smartlearn
virtual learning. real success.

MODUL 133

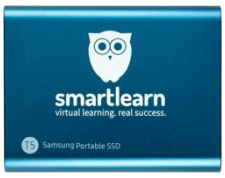
TEIL 1: EINFÜHRUNG IN RUBY, RAILS APPLIKATION ERSTELLEN, STATISCHE SEITE ERSTELLEN

Ralph Maurer / Michael Abplanalp

Inhaltsverzeichnis AB133-01

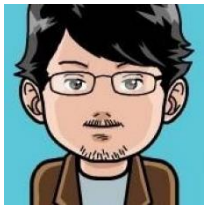
Modul 133: Webapplikationen mit Session-Handling realisieren

Teil 1: Einführung in Ruby, Rails-Applikation erstellen, statische Seite erstellen



Was ist Ruby on Rails?

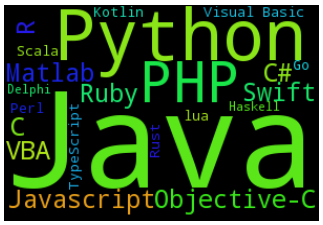
Ruby



Matz

Ruby ist eine Programmiersprache. Sie wurde vor ca. 20 Jahren von Yukihiro "Matz" Matsumoto erfunden.

Ruby ist nach den Spitzenreitern Java und Python eine der beliebtesten Programmiersprachen der Welt. Die Popularität von Ruby basiert auf dem Erfolg in der Webprogrammierung (besonders durch das Framework Rails).



Quelle: http://pypl.github.io/PYPL.html

Matz betont jedoch immer wieder, dass Ruby eine allgemeine Sprache ist wie Java und C# und nicht auf Webentwicklung reduziert werden darf.

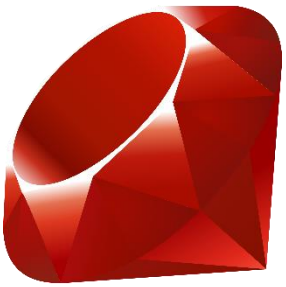
Rails

David Heinemeier Hansson

Rails ist eine Softwarebibliothek, die Ruby speziell für die Webprogrammierung erweitert. Rails wurde von dem Unternehmer und Autorennfahrer David Heinemeier Hansson entwickelt. David nannte seine Entwicklung "Ruby on Rails", heute spricht man meist nur noch von "Rails". Aktuell ist er Inhaber der Softwarefirma Basecamp.

Technisch handelt es sich bei Rails um eine Paketbibliothek (genauer eine RubyGem), die über die Befehlszeilenschnittstelle des Betriebssystems installiert wird. RubyGem ist ein Paketmanager, der es erlaubt, Ruby-Programme als Standardprogramme zu verteilen. Diese standardisierten, selbstenthaltenen und eigenständigen Rubyprogramme nennt man Gem (in Deutsch übersetzt «Edelstein»).

RubyGems



RubyGems (oder kurz Gems) ist das offizielle Paketsystem für die Programmiersprache Ruby. Es stellt ein Paketformat, ein Werkzeug zur Verwaltung von Paketen und ein Repositorium für deren Verteilung zur Verfügung. Mit ihm hat der Anwender die Möglichkeit, mehrere (zum Beispiel ältere oder jüngere) Versionen eines Programmes, Programmteiles oder einer Bibliothek gesteuert nach Bedarf einzurichten, zu verwalten oder auch wieder zu entfernen.

Bei jeder Rails-Applikation befindet sich ein *Gemfile* im Root-Ordner der Applikation. Dort befinden sich die notwendigen Einträge für die Gems.

Erstellen des Workspace Ordners

Damit wir eine Rails-App erstellen können, benötigen wir einen Speicherort und einen Namen für die neue App. Rails-Entwicklerinnen und -Entwickler nutzen oftmals die Verzeichnisse `code/` oder `projects/`. Wir wollen `workspace/` verwenden.

Erstellen Sie im `/home/vmadmin` einen entsprechenden Ordner:

```
$ cd /home/vmadmin
$ mkdir workspace
$ cd workspace
```

Einführung in Ruby

Ruby hat nichts mit Rails zu tun, hingegen hat Rails eine Menge mit Ruby zu tun. Wie wir oben bereits gesehen haben, ist Rails eine Erweiterung von Ruby und benutzt natürlich auch Ruby als Skriptsprache. Deshalb ist es wichtig, dass Sie Kenntnisse von Ruby haben.

Ruby ist eine Skriptsprache, die zur Laufzeit interpretiert wird. Der gesamte Sprachumfang von Ruby steht in der Rails-Konsole mit IRB (Interactive Ruby) zur Verfügung. Starten Sie *irb* und geben Sie "Hallo Welt" ein.

```
$ cd /home/vmadmin/workspace
$ irb
2.6.3 :001 >"Hallo Welt"
```

War das gerade das berühmte "Hallo Welt"-Programm? Nicht ganz… denn Ruby hat uns lediglich das Ergebnis des letzten Ausdrucks wiedergegeben. In der interaktiven Umgebung erstellen wir also keine Programme, sondern führen Ruby-Code direkt aus.

Um Ruby-Programme (Skripts) zu erstellen, benötigen wir einen Editor. Ein paar Editoren bzw. IDEs stehen auf der bmLP1 bereits zur Verfügung. Wir werden im Modul 133 mit *Atom* arbeiten, Sie können als Alternative einen Editor Ihrer Wahl einsetzen.

Am einfachsten öffnen Sie *Atom*, indem Sie in der Konsole ins richtige Verzeichnis wechseln und dort *Atom* aufrufen (der Punkt "." steht für das aktuelle Verzeichnis):

```
$ cd /home/vmadmin/workspace
$ atom .
```

Erstellen Sie eine neue Datei (*New File*) und schreiben Sie:

```
puts "Hallo Welt"
```

Speichern Sie anschliessend die Datei unter dem Namen *HalloWelt.rb*. Die Endiung *rb* ist wichtig, damit die Datei als Ruby-Skript erkannt wird.

Nun haben wir das "Hallo Welt"-Programm erstellt!

Ausführen können Sie das Programm in der Konsole:

```
$ cd /home/vmadmin/workspace
$ ruby HalloWelt.rb
```

Es wird "Hallo Welt" in der Konsole ausgegeben.

Tutorials

Einen guten Einstieg, um Programmiersprachen zu lernen, sind Tutorials. Deshalb ist es sinnvoll, dass Sie ein oder zwei Tutorials durchspielen. In einem zweiten Schritt werden Sie ein paar Aufgaben lösen.

Empfohlene Tutorials:

Englisches Tutorial: https://www.tutorialspoint.com/ruby/

Deutsches Tutorial: https://www.ruby-lang.org/de/documentation/quickstart/

Ausschnitt aus dem Buch *Learn Rails 5.2* von Stefan Wintermeyer:
sh-modules/iet-133/06_Diverses/Ruby Introduction.pdf

Aufgaben

Sie können die Aufgaben sowohl interaktiv in der Rails-Konsole lösen oder auch ein Ruby-Skript erstellen und dieses dann ausführen. Die Empfehlung ist, dass Sie mit Ruby-Skripts arbeiten. So können Sie den Code zu Ihrer Dokumentation aufbewahren und Sie können die Programme mit verschiedenen Werten testen.

Anstatt die Werte während der Programmausführung einzugeben, können Sie diese gleich beim Programmstart als Argumente mitgeben:

```
$ ruby aufgabe1.rb 30 55
```

Im Skript können Sie wie folgt auf die beiden Zahlen 30 und 55 zugreifen:

zahl1 = ARGV.at(0)
zahl2 = ARGV.at(1)

ARGV ist ein Array, das alle Argumente enthält.

Verzweigungen

Realisieren Sie die Aufgaben 1 + 2 mit Verzweigungen.

Aufgabe 1: Die grössere von zwei Zahlen finden

Gegeben sind die beiden Zahlen a und b. Es soll ein Programm erstellt werden, welches immer zuerst die grössere der Beiden ausgibt.

Programmstart und Ausgabe:

```
$ ruby aufgabe1.rb 30 55
55 30
```

Aufgabe 2: Zwei Zahlen sortieren

Gegeben sind die beiden Zahlen a und b. Lesen Sie das erste Argument in die Variable a und das zweite Argument in die Variable b. Am Ende des Programmes muss die grössere Zahl in Variable a stehen, Sie müssen je nachdem die Werte der beiden Variablen vertauschen. Zur Kontrolle sollen die beiden Variablen ausgegeben werden.

Programmstart und Ausgabe:

```
$ ruby aufgabe2.rb 20 31
Variable a: 31, Variable b: 20
```

while- und until-Schleifen

Realisieren Sie die Aufgabe 3 einmal mit einer while- und einmal mit einer until-Schleife.

Aufgabe 3: Versteckspiel

Als Suchender muss man zunächst bis 100 Zählen, dann laut "Ich komme..." rufen. Erst dann darf man zum Suchen aufbrechen. Dieses simple Spiel hat einen einfachen Algorithmus für das Hochzählen:
Die erste Zahl ist 1. Prüfe, ob die 100 erreicht wurde. Wenn das nicht erfüllt ist, rufe die Zahl und erhöhe diese um eins. Wenn 100 erreicht ist, dann rufe "Ich komme".

Erstellen Sie ein entsprechendes Ruby-Programm.

Ausgabe:

```
1
2
...
99
Ich komme
```

for-Schleifen

Realisieren Sie die Aufgaben 4+5 mit for-Schleifen.

Aufgabe 4: Zusammenzählen

Es soll ein Programm erstellt werden, welches die ersten 100 Zahlen zusammenzählt und das Resultat ausgibt. Für diese Aufgabe ist eine for-Schleife am besten geeignet.

Ausgabe:

```
1
3
...
5050
```

Aufgabe 5: Fakultät

Es soll ein Programm erstellt werden, welches die Fakultät einer gegebenen Zahl berechnet.

Programmstart und Ausgabe:

```
$ ruby aufgabe5.rb 10
1
2
6
...
```

Argumente, die einem Programm mitgegeben werden, sind immer Strings, auch wenn es sich um eine Zahl handelt. Sie müssen den String zuerst in eine ganze Zahl umwandeln, um das Argument in der for-Schleife verwenden zu können.

```
input.to_i
```

each-Schleifen

each-Schleifen sind praktisch identisch mit for-Schleifen. Neben dem Abarbeiten der Schleife mit einer bestimmten Anzahl können auch alle Elemente eines Arrays durchlaufen werden. Wir

werden im Modul 133 hauptsächlich mit each-Schleifen arbeiten: Oft gibt uns die Datenbank ein Array mit Datensätzen zurück, z.B. 10 Bücher. Mittels each-Schleife zeigen wir diese 10 Bücher auf der Webseite an.

Aufgabe 6: Zusammenzählen

Wechseln Sie die for-Schleife aus Aufgabe 4 mit einer each-Schleife aus und testen Sie das Ergebnis.

Aufgabe 7: Array durchlaufen

Anstatt einer fixen Anzahl von Schleifendurchläufen wollen wir nun ein Array durchlaufen. Erstellen Sie ein Array mit X Elementen (Argument beim Programmaufruf) und füllen Sie das Array mit den Werten von 1 bis X. Geben Sie danach die Werte des Arrays in einer Schleife aus.

Programmstart und Ausgabe:

```
$ ruby aufgabe7.rb 5
1
2
3
4
5
```

Methoden

Ruby-Methoden sind das Pendant zu Funktionen bzw. Methoden in anderen Programmiersprachen. Mit einer Methode werden sich wiederholenden Anweisungen in eine einzige Einheit gebracht. Der Code wird nur einmal anstatt mehrmals geschrieben.

Aufgabe 8: Grösster gemeinsamer Teiler

Berechnen Sie den grössten gemeinsamen Teiler ggT nach dem Euklidischen Algorithmus.

Hauptprogramm:
1. Einlesen der zwei Zahlen in Variablen.
2. Aufruf der Methode ggt: `ergebnis = ggt(zahl1, zahl2)`
3. Ausgabe von `ergebnis`.

Methode `ggt(zahl1, zahl2)`:

```
solange zahl2 ≠ 0
  wenn zahl1 > zahl2
    zahl1 = zahl1 – zahl2
  sonst
    zahl2 = zahl2 – zahl1
gebe zahl1 zurück
```

Programmstart und Ausgabe:

```
$ ruby aufgabe8.rb 15 20
ggT von 15 und 20 = 5
```

Aufgabe 9 (Zusatzaufgabe)
Drei Zahlen sortieren

Gegeben sind die drei Variablen a, b und c, diese werden beim Programmaufruf mitgegeben. Es soll ein Programm erstellt werden, welches die Variablen aufsteigend sortiert. Vertauschen Sie die Werte dieser Variablen so, dass zum Schluss a < b < c gilt. Prüfen Sie Ihre Methode mit allen Permutationen der Zahlen 1, 2 und 3 (es gibt insgesamt 6 Kombinationen)

Erstellen einer neuen Rails App

Bisher arbeiteten wir mit der Rails-Konsole bzw. der Linux-Konsole gearbeitet. Nun wollen wir die Ergebnisse unserer Programme im Browser betrachten. Dazu integrieren wir unseren Programmcode in eine Rails-Applikation. In dieser Aufgabe zeigen wir nur das minimal Notwendige, um zum gewünschten Ergebnis zu gelangen. Die Erklärungen dazu erhalten Sie in späteren Lektionen.

Als erstes erstellen wir eine neue Rails-Applikation.

1. Öffnen Sie eine Konsole auf der bmLP1 und wechseln Sie in den Ordner `workspace`:

```
$ cd ~/workspace
```

2. In diesem Ordner erstellen wir eine neue Rails-Applikation und nennen sie `myapp` (natürlich ist der Name frei wählbar):

```
$ rails new myapp
```

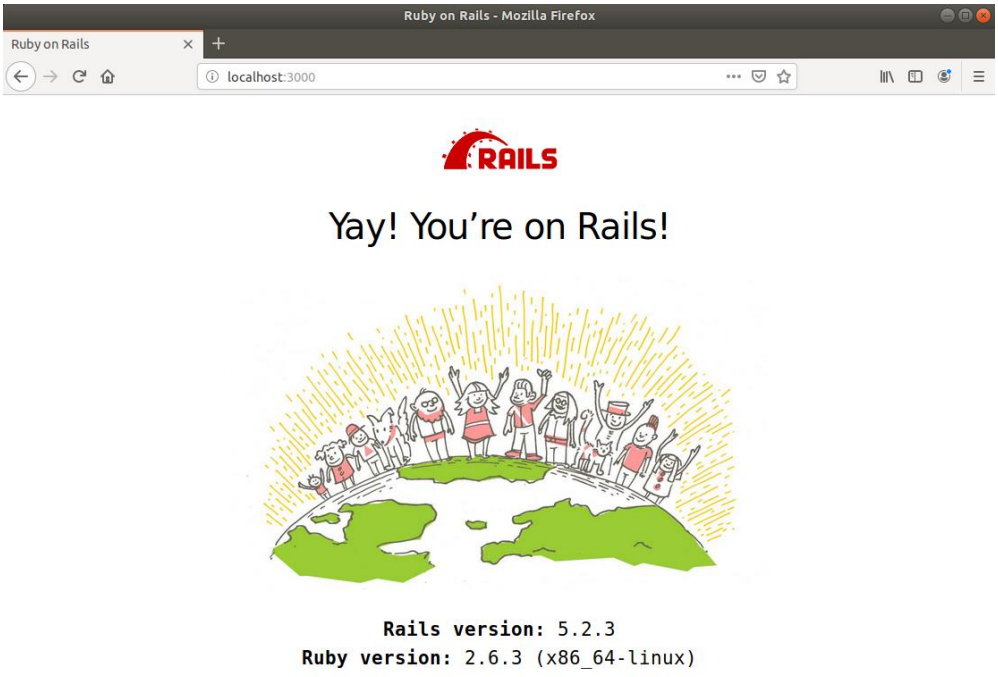
Die Applikation wird dabei mit der neusten Rails-Version erstellt, die auf Ihrem System vorhanden ist.

3. Danach starten Sie den Rails Server:

```
$ rails s
```

`rails s` ist ein Alias für `rails server`

4. Starten Sie Google Chrome oder Firefox mit der Adresse http://localhost:3000



Es wird die Startseite der Applikation angezeigt.

Die Seite "Hello World" erstellen

Vorläufig lassen wir die Seite oben "Yay! You're on Rails!" als Startseite unserer Applikation. Für die "Hello World"-Seite erstellen wir eine Unterseite.

1. Wechseln Sie in der Linux-Konsole in unser Applikationsverzeichnis. Mit einem Befehl erstellen wir einen Controller, eine zugehörige Webseite und einen Eintrag in der Datei `routes.rb`. Dieser Eintrag ist notwendig, damit die Verlinkung stimmt und die Seite gefunden wird.

```
$ cd ~/workspace/myapp
$ rails generate controller helloworld index
```

2. Starten Sie den Atom-Editor mit unserer Rails-Applikation als Startordner:

```
$ cd ~/workspace/myapp
$ atom .
```

3. Öffnen Sie die Datei `config/routes.rb` in Atom. Sie sehen den Eintrag:

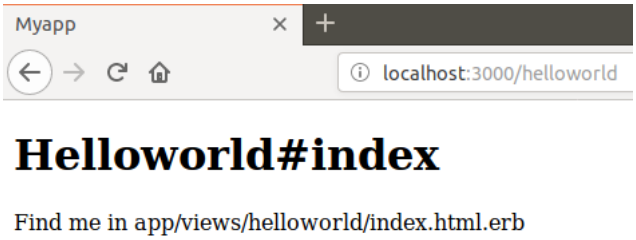
```
get 'helloworld/index'
```

Das bedeutet, dass wir im Browser unsere "Hello World"-Seite mit der URL `localhost:3000/helloworld/index` aufrufen können.

Das ist etwas zu lang, wir wollen die Seite mit `localhost:3000/helloworld` aufrufen können. Deshalb ändern wir diesen Eintrag zu:

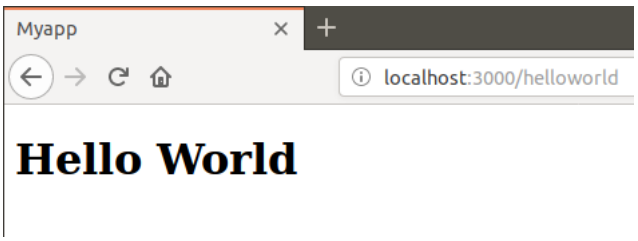
```
get 'helloworld', to: 'helloworld#index'
```

4. Geben Sie nun im Browser die URL `localhost:3000/helloworld` ein.



Wir wollen den Text der Seite anpassen, so dass nur noch "Hello World" angezeigt wird.

5. Öffnen Sie die Datei `app/views/helloworld/index.html.erb` in Atom. Das ist die Seite mit unserem HTML-Quellcode. Passen Sie den Code an, so dass nur noch "Hello World" als h1-Überschrift da steht:
`<h1>Hello World</h1>`
6. Laden Sie die Seite neu und kontrollieren Sie das Ergebnis:



Die Resultate der Aufgaben im Browser anzeigen

Wir haben mit "Hello World" lediglich eine Ausgabe gemacht, ohne irgendetwas zu verarbeiten. Wir wollen exemplarisch mit der Aufgabe 3 zeigen, wie und wo Anweisungen ausgeführt werden in Rails und wie das Ergebnis zum Browser bzw. in die HTML-Seite gelangt.

Das MVC-Muster verbietet es, funktionale Anforderungen in der View durchzuführen (dazu später mehr). Dazu gehören Anweisungen, Berechnungen und Zugriffe auf die Datenbank. Die View hat die einzige Aufgabe, die verarbeiteten Daten zur Anzeige aufzubereiten.

Funktionen werden im Controller erfasst und ausgeführt. Den Controller für unsere "Hello World"-Seite finden wir unter `app/controllers/helloworld_controller.rb`. Der Inhalt sieht wie folgt aus:

```
class HelloworldController < ApplicationController
  def index
  end
end
```

Um den Klassennamen brauchen wir uns nicht weiter zu kümmern. Den Namen der Methode `index` haben wir beim Erstellen des Controllers festgelegt (siehe vorherige Seite). In dieser Methode erstellen wir nun den Code für die Aufgabe 3 *Versteckspiel*.

1. Öffnen Sie die Datei `app/controllers/helloworld_controller.rb`.
2. Fügen Sie den folgenden Code ein:

```
def index
  i = 1
  @output = ""
  while i < 100 do
    @output += i.to_s + "<br>"
    i += 1
  end
  @output += "Ich komme..."
end
```

Erklärung:

› `@output`: Mit @ wird angegeben, dass es sich um eine Instanzvariable handelt. Nur so kann in der View auf diese Variable zugegriffen werden.
› `i.to_s`: Die Integer-Variable `i` muss zu einem String konvertiert werden.
› `+ "<br>"`: In HTML bedeutet
 ein Zeilenumbruch.
› Die Instanzvariable `@output` enthält nun den gesamten Output der Berechnung in HTML-Format.

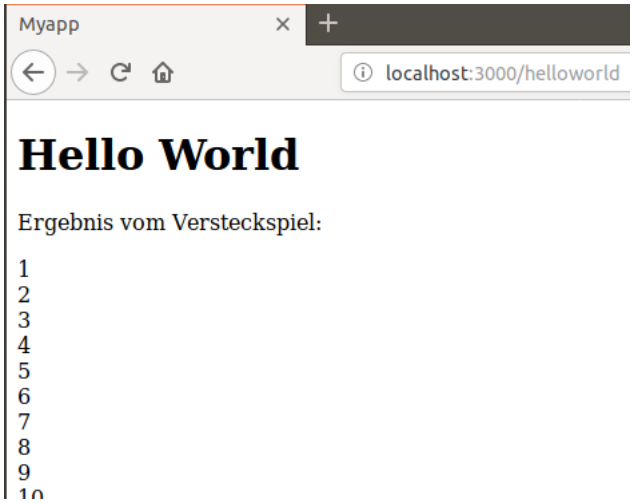
3. Nun müssen wir die Ausgabe in der View noch bewerkstelligen. Fügen Sie in `app/views/helloworld/index.html.erb` ein:

```
<h1>Hello World</h1>
<p>Ergebnis vom Versteckspiel:</p>
<p>
  <%= simple_format(@output) %>
</p>
```

Erklärung:

› `<%= … %>`: Mit dieser Markierung wird eine Ausgabe in HTML gemacht, z.B. der Inhalt von Variablen wird so ausgegeben.

› `simple_format()`: HTML-Sonderzeichen wie "<" und ">" werden normalerweise maskiert. Mit `simple_format` wird das verhindert, so dass `<br>` korrekt als Zeilenumbruch dargestellt wird.



So sieht das Ergebnis im Browser aus. Ganz unten steht noch "Ich komme…".

Zusatzaufgabe

Erstellen Sie für alle andern Aufgaben, die Sie oben erledigt haben, eine Version im Helloworld-Controller und geben Sie die Ergebnisse in der View aus.

Starten/Stoppen des Rails-Servers

Damit Sie beim nächsten Systemneustart wieder auf die Seite http://localhost:3000 gelangen können, müssen Sie den Rails-Server erneut starten:

```
$ cd ~/workspace/myapp
$ rails s
```

Der Server kann mit Hilfe seiner ProzessID wieder beendet werden. Diese wird direkt in unserem Applikationsverzeichnis gespeichert, nämlich in der Datei `home/vmadmin/workspace/myapp/tmp/pids/server.pid`

Jeder laufende Prozess unter Linux besitzt eine solche ProzessID und kann mit dem Kommando

```
$ sudo kill <ProzessID>
```

beendet werden. Damit wir nicht immer die ProzessID des Rails-Servers für das kill-Kommando auslesen müssen, bietet sich folgende Konsoleneingabe an:

```
$ sudo kill $( cat /home/vmadmin/workspace/myapp/tmp/pids/server.pid )
```

Der Server muss immer nach Änderungen an Javascript und CSS Dateien neu gestartet werden. In Rails sind Javascript und CSS-Dateien Assets genannt. Statt `css` verwendet Rails `scss`. SCSS-Dateien können gegenüber CSS auch Schleifen und Variablen behandeln.

Praktische Arbeit: Beim Systemstart den Rails-Server starten

Damit wir nicht ständig den Rails-Server manuell starten müssen, macht es durchaus Sinn, den Start bei einem Systemboot zu automatisieren. Es gibt mehrere Lösungsansätze unter Linux. Finden Sie eine funktionierende Lösung und testen Sie diese!

Beachten Sie insbesondere folgende Punkte:

1. Sie wissen, dass wir den Rails-Server mit `rails s` starten. Damit der Befehl abgesetzt werden kann, müssen Sie sich im Ordner der rails app befinden (`cd /home/vmadmin/workspace/myapp`).
2. `rails s` ist kein bekanntes Linux Kommando. Es wurde bei der Installation von Rails als Skript im Ruby Version Manager (`rvm`) hinterlegt (`source ~/.rvm/scripts/rvm && rails s`).
3. Mit `&&` können Sie mehrere Linuxkommandos als Einheit ausführen.
   Zum Beispiel:

   ```
   $ cd /home/vmadmin/workspace/myapp && source ~/.rvm/scripts/rvm
   && rails s
   ```

4. Machen Sie mehrmals folgende Kommandozeileneingaben und probieren Sie zu verstehen, was passiert:

```
$ sudo kill $( cat /home/vmadmin/workspace/myapp/tmp/pids/server.pid )
$ cd /home/vmadmin/workspace/myapp && source ~/.rvm/scripts/rvm && rails s
```

5. Sie sollten das Startkommando des Rails-Servers für unsere App in ein Bash Skript verpacken und dieses mit `gnome-session-properties` zur Ausführung bringen.

   Lesen Sie hierzu: https://wiki.ubuntuusers.de/Shell/Bash-Skripting-Guide_für_Anfänger/

Falls Sie nicht weiterkommen, wenden Sie sich an die Lehrperson.