

Using the Concurrent Versions System

LINUX ADVANCED LECTURES *Berlin, 5. September 2001*

MATTHIAS KRANZ

mskranz@acm.org

Berlin Unix User Group

Agenda

- ✍ Introduction
- ✍ Overview of CVS
- ✍ CVS Administration
- ✍ Advanced CVS
- ✍ Frontends and Tools
- ✍ Resources
- ✍ Questions & Answers

Introduction

Version Control System

- ✍ Only *one part* of a thorough Software Configuration Management Process
- ✍ What is its purpose?
 - ⇒ Tracking control of the different states of a file
 - ⇒ Easy restoring of a particular version
- ✍ What is not its purpose?
 - ⇒ No builtin mechanisms for interaction with you build environment

<POLITICS>

Why is it so important to use a free Version Control System?

</POLITICS>

Background

- ✍ Revision Control System (RCS) by Walter F. Tichy (early 1980s at Purdue)
- ✍ CVS was a bunch of shell scripts written by Dick Grune (1986)
- ✍ Rewritten in C by Brian Berliner (1989), a lot of contributions by Jeff Polk
- ✍ Jim Kingdon added network capabilities
- ✍ 1999 `Cyclic.com` was purchased by Source Gear
- ✍ Feb 2, 2000 Open Avenue aquired `Cyclic.com` -> `cvshome.org`
- ✍ `cvshome.org` is now hosted by Collab.net

Overview

Basics

- ✍ A significant amount of VCs use the *lock-modify-unlock* approach
- ✍ CVS **does not!** Really.
- ✍ Instead the CVS model looks like *copy-modify-merge*
- ✍ Resulting conflicts must be resolved before committing to the repository

Structure Of A CVS Command

cv	[cv	cv	[com	[com
----	-----	----	------	------

Examples:

- ✎ `cv` `-d` `:pserver:anonymous@cvs.gnome.org:/cvs/gnome` `login`
- ✎ `cv` `update` `-dP`
- ✎ `cv` `-Q` `-d` `/usr/local/repository` `release` `-d` `project`
- ✎ CVS provides about 25 subcommands with around 150 options, so do not try to remember all those things
- ✎ Just use
 - `$` `cv` `help`
 - `$` `cv` `<subcommand>` `help-options`

Accessing A Repository

✍ Specify the CVSROOT env variable^a

✍ Local access:

```
bob@seth:$ export CVSROOT=/usr/local/newrepository
```

✍ pserver access

```
bob@seth:$ export CVSROOT=:pserver:bob@seth:/usr/local/newrepository
```

```
bob@seth:$ cvs login
```

```
(Logging in to bob@seth)
```

```
CVS password:
```

✍ Remote shell (use SSH^b)

```
bob@seth:$ export CVS_RSH=ssh
```

```
bob@seth:$ export CVSROOT=:ext:bob@seth:/usr/local/repository
```

^aOr use the cvs option -d

^bAnd ssh-agent, ssh-add

Starting A New Project

- ✎ Assume you are working on the following project

```
bob@seth:~$ ls -R project/  
project/  
README.txt  pix  src
```

```
project/pix:  
mozilla.png  netscape.png
```

```
project/src:  
Makefile  hello.c  hello.o  helloWorld
```

- ✎ Text files, source files, binary data, ...

Importing A Project To The Repository

✎ The first real CVS command: `import`

```
bob@seth:$ cvs import -m "New project\; First import" \  
-W*.png -kb project bob start  
I project/src/hello.o  
N project/README.txt  
cvs server: Importing /usr/local/newrepository/project/src  
N project/src/Makefile  
N project/src/hello.c  
N project/src/helloWorld  
cvs server: Importing /usr/local/newrepository/project/pix  
N project/pix/mozilla.png  
N project/pix/netscape.png  
  
No conflicts created by this import
```

✎ Some files got a special treatment: `hello.o` was ignored and all `*.png` files were treated as binary data

Checking Out A Working Copy

✍ Prepare your own sandbox

```
bob@seth:$ cvs checkout project
cvs server: Updating project
U project/README.txt
cvs server: Updating project/pix
U project/pix/mozilla.png
U project/pix/netscape.png
cvs server: Updating project/src
U project/src/Makefile
U project/src/hello.c
U project/src/helloWorld
```

✍ And now: Just hack away ...

CVS Subdirs

✎ What are those CVS dirs in every project dir for?

```
bob@seth:$ cd project
bob@seth:$ ls
CVS/          src/
README.txt    pix/
bob@seth:$ cd CVS
bob@seth:$ ls
Entries       Root
Repository
bob@seth:$ cat Root
:pserver:bob@seth:/usr/local/newrepository
bob@seth:$ cat Repository
project
bob@seth:$ cat Entries
D/pix/////
D/src/////
/README.txt/1.1.1.1/Mon Mar 12 04:29:05 2001//
```

Using update

- ✎ We want to get the newest stuff^a

```
bob@seth:$ cvs update -dP
cvs server: Updating .
cvs server: Updating pix
cvs server: Updating src
M src/hello.c
```

- ✎ `-d` will bring new directories from the repository to your working copy
- ✎ `-P` will *prune* empty directories
- ✎ I often use the following construction to first see, what will happen, which won't change anything physically

```
bob@seth$ cvs -n update -dP
```

^aIf any data is merged, your original files are renamed with a preceding `.#`

Update Output

U <i>file</i>	Your copy is not modified but older than the one in the repository.
P <i>file</i>	State as above, but the server sends a patch instead of an entire <i>file</i> .
A <i>file</i>	New <i>file</i> in your working dir. Needs to be committed if it should be under control.
R <i>file</i>	<i>file</i> has been removed in your working dir. Needs to be committed if it should be removed in the repository, too.
M <i>file</i>	<i>file</i> has been modified, either only in your working dir or in the repository also but merging was successfully without conflicts.
C <i>file</i>	A conflict was detected while trying to merge your changes with changes from the repository.
? <i>file</i>	<i>file</i> is totally unknown to CVS.

Conflicts

- ✎ In most cases, conflicts arise due to a lack of communication
- ✎ (At least) two developers changed the same region in the same file

```
bob@seth:$ cvs update hello.c
RCS file: /usr/local/newrepository/project/src/hello.c,v
retrieving revision 1.1.1.1
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2 into hello.c
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in hello.c
C hello.c
```

Resolving Conflicts

✎ Edit the conflicting file again

```
#include <stdio.h>

int
main(int argc, char **argv)
{
    printf("What a boring example.\n");
    printf("Hello, World\n");
<<<<<<< hello.c
    printf("Want to buy an A?\n");
=====
    printf("Nice weather, isn't it?\n");
>>>>>>> 1.2

    return 0;
}
```

✎ Decide what to do, remove those CVS insertions and commit again

Adding A File

```
bob@seth:$ cvs add ChangeLog
cvs server: scheduling file `ChangeLog' for addition
cvs server: use 'cvs commit' to add this file permanently
bob@seth:$ cat CVS/Entries
D/pix////
D/src////
/README.txt/1.1.1.1/Mon Mar 12 04:43:14 2001//
/ChangeLog/0/dummy timestamp//
bob@seth:$ cvs commit ChangeLog
RCS file: /usr/local/newrepository/project/ChangeLog,v
done
Checking in ChangeLog;
/usr/local/newrepository/project/ChangeLog,v  <--  ChangeLog
initial revision: 1.1
done
```

Deleting A File

- ✍ First of all: Physically remove the bits ...

```
bob@seth:$ rm JustADemoDummy
bob@seth:$ cvs remove JustADemoDummy
cvs server: scheduling 'JustADemoDummy' for removal
cvs server: use 'cvs commit' to remove this file permanently
bob@seth:$ cvs ci JustADemoDummy
Removing JustADemoDummy;
/usr/local/newrepository/project/JustADemoDummy,v <-- JustADemoDummy
new revision: delete; previous revision: 1.1
done
```

- ✍ The file would not be removed from the repository
- ✍ Its moved to the special Attic directory

Getting Status Information

```
bash$ cvs -Q status
```

```
=====
```

```
File: Makefile          Status: Needs Patch
```

```
Working revision:      1.2      Mon Mar 12 13:46:29 2001
```

```
Repository revision:  1.3      /usr/local/newrepository/project/Makefile,v
```

```
=====
```

```
File: README           Status: Needs Merge
```

```
Working revision:      1.1.1.1 Mon Mar 12 16:45:19 2001
```

```
Repository revision:  1.2      /usr/local/newrepository/project/README,v
```

```
=====
```

```
File: main.c           Status: Up-to-date
```

```
Working revision:      1.1.1.1 Mon Mar 12 13:29:03 2001
```

```
Repository revision:  1.1.1.1 /usr/local/newrepository/project/src/main.c,v
```

```
...
```

Viewing The Differences

- ✎ What exactly would happen, if I update a particular file

```
bob@seth:$ cvs -Q diff -c hello.c
RCS file: /usr/local/newrepository/project/src/hello.c,v
retrieving revision 1.4
diff -c -r1.4 hello.c
*** hello.c 2001/03/20 09:50:03 1.4
--- hello.c 2001/03/20 10:15:38
*****
*** 5,11 ****
    {
        printf("What a boring example.\n");
        printf("Hello, World\n");
!       printf("Want to buy an A?\n");

        printf("Added new code, like this printf\n");
```

Continued on next slide

Viewing The Differences (cont.)

```
--- 5,11 ----
{
    printf("What a boring example.\n");
    printf("Hello, World\n");
!   printf("Want to buy an A?\n");

    printf("Added new code, like this printf\n");
```

Viewing The Logs Of Your Project

✎ Browsing through the (hopefully) meaningful log messages

```
bob@seth$ cvs log hello.c
revision 1.2
date: 2001/03/12 08:47:04; author: alice; state: Exp; lines: +2 -0
Committed first
-----
revision 1.1
date: 2001/03/12 04:09:09; author: bob; state: Exp;
branches: 1.1.1;
Initial revision
-----
revision 1.1.1.1
date: 2001/03/12 04:09:09; author: bob; state: Exp; lines: +0 -0
New project\; First import
-----
revision 1.3.2.1
date: 2001/03/12 09:48:07; author: alice; state: Exp; lines: +0 -1
Contained a sentence which does not belong to customers.
```

Tagging

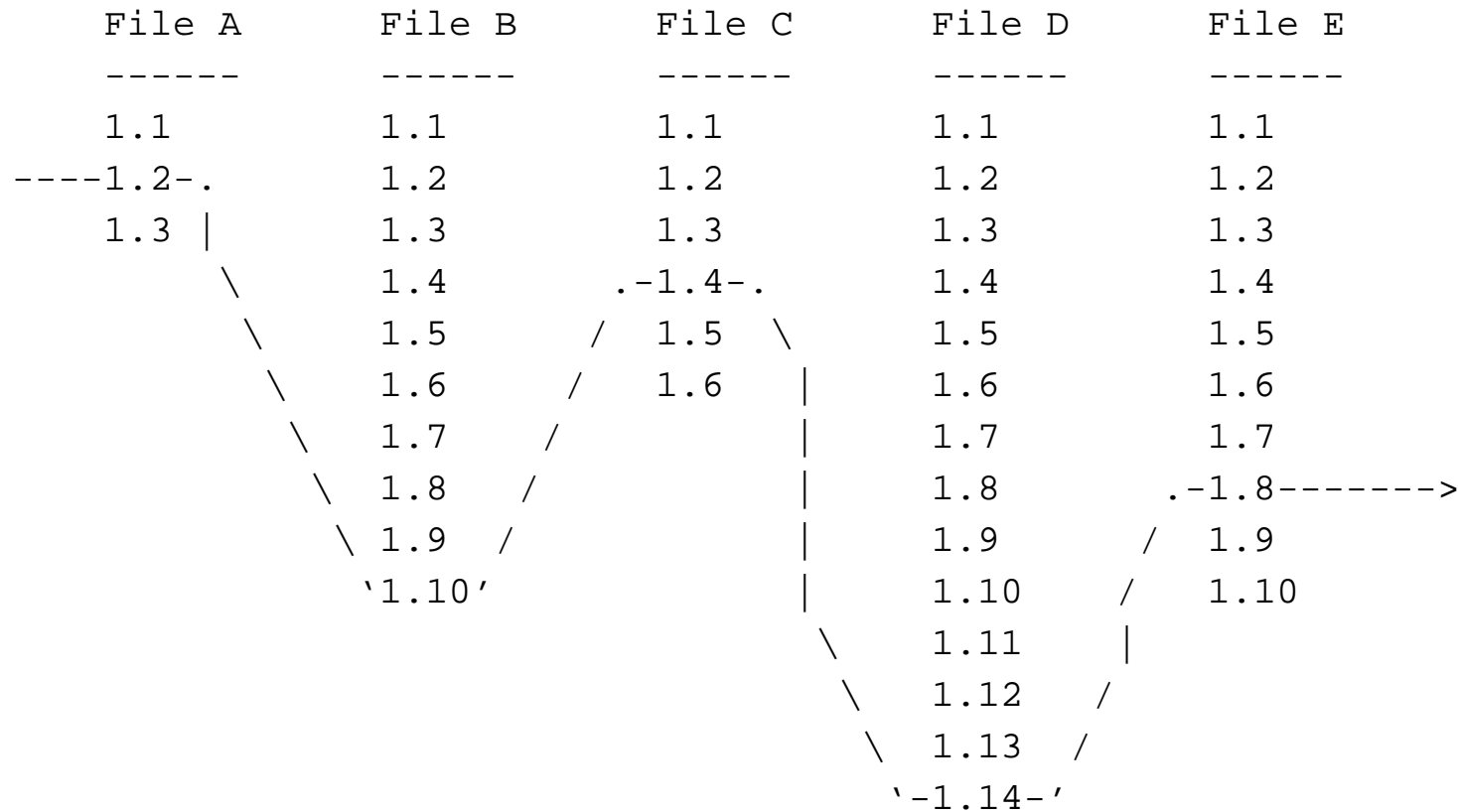
- ✎ Could you remember the revision numbers of several hundreds of files?

```
bob@seth:~$ cvs tag Example-Tag-0-0-1
cvs server: Tagging .
T ChangeLog
T README.txt
cvs server: Tagging pix
T pix/mozilla.png
T pix/netscape.png
T pix/opera.png
cvs server: Tagging src
T src/Makefile
T src/hello.c
T src/helloWorld
```

- ✎ All those files are now accessible via the symbolic tag

Tagging (cont.)

✎ A visualization might help understanding



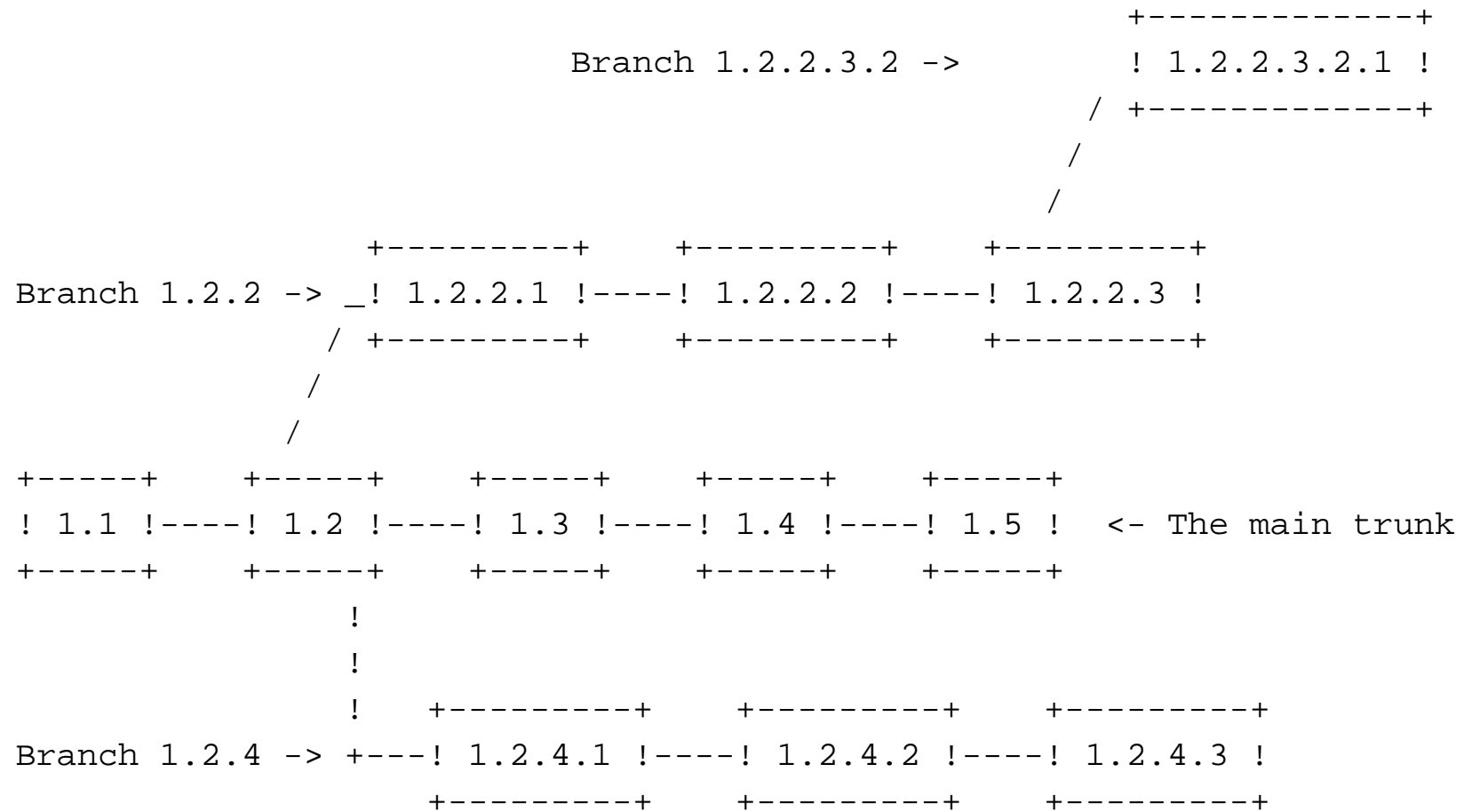
Branching

- ✍ This is what a normal revision history looks like

```
+-----+      +-----+      +-----+      +-----+      +-----+
! 1.1 !-----! 1.2 !-----! 1.3 !-----! 1.4 !-----! 1.5 !
+-----+      +-----+      +-----+      +-----+      +-----+
```

- ✍ However, sometimes you want to split up your development, e.g.
 - ⇒ for fixing a bug
 - ⇒ developing a crazy new feature

Branching (cont.)



Branching (cont.)

- ✎ Use a base tag for every branch point

```
bob@seth:$ cvs tag Root-of-Release-0-5-Bugfixes
```

- ✎ And branch off that tagged point

```
bob@seth:$ cvs tag -b Release-0-5-Bugfixes-Branch
cvs server: Tagging .
T ChangeLog
T README.txt
cvs server: Tagging pix
T pix/mozilla.png
T pix/netscape.png
T pix/opera.png
cvs server: Tagging src
T src/Makefile
T src/hello.c
T src/helloWorld
```

Working With Branches

- ✍ Get a branched version of the project

```
alice@seth:$ cvs update -r Release-0-5-Bugfixes-Branch
```

- ✍ View at the status ...

```
...  
File: hello.c           Status: Up-to-date  
  
Working revision:      1.3  
Repository revision:  1.3  
/usr/local/newrepository/project/src/hello.c,v  
Sticky Tag:           Release-0-5-Bugfixes-Branch (branch: 1.3.2)  
Sticky Date:          (none)  
Sticky Options:       (none)  
...
```

Committing On Branches

- ✍ Commit your changes as usual

```
alice@seth:$ cvs ci src/hello.c
Checking in src/hello.c;
/usr/local/newrepository/project/src/hello.c,v <-- hello.c
new revision: 1.3.2.1; previous revision: 1.3
done
```

- ✍ CVS will automatically commit on the branch, since there exists a *sticky tag* in your local working copy
- ✍ This can be checked with the `status` command

Merging Branches

- ✍ You want to merge a branch with the main trunk

```
bob@seth:$ cvs update -j Release-0-5-Bugfixes-Branch
cvs server: Updating .
cvs server: Updating pix
cvs server: Updating src
RCS file: /usr/local/newrepository/project/src/hello.c,v
retrieving revision 1.3
retrieving revision 1.3.2.1
Merging differences between 1.3 and 1.3.2.1 into hello.c
```

- ✍ If there are no conflicts, you could just commit the merged version

CVS Administration

Setting Up A Repository

- ✎ Initialize the repository, i.e. creates the directory and some administrative data^a

```
seth:~# cvs -d /usr/local/newrepository/ init
```

- ✎ Change the group ownership to cvs and ...

```
seth:~# chgrp -R cvs /usr/local/newrepository/  
seth:~# chmod ug+rx /usr/local/newrepository/CVSROOT/
```

- ✎ ... put every potential CVS user in the cvs group

^arequires write access to the destination of course

Setting Up A Password Authenticating Server

✎ Adjust /etc/inetd.conf

```
cvspserver stream tcp nowait root /usr/local/bin/cvs cvs \  
--allow-root=/usr/local/newrepository pserver
```

✎ Adjust /etc/services

```
2401/cvspserver
```

✎ Do not forget to restart your inetd

✎ Set up CVSROOT/passwd (optional)

```
#General structure  
#UserName:Password(encrypted):SystemUserName(opt.)  
mskranz:cvKnEg5uw0aq2  
joe:cvPX55NZtCTMs:cvs1  
anonymous:cvdQwS9zjbK/U
```

The CVSROOT/ Directory

- ✍ Contains CVS configuration files
- ✍ Should be maintained with CVS means

```
mskranz@seth:~$ cvs -d /usr/local/newrepository/ co CVSROOT
cvs checkout: Updating CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
U CVSROOT/cvswrappers
U CVSROOT/editinfo
U CVSROOT/logininfo
U CVSROOT/modules
U CVSROOT/notify
U CVSROOT/rcsinfo
U CVSROOT/taginfo
U CVSROOT/verifymsg
```

modules

- ✍ **modules: Defining alternate groupings or aliases for your projects**

```
MyProject project  
MySrc project/src
```

- ✍ **Even usable for single files**

```
BobsPart project/README.txt project/src/Makefile
```

- ✍ **An ampersand & allows you to define alternative names**

```
CustomerEdition &BobsPart
```

Advanced CVS

Binary Data

- ✍ CVS supports handling of binary files ... more or less.

```
bob@seth:$ cvs add -kb opera.png
bob@seth:$ cvs ci -m "This is a new file, containing binary data."
opera.png
RCS file: /usr/local/newrepository/project/pix/opera.png,v
done
Checking in opera.png;
/usr/local/newrepository/project/pix/opera.png,v  <--  opera.png
initial revision: 1.1
done
```

- ✍ CVS will save a full copy for every revision
- ✍ If CVS did not recognize the appropriate handling automatically, you can adjust it with

```
bob@seth:$ cvs admin -kb opera.png
```

Using Keywords

- CVS supports the automatic expansion of several keywords, e.g. `Id` would expand to

```
$Id: Makefile,v 1.2 2001/03/12 15:10:08 bob Exp $
```

Date	Date of last commit
Author	Author of last commit
Id	Filename, revision, date, and author
Revision	Revision after last commit
Source	Path to corresponding repository file
Log	Accumulating the log messages

Zooming in the history

- ✍ annotate shows who has touched which line in a file

```
bob@seth:$ cvs annotate hello.c
Annotations for hello.c
*****
1.1          (bob      12-Mar-01): #include <stdio.h>
1.1          (bob      12-Mar-01):
1.1          (bob      12-Mar-01): int
1.1          (bob      12-Mar-01): main(int argc, char **argv)
1.1          (bob      12-Mar-01): {
1.2          (alice    12-Mar-01):     printf("What a boring
example.\n");
1.1          (bob      12-Mar-01):     printf("Hello, World\n");
1.3          (bob      12-Mar-01):     printf("Want to buy an A?\n");
1.1          (bob      12-Mar-01):
1.4          (bob      12-Mar-01):     printf("Added new code, like
this printf\n");
1.4          (bob      12-Mar-01):
1.1          (bob      12-Mar-01):     return 0;
1.1          (bob      12-Mar-01): }
```

Watches

✍ Uncomment the last line in `CVSROOT/notify` which is responsible for mailing notifications to users

✍ These users should be listed in `CVSROOT/users`

```
mskranz:mskranz@acm.org  
bob:bob@gromit.in-berlin.de  
alice:alice@gromit.in-berlin.de
```

✍ Set a watch, if you want to be notified

```
alice@seth$ cvs watch add ChangeLog
```

✍ Other developers must use `edit` before they start

```
bob@seth$ cvs edit ChangeLog
```

✍ A `commit` will remove you from the editor's list

✍ An `unedit` would be used if you don't want to `commit` nor to `edit` the file any longer

Watches (cont.)

- ✎ An unedit on a modified file would lead to the following situation

```
bob@seth$ cvs unedit hello.c
hello.c has been modified; revert changes? y
```

- ✎ Your changes would be reverted and all watchers be notified that you are not editing the file anymore
- ✎ With the option `-a` you can specify which specific actions you want to watch
- ✎ Want to see who is watching what?

```
bob@seth$ cvs watchers hello.c
hello.c alice  edit unedit  commit
```

Frontends and Tools

pcl-cvs – An Emacs Interface

- ✎ Original author: Per Cederqvist
Current Maintainer: Stefan Monnier
`ftp://rum.cs.yale.edu/pub/monnier/pcl-cvs/`
- ✎ Native VC Interface vs. `pcl-cvs`
- ✎ VC was designed for several backends, not particular for CVS
- ✎ VC's view is a more file-based view
- ✎ Disadvantage of `pcl-cvs` is its tricky installation

Graphical Frontends

- ✍ tkCVS – Tcl/Tk Client (<http://tkcvs.sourceforge.net>)
- ✍ jCVS – Java Client (<http://www.jcvs.org>)
- ✍ Pharmacy – Gnome Client (<http://pharmacy.sourceforge.net>)
- ✍ linCVS – KDE Client (<http://ppprs1.phy.tu-dresden.de/~trogisch/lincvs/lincvsen.html>)
- ✍ ...
- ✍ WebCVS – Web-Frontend, only for browsing
(<http://www.cyclic.com/cyclic-pages/web-cvsweb.html>)
- ✍ Bonsai – Mozilla's Web-Frontend
(<http://www.mozilla.org/bonsai.html>)
- ✍ ...

Resources

Resources

- ✍ The Central Site
 - ⇒ <http://www.cvshome.org>
- ✍ Pascal Molli's CVS bubbles
 - ⇒ <http://www.loria.fr/~molli/cvs-index.html>
- ✍ "Version Management with CVS" by Per Cederqvist et al
 - ⇒ included also in the source distribution of CVS
- ✍ "Open Source Development with CVS" by Karl Fogel
 - ⇒ <http://cvsbook.red-bean.com/>
- ✍ Mailing-List and Usenet
 - ⇒ info-cvs-request@gnu.org
 - ⇒ comp.software.config-mgmt

Q&A Session

Thank you!

Matthias Kranz

mskranz@acm.org

<http://www.buug.de/~mkr/>