Version Control Systems

CVS & Co

Georg Martius

Institute für Informatik
Universität Leipzig
<georg.martius@web.de>

7th April 2005

Outline

What is CVS?

- CVS: Concurrent Versions System
- Manages files for concurrent editing
- Keeps a history of all changes
- Provides access to all prior versions a file
- Safety net

Where do I need it?

- Software projects:
  - Multiple developers
  - Versioning
  - Branches
    - Maintain patches for older releases, while
    - Working on current version.
- Papers / Documents:
  - Many authors
  - Single author
    - Different working locations (Office/Home)
    - Reviser is correcting while last minute changes are done

Where do I need it?

- Software projects:
  - Multiple developers
  - Versioning
  - Branches
    - Maintain patches for older releases, while
    - Working on current version.
- Papers / Documents:
  - Many authors
  - Single author
    - Different working locations (Office/Home)
    - Reviser is correcting while last minute changes are done

Outline

Basics

- Client - Server based
- Server side storage is called *Repository*, identified with a **Path** (cvsroot): `[:method:][[user]@]hostname:/path/to/reposit`
- Modules are directories inside the repository, identified with a **Name**, such as `box`
- General procedure
  - creating a module (once **import**)
  - get a local copy (**checkout**)
  - change some files
  - check for changes from others (**update**)
  - make changes persistent (**commit**)

Basics

- Client - Server based
- Server side storage is called *Repository*, identified with a **Path** (cvsroot):
  `[:method:][[user]@]hostname:/path/to/reposit`
- Modules are directories inside the repository, identified with a **Name**, such as `box`
- General procedure
  - creating a module (once **import**)
  - get a local copy (**checkout**)
  - change some files
  - check for changes from others (**update**)
  - make changes persistent (**commit**)

Basics

- Client - Server based
- Server side storage is called *Repository*, identified with a **Path** (cvsroot): `[:method:][[user]@]hostname:/path/to/reposit`
- Modules are directories inside the repository, identified with a **Name**, such as `box`
- General procedure
  - creating a module (once **import**)
  - get a local copy (**checkout**)
  - change some files
  - check for changes from others (**update**)
  - make changes persistent (**commit**)

Basics

- Client - Server based
- Server side storage is called *Repository*, identified with a **Path** (cvsroot): `[:method:][[user]@]hostname:/path/to/reposit`
- Modules are directories inside the repository, identified with a **Name**, such as `box`
- General procedure
  - creating a module (once **import**)
  - get a local copy (**checkout**)
  - change some files
  - check for changes from others (**update**)
  - make changes persistent (**commit**)

Our Example Scenario

```
$ ls -R box
    box:
    doc.tex prog.c
```

import

```
$ cd box
box$ cvs import box me start
    No conflicts created by this import
box$ cd ..
$ rm -r box
```

Our Example Scenario

```
$ ls -R box
    box:
    doc.tex prog.c
```

import

```
$ cd box
box$ cvs import box me start
    No conflicts created by this import
box$ cd ..
$ rm -r box
```

Checkout
Getting a working copy

Specify CVSROOT

```
$ export CVSROOT=":ext:me@machine:/home/cvsroot"
```

checkout

```
$ cvs checkout box
   cvs server:  Updating box
   U box/doc.tex
   U box/prog.c
$ ls box
   CVS prog.c doc.tex
```

Checkout
Getting a working copy

Specify CVSROOT

```
$ export CVSROOT=":ext:me@machine:/home/cvsroot"
```

checkout

```
$ cvs checkout box
    cvs server:  Updating box
    U box/doc.tex
    U box/prog.c
$ ls box
    CVS prog.c doc.tex
```

Update
Getting the newest version

Assume we have edited the file `doc.tex`.

status

```
$ cd box
box$ cvs status doc.tex
   ===================================================================
   File:  doc.tex Status:  Locally Modified
   Working revision:  1.1
   Repository revision:  1.1
```

update

```
box$ cvs update
   cvs server:  Updating .
   M doc.tex
```

Update
Getting the newest version

Assume we have edited the file `doc.tex`.

status

```
$ cd box
box$ cvs status doc.tex
   ===================================================================
   File:  doc.tex Status:  Locally Modified
   Working revision:  1.1
   Repository revision:  1.1
```

update

```
box$ cvs update
   cvs server:  Updating .
   M doc.tex
```

Commit

Make changes persistent and visible for the rest

commit

```
box$ cvs commit

  cvs commit: Examining .
  Checking in doc.tex;
  /home/cvsroot/box/doc.tex,v  <--  doc.tex
  new revision: 1.2; previous revision: 1.1
  done
  Checking in prog.c;
  /home/cvsroot/box/prog.c,v  <--  prog.c
  new revision: 1.2; previous revision: 1.1
  done
```

Commit comments are requested.

Add/Remove

add

```
box$ mkdir utils
box$ touch utils/string.h
box$ cvs add utils
box$ cd utils
box/utils$ cvs add string.h
```

remove

```
box/utils$ cvs remove -f string.h
```

`-f`: force → deletes the file

"new" update

```
box$ cvs update -dP
```

`-d`: create directories `-P`: prune empty directories

Add/Remove

add

```
box$ mkdir utils
box$ touch utils/string.h
box$ cvs add utils
box$ cd utils
box/utils$ cvs add string.h
```

remove

```
box/utils$ cvs remove -f string.h
```

`-f`: force → deletes the file

"new" update

```
box$ cvs update -dP
```

`-d`: create directories `-P`: prune empty directories

Add/Remove

add

```
box$ mkdir utils
box$ touch utils/string.h
box$ cvs add utils
box$ cd utils
box/utils$ cvs add string.h
```

remove

```
box/utils$ cvs remove -f string.h
```

`-f`: force → deletes the file

"new" update

```
box$ cvs update -dP
```

`-d`: create directories `-P`: prune empty directories

Outline

Example Code

Version 1.2

Prog.c

```
#include <stdio.h>
void main(){
  printf("Hallo World");
}
```

Merging
Concurrency and the consequences

Alice's version

```
#include <stdio.h>
#include <stdlib.h>
void main(){
  printf("Hallo World");
  printf("% i", rand());
}
```

Bob's version

```
#include <stdio.h>
int main(){
  printf("Hallo World");
}
```

Alice commits

```
box$ cvs commit

  /home/cvsroot/box/prog.c,v  <--  prog.c
  new revision: 1.3; previous revision: 1.2
  done
```

Merging
Concurrency and the consequences

Alice's version

```
#include <stdio.h>
#include <stdlib.h>
void main(){
  printf("Hallo World");
  printf("% i", rand());
}
```

Bob's version

```
#include <stdio.h>
int main(){
  printf("Hallo World");
}
```

Alice commits

```
box$ cvs commit

  /home/cvsroot/box/prog.c,v  <--  prog.c
  new revision: 1.3; previous revision: 1.2
  done
```

Merging (cont.)
Bob needs to update before commit

Bob updates

```
box$ cvs update -dP

  RCS file: /home/cvsroot/box/prog.c,v
  retrieving revision 1.2, retrieving revision 1.3
  Merging differences between 1.2 and 1.3 into prog.c
  cvs server: conflicts found in prog.c
  C prog.c
```

Merging (cont.)
Bob needs to resolve the conflict

prog.c (merged with conflicts)

```
#include <stdio.h>
  <<<<<<< prog.c
  int main(){
  =======
  #include <stdlib.h>
  void main(){
  >>>>>>> 1.3
      printf("Hallo World");
      printf(" %i", rand());
  }
```

After resolving the conflict Bob can **commit** his version.
Conflicts are usually a communication problem!

Outline

Install Repository
Administation

- setup of a repository has to be done once
- usually done as `root`
- create a directory that gets backuped e.g. `/home/cvsroot`
- create a group `cvs`

init

```
$ mkdir /home/cvsroot
$ export CVSROOT="/home/cvsroot"
$ cvs init
$ chgrp -R cvs $CVSROOT
$ chmod -R g+w $CVSROOT
```

Authentication
Remote Access Method

pserver (`:pserver:`)

- CVS `passwd` file with possible mapping to system user
- read-only access possible
- plaintext password transmission (insecure)
- useful for anonymous read-only access

ssh (`:ext:`)

- requires system user
- uses secure shell for any interaction
- requires
  `$ export CVS_RSH="ssh"`

pserver (`:pserver:`)

- CVS `passwd` file with possible mapping to system user
- read-only access possible
- plaintext password transmission (insecure)
- useful for anonymous read-only access

ssh (`:ext:`)

- requires system user
- uses secure shell for any interaction
- requires
  `$ export CVS_RSH="ssh"`

Keyword Substitution

- `$keyword:...$` is automatic replaced on **commit**
- comment style of programming language used
- Important keywords:
  - `$Id$`
  - `$Log$`

Example

```
/* $Id: prog.c,v 1.7 2005/04/03 17:34:06 georg Exp $ */
// $Log: prog.c,v $
// Revision 1.7  2005/04/03 17:34:06  georg
// keywords inserted
//
#include <stdio.h>
...
```

Tags

- Any file can carry multiple tags at each revision
- Tags are used for:
  - Releases
  - Branches

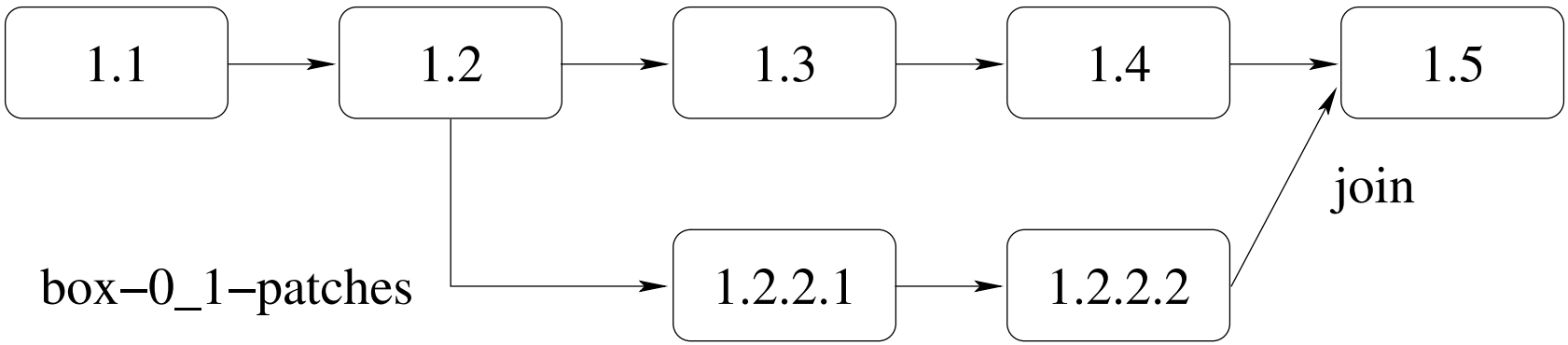
tag

```
box$ cvs tag box-0_1 .
$ cvs checkout -r box-0_1 box
```

Branches

create a branch

```
box$ cvs tag -b box-0_1-patches
```



merging branches

```
$ cvs checkout box  (revision 1.4)
$ cd box
box$ cvs update -j box-0_1-patches
```

Ignored Files and Binary Files

- `cvs update` prints for any file it doesn't know a line to warn you
- CVS ignores by default files like `*.o, *.bak, CVS, *~ , ...`
- Placed that are added up to the list of ignored files:
  - Per-repository list in `$CVSROOT/CVSROOT/cvsignore`
  - Per-user list in `$HOME/.cvsignore`
  - Environment variable `$CVSIGNORE`
  - Inside the directory tree a `.cvsignore` is valid for the subtree
- A single `!` clears the list
- Binary files should be added with
  `$ cvs add -kb FILE`
  to avoid time consuming diffs

LaTeX & CVS
some hints

- tricky to embed keywords because of $
- package `rcs`
- each sentence and sub-sentence in a new lines (better merges)

Example

```
\documentclass{article}
\usepackage{rcs}
\RCS $Id: doc.tex,v 1.5 2005/04/03 20:53:27 georg Exp $

\begin{document}
\markright{\RCSId}
Long sentences should go on serveral lines
 especially if subordinate clauses are involved
 which can be commented quite easily.
...
```

Outline

Frontends

- `lincvs`
- `cervisia`
- `emacs`
- most IDEs

LinCVS

Also available for M$ Windows

Outline

Subversion
"Improved CVS"

- Considered as
- Nearly the same interface as CVS
- Renaming/moving is properly supported
- Directories are handled as entities
- Truly atomic commits
- Revisions global, not on per-file basis

Arch
Distributed Approach

- Gnu Arch alias `tla`
- No central server
- Commit = branch-merge
- Just on Unix
- Many small programs
- Complicated usage

Darcs

Distributed Approach "Take it easy!"

- No central server - every repository is a server
- Changeset-oriented
- No revision numbers, just tags
- Patches (changes) can be send by SSH/SFTP, HTTP, Email
- New concept, aiming for small projects
- Implemented in Haskell
- Easy to use

Summary

- CVS is useful for nearly everyone
- Assists colleraborative work on text-based documents
- Frontends/IDE-integration provide easy to use
- There are also other version control systems

Thanks!