Installationsanleitung für C++-Programmierung mit Cygwin: Linux-Konsole (und mehr) in Win7

(Version dieser Anleitung: V1.4, erstellt von Jan Reisacher, 2014)

Diese Anleitung ist für diejenigen gedacht, die „puristisch“ mit Konsole und Vim programmieren wollen. Das kann ich als ehemaliger c++-Neuling wirklich empfehlen. Auch wer später auf eine komfortablere Programmierumgebung* umsteigen will, sollte sich zuvor ein wenig mit dem puristischen Ansatz beschäftigen, damit man die Grundlagen erlebt hat.
Das wäre natürlich direkt mit einem echten Linux möglich. Aber vor allem diejenigen, die gerade erst richtig Programmieren lernen, haben vielleicht nicht die Nerven, gleichzeitig c++ und Linux zu lernen. Oder weil Linux nicht läuft, z.B. wenn das Touchpad nicht erkannt wird oder weil kein Speicherplatz für ein Linux in einer Virtual Box da ist. Oder weil die Virtual Box ganz Windows schreddert oder Linux von Solidworks zerlegt wird.
Cygwin macht die Linux-Konsole unter der gewohnten Windows-Umgebung zugänglich, erstellt in einem Ordner die Linux-Ordnerstruktur und stellt für verschiedene Programmiersprachen die grundlegenden Funktionen bereit. Es gibt neben einigen Funktionsunschönheiten aber auch (mindestens) einen größeren Nachteil: manches Zusatztool gibt’s im Original für Windows einfach nicht (z.B. Valgrind, siehe Kapitel 12 für Alternativen). Wer kein Profi-Programm schreiben will, kommt lange auch ohne Valgrind zurecht.
Linux lernen schadet nicht – aber wer das auf später verschieben will: hier eine *ausführliche* Anleitung für Cygwin. Viele Kapitel sind optional und können theoretisch übersprungen werden. Neuinstallationen gehen deutlich schneller.

* (Es gibt eine gratis Uni-Lizenz für Visual Studio und VS Express gibt’s gratis. NetBeans gibt es für Windows, aber das benötigt dann auch Cygwin.)

Hinweise zur Notation:
Sind Wahlmöglichkeiten gegeben, so ist das in Sternchen Angegebene die *getestete Auswahl* (auf 64bit-Win7-HomePrem. bzw. 64bit-Win7-Prof.).
Mit [eckigen Klammern] werden Platzhalter dargestellt.

Hinweise zur Anleitung:
Bitte immer erst den *ganzen* Text eines Listen*punkts* durchlesen, dann diesen ausführen.
Wichtig: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch *als Enter* mit kopiert! Dann werden die Befehle nicht oder nicht korrekt ausgeführt!

Inhaltsverzeichnis

1 Cygwin installieren

- Von https://cygwin.com/ die setup.exe (*32 Bit* oder 64 Bit) runterladen, ausführen. Sollte irgendwann einmal die Verbindung zu den Servern nicht mehr aufgebaut werden können, dann liegt es an einer zu alten setup.exe-Version.
- Im Installer nacheinander folgende Antworten eingeben: *Install from Internet* > *[beliebigen Pfad, nur nicht C:\ oder all Users]* > *[default Pfad]* > *ftp://ftp.inf.tu-dresden.de* > Warnmeldung akzeptieren > Großes Auswahlfenster erscheint, erst mal mit der Default-Auswahl weiter > Installation > Icons anlegen > Fertig.
  (Anmerkung: Mit Hilfe der setup.exe können jederzeit Dateien nachgeladen werden. Entweder im großen Auswahlfenster durch einmaligen Klick auf „Skip“ Neues auswählen, dann eventuelle Dependencys akzeptieren. Mit der Schaltfläche „View“ kann dabei durch verschiedene Ansichten geschaltet werden. Oder: über die Windows-Konsole wie in Schritt 2.)

2 Cygwin Plugins installieren

- Windows Konsole öffenen (Start > cmd.exe, Rechtsklick > als Administrator). Dann den *Pfad der setup.exe* auswählen. Mit einem der folgenden Befehle Plugins installieren. Als Verbindungs-Einstellungen werden automatisch die der Installation aus Schritt 1 verwendet. (Die setup.exe heißt eventuell anders.)
  Minimalauswahl (~400 MB):

```
[Path>] setup.exe -q -P gcc-g++,make,ncurses,python,vim,gdb
```

  Eine weitere getestete Auswahl (>450 MB):

```
[Path>] setup.exe -q -P gcc-g++,make,mingw-
runtime,ncurses,perl,python,ruby,rxvt,vim,gdb
```

  Achtung!: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch als Enter mit kopiert! Dann funktioniert es nicht richtig!
  Deswegen die zweite Auswahl nochmal als Einzeiler:

```
[Path>] setup.exe -q -P gcc-g++,make,mingw-runtime,ncurses,perl,python,ruby,rxvt,vim,gdb
```

- Enter, installieren lassen. Eine neue Cygwin-Terminal-Verknüpfung erscheint auf dem Desktop. Die Windows-cmd-Konsole schließen.
  (Ncurses momentan noch nicht verwendbar – siehe bei Bedarf später Kapitel 7.)

3 Optional: Vim und gcc-g++ Compiler testen

- Vim starten: In das neue Cygwin-Terminal: „vim“ oder unter \Cygwin\bin\vim-nox.exe. Falls die Tastatureingabe im Insert-Modus komisch ist (vielleicht gehört das so, ich kenn mich mit vim nicht so aus), dann einfach separates vim installieren. (Siehe Kapitel 9.)
- Mit Vim die Datei CompileTest.cpp öffnen. Schlüsselworte wie int oder „Text-in-Anführungszeichen“ sollten bunt markiert sein.
  Sonst: `:syntax enable` eintippen, damit Code erkannt und markiert wird.

Inhalt CompileTest.cpp

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
  if (argc != 2) {
    printf("Wrong number of parameters. Usage: ./CompileTest
<Number>\n");
    exit(1);
    }
  printf("Hello human!");
}
```

- Mit `:q` Vim beenden (bzw. `:q!` zum speichern und beenden, `:qa` zum Beenden ohne speichern).

- Cygwin-Terminal starten und mit den üblichen Linux-Konsolenbefehlen, den Pfad auswählen, in dem CompileTest.cpp und MyFirstTest.cpp sind. Befehle dafür:

Befehl	Beispiel	Auswirkung
cd [Laufwerk]:	cd E:	Laufwerk auswählen
cd [Pfadstücke]	cd Eigene\ Dateien/…	Pfad wählen (einz. Leerz. mit \ markieren)
cd ..	cd ..	eine Ordner-Ebene höher (mit Leerzeich.)
dir	dir	Dateien im aktuellen Ordner anzeigen
TAB-Taste	cd Mus → cd Musik	Auto-Vervollständigen(Case-Sensitive!)
ALT+Pfeiltasten	ALT+Pfeil-nach-Oben	Zuvor eigetippte Befehle durchsuchen

Anmerkung: copy/paste im Cygwin-Terminal geht so: Text markieren + Strg (schon fertig), Einfügen mit Shift+Einfg. Sind in einem eingefügten Pfad Leerzeichen, gesamten Pfad in „Anführungszeichen“ setzen.

- Ist der Zielpfad eingegeben, folgende Befehle eingeben:
  `g++ -o CompileTest.exe CompileTest.cpp`
  Im Ordner entsteht eine neue .exe Datei. (Das .exe könnte weggelassen werden, da man das Programm momentan eh nur per Konsole starten kann. („cygwin1.dll fehlt“))

- `./CompileTest.exe`
  Es sollte “`Wrong number of parameters. Usage: ./CompileTest <Number>`” erscheinen.

- `./CompileTest.exe 100`
  Jetzt sollte „`Hello human!`“ erscheinen. Wenn kein Fehler auftritt, scheint es zu laufen. (Cygwin-Terminal für die Schritte 4-6 geöffnet lassen.)

4 Die codestyle-Datei cpplint.py runterladen und mit Python nutzen

- In Cygwin mal nur „python“ eintippen und schauen, ob es gefunden wird. Es sollte Text erscheinen, der auf die Hilfe (und copyright, credits, license) aufmerksam macht. (Beim Hilfe angucken kommt man durch: „quit“ +enter und anschließend: quit() +enter wieder zurück.)

- Die cpplint.py von der AD-Wiki, Daphne > SVN > xx123 > CodingStandards > cpp runterladen und einen Ordner höher als die .cpp-Dateien speichern. Beispiel: „…/xx123/uebungsblatt-01/*.cpp“ dann also *neben* „uebungsblatt-01“. Diese Datei soll mit in das SVN hochgeladen werden.

- Per Cygwin-Konsole: `python ../cpplint.py CompileTest.cpp` eintippen.

- Fall 1: Es tritt folgende Fehlermeldung auf:
  `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
  Dann scheint beim Runterladen der cpplin.py vom AD-Wiki was schiefgegangen zu sein: Es wurden wohl die Formatierungs-Befehle mit kopiert. Die cpplint.py > rechtsklick öffnen mit WordPad/Vim/(Editor). Eigentlich sollte hier „`#!/usr/bin/python2.4`“ in der ersten Zeile, dann `Copyright (c) [Jahr] Google blabla`, und danach lauter Lizenzvereinbarungs-Text mit # davor stehen. Wenn der Fehler auftritt, stehen wohl überall unzählige <…>-Klammern. Also noch mal

irgendwie richtig runterladen – z.B.: *Workaround: Auf Daphne die cpplint.py öffnen, den ganzen Text kopieren (An den Anfang klicken (vor ein „#“), runter scrollen, shift + an das Ende klicken (hinter „main()“)) und in ein neues .txt-file auf dem eigenen PC einfügen, welches anschließend den Namen cpplint.py bekommt. Diese natürlich an den oben genannten Pfad speichern.*
Nochmal in die Cygwin-Konsole: `python ../cpplint.py CompileTest.cpp`
Jetzt sollte Fall 2 greifen.

- Fall 2: Es tritt folgender, **guter** Fehler auf:

```
Ich@Mein-PC /cygdrive/[meinPfad]/xx123/testingFolder
$ python ../cpplint.py CompileTest.cpp
CompileTest.cpp:13:  Line ends in whitespace. Consider deleting
these extra spaces.  [whitespace/end_of_line] [4]
Done processing CompileTest.cpp
Total errors found: 1
```

Das heißt, der Stylecheck hat die am Ende von Zeile 13 der CompileTest.cpp versteckten Leerzeichen gefunden. Bei Bedarf diese löschen und wie gehabt nochmal testen; dann hoffentlich fehlerlos.

5 gtest installieren, um test units nutzen zu können

- gtest-1.6.0.zip an einen Ort entpacken (z.B. mit 7-zip), **dessen Pfad keine Leerzeichen enthält!**

- *Aus: https://code.google.com/p/tonatiuh/wiki/InstallingGoogleTestForWindows
To avoid compilation errors, go to " \gtest-[version]\include\gtest\internal\gtest-port.h" file and insert:

```
#define GTEST_HAS_PTHREAD 0
```

before line ~~144~~:

```
#ifndef GTEST_HAS_PTHREAD
```

Wobei statt ~~144~~ wohl 414 gemeint war! (Selber Suchfunktion nutzen; dabei das #ifndef mitsuchen.)*

- Jetzt wie bei http://ad-wiki.informatik.uni-freiburg.de/teaching/ProgrammierenCplusplusSS2012/GTest:
Im Cygwin-Terminal mit cd nach \gtest-[version] wechseln. Dann nacheinander in die

Konsole tippen und jeweils mit Enter ausführen:

./configure

make

Kurz warten bis fertig. Irgendwo mitten im Konsolen-Output steht wahrscheinlich:

```
*** Warning: This system can not link to static lib archive lib/libgtest.la.
*** I have the capability to make that library automatically link in when
*** you link to this library.  But I can only do this if you have a
*** shared version of the library, which you do not appear to have.
```

Ignorieren. Macht vllt. irgendwann mal Probleme, aber vielleicht auch nie.

- Jetzt nachschauen, ob in folgenden Ordnern etwas drin ist und diese dann kopieren:

Hier nachschauen:	Anzahl Dateien:	Hierhin kopieren:
…\gtest-1.6.0\include\gtest	ganzen Ordner	/cygwin/usr/local/include
…\gtest-1.6.0\lib/.libs/*	6	/cygwin/usr/local/lib

`/cygwin/usr/local/include` muss evtl. neu erstellt werden.

- Besonders folgende, wichtige Files sollten jetzt vorhanden sein:
  `/usr/local/include/gtest/gtest.h`
  `/usr/local/lib/libgtest.a`
  ~~`/usr/local/lib/libgtest.so`~~
  (Sollte `libgtest.so` überraschenderweise dabei sein, wäre toll; wenn nicht, egal.)

6 Optional: gtest mit Mini-Programm testen

- In der Cygwin-Konsole den Pfad zum MyFirstTest.cpp öffnen.

Inhalt MyFirstTest.cpp

```
#include <gtest/gtest.h>

// Silly test program. Is really true == true?
TEST(MyFirstTest, 1) {
  ASSERT_TRUE(true);
}

// Main program runs all tests.
int main(int argc, char** argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

- In der Cygwin-Konsole folgendes eingeben, um den MyFirstTest.cpp zu compilieren:

`g++ -o MyFirstTest.exe MyFirstTest.cpp -I../gtest/include/`

`/usr/local/lib/libgtest.a –lpthread`

Wichtig: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch *als Enter* mit kopiert! Dann werden die Befehle nicht oder nicht korrekt ausgeführt! Darum hier nochmal als Einzeiler:

```
g++ -o MyFirstTest.exe MyFirstTest.cpp -I../gtest/include/ /usr/local/lib/libgtest.a –lpthread
```

(Diese Pfadangabe muss später auch so in das Makefile. Auch Jenkins versteht das.)

- Es erscheint *keine* Ausgabe. (Normalerweise erscheinen die Testergebnisse beim compilieren, das hier ist etwas vereinfacht zum Testen der Tests.)

- Dann die MyFirstTest.exe wie immer ausführen: `./MyFirstTest`
  Es sollte das Folgende erscheinen:

```
$ ./MyFirstTest.exe
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from MyFirstTest
[ RUN      ] MyFirstTest.1
[       OK ] MyFirstTest.1 (0 ms)
[----------] 1 test from MyFirstTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
```

- Die gtest-Installation ist dann funktionsfähig. (abgesehen von der `libgtest.so`, die anscheinend nicht benötigt wird.)

7 Optional: Ncurses fertig installieren

- Ncurses ist eine Bibliothek, die die „Steuerung“ des Terminals ohne ANSI Escape Codes ermöglicht – z.B. am bestimmte Positionen schreiben, Tastendrücke und Mausklicks auslesen, Fensterboxen anlegen, usw. Relativ einfach und sehr angenehm.

- In Kapitel 2 wurden bereits einige Dateien für Ncurses installiert. Diese reichen jedoch noch nicht aus: Die setup.exe (ohne Konsole) starten, nach "ncurses" suchen.

  "libncurses-devel: (devel) libraries for terminal handling"

  Durch einen Klick auf „Skip“ wird die zu installierende Versionsnummer eingeblendet. Installieren, fertig.

- Beim Compilieren fügen normale Linuxer ein -lncurses in ihr Makefile ein und es funktioniert alles. Unter Windows wird dies einfach ersetzt durch:
  `-I /usr/include/ncurses /lib/ncurses/libncurses.a`
  Beispiel: g++ [Filename].cpp -I /usr/include/ncurses /lib/ncurses/libncurses.a
- Jenkins kann das jedoch nicht! Der braucht das normale -lncurses. (Tipp: beide Fälle ins Makefile schreiben und den jeweils nicht benötigten mit # auskommentieren.)

- Man kann bei der ncurses-Installation die gtest-Dateien kaputt machen! Klingt komisch, ist aber so. Gtest neu bauen und nochmal einfügen und alles geht.

8 Optional: Die vimrc der Vorlesung einbinden

- Von der AD-Wiki unter "Editoren, insbesondere Vim" > "Die persönliche .vimrc von Hannah Bast" runterladen – oder die etwas veränderte Ausgabe dieser Datei verwenden, die dieser Anleitung beiliegt.
  Die runtergeladene Datei umbenennen in _vimrc (.vimrc geht theoretisch auch. Der Punkt markiert unter Linux versteckte Dateien, lässt sich in Windows aber nicht direkt eingeben).

- (Wie hier: http://cygwin.com/faq/faq.setup.html#faq.setup.home)
  Start > Ausführen: cmd.exe > *als Admin*
  Eintippen (mit diesem Befehl wird noch nichts geändert): `set HOME`
  Möglicher Output:

```
HOME=E:\Eigene Dateien\[MyName]\Documents\PSpice Workspace
HOMEDRIVE=C:
HOMEPATH=\Users\[MyName]
```

- Also *müsste* die _vimrc gespeichert werden unter:
  `HOME=E:\Eigene Dateien\[MyName]\Documents\PSpice Workspace`.
  Das Cygwin-Terminal legt auch seine Einstellungsdateien („skeleton files“) dort ab. Das ist ausgesprochen doof. Am einfachsten ergibt man sich aber und speichert die vimrc da hinein und gut ist. Alternativ muss man – wie im übernächsten Punkt folgt – die Home-Pfad-Variable ändern.
- Es kann auch sein, dass die erste Zeile komplett fehlt, d.h. die HOME-Variable existiert noch nicht. Dann verwendet Cygwin automatisch den „richtigen“ Pfad; sollte je-

doch ein anderes Programm (z.B. PSpice) später die Pfad-Variable neu anlegen, würde Cygwin diesen neuen Pfad übernehmen und nur dort nach der vimrc suchen.

- Wie man die Home-Variable verändert: Start > Rechtsklick auf Computer > Eigenschaften > Erweiterte Systemeinstellungen > Erweitert > Umgebungsvariablen. In beiden Tabellen die Variable `HOME` suchen. Eigentlich sollte man hier mit Semikolon getrennt mehrere Home-Pfade eingeben können, aber zumindest meine Cygwin-Version scheint leider zu blöde zu sein, mehrere Pfade auseinanderhalten zu können. Existiert bereits eine Home-Variable (z.B. von PSpice), dann diese umbenennen (z.B. in `HOME_PSPICE`). Jetzt funktioniert PSpice evtl. nicht mehr richtig. Wenn HOME nicht (mehr) existiert, neu erstellen (am besten unter Systemvariablen) und festlegen auf: * `/cygdrive/[cygwin-Pfad]/cygwin/home/[Benutzername]` *
- Wieder mit der cmd.exe und `set HOME` nachschauen, ob's funktioniert hat. Dann Cygwin-Terminal öffnen. Wenn der frühere Pfad abgeändert wurde, erscheint eine Meldung: „`Copying skeleton files.`“ [neue .xxx Dateien werden angelegt.]
- Dann die _vimrc ebenfalls in den neuen Home-Ordner speichern.

- Wenn in Vim in einer Spalte links neben dem Code die Zeilennummern eingeblendet sind, dann ist die _vimrc aktiv.

Anmerkung – Aus dem Vim-Manual ~"Starting 4.3":
Recommended place for your personal initializations:
MS-DOS and Win32: `$HOME/_vimrc` or `$VIM/_vimrc`

9 Optional: Ein zweites Vim installieren

- Das kann man machen, um ein Windows-Editor-ähnlicheres, nicht komplett konsolengebundenes Vim zu haben. Man kann z.B. per Reiter „Datei“ > „Speichern unter“ nutzen oder ebenso per Reiter auf die Optionen zugreifen. Sonst ist alles gleich, man kann es bei Bedarf immer noch komplett mit den Konsolen-Befehlen steuern. Falls man z.B. den Microsoft-Editor mal mit Vim ersetzen will.

- Von http://www.vim.org/download.php unter der Überschrift „PC: MS-DOS and MS-Windows“ die .exe (z.B. *gvim73_46.exe*) runterladen. Installieren: beliebigen Pfad, nur nicht unter „Cygwin“ oder einen Cygwin-Unterordner. Getestet: *full install*

- Im Ordner der Vim-Installation, unter [Laufwerk]:\ [Vim-Pfad]\, muss nun ebenfalls die _vimrc (oder .vimrc) gespeichert werden; neben die Ordner „vim[Version]“ und „vimfiles“. Ein bestehendes vimrc ohne Punkt oder Unterstrich kann bestehen bleiben.

- Wenn nun in Vim in einer Spalte links neben dem Code die Zeilennummern eingeblendet sind, dann ist die _vimrc aktiv.

10 Optional: C++11-Funktionen verwenden

- Will man C++11-Code ausführen, erscheint eine Warnung, man solle im Makefile -std=c++11 oder -std=gnu++11 einfügen. Es funktioniert aber nur Zweites. Bei ersterem erscheint: "...FOpen nicht gefunden..." oder so ähnlich. -std=c++0x funktioniert ebenfalls nicht. Also:
  `g++ ... -std=gnu++11`

11 Hinweise

- Für alle, die ihre „Cygwin Terminal.exe“ verloren haben:
  `[Mein Pfad]\Cygwin\bin\mintty.exe -i /Cygwin-Terminal.ico -`
  So wird’s vom Installationssetup erstellt. Das Minus am Schluss gehört dazu.
- Die Windows-cmd-Konsole ist von den Befehlen ähnlich zum Linux-Terminal, aber im Ergebnis manchmal leicht verschieden (Beispiele: Auto-Vervollständigung mit Tab; Pfadangaben: Windows \, Linux /). Befehle gibt’s im Web: Start Pagen/Googlen.
- Die Auto-Vervollständigung mit TAB ist beim Cygwin-Terminal Case-Sensitive! (Jenkins übrigens auch!)
- Wenn nach einer Konsoleneingabe nichts passiert, erst überlegen / nachlesen, ob das nicht genau so sein soll (eigene Erfahrungen…).
- Man kann das Terminal breiter ziehen. Zuerst ändert sich nichts, doch *neue* Ausgaben werden dann auf ganzer Breite ausgegeben. Sehr praktisch bei langen, (reproduzierbaren) Fehlermeldungen und ASCII-Spielen.
- Die cpplint.py der Vorlesung funktioniert nur unter Python 2.x. Python 3.x ist inkompatibel zu Python 2.x.
- Vim kann in Windows wie üblich als Standardöffneprogramm für .py, .h und .cpp registriert werden.
- Bei Verwendung von Vim in der Cygwin-Konsole empfiehlt sich, für die bessere Lesbarkeit in den Cygwin-Optionen den Hintergrund zu ändern. Ganz weiß ist jedoch auch ungeschickt: Es gibt weiße Schrift… *Empfehlung: ein sehr, sehr dunkles, angenehmes Rot*
- In der vimrc ist z.B. enthalten:

```
"" When inserting TABs replace them with the appropriate number of spaces
set expandtab
"" But TABs are needed in Makefiles
au BufNewFile,BufReadPost Makefile se noexpandtab
```

- Nicht nur die _vimrc kann angepasst werden: Kommandos, die in die `Cywin\etc\bash.bashrc` geschrieben werden, werden beim Konsole öffnen ausgeführt. Sehr praktisch: `cd „[Pfad meiner C++-Programme]“`
  Es können darin auch neue Kommandos festgelegt werden: `alias ll=‚ls -l‘`
- Tolles Programm für die SVN-Nutzung: tortoiseSVN. *(Version 17.12 funktioniert mit Daphne, Version 18.0 nicht. Wurde aber vielleicht inzwischen geändert.)*

- Die Ausgabe eines Programms kann in eine Datei umgeleitet werden: `./[ProgrammName]Main.exe > [DokName].txt` oder `[DokName].csv` (für Excel) funktioniert beides. Ebenso `./HeapSorterMain.exe | tee out.txt`

12 Dinge, die (noch) nicht gehen

- Ob SVN commit & checkout via Cygwin funktionieren, wurde nicht getestet. Siehe für Ersatz im Kapitel Hinweise: tortoiseSVN.
- gprof (-pg) ist nicht im Cygwin Standard-Paket enthalten. (Prüfen, ob über setup.exe installierbar)
- Gnuplot (Prüfen, ob über setup.exe installierbar).
- In Cygwin und Vim Strg-c und Strg-v. Sollte irgendwie lösbar sein. (Man gewöhnt sich aber auch an Strg und Shift+Einfg).
- Offizielle Cygwin Website sagt definitiv: "valgrind is not supported."
  Valgrind kann anscheinend mehrere verschiedene Dinge. Auf der Suche nach „valgrind substitutes“ werden im stackoverflow-Forum folgende „Open Source“ und „Free Tools“ empfohlen – alle nicht getestet und deswegen evtl. falsch einsortiert:

Valgrind:	Ersatz:
1. MemCheck	Dr. Memory, UMDH, AppVerifier
2. Callgrind	verysleepy, AMD CodeAnalyst™ Performance Analyzer, Windows Performance Analysis Tools
3. Massif	VMMap, !heap command in windbg
4. Cachegrind	Windows Performance Tools von oben – nicht so gut und einfach wie Cachegrind
5. DRD	nichts. Ähnlich: „Lock“ Checker des AppVerifier
Abschlussbemerkung des Users: LeakDiag, Deleaker, UMDH, App Verifier, DebugDiag... deleaker - the best of these utilities...	

  http://stackoverflow.com/questions/413477/is-there-a-good-valgrind-substitute-for-windows

13 Weblinks zu Cygwin-Anleitungen

http://www.tu-chemnitz.de/urz/kurse/unterlagen/cygwin/

http://cygwin.com/cygwin-ug-net/overview.html