

Mainframe CVS at Rocket Software

Extending the functionality of the
z/OS Unix System Services CVS client

Lisa Bates
lbates@rs.com

April, 2006



Rocket® Rocket Software



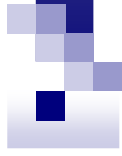
Background

- Rocket wanted to standardize on one source code / version control system that could be used on ALL platforms, instead of:
 - Visual SourceSafe on Windows
 - Plans, intentions to use Subversion
 - Widespread use of CVS on *nix
 - limited use of SCLM on TSO
 - Nightly GDG backup of TSO libraries
- I was “voluntold” to create the TSO CVS tool Suite



Benefits of Mainframe CVS

- Facilitate Branching of code
- Allow multiple developers secure means of maintaining same source files without ‘stepping’ on each other
- Source changes are managed at the ‘logical’ level, and as often as is warranted for your project.
- Better means of archiving code, versus the system backup process(es) that run today. Can have incremental iterations of code changes during the day.
- Integration with RockeTrack: comments associated with **Commit** to a Repository, flow over to a referenced RockeTrack ticket.
- The same code base is available to almost all systems. For example, you could develop a product on RSPLX01, and **Checkout** (or use **Prepare**) on RS17, perhaps using a ‘JES3’ branch.



Real problems that are solved using Mainframe CVS Tools

- A member of your team has left the company. You want to evaluate what changes that person made in order to decide what to keep and what to discard.
- Address 'the single-threaded' issue of sending fixes to IBM.
- Problems found in QA testing don't require that all current development be backed off when a something needs to be recut.
- Reduce the exposure of losing source changes because multiple developers are working in the same library. Each developer works in their own working dataset.



Overview: Define the terms

- **CVS**
CVS is an acronym for Concurrent Versioning System. It lets many developers work on a shared set of source files independently at the same time.
- **Repository**
A database that keeps track of all your sources and every change that you make to them. The CVS commands help manage concurrent access and changes to your sources. The repository is located on our network. You can get any past '*committed*' version of the sources at any time from the repository. Each project has its own repository, so your development team can control your own repository.
- **Working Dataset**
Each developer has their own copy of part or all of the sources, which are in a Working Dataset. This is your own private development area, where you make changes and test them independently of other developers. You can create temporary members in your working dataset which are not sources controlled by CVS, and the CVS commands will ignore them. Your project will usually consist of several related datasets.
- **Revision Tree**
Each source file stored in the repository can have many versions, that are organized in a tree structure. Each node on the tree is called a "revision", and the edges represent the development path from one change to the next. Each source file has its own separate tree of revisions, and each revision has a unique revision number associated with it. Each revision has a multi-part revision number (eg. 1.42, 1.30.4.2), but these have nothing to do with release version numbers. The last part of the revision number is incremented for every commit made.



Overview: Define the terms

- **Trunk**
The trunk of the revision tree represents production releases, and will be managed by the packaging team. The most recent revision on the trunk is known by the special tag name "HEAD", and all commits to the trunk create a new revision after the head and become the new head.
 - **Branch**
A development branch can be created from any revision, and lets you split lines of development apart. When your working dataset is checked out from a branch, all commits made from it are made to the branch instead of the trunk. A branch has a tag name which works like "HEAD" to refer to the most recent revision on that branch. You can branch from branches, which is what creates the revision tree structure. You can merge any revision from one branch into any other branch.
- We use branches to represent maintenance development, and branches from maintenance branches to represent RockeTrack issue and APAR development.
- **Tag**
A tag is a symbolic name for a revision of a source file. Revision numbers can get confusing quickly, so we usually tag particular revisions to indicate their purpose. Putting the same tag on many files is how you tell CVS to remember which revisions of each source was in a particular release, or APAR, or whatever.
 - **Module**
In CVS-speak, a "module" is a subset of a repository. We use CVS modules to represent datasets within a project. For example, a project might have CVS modules named "ASM", "JCLLIB", "CPP" and so on. You can check out individual CVS modules from your repository into a working dataset, or all or some of the CVS modules into several related working datasets.



Assumptions

- Modules are all at same level in CVS repository. Usually at same level as CVSROOT.
- The CVSROOT/modules file contains an entry for each module to be managed, as in
moduleName pathToModule
- Typically, the modules file might look like
asm asm
ispplib ispplib
isptlib isptlib
maclib maclib
proclib proclib

3 Modes:

ISPF Panels, Batch, TSO Command Line

- ISPF Interface. Series of ISPF panels for each function.
- Dynamically turn on/off Batch mode. Use Panel interface to execute commands, Batch jobs are generated and can be reused as needed.
- Command line. Each CVS function is implemented as a Rexx command. The ISPF interface call these Rexx commands to do the work. Syntax for all commands is:

command repository module datasetStem -options

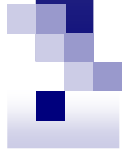
checkout zcvs exec pdbate.demo -r oorexx

- *Checkout from ZCVS repository, the exec module, write to a HLQ that begins with PDBATE.DEMO and get from the oorexx Branch*



General Comments: 1

- Almost all the commands operate on a working dataset prefix, also called the Target HLQ.
- When initially putting content into the repository, this Target HLQ might reference a PRD or MNT dataset.
- The sequence of commands that put content into the repository (checkout, add, commit), do not require **write** access to the dataset.
- Ideally, the Target HLQ should reference a dataset that is your own private “sandbox copy”. This is not a requirement, but rather a recommendation.



General Comments: 2

- Most of the commands let you operate at the dataset or member level. If you don't specify a member or member pattern, the command will do that function on the entire dataset.
- Most of the ISPF Panel commands have an option to do that operation on "All modules" .
- This is a useful feature and recommended once you know how the commands and panels operate.

Recommend to not use while learning how to use the cvs Tool Suite.



CVS Basic Operations: 1

- **Login**
CVS won't let you access a repository until it knows who you are, so you have to log in. You only have to do this once for each repository, CVS will remember your password and reuse it after that. Your password should be the same as the one you use for network logins. If it is not, get a password from your project leader.
- **Checkout**
This is the command that creates a working dataset for you. You must list one or more module names to tell CVS what you want from the repository. You can also give other options to get specific versions of the sources from the repository. By default, you get the most recent versions of all the files.
- **Prepare**
a Rocket developed command that does several CVS commands in one command, implementing the branching model for the PRD – MNT - WRK set of datasets.
- **Add**
Use this to add new files to the CVS repository. Note that this doesn't immediately put things into the repository; it just marks them to be added when you commit your changes.
- **Remove**
You shouldn't be using this command, and the only reason it's listed here is to say that. Well, there are two cases in which it might be needed: if you add a file by mistake, or with the wrong name, and want to fix it. Another reason to remove a file if the file is no longer necessary for any future development of the product. Remove only terminates the revision tree, all previous revisions are still in the repository. Remove operations takes effect on the next commit



CVS Basic Operations: 2

- **Update**

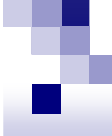
This is how you get other people's changes into your working dataset. You can update the entire working dataset, parts of it, or individual files. You should update your working dataset frequently, so that you keep up with the code. You might wish to delay updating if you're tracking down some bug and need to keep a stable environment while you try different tests. Actually, that's a good reason to make a separate working dataset.

When you update, CVS will list the files it is changing in your working dataset. You should always look through this output for conflicts between changes you made and changes other people made. CVS will merge changes within the same file as long as they aren't on the same line. If you and someone else tried to change the same line, CVS will mark it within the file with lines containing many angle-brackets. You must find them and fix the conflict.

- **commit (checkin)**

This is how you put your changes back into the repository. Any files you have modified or added (or removed) are put into the repository as new versions of those files. You can commit individual files, portions of your working dataset, or your entire working dataset. You will be asked for a comment, and you should *always* provide a detailed comment that describes the changes you have made and why they were made. This comment will be attached to all the files you changed, so it's best to commit just one or a few files at a time so you can give each one a useful comment.

It is always a good idea to update the files you have changed before trying to commit them. Doing this will show you if there are any conflicts with changes made by others.



CVS Basic Operations: 3

- **Diff**

This is how you can see the differences between what is in your working dataset and what is in the repository. It's very useful for resolving conflicts. You can diff individual files, or many files (which just diffs each file sequentially). Diffing many files together generates output that is difficult to read and comprehend. There are options that let you specify a particular version within the repository to compare your file to.

- **Status**

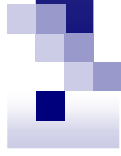
This will tell you what version of a file is in your working dataset, and other information about the file with respect to the repository. Use this to see what versions of files you are working with.

- **Log**

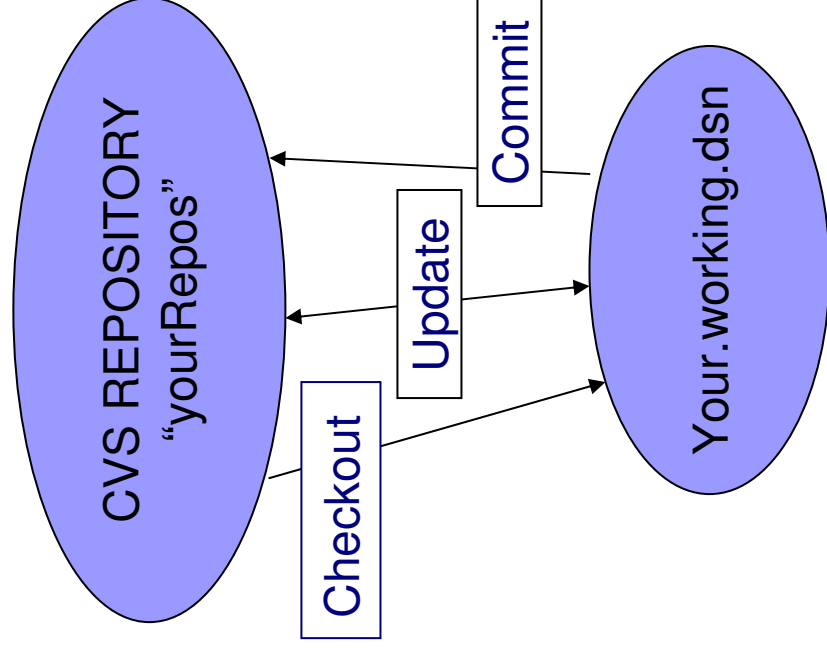
This will list every version of a file (or set of files), telling you who made the change and showing you the comment they added that describes what the change was and why they made it. Use this to track down why changes were made to files.

- **History**

Tells you what changes that have been made to the entire repository. You can get a summary of specific operations done by specific users.



CVS FLOW





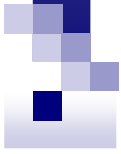
Userid Definitions

- On TSO, your userid has a UID assigned to it. This value should be the same value as the UID assigned to the ldapDomain id (which is same as the Windows Domain id. This has the result of
 pdbate has the same identity as lbates
- Anytime you ‘authenticate’ / (login) to a repository, do so with your “primary id”. A file is saved in your USS home directory containing the string
 :pserver:lbates@rscvs.rocketsoftware.com:2401/cvs/YOURREPOS encryptPW
- Most people should leave the ldapDomain id presented on the panel and just enter the LDAP password.



Initial Startup Possibilities

- V;C from primary menu. You have never authenticated to any repository yet. You will see Repository Login screen.
- V;C and you have previously used cvsISPF. You will see cvsISPF main menu, with the 'last used' Repository active – in top right corner.
- **TSO CVSISPF otherRepos** . If you have authenticated to otherRepos, that repository becomes the active one.
- Only one Repository can be active at any given time, but you can easily switch between repositories using the E – Activate Repository Menu.



Login

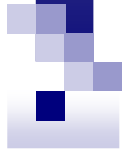
If you type the correct password, you will then be browsing a "results file"

```
***** Top of Data *****
debug: retcode = 0
(Logging in to lbates@rscvs.rocketsoftware.com)
***** Bottom of Data *****
```

```
- zCVS v.1.0 ----- Active repository ID:
Login Repository
Enter command ==>
-----
Repository details
Repository ID: yourRepos
User name: lbates
Password:..... yourLDAPpass
/ Activate after login

Server details
Host name: rscvs.rocketsoftware.com
Port number:
```

After successful login, All subsequent panels carry **yourRepos** in top right corner



MetaData Attributes: Used When Adding Modules

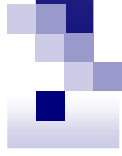
The Project Lead must do the initial Add of the modules (Low Level Qualifiers) to the repository.

Once modules are added any team member can change attributes per module.

The following are the attributes that are relevant (**default**)

- Dataset Type: **PDS** | PDSE
 - Record Format: **FB** F FBA V VB
 - Record Length: **>= 80** <= 255
 - Allocation Type: **TRK** CYL
 - Primary Allocation: **15** (valid range >0)
 - Secondary Allocation: **5** (valid range >0)
 - Directory Blocks: **20** (valid range >0)
 - Strip Sequence Numbers: **N** Y
 - Data Content: Text or Binary **T** B
- Used when allocating new datasets
- Used any time content sent to repository





Dilema: To productize (or OpenSource) or not ?

- We rely on and expect a number of Rocket specific infrastructure resources.
- The performance needs to be improved, I am confident that it would benefit from a TSO ooRexx overhaul ... May the discussion prosper and continue !!!
- How to deal with a “SourceForge” repository and the resulting deep repository tree.