

Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

**INSTRUÇÕES DE USO DE UM SISTEMA DE
CONTROLE DE VERSÕES RCS - REVISION
CONTROL SYSTEM**

**FÁTIMA YAMASHIRO
LAIRCE CASTANHERA BERALDI
ROSELY SANCHES**

Nº 44

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Set./1996

SYSNO	<u>906205</u>
DATA	<u> / / </u>
ICMC - SBAB	



UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS DE SÃO CARLOS
Departamento de Ciências da Computação e Estatística

Instruções de Uso de Um
Sistema de Controle de Versões
RCS - Revision Control System

Fátima Yamashiro
Lairce Castanhera Beraldi
Rosely Sanches

Setembro de 1996

Agradecimentos

À FAPESP e ao CNPq pelo apoio financeiro

Índice

1. Introdução	1
2. Apresentação do RCS	3
2.1. Funções do RCS	3
2.2. Comandos	4
2.3. Forma de armazenamento	4
2.4. Identificação automática	7
2.5. Controle de acesso e sincronização	9
2.6. Controle de configurações	9
3. Como iniciar o RCS	10
3.1. Detalhes Administrativos	10
3.2. Armazenando a Primeira Revisão	10
3.3. Recuperando e Editando Revisões	11
4. Comandos	12
4.1. <u>ci</u>	12
4.2. <u>co</u>	19
4.3. <u>ident</u>	23
4.4. <u>rcs</u>	24
4.5. <u>rcsclean</u>	29
4.6. <u>rcsdiff</u>	30
4.7. <u>rcsfreeze</u>	31
4.8. <u>rcsmerge</u>	32
4.9. <u>rlog</u>	33
5. Referências Bibliográficas	36

Índice de Figuras

Figura 2.1 - Uma árvore de revisões.....	6
--	---

Índice de Quadros

Quadro 2.1 - Comandos do RCS.....	5
Quadro 2.2 - <i>Keywords</i> e seus valores correspondentes	7

1. Introdução

“Durante o ciclo de vida do software, muitos itens de informação são produzidos. Esses itens englobam programas de computador (fonte e executável), documentos que descrevem o software (direcionados aos técnicos e aos usuários), estruturas de dados (contidas nos programas ou externas a ele), itens de teste, documentos de alteração, enfim toda a informação relacionada ao software durante sua construção e posteriormente nas eventuais manutenções efetuadas no mesmo. A todos esses itens de informação dá-se o nome de *Configuração de Software*.

A configuração de software é uma arma poderosa para o profissional de manutenção entender um software escrito e modificado por diferentes pessoas. No entanto, é muito importante a existência de itens de informação completos e atualizados. A desatualização da configuração acontece porque, cada vez que o software é submetido a uma alteração formal, os programas são alterados e essas alterações podem implicar em mudanças também nos itens de informação relacionados ao software. Para que seja possível ter uma configuração de software atualizada, é necessário que os itens de informação sejam “controlados” durante todo o ciclo de vida do software. A existência de uma configuração adequada e a garantia que os itens de informação dessa configuração reflitam a realidade, depende de efetivos *Procedimentos de Gerenciamento da Configuração*.”

Segundo Pressman [Pre92], o gerenciamento de configuração de software pode ser descrito como o procedimento de gerenciamento que tem como principais atividades:

A. Identificação da configuração: É a tarefa de garantir que cada item que compõe a configuração do software seja identificado de maneira consistente e significativa. Várias ferramentas automatizadas (por exemplo, CCC, RCS, SCCS, CMS, NSE) foram desenvolvidas para auxiliar nas tarefas de identificação (e outras atividades do gerenciamento de configuração de software).

B. Controle de versões: São procedimentos aplicados para controlar as diferentes versões dos itens de configuração de software e controlar as versões do software liberadas. As várias versões surgem através da evolução do software como um todo. Essas versões podem ainda conter variações, que são as variantes de uma mesma versão, pois cada versão é composta de vários componentes que, em alguns casos, podem necessitar conter características diferentes.

C. Controle de mudanças: É a combinação de procedimentos humanos e ferramentas automatizadas para proporcionar um mecanismo para o controle das mudanças nos objetos de configuração. Dois elementos importantes do controle de mudanças são o controle de acesso e o controle de sincronização. O controle de acesso governa quais engenheiros de software têm autoridade para acessar e modificar um objeto da configuração em particular. O controle de sincronização ajuda a garantir que mudanças paralelas, executadas por duas pessoas diferentes, não se sobreponham uma à outra.

D. Auditoria de configuração: A auditoria de configuração de software é uma atividade que tem como objetivo garantir que as mudanças foram adequadamente implementadas. A auditoria pergunta e responde a questões que avaliam essas mudanças.

E. Relato de status: Esta atividade, às vezes denominada *contabilidade de status*, ajuda a eliminar problemas causados pela falta de comunicação entre os membros do projeto, através do registro e relatório das informações que são necessárias para a avaliação da mudança de uma configuração.

Como podemos ver, o gerenciamento de configuração de software necessita de ferramentas automatizadas que tornem essa tarefa mais simples. O RCS é uma dessas ferramentas. Ele automatiza o controle de versões, dando suporte também para o controle de mudanças. Apesar de não ser propriamente uma ferramenta de gerenciamento de configuração (ele implementa apenas algumas atividades do gerenciamento de configuração), esse sistema fornece um grande auxílio para facilitar essa importante tarefa.

2. Apresentação do RCS

O RCS (*Revision Control System*) é uma ferramenta de auxílio ao controle de versões. É baseado em um conjunto de comandos UNIX e foi desenvolvido por Walter Tichy, da Universidade de Purdue, em 1985, como um melhoramento do SCCS (*Source Code Control System*).

Sua principal função é controlar grupos de revisões. Um grupo de revisões é um conjunto de arquivos texto (chamados revisão) que mantêm uma seqüência cronológica entre si, e que representam a seqüência de evolução do texto inicial. Essas revisões são armazenadas em forma de uma árvore, sendo que a primeira revisão é chamada raiz e a última revisão do tronco é chamada *head*.

A vantagem do uso desse sistema concentra-se no fato de que para se modificar um documento sob controle do RCS, é preciso antes recuperá-lo do sistema. O RCS então gerencia o acesso e as modificações realizadas sobre o documento. Outra vantagem é que ao invés de armazenar integralmente todas as versões de um mesmo documento, o RCS guarda apenas as diferenças existentes entre as versões. Isso implica em uma economia grande de espaço de armazenamento.

O RCS foi originalmente projetado para manipular apenas código fonte. Mais tarde percebeu-se que ele seria útil também para controlar qualquer arquivo que fizesse parte da configuração do software. No entanto, ele pode trabalhar apenas com arquivos texto e código objeto, e sua utilização em arquivos não-texto (binários) resulta em corrupção dos dados. Arquivos gráficos gerados pelo Xfig do UNIX, por exemplo, podem ser controlados normalmente, já que são gravados em formato texto.

A interface com o usuário consiste apenas de comandos de linha, o que torna difícil para o usuário principiante explorar toda a potencialidade do sistema, devido ao enorme número de opções fornecidas. Entretanto, no início, o usuário precisa apenas conhecer os dois comandos principais, *ci* e *co*. Após se familiarizar com o sistema, poderá conhecer os outros comandos e utilizá-los quando necessário.

Além da versão para UNIX, existe uma versão para DOS, também de domínio público. No entanto, para este trabalho, foi estudada apenas a versão para UNIX.

2.1. Funções do RCS

- Armazena e recupera múltiplas revisões de texto. O RCS salva todas as revisões antigas de maneira eficiente, com economia de espaço. Mudanças não destroem o original, pois as revisões anteriores permanecem acessíveis (o que não é possível em sistemas operacionais comuns). Revisões podem ser recuperadas de acordo com números de revisão, nomes simbólicos, datas, autores e estados.

- Mantém uma história completa das mudanças. O RCS registra todas as mudanças entre as revisões automaticamente. Junto ao texto de cada revisão, o RCS armazena o autor, a data e hora de armazenamento e uma mensagem de *log* resumindo a mudança. O sistema de *log* torna fácil constatar o que aconteceu a um módulo, sem ter que comparar listagens-fonte.

- Resolve conflitos de acesso. Quando dois ou mais programadores querem modificar a mesma revisão, o RCS os alerta e impede que uma modificação corrompa a outra. Além disso, permite que um usuário bloqueie (*lock*) uma revisão, garantindo que somente este usuário possa armazenar uma modificação da revisão bloqueada.

- Mantém uma árvore de revisões. O RCS pode manter linhas separadas de desenvolvimento para cada módulo (armazenadas em ramos diferentes). Ele armazena uma estrutura de árvore que representa as relações ancestrais entre revisões.

- Une duas revisões, resolvendo conflitos. Duas linhas separadas de desenvolvimento de um módulo podem ser unidas. Se as revisões a serem unidas afetam as mesmas seções de código, o RCS alerta o usuário sobre mudanças sobrepostas.

- Controla versões e configurações. As revisões podem receber nomes simbólicos e atributos de estado como experimental (Exp), estável (Stab), liberada (Rel), etc. Com essas facilidades, configurações de módulos podem ser descritas simples e diretamente.

- Identifica automaticamente cada revisão com nome, número, hora de criação, autor, etc. A identificação é como um selo que pode ser "colado" em um lugar apropriado no texto de uma revisão. A identificação torna simples determinar quais revisões de quais módulos constituem uma dada configuração.

- Minimiza o armazenamento secundário. O RCS precisa de pouco espaço extra para as revisões (apenas as diferenças). Se revisões intermediárias são apagadas, os deltas (diferenças) correspondentes são gerados apropriadamente.

2.2. Comandos

O RCS possui nove comandos, sendo que o comando `rcsfreeze` ainda é apenas um *shell script*. O quadro 2.1 apresenta todos os comandos com uma descrição resumida de suas funções.

2.3. Forma de armazenamento

O RCS armazena as revisões em forma de uma árvore, onde há uma raiz (normalmente a revisão 1.1), um tronco e ramos. Um número de revisão é normalmente formado por dois campos separados por um ponto, sendo o primeiro chamado número de versão e o segundo, número de nível. O comando `ci` determina automaticamente um novo número de revisão incrementando o número de nível da revisão anterior, a partir da qual a nova revisão foi gerada. O número de versão deve ser incrementado explicitamente pelo usuário, quando achar necessário, através da opção `-r` do comando `ci`. O número de versão indica qual a versão em que aquele módulo do software se encontra.

Dois campos a mais no número de revisão indicam que esta revisão faz parte de um ramo. O terceiro campo indica o número do ramo e o quarto campo indica o nível da revisão dentro desse ramo. Por exemplo, os números de revisão 1.3.1.1 e 1.3.1.2 indicam respectivamente a primeira e a segunda revisões do primeiro ramo da revisão 1.3; as revisões do

segundo ramo da revisão 1.3 são numeradas como 1.3.2.1, 1.3.2.2, e assim por diante. Além disso, pode-se ter ramos a partir de revisões de outros ramos.

Por questão de economia de espaço, o RCS armazena as revisões na forma de deltas, isto é, apenas as diferenças entre as revisões, sendo apenas a última revisão armazenada integralmente, a qual é chamada de cabeça (*head*) da árvore. A interface com o usuário não permite a visualização dos deltas.

Quadro 2.1 - Comandos do RCS

Comando	Descrição
ci	<i>Check-in</i> revisões, isto é, coloca sob controle do RCS novas revisões de um arquivo.
co	<i>Check-out</i> revisões, isto é, obtém uma cópia de uma revisão específica que está sob controle do RCS.
ident	Imprime uma cópia dos identificadores contidos no arquivo especificado. Os identificadores contém algumas informações sobre o <i>status</i> do arquivo.
rscs	É o comando utilizado para gerenciar o sistema. Modifica atributos de arquivos sob controle do RCS, tais como lista de acesso, texto descritivo, etc.
rscs-clean	Limpa o diretório de trabalho, removendo os arquivos que foram extraídos mas nunca modificados.
rscs-diff	Compara duas revisões, imprimindo as diferenças existentes.
rscs-freeze	“Congela” uma configuração, ou seja, atribui o mesmo nome ou número simbólico a uma dada revisão de todos os arquivos sob controle do RCS.
rscs-merge	Une duas revisões, em relação a um antecessor comum.
rlog	Imprime informações sobre um arquivo sob controle do RCS.

Um delta é uma sequência de comandos de edição que transforma uma *string* em outra. Os deltas empregados pelo RCS são baseados em linha, o que significa que os únicos comandos de edição permitidos são a inserção e remoção de linhas. Mesmo que apenas um caracter em uma linha seja mudado, a linha inteira é considerada como mudada. O comando `rscs-diff`, que mostra as diferenças existentes entre duas revisões, funciona de acordo com esse conceito, comparando linha por linha e mostrando as linhas que diferem entre as revisões.

Os deltas reduzem o espaço consumido, mas aumentam o tempo de acesso. Para obter um tempo de acesso razoavelmente rápido, o RCS utiliza o conceito de delta negativo, isto é, armazena-se integralmente a revisão mais recente do tronco, enquanto todas as outras revisões são armazenadas na forma dos deltas necessários para recuperá-las. Esta forma de

armazenamento diminui o tempo de acesso, já que de acordo com uma pesquisa efetuada na universidade de Purdue, verificou-se que 95% das operações de *check-out* (recuperação de revisões de arquivos sob controle do sistema) eram realizadas sobre a última revisão.

Para os ramos, o RCS utiliza deltas positivos, isto é, são armazenados os deltas necessários para se recuperar as revisões mais recentes. Assim, uma revisão do ramo é montada a partir do seu ancestral no tronco, não precisando armazenar a revisão inteira.

Os ramos são uma estrutura útil, já que o RCS não permite o armazenamento de revisões intermediárias às já existentes, o que muitas vezes é necessário. Para resolver esse problema, pode-se armazenar temporariamente uma revisão em um ramo e mais tarde uni-la a uma revisão do tronco através do comando *rctmerge*.

A figura 2.1 apresenta um esquema de uma árvore de revisões. Os deltas são representados por triângulos e as flechas indicam as revisões que podem ser geradas a partir de cada delta. A revisão mais recente (*head*) é representada por um quadrado.

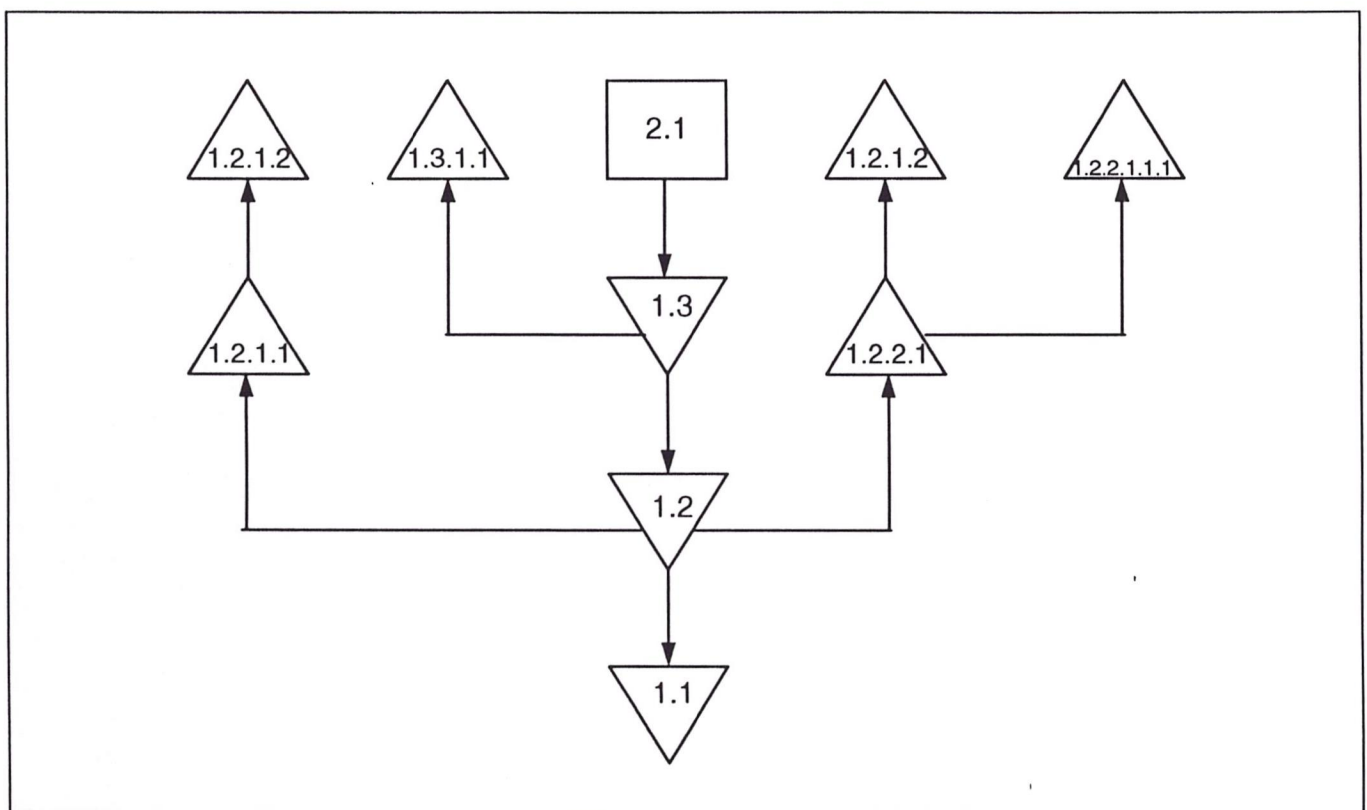


Figura 2.1 - Uma árvore de revisões

Por exemplo, se o usuário deseja recuperar a revisão 1.2.1.1, são necessários os seguintes passos: primeiro é gerada a revisão 1.3, a partir da revisão 2.1 (*head*), utilizando o delta 1.3; a seguir, através da aplicação do delta 1.2 sobre a revisão 1.3, é obtida a revisão 1.2; finalmente, a partir da revisão 1.2 e do delta 1.2.1.1, a revisão 1.2.1.1 é recuperada e fornecida ao usuário.

2.4. Identificação automática

O RCS permite a identificação automática dos arquivos por ele controlados através do uso de *keywords* colocadas no arquivo fonte em forma de comentário. Esses identificadores são expandidos (substituídos pelas informações correspondentes) pelo comando `co`, ao se recuperar (*check-out*) uma revisão, e as informações são atualizadas automaticamente. Por exemplo, o identificador `$Log$` acumula todas as mensagens de *log* que são solicitadas durante o *check-in*, permitindo que se mantenha a história completa de uma revisão diretamente dentro dela.

Inicialmente, o usuário insere no arquivo *strings* da forma `$keyword$`. Ao realizar um *check-out*, o comando `co` substitui essas *strings* por *strings* da forma `$keyword:valor$`. Se uma revisão contendo identificadores assim expandidos é novamente *checked-in*, o campo *valor* será substituído durante o próximo *check-out*, isto é, os valores de *keywords* são atualizados automaticamente a cada *check-out*.

O RCS também permite o uso de *keywords* no código objeto. Para propagar um identificador no código objeto, basta colocá-lo em uma *string* de caracteres literais. Em C, isto é feito do seguinte modo:

```
static char rcsid[] = "$Id$";
```

O comando `ident` extrai uma cópia de tais identificadores de qualquer arquivo de trabalho, inclusive do código objeto, devidamente expandidos e atualizados pelo comando `co`.

Arquivo de trabalho é o arquivo que pode ser editado pelo usuário, enquanto arquivo RCS é o nome dado ao arquivo sob controle do RCS. Um nome de arquivo contendo um sufixo RCS representa um arquivo RCS, enquanto todos os outros são assumidos como sendo arquivos de trabalho contendo novas revisões. Em ambiente UNIX, o sufixo normalmente utilizado é `.v`. Por exemplo, o arquivo RCS correspondente ao arquivo de trabalho *exemplo.c* é *exemplo.c.v*.

O sistema automaticamente armazena algumas informações importantes para o controle, mesmo que se decida pela não utilização de *keywords*. Porém, essas informações não são visíveis no próprio texto da revisão, como as *keywords*, e nem podem ser extraídas pelo comando `ident`.

O quadro 2.2 apresenta todas as *keywords* com os correspondentes valores e um exemplo ilustrativo.

Quadro 2.2 - *Keywords* e seus valores correspondentes

<code>\$Author\$</code>	Imprime o <i>login name</i> do usuário que depositou (<i>checked-in</i>) a revisão. Ex.: <code>\$Author: fatima \$</code>
<code>\$Date\$</code>	Imprime a data e a hora em que a revisão foi depositada. A hora <i>default</i> adotada é a da UTC (<i>Coordinated Universal Time</i>), que é três horas adiantada com relação ao horário de Brasília. Ex.: <code>\$Date: 1995/07/14 16:05:23 \$</code>

(continua)

Quadro 2.2 - *Keywords* e seus valores correspondentes (continuação)

\$Header\$	Imprime um cabeçalho padrão contendo o nome do arquivo RCS com o caminho completo (diretório e subdiretórios a que pertence), o número de revisão, a data de <i>check-in</i>, o autor, o estado e o bloqueador (<i>locker</i>), se a revisão estiver bloqueada. Ex.: \$Header: /master/home/ESwSI/fatima/RCS/exemplo.c,v 1.1 1995/07/14 16:05:23 fatima Exp fatima \$
\$Id\$	O mesmo que \$Header\$, exceto que o nome do arquivo RCS não é acompanhado do caminho. Ex.: \$Id: exemplo.c,v 1.1 1995/07/14 16:05:23 fatima Exp \$
\$Locker\$	Imprime o <i>login name</i> do usuário que bloqueou a revisão (o valor é vazio se a revisão não está bloqueada). Ex.: \$Locker: fatima \$ ou \$Locker: \$
\$Log\$	Imprime a mensagem de <i>log</i> fornecida durante o <i>check-in</i>, precedida por um cabeçalho contendo o nome do arquivo RCS, o número de revisão, o autor e a data de <i>check-in</i>. As mensagens de <i>log</i> existentes anteriormente são mantidas, apenas inserindo-se a nova mensagem. Isto é útil para acumular a história completa das mudanças no próprio código fonte. Como as revisões são armazenadas na forma de diferenças, cada mensagem de <i>log</i> é armazenada fisicamente apenas uma vez, independente do número de revisões, o que implica em um gasto de espaço insignificante.
\$RCSfile\$	Imprime o nome do arquivo RCS sem o caminho. Ex.: \$RCSfile: exemplo.c,v \$
\$Revision\$	Imprime o número de revisão determinado para a revisão. Ex.: \$Revision: 1.1 \$
\$Source\$	Imprime o caminho completo do arquivo RCS. Ex.: \$Source: /master/home/ESwSI/fatima/RCS/exemplo.c,v \$
\$State\$	Imprime o estado determinado para a revisão através da opção <i>-s</i> do comando <i>rcs</i> ou <i>ci</i>. Ex.: \$State: Exp \$

2.5. Controle de acesso e sincronização

O controle de acesso, que determina quais engenheiros de software têm permissão para acessar e modificar uma revisão específica, pode ser feito através de uma lista de acesso, relativa a cada arquivo RCS, contendo os *login names* dos engenheiros que têm a permissão.

Para efetuar o controle de sincronização, o RCS possui um sistema de *lock*, através do comando `co` com a opção `-l`, que bloqueia o objeto que foi recuperado (*checked-out*), impedindo que qualquer outra atualização possa ser feita sobre a mesma revisão, até que a revisão modificada seja recolocada sob controle do RCS. Isto impede que mudanças paralelas e conflitantes sejam armazenadas ao mesmo tempo, comprometendo uma à outra.

2.6. Controle de configurações

O comando `rcsfreeze` permite “congelar” uma configuração. Isto é feito associando um nome simbólico e comum a todos os itens que compõem a configuração do software. Assim, todos os itens podem ser acessados ao mesmo tempo através de um nome comum, compondo, assim, uma versão do software. Infelizmente, o RCS não suporta o conceito de variantes.

3. Como iniciar o RCS

Esta seção fornece algumas diretrizes para auxiliar o usuário principiante a começar a utilizar o RCS. No entanto, para explorar melhor os recursos do sistema, o usuário não deve ficar restrito a essas instruções simples. Todos os comandos são descritos detalhadamente na próxima seção.

3.1. Detalhes Administrativos

Antes de se começar a utilizar o RCS, é preciso definir a parte administrativa, como formação das equipes, determinação dos gerentes, etc., que o RCS não administra, apesar de deixar subentendido que tais formalismos são necessários.

É preciso haver um gerente, ou superusuário, em cuja área ficarão os arquivos RCS sob responsabilidade do seu grupo.

Inicialmente, pode-se criar um diretório chamado RCS para armazenar todos os arquivos que deverão ficar sob controle do RCS (apenas por uma questão de organização; não é obrigatório fazer isso). Isso é feito através do comando:

```
% mkdir RCS
```

Cada programador poderá trabalhar em sua própria área, escolhendo um diretório de trabalho e colocando dentro dele um *symbolic link*, chamado RCS, para o diretório que contém os arquivos RCS com que irá trabalhar. Esse *symbolic link* permite que o programador utilize os comandos `co` e `ci` a partir de sua própria área, especificando apenas o nome do arquivo. O *symbolic link* pode ser criado utilizando-se o comando `ln` do UNIX, com a opção `-s`. Por exemplo, para criar um *symbolic link* entre o diretório atual e o diretório RCS localizado em `/master/home/ESwSI/fatima`, basta utilizar o comando:

```
% ln -s ~fatima/RCS RCS
```

Os programadores também podem trabalhar todos na mesma área, mas deve-se tomar cuidado ao utilizar os comandos `co` e `ci`, pois os arquivos de trabalho do mesmo arquivo RCS têm sempre o mesmo nome, podendo causar confusão quando se precisa manipular revisões diferentes do mesmo arquivo. Essas confusões podem surgir porque, a não ser que o arquivo contenha algum identificador, é impossível saber diretamente qual revisão deu origem ao arquivo de trabalho.

3.2. Armazenando a Primeira Revisão

Antes de utilizar qualquer comando do RCS, é preciso inserir os identificadores desejados dentro do arquivo, como comentários. Em geral, apenas os identificadores `$Id$` e `$Log$` já são suficientes para identificar a revisão.

A seguir, utilize o comando de *check-in* (`ci`). Para colocar o arquivo, por exemplo `teste.c`, sob controle do RCS, basta chamar:

```
% ci teste.c
```


Esse comando cria um arquivo RCS no diretório RCS, armazena teste.c como a revisão 1.1, e apaga teste.c. Ele também pede uma descrição, que deve ser um resumo do conteúdo do arquivo. Todos os comandos de *check-in* posteriores irão pedir uma mensagem de *log*, que deve resumir as mudanças feitas.

Os arquivos RCS se distinguem dos arquivos de trabalho por um sufixo, geralmente o sufixo ,v. No exemplo, seria criado o arquivo teste.c,v. O RCS também permite que se utilize o sufixo vazio, o que implica que os arquivos RCS e de trabalho terão o mesmo nome. Neste caso, todos os arquivos RCS deverão obrigatoriamente estar dentro de um diretório chamado RCS.

3.3. Recuperando e Editando Revisões

Para recuperar o arquivo de trabalho teste.c, basta usar o comando

```
% co teste.c
```

Este comando extrai do arquivo RCS teste.c,v a última revisão, isto é, faz uma cópia dessa revisão e a coloca no arquivo de trabalho teste.c. Esse arquivo terá permissão apenas de leitura. Se você quer editar teste.c, deve bloqueá-lo através do sistema de *lock* com o seguinte comando:

```
% co -l teste.c
```

O sistema de *lock* garante que somente você possa armazenar (*check-in*) a próxima atualização, evitando problemas se várias pessoas trabalham no mesmo arquivo. Mesmo se uma revisão está bloqueada (*locked*), ela ainda pode ser recuperada (*checked-out*) para leitura, compilação, etc. Tudo que o sistema de *lock* faz é evitar um *check-in* por parte de qualquer outro que não seja o bloqueador.

Após editar o arquivo, você pode recolocá-lo sob controle do RCS simplesmente chamando novamente

```
% ci teste.c
```

O número de revisão será incrementado automaticamente.

Se você tentar armazenar um arquivo que você não bloqueou ao recuperar, o comando *ci* irá imprimir uma mensagem de erro e interromper a execução. No entanto, será muito tarde para dar um comando *co* com *locking*, pois outro *check-out* iria sobrepor suas modificações. Ao invés disso, você pode chamar

```
% rcs -l teste.c
```

Esse comando irá bloquear a última revisão para você, a menos que alguém já a tenha bloqueado antes. Neste caso, você terá que negociar com essa pessoa, ou, pelo menos temporariamente, armazenar a sua revisão em um ramo, especificando o número com a opção *-r*. Para criar um ramo a partir de qualquer revisão, não é preciso que essa revisão esteja bloqueada.

4. Comandos

Como já foi visto anteriormente, o RCS tem nove comandos básicos para realizar o controle de versões. No entanto, o usuário principiante precisa apenas saber utilizar os comandos `ci` e `co`, e com o tempo irá descobrindo quando os outros comandos são úteis.

O comando `rscsfreeze` é apenas um *shell script*, uma versão teste de um comando para congelar configurações, que não deve, como o próprio autor ressalta, ser considerado como a melhor solução para realizar essa tarefa.

A seguir são descritos os comandos do RCS com todas as respectivas opções.

Os colchetes indicam opcionalidade.

Podemos observar que há várias opções que podem ser utilizadas em vários comandos diferentes. Em geral, essas opções desempenham a mesma função e têm a mesma sintaxe em qualquer dos comandos.

4.1. `ci`

`ci [opções] nome_arquivo`

O comando `ci` armazena (*check-in*) novas revisões em arquivos RCS.

Para poder utilizar este comando, o *login name* do usuário deve estar na lista de acesso do arquivo, a menos que a lista esteja vazia, ou o usuário seja o superusuário ou o proprietário do arquivo (o arquivo está armazenado em sua área). Cada arquivo RCS tem sua própria lista de acesso. Além disso, é necessário que haja alguma revisão bloqueada pelo usuário.

O comando `ci` deposita o conteúdo de cada arquivo de trabalho no correspondente arquivo RCS, removendo o arquivo de trabalho, a não ser que o usuário forneça a opção `-u` ou `-l`. Como já foi dito anteriormente, um arquivo RCS é todo aquele que contém um sufixo RCS (em geral, no UNIX, o sufixo *default* é `,v`), enquanto todos os outros são considerados arquivos de trabalho contendo novas revisões.

Ao se depositar (*check-in*) uma nova revisão, o comando `ci` pede ao usuário uma mensagem de *log*. Essa mensagem deve resumir a mudança feita e servirá posteriormente para que o usuário identifique rapidamente as mudanças efetuadas na revisão em questão. Ela deve terminar com fim-de-arquivo ou uma linha contendo apenas ponto-final (':').

Se o número da revisão não é especificado pelo usuário (através de uma das opções), este número é gerado automaticamente, incrementando o número de nível da revisão bloqueada pelo usuário. Se o usuário tiver mais de uma revisão bloqueada em seu nome, o comando `ci` é abortado e pede-se que o usuário especifique o número de revisão.

Ao efetuar o *check-in*, o comando *ci* verifica as diferenças existentes entre a nova revisão e a revisão anterior, para gerar o delta. Se a nova revisão não contém nenhuma mudança, o usuário é avisado de que o arquivo não foi modificado e o *lock* é removido.

Por exemplo, ao armazenar a segunda revisão do arquivo teste.c, teríamos:

```
% ci teste.c
teste.c,v <-- teste.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Acrescimo da funcao testel().
>> .
done
```

Se o arquivo RCS ainda não existe, *ci* o cria e deposita o conteúdo do arquivo de trabalho como a revisão inicial (o número *default* é 1.1, mas *ci* permite que o usuário especifique o número que desejar). A lista de acesso é inicializada como vazia. Ao invés da mensagem de *log*, é requerido um texto descritivo, que deve resumir o conteúdo do arquivo armazenado.

Exemplo:

```
% ci exemplo.c
exemplo.c,v <-- exemplo.c
enter description, terminated by single '.' or end of file:
NOTE: This is NOT the log message!
>> Programa simples para exemplo dos comandos do RCS.
>> .
initial revision: 1.1
done
```

As opções do comando *ci* permitem ao usuário:

- especificar o número de revisão da nova revisão (opção *-r*);
- realizar um comando *co* sobre a nova revisão, automaticamente, podendo bloqueá-la ou não (*-l* ou *-u*);
- impor a hora de modificação da revisão recuperada (*-M*);
- obrigar que a nova revisão seja armazenada, mesmo se não contém modificações (*-f*);
- especificar a data e hora de *check-in*, autor e estado da revisão (*-d*, *-w*, *-s* ou *-k*);
- modificar mensagens de *log* (*-m*);
- determinar para a nova revisão um nome simbólico, que poderá ser utilizado para identificar a revisão da mesma forma que o número de revisão (*-n* ou *-N*);
- fornecer o texto descritivo do arquivo RCS (*-t*);
- especificar os possíveis sufixos utilizados para o arquivo RCS (*-x*);

- determinar a versão do RCS a ser emulada (-v);
- permitir ou não permitir interação (-I ou -q).

A seguir, todas as opções são explicadas mais detalhadamente:

`-r[rev]` : Armazena (*checks-in*) uma nova revisão atribuindo a ela o número rev, libera o *lock* correspondente e remove (apaga) o arquivo de trabalho. É o *default*.

rev deve ser maior que o número da última revisão ou, caso contrário, deve indicar o início de um novo ramo.

Se rev for omitido, *ci* obtém o número de revisão incrementando o número da revisão precedente (aquela que estiver bloqueada pelo usuário).

Esta opção não tem o significado usual em *ci*. Em outros comandos RCS, `-r` simplesmente especifica um número de revisão, mas em *ci* ele também libera um *lock* e remove o arquivo de trabalho.

Ex.:

```
% ci -r1.3 exemplo.c
```

irá armazenar a nova revisão contida no arquivo de trabalho *exemplo.c*, com o número de revisão 1.3. Se a revisão 1.2 for a última do ramo, o comando acima sem a opção `-r` terá o mesmo efeito, já que *ci* incrementa automaticamente o número de revisão.

```
% ci -r1.1.2 exemplo.c
```

irá iniciar o segundo ramo da revisão 1.1, armazenando a nova revisão no início desse novo ramo. Este comando é equivalente a

```
% ci -r1.1.2.1 exemplo.c
```

```
% ci -r2 exemplo.c
```

irá iniciar a versão 2 de *exemplo.c*, armazenando a nova revisão com o número 2.1. Essa “liberação” (*release*) da versão 1 pode ser feita a qualquer momento, desde que a equipe esteja de acordo com essa decisão.

`-l[rev]` : O mesmo que `-r`, exceto que ele executa um `co -l` adicional para a revisão depositada. Assim, a revisão depositada é imediatamente recuperada novamente e bloqueada (*locked*).

Isto é útil para salvar uma revisão que ainda se quer continuar editando após o *check-in*.

Ex.:

```
% ci -l1.3 exemplo.c
```

terá o mesmo resultado do comando `ci` normal, só que ao invés de remover o arquivo de trabalho, recupera-o imediatamente após o *check-in* através de um comando `co`, bloqueando a revisão para o usuário. Dessa forma, o arquivo de trabalho continuará disponível para modificações logo após o *check-in*.

`-u[rev]` : O mesmo que `-l`, exceto que a revisão depositada não é bloqueada. Isto permite que se leia o arquivo de trabalho imediatamente após o *check-in*.

Esta opção é útil se o arquivo é necessário para compilação após o *check-in*.

Ex.:

```
% ci -u1.3 exemplo.c
```

funciona do mesmo modo que `ci -l`, mas sem bloquear o arquivo de trabalho. Assim, o arquivo ficará disponível apenas para leitura, sendo útil quando não se quer acessar o arquivo, sem modificá-lo.

Obs.: As três opções acima são mutuamente exclusivas. Por exemplo, `ci -u -r` é equivalente a `ci -r`, porque `-r` não leva em conta `-u`. Além disso, se mais de um número de revisão é especificado pelas opções acima, `ci` imprime uma mensagem de advertência e executa o comando sobre a última revisão especificada.

`-f[rev]` : Obriga um depósito. A nova revisão é depositada mesmo se ela não é diferente da revisão precedente.

Ex.:

```
% ci -f1.3 exemplo.c
```

irá armazenar a nova revisão sem testar se o arquivo foi modificado ou não. Sem essa opção, o comando `ci` compara a nova revisão com a revisão precedente. Caso não haja modificações, `ci` apenas imprime uma mensagem de advertência avisando que o arquivo não foi modificado, e não executa nada.

`-k[rev]` : Procura os valores de *keywords* no arquivo de trabalho para determinar seu número de revisão, data de criação, estado e autor e designa esses valores à revisão depositada, ao invés de computá-los localmente. Isto é, reutiliza os dados anteriores, ao invés de utilizar os dados atuais. Se o número de revisão é fornecido através de alguma opção, utiliza esse número, e não o número encontrado no identificador. Também gera uma mensagem *default* de *login* observando o *login* do usuário e a data real de *check-in*. Se não há *keywords* no arquivo e nem são especificadas através de opções, utiliza as informações atuais, gerando, mesmo assim, a mensagem de *log*. Esta opção é útil para distribuição de *software*. Uma revisão que é mandada a vários

locais deveria ser armazenada (*checked-in*) com a opção `-k` nesses locais para preservar o número, data, autor e estado originais.

Ex.:

```
% ci -k exemplo.c
```

irá atribuir à nova revisão a data de criação, estado e autor relativos à revisão precedente. O número de revisão é determinado incrementando-se o número de revisão contido nas *keywords* do arquivo. A mensagem *default* de *log* avisa que foi utilizado a opção `-k` e informa o autor e data reais de *check-in*.

`-q[rev]` : Modo não-interativo; os resultados ou mensagens de erro não são informados (impressos na saída padrão). Uma revisão que não é diferente da precedente não é depositada, a menos que a opção `-f` seja utilizada.

`-I[rev]` : Modo interativo; o usuário é *prompted* e questionado mesmo se a entrada padrão não é um terminal. É o oposto de `-q`.

`-d[data]` : Utiliza data para a data e hora de *check-in*. Se houver espaço entre o dia e a hora, é preciso que data esteja entre aspas. O formato default para data é dado pela seguinte ordem: ano, mês, dia, hora, minuto e segundo (do mais para o menos significativo). Esta opção é útil para "mentir" (*lying*) sobre a data de *check-in*, e para a opção `-k`, quando nenhuma data está disponível. Se data está vazio, a data da última modificação do arquivo de trabalho é usada.

Ex.:

```
% ci -d"1995/02/28 14:10:00" exemplo.c
```

irá registrar o dia 28/02/95 e a hora 14:10hs como a data de *check-in* da nova revisão, qualquer que seja a data atual.

`-M[rev]` : Registra a hora da modificação de qualquer novo arquivo de trabalho como a data da revisão recuperada. Esta opção é útil quando utilizada em conjunto com `-l` ou `-u`.

Por exemplo, o comando

```
% ci -d -M -u exemplo.c
```

não altera a hora de modificação de *exemplo.c*, mesmo se o conteúdo desse arquivo muda devido a substituições de *keywords* (o comando `co` substitui as *keywords* sempre que é executado).

`-msg` : Usa a *string msg* como a mensagem de *log* para todas as revisões que estão armazenadas, removendo as mensagens de *log* antigas.

Ex.:

```
% ci -m"Teste da opção -m" exemplo.c
```

irá substituir todas as mensagens de *log* das outras revisões pela mensagem fornecida. Para substituir a mensagem de apenas uma revisão, deve-se usar o comando `rcs`.

-nnome : Designa o nome simbólico nome para o número da revisão que está sendo armazenada. O comando `ci` com esta opção imprime uma mensagem de erro se nome já está designado a outra revisão.

Ex.:

```
% ci -nteste exemplo.c
```

estabelece o nome simbólico "teste" à nova revisão. Quando se quiser recuperar essa revisão, ao invés do número de revisão, basta fornecer esse nome simbólico.

-Nnome : O mesmo que `-n`, exceto que não leva em consideração uma prévia designação de nome. Neste caso, simplesmente associa o nome simbólico nome à revisão de número rev, e a revisão que antes estava associada a nome passa a não ter nenhum nome simbólico.

-sestado : Estabelece o estado da revisão que está sendo armazenada como estado. Os estados possíveis sugeridos pelo RCS são: Exp (experimental), Stab (estável) e Rel (liberada); no entanto ele não impede que o usuário determine quaisquer outros nomes para os estados. O estado *default* é Exp.

Ex.:

```
% ci -sExp exemplo.c
```

-tarquivo : Utiliza o texto contido em arquivo como o texto descritivo do arquivo RCS. Este arquivo não pode começar com '-' (provavelmente para não confundir com a opção `-t-string`, explicada logo abaixo).

Ex.:

```
% ci -ttexto.txt exemplo.c
```

assume tudo que estiver contido no arquivo `texto.txt` como texto descritivo do arquivo RCS `exemplo.c,v`.

-t-string : Utiliza o texto de string como o texto descritivo do arquivo RCS. Esse texto deve estar entre aspas.

Ex.:

```
% ci -t-"Exemplo para a opcao -t" exemplo.c
```

assume a *string* "Exemplo para a opcao -t" como texto descritivo do arquivo RCS exemplo.c,v.

Obs.: A opção `-t`, em ambas as suas formas, tem efeito apenas durante um *check-in* inicial (a primeira revisão a ser inserida); em outros casos, é simplesmente ignorada. Durante o *check-in* inicial, se `-t` não é fornecida, `ci` obtém o texto descritivo através da entrada padrão (em geral o teclado), terminando em fim-de-arquivo ou uma linha contendo apenas um ponto-final ('.'). O texto é solicitado ao usuário se a interação é possível (veja a opção `-I`).

`-wlogin` : Utiliza *login* como valor para o campo do autor da revisão depositada. É útil para "mentir" sobre o autor, e para a opção `-k` se nenhum autor está disponível.

Ex.:

```
% ci -wjoao exemplo.c
```

`-vn` : Emula a versão *n* de RCS, onde *n* pode ser 3, 4 ou 5.

Isto pode ser útil quando se mistura arquivos RCS atuais com outros que estão rodando em versões mais antigas.

`-xsuffixos` : Especifica os possíveis sufixos para arquivos RCS, separados por '/'. Os nomes dos arquivos RCS procurados são obtidos acrescentando-se cada sufixo ao nome do arquivo de trabalho. Os sufixos são testados na ordem em que aparecem na lista sufixos, até que se encontre o arquivo procurado.

Por exemplo, o comando:

```
% ci -x,v/ teste.c
```

especifica dois sufixos possíveis para o arquivo RCS correspondente ao arquivo de trabalho teste.c: `,v` e o sufixo vazio. Assim, primeiramente o comando `ci` irá procurar por um arquivo RCS chamado teste.c,v e, caso este não seja encontrado, testa a segunda possibilidade, que é um arquivo RCS com o mesmo nome do arquivo de trabalho.

4.2. `co`

`co [opções] nome_arquivo`

O comando `co` recupera (*check-out*) uma cópia de uma revisão de cada arquivo RCS indicado por 'nome_arquivo' e a armazena no correspondente arquivo de trabalho. Se já existe um arquivo de trabalho com permissão de escrita, o usuário é questionado se o arquivo deve ser sobreposto ou não.

`co` sempre expande os identificadores contidos nos arquivos, conforme foi explicado anteriormente.

Se o comando `co` é aplicado a um arquivo RCS que não contém revisões, ele cria um arquivo de trabalho vazio.

As opções do comando `co` permitem:

- especificar o número da revisão a ser recuperada, podendo também bloqueá-la ou desbloqueá-la (opções `-r`, `-l` ou `-u`);
- obrigar a sobreposição (*overwrite*) do arquivo de trabalho, se este já existir (`-f`);
- modificar a forma de expansão de *keywords* (`-kkv`, `-kkv1`, `-kk`, `-ko` ou `-kv`);
- imprimir a revisão recuperada na saída padrão (`-p`);
- selecionar a revisão desejada através da data de *check-in*, estado ou autor (`-d`, `-s` ou `-w`);
- impor a hora de modificação do arquivo de trabalho gerado (`-M`);
- gerar uma nova revisão, através da junção de revisões especificadas (`-j`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`);
- permitir ou não permitir interação (`-I` ou `-q`).

Seguem as explicações detalhadas de cada opção:

`-r[rev]` : Recupera a última revisão cujo número é menor ou igual a rev.

Se rev indica um ramo ao invés de uma revisão, a última revisão naquele ramo é recuperada. Se rev é omitido, a última revisão no ramo *default* (veja a opção `-b` de *rcs*) é recuperada. Se rev é \$, o comando `co` determina o número de revisão a partir dos valores de *keyword* no arquivo de trabalho. De outro modo, uma revisão é composta de um ou mais campos numéricos ou simbólicos separados por períodos. O

equivalente numérico do campo simbólico é especificado com a opção `-n` dos comandos `ci` e `rco`.

A opção `-r$` só funciona se há um arquivo de trabalho no diretório atual contendo um identificador com o número da revisão. Nesse caso, essa revisão será recuperada (*checked-out*) novamente.

Ex.:

```
% co -r1.3 exemplo.c
```

irá recuperar a revisão 1.3 do arquivo RCS exemplo.c,v, armazenando-a no arquivo de trabalho exemplo.c.

```
% co -r1.1.2 exemplo.c
```

irá recuperar a revisão mais recente do segundo ramo da revisão 1.1.

```
% co -r2 exemplo.c
```

irá recuperar a revisão mais recente da versão 2 de exemplo.c.

`-l [rev]` : O mesmo que `-r`, exceto que também bloqueia (*lock*) a revisão recuperada para o usuário.

`-u [rev]` : O mesmo que `-r`, exceto que desbloqueia (*unlock*) a revisão recuperada se ela foi bloqueada pelo próprio usuário.

Se `rev` é omitido, `-u` recupera a revisão bloqueada pelo usuário, se houver alguma; caso contrário, recupera a última revisão no ramo *default*.

`-f [rev]` : Obriga a sobreposição (*overwrite*) do arquivo de trabalho; é útil quando utilizado em conjunto com `-q`. Se esta opção não for utilizada e já houver no diretório um arquivo de trabalho com permissão de escrita, `co` pergunta ao usuário se deve sobrepor o arquivo.

`-kkv` : Gera *strings* de *keywords* utilizando o formato *default*, por exemplo, `$Revision: 5.7 $` para a *keyword* *Revison*. O *login name* do bloqueador (*locker*) é inserido ao valor das *strings* das *keywords* *Header*, *Id* e *Locker*, apenas quando o arquivo está sendo bloqueado no momento, através dos comandos `ci -l` ou `co -l`.

`-kkvl` : O mesmo que `-kkv`, exceto que o *login name* do bloqueador é sempre inserido se a revisão dada está bloqueada, ou está sendo bloqueada no momento.

-kk : Gera apenas os nomes de *keywords* em *strings* de *keywords*, omitindo seus valores (isto é, não expande os identificadores).

Por exemplo, para a *keyword* *Author*, gera a *string* *\$Author\$*, ao invés de *\$Author: fatima \$*. Esta opção é útil para se ignorar as diferenças devido à substituição de *keyword*, quando queremos comparar diferentes revisões de um arquivo.

-ko : Ao invés de expandir os identificadores, substitui as *keywords* pelos valores que tinham antes da revisão ser armazenada. (Isto é, substitui pelos valores que foram expandidos ao dar-se *co* na revisão anterior, antes de modificá-la.)

-kv : Gera apenas os valores *keywords* para *strings* de *keyword*, apagando os identificadores. Por exemplo, para a *keyword* *Author*, gera a *string* *fatima*, ao invés de *\$Author: fatima \$*. Esta opção não pode ser usada em conjunto com *-l*, porque este arquivo não deverá ser modificado (editado), já que perdeu as *keywords*.

-p[rev] : Imprime a revisão recuperada na saída padrão ao invés de armazená-la em um arquivo de trabalho.

Esta opção é útil quando *co* faz parte de um *pipe*.

-q[rev] : Faz com que o RCS funcione sem imprimir os resultados.

-I[rev] : Faz com que o RCS rode interativamente, mesmo que a entrada padrão não seja um terminal.

-ddata : Recupera a última revisão no ramo selecionado cuja data/hora de *check-in* é menor ou igual a data.

A data e hora podem ser dados em formato livre (explicado em *ci*), entre aspas.

Ex.:

```
% co -d"1995/02/28 14:10:00" exemplo.c
```

irá recuperar a revisão que foi armazenada no dia 28/02/95 às 14:10hs, se tal revisão existir.

-M[rev] : Assume como a data de modificação do novo arquivo de trabalho, a data em que a revisão recuperada foi originalmente armazenada (*checked-in*).

Ex.:

Se a revisão 1.3 do arquivo exemplo.c foi armazenada em 01/01/95 às 12:00hs, e em 20/03/95 às 15:07hs for utilizado o comando

```
% co -M1.3 exemplo.c
```

a revisão recuperada será armazenada no arquivo de trabalho exemplo.c e será registrado como a data de criação (ou modificação) de exemplo.c não a data atual, "1995/03/20 15:07:00", mas a data em que a revisão 1.3 foi armazenada, "1995/01/01 12:00:00".

-sestado : Recupera a revisão mais recente no ramo selecionado cujo estado é estado.

Ex.:

```
% co -sExp exemplo.c
```

-w[login] : Recupera a última revisão no ramo selecionado que foi *checked-in* pelo usuário cujo *login name* é login.

Se o argumento login é omitido, o *login name* do usuário é assumido.

-jjoinlist : Gera uma nova revisão, que é a junção das revisões dadas na joinlist.

Esta opção é extremamente obsoleta se comparada ao comando `rcsmerge`, mas é mantida para compatibilidade com versões anteriores.

A joinlist é uma lista de pares de números de revisão (numéricos ou simbólicos) da forma rev2:rev3. Esses pares devem ser separados por vírgula. Para o par inicial, rev1 indica a revisão selecionada por uma das opções acima (`-f`, `...`, `-w`). Para todos os outros pares, rev1 é a revisão gerada pelo par anterior. (Assim, o resultado de uma junção se torna a entrada da próxima.)

Para cada par, `co` une as revisões rev1 e rev3 em relação a rev2. Isto significa que todas as mudanças que transformam em rev1 são aplicadas a uma cópia de rev3. Isto é particularmente útil se rev1 e rev3 são os fins de dois ramos que têm rev2 como ancestral comum. Se rev1<rev2<rev3 e pertencem ao mesmo ramo, a união gera uma nova revisão que é como rev3, mas com todas as mudanças que levam de rev1 a rev2 desfeitas. Se mudanças de rev2 a rev1 se sobrepõem a mudanças de rev2 a rev3 (isto é, são feitas alterações sobre as mesmas linhas de código), `co` relata esses conflitos (*overlaps*) da mesma maneira que o `rcsmerge`, abortando a operação.

Para o par inicial, rev2 pode ser omitido. O *default* para rev2 é o ancestral comum. Se quaisquer dos argumentos indicam ramos, as últimas revisões daqueles ramos são assumidas. As opções `-l` e `-u` respectivamente bloqueiam ou desbloqueiam rev1.

Ex.:

```
% co -u1.3 -d1.2:1.2.1 exemplo.c
```

irá unir as revisões 1.3 e a última revisão do primeiro ramo da revisão 1.2, tomando como base a revisão 1.2. Além disso, desbloqueia a revisão 1.3, se estiver bloqueada. O resultado é armazenado no arquivo de trabalho.

-v_n : Emula a versão n de RCS. Veja *ci* para maiores detalhes.

-xsufixos : Usa sufixos para caracterizar arquivos RCS. Os nomes dos arquivos RCS, serão determinados através do acréscimo de cada um dos sufixos contidos na lista sufixos, separados por '/', ao nome do arquivo de trabalho. *co* irá testar cada um dos sufixos, seguindo a ordem dada, até encontrar o respectivo arquivo.

4.3. ident

ident [-q] nome_arquivo

O comando *ident* identifica arquivos RCS.

Procura todas as ocorrências de identificadores do tipo *\$keyword: . . . \$* nos arquivos indicados ou, se não é dado nenhum nome de arquivo, na entrada padrão, e imprime todos na tela.

Esses identificadores são normalmente expandidos automaticamente pelo comando *co*, mas também podem ser inseridos manualmente.

Se não houver identificadores expandidos, o comando *ident* imprime uma mensagem de advertência. O mesmo ocorre se não houver nenhum arquivo de trabalho no diretório. Se a opção *-q* estiver presente, não imprime essas mensagens.

O comando *ident* pode trabalhar em arquivos objeto tão bem quanto em arquivos texto. Para isto, basta colocar os identificadores em *strings* de caracteres. Por exemplo, em C, poderíamos fazer isso através do seguinte código:

```
char rcsid[] = "$Id$";
```

Ao compilarmos o programa *prog.c*, que contém o código acima, em *prog.o*, e utilizarmos o comando *ident* para os dois arquivos, veremos que as saídas serão iguais.

4.4. rcs

r_{cs} [opções] nome_arquivo

O comando `rcs` cria novos arquivos RCS ou muda atributos daqueles já existentes. Um arquivo RCS contém múltiplas revisões de texto, uma lista de acesso, uma mensagem de mudança (*log*), texto descritivo e alguns atributos de controle. Para utilizar este comando em um arquivo RCS, é preciso que o *login name* do usuário esteja na lista de acesso deste arquivo, a não ser que a lista esteja vazia, o usuário seja o proprietário do arquivo (o arquivo está armazenado na sua área) ou o superusuário, ou a opção `-i` esteja presente (já que esta opção cria um novo arquivo RCS).

As opções do comando `rcs` permitem:

- criar e inicializar um novo arquivo RCS vazio (`-i`);
- acrescentar ou excluir *login names* da lista de acesso (`-a`, `-A` ou `-e`);
- especificar o ramo *default* (`-b`);
- especificar o *comment leader* - o símbolo que é colocado no início de cada linha da mensagem de *log* inserida no texto pelo identificador `$Log$` (`-c`);
- especificar o tipo de substituição de *keyword* a ser utilizado (`-k`);
- bloquear ou desbloquear uma revisão especificada (`-l` ou `-u`);
- modificar o *locking* para *strict* ou não-*strict* (`-L` ou `-U`);
- substituir a mensagem de *log* de uma revisão especificada (`-m`);
- associar ou desassociar um nome simbólico a um ramo ou revisão dada (`-n` ou `-N`);
- apagar revisões especificadas (`-o`);
- modificar o atributo estado de uma revisão dada (`-s`);
- substituir o texto descritivo do arquivo RCS (`-t`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`);
- permitir ou não permitir interação (`-I` ou `-q`).

As explicações detalhadas de cada opção são fornecidas a seguir:

-i : Cria e inicializa um novo arquivo RCS, sem armazenar qualquer revisão.

Equivale a um comando `ci` inicial. Se o nome do arquivo a ser criado não estiver com o caminho como prefixo, ele tenta colocá-lo primeiro no subdiretório `./RCS`, e depois no diretório atual. Se já existir um arquivo RCS com o mesmo nome, imprime uma mensagem de erro.

-alogins : Anexa à lista de acesso do arquivo RCS a lista de *login names* logins.

Logins é formada por vários *login names*, separados por vírgula.

Ex.: O comando

```
rsc -alairce,elaine exemplo.c
```

irá acrescentar os *login names* `lairce` e `elaine` à lista de acesso do arquivo RCS `teste.c`, mantendo aqueles que já existiam.

-Aarquivo : Anexa à lista de acesso do arquivo RCS a lista de acesso de arquivo (outro arquivo RCS).

Ex.:

```
% rsc -Ateste.c exemplo.c
```

irá acrescentar os *login names* contidos na lista de acesso do arquivo `teste.c` à lista de acesso do arquivo `exemplo.c`, mantendo aqueles que já existiam.

-elogins : Exclui da lista de acesso do arquivo RCS os *login names* que aparecem na lista logins.

Esta lista tem o mesmo formato da lista de *login names* utilizada com a opção `-a`. Se algum *login name* da lista logins não estiver na lista de acesso, o `rsc` simplesmente ignora-o.

-b[revisão] : Estabelece revisão como o ramo *default*.

Se revisão é omitida, o ramo mais alto do tronco (aquele cujo número de revisão é maior) é estabelecido dinamicamente como o ramo *default*.

-cstring : Estabelece string como o *comment leader*.

O *comment leader* é colocado no início de todas as linhas de mensagens de *log* inseridas pelo identificador `$Log$` durante o *check-out*, para que essas linhas sejam consideradas como comentário. É útil para linguagens de programação cujos

comentários não são de múltiplas-linhas. Ao se criar um novo arquivo RCS, através do comando `ci` ou um `rcs -i` sem a opção `-c`, o *comment leader* é determinado a partir do sufixo do arquivo de trabalho: para arquivos com extensão `.c`, o *comment leader default* é `*`, enquanto que para os arquivos com qualquer outra extensão, o *default* é `#`. *String* não precisa vir entre aspas. Se for utilizado a opção `-c` sem nenhuma *string*, assume como branco (*comment leader* = `""`).

Ex.: O exemplo abaixo estabelece o *comment leader* para o arquivo `sapes.prg`, que é um programa implementado em *clipper*, cujas linhas de comentários são iniciadas pela *string* `//`.

```
rcs -c// sapes.prg
```

`-ksubst` : Usa o tipo *subst* de substituição de *keyword* (pode ser `-kkv`, `-kkvl`, `-kk`, `-ko` ou `-kv`). Esses tipos de substituição de *keyword* são os mesmos vistos em `co`.

`-l[rev]` : Bloqueia (*lock*) a revisão cujo número é *rev*, a menos que ela já esteja bloqueada por outra pessoa.

Se *rev* indica um ramo, a última revisão do ramo é bloqueada. Se *rev* é omitida, bloqueia a última revisão do ramo *default* (normalmente o tronco). Pode-se desbloquear uma revisão com `ci` ou `rcs -u`. O bloqueio (*lock*) de revisões evita que modificações conflitantes sejam sobrepostas.

Ex.:

```
rcs -l1.1 teste.c
```

O comando acima apenas bloqueia a revisão número 1.1 (isto é, a primeira revisão inserida) do arquivo `teste.c,v`.

`-u[rev]` : Desbloqueia (remove o *lock*) a revisão cujo número é *rev*.

Se *rev* é um ramo, desbloqueia a última revisão bloqueada pelo usuário. Normalmente, apenas a pessoa que bloqueou a revisão pode desbloqueá-la. No entanto, outra pessoa pode desbloquear essa revisão, quebrando o *lock*. Isto causa uma mensagem de *mail* para ser enviada ao bloqueador original. A mensagem irá conter um comentário que será solicitado ao usuário que quebrou o *lock*. Esse comentário é terminado por fim-de-arquivo ou por uma linha contendo apenas um ponto-final (`'.'`).

`-L` : Estabelece o *locking* como *strict*.

Strict locking significa que o proprietário do arquivo RCS também precisa bloquear o arquivo para poder *check-in*. O *strict locking* deve ser usado para arquivos que são compartilhados entre vários usuários.

-U : Estabelece o *locking* como não-strict.

Locking não-strict significa que o proprietário do arquivo não precisa bloquear (*lock*) a revisão para poder *check-in*. Este tipo de *locking* não deve ser usado para arquivos que são compartilhados entre vários usuários. O *locking default* é normalmente strict.

-mrev:msg : Substitui a mensagem de *log* da revisão de número rev por msg.

Msg deve vir entre aspas.

Ex.:

```
rcs -m2.1:"Este é um teste." testel.c
```

-nnome[:rev] : Associa o nome simbólico nome ao ramo ou revisão de número rev.

O nome simbólico é uma *string* que pode ser utilizada para identificar uma revisão, substituindo o número de revisão. Se ':' e 'rev' são omitidos, deleta o nome simbólico nome. Imprime uma mensagem de erro se nome já está associado a outra revisão. Se a revisão já tem um nome simbólico associado, substitui pelo novo nome.

Ex.:

```
% rcs -nteste:1.3 exemplo.c
```

atribui o nome simbólico "teste" à revisão 1.3 de exemplo.c. Quando essa revisão precisar ser recuperada mais tarde, pode-se identificá-la pelo nome "teste" ao invés do número 1.3. Por exemplo:

```
% co -rteste exemplo.c
```

-Nnome[:rev] : Tem a mesma função de -n, só que não dá erro se o nome já estiver associado a outra revisão. Neste caso, simplesmente associa o nome simbólico nome à revisão de número rev, e a revisão que antes estava associada a nome passa a não ter nenhum nome simbólico.

-ointervalo : Apaga as revisões dadas por intervalo.

Se intervalo consistir apenas de um único número de revisão, essa revisão será deletada. Se intervalo for o número de um ramo, a última revisão desse ramo será deletada. Um intervalo da forma rev1:rev2 indica todas as revisões de rev1 a rev2 no mesmo ramo; :rev indica todas as revisões desde o início do ramo que contém rev, até e incluindo rev; e rev indica todas as revisões a partir de rev até o fim do ramo que contém rev. Para poder ser deletada, nenhuma das revisões pode conter ramos ou estar bloqueada.

Ex.: Para apagar todas as revisões contidas no ramo 1.2.1 do arquivo exemplo.c, que vão desde 1.2.1.1 até 1.2.1.4, pode-se utilizar qualquer um dos comandos abaixo:

```
% rcs -o1.2.1.1:1.2.1.4 exemplo.c
% rcs -o:1.2.1.4 exemplo.c
% rcs -o1.2.1.1: exemplo.c
```

-q : Faz com que o RCS funcione sem imprimir os resultados, isto é, não-interativamente.

-I : Faz com que o RCS rode interativamente, mesmo que a entrada padrão não seja um terminal.

-sestado[:rev] : Estabelece o atributo de estado da revisão de número rev como sendo estado.

Se rev indica um ramo, assume a última revisão desse ramo. Se rev é omitido, assume a última revisão do ramo *default*. Qualquer identificador é aceitável para estado. Um conjunto útil de estados é: Exp (para experimental), Stab (para estável) e Rel (para *released* - liberado). Por *default*, o comando *ci* atribui Exp para o estado de uma revisão.

Ex.:

```
% rcs -sRel:1.3 exemplo.c
```

-t[arquivo] : Substitui o texto descritivo do arquivo RCS pelo texto contido em arquivo. O caminho (*pathname*) do arquivo não pode iniciar com '-'. Se arquivo é omitido, obtém o texto da entrada padrão, terminando com um fim-de-arquivo ou uma linha contendo apenas um ponto-final ('.').

-t-string : Substitui o texto descritivo do arquivo RCS pelo texto de string.

-vn : Emula a versão n de RCS.

-xsufixos : Usa sufixos para caracterizar arquivos RCS. O sufixo *default* é ,v.

4.5. rcsclean

`rcsclean [opções] nome_arquivo`

O comando `rcsclean` limpa o diretório de trabalho, removendo arquivos que foram *checked-out*, mas não foram modificados.

Para cada `nome_arquivo` dado, o `rcsclean` compara o arquivo de trabalho com uma revisão do correspondente arquivo RCS. Se alguma diferença é encontrada, nada é feito. Caso contrário, primeiro desbloqueia a revisão, se a opção `-u` é fornecida, e depois remove o arquivo de trabalho, a menos que este esteja disponível para escrita e a revisão esteja bloqueada (isto é, a opção `-u` não foi fornecida). Registra na tela todas as ações tomadas.

O número da revisão com a qual o arquivo de trabalho será comparado pode ser especificado através das opções `-n`, `-q`, `-r` ou `-u`. Se nenhuma revisão é especificada, então se `-u` é fornecido e o usuário tem uma revisão bloqueada, `rcsclean` usa essa revisão; caso contrário, usa a última revisão do ramo *default* (normalmente a raiz).

Se nenhum `nome_arquivo` é dado, irá limpar todos os arquivos que estão no diretório atual.

As opções do comando `rcsclean` permitem:

- especificar o tipo de substituição de *keyword* a ser utilizado (`-k`);
- imprimir na tela o resultado do comando, sem no entanto executá-lo realmente (`-n`);
- evitar a impressão na tela das ações que foram executadas (`-q`);
- especificar a revisão a ser comparada (`-r`);
- desbloquear a revisão especificada (`-u`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`).

A seguir, cada opção é explicada com mais detalhes:

`-ksubst` : Usa o tipo *subst* de substituição de *keyword*, ao recuperar a revisão para comparação. Esses tipos de substituição de *keyword* são os mesmos vistos em `co`.

`-n[rev]` : Apenas imprime na tela o que o comando iria fazer, sem no entanto remover qualquer arquivo ou desbloquear qualquer revisão. Isto é, ao invés de realmente executar a ação, simplesmente conta qual ação seria feita.

- q[rev] : Não imprime na saída padrão as ações tomadas.
- r[rev] : Especifica ao comando qual revisão deve ser utilizada para a comparação.
- u[rev] : Irá desbloquear a revisão rev se esta estiver bloqueada e não for encontrada nenhuma diferença.
- vn : Emula a versão n de RCS.
- xsufixos : Usa sufixos para caracterizar arquivos RCS. O sufixo *default* é ,v.

4.6. rcsdiff

rcsdiff [opções] nome_arquivo

O comando `rcsdiff` compara duas revisões de arquivos RCS.

O `rcsdiff` executa o comando *diff*, do UNIX, para comparar duas revisões de cada arquivo RCS dado.

Zero, uma ou duas revisões podem ser especificadas com a opção `-r`. Se as duas revisões são omitidas, o `rcsdiff` compara a última revisão do ramo *default* (normalmente o tronco) com o conteúdo do correspondente arquivo de trabalho. Isto é útil para determinar o que foi mudado desde o último *check-in*. Se uma revisão é dada e a outra omitida, o `rcsdiff` compara a revisão dada com o conteúdo do correspondente arquivo de trabalho. Se ambas as revisões são dadas, o `rcsdiff` compara essas duas revisões do arquivo RCS.

As opções do comando `rcsdiff` permitem:

- especificar o tipo de substituição de *keyword* a ser utilizado (`-k`);
- evitar que os resultados sejam impressos na tela (`-q`);
- especificar os números das revisões a serem comparadas (`-r`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`).

Cada opção é explicada detalhadamente a seguir:

- ksubst : Usa o tipo subst de substituição de *keyword*. Esses tipos de substituição de *keyword* são os mesmos vistos em *co*.
- q : Modo não-interativo; não imprime os resultados na tela.
- rrev : Determina qual o número da revisão que vai ser comparada
- vn : Emula a versão n de RCS.
- xsufixos : Usa sufixos para caracterizar arquivos RCS. O sufixo *default* é *,v*.

4.7. rcsfreeze

`r``c``s``f``r``e``e``z``e` [nome]

O comando `r``c``s``f``r``e``e``z``e` congela uma configuração de arquivos sob o controle do RCS.

O que ele realmente faz é associar um número de revisão simbólico a um conjunto de arquivos que formam uma configuração válida.

A idéia é executar `r``c``s``f``r``e``e``z``e` cada vez que uma nova versão é armazenada (*checked-in*). Um identificador único de nome simbólico *C\_número*, onde *número* é incrementado toda vez que `r``c``s``f``r``e``e``z``e` é executado, é então associado à revisão mais recente do tronco principal de cada arquivo RCS.

Um argumento nome opcional determina um nome simbólico à configuração. O identificador único continua sendo gerado, mas não irá aparecer como parte do nome de revisão simbólico nos arquivos RCS reais.

Ao executar esse comando, uma mensagem de *log* será requisitada ao usuário para futura referência.

O `r``c``s``f``r``e``e``z``e` trabalha em todos os arquivos do diretório ao mesmo tempo.

Há dois arquivos auxiliares utilizados pelo `r``c``s``f``r``e``e``z``e` para guardar o *status* das configurações. Esses dois arquivos são arquivos ocultos (arquivos cujos nomes são precedidos por um ponto), e para vê-los, basta utilizar o comando `ls -a`.

Os dois arquivos são:

.rcsfreeze.ver - guarda o número da última versão congelada.

.rcsfreeze.log - armazena todas as mensagens de *log* em ordem a partir da mais recente.

4.8. rcsmerge

`rcsmerge [opções] nome_arquivo`

O comando `rcsmerge` une duas revisões de arquivos RCS.

Ele incorpora as mudanças entre duas revisões de um arquivo RCS no correspondente arquivo de trabalho.

Pelo menos uma revisão deve ser especificada com uma das opções descritas abaixo, geralmente `-r`. Podem ser especificadas no máximo duas revisões.

As opções do comando `rcsmerge` permite:

- especificar o tipo de substituição de *keyword* a ser utilizado (`-k`);
- enviar o resultado da operação para o saída padrão (`-p`);
- evitar que os resultados sejam impressos na tela (`-q`);
- especificar a revisão a ser tomada como base para a união (`-r`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`).

Todas as opções são descritas com mais detalhes a seguir:

`-ksubst` : Usa o tipo subst de substituição de *keyword*. Esses tipos de substituição de *keyword* são os mesmos vistos em `co`.

`-p[rev]` : Envia o resultado para a saída padrão ao invés sobrepor o arquivo de trabalho.

`-q[rev]` : Modo não-interativo; não imprime os resultados na tela.

-r[rev] : Une em relação à revisão rev. Se rev é omitido, é considerada a última revisão do ramo *default*, normalmente o *head*.

-vn : Emula a versão n de RCS.

-xsufixos : Utiliza sufixos para caracterizar arquivos RCS.

4.9. rlog

`rlog [opções] nome_arquivo`

O comando `rlog` imprime informações sobre arquivos RCS.

Pode imprimir as seguintes informações para cada arquivo RCS: nome do arquivo RCS, nome do arquivo de trabalho, *head* (isto é, o número da última revisão do tronco), ramo *default*, lista de acesso, *locks*, nomes simbólicos, sufixo, *comment leader*, número total de revisões, número de revisões selecionadas para impressão, e texto descritivo. A seguir, vêm os registros das revisões selecionadas, em ordem cronológica inversa, para cada ramo. Para cada revisão, o comando `rlog` imprime número de revisão, autor, data/hora de armazenamento (*check-in*), estado, número de linhas adicionadas/deletadas (em relação à revisão anterior), bloqueador (*locker*) da revisão (se houver), e mensagem de *log*. Se nenhuma opção é fornecida, `rlog` imprime todas as informações. As opções restringem essa saída.

As opções do comando `rlog` permitem:

- ignorar arquivos RCS que não têm revisões bloqueadas (`-L`);
- imprimir apenas as informações desejadas (`-R`, `-h` ou `-t`);
- selecionar as revisões cujas informações são desejadas (`-d`, `-l`, `-r`, `-s` ou `-w`);
- especificar os possíveis sufixos utilizados para o arquivo RCS (`-x`);
- determinar a versão do RCS a ser emulada (`-v`).

Abaixo são descritas detalhadamente cada uma das opções:

`-L` : Ignora os arquivos RCS que não têm nenhuma revisão bloqueada (*locked*). É útil para se usar em combinação com as opções `-h`, `-l` e `-R`.

- R : Imprime só o nome do arquivo RCS. É útil quando se quer obter o nome de um arquivo RCS a partir do arquivo de trabalho.
- h : Imprime apenas o nome do arquivo RCS, o nome do arquivo de trabalho, o *head*, o ramo *default*, a lista de acesso, *locks*, nomes simbólicos, sufixo, *comment leader* e o número total de revisões.
- t : Imprime o mesmo que -h, mais o texto descritivo.
- b : Imprime informações sobre as revisões no ramo *default*, normalmente o ramo mais alto no tronco.
- ddatas : Imprime informações sobre as revisões com data/hora de *check-in* nos intervalos dados pela lista datas, separados por ponto-e-vírgula.

Um intervalo da forma d1<d2 ou d2>d1 seleciona as revisões que foram depositadas entre d1 e d2, inclusive. Um intervalo da forma <d ou d> seleciona todas as revisões cuja data de *check-in* é d ou anterior. Um intervalo da forma d< ou >d seleciona todas as revisões cuja data é d ou posterior. Um intervalo da forma d seleciona a última revisão cuja data é d ou anterior. As datas/horas devem estar em formato livre, como foi explicado anteriormente em *co*. As aspas são normalmente necessárias, principalmente quando se utiliza > e <. Observe que o separador é um ponto-e-vírgula.

Ex.:

```
% rlog -d"1995/07/05;1995/07/25" exemplo.c
```

imprime as informações de todas as revisões que foram armazenadas entre os dias 5 e 25 de julho de 1995.

- l[lockers] : Imprime informações apenas sobre as revisões que estão bloqueadas. Além disso, se é dada a lista lockers de *login names* separados por vírgula, ignora todas as revisões bloqueadas por outros cujos *login names* não estão na lista lockers.

Ex.:

```
% rlog -lfatima,lairce exemplo.c
```

- r[revisões] : Imprime informações sobre as revisões dadas pela lista revisões de revisões e intervalos separados por vírgula.

Um intervalo da forma rev1:rev2 indica as revisões de rev1 a rev2 no mesmo ramo; :rev indica as revisões do início do ramo até e inclusive rev; e rev: indica as revisões a partir de rev até o fim do ramo que contém rev. Se o argumento é um ramo, indica

todas as revisões daquele ramo. Um ramo seguido por um ponto ('.') indica a última revisão daquele ramo. Uma opção `-r` sem revisões indica a última revisão do ramo *default*, normalmente o tronco.

Ex.:

```
% rlog -l1.2:1.4 exemplo.c
```

`-sestados` : Imprime informações sobre as revisões cujos atributos de estado são um dos estados, separados por vírgula, dados pela lista estados.

Ex.:

```
% rlog -sStab,Rel exemplo.c
```

`-w[logins]` : Imprime informações sobre as revisões que foram *checked-in* por usuários cujos *login names* estão na lista logins, separados por vírgula.

Se a lista logins é omitida, é assumido o *login name* do usuário.

`-vn` : Emula a versão n de RCS.

`-xsufixo` : Usa sufixo para caracterizar arquivos RCS.

Obs.: É importante observar que o comando `rlog` imprime a intersecção das revisões selecionadas através das opções `-d`, `-l`, `-s` e `-w`, interseccionada com a união das revisões selecionadas através de `-b` e `-r`.

5. Referências Bibliográficas

- [Boa94] BOAS, A. V., *RCS - Revision Control System*. Departamento de Sistemas de Operação - Projeto ASSISTE, CPqD - Telebras, 1994
- [Bro91] BROWN, H., *Like a Version*. Computer Language, agosto de 1991
- [Buc94] BUCKLEY, F. J., *Implementing a Software Configuration Management Environment*. IEEE Computer, fevereiro de 1994
- [Pre92] PRESSMAN, R. S., *Software Engineering - A Practitioner's Approach*. MacGraw-Hill, Inc., 3a. edição, 1992
- [San94] SANCHES, R., REINHARD, N., *A Influência da Configuração de Software na Manutenção*. Anais do VIII Simpósio Brasileiro de Engenharia de Software, 1994
- [Tic85] TICKY, W. F., *RCS - A System for Version Control*. Software - Practice and Experience 15, julho de 1985