

Übungen zu Systemprogrammierung 1 (SP1)

Ü1 – Einführung

J. Schedel, C. Erhardt, A. Ruprecht, J. Kleinöder

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 2015 – 17. bis 23. April 2015

https://www4.cs.fau.de/Lehre/SS15/V_SP1

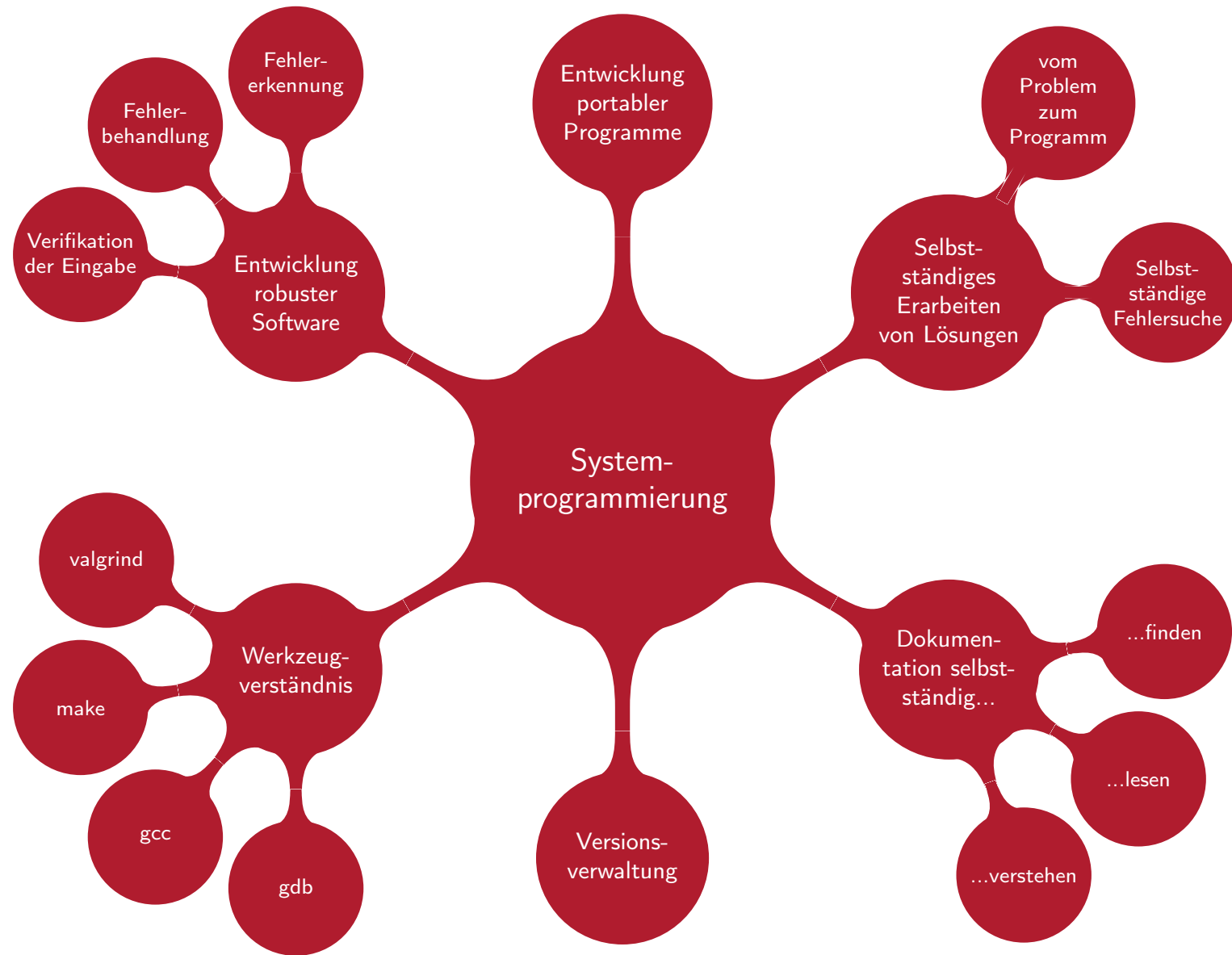


Agenda

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



Lernziele Systemprogrammierung



Tafelübungen und Besprechungen

- Vorstellung von Betriebssystemkonzepten und Werkzeugen
- Einführung in die Verwendung der Schnittstellen
- Erarbeiten eines kleinen Programmes (aktive Mitarbeit!)
- Besprechung der Abgaben und allgemeiner Fallstricke

Praktischer Teil – Aufgaben

- Arbeiten mit der Betriebssystemschnittstelle
- Fehlersuche und Fehlerbehebung
- Verwenden der vorgestellten Werkzeuge
- Hilfestellung in den Rechnerübungen



Agenda

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



- Ausgabe neuer Aufgaben in den Tafelübungen
 - Aufgabenstellung meist recht knapp
 - Nicht alles bis in letzte Detail spezifiziert
 - Gegebene Spezifikationen sind dennoch zwingend einzuhalten
- Selbstständiges Bearbeiten der Aufgaben (vorzugsweise im CIP)
 - bei Problemen hilft z. B. ein Besuch in den Rechnerübungen
- Korrektur und Bewertung erfolgt durch den jeweiligen Tafelübungsleiter
 - korrigierte Ausdrucke werden in den Besprechungen ausgegeben
 - teilweise auch elektronisch zur Verfügung gestellt
 - eigenes Ergebnis nach Login im *WAFFEL* einsehbar
- Übungspunkte können das Klausurergebnis verbessern (Bonuspunkte)
 - Abschreibtests
 - Vorstellen der eigenen Lösungen



Praktischer Teil – Bearbeitung der Aufgaben

- einzeln oder in Zweier-Teams je nach Aufgabe
 - bei Teamarbeit müssen beide Partner in der **gleichen** Tafelübung sein
- Bearbeitungszeitraum ist angegeben in Werktagen (bei uns: Montag bis Freitag)
 - Bearbeitungszeitraum beinhaltet den Tag der Tafelübung
 - Feiertage (01.05., 14.05., 25.05., 04.06.) und vorlesungsfreie Tage (26.05.) sind nicht enthalten
 - Abgabetermin kann per Skript erfragt werden
- plant für die Bearbeitung einer Aufgabe **mindestens** 8–16 Stunden (in Worten: ein bis zwei **Tage**) ein
 - langer Bearbeitungszeitraum bietet euch Flexibilität bei der Arbeitsverteilung
 - Rückmeldung über wirkliche Bearbeitungszeit erwünscht



- Forum: <https://fsi.cs.fau.de/forum/18>
 - inhaltliche Fragen zum Stoff oder den Aufgaben
 - allgemein alles, was auch für andere Teilnehmer interessant sein könnte
- Mailingliste: i4sp@cs.fau.de
 - geht an alle Übungsleiter
 - Angelegenheiten, die nur die eigene Person/Gruppe betreffen
- Rechnerübungen
 - Hilfe bei konkreten Problemen (z. B. Quellcode kompiliert nicht)
 - **kein** Händchenhalten, während ihr die Tastatur bedient :)
 - angebotene Termine siehe Homepage
- der eigene Übungsleiter
 - Fragen zur Korrektur
 - fälschlicherweise positiver Abschreibetest



Agenda

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse**
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



Voraussetzungen

- UNIX-Grundkenntnisse werden vorausgesetzt
 - Übungsleiter sind in den Rechnerübungen bei Bedarf behilflich
- Zur Auffrischung: UNIX-Einführung der FSI
<https://fsi.cs.fau.de/vorkurs>



Dokumentation aus 1. Hand: Manual-Pages

- Aufgeteilt in verschiedene *Sections*

- 1 Kommandos
 - 2 Systemaufrufe
 - 3 Bibliotheksfunktionen
 - 5 Dateiformate (Spezielle Datenstrukturen etc.)
 - 7 verschiedenes (z. B. Terminaltreiber, IP)

- Angabe normalerweise mit *Section*: `printf(3)`

- Aufruf unter Linux:

```
> # man [section] begriff  
> man 3 printf
```

- Suche nach *Sections*: `man -f begriff`

- Suche nach Manual-Pages zu einem Stichwort: `man -k stichwort`



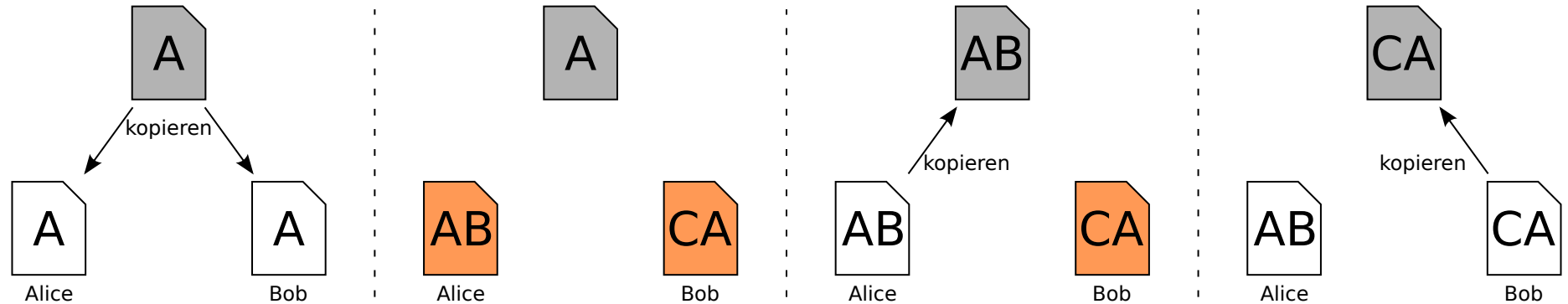
Agenda

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN**
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



Warum Versionsverwaltung?

- Gemeinsames Bearbeiten einer Datei kann zu Problemen führen:

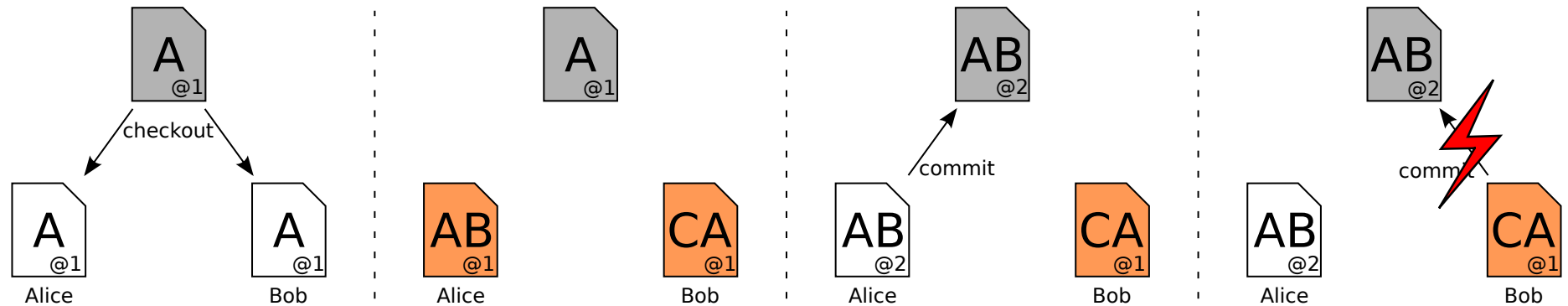


- Modifikationen werden nicht erkannt
- Änderungen von Alice gehen unbemerkt verloren

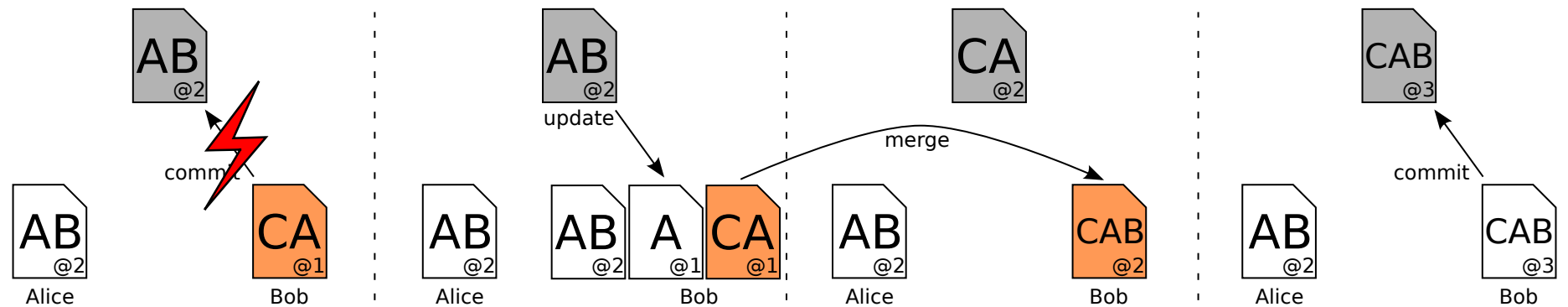


Warum Versionsverwaltung?

■ Versionsnummer zur Erkennung von Modifikationen



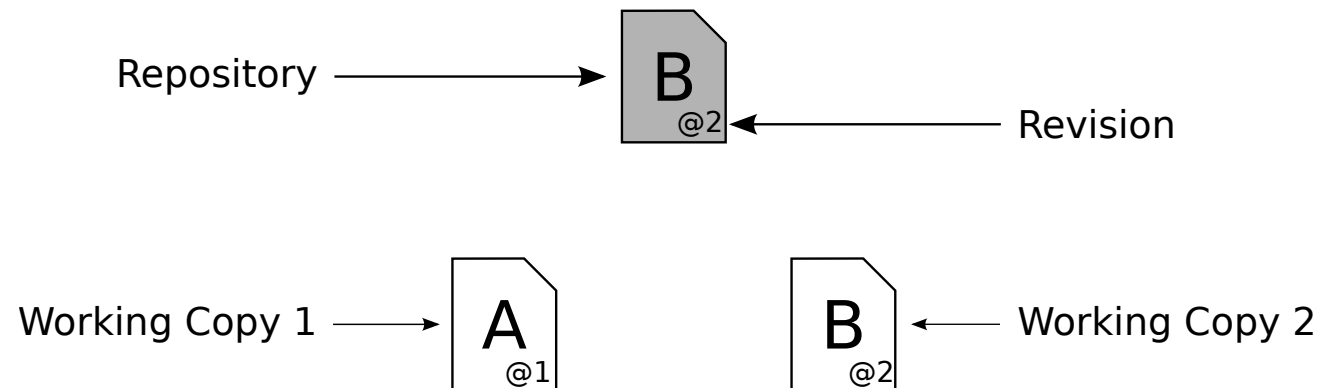
■ Entstandener Konflikt muss lokal gelöst werden



Das Versionsverwaltungssystem Subversion (SVN)

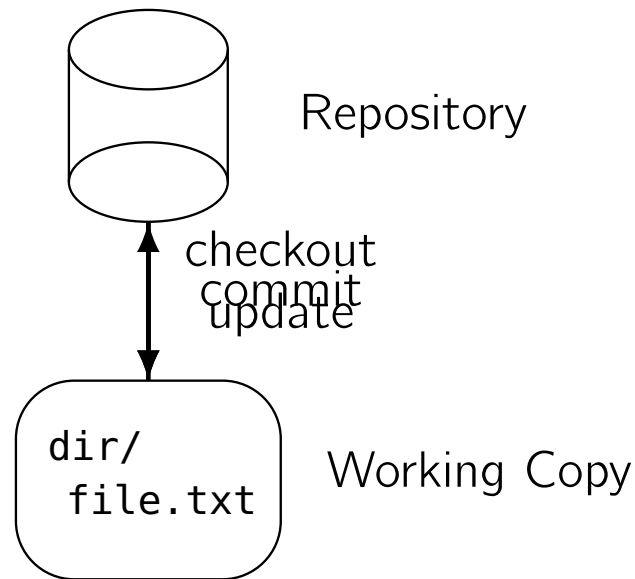
- SVN bietet Versionsverwaltung für Dateien und Verzeichnisse
- Speichert Zusatzinformationen zu jeder Änderung
 - Name des Ändernden
 - Zeitpunkt
 - Kommentar
- Ausführliche SVN-Dokumentation im Subversion-Buch
<http://svnbook.red-bean.com>
- Kommando `svn`
- Grafische Frontends
 - KDESvn (Linux)
 - Tortoise SVN (Windows)
 - SCPlugin (Mac OS X)
- SP-Abgabesystem verwendet Subversion





- **Repository**: zentrales Archiv aller Versionen
 - Zugriff erfolgt beispielsweise per Internet
- **Revision** (Versionsnummer)
 - Fortlaufend ab Revision 0
- **Working Copy** (Arbeitskopie)
 - lokale Kopie einer bestimmten Version des Repositories
 - kann versionierte und unversionierte Dateien und Verzeichnisse enthalten
 - es kann mehrere Arbeitskopien zu einem Repository geben (z. B. CIP/daheim)





- `checkout/co`: Anlegen einer neuen Arbeitskopie
- `update/up`: Neuste Revision aus dem Repository holen
 - Bezieht sich auf aktuelles Verzeichnis und alle enthaltenen Verzeichnisse
- `commit/ci`: Einbringen einer neuen Version in das Repository



- Beim Aufruf von `svn commit` öffnet sich ein Editor zum Eingeben des commit-Kommentars
 - Im CIP wird standardmäßig der Editor `nano` verwendet
 - Anderer Editor kann über die Umgebungsvariable `EDITOR` eingestellt werden

```
> export EDITOR=nano
```

 - Umgebungsvariable ist nur in dieser Shell-Sitzung gültig
 - Durch Eintragen des Kommandos in die Konfigurationsdatei der eigenen Shell (z. B. `.bashrc`) wird der Standardeditor für jede neue Shell geändert

- Übergabe des Kommentars als Argument von `svn commit`

```
> svn commit -m "Ich schreibe lieber gleich in die Befehlszeile  
und nicht in den Editor"
```



- **add:** Dateien unter Versionskontrolle stellen
 - Bei einer leeren Arbeitskopie müssen entsprechende Dateien oder Verzeichnisse erst eingefügt werden!
- **del/remove/rm:** Dateien lokal löschen und nicht länger unter Versionskontrolle halten
- **status/st:** Änderungen der Arbeitskopie anzeigen

```
> svn status
A    aufgabe1/lilo.txt
M    aufgabe1/lilo.c
?    aufgabe1/lilo
!    aufgabe1/lilo.o
```

- A** Datei wurde unter Versionskontrolle gestellt
- M** Dateiinhalt wurde verändert
- ?** Datei steht nicht unter Versionskontrolle
- !** Datei steht unter Versionskontrolle, ist aber nicht mehr in der Arbeitskopie vorhanden



Agenda

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem**
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



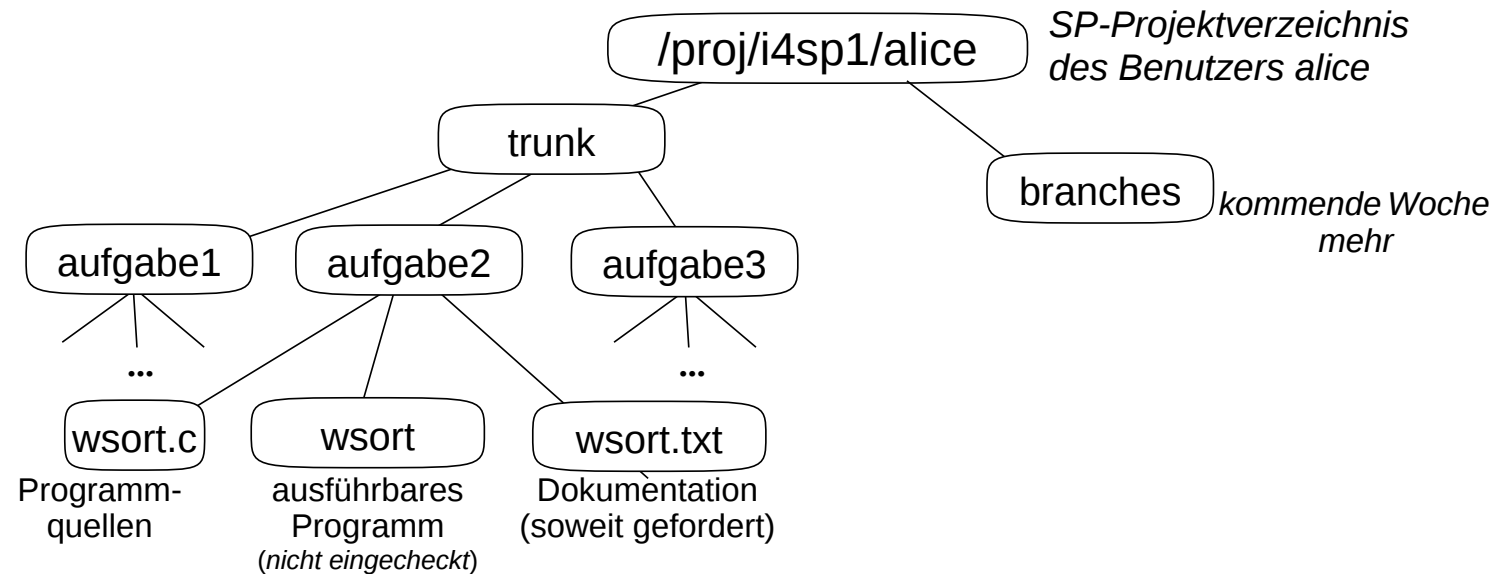
- Für jeden Teilnehmer wird folgendes bereitgestellt:
 - ein Repository `https://www4.cs.fau.de/i4sp/ss15/sp1/<login>`
 - ein Projektverzeichnis `/proj/i4sp1/<login>` mit Arbeitskopie
- Die Erzeugung erfolgt in der Nacht nach der *WAFFEL*-Anmeldung

SVN-Passwort

- Zum Zugriff aufs Repository muss ein SVN-Passwort gesetzt werden
 - ```
> /proj/i4sp1/bin/change-password
```
  - Das Passwort wird innerhalb der nächsten Stunde aktiv



# Aufbau des SP-Repositories



- `trunk` enthält ein Unterverzeichnis `aufgabeX` für jede Aufgabe
- unterhalb von `branches` **nichts** editieren oder von Hand ändern



# Abgabe einer Aufgabe

- Zur Abgabe folgendes Skript aufrufen

```
> /proj/i4sp1/bin/submit aufgabe1
```

- dieses gibt die aktuellste Version der Lösung zu Aufgabe 1 ab

- mehrmalige Abgabe ist möglich

- durch erneuten Aufruf des *submit*-Skripts
- gewertet wird die letzte rechtzeitige Abgabe

- **Eigener** Abgabetermin kann per Skript erfragt werden

```
> /proj/i4sp1/bin/get-deadline aufgabe1
Dein Abgabezeitpunkt fuer die Aufgabe 1: hello ist 22.04.2015
um 17:30:00 Uhr
```

- Abgaben nach dem Abgabezeitpunkt sind möglich

- bei Vorliegen eines triftigen Grundes
- Wertung nur nach expliziter Rücksprache mit dem Übungsleiter
- ansonsten wird letzte rechtzeitige Abgabe gewertet



# Agenda

---

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen**
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



- Übersetzen einer Quelldatei mit gcc

```
> gcc -o test test.c
```

- Zur Erinnerung: Starten der ausführbaren Datei test mit ./test

- Verhalten des gcc kann durch Optionen beeinflusst werden

- -g: Erzeugt Debug-Symbole in der ausführbaren Datei
- -c: Übersetzt Quellcode in Maschinencode, erzeugt aber kein ausführbares Programm
- -Wall: aktiviert weitere Warnungen, die auf mögliche Programmierfehler hinweisen
- -Werror: gcc behandelt Warnungen wie Fehler



## ■ *implicit declaration of function 'printf'*

- bei Bibliotheksfunktionen fehlt entsprechendes `#include`
  - entsprechende Manual-Page gibt Auskunft über den Namen der nötigen Headerdateien

```
> man 3 printf
SYNOPSIS
 #include <stdio.h>

 int printf(const char *format, ...);
```

- bei einer eigenen Funktionen fehlt die Forward-Deklaration

## ■ *control reaches end of non-void function*

- in der Funktion, die einen Wert zurückliefern soll, fehlt an einem Austrittspfad eine passende `return`-Anweisung



# Agenda

---

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme**
- 1.8 Gelerntes anwenden



# Portable Programme

- Entwicklung portabler Programme durch Verwendung definierter Schnittstellen

## ANSI C99

- Normierung des Sprachumfangs der Programmiersprache C
- Standard-Bibliotheksfunktionen, z. B. `printf`, `malloc`

## Single UNIX Specification, Version 4 (SUSv4)

- Standardisierung der Betriebssystemschnittstelle
- Wird von verschiedenen Betriebssystemen implementiert:
  - Solaris, HP-UX, AIX (*SUSv3*)
  - Mac OS X (*SUSv3*)
  - ... und natürlich Linux (*SUSv4, aber nicht offiziell zertifiziert*)



# ... und was ist mit Windows?

## ANSI C99

- Von Microsoft Visual C/C++ nicht unterstützt :-(
- GCC läuft auch unter Windows :-)

## Single UNIX Specification, Version 4 (SUSv4)

- Von Microsoft Windows nicht unterstützt :-(
- UNIX-Kompatibilitätsschicht für Windows: Cygwin (<https://cygwin.com/>)
  - ... ist aber eher frickelig :-|

- Ihr wollt eure SP-Programme unter Linux entwickeln. Wirklich.



# Agenda

---

- 1.1 Allgemeines
- 1.2 Organisatorisches
- 1.3 Linux-Kenntnisse
- 1.4 Versionsverwaltung mit SVN
- 1.5 SP-Abgabesystem
- 1.6 Übersetzen von Programmen
- 1.7 Portable Programme
- 1.8 Gelerntes anwenden



## „Aufgabenstellung“

- Leere Datei für Aufgabe 1 abgeben
  - Erforderliche Dateien: `hello.c`
- „Zuhause“ neue Arbeitskopie anlegen

