

Merkblatt für SVN und Microsoft Visual Studio

Versionskontrolle mit subversion (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@svnavab.informatik.tu-muenchen.de/svn/projsysentwss07/trunk/team1
```

Hier muss natürlich statt `username` der jeweilige Benutzername und statt `team1` der jeweilige Teamname 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@svnavab.informatik.tu-muenchen.de` beschreibt den Benutzer, der eine working copy anlegen will und den remote host, auf dem das repository liegt
- `/svn/projsysentwss07/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/Verzeichnisses, 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.

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 neuste 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.

Programmieren mit Microsoft Visual Studio IDEs (Integrated Development Environments) sind Programme, die das Entwickeln von Software erleichtern sollen. In einer grafischen Benutzeroberfläche werden verschiedene Werkzeuge angeboten, die das Programmieren in großen Projekten vereinfachen. Neben Fenstern, die die einzelnen Dateien strukturiert anzeigen und Dialogen, die Compilerflags komfortabel konfigurieren lassen gibt es auch fortgeschrittene Werkzeuge wie Debugger zum Fehlerfinden und Profiler zum Aufspüren von ressourcenintensiven Programmteilen. Prinzipiell könnte man alles auch per Kommandozeile bedienen, jedoch kann man mit einer IDE effizienter und übersichtlicher Arbeiten. Eine der häufig verwendeten IDEs zur Entwicklung

von C++-Programmen ist Visual Studio .NET 2005. Jeder Student der TU München kann eine kostenlose Version dieser IDE über den Maniac-Server beziehen (<https://prod.maniac.tum.de/ManiacGUI>).

Im Visual Studio bilden mehrere Dateien ein Projekt (*.vcproj). Jedes Projekt hat ein Ziel, das kompiliert wird. In unserem Fall ist das eine exe-Datei. Man kann mehrere Projekte in einer Solution bündeln.

- a) Öffnen Sie das Visual Studio und wählen Sie `File → New → Project`. Als Vorlage dient Ihnen das `Win32 Console Project`. Geben Sie Ihrem Projekt einen Namen und bestätigen Sie mit `OK` und drücken Sie `Next`. Aktivieren Sie unter den `Project Settings` das Kontrollkästchen `Empty Project`.
- b) Nun haben Sie eine Solution mit einem Projekt erstellt. Im `Solution Explorer` können Sie bereits vorhandene Dateien integrieren oder neue hinzufügen. Klicken Sie dazu mittels Rechtsklick auf einen Ordner und dann mit `Add → Existing Item` oder `Add → New Item` zu Ihrem Projekt hinzufügen. Mit `Add New Item` fügen Sie eine neue Datei ein.
- c) Erstellen Sie eine ausführbare Datei, indem Sie auf `Build → Build Solution` klicken, oder drücken Sie `Ctrl-Alt-B`. Die ausführbare Datei liegt dann in einem Unterordner des `Solution-Ordners` mit Namen `Debug`.
- d) Sie können das Programm mit `F5` starten (und automatisch kompilieren, wenn nötig). Sie werden nun beobachten, dass kurz ein Konsolenfenster öffnet, sich dann aber wieder schließt, da das Programm terminiert. Sie können Ihr Programm mittendrin anhalten, indem Sie einen Breakpoint kurz vor dem Verlassen der `main`-Methode setzen (in dieser Zeile kurz auf den grauen Balken links vom Code klicken). Das Programm kann im Debugmodus Zeile für Zeile durchschritten werden.

Die Dateien müssen so in das auf den SVN Server eingechekkt sein, dass wir mit einem einzigen Klick das Programm kompilieren können. Bitte also die `.sln`, `.vcproj`, `.cpp` und `.h` Dateien einchecken, jedoch keine anderen!