

Merkblatt für SVN und GCC

SVN SVN steht für subversion und ist ein frei erhältlicher Standard für Versionskontrolle. Versionskontrolle ist notwendig, wenn mehrere Programmierer an einem System arbeiten und gleichzeitig Dateien editieren und verändern. Dann muss ein konsistenter Zustand des Gesamtsystems aufrechterhalten werden, wobei SVN behilflich sein kann. Das SVN verfügt über ein gemeinsames *Repository*, einem großen Pool, wo alle Daten liegen. Das Repository ist in Module aufgeteilt, die abgegrenzte Einheiten darstellen und sich gegenseitig nicht beeinflussen sollten. Ein Modul könnte beispielsweise den Quellcode für ein System beinhalten. Wenn nun ein Programmierer an dem System etwas verändern will, kann er sich die Dateien vom Repository auf seinen lokalen Rechner kopieren, er erstellt sich eine sogenannte *Working Copy*. In SVN-Sprache macht der Programmierer dann einen *checkout*. Seine kopierte Version kann er nun beliebig verändern; wenn er mit seinen Änderungen fertig ist (diese sollten den Code nach Möglichkeit immer lauffähig halten) kann er seine Version des Codes wieder in das Repository zurückschreiben. In SVN-Sprache sagt man, der Programmierer macht einen *commit*. Das SVN prüft jetzt, ob auch noch andere Programmierer dieselben Stellen im Code verändert haben und versucht einen konsistenten Zustand herzustellen. Falls dies nicht möglich ist, gibt es einen Konflikt und die Programmierer oder ein Versionsverwalter muss ihn manuell lösen.

Falls ein Modul schon ausgecheckt ist und wieder daran gearbeitet werden soll, sollte vorher ein *update* gemacht werden, um die Version des Moduls auf den aktuellsten Stand zu bringen.

Im folgenden werden alle relevanten SVN-Befehle kurz beschrieben. Im Linux-svn client können diese Befehle direkt nach dem executable `svn` geschrieben werden.

- `checkout <Modulname>`: Erzeugt eine working copy auf dem lokalen Rechner
- `update <Modulname | Dateiname | Ordner>`: Bringt die Datei/das Modul auf den aktuellen Stand wie er im Repository abgespeichert ist. Falls die eigene working copy aktueller als die Dateien im Repository sind, werden diese Änderungen nicht gelöscht.
- `add <Ordner | Datei>`: Fügt dem Repository eine Datei hinzu (nachher immer `svn commit` ausführen)
- `remove <Datei>`: Löscht eine Datei aus dem Repository (dafür darf sie in der working copy nicht mehr vorhanden sein, nachher immer `svn commit` ausführen)
- `commit <Datei | Ordner | Modul>`: Schreibt alle Änderungen von der lokalen working copy ins Repository.

Um eine SVN working copy auf einem lokalen Rechner zu speichern gibt es zwei Möglichkeiten. Man kann entweder das Kommandozeilentool und die oben aufgeführten Befehle oder eine graphische Oberfläche (s.u.) verwenden. Wenn man einen checkout von ausserhalb des TU Netzwerkes machen will, muss man den Pfad plus Netzwerk mit der `-d` Option angeben: Wenn man

beispielsweise das Modul `team1` von den Lehrstuhlrechnern des Lehrstuhls 16 auschecken will, gibt man folgendes ein:

```
svn checkout  
svn+ssh://username@atnavab5.informatik.tu-muenchen.de/home/shared/progprakt/trunk/team1
```

Hier muss natürlich statt `username` der jeweilige Benutzername und statt `team1` die jeweilige Teamnummer eingesetzt werden. Der String, der `svn` folgt, ist folgendermaßen aufgeteilt:

- `checkout` ist der `svn`-Befehl für einen checkout.
- `svn+ssh` steht dafür, dass eine *secure shell* zum repository-server aufgebaut werden soll.
- `username@atnavab5.informatik.tu-muenchen.de` beschreibt den Benutzer, der eine *working copy* anlegen will und den remote host, auf dem das repository liegt
- `/home/shared/svn/trunk` beschreibt den Pfad auf dem remote host, wo sich das `svn` root-directory befindet, wo also alle module liegen.
- `team1` ist der Name des Moduls, das ausgecheckt werden soll.

Für diesen checkout muss auch eine Umgebungsvariable gesetzt sein, nämlich `SVN_EDITOR`. Um z.B. `vi` als Standardeditor zu setzen, muss ein `export SVN_EDITOR=vi` gesetzt werden. Um diesen export nicht immer wieder neu tätigen zu müssen, kann man unter Linux das export auch in die `.bashrc` Datei im Home-Verzeichnis schreiben, da diese beim Starten der Bash-Shell automatisch geladen wird.

Das Kommandozeilentool kann man unter Linux sowie unter Windows mit CygWin benutzen. In Windows gibt es allerdings ein nettes Tool mit Benutzeroberfläche, *TortoiseSVN*, das man sich kostenlos unter <http://tortoisesvn.tigris.org/> herunterladen kann. Von uns wird die Version 1.8 empfohlen, da erst ab dieser Version keine Konflikte mehr durch commits von Windows und Unix entstehen können.

Nach der Installation von TortoiseSVN kann man im Windows Explorer mit einem Rechtsklick das Menü *TortoiseSVN* erreichen. Unter *Settings* → *Network* stellt man als SSH-Client den von Tortoise mitgelieferten Client *TortoisePlink* ein. Nun erstellt man sich ein lokales Arbeitsverzeichnis und wählt mittels Rechtsklick, den Befehl *checkout*. Im erscheinenden Dialog muss das Verzeichnis des SVN Repositories (s.o.) angegeben werden. Mit den Befehlen *Update* und *Commit* kann man seine lokale Kopie mit dem Repository abgleichen. *Update* lädt die neueste Version vom Server und *Commit* die lokale Kopie auf den Server. Neu angelegte Dateien müssen zuvor mittels *Add* hinzugefügt und später mit *Commit* hochgeladen werden.

GCC GCC ist die Abkürzung von "GNU Compiler Collection". GCC ist eine Sammlung von Kompilern für eine Auswahl von den wichtigen Programmiersprachen. Diese Sammlung beinhaltet unter anderem Kompiler C, C++, Objective-C, Java, Fortran, and Ada. Der Kompiler für C++ ist `g++`.

Kompilieren eines Single-Source C++ Programms

Um ein Programm bestehend aus einer Quelldatei zu kompilieren wird der Befehl

```
g++ main.cpp
```

verwendet. Hierbei wird mit main.cpp die Datei angegeben, die kompiliert werden soll. Die ausführbare Datei ist a.out und das Programm kann durch

```
./a.out
```

ausgeführt werden.

Kompilerflags

Eine vollständige Liste mit allen Kompilerflags kann man auf der Manpage abrufen.

```
man g++
```

Im folgenden werden die wichtigsten Flags kurz aufgelistet:

- `-o [outfile]`: gibt den Namen der ausführbaren Datei an (default a.out).
- `-c [cppfile]`: erzeugt ein Objektfile (.o), das durch späteres Binden (Linken) zu einem ausführbaren Programm wird.
- `-W[warning]`: gibt an welche Warnungen ausgegeben werden (z.B. unbenutzte Variablen, ...). Hier ist `-Wall` ein guter Wert, der zu einer sauberen Implementierung beiträgt.
- `-I[directory]`: gibt alle Verzeichnisse an, in denen Dateien liegen, die durch `# include` eingebunden werden.
- `-L[directory]`: gibt alle Verzeichnisse an, in denen Bibliotheken liegen, die eingebunden werden (zuvor Kompilierte Quelldateien zu den Headern).
- `-l[library]`: gibt die Bibliothek an, die eingebunden werden soll.
- `-D[macro=def]`: um Macros zu definieren.
- ... und viele weitere Optionen, die nach und nach eingeführt werden. Der interessierte Leser sei auf die Man-Pages oder die GNU GCC Internetseiten verwiesen.

Das obligatorische Hello World Programm

Um als Beispiel das einfache und am häufigsten implementierte Programm zu geben, wird im folgenden der Code und die Anweisungen um diesen Code zu Kompilieren und Ausführen angegeben. Hierfür wird in einem beliebigen Texteditor (vim, xemacs, joe) das folgende Programm eingegeben.

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
```

```
{  
    cout << "Hello World!" << endl;  
    return 0;  
}
```

Diese Zeilen werden in der Datei HelloWorld.cpp abgespeichert. Mit dem Befehl

```
g++ -o HelloWorld HelloWorld.cpp
```

wird das Programm kompiliert. Es entsteht die ausführbare Datei HelloWorld. Mit

```
./HelloWorld
```

kann diese ausgeführt werden. Die Zeile "Hello World!" wird auf die Konsole ausgegeben.