SVN Usage

How to access

- Full information is available here: http://svn.web.cern.ch/svn/howto.php

Web Browsing

- Authenticated with CERN account
  - https://svnweb.cern.ch/cern/wsvn/etics
  - https://svnweb.cern.ch/cern/wsvn/emisa2
- Non authenticated
  - http://svnweb.cern.ch/world/wsvn/etics
  - http://svnweb.cern.ch/world/wsvn/emisa2

Fisheye Browsing

- https://tomtools.cern.ch/fisheye/browse/ETICS

How is organized

- Unfortunately the two repositories have different structures:
  - ETICS
    - "trunk"
      - module
      - module
      - module
      - [...]
    - "branches"
      - module
        - branch
        - branch
        - [...]
      - module
        - branch
        - [...]
    - "tags"
      - module
        - tag
        - tag
        - [...]
  - EMISA2
    - module
      - "trunk"
      - "branches"
        - branch
        - branch
      - "tags"
        - tag
        - tag
    - module
      - "trunk"
      - "branches"
        - [...]
      - "tags"
        - [...]
    - [...]

Permissions

- Read access is public
- To have write access, please contact Vitor Gouveia

How to checkout

Command-Line

$yum install subversion

or

$ apt-get install subversion

$ svn co https://svn.cern.ch/reps/<etics or emisa2> (append here the trunk/tags/branches and the module)
$ svn co svn+ssh://svn.cern.ch/reps/<etics or emisa2> (append here the trunk/tags/branches and the module)

or if you use a different user name:
$ svn co svn+ssh://<lxplus_username>@svn.cern.ch/reps/<etics or emisa2>

Eclipse

Install Subeclipse or subversive

How to checkout in ETICS configurations

With the old ETICS CVS repository the checkout command in the configurations in ETICS was:

- cvs -d ${vcsroot} co -r ${tag} ${moduleName}

From now on, depending of what is being checkout the command is:

- Trunk: svn co ${vcsroot}/trunk/${moduleName} ${moduleName}
- Branches: svn co ${vcsroot}/branches/${moduleName}/${tag} ${moduleName}
- Tags: svn co ${vcsroot}/tags/${moduleName}/${tag} ${moduleName}

Where ${vcsroot} is http://svnweb.cern.ch/guest/etics and ${tag} the name of the tag/branch of the module.

How to migrate from CVS

- If you are using a SVN module for the first time and you used to use the CVS module before
  - Identify the latest version of the module
  - Commit the last version in the Trunk (see below how to commit directory to directory)
  - Proceed as specified in the guidelines for new development

How to commit a SVN directory to another directory

To be used only when migrating new modules from CVS and the TRUNK is really old and not influent anymore. Otherwise use the MERGE instructions.

- When a branch has to be ported to the trunk before tagging a version
- When a version has to be committed to a branch to rollback some changes
- In general when it is needed to commit one SVN dir to another SVN dir as it is (no merging verification)

The best way to do this is using the command-line and rsync:

- Checkout the source and destination directories with Eclipse (with Eclipse checkout two different projects)
- Enter the source directory (usually the project directory inside the Eclipse workspace) and execute a rsync dry run to see what would be the result:

```
rsync -v --dry-run --checksum --recursive --delete --exclude .svn --exclude .project --exclude
.classpath --exclude bin . &lt;DESTINATION DIRECTORY INSIDE THE ECLIPSE WORKSPACE&gt;
```

- Verify the output of the command. Only the files to update (the files which differ from source and destination) are listed. An error appears for each directory to be removed (present in destination but not in source) because the directories are not empty (it is a dry run and inside there is at least .svn)
- From the same source directory execute the rsynch (this time without --dry-run to actually copy files and with --force to remove all non-empty directories):

```
rsync -v --checksum --recursive --delete --force --exclude .svn --exclude .project --exclude
.classpath --exclude bin . &lt;DESTINATION DIRECTORY INSIDE THE ECLIPSE WORKSPACE&gt;
```

- Verify that rsync does not print any errors.
- Now refreshing the Eclipse project
- Commit the destination directory with SVN or Eclipse.

How to Merge a SVN directory with another directory

- Merge means consider the updates done in two directories (usually trunk and a branch) from the branch point on and try to resolve possible conflicts before creating a single version with all updates
- To merge we need the branch starting point which is mentioned in the branch name
    - Find the revision number of the tag from which the branch has been created
-

TODO

What to commit

- Please DO NOT COMMIT BINARIES such as:
    - jar files
    - tar.gz
    - rpm
    - etc.
- Please DO NOT COMMIT ANY FILE WHICH IS LOCAL TO YOUR COMPUTER such as:
    - .project
    - .classpath
    - configuration files which include paths to your machine or test server URLs
    - .cvsignore
    - .svnignore
    - etc.
- A good way of avoiding to commit these files is adding them to .svnignore
    - remember to add .svnignore itself to .svnignore otherwise you will commit your personal exclude list!

Module structure

- This is of course valid only for new modules or for new files to be added in the SVN.
    - The already existing structures stay unchanged. It would take too long to fix everything.
    - Let's just keep the structure in mind so that if changes are needed, they are done according to the guidelines.

- Please use the following SVN directory structure to organize your files within your module:
    - **module root**: used for LICENSE CHANGELOG MAINTAINERS and build instructions (build.xml, pom.xml, setup.py, makefile, etc)
        - **src**: source code organized in subdirectory as the language recommends (Java uses package directories)
            - **autogen**: if for some reasons it is better to commit the automatically generated code instead of re-generating it at every build, it goes in this directory
        - **test**: Unit test source code (JUnit, PyUnit, etc.)
            - **resources**: anything needed by the tests apart from the test code itself
        - **doc**: any kind of documentation hand written
            - **autogen**: if for some reasons it is better to commit the automatically generated documentation (java-doc) instead of re-generating it at every build, it goes in this directory
        - **config**: all configuration file templates. These files MUST not contain PATHS of the developer machine or testing URLs
        - **project**: all files required for the application to run or to be packaged properly (web.xml, image files, MANIFEST, etc) unless they need to be stored in the source code tree
- Each module root MUST HAVE:
    - A LICENSE file with the Apache 2 license. Please use this http://www.apache.org/licenses/LICENSE-2.0.txt
    - A MAINTAINERS file with the list of maintainers of the module with the following format:

```
The maintainers of the &lt;MODULE-NAME&gt; are:

- &lt;NAME&gt; &lt;SURNAME&gt; - &lt;INSTITUTE&gt; - &lt;&lt;EMAIL ADDRESS&gt;&gt; [From version
&lt;XXX&gt; to version &lt;YYY&gt;]
- &lt;NAME&gt; &lt;SURNAME&gt; - &lt;INSTITUTE&gt; - &lt;&lt;EMAIL ADDRESS&gt;&gt;
```

Example:

```
The maintainers of the ETICS Repository WebService are:

- Tomasz Kokoszka - CERN - &lt;Tomasz.Kokoszka@cern.ch&gt; [From version 1.2.1 to version 2.4.0]
- Lorenzo Dini - CERN - &lt;Lorenzo.Dini@cern.ch&gt;
```

- A RELEASE-NOTES file with the user-oriented description of all the versions already committed
- A CHANGE-LOG file with the developer-oriented description of all the versions already committed
  - Both files must contain a list of the following format:

```
Release &lt;VERSION&gt; (&lt;YEAR&gt;-&lt;MONTH&gt;-&lt;DAY&gt;)
----------------------------
 - &lt;change log&gt; &lt;when possible GGUS or JIRA or Savannah number&gt;
 - &lt;change log&gt;
```

Example:

```
Release 1.4.0 (2010-04-09)
----------------------------
 - multi-node test support
 - multi-node report merging
 - multi-node scratch areas
 - External Repository Browsing

Release 1.4.1 (2010-07-28)
----------------------------
 - Creation of YUM repositories even without new packages (GGUS 60353)

Release 1.4.3 (2010-08-18)
----------------------------
 - SubmissionReport fix when repeating already registered packages

Release 1.4.4 (2010-08-24)
----------------------------
 - "checkout" job type added to submission workflow
 - changed key of external repository cache to solve Maven problem
```

- An INSTALL file with the installation instructions (clean, install, test, version):
  - Example:

```
[CLEAN]
If there is an old version of the ETICS Repository Webservice:
 Remove the repository folder in ${CATALINA_HOME}/webapps (if any)
 Remove the repository folder in ${CATALINA_HOME}/work/Catalina/localhost (if any)
 Remove the file eticsRepositoryWSContext.xml from ${CATALINA_HOME}/conf/Catalina/localhost
 Remove the file eticsRepositoryWS.war from ${ETICS_SERVICE_HOME}/share/webapps
 Remove the configuration files starting with eticsRepositoryWS* from ${ETICS_SERVICE_HOME}/etc

[INSTALL]
Check that the file ${ETICS_HOME}/etc/eticsRepositoryWSContext.xml:
 Has "docBase" configured with ${ETICS_HOME}/share/webapps/eticsRepositoryWS.war
 Has "configurationFile" configured with ${ETICS_HOME}/etc/eticsRepositoryWSConfiguration.xml
Copy ${ETICS_HOME}/etc/eticsRepositoryWSContext.xml to ${CATALINA_HOME}/conf/Catalina/localhost/
Edit the file ${ETICS_HOME}/etc/eticsRepositoryWSConfiguration.xml
 Follow the directions in the file if any other operation is needed (ie file permissions).
Restart Tomcat (if the Tomcat Context is not automatically updated)

[TEST]
To test if the deployment is successful try to access the page configured in the configuration file
serverPath property
 (Usually https://hostname:8443/repository/services/RepositoryService)


[VERSION]
To check web service version visit page /getVersion.jsp in application context configured in the
configuration
file serverPath property
 (Usually https://hostname:8443/repository/services/RepositoryService?method=getVersion )
```

Commit comments

- When committing a merged-branch in the trunk the comment must be: **Branch <BRANCH-NAME> merged: <MAIN CHANGE(S) FROM THE BRANCH>**
- When tagging a version, the comment must be: **Version <XXX>: <MAIN CHANGE(S) FROM THE LAST VERSION>**
- When branching the comment must be: **Starting branch <BRANCH-NAME>: <SHORT DESCRIPTION OF THE PURPOSE OF THE BRANCH>**

- JIRA allows to bind a SVN commit to a JIRA issue.
- If you are committing some code to fix a JIRA Bug or to implement a JIRA improvement or related anyway to a JIRA issue, please use the following format in the COMMIT COMMENT (you need to add the JIRA Issue ID somewhere in the message):
    - **EMITOOLS-XXX - Your descriptive commit message goes here**
    - In this case the commit will be linked to the JIRA Issue EMITOOLS-XXX
- http://confluence.atlassian.com/display/JIRA/Viewing+an+Issue's+FishEye+Changesets

Where/How to commit, tag and branch

- Please follow the following guidelines:
    - Trunk
        - The main development is done directly in the trunk
        - A new version should be **always** tagged from trunk
        - When tagging a version, the SVN commit must be the new release notes and change logs from the last tagged version (taken from RELEASE-NOTES and CHANGE-LOG)
    - Branches:
        - A branch can be used for evolving a version in parallel to the trunk.
        - The branch should be named etics-<subsystem>-<component>_branch_X_Y_Z_A
        - The version is the same as the trunk
        - A branch should not be tagged, should be merged with the trunk
        - Once a branch has been merged back to trunk and no longer is used, delete it!
        - When branching the comment must be: **Starting branch <BRANCH-NAME>: <SHORT DESCRIPTION OF THE PURPOSE OF THE BRANCH>**
    - Tags are named etics-<subsystem>-<component>_R_X_Y_Z_A
        - **DO NOT COMMIT IN TAGS**
        - All tags starting from 30/11/2011 should be stable at all times, compiling and with every feature completed
        - When tagging a version, the comment must be: **Version <XXX>: <MAIN CHANGE(S) FROM THE LAST VERSION>**
    - Versions: X Y Z A are:
        - X: Major Version (Non backward compatible change) - This is increased in big changes - Starts from 0 and goes to 1 when the first feature complete version is available
            - Y: Minor Version (Backward compatible change) - This is increased normally at any small feature update. This is the most frequent upgrade. - Starts from 1
            - Z: Revision Version (bugfixes) - This is increased if NO NEW FEATURES are added - Starts from 0
            - A: Age: Only if no code is changed, only configuration files or build instructions - Starts from 1
            - We usually start a new component version with 0_1_0_1