

Übung zu Betriebssysteme

Organisation und Einführung

19. & 22. Oktober 2018

Andreas Ziegler
Bernhard Heinloth

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

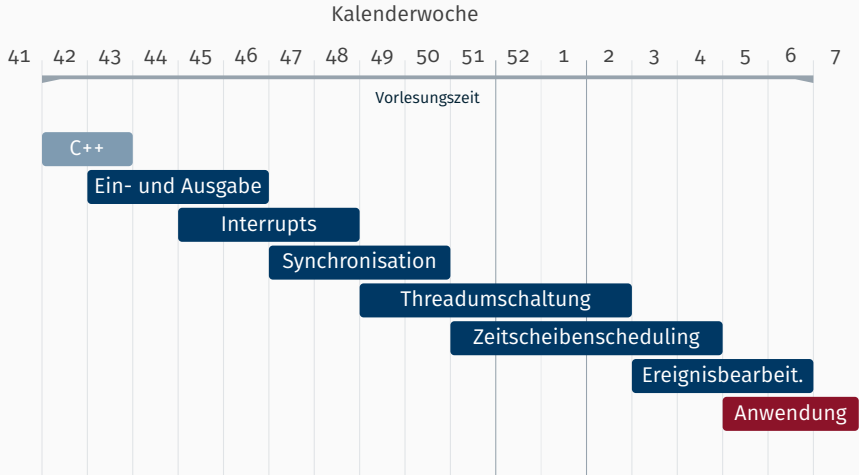
OOSTuBS
single-core
5 ECTS Modul



MPSTuBS
multi-core
7.5 ECTS Modul



Zeitplan (Übungsaufgaben)



Zeitplan (Woche)



Zeitplan (Semester)

Oktober 2018

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 2018

		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 2018

						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 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			

Februar 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		

Vorlesung
im Aquarium

Tafelübung
für neue Aufgabe
im Aquarium

Abgabe der Aufgabe in
der Rechnerübung
im Huber-CIP

- 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=356**
- Informationen und Aufgabenstellung auf **www4.cs.fau.de/Lehre/WS18/V_BS/**
- Quelltextvorlagen der Aufgaben im Gitlab
 - <https://gitlab.cs.fau.de/i4/bs/oostubs>
 - <https://gitlab.cs.fau.de/i4/bs/mpstubs>

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.

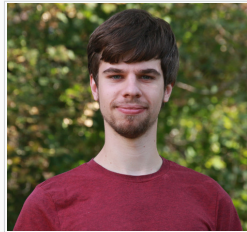
**Sie sind nicht zum Selbststudium geeignet
und auch nicht auf Druck optimiert!**



Harald Böhm



Judith Hemp



Jakob Wagner

- Rechnerübung
- Tafelübung
- Mail an **bsstud@lists.informatik.uni-erlangen.de**
- **#faii4bs** im IRCnet
- XMPP-MUC **i4bs@conference.cs.fau.de**
- **Raum 0.055** in der Martensstr. 1 (begründeter Notfall)

Eine kleine Einführung in Versionsverwaltungssysteme



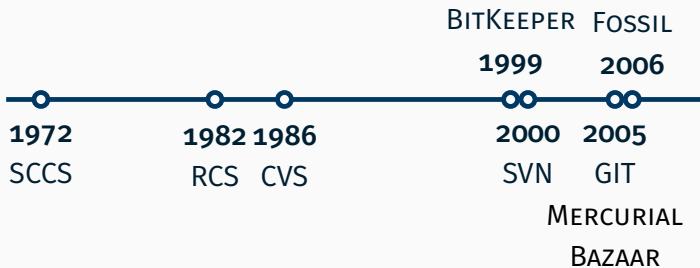
1972
SCCS

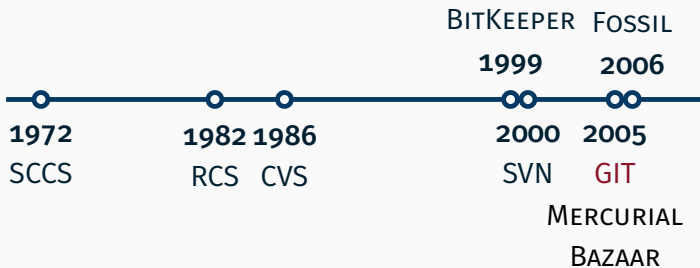










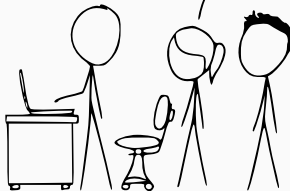


- Integrität durch SHA-1 Hash
- vollständiges Speichern der Daten (*snapshot*)
- dezentral (*clone*)
- nicht-lineare Entwicklung (*branch*)

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.



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

```
01 heinloth:~/beispiel$ touch README.md
02 heinloth:~/beispiel$ git add README.md
03 heinloth:~/beispiel$ git commit -m "initialer commit"
04 [master (Basis-Commit) 1845aed] initialer commit
05 1 file changed, 0 insertions(+), 0 deletions(-)
06 create mode 100644 README.md
```

1845aed



```
01 heinloth:~/beispiel$ echo "23" > foo
02 heinloth:~/beispiel$ git add foo
03 heinloth:~/beispiel$ git commit -m "hier sollte eine
    gute nachricht stehen"
04 [master 2654fa3] hier sollte eine gute nachricht stehen
05 1 file changed, 1 insertion(+)
06 create mode 100644 foo
```

1845aed 2654fa3



```
01 heinloth:~/beispiel$ echo "42" > foo
02 heinloth:~/beispiel$ git add foo
03 heinloth:~/beispiel$ git commit -m "noch mal ein commit"
04 [master fef94c1] noch mal ein commit
05 1 file changed, 1 insertion(+), 1 deletion(-)
```



```
01 heinloth:~/beispiel$ echo "1337" > bar
02 heinloth:~/beispiel$ echo "zweiundvierzig" > foo
03 heinloth:~/beispiel$ git add bar foo
04 heinloth:~/beispiel$ git commit -m "aenderungsnachricht"
05 [master bff926a] aenderungsnachricht
06 2 files changed, 2 insertions(+), 1 deletion(-)
07 create mode 100644 bar
```



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



master

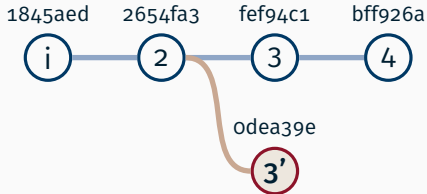
```
01 heinloth:~/beispiel$ git checkout -b foobaz 2654fa3
02 Gewechselt zu einem neuem Branch 'foobaz'
03 heinloth:~/beispiel$ git shortlog
04 Bernhard Heinloth (2):
05   initialer commit
06   hier sollte eine gute nachricht stehen
```



master

foobaz

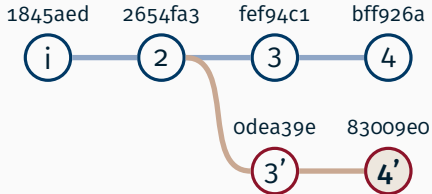
```
01 heinloth:~/beispiel$ echo "3.141" > baz
02 heinloth:~/beispiel$ git add .
03 heinloth:~/beispiel$ git commit -m "neue datei"
04 [foobaz 0dea39e] neue datei
05 1 file changed, 1 insertion(+)
06 create mode 100644 baz
```



master

foobaz

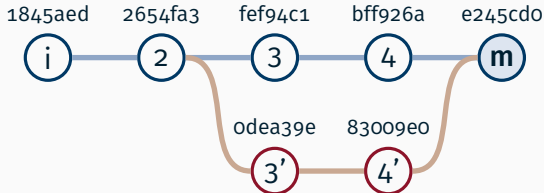
```
01 heinloth:~/beispiel$ echo "pi" > baz
02 heinloth:~/beispiel$ git commit -am "blub"
03 [foobaz 83009e0] blub
04 1 file changed, 1 insertion(+), 1 deletion(-)
```



master

foobaz

```
01 heinloth:~/beispiel$ git checkout master
02 heinloth:~/beispiel$ git merge foobaz -m "merge commit"
03 Merge made by the 'recursive' strategy.
04   baz | 1 +
05   1 file changed, 1 insertion(+)
06   create mode 100644 baz
```



Alternativ: Die Geschichte neu schreiben

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

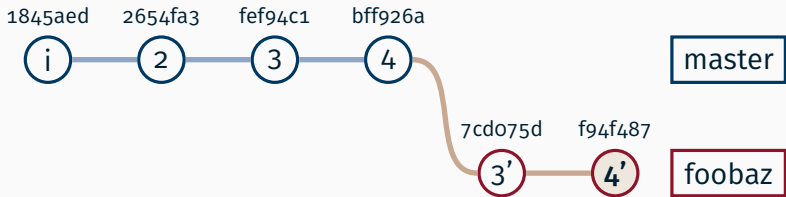


master

foobaz

Alternativ: Die Geschichte neu schreiben

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



Workflow

MPSTuBS Vorlage



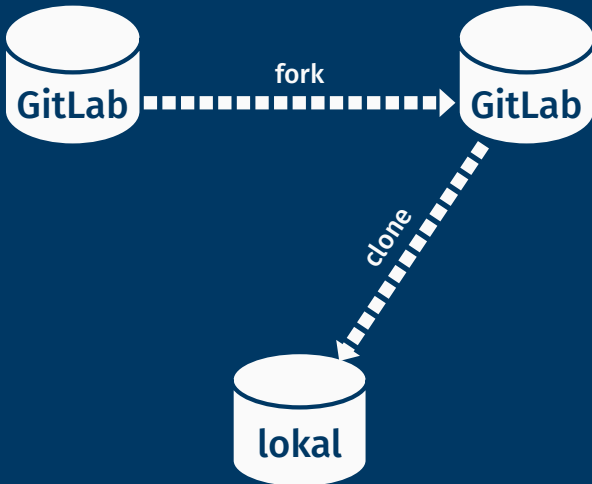
MPSTuBS Vorlage

privates Repository



MPStuBS Vorlage

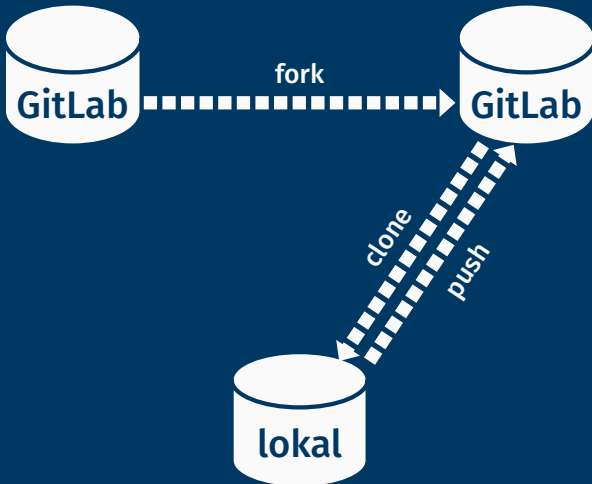
privates Repository



Arbeitskopie

MPStuBS Vorlage

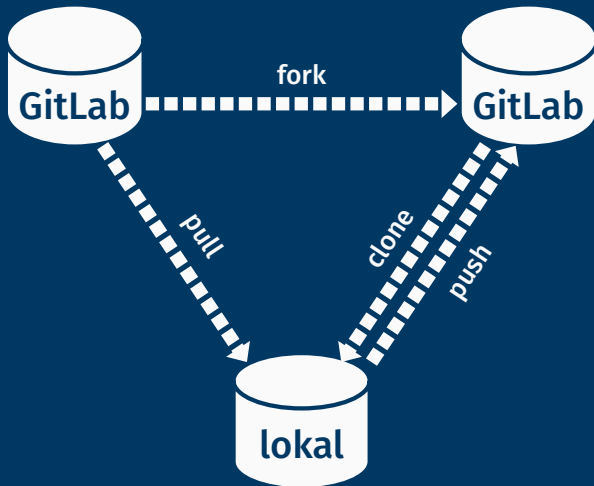
privates Repository



Arbeitskopie

MPStuBS Vorlage

privates Repository



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" 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++

```
#include <stdio>
```

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

```
#include <iostream>
```

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

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

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

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

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

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;  
}
```

```
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`

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!

```
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; };
};
```

```
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
```

```
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=(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:

O_Stream& O_Stream :: operator<< (bool b) {...}
Rückgabewert Namespace Überladung Parameter Rumpf

Operatorüberladung (sinnvoll)

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

Beispiel O_Stream:

O_Stream& O_Stream :: operator<< (bool b) {...}
Rückgabewert Namespace Überladung Parameter Rumpf

Manipulatoren:

O_Stream& f(O_Stream&){...}

Fragen?