Grundkurs C++

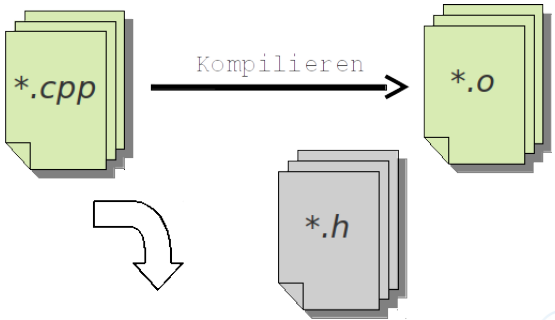
Buildsysteme
Versionsverwaltung mit git

Martin Knopp, Martin Gottwald, Stefan Röhrl

25.04.2018

Buildsysteme

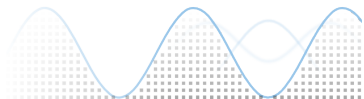
- Beispielhaftes Übersetzungsszenario:
- main.cpp, lcdrange.cpp, lcdrange.h



Headerdateien

```
//blackbox.h
#ifndef BLACKBOX_H
#define BLACKBOX_H

#include "port.h"

class BlackBox {
    public:
        BlackBox();
        ~BlackBox();
    private:
        char buf;
        void transfer(char &data);
        Port *inport;
        Port *outport;
};
#endif
```

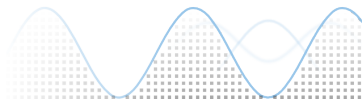
```
//port.h
#ifndef PORT_H
#define PORT_H

class Port {
    public:
        Port(int id=0);
    private:
        int portNr;
};
#endif
```



Headerdateien II (funktioniert nicht!)

```
//blackbox.h
#ifndef BLACKBOX_H
#define BLACKBOX_H

#include "port.h"

class BlackBox {
    public:
        BlackBox();
        ~BlackBox();
    private:
        char buf;
        void transfer(char &data);
        Port *inport;
        Port *outport;
};
#endif
```

```
//port.h
#ifndef PORT_H
#define PORT_H

#include "blackbox.h"

class Port {
    public:
        Port(int id=0);
    private:
        int portNr;
        BlackBox *myBB;
        BlackBox *remBB;
};
#endif
```



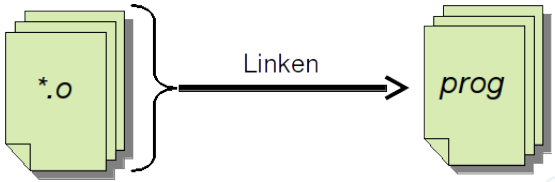
Headerdateien III

```
//blackbox.h
#ifndef BLACKBOX_H
#define BLACKBOX_H

#include "port.h"

class Port;

class BlackBox {
    public:
        BlackBox();
        ~BlackBox();
    private:
        char buf;
        void transfer(char &data);
        Port *inport;
        Port *outport;
};
#endif
```

```
//port.h
#ifndef PORT_H
#define PORT_H

#include "blackbox.h"

class BlackBox;

class Port {
    public:
        Port(int id=0);
    private:
        int portNr;
        BlackBox *myBB;
        BlackBox *remBB;
};
#endif
```

Buildsysteme

- Build-Vorgang:
- Verzeichnis *tutorial7*
- main.cpp, lcdrange.cpp, lcdrange.h
- Compiler
- main.o, lcdrange.o



Manueller Build-Vorgang

- g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -o lcdrange.o lcdrange.cpp
- g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -o main.o main.cpp
- moc-qt4 -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 lcdrange.h -o moc_lcdrange.cpp
- g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -o moc_lcdrange.o moc_lcdrange.cpp
- g++ -Wl,-O1 -o tutorial7 lcdrange.o main.o moc_lcdrange.o -lpthread

Niemand will das von Hand machen!

make

- Compilieren und Linken wird von *make* automatisiert.
- Informationen zu Quelltextdateien und Libraries steht in Makefiles.
- Buildvorgang:
  - Aufruf von make

qmake

- Qt-spezifische Erweiterungen (z.B. QObject, signals, slots).
- Standard-C++-Compiler kennt diese Erweiterungen nicht.
- *Meta Object Compiler (moc)* erzeugt Standard-C++ Dateien.
- Viele Einzelschritte
- Außerdem: *User Interface Compiler (uic)*: QtDesigner

- Vereinfachung durch Verwendung eines Buildsystems: **qmake**

qmake

```
qmake -project
qmake
make
```

(Projektdatei *.pro)
(Makefiles(s) erzeugen)
(*moc*en, kompilieren, linken)

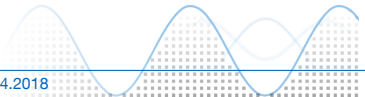
- qmake wird nur aufgerufen, wenn Dateien hinzukommen bzw. entfernt werden.
- Bei Problemen *.pro bearbeiten, nicht aber die Makefiles

qmake – beispielhafte *.pro Datei

```
HEADERS += port.h \
           blackbox.h
SOURCES += build.cpp \
           port.cpp \
           blackbox.cpp
TARGET=../bin/build
```

Teil II

Versionsverwaltung mit git

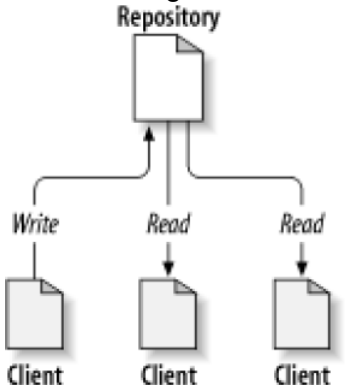


Warum?

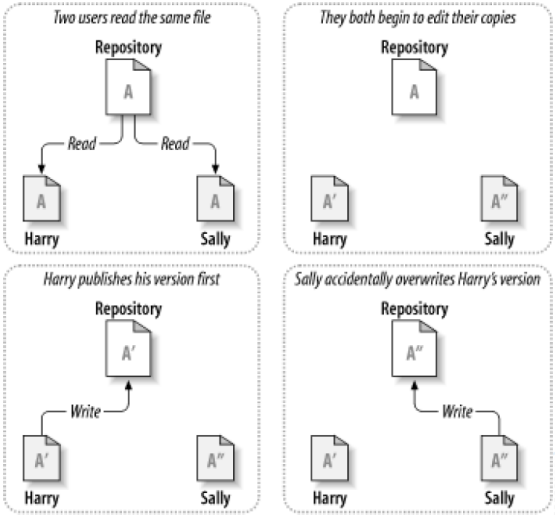
- Programm wird laufend fortentwickelt
- Mehrere Leute arbeiten an einem Programm
- Jeder soll mit möglichst aktueller Version des Programms arbeiten
- Fehler bei Austausch von Dateien sollen vermieden werden, bzw. problemlos korrigiert werden können
- Programmierer sollen
  - …wissen, was wann geändert wurde
  - …Zugriff auf ältere Versionen haben
  - …erkennen, wenn Veränderungen zur gleichen Zeit von verschiedenen Leuten gemacht wurden

Das Repository

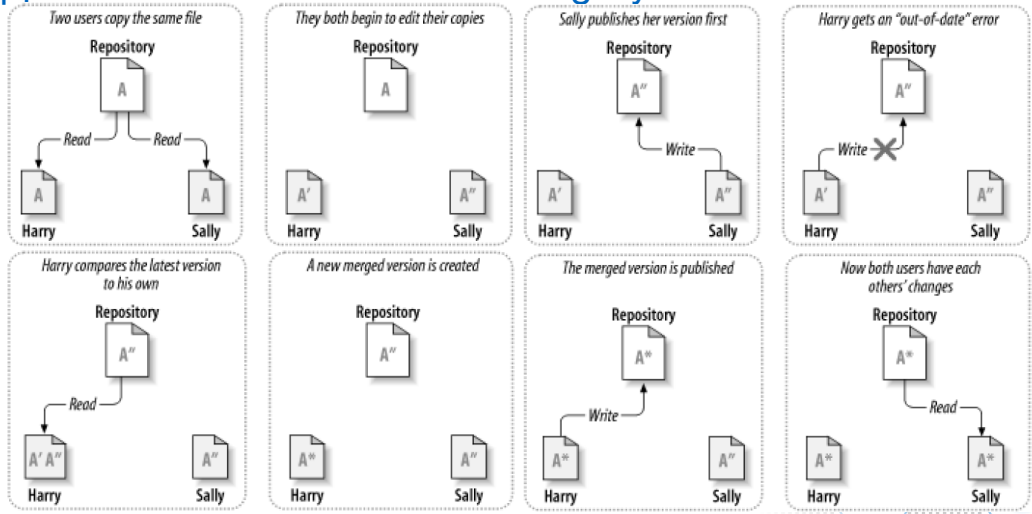
Engl. für Behälter, Ablage, Aufbewahrungsort



Repository – Problemstellung Gruppenarbeit



Gruppenarbeit mit Versionsverwaltungssystemen



Git am LDV

- GitHub-ähnliche Oberfläche unter http://gitlab.ldv.ei.tum.de/
  Login mit LRZ-Kennung & Passwort
- Namen & Mailadresse setzen für Commits:

```
git config --global user.name "Erika Mustermann"
git config --global user.email "e.mm@mytum.de"
```

```
mkdir gruppe1_kurzprojekt
cd gruppe1_kurzprojekt
git init
cp /Pfad_zu_Dateien/* ./
git add *
git commit -m 'first commit'
git remote add origin https://gitlab.ldv.ei.tum.de/cpp-gk18/gruppe1_kurzprojekt
git push -u origin master
```

git für Windows:
https://gitforwindows.org/

Was gehört ins Repository, was nicht?

- Textdateien, die Veränderungen unterliegen:
  - *.cpp, *.h
  - Dokumentation
  - evtl. Projektdateien (z.B. *.pro, CMakeLists.txt) u.ä. (z.B. selbsteditierte Makefiles)
- Beim Übersetzen erzeugte Binärdateien bzw. mit moc/uic erzeugte Dateien (generell alle automatisch generierten Dateien):
  - differenzieller Mechanismus liefert Irrelevantes
  - Repository verfettet
  - Abgleich dauert länger
- Was sonst zum Projekt gehört:
  - Bilder
  - Musik und Effektsounds
  - Konfigurationsdateien für eigenes Programm
  - Testdaten

Normales Arbeiten mit git

- Einmalig: `git clone https://gitlab.ldv.ei.tum.de/cpp-gk18/playground.git`
- Arbeitskopie auf den aktuellen Stand bringen:
  ```
  git pull
  ```
- Änderungen der Arbeitskopie ins Repository (wenn unverändert):
  ```
  git status
  git commit
  git push
  ```
- Änderungen der Arbeitskopie ins Repository (wenn verändert):
  ```
  git status
  git pull
  git commit
  git push
  ```

Zu beachten

- git soll bestimmte Dateien/Ordner ignorieren:
  - Datei „.gitignore“ im Hauptverzeichnis erstellen.
  - Jede Zeile gibt eine Datei oder einen Ordner an, den git ignorieren soll
  - Pfade beziehen sich auf das Hauptverzeichnis
  - Platzhalter (*, ?) sind erlaubt
- Finger weg von .git-Ordnern und deren Inhalt
- Weitere wichtige Kommandos:
  - `git log`
  - `git <befehl> --help`
  - `git diff`
  - `git checkout <Datei>`
  - `(git init / git clone)`
- Schöne „Cheat-Sheets“:
  - https://www.git-tower.com/blog/git-cheat-sheet/
  - https://education.github.com/git-cheat-sheet-education.pdf

Hausaufgabe

- heute Abend in Moodle
- Einfache GUI-Anwendung mit Qt
- diesmal keine Binaries nötig!
- Abgabetermin: 02. Mai 2018, 15:00 Uhr
- Bei Fragen und Problemen: cpp-tutor@ldv.ei.tum.de