Wprowadzenie do systemów kontroli wersji RCS i CVS

Grzegorz Jacek Nalepa

22.01.2001, Kraków, *Revision* : 1.5

Streszczenie

Artykuł wprowadza do problematyki kontroli wersji plików tekstowych. Przedstawione są podstawowe wymagania stawiane przed systemem kontroli wersji. Realizacja takiego systemu jest pokazana na przykładzie klasycznego już systemu RCS. Zaprezentowany jest sposób wykorzystania RCS do kontroli wersji kodu. Następnie omówiony jest system CVS będący rozszerzeniem RCS o mechanizmy pracy z drzewami plików, a także o mechanizmy pracy grupowej. Pokazane są również narzędzia graficzne wspomagające pracę z systemem CVS.

Spis treści

[1]Tekst nie ukazał się drukiem.

[2]Kontakt z autorem: mail:gjn@agh.edu.pl

[3]Tytuł angielski: *Introduction to version control with RCS and CVS*

[4]Tekst jest rozpowszechniany na zasadach licencji *GNU Free Documentation License*, której pełny tekst można znaleźć pod adresem: http://www.gnu.org/copyleft/fdl.html

1. Kontrola wersji

Jedną z cech współczesnych systemów informatycznych jest to, że ulegają one nieustannym zmianom. Te zmiany mogą być związane ze zmieniającymi się wymaganiami użytkowników, zmianami technologii na której opierają się systemy, lub z korygowaniem błędów odnajdywanych w oprogramowaniu, czy dokumentacji wchodzących w skład systemu.

Abstrahując od samego charakteru wspomnianych zmian, pojawia się istotny problem kontroli, rejestrowania i dokumentowania samych zmian. Dzięki tego typu kontroli odbiorca, czy użytkownik systemu ma możliwość zorientowania się w zmianach zachodzących w funkcjonalności systemu, lub w usprawnieniach w jego działaniu. Natomiast twórcy systemu mają lepsze możliwości analizy własnej pracy.

W przypadku współczesnego oprogramowania, najistotniejsze wydają się zmiany w kodzie źródłowym, dokumentacji czy plikach konfiguracyjnych. Dlatego też system kontroli zmian zachodzących w oprogramowaniu jest bezcennym narzędziem dla szeroko rozumianych developerów.

Kod źródłowy programów oraz ich dokumentacja składa się z wielu różnego rodzaju plików. W zdecydowanej większości przypadków są to pliki tekstowe. Tak więc problem kontroli zmian zachodzących w oprogramowaniu można sprowadzić do problemu kontroli zmian, zachodzących w kolejnych wersjach plików tekstowych.

1.1. Funkcje systemu kontroli wersji

Najważniejsze funkcje tak rozumianego systemu kontroli wersji oprogramowania można przedstawić następująco:

- przechowywanie i kontrola dostępu do plików tekstowych wchodzących w skład oprogramowania,
- śledzenie zmian zachodzących w poszczególnych plikach,
- przechowywanie całej historii zmian,
- udostępnianie każdej kolejnej wersji poszczególnych plików,
- możliwość tworzenia, śledzenia, różnych konfiguracji danego programowania,
- pełne dokumentowanie wprowadzanych zmian.

Dodatkowo tego typu system może mieć szereg funkcji związanych z organizacją pracy zespołu rozwijającego oprogramowanie, takich jak:

1. kontrola dostępu do plików dla różnych osób,
2. synchronizacja zmian wprowadzanych przez różnych autorów,
3. rozwiązywanie konfliktów pomiędzy kolidującymi ze sobą zmianami,
4. praca w środowisku rozproszonym w sieci komputerowej,
5. kontrola faz procesu rozwijania oprogramowania, na przykład wymuszanie faz testowania czy dokumentowania zmian.

1.2. Dostępne systemy kontroli wersji

Technologia *opensource* na której oparte są systemy GNU/Linux czy BSD udostępnia szereg zaawansowanych systemów kontroli wersji, które realizują wymienione powyżej funkcje. Najbardziej znane z nich to systemy *RCS*, *CVS*. Nieco mniej rozpowszechnione, choć równie interesujące są *Aegis* i *PRCS*.

Wszystkie z tych systemów realizują najważniejsze funkcje podane w 1. części punktu 1.1. Dodatkowo systemy CVS i Aegis realizują funkcje 1 – 4, a w systemie Aegis położono nacisk na funkcję 5. Autorzy PRCS stworzyli rozbudowane mechanizmy synchronizacji kilku różnych wersji plików.

Artykuł zawiera obszerne wprowadzenie do systemu RCS i opartego na nim CVS, które stały się *de facto* standardem dla systemów kontroli wersji w środowisku *opensource*.

2. System RCS

System RCS [1] – *Revision Control System* – jest pakietem poleceń pracujących w środowisku Unix/GNU/Linux, realizującym kontrolę wersji w plikach tekstowych. Autorem RCS jest Walter F. Tichy. System powstał na znanym wydziale informatyki Uniwersytetu Purdue w Indianie.

RCS zarządza *grupami wydań* (edycji) (ang. *revision*) plików. Przez grupę wydań rozumie się pewien zbiór dokumentów tekstowych powiązanych ze sobą w ten sposób, że kolejne pliki powstawały przez modyfikacje poprzednich. System RCS organizuje tak rozumiane wydania w drzewo, przedstawiające historię zmian wydań. W praktyce, wydanie można nazywać *wersją* w szerszym rozumieniu tego słowa.

System automatyzuje proces tworzenia i przechowywania kolejnych wersji projektu. Zapewnia również mechanizmy pozwalające na dokładne określanie różnić pomiędzy wersjami oraz scalanie różnych wersji. Wszystkie zmiany w kolejnych wersjach są rejestrowane i dokumentowane, co umożliwia w każdej chwili na uzyskanie opisu historii zmian.

Wprawdzie RCS był tworzony z myślą głównie o zarządzaniu wersjami kodu źródłowego programów, lecz w praktyce pracuje z dowolnymi plikami tekstowymi. Oznacza to, że może być stosowany do dowolnych zmieniających się plików tekstowych, takich jak kody źródłowe w językach programowania, czy skryptowych, dokumenty tekstowe, dokumenty w językach opisu tekstu takich jak LaTeX, a także HTML czy XML. Oznacza to, iż RCS jest w stanie zarządzać nie tylko wersjami kodu programów, lecz również dokumentacją w różnych formatach.

2.1. Funkcje RCS

Główne funkcje systemu RCS [2] są następujące:

- Przechowywanie i udostępnianie kolejnych wersji plików. Kolejne wersje są przechowywane w *repozytorium RCS* (ang. *repository*) w taki sposób, aby zajmowały możliwie mało miejsca. Repozytorium ma postać specjalnego pliku RCS, mającego rozszerzenie *,v*. Modyfikacje pliku nie niszczą jego poprzedniej zwartości, gdyż w każdej chwili dostępne są jego poprzednie wersje. Wersje mogą być udostępniane w zależności od ich numerów, dat tworzenia, autorów, lub nazw symbolicznych.

- Rejestrowanie zmian i przechowywanie historii zmian plików. System nie tylko rejestruje same zmiany pliku, lecz również umożliwia ich opisywanie. Rejestrowane są również informacje o tym kto i kiedy modyfikował plik. Tworzona jest w ten sposób dokumentacja opisująca ewolucję pliku.

- Ewolucja kolejnych wersji pliku jest opisywana przy pomocy struktury drzewiastej. Umożliwia to prześledzenie zmian, jak również *rozgałęzianie* (ang. *branching*) procesu rozwoju projektu, przykładowo na wersję stabilną i niestabilną, czy podstawową i rozszerzoną.
- Scalanie (ang. *merging*) kolejnych wersji i rozgałęzień wersji pliku. Podczas scalania system porównuje wersje i pozwala na rozwiązywanie konfliktów zachodzących w przypadku występowania kolidujących ze sobą zmian w pliku.
- Kontrola dostępu do repozytorium. RCS pozwala na *blokowanie dostępu* (ang. *locking*) do plików znajdujących się w repozytorium tak, aby tylko jedna osoba mogła je modyfikować. Może to zapobiegać powstawaniu konfliktów.
- Kontrolowanie *konfiguracji* i *edycji* plików. Poprzez nadawanie nazw symbolicznych i rozgałęzianie wersji plików można kontrolować różne edycje plików wchodzących w skład projektu. Możliwe jest stworzenie *edycji* projektu oznaczonej przez nazwę symboliczną, w skład której wchodzą pliki o różnych wersjach numerycznych RCS.

Szczegółowy opis funkcji RCS można znaleźć w dokumencie *Functions of RCS*, towarzyszącym artykułowi [1].

2.2. Architektura RCS

System RCS składa się z następujących programów:

ci – *check in revision*
Polecenie *ci* umieszcza aktualnie edytowaną wersję pliku w repozytorium. Jeżeli repozytorium nie istnieje, to zostanie utworzone. W domyślnym trybie pracy *ci* przenosi aktualną roboczą wersję pliku, to znaczy usuwa go z systemu plików. Autor zmian jest również proszony o ich opisanie. Wraz z samymi zmianami, w repozytorium jest rejestrowany opis, data oraz informacje o autorze zmian.

Ci automatycznie przydziela nowy numer wersji oraz porównuje go z już występującymi w repozytorium.

W trakcie procesu umieszczania pliku w repozytorium uaktualnione są znajdujące się w pliku znaczniki RCS związane z z wersją, danymi autora i historią zmian.

co – *check out revision*
Polecenie *co* udostępnia z repozytorium określoną wersję pliku w zależności od podanego numeru wersji, daty modyfikacji, lub danych autora.

W trakcie odtwarzania pliku z repozytorium są w nim umieszczane odpowiednie wartości znaczników RCS.

rcs – *change RCS file attributes*
Jest to polecenie administracyjne pozwalające na operowanie na repozytorium. Umożliwia między innymi blokowanie dostępu, zmianę atrybutów, nadawanie nazw symbolicznych wersjom, a także usuwanie określonych wersji z repozytorium.

rcsdiff – *compare revisions*
Pozwala na porównanie zmian pomiędzy wersjami i podanie ich w formacie programu *diff*.

rcsmerge – *merge revisions*
Umożliwia scalanie dwóch wersji, które powstały w wyniku rozgałęzienia zmian w pliku. Wersje muszą mieć tego samego przodka w drzewie opisującym historię zmian.

rlog – *read log*
Wyświetla zarejestrowane informacje na temat zmian wprowadzonych w pliku.

ident – *extract identification*
Odczytuje wartości znaczników RCS z edytowanego pliku.

rcslean – *clean working directory*
Usuwa z bieżącego katalogu niezmodyfikowane pliki, porównując je z repozytorium.

rcsfreeze – *freeze configuration*
Pozwala na przypisywanie tej samej nazwy symbolicznej różnym wersjom plików. Tworzy tym samym zbiór plików, nazywany przez RCS konfiguracją lub edycją.

Najistotniejsze do pracy z RCS są polecenia *ci*, *co* oraz *rcs*. W przypadku prostych projektów, składających się tylko z kilku plików mogą wystarczyć do obsługi systemu.

2.3. Instalacja i konfiguracja

Instalacja pakietu jest niezwykle prosta. Sprowadza się do zainstalowania pakietu RCS z pliku deb, rpm lub binarnego tar.gz. Ewentualnie można skompilować pakiet z kodu źródłowego. System nie wymaga do pracy żadnych dodatkowych pakietów czy bibliotek.

W skład pakietu oprócz podanych powyżej poleceń wchodzą dotyczące ich strony man, a także dodatkowa dokumentacja, w tym artykuł [1] W. F. Tichy'ego.

Po zainstalowaniu pakiet nie wymaga praktycznie żadnej konfiguracji. Użytkownik może od razu przejść do pracy z systemem przez założenie nowego repozytorium dla jakiegoś pliku.

W przypadku systemu Debian/GNU instalacja i konfiguracja pakietu sprowadza się do wydania polecenia:

```
# apt-get install rcs
```

2.4. Jak używać RCS

Wykorzystanie systemu RCS warto prześledzić na przykładzie bardzo prostego projektu składającego się na początku z jednego pliku. Projekt polega na zaimplementowaniu klasycznej aplikacji *Helloworld*, wspomnianej między innymi przez B. W. Kernighana i D. M. Ritchiego w ich znanej książce ([4]).

2.4.1. Tworzenie i edycja wersji

Pracą z systemem kontroli wersji należy rozpocząć od założenia repozytorium, czyli pliku o rozszerzeniu `,v`. Domyślnie, taki plik powinien się znaleźć w podkatalogu `RCS` w katalogu bieżącym w którym znajduje się edytowany plik. Jednak używanie katalogu `RCS` nie jest konieczne.

Istnieją dwie możliwości tworzenia repozytorium. Można wykorzystać się do tego program *rcs* z opcją `-i`, która oznacza utworzenie i inicjalizację nowego repozytorium. System RCS pyta o jego opis. Można również założyć ręcznie katalog `RCS`, a plik repozytorium zostanie automatycznie założony przy pierwszym uruchomieniu *ci*.

```
$ cd helloworld
$ rcs -i helloworld
RCS file: helloworld,v
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
```

```
>> Projekt helloworld prezentuje możliwości RCS
>> .
done
$ ls
helloworld,v
```

Projekt *Helloworld* znajduje się w katalogu `helloworld` i składa się na początku z jednego pliku, `helloworld.c`, zawierającego kod źródłowy. Pierwsza wersja pliku wygląda następująco:

```
$ cat helloworld.c
#include <stdio.h>
main () {
    printf ("Hello world!\n");
}
```

Aby włączyć (ang. *checkin*) plik do repozytorium należy wydać polecenie:

```
$ ci helloworld.c
helloworld.c,v  <--  helloworld.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Plik zawiera kod źródłowy programu helloworld
>> .
initial revision: 1.1
done
$ ls
helloworld.c,v
```

Jak widać, system RCS zapytał o opis pliku i przydzielił mu pierwszy, domyślny numer wersji 1.1. Co istotne, RCS przeniósł plik `helloworld.c` do repozytorium i usunął go z systemu plików.

Aby zmodyfikować plik z kodem źródłowym należy go uzyskać z repozytorium:

```
$ co helloworld.c
helloworld.c,v  -->  helloworld.c
revision 1.1
done
```

W katalogu bieżącym został stworzony egzemplarz roboczy pliku `helloworld.c` w wersji aktualnej, w tym przypadku 1.1. Zawartość pliku nie uległa zmianie, bo jak dotychczas nie był poddany kolejnej edycji.

Jednak tak uzyskanego pliku nie można edytować, brak jest praw do zapisu. Dzieje się tak dlatego, aby nie doszło do sytuacji w której kilka osób edytuje równocześnie ten sam plik. Aby była możliwa edycja pliku, należy podczas uzyskiwania go z repozytorium, zablokować (ang. *lock*) dostęp do niego innym użytkownikom. Służy do tego opcja `-l` (ang. *lock*) polecenia *co*.

```
$ co -l helloworld.c
helloworld.c,v  -->  helloworld.c
revision 1.1 (locked)
done
```

Podczas gdy jeden użytkownik zablokował dostęp do pliku, inni nie mogą wprowadzać do niego zmian. Próba zablokowania pliku przez innego użytkownika spowodowałaby pojawienie się komunikatu:

```
$ co -l helloworld.c
helloworld.c,v  -->  helloworld.c
co: helloworld.c,v: Revision 1.1 is already locked by gjn.
```

W ten sposób system RCS wymusza synchronizację pracy programistów, jeden plik może być w danej chwili edytowany przez tylko jedną osobę.

Po wprowadzeniu zmian do pliku wygląda on następująco:

```
$ cat helloworld.c
/* Kod programu Helloworld */
#include <stdio.h>

void main (int argc, char *argv[]) {
    printf ("Hello world!\n");
}
```

Można go ponownie umieścić w repozytorium:

```
$ ci helloworld.c
helloworld.c,v  <--  helloworld.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> dodane param. linii poleceń i opis
>> .
done
```

Wraz z plikiem został umieszczony krótki opis wprowadzonych zmian.

Jeżeli chce się zmienić numer wersji na inny niż kolejny, proponowany przez RCS, to można to zrobić podając go przy poleceniu *ci*. Na przykład:

```
$ ci -l -r2.0 helloworld.c
```

Opcja `-l` polecenia *ci* powoduje, że plik nie tylko zostanie umieszczony w repozytorium, ale następnie od razu zostanie z niego udostępniony. Innymi słowy wywołanie `ci -l` jest równoważne sekwencji `ci ; co -l`.

Podając poleceniu *co* konkretny numer wersji pliku, można uzyskać z repozytorium dowolną z wcześniejszych wersji pliku. Na przykład polecenie:

```
$ co -r1.1 helloworld.c
```

uzyska z repozytorium wersję 1.1 pliku, bez względu na to, jaka była ostatnia edytowana wersja.

2.4.2. Porównywanie wersji

Mając w repozytorium kilka wersji można porównać zmiany jakie zostały w nich wprowadzone. Służy do tego polecenie *rcsdiff* Aby porównać zmiany pomiędzy dwoma podanymi wersjami, w tym przypadku 1.1 i 1.2 należy wykonać:

```
$ rcsdiff -r1.1 -r1.2 helloworld.c
+=================================
RCS file: helloworld.c,v
retrieving revision 1.1
retrieving revision 1.2
diff -r1.1 -r1.2
```

```
0a1
> /* Kod programu Helloworld */
2c3,4
< main () {
---
>
> void main (int argc, char *argv[]) {
```

Zostały wyświetlone zmiany w formacie programu *diff (1)*. Użycie *rcsdiff* bez opcji powoduje porównanie aktualnej wersji roboczej pliku z wersją ostatnio umieszczoną w repozytorium.

Próba kompilacji aktualnej wersji programu wykaże, że jest w nim drobna usterka, którą warto poprawić:

```
$ co -l helloworld.c
$ emacs helloworld.c
$ rcsdiff helloworld.c
===================================
RCS file: helloworld.c,v
retrieving revision 1.2
diff -r1.2
4c4
< int main (int argc, char *argv[]) {
---
> main (int argc, char *argv[]) {
6d5
<     return 0;
$
```

Opcja `-l` polecenia *ci* powoduje, że plik nie tylko zostanie umieszczony w repozytorium, ale następnie od razu zostanie z niego udostępniony. Innymi słowy wywołanie `ci -l` jest równoważne sekwencji `ci ; co -l`.

```
$ ci -l helloworld.c
helloworld.c,v  <--  helloworld.c
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single '.' or end of file:
>> poprawiony typ zwracany w main()
>> .
done
$ ls -l helloworld.c
-rw-r--r--    1 gjn   gjn        133 Jan 21 15:16 helloworld.c
```

W tej chwili są już trzy kolejne wersje pliku. Można w związku z tym przeprowadzić porównanie różnic pomiędzy każdą z nich.

```
$ rcsdiff -r1.1 -r1.3 helloworld.c
===================================
RCS file: helloworld.c,v
retrieving revision 1.1
retrieving revision 1.3
diff -r1.1 -r1.3
0a1
```

```
> /* Kod programu Helloworld */
2c3,4
< main () {
---
>
> int main (int argc, char *argv[]) {
3a6
>     return 0;
```

2.4.3. Rejestrowanie opisów zmian

Opisy zmian, zachodzących w kolejnych wersjach, które umieszcza się w repozytorium, mogą być przeglądane dzięki poleceniu *rlog (1)*. Możliwe jest przeglądanie opisów w zależności od wersji, daty, czy autora zmiany. Poniżej jest pokazany przykład wyświetlenia opisów zmian dla podanych wersji pliku:

```
$ rlog -r1.2 helloworld.c
RCS file: helloworld.c,v
Working file: helloworld.c
head: 1.3      branch:
locks: strict gjn: 1.3
access list:     symbolic names:
keyword substitution: kv
total revisions: 3;  selected revisions: 1
description:
Plik zawiera kod źródłowy programu helloworld
----------------------------
revision 1.2
date: 2001/01/21 12:14:56;  author: gjn;
   state: Exp;  lines: +3 -1
dodane param. linii poleceń i opis
============================================
```

2.4.4. Znaczniki RCS

Często przydatne jest zapisywanie informacji o zachodzących zmianach bezpośrednio w plikach kontrolowanych przez RCS. Jest to możliwe dzięki umieszczaniu w plikach specjalnych *znaczników RCS*. System definiuje ponad 10 różnych znaczników. Wszystkie znaczniki są opisane w podręczniku polecenia *ident (1)* i mają postać: `$znacznik:  napis$` .

Znaczniki umieszcza się bezpośrednio w pliku, tak jak w poniższym przykładzie:

```
/* Program: $ RCSfile:$
 * Autor:   $ Author:$
 * Wersja:  $ Revision:$
 */
```

W czasie udostępniania wersji roboczej pliku przez polecenie *co*, w miejsce znaczników są wstawiane odpowiednie informacje.

```
$ ci -l helloworld.c
helloworld.c,v  <--  helloworld.c
new revision: 1.4; previous revision: 1.3
```

```
done
$ cat helloworld.c
/* Program: $ RCSfile: helloworld.c,v $
 * Autor:   $ Author: gjn $
 * Wersja:  $ Revision: 1.4 $
 */
```

2.5. RCS i GNU Emacs

Komfort pracy z RCS można znacznie zwiększyć, jeżeli używa się edytora GNU Emacs. Jak powszechnie wiadomo, Emacs ma wiele zaawansowanych modułów integrujących go z szeregiem zewnętrznych narzędzi. Wśród nich jest również zintegrowany moduł współpracy z systemem kontroli wersji RCS.

Moduł *Version Control* składa się z ponad 40 poleceń obsługujących wszystkie funkcje RCS i dodatkowe makro polecenia wykorzystujące RCS. Z punktu widzenia początkującego użytkownika najważniejsza jest funkcja *vc-next-action*, standardowo przypisane do polecenia Emacsa *C-x v v*. Odpowiada ona wykonaniu polecenia *ci -l* na aktualnym buforze Emacsa. W trakcie jej wykonywania Emacs otwiera nowe okno pozwalające na wprowadzenie opisu dokonanej zmiany. Wyjście z okna opisu jest możliwe przez polecenie *C-c C-c*. Następnie bufor jest automatycznie zaznaczany jako tylko do odczytu. Po wykonaniu polecenia *C-x v v* rejestrowana jest nowa wersja pliku w systemie RCS.

Kontynuowanie edycji jest możliwe po wykonaniu kolejnego *C-x v v*. Bufor Emacsa przechodzi wtedy w tryb zapisu-odczytu. Aktualna wersja pliku jest wyświetlana w linii trybu edytora Emacs, przy dolnej krawędzi aktualnego okna. Inne polecenia trybu *VC* są dostępne z menu Emacsa. Wszystkie jego możliwości są szczegółowo opisane w dokumentacji info edytora. Praca z trybem *VC* jest pokazana na Rysunku 1.

Tryb *Version Control* znakomicie upraszcza korzystanie z systemu RCS z poziomu Emacsa, pozwalając na kontrolowanie wersji plików bez opuszczania edytora.

2.6. Zastosowania i ograniczenia systemu

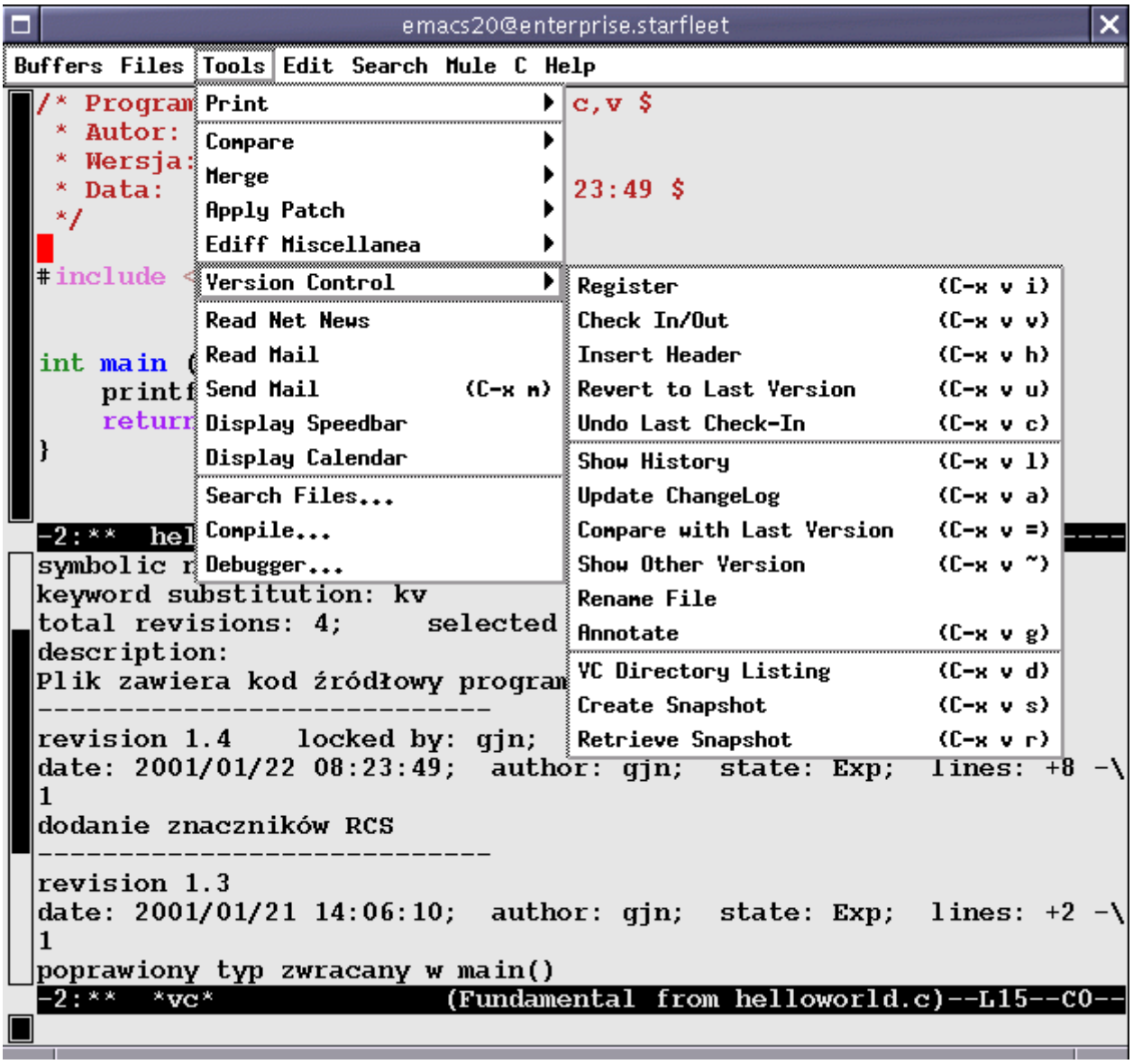
Jak można zauważyć, korzystanie z RCS jest niezwykle proste. System jest również bardzo uniwersalny, można go stosować do dowolnych projektów wykorzystujących pliki tekstowe.

RCS jest dobrze udokumentowany. Do standardowej dystrybucji zawierającej podręczniki man do każdego z poleceń, dołączony jest dokument [2], a także artykuł autora systemu [1]. Poza tym, początkujący użytkownicy mogą skorzystać z dokumentu RCS-mini-HOWTO ([3]).

Do licznych zalet RCS należy również jego prostota i stabilność. Używany od wielu lat system, można w tej chwili uznać za praktycznie bezbłędny. Również fakt, że system nie wymaga skomplikowanej konfiguracji, korzystnie korzystnie wpływa na jego przydatność szczególnie dla początkujących użytkowników.

Pomimo wszystkich, wymienionych powyżej, zalet, można dostrzec kilka, niekiedy dość istotnych, ograniczeń RCS. Najważniejsze z nich to:

- RCS został stworzony z myślą o zarządzaniu projektami składającymi się ze stosunkowo niewielkiej liczby plików, jego zastosowanie do dużych zbiorów plików może być bardziej kłopotliwe.
- System zaprojektowano z myślą o projektach mieszczących się w jednym, lub kilku katalogach, a co za tym idzie nie jest w stanie zarządzać dużymi drzewami plików spotykanymi w wielu współczesnych projektach.



Rysunek 1: Praca z RCS w GNU Emacs

- Leżący u podstaw RCS mechanizm blokowania dostępu do plików, służący do synchronizacji pracy autorów, może w istotny sposób spowalniać i ograniczać pracę nad projektem, uniemożliwiając równoległą pracę wielu osób.
- System nie pozwala na rozproszenie pracy zespołu, gdyż nie oferuje sieciowych metod dostępu do repozytoriów, co w dobie globalnych sieci należy uznać za poważny brak.

Wszystkie te ograniczenia znosi, oparty na RCS, system CVS.

3. System CVS

Pierwsza wersja CVS, autorstwa Dicka Grune'a, miała postać skryptu shella i pojawiła się w grudniu 1986 na liście `comp.sources.unix`. Choć w dzisiejszym CVS, *Concurrent Versions System*, nie ma już wiele z tamtego kodu, to pozostały algorytmy rozwiązywania konfliktów opracowane przez Grune'a. Głównymi autorami aktualnej wersji CVS są Brian Berliner i Jeff Polk. Poza nimi system udoskonalało wielu innych programistów i należy go dziś uznać za bardzo dojrzały i stabilny.

Po zaprezentowanym w pierwszej części artykułu omówieniu zadań i funkcji systemów kontroli wersji, a także po zaprezentowaniu RCS, warto skupić się nad tym, co nowego wnosi CVS.

3.1. Funkcje CVS

System CVS ma wszystkie funkcje systemu kontroli wersji (opisane w punkcie 1.1) a dodatkowo umożliwia:

- zarządzanie dużymi projektami liczącymi setki i tysiące plików,
- zarządzanie projektami składające się ze złożonych drzew katalogów,
- równoległą pracę wielu autorów nad tymi samymi plikami,
- synchronizację zmian wprowadzanych równocześnie przez wielu autorów,
- decentralizację i rozproszenie pracy zespołu autorów,
- synchronizację repozytoriów rozproszonych w sieci komputerowej,
- złożoną konfigurację i zarządzanie repozytorium.

3.2. Architektura CVS

System CVS jest oparty na RCS. Realizuje wszystkie funkcje RCS, oraz wykorzystuje go do rejestrowania modyfikacji w poszczególnych plikach którymi zarządza. Repozytorium CVS przechowuje informacje na temat wielu drzew katalogów zawierających pliki objęte kontrolą wersji. Drzewa mogą być organizowane w osobne moduły czy projekty.

W przeciwieństwie do RCS, który składa się z szeregu krótkich programów realizujących poszczególne funkcje, CVS składa się z jednego programu o nazwie *cvs* udostępniającego polecenia uruchamiające wszystkie funkcje systemu. Najważniejsze z nich to:

init dokonuje inicjalizacji nowego repozytorium CVS,

checkout udostępnia wskazany moduł projektu (drzewo plików) z repozytorium w postaci kopii roboczej, (odpowiednik *co* w RCS),

import pozwala na dodanie do kopii roboczej projektu całego drzewa plików,

add pozwala na dodanie do kopii roboczej projektu pliku lub katalogu,

remove umożliwia usunięcie z kopii roboczej projektu pliku lub katalogu,

update dokonuje uaktualnienia kopii roboczej z repozytorium projektu,

commit przeprowadza synchronizację kopii roboczej z repozytorium, umieszczając w nim zmiany wprowadzone w kopii roboczej, (odpowiednik *ci* w RCS),

diff pokazuje zmiany pomiędzy różnymi wersjami plików lub całych drzew projektu, (odpowiednik *rcsdiff* w RCS),

log wyświetla historię i opisy zmian plików lub całych drzew projektu, (odpowiednik *rlog* w RCS),

Listę wszystkich poleceń można zobaczyć pisząc:

```
cvs --help-commands
```

3.3. Instalacja i konfiguracja

Pakiet CVS można skompilować z wersji źródłowej, lub zainstalować z pakietu binarnego. Podobnie jak RCS jest dostępny na dowolną platformę GNU/Linux, Unix. Przykładowo instalacja CVS w systemie Debian/GNU może być następująca:

```
$ apt-get install cvs cvs-doc cvs-book
```

Oprócz samego pakietu CVS zostanie zainstalowany pakiet zawierający dokumentację, a także książka [10].

Do poprawnej pracy, CVS wymaga ustalenia głównego katalogu, w którym znajdzie się repozytorium CVS Konfiguracja polega na ustawieniu zmiennej środowiskowej `CVSROOT`, na przykład tak:

```
$ export CVSROOT=/usr/src/CVS
```

Tego typu ustawienie można zapisać w plikach startowych powłoki, przykładowo `/etc/profile`. Dodatkowo, każdy z użytkowników może skonfigurować pakiet poprzez własny plik inicjalizacyjny `~/.cvsrc`. W pliki można między innymi podać opcje, dla różnych poleceń *cvs*.

3.4. Praca z CVS

Pracę z CVS można zaprezentować na nieco rozszerzonej wersji HelloWorld, składającej się z następujących plików:

```
helloworld
helloworld/Makefile
helloworld/README
helloworld/src/helloworld.c
helloworld/doc/Users_Guide
```

Należy rozpocząć od konfiguracji i inicjalizacji repozytorium:

```
$ export CVSROOT=/home/gjn/CVS
$ cvs init
```

Następnie należy przejść do katalogu, który chcemy przenieść do repozytorium CVS i dokonać jego *importu*, podając nazwę projektu i dwa dodatkowe znaczniki dotyczące wersji projektu.

```
$ cd helloworld
$ cvs import helloworld V1_0 R1_0
N helloworld/Makefile
N helloworld/README
cvs import: Importing /home/gjn/CVS/helloworld/src
U helloworld/src/helloworld.c
N helloworld/src/helloworld.c,v
cvs import: Importing /home/gjn/CVS/helloworld/doc
U helloworld/doc/Users_Guide
N helloworld/doc/Users_Guide,v

No conflicts created by this import
```

W trakcie pracy z CVS program pozwala na wprowadzanie opisów modyfikacji projektu. Otwierany jest wtedy domyślny edytor, ustawiony w zmiennej `EDITOR`.

Aby rozpocząć pracę nad projektem znajdującym się w repozytorium, należy najpierw uzyskać (ang. *checkout*) kopię roboczą projektu z repozytorium:

```
$ cvs checkout helloworld
cvs checkout: Updating helloworld
U helloworld/Makefile
U helloworld/README
cvs checkout: Updating helloworld/doc
U helloworld/doc/Users_Guide
cvs checkout: Updating helloworld/src
U helloworld/src/helloworld.c
```

W katalogu bieżącym zostanie otworzona pełna kopia robocza drzewa katalogów projektu. Można teraz rozpocząć edycję plików.

Jeżeli chce się dodać albo usunąć jakieś pliki lub katalogi z kopii roboczej projektu, należy to zrobić przy pomocy poleceń *cvs add* i *cvs remove*.

```
$ cvs add INSTALL
cvs add: scheduling file ‘INSTALL’ for addition
cvs add: use ’cvs commit’ to add this file permanently
```

Plik, w tym przypadku `INSTALL`, zostanie dodany do repozytorium projektu w czasie umieszczania zmian z całej kopii roboczej.

System CVS nie blokuje w żaden sposób dostępu do plików, lecz zawsze po operacji *cvs checkout* tworzy nową kopię roboczą dla konkretnego użytkownika. Dlatego w tym samym czasie może pracować wielu autorów nad własnymi egzemplarzami kodu. W trakcie pracy każdy autor może porównać swoją kopię roboczą ze stanem repozytorium, aby sprawdzić, czy inne osoby nie wprowadziły jakichś zmian do projektu.

```
$ cvs update
cvs update: Updating .
A INSTALL
M README
cvs update: Updating doc
cvs update: Updating src
```

Jak widać został zmodyfikowany plik `README` i dodany plik `INSTALL`.

Po wprowadzeniu modyfikacji autor umieszcza swoją kopię roboczą w repozytorium. Jest to złożony proces, ponieważ CVS musi porównać tę kopię ze stanem repozytorium, które od czasu utworzenia kopii roboczej, mogło być wielokrotnie zmieniane przez innych autorów. W czasie tego porównywania, CVS rozwiązuje ewentualne konflikty w modyfikacjach. W razie potrzeby autor jest proszony o potwierdzenie odpowiednich zmian.

```
$ cvs commit
cvs commit: Examining .
cvs commit: Examining doc
cvs commit: Examining src
RCS file: /home/gjn/CVS/helloworld/INSTALL,v   done
Checking in INSTALL;
/home/gjn/CVS/helloworld/INSTALL,v  <--  INSTALL
initial revision: 1.1   done
Checking in README;
/home/gjn/CVS/helloworld/README,v  <--  README
new revision: 1.2; previous revision: 1.1   done
```

Poza przedstawionymi powyżej funkcjami CVS ma rzecz jasna różnorodne funkcje pozwalające uzyskiwać informacje na temat wprowadzonych zmian i ich historii. Są one dostępne przez polecenie *cvs log*, które działa analogicznie do *rlog*, z tym, że może podać informacje dotyczące wszystkich plików w projekcie.

Polecenie *cvs diff* działa analogicznie do *rcsdiff* i pozwala na porównywanie różnych wersji odpowiednich plików wchodzących w skład projektu.

Przy pomocy *cvs history* można oglądnąć historię cyklu pracy nad projektem. Udostępniane są również informacje na temat tego kto i kiedy wykonywał polecenia *checkout*, *checkin*, *rtag*, *update* i *release*.

3.5. Przenoszenie projektów z RCS

Projekty, które wcześniej były zarządzane przez system RCS można przenieść do CVS. Ponieważ CVS wykorzystuje RCS do rejestrowania zmian w plikach, do przeniesienia plików projektu należy tylko uzupełnić odpowiednie meta dane CVS dotyczące samego projektu i jego statusu w repozytorium CVS. Dokument [3] zawiera skrypt pozwalający zautomatyzować tę operację.

3.6. CVS i GNU Emacs

Podobnie jak RCS, system CVS jest dobrze zintegrowany z edytorem GNU Emacs, stanowiącym podstawowe narzędzie wielu programistów należących do społeczności *opensource*.

Standardowy pakiet *Version Control* potrafi współpracować nie tylko z RCS, lecz również z CVS. Jednak ponieważ funkcje *VC* są dość skromne w porównaniu do funkcji CVS, warto wykorzystać pakiet *PCL–CVS*.

Pakiet *PCL–CVS* jest zaawansowanym narzędziem integrującym GNU Emacs z systemem CVS. Autorem pakietu jest Per Cederqvist, znany autor podręcznika do systemu CVS ([6]). *PCL–CVS* pozwala na przeglądanie stanu plików znajdujących się pod kontrolą CVS. Na plikach można wykonywać praktycznie wszystkie operacje udostępniane przez CVS. Pakiet jest rzecz jasna w pełni obsługiwany z poziomu Emacsa, a wynik działania jego funkcji jest na bieżąco przechowywany w buforach edytora.

4. Narzędzia dla CVS

Duża popularność CVS sprawiła, że pojawiło się wiele dodatkowych narzędzi, które z nim współpracują. Powstało wiele interesujących interfejsów graficznych ułatwiających pracę z projektami zarządzanymi przez CVS.

Dwa uniwersalne narzędzia to *TkCVS* i *LinCVS*. Interfejs pierwszego z nich jest oparty na pakiecie Tcl/Tk, podczas gdy drugi wykorzystuje bibliotekę Qt. *TkCVS* jest dostępny pod adresem `www.twobarleycorns.net/tkcvs.html`, lub na serwisie SourceForge `sourceforge.net/projects/tkcvs`. Program *LinCVS* znajduje się na stronie `www.lincvs.org`.

Obydwa programy mają podobne funkcje, do których należą między innymi:

- udostępnianie zawartości katalogu kontrolowanego przez CVS w postaci graficznej,
- porównywanie wybranych plików i uzyskiwanie informacji o nich,
- graficzna reprezentacja drzewa zmian w projekcie,
- synchronizacja kopii roboczej z repozytorium.

TkCVS jest programem dojrzalszym, posiadającym funkcje niedostępne w *LinCVS*. Należą do nich między innymi funkcje pozwalające na bezpośrednią pracę z głównym repozytorium CVS w tym import modułów, generowanie łat pomiędzy różnymi wersjami modułów i inne. Przykładowa sesja pracy z *TkCVS* jest pokazana na Rysunku 2. Widoczne jest okno projektu, przeglądarka modułów w repozytorium i graficzne narzędzie do wyświetlania różnic pomiędzy wersjami plików.

5. Praca w sieci z CVS

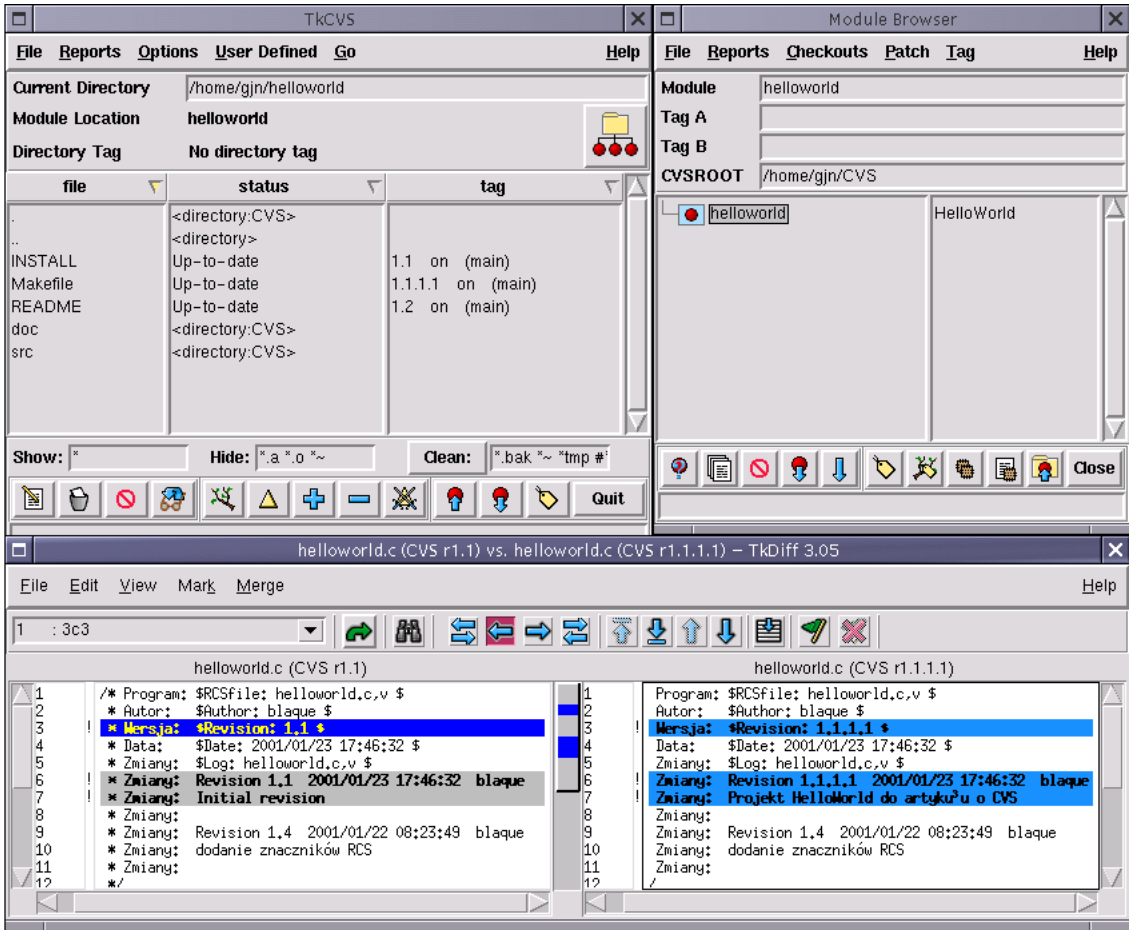
Opisując możliwości systemu CVS nie można zapomnieć o pracy w trybie klient/serwer z wykorzystaniem sieci. Do pracy w tym trybie konieczne jest dodatkowe skonfigurowanie CVS po stronie serwera. Należy w tym celu dopisać odpowiednią usługę do pliku `/etc/inetd.conf`:

```
2401  stream  tcp  nowait  root  /usr/bin/cvs
      cvs --allow-root=/usr/src/CVS pserver
```

Spowoduje to uruchamianie na porcie 2401 TCP usługi *pserver* pozwalającej na pracę CVS jako serwera. W podanym powyżej przykładzie należy rzecz jasna uzupełnić odpowiednio katalog repozytorium i ścieżkę dostępu do *cvs*.

Po stronie klienta wystarczy jedynie podać jako ścieżkę dostępu do repozytorium nazwę konta i katalog na komputerze w sieci:

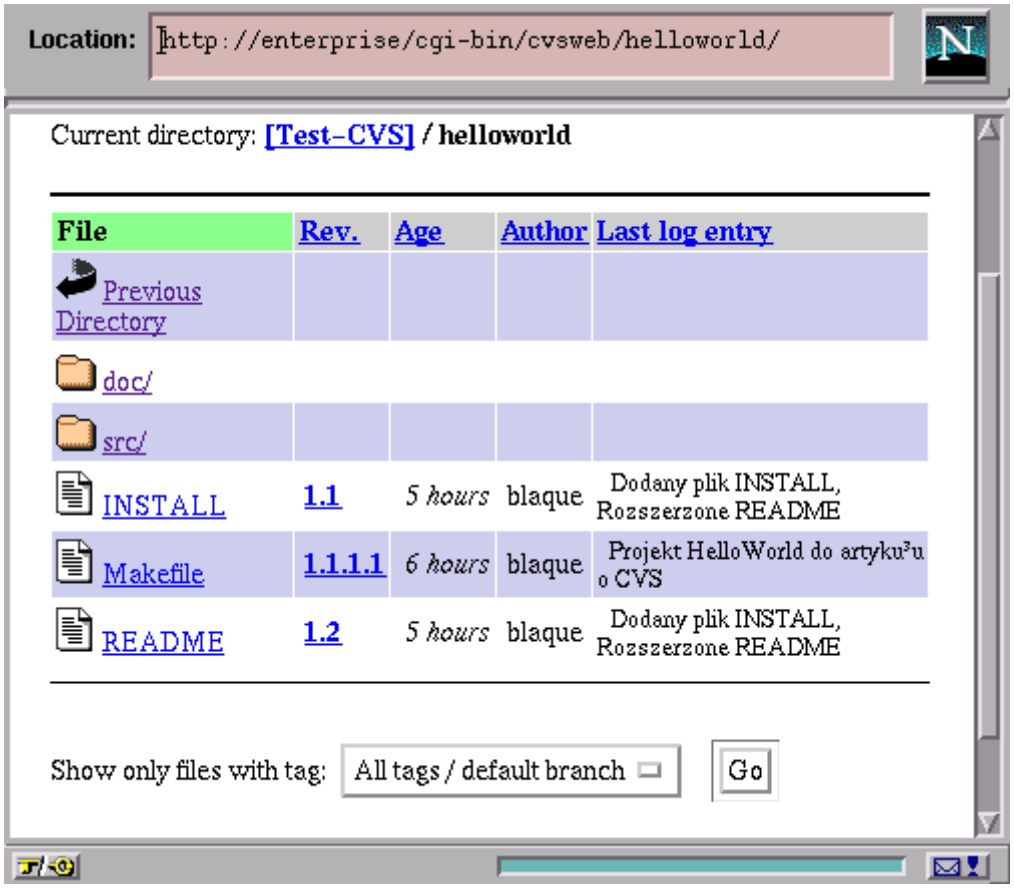
```
cvs -d :ext:user@cvs.example.org:/usr/src/cvsroot checkout helloworld
```



Rysunek 2: Praca z TkCVS

Klient CVS uzyska z repozytorium serwera drzewo projektu i utworzy na lokalnej maszynie kopie roboczą.

Ciekawym rozszerzeniem pracy z CVS w sieci jest klient *CVSweb*. Jest on zrealizowany jako skrypt CGI i umożliwia udostępnianie informacji o repozytorium przez serwer WWW. Na Rysunku 3 jest pokazane drzewo projektu HelloWorld w przeglądarce WWW podłączonej do *CVSweb*.



Rysunek 3: CVSweb

6. Podsumowanie

Zaprezentowany w artykule system RCS jest bardzo dobrym przykładem generycznego systemu kontroli wersji. CVS, stanowiący jego naturalne rozszerzenie, jest narzędziem o ogromnych możliwościach. Najlepiej świadczy o nich fakt, że zdecydowana większość projektów *opensource*, prowadzonych przez różne organizacje i firmy komercyjne opiera się właśnie na CVS. Na dzień dzisiejszy CVS stanowi *de facto* standard w sieciowych systemach zarządzania i dystrybucji kodu.

Warto jednak podkreślić, że system kontroli wersji jest przydatny praktycznie w każdej sytuacji, a nie tylko przy pracy z dużymi projektami. RCS czy CVS mogą oddać nieocenione usługi nawet pojedynczemu autorowi, czy programiście, realizującemu niewielki, składający się z zaledwie kilku plików projekt. Ich ścisła integracja z Emacsem z jednej strony, oraz narzędzia typu TkCVS z drugiej, umożliwiają szybkie wdrożenie się do pracy z nimi.

Systemy kontroli wersji stanowią dzisiaj podstawowy i niezbędny element infrastruktury projektów informatycznych. Pozwalają na ścisłą kontrolę, ale również na rejestrowanie wszyst-

kich modyfikacji projektu. Stanowią tym samym narzędzie pomocne nie tylko w zarządzaniu kodem, lecz także przy analizie pracy zespołu autorów projektu.

Literatura

[1] Walter F. Tichy, *RCS - A System for Version Control*, Department of Computer Sciences Purdue University, 1995.

[2] Walter F. Tichy, *RCSintro - Introduction to RCS commands*, 1993.

[3] Robert Kiesling, *The RCS MINI-HOWTO*, LDP, v1.4, 14 August 1997.

[4] Brian W. Kernighan, Dennis M. Ritchie, *Język ANSI C*, tłum. Danuta i Marek Kruszewscy, WNT, Warszawa 1994.

[5] *Concurrent Versions System* http://www.cvshome.org.

[6] Per Cederqvist, *Version Management with CVS*,

[7] *CVS – Concurrent Versions System*, man page.

[8] Bob Arnson, *CVS for new users*, http://www.cvshome.org/new_users.html.

[9] Brian Berliner, *CVS II: Parallelizing Software Development*, Prisma, Inc.

[10] Karl Fogel, *Open Source Development With CVS*, The Coriolis Group, http://cvsbook.red-bean.com.

[11] Alavoor Vasudevan, *CVS-RCS-HOWTO document for Linux*, LDP, v17.0, 20 Jan 2001.

[12] Per Cederqvist, *Pcl-cvs – The Emacs Front-End to CVS*.