# Vorkurs - Unix & Git

für Programmierung 2

---

Autoren: Pascal Held, Daniel Weber

9. April 2021

Universität des Saarlandes

# Organisation

# Tagesablauf

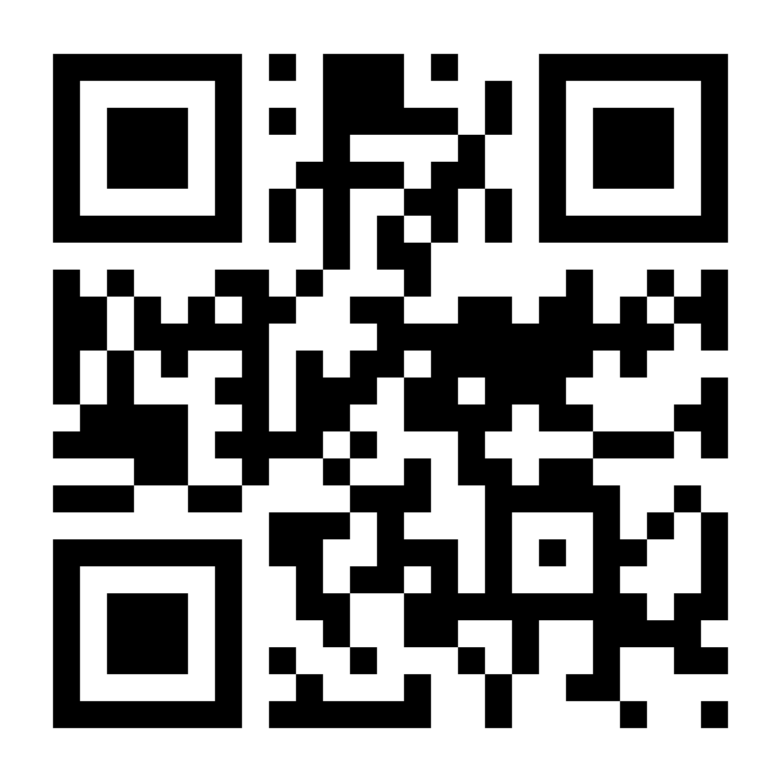
| | | |
|---|---|---|
| 09:00 | 10:00 | Unix Einführung |
| 10:00 | 11:15 | VM-Installation und Pause |
| 11:15 | 12:00 | Git Einführung |
| 12:00 | 13:00 | Mittagspause |
| 13:00 | 15:00 | Übungsblatt |
| 15:00 | 16:00 | Tutorium |

# Registrieren nicht vergessen!

(auf prog2.saarland)

# Fragerunde

# Zugang zur Live-Umfrage



http://etc.ch/nyKh

**Welches Betriebssystem nutzt du auf deinem Computer?**

- Windows
- MacOS
- Linux
- Sonstiges

## Besitzt du einen eigenen Laptop?

- Ja!
- Nein!

**Hast du bereits Erfahrung mit der Kommandozeile?**

- Ja!
- Nein!

**Hast du schonmal von Linux gehört / hast du Linux schonmal benutzt?**

- Ja, ich habe von Linux gehört.
- Noch nie gehört.
- Ja, ich habe Linux schon genutzt.
- Kann man das essen?

**Weißt du bereits, was eine virtuelle Maschine ist?**

- Ja!
- Nein!

**Ist dir der Begriff der Versionsverwaltung geläufig / hast du schonmal welche genutzt?**

- Ja, der Begriff sagt mir etwas.
- Weder noch.
- Ja, geläufig und auch schon genutzt.

# Unix Grundlagen

# Was ist Unix?

- Grundlage vieler moderner Betriebssysteme
- Unix- bzw. unixartiges System dient als Sammelbegriff für Betriebssysteme, die die Unixkonzepte umsetzen
- Linux ist eine familie von unixartigen Betriebssystemen

# Kommandos

```
$ cmd␣arg(1)␣arg(2)␣…␣arg(n)
```

---

# Kommandos

```
$ cmd␣arg(1)␣arg(2)␣…␣arg(n)
```

---

- `$` Prompt
- `cmd` Befehl/Kommando
- `arg(1)` 1. Argument
- `arg(2)` 2. Argument
- `arg(n)` *n*. Argument

# Quotierung

Kommandozeilenargumente, die Leerzeichen enthalten, müssen quotiert werden.

Dazu wird das `'`-Zeichen verwendet, welches die zu quotierende Zeichenkette beidseitig einschließt.

Die `'`-Zeichen sind nicht Teil des Arguments.

**Merke**

`$ cmd␣a␣b`

stellt einen anderen Aufruf dar als

`$ cmd␣'a␣b'`

# Dateisysteme

**Was gehört zu einem Dateisystem?**

- Dateien
- Verzeichnisse
- Sonstiges

# Dateisysteme

**Was gehört zu einem Dateisystem?**

- Dateien
- Verzeichnisse
- Sonstiges

---

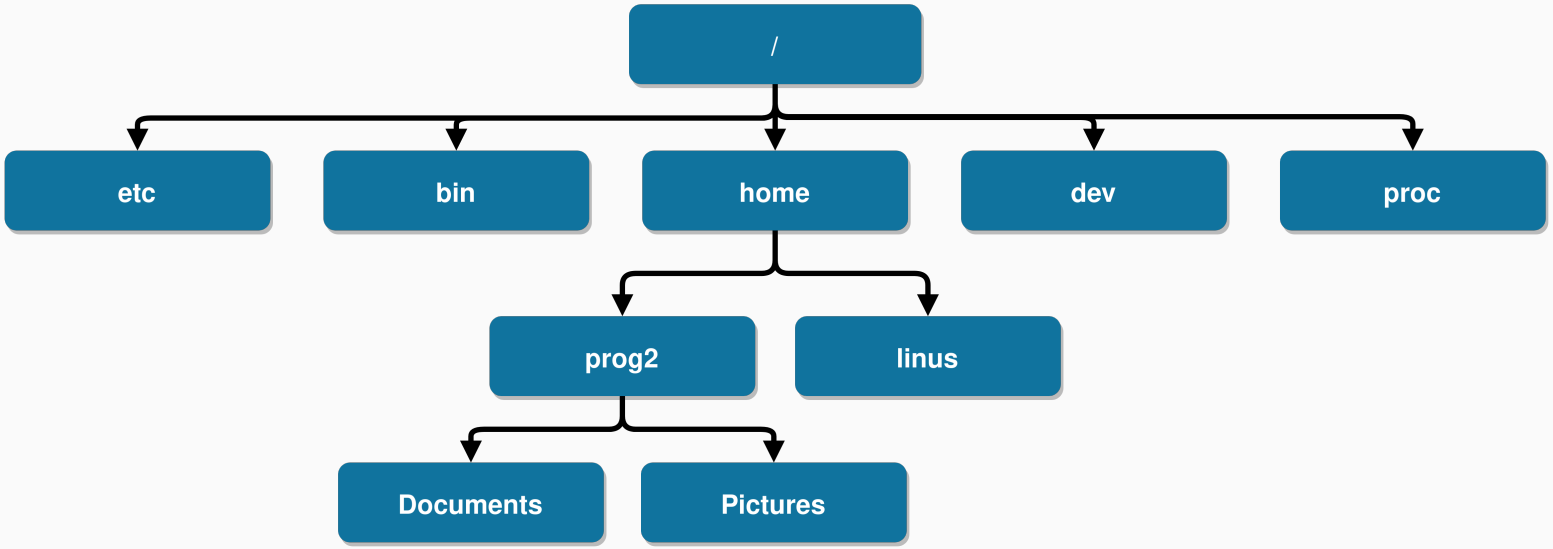
Alle drei Antworten sind richtig!

# Was ist dieses Sonstige?

- Namen
- Besitzer (Nutzer und Gruppen)
- Zugriffsrechte (lesen, schreiben und ausführen)
- Typ (Datei, Verzeichnis, symbolische Links[1] usw.)
- und noch mehr

[1] https://de.wikipedia.org/wiki/Symbolische_Verkn%C3%BCpfung

# Dateisystem Struktur




Beispiel-Pfade:

- `/home/prog2/Documents`
- `/bin`

# Pfade

**Pfad**

Ein Pfad ist eine Zeichenfolge bestehend aus Namen und Schrägstrichen. Namen sind dabei bspw. Verzeichnis- oder Dateinamen. Hierbei teilen die Schrägstriche die einzelnen Namen voneinander ab.

---

# Pfade

**Pfad**

Ein Pfad ist eine Zeichenfolge bestehend aus Namen und Schrägstrichen. Namen sind dabei bspw. Verzeichnis- oder Dateinamen. Hierbei teilen die Schrägstriche die einzelnen Namen voneinander ab.

---

**Welche Zeichenketten stellen gültige Pfade dar?**

- `/`
- `/b`
- `a/b`
- `a`
- `//`
- `a/b/c`
- `a//c`
- `//c`

# Pfade

**Welche Zeichenketten stellen gültige Pfade dar?**

- `/`
- `/b`
- `a/b`
- `a`
- `//`
- `a/b/c`
- `a//c`
- `//c`

---

Alle Antworten sind korrekt!

# Pfade

**absolute vs. relative Pfade**

Ein Pfad heißt absolut, wenn er mit einem Schrägstrich beginnt.

Alle anderen heißen relativ.

# Besondere Verzeichnisse/Dateien

| | |
|---|---|
| `/` | Root |
| `.` | Self |
| `..` | Parent |
| `~` | Home |
| `.<name>` | versteckte Dateien und Verzeichnisse |

# Fragen?

# Recap: Kommandos

```
$ cmd␣arg(1)␣arg(2)␣…␣arg(n)
```

---

- `$` Prompt
- `cmd` Befehl/Kommando
- `arg(1)` 1. Argument
- `arg(2)` 2. Argument
- `arg(n)` *n*. Argument

# Einfache Kommandos

`$ exit` schließt die Kommandozeile

# Einfache Kommandos

`$ exit` schließt die Kommandozeile

`$ date` gibt das aktuelle Datum mit Uhrzeit aus

# Einfache Kommandos

`$ exit` schließt die Kommandozeile

`$ date` gibt das aktuelle Datum mit Uhrzeit aus

`$ echo 'Hallo Welt!'` gibt „Hallo Welt!“ wieder aus

# Hilfe!

Wo bekomme ich Hilfe?

## Wo bekomme ich Hilfe?

- Handbuchseiten (manualpages)

```
$ man cmd
```

## Wo bekomme ich Hilfe?

- Handbuchseiten (manualpages)

```
$ man cmd
```

- man-ception

```
$ man man
```

## Wo bekomme ich Hilfe?

- Handbuchseiten (manualpages)

```
$ man cmd
```

- man-ception

```
$ man man
```

- das Internet (Google & Co., StackExchange, ...)

# Recap: Pfade

**Pfad**

Ein Pfad ist eine Zeichenfolge bestehend aus Namen und Schrägstrichen. Namen sind dabei bspw. Verzeichnis- oder Dateinamen. Hierbei teilen die Schrägstriche die einzelnen Namen voneinander ab.

---

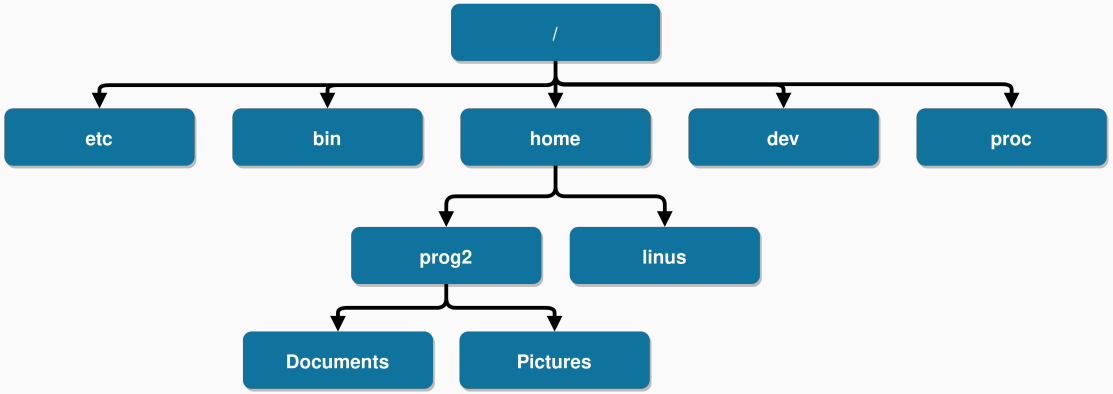
**absolute vs. relative Pfade**

Ein Pfad heißt absolut, wenn er mit einem Schrägstrich beginnt. Alle anderen heißen relativ.

# Recap: Besondere Verzeichnisse/Dateien

| | |
|---|---|
| `/` | Wurzelverzeichnis |
| `.` | Self |
| `..` | Parent |
| `~` | Home |
| `.<name>` | versteckte Dateien und Verzeichnisse |

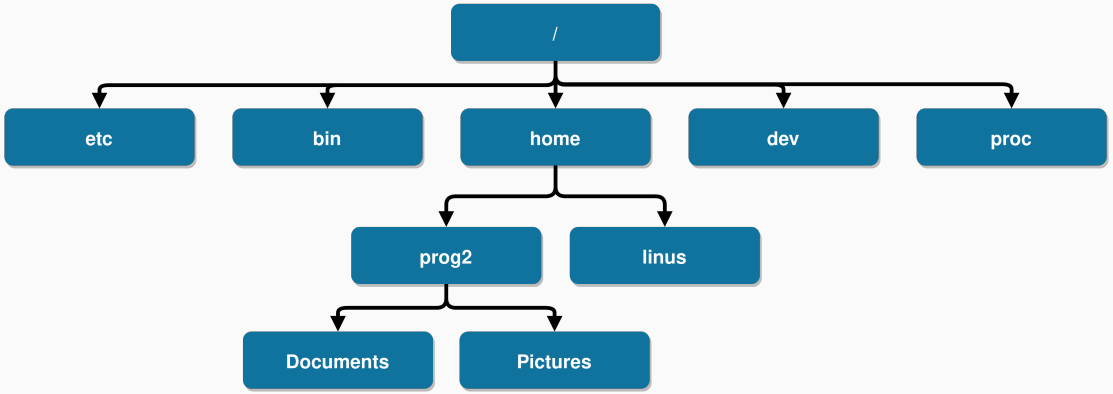
# Arbeitsverzeichnis




`$ pwd` (print working directory) gibt den (absoluten) Pfad zu dem aktuellen Arbeitsverzeichnis aus

Beispiel: `$ pwd` im ordner `etc` gibt `/etc` aus.

# Arbeitsverzeichnis wechseln




`$ cd <pfad>` wechselt das Arbeitsverzeichnis zu dem durch den Pfad spezifizierten Verzeichnis.

`cd` ist eine Abkürzung fuer „change directory “.

Beispiel: `$ cd /home/prog2`

# Verzeichnisinformationen

**Wofür steht `ls`?**

- lame stuff!?
- langsam (aber) sicher
- list

# Verzeichnisinformationen

**Wofür steht `ls`?**

- lame stuff!?
- langsam (aber) sicher
- list

---

„List“ ist die richtige Antwort.

---

# Verzeichnisinformationen

**Wofür steht `ls`?**

- lame stuff!?
- langsam (aber) sicher
- list

---

„List“ ist die richtige Antwort.

---

`$ ls` listet also die Inhalte des Arbeitsverzeichnisses auf

# Recap: Dateisysteme: Was ist dieses Sonstige?

- Namen
- Besitzer (Nutzer und Gruppen)
- Zugriffsrechte (lesen, schreiben und ausführen)
- Typ (Datei, Verzeichnis, symbolische Links usw.)
- und noch mehr

---

# Recap: Dateisysteme: Was ist dieses Sonstige?

- Namen
- Besitzer (Nutzer und Gruppen)
- Zugriffsrechte (lesen, schreiben und ausführen)
- Typ (Datei, Verzeichnis, symbolische Links usw.)
- und noch mehr

---

**Merke**

`$ ls` zeigt diese Informationen an, wenn es mit der Option `-l` aufgerufen wird. Also `$ ls -l`. Dabei steht das `-l` für „long“.

# Recap: Besondere Verzeichnisse/Dateien

| | |
|---|---|
| `/` | Wurzelverzeichnis |
| `.` | Self |
| `..` | Parent |
| `~` | Home |
| `.<name>` | versteckte Dateien und Verzeichnisse |

# Recap: Besondere Verzeichnisse/Dateien

| | |
|---|---|
| `/` | Wurzelverzeichnis |
| `.` | Self |
| `..` | Parent |
| `~` | Home |
| `.<name>` | versteckte Dateien und Verzeichnisse |

**Merke**

`$ ls` zeigt versteckte Dateien und Verzeichnisse an wenn es mit `-a` aufgerufen wird. Also `$ ls -a`. Dabei steht das `-a` für „all“.

# Verzeichnisteilbaum ausgeben

`$ tree` gibt den Verzeichnisteilbaum relativ zum Arbeitsverzeichnis aus

# Dateiinhalte ausgeben

`$ cat` kann genutzt werden, um Dateiinhalte auszugeben

Beispiel: `$ cat README.md`

# Verzeichnisbaum modifizieren

`$ touch` erstellt eine Datei

`$ rm` (remove) entfernt eine Datei

# Verzeichnisbaum modifizieren

`$ touch` erstellt eine Datei

`$ rm` (remove) entfernt eine Datei

`$ mkdir` (make directory) erstellt ein Verzeichnis

`$ rm -r` (remove recursively) entfernt ein Verzeichnis

# Verzeichnisbaum modifizieren

`$ touch` erstellt eine Datei

`$ rm` (remove) entfernt eine Datei

`$ mkdir` (make directory) erstellt ein Verzeichnis

`$ rm -r` (remove recursively) entfernt ein Verzeichnis

`$ cp` (copy) kopiert eine Datei / ein Verzeichnis

`$ mv` (move) verschiebt eine Datei / ein Verzeichnis

# Verzeichnisbaum modifizieren

`$ touch` erstellt eine Datei

`$ rm` (remove) entfernt eine Datei

`$ mkdir` (make directory) erstellt ein Verzeichnis

`$ rm -r` (remove recursively) entfernt ein Verzeichnis

`$ cp` (copy) kopiert eine Datei / ein Verzeichnis

`$ mv` (move) verschiebt eine Datei / ein Verzeichnis

**Knobelaufgabe**

Wie kann man mit diesen Befehlen eine Datei / ein Verzeichnis umbenennen? *Hinweis:* Es gibt einen schweren und einen leichten Weg.

# Make

Programmierprojekte benutzen oft „Makefiles “.

Leicht zu überprüfen mit `$ ls`.

# Make

Programmierprojekte benutzen oft „Makefiles “.

Leicht zu überprüfen mit `$ ls`.

`$ make` führt den Default-Befehl des Makefiles aus.

`$ make <target>` führt einen speziellen Make-Befehl aus.

Beispiel: `$ make clean` wird oft benutzt um generierte Dateien zu löschen.

# Übersicht Kommandos

- `$ cat`
- `$ cd`
- `$ cp`
- `$ date`
- `$ echo`
- `$ exit`
- `$ ls`
- `$ make`
- `$ man`
- `$ mkdir`
- `$ mv`
- `$ pwd`
- `$ rm`
- `$ touch`
- `$ tree`

# Hilfreiche Tasten(kombinationen)

- `Tab` Autovervollständigung
- `Strg + C` Abbruch
- `Strg + D` EOF senden (noch mehr Abbruch)
- `Strg + Shift + C` Kopieren
- `Strg + Shift + V` Einfügen

# Für Neugierige

- `$ awk`
- `$ chgrp`
- `$ chmod`
- `$ chown`
- `$ df`
- `$ du`
- `$ find`
- `$ grep`
- `$ head`
- `$ ps`
- `$ scp`
- `$ sed`
- `$ sort`
- `$ ssh`
- `$ tail`
- `$ tr`
- `$ uniq`
- `$ wc`
- und viele mehr ...

# Für Experimentierfreudige

`$ man bash` ist ein guter Einstiegspunkt, um noch mehr zu lernen.

Diese Handbuchseite geht auf sehr viele Dinge ein. Einige Interessante sind:

- pipes
- quotes
- redirections
- shell-scripts
- die Liste geht beliebig weiter ...

# Nano / Vim

`$ nano` ist ein einfacher Texteditor für die Kommandozeile.

`$ vim` (bzw. `$ vimtutor` für Anfänger) ermöglicht sehr effiziente Wege Text oder Code zu bearbeiten.

**WARNUNG:** Vim hat eine steile Lernkurve!

P.S.: Vim beendet man mit: `<Esc>` → `:` → `q!` → `<Return>`

# Fragen?

# Tagesablauf

| | | |
|---|---|---|
| 09:00 | 10:00 | Unix Einführung |
| 10:00 | 11:15 | VM-Installation und Pause |
| 11:15 | 12:00 | Git Einführung |
| 12:00 | 13:00 | Mittagspause |
| 13:00 | 15:00 | Übungsblatt |
| 15:00 | 16:00 | Tutorium |

# Git Grundlagen

# Motivation

Was ist Versionsverwaltung?

## Was ist Versionsverwaltung?

Ein System mit dem Änderungen erfasst, gesichtet und wiederhergestellt werden können.

## Was ist Versionsverwaltung?

Ein System mit dem Änderungen erfasst, gesichtet und wiederhergestellt werden können.

## Wieso brauchen wir Versionsverwaltung?

## Was ist Versionsverwaltung?

Ein System mit dem Änderungen erfasst, gesichtet und wiederhergestellt werden können.

## Wieso brauchen wir Versionsverwaltung?

For starters: Die Projekte in Prog2 werden mittels des Versionsverwaltungsprogrammes Git eingereicht. **Alle Projekte!**

# Eigenschaften

Welche Eigenschaften sollte eine Versionsverwaltung haben? Sie sollte ...

# Eigenschaften

Welche Eigenschaften sollte eine Versionsverwaltung haben? Sie sollte ...

- ...Unterschiede/Änderungen zwischen Versionen erkennen und anzeigen können.
- ...eine Version festhalten/aufzeichnen können.
- ...die Möglichkeit bieten Versionen benennen zu können.
- ...alte/vorherige Versionen zugänglich machen.
- ...schnell arbeiten.
- ...die Möglichkeit bieten Versionen und damit Änderungen zu kommentieren.
- ...eine Übersicht über alle Versionen anzeigen können.
- ...Änderungen rückgängig machen können.
- ...mit anderen Nutzern synchronisierbar sein.

Git erfüllt all diese Kriterien. Darüber hinaus:

Git erfüllt all diese Kriterien. Darüber hinaus:

Verteilt $\rightarrow$ ermöglicht einfaches paralleles Arbeiten

Git erfüllt all diese Kriterien. Darüber hinaus:

Verteilt $\rightarrow$ ermöglicht einfaches paralleles Arbeiten

Versionen werden mit Stempel versehen (Datum, Name, Mail etc.)

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits
- Referenzen – Branches – Tags

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits
- Referenzen – Branches – Tags
- `HEAD`

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits
- Referenzen – Branches – Tags
- `HEAD`
- `master` (oder `main`)

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits
- Referenzen – Branches – Tags
- `HEAD`
- `master` (oder `main`)
- `.git` Verzeichnis

# Git Konzepte

- `$ git subcmd arg(1) arg(2) … arg(n)`
- Arbeitsversion/-kopie – Working Copy
- Index/Staging Area
- Commits
- Referenzen – Branches – Tags
- `HEAD`
- `master` (oder `main`)
- `.git` Verzeichnis
- `.gitignore` Datei

# Hilfe!

`$ git help subcmd` zeigt die Handbuchseite für das gegebene Git-Kommando an.

# Konfiguration

`$ git config` dient der Konfiguration von Git Repositories.

- `-- global` richtet die globale Konfiguration ein.
  - `user.name` Name des Nutzers
  - `user.email` E-Mail des Nutzers

# Das erste Git Repository

`$ git init` erstellt ein leeres Git Repository.

# Das erste Git Repository

`$ git init` erstellt ein leeres Git Repository.

`$ git status` zeigt den Status des Git Repository an.

# Grundlegende Befehle

`$ git log` zeigt den Versionsverlauf an.

# Grundlegende Befehle

`$ git log` zeigt den Versionsverlauf an.

`$ git diff` zeigt die Unterschiede zwischen Versionen an.

# Grundlegende Befehle

`$ git log` zeigt den Versionsverlauf an.

`$ git diff` zeigt die Unterschiede zwischen Versionen an.

`$ git add` fügt die gegebenen Dateien dem Index hinzu.

# Grundlegende Befehle

`$ git log` zeigt den Versionsverlauf an.

`$ git diff` zeigt die Unterschiede zwischen Versionen an.

`$ git add` fügt die gegebenen Dateien dem Index hinzu.

`$ git commit` erstellt einen Commit basierend auf dem Stand des Indexes.

# Synchronisieren

`$ git clone` gegeben ein anderes (entferntes) Repository, erstelle eine lokale Kopie des Repository.

# Synchronisieren

`$ git clone` gegeben ein anderes (entferntes) Repository, erstelle eine lokale Kopie des Repository.

`$ git push` lade lokale Änderungen (Commits) zu dem entfernten Repository hoch.

# Synchronisieren

`$ git clone` gegeben ein anderes (entferntes) Repository, erstelle eine lokale Kopie des Repository.

`$ git push` lade lokale Änderungen (Commits) zu dem entfernten Repository hoch.

`$ git pull` lade Änderungen an dem entfernten Repository herunter.

# Weiterführende Befehle

- `$ git branch`
- `$ git merge`
- `$ git tag`
- `$ git clean`
- `$ git fetch`
- `$ git rebase`
- `$ git revert`
- `$ git rm`

# Weiterführendes Material

Buch: „Pro Git“ `https://git-scm.com/book/de/v1`

# Fragen?

# Vielen Dank!