

## U5-1 Überblick

- Besprechung 3. Aufgabe (mini\_sh)
- Fragen zur Aufgabe 4 (malloc) ???
- Erstellen von C-Funktionsbibliotheken
- RCS

## 2 Static Libraries

- Werkzeuge: ar und ranlib
  - ar: verwaltet Archive - vor allem für Objekt-Dateien genutzt
    - ▶ Erzeugen eines Archivs libutil.a aus mehreren .o-Dateien
 

```
ar rc libutil.a file1.o file2.o file3.o ...
```
  - ranlib: (oder ar -s) erzeugt ein Inhaltsverzeichnis für das Archiv
    - ▶ enthält alle Symbole (= globale Variablen und Funktionen) damit der Binder schneller die benötigten .o-Dateien im Archiv auffinden kann
 

```
ranlib libutil.a
```
- Angabe der Bibliothek beim Binden
 

```
gcc -static prog.c -L. -lutil -o prog
```

  - ▶ -L. : Bibliotheken werden auch im aktuellen Directory (.) gesucht (sonst nur Standard-Directories wie z. B. /lib oder /usr/lib)
  - ▶ -lutil: Bibliothek mit Namen libutil.a wird gesucht

## U5-2 Erstellen von C-Funktionsbibliotheken

### 1 Überblick

- statische Bibliotheken
  - ▶ Archiv, in dem mehrere Objekt-Dateien (.o) zusammengefasst werden
  - ▶ beim statischen Binden eines Programms werden die benötigten Objekt-Dateien zu der ausführenden Datei hinzukopiert
  - ▶ Bibliothek ist bei der Ausführung des Programms nicht mehr sichtbar
- dynamische, gemeinsam genutzte Bibliotheken (shared libraries)
  - ▶ Zusammenfassung von übersetzten C-Funktionen
  - ▶ beim Binden werden Referenzen auf die Funktionen offen gelassen
  - ▶ Shared Library ist nur einmal im Hauptspeicher vorhanden
  - ▶ Shared Library wird in virtuellen Adressraum dynamisch gebundener Programme beim Laden eingeblendet, noch offene Referenzen werden danach gebunden

### 3 Shared Libraries

- Kein Dateiarchiv sondern eine ladbare Funktionssammlung
  - ▶ Erzeugen mit cc
- Code der Funktionen liegt nur einmal im Hauptspeicher, kann aber in verschiedenen Anwendungen an unterschiedlichen Adressen im virtuellen Adressraum (siehe Vorlesung Kap. 7.1) positioniert sein
  - ▶ keine absoluten Adressen (Sprünge, Unterprogrammaufrufe) im Code erlaubt -> PIC (*position independent code*)
  - ▶ muss beim Compilieren der Quellen berücksichtigt werden
 

```
gcc -fPIC -c file1.c
gcc -fPIC -c file2.c
...
```
- Bibliothek wird durch Binden mehrerer .o-Dateien erzeugt
 

```
gcc -shared -o libutil.so file1.o file2.o ...
```

### 3 Shared Libraries (2)

- Beim Binden einer Anwendung werden Funktionen nicht aus Bibliothek kopiert

```
gcc prog.c -L. -lutil -o prog
```

- Aufruf analog zum statischen Binden (aber Option -static hat dort verhindert, dass dynamisch gebunden wird)
- Bibliothek `libutil.so` wird gesucht

- Endgültiges Binden erfolgt erst beim Laden

- Beim Laden von `prog` (exec) wird zunächst der *dynamic linker/loader* (`ld.so`) geladen
- `ld.so` lädt `prog` und die Bibliothek (wenn noch nicht im Hauptspeicher vorhanden) und bindet noch offene Referenzen
- Bibliothek wird von `ld.so` in mehreren Directories gesucht (über Environment-Variable `LD_LIBRARY_PATH` einstellbar)

### 2 Einführung (2)

- RCS besteht aus einer Reihe von Kommandos, die es dem Benutzer erlauben
  - ◆ Dateien unter RCS-Kontrolle zu stellen und Kopien aller Versionen zu bekommen, die danach erstellt wurden
  - ◆ eine Version zum Editieren zu entnehmen und diese gegen gleichzeitige Änderungen zu sperren
  - ◆ Neue Versionen (mit Kommentar) zu erzeugen
  - ◆ Unbrauchbare Änderungen rückgängig zu machen
  - ◆ Zustandsinformation von Dateien abzufragen
    - Zeilenweise Unterschiede zwischen verschiedenen Versionen auszugeben
    - Log-Informationen über Versionen: Urheber, Datum, usw.

## U5-3 Revision Control System – RCS

### 1 Einführung

- RCS ist ein Versionskontrollsystem, das
  - ◆ Änderungen an Dateien mit dem Namen des Ändernden, dem Zeitpunkt und einem Kommentar speichert
  - ◆ Zugriffe auf Versionen kontrolliert und koordiniert
  - ◆ eindeutige Identifizierung verwendeter Versionen erlaubt
  - ◆ redundante Speicherung von Versionen vermeidet
    - es wird jeweils die letzte Version einer Datei gespeichert
    - zusätzlich werden sog. *reverse deltas* (Beschreibungen, wie aus Version `n` Version `n-1` erzeugt wird) abgelegt

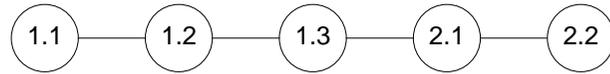
### 3 Terminologie

- Delta
  - ◆ Menge von zeilenweisen Änderungen an der Version einer Datei unter der Kontrolle von RCS (die Begriffe "Version" und "Delta" werden oft synonym gebraucht)
- Revision-Id
  - ◆ Jede Version erhält zur Identifikation eine Identifikation zugewiesen: `Release-Nummer.Level-Nummer`
- RCS-Datei
  - ◆ enthält die neueste Version und alle vorhergehenden Versionen in Form von Deltas zusammen mit Verwaltungsinformationen
  - ◆ der Dateiname endet auf `,v`, die RCS-Datei ist entweder im Unterdirectory `RCS`, oder im gleichen Directory wie die Arbeitsdatei abgelegt
- Arbeitsdatei
  - ◆ Kopie einer Version aus der RCS-Datei

## 4 Nummerierung von Versionen

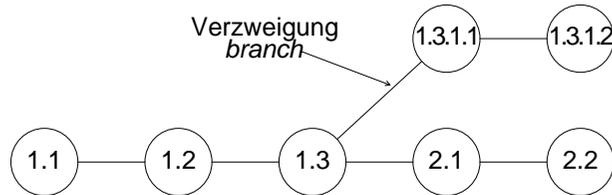
- Versionen werden ausgehend von der Ur-Version nummeriert:

release.level



- Versionen in einer Verzweigung erhalten

release.level.branch.branchlevel



## 5 Kommandos — *ci(1)*

- check in RCS-Revisions* — Erzeugen neuer Versionen
  - ◆ *ci(1)* übernimmt neue Versionen in RCS-Dateien
  - ◆ die neue Version wird aus der jeweiligen Arbeitsdatei entnommen, die Arbeitsdatei wird anschließend gelöscht
  - ◆ existierte zu der Arbeitsdatei noch keine RCS-Datei, wird eine neue RCS-Datei erzeugt
- Aufrufsyntax (nur die wichtigsten Optionen angegeben!):

```
ci [-rrev] [-lrev] [-urev] filename ...
```

- rrev** die neue Version erhält Version **rev**
  - **rev** muß größer als die letzte existierende Version sein
  - soll eine neue *Release* erzeugt werden, genügt die Angabe der *Release*-Nummer (z. B. **-r5**)
- lrev** wie **ci -r**, anschließend wird automatisch ein **co -l** durchgeführt
- urev** wie **ci -r**, anschließend erfolgt ein **co**

## 5 Kommandos — Überblick

- ci(1)*** ***check in***  
speichert die Arbeitsdatei als neue Version in der RCS-Datei ab falls noch nicht vorhanden, wird ein neue RCS-Datei erzeugt
- co(1)*** ***check out***  
extrahiert eine existierende Version aus der RCS-Datei (nur zum Lesen oder exklusiv zum Schreiben)
- racs(1)*** Modifikation von RCS-Datei-Attributen
- rlog(1)*** Ausgabe von *log*-Information und RCS-Datei-Attributen
- ident(1)*** extrahiert RCS-Identifikatoren aus einer Datei
- rscsclean(1)*** nicht-modifizierte Arbeitsdateien löschen
- rscsdiff(1)*** *diff* zwischen Versionen einer RCS-Datei
- rscsmerge(1)*** erzeugt aus zwei Versionen (insbes. bei Verzweigungen) eine neue Version

- bei allen Kommandos kann als **filename** immer sowohl der Arbeitsdateiname oder der RCS-Dateiname angegeben werden

## 5 Kommandos — *ci(1)*

- Beispiel ***ci, rlog***

```
% ci prog.c
RCS/prog.c,v <-- prog.c
initial revision: 1.1
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Program to demonstrate RCS
>> .
done
% rlog prog.c

RCS file: RCS/prog.c,v
Working file: prog.c
head: 1.1
branch:
locks: strict
access list:
symbolic names:
comment leader: " * "
keyword substitution: kv
total revisions: 1;      selected revisions: 1
description:
Program to demonstrate RCS
-----
revision 1.1
date: 1992/07/20 11:56:43; author: jklein; state: Exp;
Initial revision
=====
%
```

## 5 Kommandos — *co(1)*

- ◆ *check out RCS Revisions* — Versionen entnehmen
- **co(1)** entnimmt eine Version aus allen angegebenen RCS-Dateien
- die entnommene Version wird als Arbeitsdatei abgespeichert
- der Name der Arbeitsdatei ergibt sich aus dem Namen der RCS-Datei, wobei die Endung `,v` und ggf. der Pfad-Prefix `RCS/` weggelassen werden
- ◆ Aufrufsyntax (nur die wichtigsten Optionen angeben!):
 

```
co [-rrev] [-lrev] [-urev] filename ...
```

  - rrev** extrahiert die neueste Version, der Versionsnummer kleiner oder gleich **rev** ist
  - lrev** wie `co -r`, extrahiert die Version für den Aufrufer exklusiv zum Schreiben (für weitere `co`-Aufrufe gesperrt)
  - urev** wie `co -r`, falls eine Sperre der Version durch den Aufrufer existiert, wird diese aufgehoben

## 6 Identifikation von RCS-Versionen

- RCS ersetzt bei einem **check out** im Text alle Vorkommen der Zeichenkette
 

```
$Id$
```

 durch
 

```
$Id: filename revisionnumber date time author state locker$
```
- **co(1)** sorgt dafür, daß diese Zeichenkette automatisch auf aktuellem Stand gehalten wird
- um diese Zeichenkette in Objekt-Code zu implantieren, reicht es, sie in als *String* im Programm anzugeben — in C z. B.
 

```
static char rcsid[] = "$Id$";
```
- mit dem Kommando **ident(1)** können solche RCS-Identifikatoren aus beliebigen Dateien extrahiert werden
  - ↳ damit ist z. B. feststellbar, aus welchen Versionen der Quelldateien ein ausführbares Programm entstanden ist

## 5 Kommandos — *co(1)*

### ■ Beispiel *co, rlog*

```
% co -l prog.c
RCS/prog.c,v --> prog.c
revision 1.1 (locked)
done
% rlog prog.c

RCS file: RCS/prog.c,v
Working file: prog.c
head: 1.1
branch:
locks: strict
      jklein: 1.1
access list:
symbolic names:
comment leader: " * "
keyword substitution: kv
total revisions: 1;      selected revisions: 1
description:
Program to demonstrate RCS
-----
revision 1.1   locked by: jklein;
date: 1992/07/20 11:56:43; author: jklein; state: Exp;
Initial revision
=====
%
```