Übung zu Betriebssysteme

Organisation und Einführung

17. & 18. Oktober 2019

Andreas Ziegler, Bernhard Heinloth, Christian Eichler

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Organisation des Übungsbetriebs

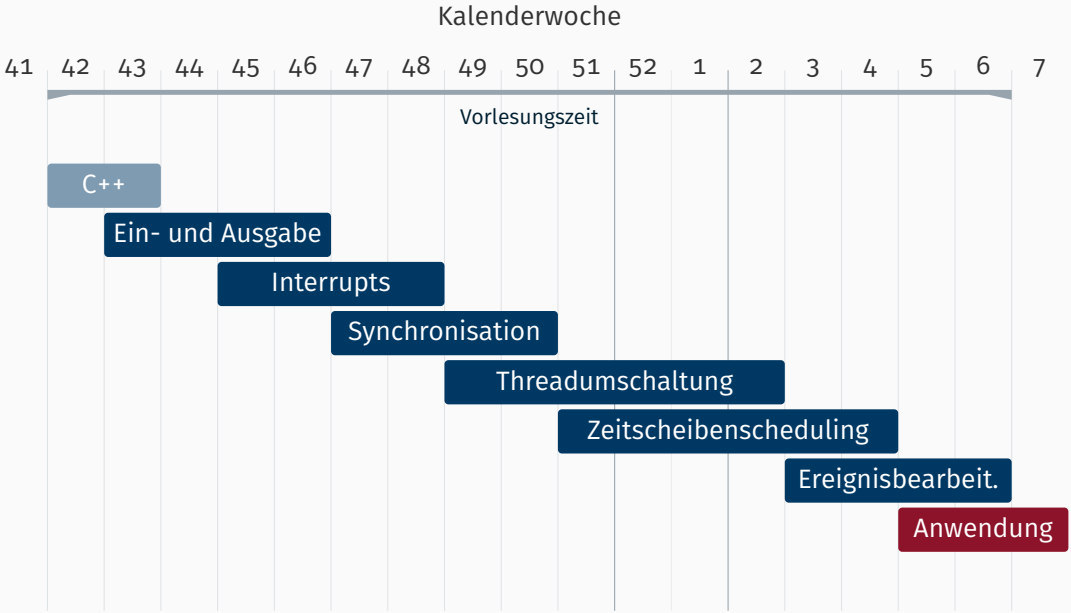
OOStuBS
single-core
5 ECTS Modul

MPStuBS
multi-core
7.5 ECTS Modul

OOStuBS
single-core
5 ECTS Modul

MPStuBS
multi-core
7.5 ECTS Modul

Zeitplan (Übungsaufgaben)



Zeitplan (Woche)

Zeitplan (Semester)

Oktober 2019

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November 2019

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Dezember 2019

						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Januar 2020

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Februar 2020

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

Vorlesung

Tafelübung für neue Aufgabe im Aquarium

Seminar im Aquarium

Abgabe der Aufgabe in der **Rechnerübung** im WinCIP

- Abgabe nur in festen **2er Gruppen**
- eine (obligatorische) Tafelübung pro Aufgabe
- Anmeldung zur Tafelübung (bis 31. Oktober) im Waffel auf **waffel.informatik.uni-erlangen.de/signup?course=380**
- Informationen und Aufgabenstellung auf **www4.cs.fau.de/Lehre/WS19/V_BS/**
- Quelltextvorlagen der Aufgaben im Gitlab

  `https://gitlab.cs.fau.de/i4/bs/oostubs`
  `https://gitlab.cs.fau.de/i4/bs/mpstubs`

Zusatzseminare

- Termine
  - jeweils Donnerstags ab 12:15 Uhr im Aquarium
  - Dauer ca. 45 Minuten (?)

  heute Einführung in Git & C++
  31.10. Fehlersuche mit dem GDB *(Christian Eichler)*
  14.11. (Ur)Laden des x86er *(Bernhard Heinloth)*
  28.11. Programmierung in Assembler *(Andreas Ziegler)*
- Ziele
  - Unterstützung für die Übungen
  - besseres Verständnis der Zusammenhänge im Hintergrund

  → unter'm Strich (hoffentlich) Zeit- & Stressersparnis

Zusatzseminare

- Termine
  - jeweils Donnerstags ab 12:15 Uhr im Aquarium
  - Dauer ca. 45 Minuten (?)

  heute Einführung in Git & C++
  31.10. Fehlersuche mit dem GDB *(Christian Eichler)*
  14.11. (Ur)Laden des x86er *(Bernhard Heinloth)*
  28.11. Programmierung in Assembler *(Andreas Ziegler)*
- Ziele
  - Unterstützung für die Übungen
  - besseres Verständnis der Zusammenhänge im Hintergrund

  → unter'm Strich (hoffentlich) Zeit- & Stressersparnis
- Teilnahme ist **freiwillig**
- **nicht** prüfungsrelevant

Zusatzseminare

- Termine
  - jeweils Donnerstags ab 12:15 Uhr im Aquarium
  - Dauer ca. 45 Minuten (?)

  heute Einführung in Git & C++
  31.10. Fehlersuche mit dem GDB *(Christian Eichler)*
  14.11. (Ur)Laden des x86er *(Bernhard Heinloth)*
  28.11. Programmierung in Assembler *(Andreas Ziegler)*
- Ziele
  - Unterstützung für die Übungen
  - besseres Verständnis der Zusammenhänge im Hintergrund

  → unter'm Strich (hoffentlich) Zeit- & Stressersparnis
- Teilnahme ist **freiwillig**
- **nicht** prüfungsrelevant
- Feldversuch dieses Semester
  - Bitte Feedback geben!

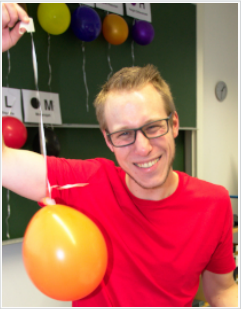
Die Übung wird zwar durch Folien unterstützt – diese dienen jedoch ausschließlich als ergänzende Veranschaulichung des zu vermittelnden Stoffes und haben somit keinen Anspruch auf Vollständigkeit.

Sie sind nicht zum Selbststudium geeignet
und auch nicht auf Druck optimiert!

Übungsleiter (ABC-Truppe)

Andreas Ziegler

Bernhard Heinloth



Christian Eichler

Ausgebildete Ersthelfer +

Helene Gsänger

Harald Böhm

Ausgebildete Ersthelfer +

Helene Gsänger

Harald Böhm

Gummi N.T.

Kommunikation

Eskalationsstufen

- Rechnerübung
- Tafelübung
- **#faui4bs** im IRCnet
- XMPP-MUC **i4bs@conference.cs.fau.de**
- Mail an **i4stubs@lists.informatik.uni-erlangen.de**
- **Raum 0.055** in der Martensstr. 1 (begründeter Notfall)

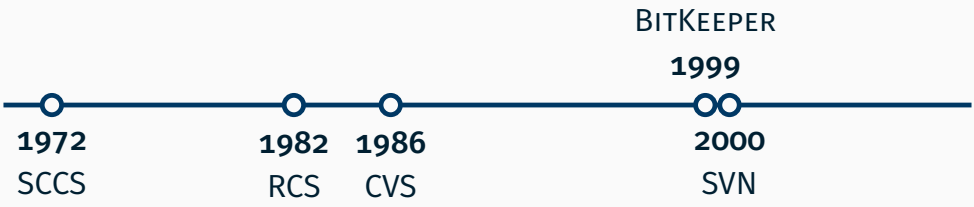
Eine kleine Einführung in Versionsverwaltungssysteme

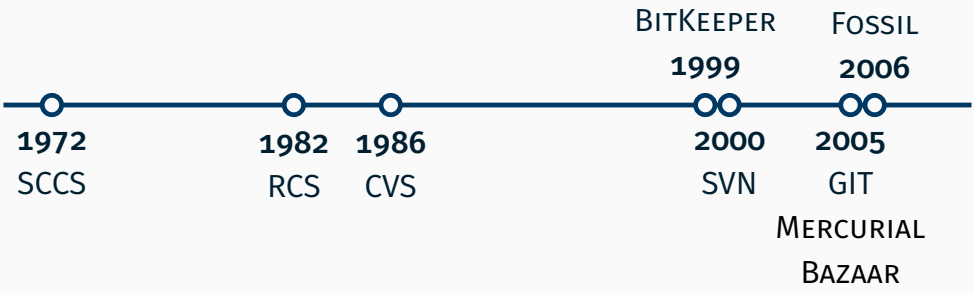
1972
SCCS

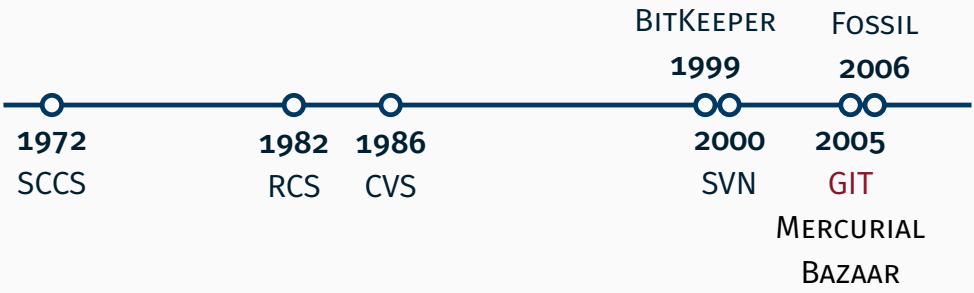

1972
SCCS
1982
RCS


1972
SCCS
1982
RCS
1986
CVS


1972
SCCS
1982
RCS
1986
CVS
2000
SVN

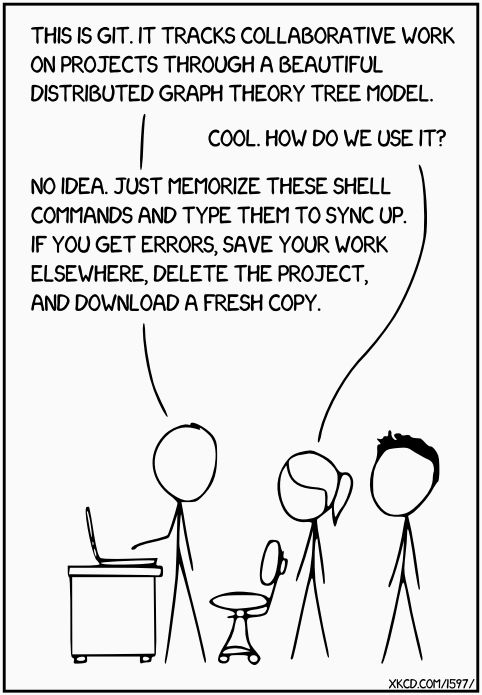

BitKeeper
1999
1972
SCCS
1982
RCS
1986
CVS
2000
SVN


BitKeeper
1999
Fossil
2006
1972
SCCS
1982
RCS
1986
CVS
2000
SVN
2005
GIT
Mercurial
Bazaar


BitKeeper
Fossil
1999
2006
1972
SCCS
1982
RCS
1986
CVS
2000
SVN
2005
GIT
Mercurial
Bazaar

Schlüsselkonzepte von GIT

- Integrität durch SHA-1 Hash
- vollständiges Speichern der Daten (*snapshot*)
- dezentral (*clone*)
- nicht-lineare Entwicklung (*branch*)

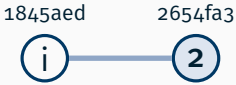

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.
XKCD.COM/1597/

```
heinloth:~$ mkdir beispiel
heinloth:~$ cd beispiel
heinloth:~/beispiel$ git init
Leeres Git-Repository in  /beispiel/.git/ initialisiert
```

```
heinloth:~/beispiel$ touch README.md
heinloth:~/beispiel$ git add README.md
heinloth:~/beispiel$ git commit -m "initialer commit"
[master (Basis-Commit) 1845aed] initialer commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

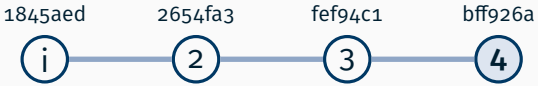
1845aed

```
heinloth:~/beispiel$ echo "23" > foo
heinloth:~/beispiel$ git add foo
heinloth:~/beispiel$ git commit -m "hier sollte eine gute nachricht stehen"
[master 2654fa3] hier sollte eine gute nachricht stehen
 1 file changed, 1 insertion(+)
 create mode 100644 foo
```



```
heinloth:~/beispiel$ echo "42" > foo
heinloth:~/beispiel$ git add foo
heinloth:~/beispiel$ git commit -m "noch mal ein commit"
[master fef94c1] noch mal ein commit
 1 file changed, 1 insertion(+), 1 deletion(-)
```

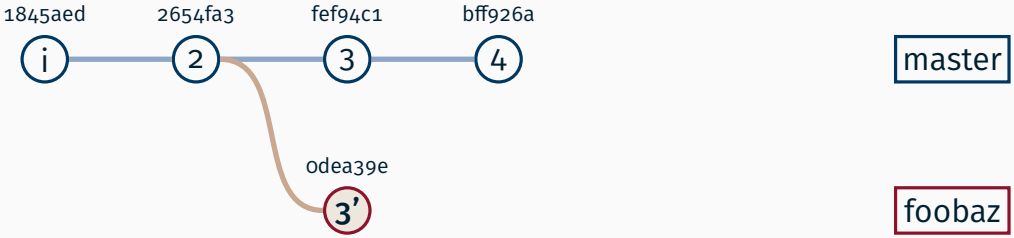
```
heinloth:~/beispiel$ echo "1337" > bar
heinloth:~/beispiel$ echo "zweiundvierzig" > foo
heinloth:~/beispiel$ git add bar foo
heinloth:~/beispiel$ git commit -m "aenderungsnachricht"
[master bff926a] aenderungsnachricht
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 bar
```



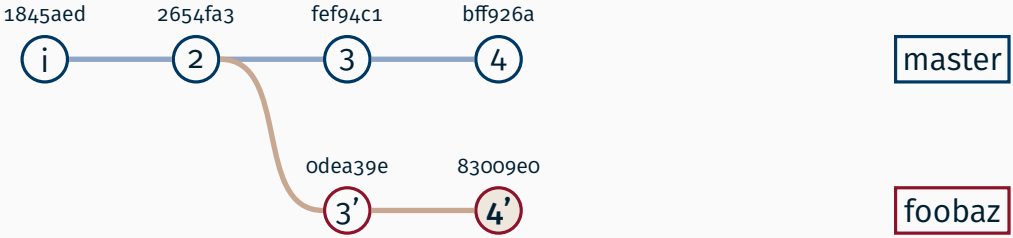
```
01 heinloth:~/beispiel$ git status
02 Auf Branch master
03 nichts zu committen, Arbeitsverzeichnis unverändert
```

```
heinloth:~/beispiel$ git checkout -b foobaz 2654fa3
Gewechselt zu einem neuem Branch 'foobaz'
heinloth:~/beispiel$ git shortlog
Bernhard Heinloth (2):
 initialer commit
 hier sollte eine gute nachricht stehen
```

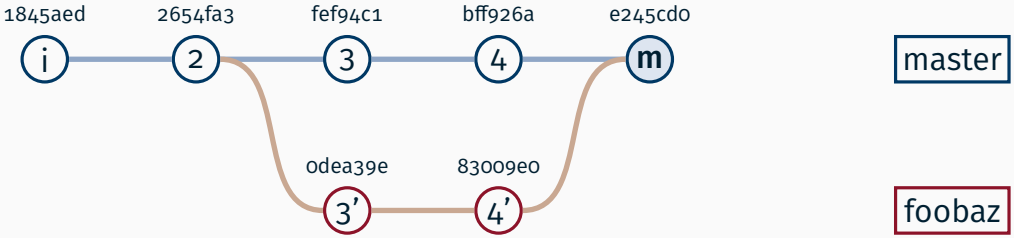
```
heinloth:~/beispiel$ echo "3.141" > baz
heinloth:~/beispiel$ git add .
heinloth:~/beispiel$ git commit -m "neue datei"
[foobaz 0dea39e] neue datei
 1 file changed, 1 insertion(+)
 create mode 100644 baz
```



```
heinloth:~/beispiel$ echo "pi" > baz
heinloth:~/beispiel$ git commit -am "blub"
[foobaz 83009e0] blub
 1 file changed, 1 insertion(+), 1 deletion(-)
```

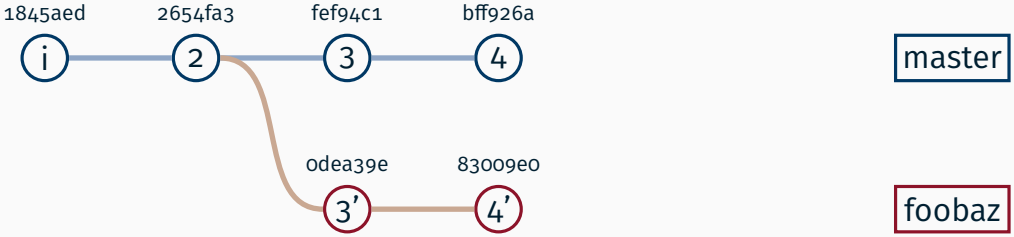


```
heinloth:~/beispiel$ git checkout master
heinloth:~/beispiel$ git merge foobaz -m "merge commit"
Merge made by the 'recursive' strategy.
 baz | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 baz
```



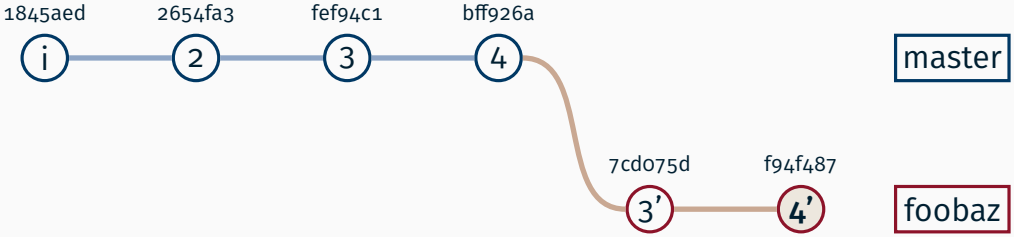
Alternativ: Die Geschichte neu schreiben

```
heinloth:~/beispiel$ git rebase master
Zunächst wird der Branch zurückgespult,
um Ihre Änderungen darauf neu anzuwenden...
 Wende an: blub
 Wende an: neue datei
```



Alternativ: Die Geschichte neu schreiben

```
heinloth:~/beispiel$ git rebase master
Zunächst wird der Branch zurückgespult,
um Ihre Änderungen darauf neu anzuwenden...
 Wende an: blub
 Wende an: neue datei
```

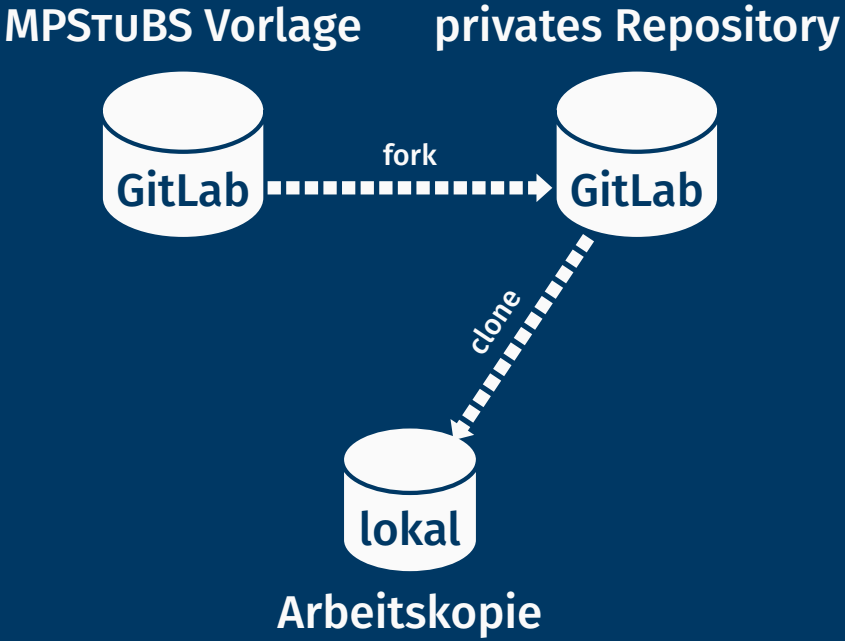


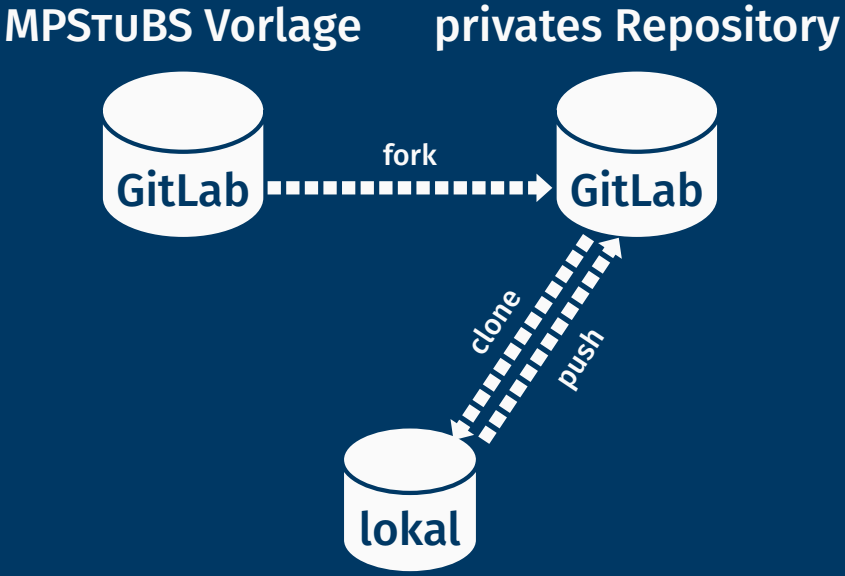
Workflow

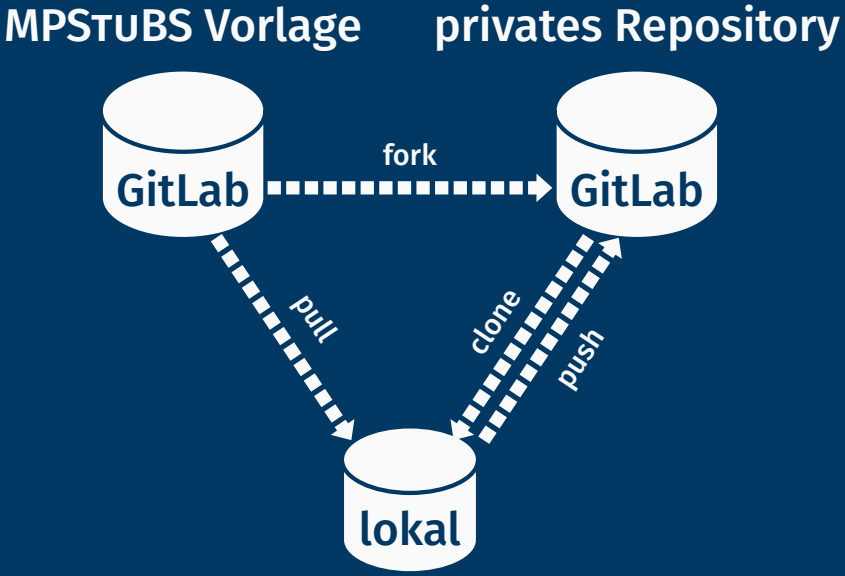
MPStuBS Vorlage

MPStuBS Vorlage

privates Repository


MPStuBS Vorlage
privates Repository
GitLab
fork
GitLab
clone
lokal
Arbeitskopie


MPStuBS Vorlage
privates Repository
GitLab
fork
GitLab
clone
push
lokal
Arbeitskopie


MPStuBS Vorlage
privates Repository
GitLab
fork
GitLab
pull
clone
push
lokal
Arbeitskopie

Cheatsheet

git init	neues Repository im aktuellen Verzeichnis erstellen
git add *Datei*	Datei als Kandidat für den nächsten *commit* markieren
git commit	Änderungen versionieren
git diff	unversionierte Änderungen anzeigen
git show	neuste (versionierte) Änderungen anzeigen
git status	Änderungen zum Vorgänger anzeigen
git branch	verfügbare Zweige anzeigen
git log	Historie anzeigen
man git-*Option*	Hilfe anzeigen, z.B. `man git-add`

Cheatsheet (entfernte Quellen)

git clone *URL*	initiales Kopieren von einer Quelle
git fetch *Name*	Änderungen aus entfernter Quelle holen
git pull *Name*	kurz für holen und zusammenfügen
git checkout *Zweig*	Aktuellen Zweig wechseln
git remote add *URL*	entfernte Quellen hinzufügen
git push *Name*	in entfernte Quelle übertragen

C++ Crashkurs

From: Linus Torvalds
Subject: Re: Compiling C++ kernel module + Makefile
Date: Mon, 19 Jan 2004 22:46:23 -0800 (PST)

On Tue, 20 Jan 2004, Robin Rosenberg wrote:
>
> This is the "We've always used COBOL^H^H^H^H" argument.

In fact, in Linux we did try C++ once already, back in 1992.

It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.

The fact is, C++ compilers are not trustworthy. They were even worse in
1992, but some fundamental facts haven't changed:

 - the whole C++ exception handling thing is fundamentally broken. It's
   especially broken for kernels.
 - any compiler or language that likes to hide things like memory
   allocations behind your back just isn't a good choice for a kernel.
 - you can write object-oriented code (useful for filesystems etc) in C,
   without the crap that is C++.

In general, I'd say that anybody who designs his kernel modules for C++ is
either
 (a) looking for problems
 (b) a C++ bigot that can't see what he is writing is really just C anyway
 (c) was given an assignment in CS class to do so.

Feel free to make up (d).

        Linus

C++: As Close as Possible to C, but no Closer

Andrew Koenig and Bjarne Stroustrup, The C++ Report. July 1989

einfaches ANSI C ist (fast) valides C++

```cpp
#include <cstdio>


int main() {
    const char * str = "Informatik";
    int n = 4;
    printf("%s %d\n", str, n);
    return 0;
}
```

```cpp
#include <iostream>


int main() {
    const char * str = "Informatik";
    int n = 4;
    std::cout << str << ' ' << n << std::endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main() {
    const char * str = "Informatik";
    int n = 4;
    cout << str << ' ' << n << endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main() {
    const string str = "Informatik";
    int n = 4;
    cout << str << ' ' << n << endl;
    return 0;
}
```

Namensräume

```
namespace Foo {
    int getNum() {
        return 23;
    }
}

namespace Bar {
    int getNum() {
        return 42;
    }
}

std::cout << Foo::getNum() << std::endl
          << Bar::getNum() << std::endl;
```

Referenzen

```
int foo = 23;
int& bar = foo;
std::cout << bar << std::endl;

bar = 42;
std::cout << foo << std::endl;
```

Referenzen

```
int foo = 23;
int& bar = foo;
std::cout << bar << std::endl;

bar = 42;
std::cout << foo << std::endl;
```

Unterschied zu Zeiger

- Initialisierung bei Definition
- kann nicht geändert werden
- kann nicht `NULL` sein

Referenzen als Funktionsparameter

```
void inc(int& i) {
    i++;
}

int foo = 23;
inc(foo);

std::cout << foo << std::endl;
```

Defaultparameter

```
void inc(int& i, int n = 1) {
    i += n;
}

int foo = 23;
inc(foo);
inc(foo, 2);
std::cout << foo << std::endl;
```

Überladen von Funktionen

```
bool isZero(int i){
    return i == 0;
}

bool isZero(double i){
    const double eps = 0.00001;
    return i < eps && i > -eps;
}
```

Strukturen


```
struct Disp {
    char val;
    int x, y;
};
```

```
struct Disp {
    char val;
    int x, y;
};
```

```
struct Disp p;
p.val = 'X';
p.x = 2;
p.y = 0;
std::cout << p.val << std::endl;
```

```
struct Disp {
    char val;
    int x, y;

    Disp() {
       val = 'X';
       this->x = 2;
       this->y = 0;
    }
};



Disp p;
std::cout << p.val << std::endl;
```

```
struct Disp {
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0) {
       val = c;
       this->x = x;
       this->y = y;
    }
};



Disp p('X', 2);
std::cout << p.val << std::endl;
```

```
struct Disp {
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0) : val(c),
        x(x), y(y) {}
};
```

```
Disp p('X', 2);
std::cout << p.val << std::endl;
```

```
struct Disp {
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0) : val(c),
        x(x), y(y) {}

    Disp(int p) : val('X'),
        x(p % 80), y(p / 80) {}
};



Disp p(2);
std::cout << p.val << std::endl;
```

```
int count = 0;
struct Disp {
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0) : val(c),
        x(x), y(y) { count++; }

    Disp(int p) : val('X'),
        x(p % 80), y(p / 80) { count++; }

    ~Disp() { count--; }
};

Disp p(2);
std::cout << p.val << "-" << count << std::endl;
```

```
struct Disp {
    static int count;
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0) : val(c),
        x(x), y(y) { count++; }

    Disp(int p) : val('X'),
        x(p % 80), y(p / 80) { count++; }

    ~Disp() { count--; }
};
int Disp::count = 0;
Disp p(2);
std::cout << p.val << "-" << Disp::count << std::endl;
```

```
struct Disp {
    static int count;
    char val;
    int x, y;

    Disp(char c, int x = 0, int y = 0);
    Disp(int p);
    ~Disp();
};

int Disp::count = 0;

Disp::Disp(char c, int x, int y) : val(c), x(x), y(y) {
    count++;
}
// ebenso Disp::Disp(int p),  Disp::~Disp()
```

```
struct Disp {
  private:
    static int count;
    char val;
    int x, y;

  public:
    Disp(char c, int x = 0, int y = 0);
    Disp(int p);
    ~Disp();
};

// Übersetzerfehler bei:
std::cout << p.val << "-" << Disp::count << std::endl;
```

```
class Disp {
  private:
    static int count;
    char val;
    int x, y;

  public:
    Disp(char c, int x = 0, int y = 0);
    Disp(int p);
    ~Disp();
};
```

```
class Disp {
    static int count;
    char val;
    int x, y;

  public:
    Disp(char c, int x = 0, int y = 0);
    Disp(int p);
    ~Disp();
};
```

Unterschied Standardsichtbarkeit

private bei class

public bei struct (und union)

Vererbung

Syntax: `class Abgeleitet: Vererbungsart Basis`

Vererbung

Syntax: `class Abgeleitet: Vererbungsart Basis`

```
class Foo {
    int n;
  protected:
    int f1(int i){
        return i * n;
    }
};
```

Vererbung

Syntax: `class` **Abgeleitet:** Vererbungsart **Basis**

```
class Foo {
    int n;
  protected:
    int f1(int i){
        return i * n;
    }
};
```

```
class Bar : Foo {
  public:
    int n;  // eigene Var
    int f2(int i){
        return f1(i) * n;
    }
};
```

Vererbung

Syntax: `class` **`Abgeleitet:`** `Vererbungsart` **`Basis`**

```
class Foo {
    int n;
  protected:
    int f1(int i){
        return i * n;
    }
};
```

```
class Bar : Foo {
  public:
    int n;  // eigene Var
    int f2(int i){
        return f1(i) * n;
    }
};
```

Vererbungsart	Elemente aus **Basis**
`public`	`public` und `protected` bleiben
`protected`	`public` und `protected` werden zu `protected`
`private`	`public` und `protected` werden zu `private`

Ohne Angabe der Vererbungsart wird `private` verwendet.

Vererbung

Syntax: `class` **`Abgeleitet:`** `Vererbungsart` **`Basis`**

```
class Foo {
    int n;
  protected:
    int f1(int i){
        return i * n;
    }
};
```

```
class Bar : Foo {
  public:
    int n;  // eigene Var
    int f2(int i){
        return f1(i) * n;
    }
};
```

Vererbungsart	Elemente aus **Basis**
`public`	`public` und `protected` bleiben
`protected`	`public` und `protected` werden zu `protected`
`private`	`public` und `protected` werden zu `private`

Ohne Angabe der Vererbungsart wird `private` verwendet.

`private` Elemente können **nie** von außen erreicht werden!

Mehrfachvererbung

```
class FooBaz: public Foo, protected Baz
{
    // ...
}
```

Virtuelle Methoden

```
class Foo {
  public:
    void f1() { cout << "Foo::f1" << endl; };
    virtual void f2() { cout << "Foo::f2" << endl; };
    virtual void f3() = 0;
};
class Bar : public Foo {
  public:
    void f2() { cout << "Bar::f2" << endl; }
    void f3() { cout << "Bar::f3" << endl; }
};
class Baz : public Foo {
  public:
    void f3() { cout << "Baz::f3" << endl; }
};
```

Virtuelle Methoden

```
class Foo {
  public:
    void f1() {...};
    virtual void f2() {...};
    virtual void f3() = 0;
};
class Bar : public Foo {
  public:
    void f2() {...}
    void f3() {...}
};
class Baz : public Foo {
  public:
    void f3() {...}
};
```

```
Foo foo;
// Nicht erlaubt
// Übersetzerfehler
```

Virtuelle Methoden

```
class Foo {
  public:
    void f1() {...};
    virtual void f2() {...};
    virtual void f3() = 0;
};
class Bar : public Foo {
  public:
    void f2() {...}
    void f3() {...}
};
class Baz : public Foo {
  public:
    void f3() {...}
};
```

```
Bar bar;
bar.f1();
// "Foo::f1"
bar.f2();
// "Bar::f2"
bar.f3();
// "Bar::f3"
```

Virtuelle Methoden

```
class Foo {
  public:
    void f1() {...};
    virtual void f2() {...};
    virtual void f3() = 0;
};
class Bar : public Foo {
  public:
    void f2() {...}
    void f3() {...}
};
class Baz : public Foo {
  public:
    void f3() {...}
};
```

```
Bar bar;
bar.f1();
// "Foo::f1"
bar.f2();
// "Bar::f2"
bar.f3();
// "Bar::f3"

Baz baz;
baz.f1();
// "Foo::f1"
baz.f2();
// "Foo::f2"
baz.f3();
// "Baz::f3"
```

Virtuelle Methoden

```
class Foo {
  public:
    void f1() {...};
    virtual void f2() {...};
    virtual void f3() = 0;
};
class Bar : public Foo {
  public:
    void f2() {...}
    void f3() {...}
};
class Baz : public Foo {
  public:
    void f3() {...}
};
```

```
Foo * foo = &bar;
foo->f1();
// "Foo::f1"
foo->f2();
// "Bar::f2"
foo->f3();
// "Bar::f3"

foo = &baz;
foo->f1();
// "Foo::f1"
foo->f2();
// "Foo::f2"
foo->f3();
// "Baz::f3"
```

Operatorüberladung

```
class WeirdInt {
  public:
    int val;
    WeirdInt(int i) : val(i) {}

    WeirdInt operator+(const WeirdInt& other) {
        return val - other.val;
    }
};
WeirdInt operator-(WeirdInt l, WeirdInt r) {
    return l.val + r.val;
}

WeirdInt i = 5;
WeirdInt n = i + 3 - 1;
std::cout << n.val << std::endl;  // Ausgabe 3
```

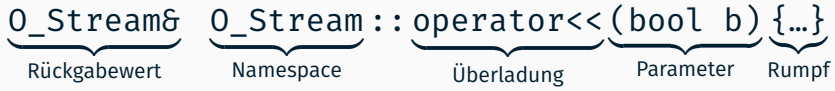
Operatorüberladung (sinnvoll)

Streamoperatoren (z.B. `cout`, abgeleitet von `ostream`)

Operatorüberladung (sinnvoll)

Streamoperatoren (z.B. `cout`, abgeleitet von `ostream`)

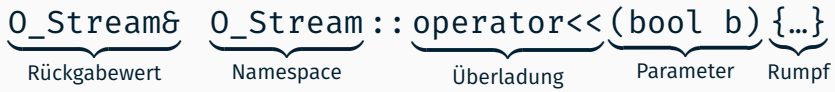
Beispiel `O_Stream`:



Operatorüberladung (sinnvoll)

Streamoperatoren (z.B. `cout`, abgeleitet von `ostream`)

Beispiel `O_Stream`:



Manipulatoren:

```
O_Stream& f(O_Stream&){…}
```

Fragen?