admin interface existed in versions prior to 2.9 (see [Configuring the FishEye web server](#) for more details).
 - [See the FishEye 2.9 Release Notes](#)
- **Q:** *Is there any way to return unique results from an EyeQL query?*
 - ▼ [Click here to expand...](#)
 - A:** It is not currently possible to return unique results.
An improvement request exists: [FE-1136](#). Your vote and comments on that issue are appreciated.

Contributing to the FishEye Documentation

Would you like to share your FishEye hints, tips and techniques with us and with other FishEye users? We welcome your contributions.

On this page:

- [Blogging your Technical Tips and Guides – Tips of the Trade](#)
- [Contributing Documentation in Other Languages](#)
- [Updating the Documentation Itself](#)
 - [Getting Permission to Update the Documentation](#)
 - [Our Style Guide](#)
 - [How we Manage Community Updates](#)

Blogging your Technical Tips and Guides – Tips of the Trade

Have you written a blog post describing a specific configuration of FishEye or a neat trick that you have discovered? Let us know, and we will link to your blog from our documentation.

Contributing Documentation in Other Languages

Have you written a guide to FishEye in a language other than English, or translated one of our guides? Let us know, and we will link to your guide from our documentation. [More...](#)

Updating the Documentation Itself

Have you found a mistake in the documentation, or do you have a small addition that would be so easy to add yourself rather than asking us to do it? You can update the documentation page directly

Getting Permission to Update the Documentation

Please submit the [Atlassian Contributor License Agreement](#).

Our Style Guide

Please read our short [guidelines for authors](#).

How we Manage Community Updates

Here is a quick guide to how we manage community contributions to our documentation and the copyright that applies to the documentation:

- **Monitoring by technical writers.** The Atlassian technical writers monitor the updates to the documentation spaces, using RSS feeds and watching the spaces. If someone makes an update that needs some attention from us, we will make the necessary changes.
- **Wiki permissions.** We use wiki permissions to determine who can edit the documentation spaces. We ask people to sign the [Atlassian Contributor License Agreement \(ACLA\)](#) and submit it to us. That allows us to verify that the applicant is a real person. Then we give them permission to update the documentation.
- **Copyright.** The Atlassian documentation is published under a Creative Commons CC BY license. Specifically, we use a [Creative Commons Attribution 2.5 Australia License](#). This means that anyone can copy, distribute and adapt our documentation provided they acknowledge the source of the documentation. The CC BY license is shown in the footer of every page, so that anyone who contributes to our documentation knows that their contribution falls under the same copyright.

RELATED TOPICS

[Author Guidelines](#)

[Atlassian Contributor License Agreement](#)

FishEye Documentation in Other Languages

Below are some links to FishEye documentation written in other languages. In some cases, the documentation may be a translation of the English documentation. In other cases, the documentation is an alternative guide written from scratch in another language. This page presents an opportunity for customers and community authors to share documentation that they have written in other languages.

Please be aware that these are external guides.

Most of the links point to external sites, and some of the information is relevant to a specific release of FishEye. Atlassian provides these links because the information is useful and relevant at the time it was written. Please check carefully whether the information is still relevant when you read it, and whether it is relevant to your version of FishEye. The information in the linked guides has not been tested or reviewed by Atlassian.

On this page:

- No guides yet

None
<i>No guides yet</i>
We do not yet have any guides to link here. Be the first to suggest one!

Adding Your Own Guide to this Page

Have you written a guide for FishEye in another language? Add a comment to this page, linking to your guide. We will include it if the content fits the requirements of this page.

Giving Feedback about One of the Guides

If you have feedback on one of the guides listed above, please give the feedback to the author of the linked guide.

If you want to let us know how useful (or otherwise) one of these guides is, please add a comment to this page.

Other Sources of Information

[FishEye documentation](#)
[Atlassian website](#)
[Atlassian blog](#)
[FishEye plugins](#)

FishEye Resources

Resources for Evaluators

- [Free Trial](#)
- [Feature Tour](#)

Resources for Administrators

- [FishEye Knowledge Base](#)
- [FishEye FAQ](#)
- [Guide to Installing an Atlassian Integrated Suite](#)
- [The big list of Atlassian gadgets](#)

Downloadable Documentation

- [FishEye documentation in PDF, HTML or XML formats](#)

Plugins

- [FishEye Developer Documentation](#)
- [Add-ons for FishEye](#)

Support

- [Atlassian Support](#)
- [Support Policies](#)

Forums

- [FishEye Forum](#)
- [FishEye Developers Forum](#)

Mailing Lists

- Visit <http://my.atlassian.com> to sign up for mailing lists relating to Atlassian products, such as technical alerts, product announcements and developer updates.

Feature Requests

- [Issue Tracker and Feature Requests for FishEye](#)

Glossary

Code repository or SCM ([Source Code Management](#)) software terminology can be confusing. This page provides definitions for some of the most commonly used terms.

On this page:

- [Branch](#)
- [Changeset](#)

- [Commit](#)
- [Committer](#)
- [CSID](#)
- [Head](#)
- [Linker](#)
- [Repository](#)
- [SCM](#)
- [Slurp](#)
- [Tag](#)
- [Trunk](#)

FishEye and its documentation uses the following terms:

Branch

A branch is an independent stream of work in a repository. For example, you might copy a set of files in the repository into a new branch, where you can carry on with a separate stream of work without cluttering up the main production area on trunk.

Different SCMs handle branching and merging in different ways. The common aspects allow users to create a branch in which to make changes which do not affect the files in other branches and the trunk development stream. These changes can then be merged into the trunk in a controlled fashion when a development unit of work is complete. Branches can also be used for experimental changes so that these do not affect the main development.

Changeset

A changeset is a collection of changes to files in a repository which are committed to the repository in a single operation with a single commit message. Not all SCMs support atomic commit operations. For these SCMs, FishEye will determine the file revisions which make up the changeset using a reliable heuristic (set of rules).

Different SCMs use different terms for the concept of a changeset, such as "changelist", which is generally interchangeable with changeset.

Commit

A commit is a single entry of content (usually source code) into a repository. It can be a single file or comprised of multiple file versions.

Committer

A committer is a user of an SCM repository who is adding content to the repository (where it will be permanently archived). Typically, a committer is a programmer who is committing source code but SCMs can also store other files such as documents, images and schematics.

CSID

An abbreviation for 'Changeset ID', this is a code that is used to reference every set of files that is committed to a repository. For example, if you commit three files to a repository, they are collectively a changeset, and will share the one CSID. CSIDs normally appear as a number, for example '**31905**'. In FishEye, CSIDs appear as links that you can click to visit the '**Changeset View**', which shows a list of the files in the left column, and the file contents or diffs in the right hand pane. In some circumstances you can hover your mouse over the CSID to see the '**Changeset Hover**' dialog, which displays the commit message, author, timestamp and files in the changeset.

Head

The "head" revision is the latest change to be made to a file in either a trunk or a branch part of a repository.

Linker

FishEye can render issue IDs or Bug IDs that appear in commit messages or comments as hyperlinks to the relevant issue/bug in your issue/bug tracker. The "linker" patterns used to detect the ID substrings can be configured separately for each repository. See [Linkers](#).

Repository

A repository is a area managed by an SCM where you store a set of related files, typically from a project or set of projects. The SCM is responsible for version controlling the files in the repository and maintaining their change history. A repository will contain the trunk and all branches for the files of the various projects. A single SCM instance can typically house multiple repositories.

SCM

SCM ([Source Code Management](#)) software is a category of computer software that archives complex sets of files, for example, all the source code comprised in a large multi-part software project. SCM systems keep copies of every version of every file, allowing you to completely restore and build any version of the software from any point in time.

Committers typically add new versions of code to the SCM once it is tested and approved by [peer code review](#) or quality assurance.

Each instance of an SCM can host multiple repositories.

Slurp

"Slurping" is a term that is synonymous with "repository scanning". FishEye must do intensive scans of the contents of repositories (SCM systems) in order to provide its lightning-fast web-based browsing of their contents. This can be referred to as a *slurp*, or *slurping*.

Tag

In SCM terminology, a "tag" is a label that is added to a number of files, to capture their collective state at a particular moment in time. A typical tag would be a specific software version number, that could be referenced to see all the files that belong to a specific version build of a software project.

Trunk

In SCM terminology, the "trunk" is the central part of the version control "tree". For example, you might copy a set of files in the repository into a new branch, where you can do new experimental work without cluttering up the main production area on trunk.

Collecting analytics for FishEye

We are continuously working to make FishEye better. Data about how you use FishEye helps us do that. We have updated our Privacy Policy so that we may collect usage data automatically, unless you disable collection. The data we collect includes information about the systems on which your installation of FishEye is operating, the features you use in FishEye, and your use of common IT terminology within the product. For more details, see our [Privacy Policy](#), in particular the 'Analytics Information from Downloadable Products' section.

See also our [End User Agreement](#).

How to change data collection settings?

You can opt in to, or out of, data collection at any time. A FishEye admin can change the data collection settings by going to **Analytics** (under 'Global Settings') in the FishEye admin area.

How is data collected?

We use the Atlassian Analytics plugin to collect event data in FishEye. Analytics logs are stored locally and then periodically uploaded to a secure location.