CVS - Workshop

ByteLABS.org

Igor Böhm

igor@bytelabs.org

Roadmap

1. Was ist CVS?
2. Das CVS Repository
3. Basic CVS usage:
   - Create | Start Repository
   - Import a Project
   - Checkout
   - Update
   - Commit
   - Add
   - Remove
   - Diff
   - Dealing with Conflicts
   - Dates, Sticky Flags, Tags and other Gotchas
4. More advanced CVS usage:
   - Branching and Merging

Was ist CVS?

- *CVS*(*ConcurrentVersionSystem*) stellt ein Frontend für *RCS*(*RevisionControlSystem*) dar.
- Im Gegensatz zu **RCS** ist es bei **CVS** aber nicht noetig *Locks* auf Dateien zu setzen die man bearbeiten will, d.h. die Handhabung in Hinsicht auf mehrere Entwickler ist besser als bei **RCS**.
- Dateien werden *ungelockt ausgecheckt*, können dann bearbeitet werden und erst beim *einchecken* werden eventuell auftretende Konflikte beseitigt.
- CVS verwaltet also eine *history (Versionen)* von source files.

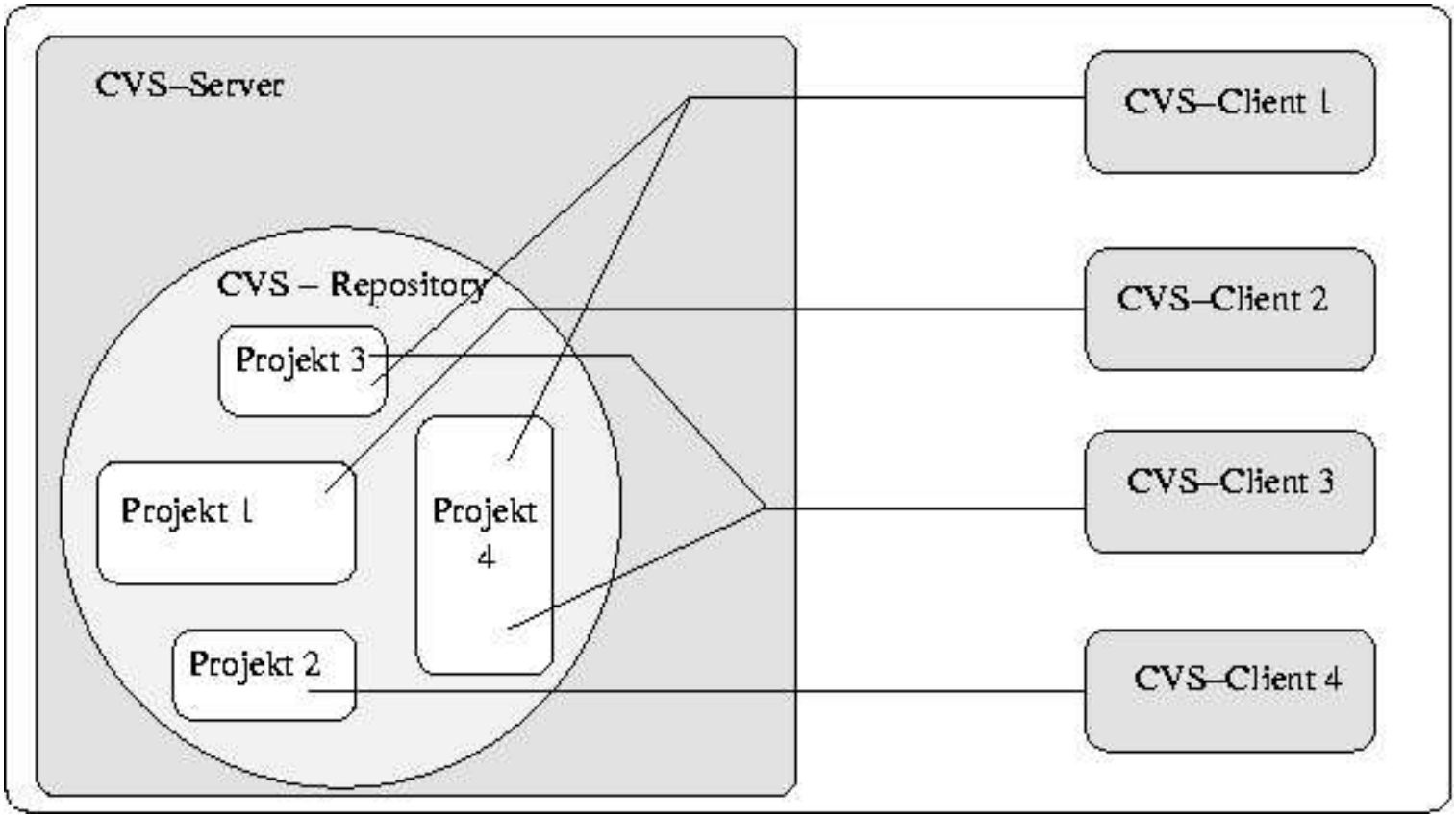
- *Paradigmen:* **Lock-modify-unlock** *vs.* **Copy-modify-merge**(CVS):
  - **Lock-modify-unlock**: Ein Developer holt sich einen exklusiven *write lock* auf das zu editierende File, Verändert das File und wenn die Änderungen abgeschlossen sind, wird der Lock wieder freigegeben und andere Entwickler können das File bearbeiten. Wenn man also ein von jemanden gelocktes File bearbeiten will, muss man zuerst warten bis der Lock freigegeben ist.
  - **Copy-modify-merge**: Ein Developer lädt sich eine Arbeitskopie herunter (*checkout*). Der Developer kann beliebig Files in seiner Arbeitskopie editieren. Während dieser Zeit können alle anderen Developer ebenfalls and den selben Dateien arbeiten. Nach getaner arbeit werden die Änderungen *commited*. Damit werden die Änderungen für alle sichtbar gemacht. Andere können das Repository immer wieder nach Änderungen abfragen und die Arbeitskopie mit dem Repository abgleichen.

- Vorteile von CVS:
  - Durch die Client-Server-Architektur ist verteiltes Programmieren von überall (Internet, Intranet etc.) möglich.
  - Das *uneingeschränkte* Check-out und Versionskontrollsystem vermeidet viele Probleme welche bei exklusiven Check-out Modellen auftreten.
  - Jeder Entwicklungsstand ist wiederherstellbar.
  - Änderungen in Quelltexten können ersichtlich gemacht werden.
  - Bei Auftreten eines Bugs kann man herausfinden, wann sich dieser eingeschlichen hat und wie er entstanden ist.
  - CVS-Client-Tools sind auf *fast* allen Plattformen verfügbar.
  - ...

Das CVS Repository

- CVS verwendet ein zentrales Repository (Behälter), welches sich in einem Verzeichnis auf einem Rechner befindet.
- Grobe Struktur des **CVS** Repository:
  - Pro Projekt ein Verzeichnis.
  - Im Projektverzeichnis wird die Verzeichnisstruktur des Projekts übernommen.
  - Jede Datei wird im jeweiligen Verzeichnis abgespeichert jedoch mit erweiterten Informationen:
    - Versionsnummer
    - Diffs zu vorherigen Versionen
    - Zeitpunkt der Änderung
    - *Kommentare* zu Änderungen von den Entwicklern
    - Name des Entwicklers
    - ...

CVS: Projekt - Repository - Server - Client



Basic CVS Usage

- Einige generelle Fragestellungen und Überlegungen:
  - Wann sollen Änderungen ins Repository übetragen werden?
    * Eine mögliche **Regel** ist nur **konsistente** Zustände ins Repository einzuchecken.
    * Der Head Branch sollte also immer lauffähig sein!
    * Weiters sollte man möglichst immer nur an **einem** Feature zur gleichen Zeit arbeiten.
    * ...
  - Das Arbeitsverzeichnis (Arbeitskopie) kann jederzeit gelöscht werden. Es gehen nur die Änderungen verloren, welche man nach dem letzten commit gemacht hat.
  - Bevor man Änderungen *eincheckt*, stellt sich oft die Frage was man überhaupt alles geändert hat? Mit *CVS* ist es, wie das nächste Beispiel zeigt, ein leichtes alle Änderungen zu protokollieren.

Änderungen Protokollieren

```
</home/../WRKDIR/driver:15>$ cvs diff -bc | tee /tmp/my_diff
Index: driver.c
===================================================================
RCS file: /home/igor/tmp/CVSROOT/driver/driver.c,v
retrieving revision 1.1.1.1
diff -b -c -r1.1.1.1 driver.c
*** driver.c 2 Jun 2004 08:23:15 -0000 1.1.1.1
--- driver.c 2 Jun 2004 09:10:13 -0000
***************                                          10
*** 7,12 ****
          gencode();
        else
          fprintf(stderr, "No code generated.\n");
!       exit(nerr == 0 ? 0 : 1);
        }

--- 7,12 -----
          gencode();
        else                                             20
          fprintf(stderr, "No code generated.\n");
!       exit(nerr == 1 ? 0 : 1);
        }
```

Create | Start a Repository

- Um eine neues **Repository** anzulegen, um darin wiederum verschiedene **Projekte** verwalten zu können, muss man nur an einer beliebigen Stelle ein Verzeichnis anlegen und die Environment Variable CVSROOT darauf zeigen lassen.
- Danach muss nur mehr mit dem Kommando cvs init das **Repository** initialisieren werden.

```
igor@puffy:$ mkdir /home/igor/cvs
igor@puffy:$ cd /home/igor/cvs
igor@puffy:$ export CVSROOT=/home/igor/cvs
igor@puffy:/home/igor/cvs $ cvs init
```

- Nun hat man ein **Repository** zur Verfügung und als nächsten Schritt wird in diesem Repository ein **Projekt** angelegt bzw. ein **Projekt** wird in das **Repository** importiert.

Import a Project

- Es gibt viele Möglichkeiten neue Projekte in ein bestehendes bzw. neu angelegtes Repository einzufügen.
- Man kann sogar Fremd-Software mischen, welche mit eigenen Änderungen bzw. Verbesserungen versehen ist, d.h. *fremde* und *lokale* Änderungen mischen.
- Für das folgende Beispiel hat man entweder schon ein bestehendes Projekt oder man kann auch ein neues Projekt so starten. Wenn man kein bestehendes Projekt hat, dann legt man sich ein Projekt Verzeichnis und evtl. einige Unterverzeichnisse an.
- Dann wechselt man in das Hauptverzeichnis des Projektes und ruft dort das *CVS*-Kommando import auf welches 3 Parameter benötigt:
  - Projektname (Verzeichnisname) unter welchem das Projekt im Repository angelegt werden soll.
  - Vendor Tag (Globaler Versionsbezeichner)
  - Release Tag (Lokaler Versionsbezeichner)

- Der Projektname ist in diesem Fall new_proj, das Vendor Tag ist BASE_01 und das Release Tag ist ist DEV_01:

```
<igor@puffy:/home/igor:9>$ mkdir cvs_proj
<igor@puffy:/home/igor:10>$ cd cvs_proj
<igor@puffy:/home/igor/cvs_proj:11>$ ls
<igor@puffy:/home/igor/cvs_proj:12>$ mkdir src include lib; touch README
<igor@puffy:/home/igor/cvs_proj:13>$ ls
README include/ lib/ src/
<igor@puffy:/home/igor/cvs_proj:14>$ echo $CVSROOT
/home/igor/cvs/
<igor@puffy:/home/igor/cvs_proj:15>$ cvs import new_proj BASE_01 DEV_01
 --> $CVSEDITOR
N new_proj/README
cvs import: Importing /home/igor/cvs/new_proj/src
cvs import: Importing /home/igor/cvs/new_proj/include
cvs import: Importing /home/igor/cvs/new_proj/lib
No conflicts created by this import
```


- Nun muss man das ins Repository importierte Projekt mit dem CVS-Kommando checkout als Arbeitskopie öffnen und kann dann Dateien hinzufügen, ändern oder löschen und die Änderungen dann mit entsprechenden CVS-Kommandos ins Repository übertragen.

CVS Checkout

- Das CVS-Kommando checkout erzeugt eine Arbeitskopie da *CVS* nicht mit gewöhnlichen Verzechnis Bäumen arbeitet. Man muss viel mehr in Verzeichnissen arbeiten die CVS für einen anlegt.
- Es ist wichtig, dass die Umgebungs Variable CVSROOT auf den Pfad des Repositories zeigt. Wenn dies der Fall ist kann man mit dem CVS-Kommando checkout die aktuelle Version des Dateibaums von einem im Repository abgelegten Projekts erzeugen.
- **ACHTUNG:** In **jedem** ausgecheckten Verzeichnis legt *cvs* ein Verzeichnis mit dem Namen **CVS** an. Dieses darf man **nicht** verändert da es CVS selbst benötigt um zu bemerken was sich seit dem letzten checkout verändert hat.

- Checkout des Beispiel Projektes new_proj:

```
<igor@puffy:/home/igor:31>$ echo $CVSROOT
/home/igor/cvs/
<igor@puffy:/home/igor:32>$ cvs checkout new_proj
cvs checkout: Updating new_proj
U new_proj/README
cvs checkout: Updating new_proj/include
cvs checkout: Updating new_proj/lib
cvs checkout: Updating new_proj/src
<igor@puffy:/home/igor:33>$ cd new_proj; ls;
CVS/  README include/ lib/ src/
```

10

- Was befindet sich im **CVS** Verzeichnis?

```
<igor@puffy:/home/igor/new_proj:24>$ ls -l CVS/
total 12
-rw-r--r-- 1 igor igor 73 Jun 3 12:59 Entries
-rw-r--r-- 1 igor igor 9 Jun 2 16:21 Repository
-rw-r--r-- 1 igor igor 16 Jun 2 16:21 Root
<igor@puffy:/home/igor/new_proj:25>$
```

CVS Update

- Das Abgleichen der eigenen Arbeitskopie mit möglichen Änderungen im Repository wird durch das CVS-Kommando update durchgeführt. Hierbei wird geprüft, ob seit dem letzten checkout, Dateien im Repository durch *andere* Benutzer verändert wurden.
- Existieren Änderungen im Repository, versucht *CVS* diese in die Arbeitskopie zu übetragen (*merge*). Bei Konflikten, welche z.B. auftreten können, indem 2 Benutzer in derselben Zeile etwas verändert haben, muss der Benutzer diese zuerst auflösen:

```
<igor@puffy:/home/igor/new_proj:55>$ echo "THIS IS A CHANGE" > README
<igor@puffy:/home/igor/new_proj:57>$ cvs up
cvs update: Updating .
M README
cvs update: Updating include
cvs update: Updating lib
cvs update: Updating src
```

- Jede geänderte Datei mit einem Indikator(Buchstabe am Anfang jeder Zeile) welcher die Art der Änderung beschreibt wird aufgeführt:
  - **(U)pdated:** Änderungen im Repository wurden in die Arbeitsdatei übernommen(*merge*). Keine Konflikte sind aufgetreten.
  - **(M)odified:** In diesem Fall gibt es 2 Möglichkeiten, entw. die Arbeitsdatei wurde verändert od. die Datei im Repository. Im ersten Fall (*Arbeitsdatei wurde geändert*) werden die Änderungen erst bei einem **commit** ins Repository gespielt, im zweiten Fall (*das Repository hat sich geändert*) wurden die Änderungen in die Arbeitsdatei gemischt(*merge*), ohne dass Konflikte aufgetreten sind.
  - **(C)onflicts:** Es sind Konflikte zwischen Änderungen in der Arbeitsdatei und Änderungen im Repository aufgetreten welche aufgelöst(*resolved*) werden müssen.
  - **(A)dded:** Arbeitsdatei wird (nach commit) in Repository übernommen.
  - **(R)emoved:** Arbeitsdatei wird (nach commit) aus dem Repository gelöscht.
  - **(?):** Arbeitsdatei: Keine korrespondierende Datei im Repository.

CVS Commit

- Wenn keine Konflikte mehr vorhanden sind bzw. behoben sind, so können alle Änderungen mit dem CVS-Kommando commit ins Repository übernommen werden. Für jede geänderte Datei wird ein Editor aufgerufen (Environment Variable *$EDITOR* bzw. *$CVSEDITOR*) damit man Kommentare zu den Änderungen hinzufügen kann:

```
<igor@puffy:/home/igor/new_proj:24>$ cvs commit
cvs commit: Examining .
...
Log message unchanged or not specified
 --> $CVSEDITOR
...
Checking in README;
/home/igor/cvs/new_proj/README,v <-- README
new revision: 1.2; previous revision: 1.1
done
```

10

- (**ACHTUNG:** Wenn nach einem *update* Konflikte nich aufgelöst wurden, so werden die Dateien mit Konfliktmarkierungen (*siehe Dealing with Conflicts*) ins Repository übertragen)

CVS Add

- Wenn man eine neue Datei im Arbeitsverzeichnis angelegt hat muss man dies mit dem CVS-Kommando add dem Repository hinzufügen bzw. mitteilen:

```
<igor@puffy:/home/igor/new_proj:34>$ touch BSD; echo "FREE" > BSD;
<igor@puffy:/home/igor/new_proj:36>$ cvs add BSD
cvs add: scheduling file 'BSD' for addition
cvs add: use 'cvs commit' to add this file permanently
<igor@puffy:/home/igor/new_proj:37>$ cvs commit
cvs commit: Examining .
cvs commit: Examining include
cvs commit: Examining lib
cvs commit: Examining src
 --> $CVSEDITOR
RCS file: /home/igor/cvs/new_proj/BSD,v
done
Checking in BSD;
/home/igor/cvs/new_proj/BSD,v <-- BSD
initial revision: 1.1
done
```


- Mit einem abschließenden commit erfolgt die endgültige Übernahme.

CVS Remove

- Um eine Datei aus dem Repository zu löschen muss diese erst im Arbeitsverzeichnis gelöscht werden um sie dann mit dem CVS-Kommando remove tatsächlich aus dem Repository löschen zu können:

```
<igor@puffy:/home/igor/new˙proj:48>$ rm BSD
<igor@puffy:/home/igor/new˙proj:49>$ cvs remove BSD
cvs remove: scheduling ‘BSD’ for removal
cvs remove: use ’cvs commit’ to remove this file permanently
<igor@puffy:/home/igor/new_proj:50>$ cvs commit -m "Removed BSD."
cvs commit: Examining .
cvs commit: Examining include
cvs commit: Examining lib
cvs commit: Examining src
Removing BSD;
/home/igor/cvs/new_proj/BSD,v <-- BSD
new revision: delete; previous revision: 1.1
done
```


- Mit einem abschließenden commit erfolgt die endgültige Übernahme.
- (**ACHTUNG:** Beim Löschen von ganzen Verzeichnissen müssen erst deren Dateien aus dem Repository löschen (ggf. rekursiv))

CVS Diff

- Um sich vor einem commit nochmal die Änderungen anzeigen zu lassen, welche man in der Arbeitskopie des Repository gemacht hat, benutzt man das CVS-Kommando diff, welches auf dem klassischen UNIX Kommando diff(1) aufbaut und die gleichen Optionen hat:

```
<igor@puffy:/home/igor/new˙proj:68>$ cvs diff -u
cvs diff: Diffing .
Index: README
===================================================================
RCS file: /home/igor/cvs/new˙proj/README,v
retrieving revision 1.2
diff -u -r1.2 README
--- README 3 Jun 2004 08:51:47 -0000 1.2
+++ README 3 Jun 2004 10:20:53 -0000
@@ -1 +1,4 @@
 THIS IS A CHANGE
 +
 +This is a change I made in
 +the working directory.
 cvs diff: Diffing include
 cvs diff: Diffing lib
 cvs diff: Diffing src
```

Dealing with Conflicts

- Beim update Kommando können nun aber auch **Konflikte** auftreten wenn jemand z.B. auf der selben Zeile wie man selbst etwas geändert hat:

```
<igor@puffy:/home/igor/new˙proj:105>$ cvs update
cvs update: Updating .
RCS file: /home/igor/cvs/new˙proj/README,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into README
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in README
C README
```

10

- Der eben produziert Konflikt kann nicht automatisch von CVS aufgelöst werden und wird mit dem Indikator **C** versehen.

- Was passiert nun also alles wenn es zu einem Konflikt kommt und wie kann man diesen auflösen?
  1. Die unmodifizierte Datei, von der beide Versionen ausgegangen sind, wird als .#file.version im jeweiligen Arbeitsverzeichnis abgelegt.
  2. Die Konflikte müssen manuell aufglöst werden bevor man sie mit den Befehlen update oder commit ins Repository spielen kann.
  3. Die eigentliche Datei enthält den kompletten Inhalt beider Versionen inklusive der Konflikte. Bereiche in welchen Konflikte aufgetreten sind schauen folgendermassen aus:

```
<<<<<<< README
THIS IS A CHANGE and it is COOL
=======
THIS IS A CHANGE and it is NOT COOL

This is a change I made in
the working directory.
>>>>>>> 1.3
```

Dates, Sticky Flags, Tags and other Gotchas

- **Dates:** Oft ist es wünschenswert einen bestimmten Zustand des Repositories zu einem gewissen Zeitpunkt bzw. Datum in der Vergangenheit wiederherzustellen bzw. auszuchecken:

  $ cvs -q update -D "2004-01-14 23:59:59" GMT

- **Sticky Flags:** Was passiert nun wenn wir z.B. einen Zustand des Repositories in der Vergangenheit wiederhergestellt haben und danach nochmal update ausführen? Nun, nichts wird passieren. Aber wir wissen dass es seit diesem Datum sicher viele Änderungen gegeben hat. Warum werden diese nicht wieder in die Arbeitskopie übernommen? Hier spielt das **Sticky Flag -D** die entscheidende Rolle da das update der Arbeitskopie auf eine version nach dem **-D "2004-01-14 23:59:59"** strengstens verboten wird.

- **Tags:** Bei **Tags** handelt es sich um symbolische, von Entwicklern vergebene Namen, die für einen bestimmten "Schnappschuß" der Revisionsnummern stehen. Wenn also das Repository **getagged** wird, werden die Revisionsnummern aller enthaltenen Dateien erfaßt und unter dem **Tag**namen abgelegt. Dies ist ganz praktisch weil man dann später auf eine bestimmte Programmversion einfach über den **Tag**name zugreifen kann:

  $ cvs -q tag OPENBSD_3_5 (ein Tag setzen)

  $ cvs checkout -rOPENBSD_3_5 src (retrieve by Tagname)

- **Keyword Substitution:**

  $Author: $ of last check-in
  $Date: $ of last check-in
  $Revision: $ revision (base) of this file
  $Name: $ (sticky) symbolic revision name, if any
  $Source: $ RCS file with path
  $Id: $ some of the above
  $Header: $ as Id, with full path to RCSfile
  $Log: $ auto-append special tag (generate ci-history within file)
  Beispiel: $Id: index.html,v 1.53 2004/06/02 14:12:37 igor Exp $

- **CVS und Binary Files:** CVS ist im Umgang mit Textdateien ziemlich **smart** und konvertiert z.B. Zeilenumbrüche wenn von einem Windows Rechner auf ein UNIX Archiv zugegriffen wird *(UNIX-LF, Windows-CR LF)* automatisch. Weiters werden wie schon erwähnt spezielle *RCS* Keywords wie z.B. $Revision:$ in Textfiles erkannt und dann dementsprechend richtig ersetzt. Dieses Verhalten welches bei Textdateien sehr nützlich ist, kann sich beim Verwalten von Binärdateien wie z.B. **JPEG**s sehr unangenehm auswirken. Man stelle sich mal vor *CVS* würde blind alle *RCS* Zeichenketten die es zufällig in einem **JPEG** findet ersetzten.

- CVS Parameter -**kb** für Binary Files:

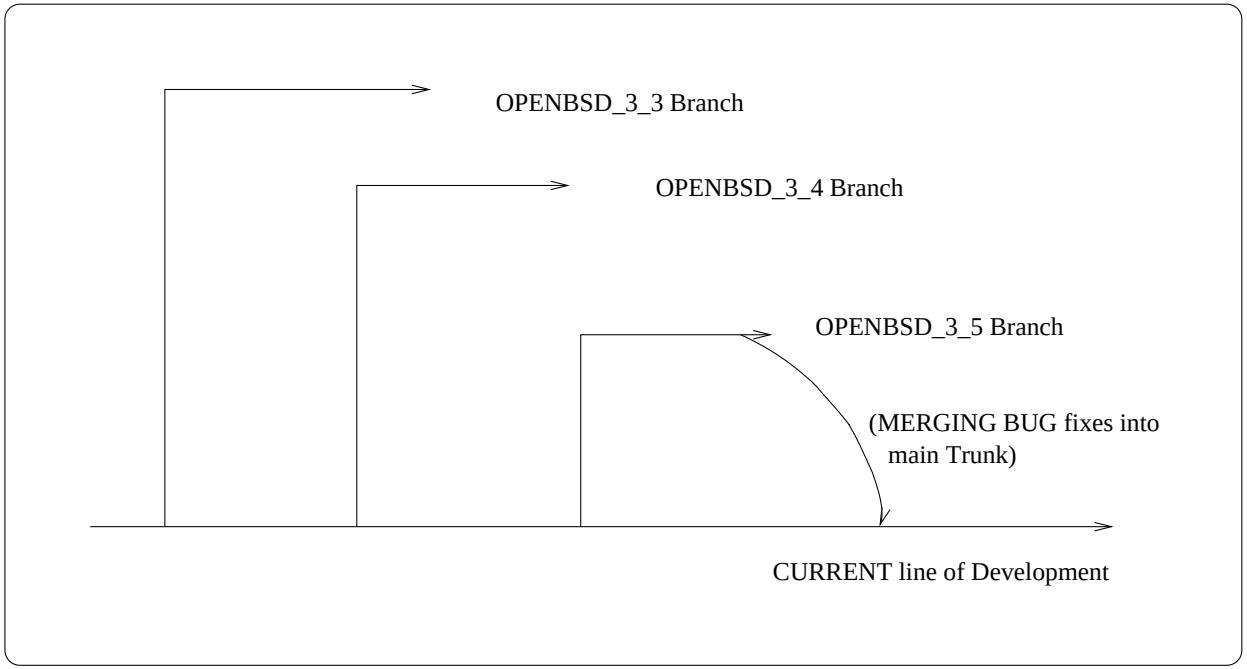
```
<igor@puffy:/home/igor/../new˙proj:13>$ cvs add -kb fabalabs1.png
<igor@puffy:/home/igor/../new˙proj:14>$ cvs ci -m "Binary File" fabalabs1.png
```

- CVS Parameter -**ko** für das Ausschalten der Schlüsselwortersetzung:

```
<igor@puffy:/home/igor/../new_proj:27>$ cvs add -ko main.c
<igor@puffy:/home/igor/../new_proj:28>$ cvs ci -m "No keyword expansion" main.c
```

More advanced CVS usage

- **(1)Branches:** Bis jetzt haben wir bei *CVS* nur die Möglichkeit betrachtet die Vergangenheit eines Projekts anhand von Tags zu untersuchen. *CVS* bietet aber auch die Möglichkeit in die "Vergangenheit" zu gehen und diese zu verändern. Um dies zu ermöglichen kann man ein Projekt in eigenständige, parallele **Branches** aufteilen. Änderungen auf einem **Branch** beeinflussen die anderen Branches nicht.



- **(2)Branches:** Bei den meisten Projekten wird es so gehandhabt, dass sobald die *-current* Entwicklungslinie ausgeliefert werden kann, ein *release*-**Branch** dessen Wurzel in der *-current* Entwicklunslinie liegt, angelegt wird. Neue Entwicklungen und Anforderungen werden weiter in der *-current* Entwicklungslinie gemacht, während Bugs welche z.B. bei einem Kunden auftreten im *release*-**Branch** gefixt werden und die Fixes wiederum dem Kunden zur Verfügung gestellt werden.
- **Merging:** Wie schon erwähnt beeinflusst eine Änderungen in einem bestimmten **Branch** keine anderen Branches. Aber oft will bzw. muss man Bugfixes in anderen *release*-**Branches** und auch in der *-current* Entwicklungslinie einfügen. Das **mergen** des Bugfixes in die *-current* Entwicklungslinie kann man mit dem einspielen eines *Patches*, welcher gegen die Version der Wurzel des Branches erstellt wurde, vergleichen.

Creating Branches

- Mit dem CVS-Parameter **-b** erzeugt man einen **Branch**:

```
<igor@puffy:/home/igor/new_proj:4>$ cvs -q tag -b RELEASE_BRANCH_0_1
T README
T main.c
```

- Warum benutzt man aber den CVS-Parameter **tag** um Branches zu erzeugen? Die Erklärung hierfür ist ganz einfach, denn der **tag** Name wird als *Label* dienen damit man den **Branch** später auschecken kann. D.h. aber wiederum das Branch **tags** genauso aussehen wie NON-branch **tags** und somit für beide Varianten die selben Namens Restriktionen gelten.
- Tipp: Es ist hilfreich das Wort *branch* im **tag name** selbst zu benutzen wenn man einen **Branch** anlegt, um Branch Tags leichter von anderen Tags zu unterscheiden.

- **Branch** auschecken:

```
<igor@puffy:/home/igor/new_branch:20>$ cvs co -r RELEASE_BRANCH_0_1 new_proj
cvs checkout: Updating new_proj
U new_proj/README
U new_proj/main.c
cvs checkout: Updating new_proj/include
cvs checkout: Updating new_proj/lib
cvs checkout: Updating new_proj/src
```

- Was hat sich nun geändert?

```
<igor@puffy:/home/igor/new_branch/new_proj:32>$ cvs -q status README
===================================================================
File: README    Status: Up-to-date

   Working revision: 1.3 Thu Jun 3 10:59:59 2004
   Repository revision: 1.3 /home/igor/cvs/new_proj/README,v
   Sticky Tag:     RELEASE_BRANCH_0_1 (branch: 1.3.2)
   Sticky Date:    (none)
   Sticky Options: (none)
```

10

- Machen wir nun eine kleine Änderung im **Branch** und schauen was passiert:

```
<igor@puffy:/home/igor/new_branch/new_proj:35>$ vim README
 --> EDIT FILE WITH EDITOR
<igor@puffy:/home/igor/new_branch/new_proj:36>$ cvs ci -m "CHANGED BRANCH"
...
<igor@puffy:/home/igor/new_branch/new_proj:37>$ cvs -q status README
===================================================================
File: README    Status: Up-to-date

   Working revision: 1.3.2.1 Wed Jun 9 00:11:45 2004
   Repository revision: 1.3.2.1 /home/igor/cvs/new_proj/README,v
   Sticky Tag:     RELEASE_BRANCH_0_1 (branch: 1.3.2)
   Sticky Date:    (none)
   Sticky Options: (none)
```

10

- Verglichen mit dem Output auf der vorigen Folie fällt auf, dass sich die **Revision** Nummern vom File *README* ganz merkwürdig verändert hat. Die neue *Revision* Nummer resultiert nun aus der *Branch* Nummer plus einer extra Ziffer. Bei diesen *Revision* Nummern handelt es sich aber eher um CVS-Internals. Man sollte einen Branch immer über den **Tag** Namen ansprechen und nicht über die **Revision** Nummer.

Merging Branches

- Nun wollen wir die Veränderungen in unserem **Branch** auch in die *-current* Entwicklungslinie einfliessen lassen. Dazu prüfen wir zuerst mit **diff** ob das überhaupt nötig ist:

```
<igor@puffy:/home/igor/new_proj:45>$ cvs diff -u -r RELEASE_BRANCH_0_1
cvs diff: Diffing .
Index: README
===================================================================
RCS file: /home/igor/cvs/new_proj/README,v
retrieving revision 1.3.2.1
retrieving revision 1.3
diff -u -r1.3.2.1 -r1.3
--- README 9 Jun 2004 00:12:00 -0000 1.3.2.1
+++ README 3 Jun 2004 10:59:59 -0000 1.3
@@ -1,4 +1,4 @@
 THIS IS A CHANGE and it is NOT COOL
-BRANCH CHANGE
+
  This is a change I made in
   the working directory.
   cvs diff: tag RELEASE_BRANCH_0_1 is not in file fabalabs1.png
.....
```

10

- Um die Änderungen nun aus dem anderen **Branch** zu übernehmen, wird uns der CVS-Parameter **-j** wie *join* und das CVS-Kommando **update** dienen:

```
<igor@puffy:/home/igor/new_proj:54>$ cvs -q update -j RELEASE_BRANCH_0_1
RCS file: /home/igor/cvs/new_proj/README,v
retrieving revision 1.3
retrieving revision 1.3.2.1
Merging differences between 1.3 and 1.3.2.1 into README
<igor@puffy:/home/igor/new_proj:55>$ cat README
THIS IS A CHANGE and it is NOT COOL
BRANCH CHANGE
This is a change I made in
the working directory.
```

10

- **VORSICHT:** Die Änderungen aus dem anderen **Branch** wurden einstweilen nur in die Arbeitskopie übernommen. Um diese auch ins **Repository** zu übernehmen, muss man noch das CVS-Kommando commit benutzen.

Thx for your attention! Any more questions left?

List of Slides