Installationsanleitung für C++-Programmierung mit der Cygwin-Konsole auf Win7

Diese Anleitung ist vor allem für diejenigen gedacht, die größere Probleme mit Linux haben, z.B. wenn das Touchpad nicht erkannt wird oder beim Installieren einer virtual box das gesamte System krepiert (ist schon passiert – Backup machen). Denn die grundlegenden Funktionen erfüllt Cygwin, doch neben einigen Funktionsunschönheiten gibt es auch echte Nachteile: manches Zusatztool gibt's für Windows einfach nicht (z.B. Valgrind).
Deswegen ist eine virtual box oder ein echtes Linux wohl die langfristig bessere Wahl. Wer trotzdem Cygwin präferiert – hier eine detaillierter Einstieg. Alles natürlich auf eigenes Risiko.

Hinweis: Sind Wahlmöglichkeiten gegeben, so ist das in Sternchen Angegebene die *getestete Auswahl* (auf 64bit-Win7-HomePrem. bzw. 64bit-Win7-Prof.). Außerdem: Bitte immer erst den ganzen Text eines Listenpunkts durchlesen, dann ausführen.
Wichtig: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch als Enter mit kopiert! Das führt zu Fehlverhalten!

Inhaltsverzeichnis

1 Cygwin installieren

- Von http://cygwin.com/install.html die setup.exe runterladen, ausführen. (~90 MB)
- Im Installer nacheinander folgende Antworten eingeben: *Install from Internet* > *[beliebigen Pfad, nur nicht direkt C:\] & all Users* > *[default Pfad]* > *ftp://ftp.inf.tu-dresden.de* > Warnmeldung akzeptieren > Großes Auswahlfenster erscheint, erst einmal 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 einfacher: 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. Den Pfad erweitern (evtl. keine Leerzeichen nach den Kommata)
  Achtung!: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch als Enter mit kopiert! Das führt zu Fehlfunktion!
  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
  ```

- Enter, installieren lassen. (Als Verbindungs-Einstellungen werden automatisch die der Installation aus Schritt 1 verwendet.) Eine Cygwin-Terminal-Verknüpfung erscheint auf dem Desktop. Die Windows-cmd-Konsole schließen.
  (Ncurses momentan noch nicht verwendbar – siehe bei Bedarf 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.
- 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 Up → cd Updates	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 Per 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:
  ./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/herstellbar sein, wäre das sehr gut!)

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`
  Achtung!: Beim Kopieren von Zweizeilern aus diesem Dokument in die Konsole wird der Zeilenumbruch als Enter mit kopiert! Das führt zu Fehlfunktion!
  In zwei Hälften unterteilt kopieren, evtl. überzählige Leerzeichen (> 1) löschen.
  Zwischen …`/include/` und `/usr/`… steht ein Leerzeichen.
  (Diese Pfadangabe muss später auch so in das Makefile. Versteht auch Jenkins – erfolgreich ausprobiert.)

- 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 nicht installierbar ist.)

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, Farbig schreiben, usw. Diese bietet sich für die Bearbeitung des Abschluss-Projektes an.
- 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 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. Neu bauen und nochmal einfügen hilft.

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`.
  Da das ausgesprochen doof wäre, wie gleich 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 jedoch ein anderes Programm (z.B. PSpice) später die Pfad-Variable neu anlegen, würde Cygwin diesen neuen Pfad übernehmen. Darum ebenfalls die Home-Pfad-Variable festlegen:

- Start > Rechtsklick Computer > Eigenschaften > Erweiterte Systemeinstellungen > Erweitert > Umgebungsvariablen. In beiden Tabellen die Variable `HOME` suchen, wenn bereits existent, (z.B. von PSpice, das man hoffentlich nie/selten nutzt), diese umbenennen (z.B. in `HOME_PSPICE`). Jetzt funktioniert PSpice evtl. nicht mehr richtig. (Cygwin scheint leider zu blöde zu sein, mehrere durch Semikolon getrennte Pfade auseinanderhalten zu können.) Wenn HOME nicht (mehr) existiert, neu erstellen (am besten unter Systemvariablen) und festlegen auf:
  `* /cygdrive/[cygwin-Pfad]/cygwin/home/[Benutzername] *`
  (Vielleicht geht auch ein anderer Pfad mit dem Präfix `/cygdrive/`).

- 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.]

- Evtl. veränderte, alte Einstellungsdateien vom alten Home-Verzeichnis herüber kopieren und die frisch angelegten überschreiben. Dann die _vimrc ebenfalls in den neuen Home-Ordner speichern.

- Wenn nun 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 M$-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 Hinweise

- 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 im Web: „Start Page“ / „Google“.
- 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.
- 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.
- 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 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, Version 18.0 nicht)*

11 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: über setup.exe nachinstallierbar?)
- In Cygwin und Vim Strg-c und Strg-v. Sollte irgendwie lösbar sein.
- 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: 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

12 Weblinks zu Cygwin-Anleitungen

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

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