

# Einführung

## Sonderübung

Luise Zieger & Lea Happel

PROG Sonderübung

October 9, 2018

1. Ablauf
  - Verweise
2. Arbeiten mit Linux
  - Ubuntu
  - Shell/ Bash/ Command Prompt
  - Übungsaufgaben
  - Editor
3. Unser erstes FORTRAN-Programm
  - Hello World
  - Der GNU Fortran Compiler
4. Übungsaufgaben

# Über diesen Kurs

## Vorwissen

- Ihr habt bereits einen Computer benutzt
- Wir arbeiten an den Rechnern im PC-Pool (möglich ist natürlich auch die Benutzung des eigenen PCs)
- Eventuell habt ihr schon eine (andere) Programmiersprache kennengelernt

## Ablauf

- Insgesamt haben wir ca. 12 Stunden
- Jede Stunde behandelt ein Thema und liefert Übungen, um das neu gewonnene Wissen anzuwenden
- Fragen zu den aktuellen Programmieraufgaben können besprochen werden

# Einige Quellen

- Ihr könnt eure Tutoren fragen
- Opal-Kurs zur Vorlesung  
`https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/18562318336/CourseNode/76989518061579`
- Fortran 95 - Nachschlagewerk vom RRZN Hannover
- Material-Repository  
`https://gitlab.math.tu-dresden.de/wwalter/PROG-material-public`

# Ubuntu

- Linux-Distribution
- d. h. vollständiges Linux-basiertes (bzw. Debian-basiertes) Betriebssystem
- einfache Installation und Bedienung
- andere Betriebssysteme möglich, allerdings mitunter schwierig in Bezug auf den Compiler (OS X: gfortran native)
- Bedienung mittels shell/ bash

# Shell/ Bash/ Kommandozeile

## Hier: Verwendung der Bash

Was ihr bei Standardkonfiguration sehen werdet, sieht in etwa so aus:

`<username> @ <hostname> ~ $`

- `<username>` ist der Name, mit dem ihr auf dem Rechner angemeldet seid
- `<hostname>` ist der Name, wie sich der Computer selbst nennt
- Tilde (`~`) bezeichnet das aktuelle directory (Ordner)
- `~` steht für das home directory
- Zeile zur Befehlseingabe heißt prompt
- eingegebene Befehle werden mit Enter bestätigt, danach wird die Eingabe gelesen, interpretiert und ausgeführt

# Liste wichtiger Befehle

- Worte in eckigen Klammern (<>) sind Platzhalter für Parameter
- Syntax ist <befehl> <parameter>
- Eingabe von Parametern entweder wie file names oder mittels dash (–) vorneweg
- Leerzeichen zur Abtrennung von Befehlen und Parametern

# Liste wichtiger Befehle

|   |  |
|---|--|
| <code>man &lt;befehlsname&gt;</code>              | öffnet das Manual (Handbuch) zum entsprechenden Befehl (schließe mit <b>q</b> für <b>quit</b> )                  |
| <code>ls</code>                                   | listet den Inhalt des Directories auf  |
| <code>ls -alh</code>                              | zeigt alle Inhalte (auch versteckte ( <b>a</b> )) gelistet ( <b>l</b> ) mit verständlichen Angaben ( <b>h</b> )) |
| <code>pwd</code>                                  | gibt das aktuelle Directory an (print working directory)   |
| <code>uname -a</code>                             | gibt Systeminformationen an  |
| <code>whoami</code>                               | gibt den Username zurück   |
| <code>cd &lt;directory&gt;</code>                 | geht in das Directory <directory>  |
| <code>mkdir &lt;directory&gt;</code>              | erstellt ein Verzeichnis namens <directory> im aktuellen Verzeichnis   |
| <code>mv &lt;old file&gt; &lt;new file&gt;</code> | benennt <old file> in <new file> um (umbenennen und verschieben ist gleiche Aktion)                              |
| <code>rm &lt;file&gt;</code>                      | löscht <file>  |
| <code>echo &lt;params&gt;</code>                  | gibt <params> zurück   |



# Liste wichtiger Befehle

|   |  |
|---|--|
| <code>&lt;command&gt; &amp;</code>  | führt <code>&lt;command&gt;</code> im Hintergrund aus  |
| <code>&lt;command 1&gt; &amp;&amp;</code><br><code>&lt;command 2&gt;</code> | führt <code>&lt;command 1&gt;</code> aus und anschließend <code>&lt;command 2&gt;</code> , sofern bei der Ausführung von <code>&lt;command 1&gt;</code> kein Error auftrat |
| <b>touch</b> <code>&lt;file&gt;</code>                                      | erstellt <code>&lt;file&gt;</code> oder ändert dessen Zeitstempel  |
| <code>cat &lt;file&gt;</code>   | gibt den Inhalt von <code>&lt;file&gt;</code> aus  |
| <code>grep &lt;pattern&gt;</code>   | durchsucht nach <code>&lt;pattern&gt;</code>   |

Jedes Directory beinhaltet die Directories `.` (aktuelles Directory) und `..` (darüberliegendes Directory).

Versteckte Files und Directories beginnen mit einem Punkt (`.`).

# Übungsaufgaben

- Suche dir einen Ort, an dem du ein neues Directory erstellst, in welchem du im Laufe des Semesters alle Übungsaufgaben abspeicherst.
- Erstelle nun in diesem neuen Directory weitere Subdirectories für die ersten 12 Übungsaufgaben.

# Lösungen der Übungsaufgaben

- Suche dir einen Ort, an dem du ein neues Directory erstellst, in welchem du im Laufe des Semesters alle Übungsaufgaben abspeicherst.
- Erstelle nun in diesem neuen Directory weitere Subdirectories für die ersten 12 Übungsaufgaben.

```
mkdir PROG-WiSe18 && cd PROG-WiSe18
```

# Lösungen der Übungsaufgaben

- Suche dir einen Ort, an dem du ein neues Directory erstellst, in welchem du im Laufe des Semesters alle Übungsaufgaben abspeicherst.
- Erstelle nun in diesem neuen Directory weitere Subdirectories für die ersten 12 Übungsaufgaben.

```
mkdir PROG-WiSe18 && cd PROG-WiSe18
```

```
mkdir Aufgabe{1..12}
```

# Shortcuts der Shell

|                 |   |
|-----------------|---|
| Ctrl + C        | Beendet das laufende Programm und geht zur Eingabe zurück                                   |
| ↑               | Holt die vorherige Eingabe in die Prompt zurück   |
| Ctrl + R        | Durchsucht die Befehlshistorie  |
| ↓               | Geht zum nächsten Befehl in der Befehlshistorie   |
| Ctrl + ←/ →     | Geht ein Wort nach links/ rechts  |
| Alt + backspace | Löscht das letzte Wort  |
| Ctrl + K        | Löscht die Zeile nach dem Cursor  |
| Alt + D         | Löscht das Wort nach dem Cursor   |
| Ctrl + T        | Vertauscht die letzten beiden Buchstaben  |
| Alt + T         | Vertauscht die letzten beiden Wörter  |
| TAB             | Autovervollständigung (sofern eindeutig, sonst Aufzeigen der möglichen Vervollständigungen) |
| Ctrl + D        | Beendet die Shell (logout)  |

# Weitere Syntax

## Pipe

Die sogenannte Pipe (`|`) nimmt den Output eines Programms/ Befehls und übergibt diesen an den folgenden Befehl.

Beispiel:

```
fortune | cowsay
```

Nützlich ist dies bei eurer ersten Übungsaufgabe. Das Programm erwartet zwei Ganzzahlen als Eingabe. Falls ihr euer Programm mit `roundingerror` bezeichnet habt, dann könnt ihr die Ausführung folgendermaßen abkürzen:

```
echo "665857" "470832" | ./roundingerror
```

# Weitere Syntax

Beispiel (**grep**):

Wir suchen alle Funktionsnamen im File ivalmod.f95.

```
cat ivalmod.f95 | grep -n -i function
```

Unabhängig von der Groß- und Kleinschreibung werden alle Zeilen (mit Zeilennummer) ausgegeben, die das Schlüsselwort 'function' enthalten.

## Files beschreiben

Output kann in Files geschrieben werden. Mit > beschreibt man einen File neu (der alte Inhalt wird überschrieben). Mit >> schreib man an das Ende eines Files.

```
echo "665857" "470832" | ./roundr > file1
```

```
echo "665857" "470832" | ./roundr >> file1
```

# Editor

Zum Erstellen eurer Programme könnt ihr den Editor eurer Wahl verwenden.  
Zum Beispiel:

- gedit
- kate
- vim
- emacs
- SublimeText
- Xcode
- ATOM
- etc.



# Unser erstes FORTRAN-Programm

```
PROGRAM hello  
    implicit none  
  
    write(*,*) 'Hello World'  
END PROGRAM
```

## Aufgabe

Öffne den Editor deiner Wahl und schreibe dein eigenes Hello-World-Programm. Speichere es in einem entsprechenden Verzeichnis ab.

# Der GNU Fortran Compiler

Der geschriebene Fortran-Quellcode (source code) eures Programmes wird in einem File <name>.f95 abgespeichert und kann dann mit dem Compiler gfortran in Maschinensprache übersetzt werden. Im Folgenden nennen wir unseren Programmcode source.f95. Mittels

```
gfortran source.f95
```

wird das Programm übersetzt und ein ausführbares file (executable) namens a.out erstellt.

Um die Executable nach Belieben zu benennen, gibt es den Parameter -o.

```
gfortran -o myAwesomeExecutable source.f95
```

Mit dem Parameter -Wall kann man sich beim Kompilieren alle Warnungen ausgeben lassen.

```
gfortran -Wall source.f95
```

# Der GNU Fortran Compiler

Möchte man sich auf einen Standard beschränken (was eine gute Idee ist), kann man diesen mit dem `-std` parameter angeben:

```
gfortran -std=f95 source.f95
```

Wenn man mit Arrays arbeitet (zu einem späteren Zeitpunkt im Semester) und vom Compiler einen Check für zulässige Indexzugriffe haben möchte, ist das mit `-fbounds-check` möglich.

```
gfortran -fbounds-check source.f95
```

Diese Parameter sind alle kombinierbar und nachzuschlagen in der `gfortran` Manual. Zum Kompilieren mit Modulen und Bibliotheken kommen wir, wenn wir Module kennenlernen.

# Übungsaufgaben

- Löse Praxisaufgabe 1 des ersten Übungsblattes.
- ZUSATZ: Führe das Programm zur ersten Praxisaufgabe aus, indem du die Eingabewerte direkt übergibst und die Ergebnisse in einer Datei abspeicherst.