

# Programmierkurs

Steffen Müthing

Interdisciplinary Center for Scientific Computing, Heidelberg University

October 15, 2018

Organisation

Bestandsaufnahme

Unix-Einführung

Wichtige Befehle

Grundlegendes zu Ein- und Ausgabe

# Inhalte

- ▶ Praktisches Programmieren mit C++
- ▶ Umgang mit Unix-Kommandozeile (Linux, macOS)
- ▶ Vorstellung von wichtigen Themen und Werkzeugen rund ums Programmieren
  - ▶ Versionsverwaltung
  - ▶ Buildsysteme
  - ▶ Umgang mit Dokumentation
  - ▶ Fehlersuche (Debugging)

# Organisation

## Termine

**Vorlesung** Freitag, 14-16 ct  
INF 252 Hörsaal Ost

**Übungen** Montag, 11-13 und 14-16 ct  
INF 205 Computer Pool 3. Stock

**Klausur** TBA

## Informationen, Übungsblätter etc.

[conan.iwr.uni-heidelberg.de/teaching/ips\\_ws2018/](http://conan.iwr.uni-heidelberg.de/teaching/ips_ws2018/)

## Bei Fragen

Email: [ipk-ws2018@conan.iwr.uni-heidelberg.de](mailto:ipk-ws2018@conan.iwr.uni-heidelberg.de)

# Übungen

- ▶ Um Programmieren zu lernen, muss man programmieren!
- ▶ Übungszettel bestehen ausschliesslich aus Programmieraufgaben
- ▶ Neue Übungszettel freitags auf Homepage
- ▶ Nächster Montag: Bearbeiten des Zettels, Fragen an Tutoren / Dozent zum neuen Blatt
- ▶ Darauffolgender Montag: Vorstellung der Lösungen
- ▶ Votiersystem statt Abgabe der Lösungen
- ▶ Anmeldung über MUESLI, Link auf Homepage wird nach der Vorlesung freigeschaltet

# Votiersystem

- ▶ Am Anfang der Übung tragen Sie auf einem Zettel ein, welche Aufgaben Sie vorstellen möchten
- ▶ Die Tutoren kommen zu Ihnen an den Computer und lassen sich Ihre Lösung zeigen
- ▶ Ihr Programm muss nicht perfekt sein, aber der Tutor muss zumindest einen Lösungsansatz erkennen
- ▶ Falls Sie ein Programm ankreuzen und keine Lösung haben  
⇒ Aberkennung aller Punkte für das Blatt
- ▶ Falls Sie das Programm vom einem Freund kopiert haben und dem Tutor nicht erklären können  
⇒ Aberkennung aller Punkte für das Blatt
- ▶ Für die Klausurzulassung sind 50% der Punkte erforderlich

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?



# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#? C?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#? C? C++?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#? C? C++? Python?

# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#? C? C++? Python?
- ▶ Wer versteht folgendes Shell-Kommando?

```
g++ -Wall -O3 test.cc | grep "error:|warning:" > log.txt
```



# Bestandsaufnahme

- ▶ Wer hat schon einmal Linux / macOS verwendet?
- ▶ Wer hat schon einmal die Kommandozeile verwendet?
- ▶ Wer hat schon einmal programmiert?  
Javascript? Java? C#? C? C++? Python?
- ▶ Wer versteht folgendes Shell-Kommando?

```
g++ -Wall -O3 test.cc | grep "error:|warning:" > log.txt
```

- ▶ Wer versteht folgenden C++-Code?

```
auto v = std::array<int>{1,2,3,4};  
for (auto& x : v)  
    x *= x;  
std::cout << std::accumulate(v.begin(), v.end(), 0)  
           << std::endl;
```

## Warum UNIX / Linux?

Die meisten Arbeitsgruppen in Mathematik, Physik und Informatik verwenden zumindest in Teilen Linux. Daher werden sich die meisten von Ihnen früher oder später damit auseinandersetzen müssen.

Linux ist mit Abstand das am häufigsten genutzte Unix. Über mit Linux betriebene Webserver werden nicht nur große Teile des Internet zur Verfügung gestellt, grob 95% der 500 **schnellsten Rechner der Welt** basieren auf Linux, und alle unter den zehn schnellsten. Wer mit Rechnern wissenschaftlich arbeiten will, kommt an Unix nicht vorbei.

Darüber hinaus ist Linux durch seine offene Natur und die vielen Open-Source-Komponenten eine hervorragende Spielwiese für angehende Informatiker. Für jedes Programm, das auf einem normalen Linux-Desktop installiert ist, können Sie den Quellcode herunterladen und lernen, wie es programmiert wurde!

# Was ist UNIX?

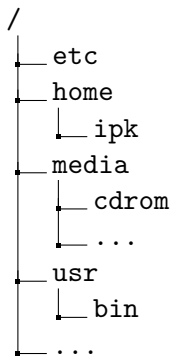
Im engeren Sinne [Unix \(1969\)](#), ein Betriebssystem, das viele Funktionen einführte, die in heutigen Betriebssystemen selbstverständlich sind.

Im weiteren Sinne jedes Betriebssystem, das sich an die UNIX-Interfaces und -Spezifikation hält:

- ▶ [Free,Net,OpenBSD](#), basierend auf einer Unix-Variante von der UC Berkeley
- ▶ [macOS](#), das auf Teilen von FreeBSD und NetBSD basiert
- ▶ [iOS](#), aus macOS hervorgegangen
- ▶ [illumos/OpenIndiana](#), basierend auf Solaris und System V
- ▶ [Linux \(1991\)](#), streng genommen kein UNIX, aber weitestgehend kompatibel
- ▶ [Android](#), von Google stark modifiziertes Linux
- ▶ Viele [embedded systems](#), oft auf Basis von BSD oder Linux, in Netzwerkroutern, Fernsehern, Robotern, Autos, Raketen, ...

# Dateisystem I

- ▶ Dateien liegen in Verzeichnissen
- ▶ Verzeichnistrenner unter Unix:  
home/user statt C:\Users\Name
- ▶ Gross- und Kleinschreibung unter Linux relevant:  
test.txt  $\neq$  Test.txt
- ▶ Linux kennt keine Laufwerksbuchstaben, alle Verzeichnisse haben eine gemeinsame Wurzel /



## Dateisystem II

- ▶ Jeder Benutzer hat ein **home directory** für eigene Dateien, normalerweise in `/home/<username>`
- ▶ Normale Benutzer haben keine Schreibrechte ausserhalb ihres home directories
- ▶ Der Administrator (heisst unter UNIX **root**) darf überall lesen und schreiben
- ▶ Jedes Programm hat ein **Arbeitsverzeichnis (working directory)**
- ▶ Dateizugriffe immer relativ zu diesem Verzeichnis
- ▶ Arbeitsverzeichnis kann gewechselt werden
- ▶ Spezielle Verzeichnis-Namen
  - . Das aktuelle Verzeichnis
  - .. Das übergeordnete Verzeichnis
  - / Das Wurzel-Verzeichnis
  - ~ Das **Home Directory** des aktuellen Benutzers (funktioniert nicht überall)

# Shell I

Linux lässt sich im Alltag heutzutage problemlos graphisch bedienen und verhält sich dann in weiten Teilen wie die bekannten Desktops von Windows und macOS. Dabei hat man die Wahl zwischen verschiedenen Oberflächen, die Windows nachahmen, Mac OS nachahmen oder gänzlich neue Wege gehen.

Für fortgeschrittene Aufgaben wie Programmieren ist es jedoch weiterhin sinnvoll, sich mit dem schwarz-weißen Terminal-Fenster und der darin laufenden **Shell** zu befassen:

- ▶ Automatisierung von monotonen und repetitiven Arbeiten
- ▶ Mit etwas Übung ist man bei vielen Aufgaben schneller als in der GUI
- ▶ Viele UNIX-Programme haben keine graphische Oberfläche (GUI) und können direkt nur über die Shell aufgerufen werden, z.B. der C++-Compiler, den wir zum Programmieren in diesem Kurs brauchen

## Shell II

```
[ipk@vm ~] _
```

- ▶ Wichtige Informationen am Anfang der Zeile (**prompt**)
  - ▶ Benutzername
  - ▶ Rechnername
  - ▶ aktuelles Verzeichnis
- ▶ Ausgabe des vollen Pfades mit **pwd**:

```
[ipk@vm ~] pwd  
/home/ipk/ipk-files
```

- ▶ Shell beenden mit **exit**:

```
[ipk@vm ~] exit  
<Fenster schliesst sich>
```

# Kommandos in der Shell

```
[ipk@vm ~] cmd -sv --opt --op2 arg1 arg2 ...
```

- ▶ Die meisten Kommandos haben ein einheitliches Interface
  - ▶ Am Anfang steht der Name des Kommandos (der Dateiname des Programms)
  - ▶ Danach folgen Optionen (beginnen mit "--")
  - ▶ Am Ende stehen die Argumente ohne "--"
- ▶ Argumente bestimmen, worauf das Kommando angewendet wird
- ▶ Optionen verändern, wie das Kommando arbeitet. Wichtige Optionen haben oft einen langen Namen und eine Abkürzung aus einem Buchstaben
  - ▶ Lange Optionen beginnen mit "----"
  - ▶ Kurze Optionen beginnen mit "--" und können gruppiert werden, z.B. "--rf"



## Hilfe zu Kommandos

- ▶ Die meisten Kommandos geben eine kurze Übersicht der erlaubten Optionen und Argumente aus, wenn man sie falsch benutzt:

```
[ipk@vm ~] rm
usage: rm [-f | -i] [-dPRrvW] file ...
        unlink file
```

- ▶ Fast alle Kommandos geben mit der Option "**--help**" einen Hilfetext aufs Terminal aus
- ▶ Für genauere Informationen gibt es die [man pages](#), die man mit dem Befehl `man` aufruft

```
[ipk@vm ~] man gcc
```

- ▶ In der man page bewegt man sich mit den Pfeiltasten und verlässt sie mit der Taste "**q**"

## Verzeichnis wechseln I

- ▶ Verzeichnis wechseln mit `cd <name>` (**change directory**):

```
[ipk@vm ~] cd Documents  
[ipk@vm Documents]
```

- ▶ Neues Verzeichnis wird im **prompt** angezeigt
- ▶ `cd` erzeugt UNIX-typisch nur bei Fehlern eine Ausgabe

```
[ipk@vm Documents] cd nonesuch  
-bash: cd: nonesuch: No such file or directory  
[ipk@vm Documents]
```

- ▶ `cd ..` kehrt ins übergeordnete Verzeichnis zurück

```
[ipk@vm Documents] cd ..  
[ipk@vm ~]
```

## Verzeichnis wechseln II

- ▶ `cd` unterstützt auch zusammengesetzte Pfade

```
[ipk@vm ~] cd Documents/c++  
[ipk@vm c++]
```

- ▶ Man kann auch `absolute` Pfade verwenden, die mit `/` beginnen und unabhängig vom aktuellen Verzeichnis sind:

```
[ipk@vm c++] cd /etc/sysconfig  
[ipk@vm sysconfig]
```

- ▶ `cd` ohne Argumente wechselt immer ins home directory

```
[ipk@vm sysconfig] cd  
[ipk@vm ~]
```

## Dateien auflisten

- ▶ `ls` (`list`) zeigt die Dateien im aktuellen Verzeichnis an

```
[ipk@vm c++] ls
helloworld    helloworld.cc
```

- ▶ Dateien, die mit einem Punkt beginnen, werden normalerweise nicht angezeigt. Dies kann man mit der `Option` `-a` ändern:

```
[ipk@vm c++] ls -a
.              ..              .versteckt
helloworld    helloworld.cc
```

- ▶ Auch `ls` akzeptiert einen Pfad als Argument

```
[ipk@vm c++] ls ~/Documents
c++
```

- ▶ `ls -l` (`long`) zeigt zusätzliche Informationen wie Besitzer, Zugriffsrechte, Dateigröße etc. an

## Dateien kopieren und verschieben

- ▶ `cp` (**copy**) kopiert, `mv` (**move**) verschiebt Dateien

```
[ipk@vm c++] cp original copy
[ipk@vm c++] ls
copy      original
```

- ▶ Man kann auch mehrere Dateien gleichzeitig kopieren. In diesem Fall muss das Ziel ein Verzeichnis sein

```
[ipk@vm c++] mkdir subdir
[ipk@vm c++] mv original copy subdir
[ipk@vm c++] ls
subdir
[ipk@vm c++] ls subdir
copy      original
```

- ▶ Mit der Option `-r` (**recursive**) kopiert man Unterverzeichnisse samt Inhalt, beim Verschieben ist diese Option nicht erforderlich.

## Dateien löschen

- ▶ `rm` (**remove**) löscht Dateien und Verzeichnisse

```
[ipk@vm c++] rm subdir/copy
[ipk@vm c++] ls subdir
original
```

- ▶ Mit der Option `-r` (**recursive**) löscht `rm` ein Verzeichnis mit allen Inhalten

```
[ipk@vm c++] rm -r subdir
[ipk@vm c++] ls
```

### Warnung

`rm` fragt nicht nach, bevor die Dateien gelöscht werden, und in der Shell gibt es keinen Papierkorb! Gelöschte Dateien können nicht wiederhergestellt werden!

## Dateien bearbeiten

- ▶ Es gibt Editoren, die im Textmodus im Terminal arbeiten (vim, nano, emacs,...), aber wir werden in dieser Vorlesung Editoren mit GUI verwenden
- ▶ Im Pool sind die Editoren gedit und kate installiert, in der VM geany und qtcreator
- ▶ Sie können den Editor entweder wie gewohnt per Doppelklick auf eine Datei im Dateimanager starten oder direkt über die Shell

```
[ipk@vm c++] geany helloworld.cc &  
[ipk@vm c++]
```

- ▶ Beim Starten von GUI-Programmen in der Shell ist es sinnvoll, ein "&" ans Ende des Befehls zu setzen. Ansonsten ist die Shell blockiert, bis Sie das gestartete Programm wieder beenden

## Dateien anzeigen

- ▶ `cat` zeigt den Inhalt von Dateien im Terminal an

```
[ipk@vm c++] ls
file1 file2
[ipk@vm c++] cat file2 file1
line in file2
line in file1
another line in file1
```

- ▶ `cat` kann bei langen Dateien unübersichtlich werden. `less` öffnet die Dateien in einem Viewer ähnlich zu dem für die man pages

```
[ipk@vm c++] less file1
```

- ▶ Navigieren mit Pfeiltasten und `"q"` zum Beenden
- ▶ Leertaste, um einen Bildschirm weiter zu springen
- ▶ `"/"` + Suchwort + Enter, um zu suchen
- ▶ `"n"`, um zum nächsten Vorkommen des Suchworts zu springen



# Dateien durchsuchen

- ▶ `grep <pattern> <datei>...` durchsucht Dateien nach einem Pattern

```
[ipk@vm c++] grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
10:operator:x:11:0:operator:/root:/sbin/nologin
```

- ▶ Das Pattern kann ein einfaches Wort sein, man kann aber auch nach komplizierte Ausdrücke mit Hilfe sogenannter **Regular Expressions** konstruieren

# I/O Streams

- ▶ UNIX-Programme kommunizieren mit dem OS über sogenannte I/O (Input/Output) **streams**
- ▶ Streams sind eine Einbahnstrasse - man kann aus ihnen entweder lesen oder Daten in sie schreiben
- ▶ Beim Start hat jedes Programm drei offene Streams
  - stdin** Die Standardeingabe liest User-Eingaben von der Konsole, ist verbunden mit **file descriptor 0**
  - stdout** An die Standardausgabe werden normale Ergebnisse des Programms ausgegeben, ist verbunden mit **file descriptor 1**
  - stderr** An die Standardfehlerausgabe werden diagnostische Meldungen wie Fehler ausgegeben, ist verbunden mit **file descriptor 2**

## Umleiten von I/O Streams I

- ▶ Normalerweise sind alle Standardstreams mit dem Terminal verbunden
- ▶ Manchmal kann es sinnvoll sein, diese Streams in Dateien umzuleiten
- ▶ `stdout` wird mit `"> datei"` in einer Datei gespeichert

```
[ipk@vm ~] ls > files
[ipk@vm ~] cat files
file1
file2
files
```

Die Ausgabedatei wird angelegt, bevor der Befehl ausgeführt wird

- ▶ Fehlermeldungen werden weiterhin im Terminal angezeigt

```
[ipk@vm ~] ls missingdir > files
ls: missingdir: No such file or directory
[ipk@vm ~] cat files
[ipk@vm ~]
```

# Umleiten von I/O Streams II

- ▶ `stdin` wird mit "`< datei`" aus einer Datei gelesen

```
[ipk@vm ~] cat # ohne Argument gibt cat stdin nach stdout aus
Eingabe am Terminal^D # (CTRL+D) beendet die Eingabe
Eingabe am Terminal
[ipk@vm ~] cat < files
file1
file2
files
```

- ▶ `stderr` wird mit "`2> datei`" in einer Datei gespeichert

```
[ipk@vm ~] ls missingdir 2> error
[ipk@vm ~] cat error
ls: missingdir: No such file or directory
```

# Kompilieren von C++-Programmen

- ▶ C++-Programme müssen vor dem Ausführen kompiliert werden
- ▶ Der Standard-C++-Compiler unter Linux heisst `g++`
- ▶ Wir verwenden in diesem Kurs stattdessen den Compiler `clang++` aus dem LLVM-Projekt, da dieser verständlichere Fehlermeldungen generiert
- ▶ Auf den Pool-Rechnern und in der VM rufen Sie den Compiler bitte als `ipkc++` auf. Dadurch werden einige sinnvolle Standardoptionen gesetzt
- ▶ Auf Ihrem eigenen Rechner sollten Sie immer die Option `-Wall` verwenden, damit der Compiler Sie bei Problemen warnt, die zwar legales C++ sind, aber wahrscheinlich von Ihnen so nicht gewollt sind