

# **Linux-Fortbildung**

## **Teil V**

# **Server**

## **Teil b**

IFB-Nr. 19.288  
22. – 23. Januar 2002  
IFB Haus Speyer

Daniel Jonietz  
([daniel@jonietz.de](mailto:daniel@jonietz.de))

## Literatur-Vorschläge

### einleitend:

- Günther, K.: **Linux, die Kurz-Referenz. Linux ge-packt**  
MITP-Verlag, Bonn 2000.  
ISBN 3-8266-0594-2 (EUR 12,68)
- Siever, E.; Spainhour, St.; Figgins, St.; Hekman, J. P.: **Linux in a nutshell**  
Deutsche Ausgabe  
O'Reilly Verlag, Köln <sup>3</sup>2001.  
ISBN 3-89721-195-5 (DM 69,-)

### vertiefend:

- Newham, C.; Rosenblatt, B.: **Lerning the bash Shell**  
O'Reilly & Associates, USA <sup>2</sup>1998.  
ISBN 1-56592-347-2 (\$ 24.95)
- Krienke, R.: **UNIX Shell-Programmierung**  
Carl Hanser Verlag, München <sup>2</sup>2001.  
ISBN 3-445-21722-3 (EUR 19,90)
- Rathmann, M.; Wieskotten C.: **Jetzt lerne ich Shell-Programmierung**  
Markt+Technik Verlag, München <sup>2</sup>2002.  
ISBN 3-8272-6241-2 (EUR 24,95)
- Friedl, J.F.F: **Reguläre Ausdrücke**  
O'Reilly Verlag, Köln 1998.  
ISBN 3-930673-62-2 (DM 59,-)

## Vorbemerkung

Dieses Skript stellt keinen Anspruch auf Vollständigkeit — an vielen Stellen sind bewußt Leerstellen angebracht. Es soll lediglich ein Skelett für eigene Notizen darstellen und die Nachbereitung vereinfachen.

## Einführung

Nachdem im Kurs „Linux Administration“ die wesentlichen Grundlagen der Shell-Programmierung gelegt wurden, sollen diese zunächst knapp wiederholt und anschließend weiter ausgebaut werden. Nebenbei werden noch einige recht nützliche Merkmale der `bash` angesprochen, die nicht nur im Zusammenhang mit der Programmierung von Nutzen sein können.

### 1 Der Prompt

Die Gestalt der Eingabeaufforderung kann selbstverständlich modifiziert werden. Dabei sind vier verschiedene Prompts vorgesehen: `PS1`, `PS2`, `PS3` und `PS4`, wovon im Moment für uns nur `PS1` und `PS2` interessant sind. Der Prompt `PS1` ist derjenige, der während der interaktiven Eingaben direkt von der Shell angezeigt wird. `PS2` erscheint als Fortsetzungsprompt, wenn die Eingabe noch nicht abgeschlossen ist.

Die Manipulation des Prompt-Erscheinungsbildes geschieht über Zuweisung von Zeichenketten an die entsprechende Variablen. Folgende Zeichen haben dabei eine besondere Bedeutung:

Formatzeichen	Bedeutung
<code>\d</code>	Datum
<code>\h</code>	Rechnername
<code>\j</code>	Anzahl der Jobs
<code>\s</code>	Name der aktuellen Shell
<code>\t</code>	Zeit im 24h-Format
<code>\T</code>	Zeit im 12h-Format
<code>\!</code>	Nummer des aktuellen Befehls in der History
<code>\W</code>	aktuelles Verzeichnis
<code>\\$</code>	zeigt ein <code>\$</code> -Zeichen für normale Benutzer und ein <code>#</code> -Zeichen für <code>root</code>
<code>\u</code>	Benutzername (Login-Name)

Die Zuweisung

```
PS1='\u@\h: [\W]>'
```

ergibt z.B. den Prompt

```
jonietz@aixd2: [VServer]>
```

## 2 Kommando-History

Die Anzeige der aktuellen Befehlsnummer im Prompt ergibt durchaus Sinn: So lassen sich alte Befehle über die History-Funktionalität wiederverwenden. Der Befehl `!nummer` führt den unter dieser Nummer gespeicherten Befehl erneut aus. Ebenso kann durch Angabe von `!zeichen` der letzte Befehl wiederholt werden, der mit `zeichen` begann: `!PS` wiederholt die oben durchgeführte Zuweisung des Formatierungsstrings an die Variable `PS1`. Eine Liste der bereits verwendeten Befehle kann mit `history` manipuliert und ausgegeben werden.

## 3 Shell-Programmierung

### 3.1 Grundlegendes

Ein Shell-Skript ist eine reguläre Textdatei, die

- in der ersten Zeile einen Pseudokommentar enthält, der angibt mit welcher Shell-Variante das Skript ausgeführt werden soll:  
`#!/bin/bash`
- in den weiteren Zeilen beliebige Shellkonstrukte, externe Kommandos (Programmaufrufe) oder durch `#` eingeleitete Kommentare enthält
- ausführbar ist.

Hinweis: Der Suchpfad des Superusers `root` enthält i.d.R. nicht das aktuelle Verzeichnis, daher sind Shell-Skripte im aktuellen Verzeichnis mit absolutem oder relativem Pfadanteil aufzurufen:

```
./skriptname
```

### 3.2 Schleifen

- `while kommandoliste; do`  
    `kommandos;`  
    `done`
- `until kommandoliste; do`  
    `kommandos;`  
    `done`
- `for i in liste; do`  
    `kommandos;`  
    `done`

- ```
for (( i=0; i<11; i++ )); do
    kommandos;
done
```

### 3.3 Bedingungen

Bedingungen werden über den Rückgabewert des Kommandos `test` geprüft. Läßt sich die Bedingung zu `TRUE` auswerten, so liefert `test` den Rückgabewert 0, den wir als `TRUE` interpretieren. Der Aufruf von bspws.

```
test 10 -eq 11
```

läßt sich zu

```
[ 10 -eq 11 ]
```

verkürzen.

### 3.4 Verzweigungen

Das `if`-Konstrukt bedient sich wieder des Rückgriffs auf die Rückgabewerte entsprechender Kommandos:

```
if kommando1; then
    kommando2;
else
    kommando3;
fi
```

Natürlich kann auch hier das Kommando `test` oder `[ ]` zur Überprüfung entsprechender Bedingungen zum Einsatz kommen.

Nützlich ist in diesem Zusammenhang auch die Existenz eines „leeren“ Kommandos entsprechend einem `nop`, das durch den Doppelpunkt `:` repräsentiert wird. Der Versuch:

```
if kommando1; then
else
    kommando3;
fi
```

scheitert, da die Shell sich über die Syntax mokiert. Abhilfe schafft:

```
if kommando1; then
:
else
kommando3;
fi
```

## 4 Ergänzungen zur Shell-Programmierung

### 4.1 Variable

Durch jede Anweisung der Form `NAME=irgendetwas` wird versucht einer Variablen namens `NAME` den Wert `irgendetwas` zuzuweisen. Existiert eine solche Variable bereits, geschieht die Zuweisung direkt, solange der Typ der Variable paßt und sie nicht schreibgeschützt ist. Existiert keine Variable dieses Namens (Groß- und Kleinschreibung ist relevant!) so wird eine untypisierte Variable angelegt und ihr der Wert `irgendetwas` zugewiesen. Der Zugriff auf eine nichtexistente Variable stellt keinen Fehler dar!

Beim Zugriff auf Variableninhalte sind diese mit einem führenden `$`-Zeichen zu kennzeichnen. Von besonderer Bedeutung ist die Variable `?`, in der der Rückgabewert des letzten Kommandos gespeichert ist, er läßt sich mittels `echo $?` ausgeben.

#### 4.1.1 Fehlerfälle

Den Fall, dass eine Variable nicht existiert, können wir bereits im positiven Sinne abfangen, indem wir einen Standardwert verwenden. Soll jedoch das Skript beim Zugriff auf eine nichtexistente Variable abbrechen — einen Fehler liefern, so kann dies folgendermaßen geschehen:

```
${variable:?Fehlertext}
```

Existiert die Variable, wird ihr Wert zurückgegeben, ansonsten bricht das Kommando mit der durch *Fehlertext* definierten Fehlermeldung ab.

#### 4.1.2 Manipulation bei der Auswertung

Angenommen einem Skript soll ein Dateiname (`test.zip`) übergeben werden; Im Skript wird er jedoch ohne Endung benötigt (`test`). Bei der Auswertung kann dies einfach durch Abschneiden von rechts erreicht werden:

```
${VAR%.tex}  
liefert test
```

Analog können Variablenwerte von links beschnitten werden, indem anstelle des `%` ein `#` verwendet wird. Soll nicht nur ein minimaler Treffer gesucht und entfernt werden, sondern gemäß der Strategie „so viel wie möglich“, so können die Operatorzeichen `%` bzw. `#` einfach doppelt angegeben werden:

```
TEST="abcdabab"
${TEST%ab} liefert abcdab
${TEST%%ab} liefert abcd
```

Auch auf einzelne Bereiche eines Variablenwertes kann zugegriffen werden, die Syntax lautet:

```
${Variablenname: von}
```

bzw.

```
${Variablenname: von: bis}
```

Sollen einzelne Bestandteile ersetzt werden, so ist das auch möglich:

```
${variable/muster/ersatz}
ersetzt die erste auf muster passende Stellen in variable durch ersatz
${variable//muster/ersatz}
ersetzt entsprechend alle passenden Stellen
```

Dabei darf *muster* auch Metazeichen wie \* und ? enthalten.

## 4.2 Here-Documents

Here-Documents sind eine Erweiterung der Eingabeumleitungen, die es ermöglichen die Shell anzuweisen die Standardeingabe *aus dem aktuellen Skript* zu lesen. Damit wird es möglich, interaktive Programme sozusagen von vorne herein mit Daten zu füttern. Dazu wird ein Begrenzer festgelegt; sobald er als Zeile in der Standardeingabe auftaucht beendet die Shell diesen Eingabekanal und interpretiert die weiteren Zeilen des Skriptes wieder als Befehle. Üblich ist der Begrenzer EOF (end-of-file). Operator ist die Kombination <<. Das im folgenden Text erscheinende >-Zeichen braucht nicht mit eingegeben zu werden, die Shell erzeugt es automatisch denn es handelt sich um den Fortsetzungsprompt PS2.

```
cat <<EOF
> erste Zeile
> zweite Zeile
> etc.
> EOF
```

behandelt die Zeilen *erste Zeile* bis *etc.* als Eingabe im Standardeingabekanal (als würde jemand zur Laufzeit die Zeilen über die Tastatur eingeben) des Programmes *cat* welches die Eingabe, wie wir wissen, einfach wieder über den Standardausgabekanal ausgibt:

```
erste Zeile
zweite Zeile
etc.
```

Dieses einfache Beispiel mag nicht sehr verlockend erscheinen, doch folgende Anwendung wird häufig benötigt (birgt aber, um es gleich vorzuschicken, eine Gefahr: Passwörter im Klartext!)

#### 4.2.1 Automatischer ftp-Upload

Standard `ftp` verlangt i.d.R. über Benutzerinteraktion, zur Laufzeit muss das Passwort eingegeben werden und festgelegt werden, welche Dateien hochgeladen werden sollen. Diese Eingaben können durch here-documents fest in ein Skript kodiert werden — die Shell erledigt später für uns die Umleitung in den Eingabekanal des `ftp`-Programms. Dabei unterdrückt die `ftp`-Option `-n` die automatische Anmeldung am System.

```
ftp -n <<EOF
> open ftp.uni-kl.de
> user anonymous gast
> pwd
> ls
> quit
> EOF
```

Das ganze läßt sich dann in ein kleines Skript verpacken und automatisch starten.

### 4.3 Funktionen

Syntax:

```
function name()
{
    befehle;
}
```

Aufruf:

```
name parameter
```

Dabei wird also aus der Schreibweise `name(parameter)` der Aufruf `name parameter`.

Auf den übergebenen Parameter kann (wie von Skripten her gewohnt) über die Variable `$1` zugegriffen werden.



## 5 Automatische zeitgesteuerte Ausführung

Häufig sollen bestimmte Aufgaben regelmäßig ausgeführt werden. Dazu gibt es unter Linux den `crontab`-Mechanismus. Soll eine bestimmte Aufgabe zwar nicht regelmäßig, aber zu einem bestimmten Zeitpunkt (z.B. „jetzt in zwei Stunden“) erledigt werden, kann `at` bemüht werden. Beide Pakete bedienen sich eigener Dämonen, die für die eigentliche Ausführung verantwortlich zeichnen, während die Kommandos `crontab` und `at`, die hier besprochen werden sollen, eigentlich nur die Benutzerschnittstelle darstellen.

### 5.1 crontab

Die `crontab` ist — wie der Name schon richtig vermuten läßt — im Wesentlichen eine Tabelle in der die Zuordnung von Ausführungszeiten und zugehörigen Kommandos verwaltet wird. Jede Zeile entspricht einem auszuführenden Kommando, über die Spalten wird der Zeitpunkt und das eigentlich auszuführende Programm festgelegt. Jeder Benutzer hat eine eigene `crontab`, der Superuser `root` kann aber alle lesen, ändern und löschen.

#### 5.1.1 Optionen im Überblick

| Option         | Bedeutung                                                                                                                                       |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>u</code> | Angabe eines Benutzernamens (z.B. <code>-u uucp</code> )                                                                                        |
| <code>l</code> | Zeigt die <code>crontab</code> an (von <code>list</code> )                                                                                      |
| <code>e</code> | Startet einen Editor (den <code>vi!</code> ) und trägt nach Abschluß der Änderungen die neue <code>crontab</code> in eine interne Datenbank ein |
| <code>r</code> | Löscht die <code>crontab</code> komplett.                                                                                                       |

#### 5.1.2 Bedeutung der Spalten

Spaltentrennzeichen sind „weiße Zeichen“, also Leerzeichen und Tabulatorzeichen. Die nächste Spalte wird durch das Auftauchen von mindestens einem solchen Zeichen erkannt.

| Spalte | Bedeutung     | Beispiel                             |
|--------|---------------|--------------------------------------|
| 1      | Minute        | 0,30      alle halbe Stunde          |
| 2      | Stunde        | 6-10      um 6, 7, 8, 9 und 10 Uhr   |
| 3      | Datum (Tag)   | 1          am Monatsersten           |
| 4      | Datum (Monat) | *          jeden Monats              |
| 5      | Wochentag     | 0,6 <b>und</b> Samstags und Sonntags |
| 6      | Kommando      | /pfad/kommando                       |

Der Tag wird durch zwei Merkmale bestimmt, zum einen durch den Tag des Monats (Datum), zum anderen durch den Tag der Woche. Sobald *eines* dieser Merkmale zutrifft, wird das Kommando ausgeführt! Soll nur einer wirken, muss der andere Eintrag mit einem \* versehen werden.

Es kann übrigens eingestellt werden, ob alle Benutzer eigene `crontabs` anlegen dürfen: `cron.allow` und `cron.deny`.

## 5.2 at

Das Kommando `at` nimmt im Wesentlichen die Aufträge vom Benutzer über den Standardeingabekanal entgegen. Soll später einer der Aufträge doch nicht ausgeführt sondern gelöscht werden, helfen `atq` und `atrm`. Der Zeitpunkt wird in Form eines Parameters mitgegeben:

```
at teatime
> date +%T
```

gibt um 16.00Uhr die aktuelle Uhrzeit aus. Dabei wird die Ausgabe des Kommandos dem Benutzer per eMail zugestellt (er muss ja zum Ausführungszeitpunkt nicht mehr am System angemeldet sein). Natürlich kann die Ausgabe mit den üblichen Mechanismen umgeleitet werden.

### 5.2.1 Zeitformate

| Format            | Bemerkung                                                                              |
|-------------------|----------------------------------------------------------------------------------------|
| HH:MM             |                                                                                        |
| midnight          | um 24.00Uhr                                                                            |
| noon              | um 12.00Uhr                                                                            |
| teatime           | um 16.00Uhr                                                                            |
| MDDYY             | nur zusammen mit einer Zeitangabe                                                      |
| DD.MM.YY          | dto.                                                                                   |
| MM/DD/YY          | dto.                                                                                   |
| today             | dto.                                                                                   |
| tomorrow          | dto.                                                                                   |
| now + <i>zeit</i> | Beispiel: now + 1 day<br>Gültige Zeiteinheiten:<br>minute(s), hour(s), day(s), week(s) |

## 6 Nicht-interactive Bearbeitung von Dateien

### 6.1 Muster

Die Metazeichen zur Unterstützung der Suche von Mustern haben wir schon im Zusammenhang mit `grep` kennengelernt, deswegen hier nur noch eine Zusammenstellung:

| Zeichen | Bedeutung                                               |
|---------|---------------------------------------------------------|
| .       | genau ein beliebiges Zeichen                            |
| *       | beliebig viele Vorkommen                                |
| +       | mindestens ein Vorkommen (nur <code>grep</code> )       |
| ?       | höchstens ein Vorkommen (nur <code>grep</code> )        |
| $\{n\}$ | $n$ -maliges Vorkommen                                  |
| ^       | Zeilenanfang                                            |
| \$      | Zeilenende                                              |
| [ ]     | jedes beliebige Zeichen innerhalb der eckigen Klammer   |
| [^ ]    | jedes beliebige Zeichen, das nicht in der Klammer steht |
| \<      | Wortanfang (nur <code>vi</code> und <code>sed</code> )  |
| \>      | Wortende (nur <code>vi</code> und <code>sed</code> )    |

Kommt ein Bindestrich - innerhalb einer Zeichenklasse vor und ist er nicht das erste Zeichen der Zeichenklasse, so kennzeichnet er einen Zeichenbereich. Als erstes Zeichen ist er normales Element:

[a-z]  
enthält alle Kleinbuchstaben

[-az]  
enthält nur die Zeichen -, a und z

### 6.2 tr

Häufig müssen einzelne Zeichenklassen umgewandelt werden, sei es um Umlaute aus einer Datei zu entfernen, sei es um die Gross-/Kleinschreibung anzupassen. Das Kommando `tr` (für translate characters) kann das Gewünschte leisten.

Dabei liest `tr` immer aus der Standardeingabe und schreibt immer auf den Standardausgabekanal und erwartet als Optionen *was* zu tun ist. Durch zwei weitere Parameter lassen sich Einschränkungen auf dem Befehl definieren. Dabei bewirkt die Option `-d` das Löschen aller Zeichen aus dem Eingabestrom, die im als Parameter übergebenen Zeichenbereich vorkommen.

Zur Vereinfachung der Eingabe sind einige Zeichenklassen vordefiniert worden:

| Klasse                 | Bedeutung       |
|------------------------|-----------------|
| <code>[:digit:]</code> | Ziffern         |
| <code>[:lower:]</code> | Kleinbuchstaben |
| <code>[:upper:]</code> | Großbuchstaben  |
| <code>[:alpha:]</code> | Buchstaben      |

Der Befehl

```
cat testdatei | tr [:upper:] [:lower:]
```

wandelt alle Großbuchstaben in die entsprechenden Kleinbuchstaben um.

## 6.3 sed

Normalerweise interagieren Editoren zur Laufzeit mit dem Benutzer. Der Benutzer tippt auf die Pfeiltasten, und der Editor folgt durch Nachrücken der Editormarke (Cursor). Der Editor `sed` (stream editor) erwartet die Festlegung der durchzuführen Aktionen bereits im Vorfeld. Dann liest er die Eingabedatei(en) *zeilenweise* ein, wendet die bereits spezifizierten Befehle auf diese Zeile an und gibt ggf. die überarbeitete Zeile aus. Durch diese Vorgehensweise sind der Größe der zu bearbeitenden Datei fast keine Grenzen gesetzt. `sed` kann die Quelldatei entweder direkt lesen, oder aber über den Eingabekanal beziehen. Die Ausgabe erfolgt immer auf die Standardausgabe, die bei Bedarf umgeleitet werden muss. Die auszuführenden Befehle können entweder direkt in der Kommandozeile angegeben werden, oder in einer eigenen Datei (Skript) zusammengefaßt werden, falls sie umfangreicher sein sollten.

### 6.3.1 Wichtige Optionen

| Option          | Bedeutung                                 |
|-----------------|-------------------------------------------|
| <code>-e</code> | der nächste Befehl ist ein Editierbefehl  |
| <code>-f</code> | das nächste Argument ist eine Skriptdatei |
| <code>-n</code> | unterdrückt die normale Ausgabe           |

### 6.3.2 Grundlegende Befehle

| Befehl               | Bedeutung                            |
|----------------------|--------------------------------------|
| <code>d</code>       | Zeile löschen                        |
| <code>s</code>       | Ersetzung vornehmen                  |
| <code>y</code>       | Zeichen umwandeln                    |
| <code>=</code>       | drucke die aktuelle Zeilennummer aus |
| <code>p</code>       | drucke die aktuelle Zeile aus        |
| <code>r datei</code> | Gib den Inhalt von <i>datei</i> aus  |

Bevor ein Befehl ausgeführt werden kann, muss noch spezifiziert werden, *wo* er ausgeführt werden soll. Beispielsweise muss die zu löschende Zeile genannt werden. Bei dieser Adressierung muss unterschieden werden zwischen festen Adressen (Zeilennummern) und Adressen die sich durch die Überprüfung einer Bedingung ergeben (Lösche alle Zeilen in denen der Begriff „Regen“ vorkommt!). Somit setzt sich der eigentliche Editierbefehl zusammen aus einer Adresskennzeichnung und den dort anzuwendenden Operationen.

### 6.3.3 Einfache Adressen

| Adresse | Bedeutung                                                      |
|---------|----------------------------------------------------------------|
| \$      | letzte Zeile der Datei                                         |
| 3       | Zeile 3                                                        |
| 3,7     | Zeilen 3 und 7                                                 |
| 3-7     | Zeilen 3 bis 7                                                 |
| /Regen/ | alle Zeilen in denen Regen steht                               |
| /regex/ | alle Zeilen die auf den Regulären Ausdruck <i>regex</i> passen |

Adressen dürfen leer sein, dann wird die jeweilige Operation auf alle Zeilen angewendet.

```
sed -n -e '$=' test
zeigt die Zeilennummer der letzten Zeile der Datei test an

sed -n -e '/Linux/p' test
gibt alle Zeilen der Datei test aus, die den Begriff Linux enthalten

sed -e '2,7d' test
gibt alle Zeilen außer der 2ten und 7ten aus

sed -n -e '/Windows/r alarm.txt' test
zeigt den Inhalt der Datei alarm.txt für jede Zeile der Datei test an,
in der der Begriff Windows gefunden wird
```

### 6.3.4 Textersetzungen

Der Editierbefehl `s` (substitute) erwartet drei Angaben:

- was soll ersetzt werden? → */Suchbegriff/*
- wodurch soll es ersetzt werden? → *Ersatztext/*
- wie soll es ersetzt werden? → *flag*

Dabei kann das zu setzende *flag* folgende Werte haben:

| Flag                  | Bedeutung                                          |
|-----------------------|----------------------------------------------------|
| <b>g</b>              | Global: alle Vorkommen in dieser Zeile ersetzen    |
| <i>n</i>              | das <i>n</i> te Vorkommen in dieser Zeile ersetzen |
| <b>p</b>              | Print: Zeilen nach der Ersetzung ausdrucken        |
| <b>w</b> <i>datei</i> | Write: Schreibe die Zeile in <i>datei</i>          |

Es können mehrere Flags miteinander kombiniert werden.

Kommt im Ersatztext ein Ampersand & vor, so wird dieses durch den vom Suchmuster getroffenen Text ersetzt.

```
sed -e 's/Linux/"&"/g' test
ersetzt alle Vorkommen von Linux durch "Linux".
```

### 6.3.5 Zeichenersetzungen

Der Editierbefehl `y` erwartet zwei gleich große Zeichenklassen, er bildet jedes Zeichen der ersten Klasse auf das entsprechende Zeichen der zweiten Klasse ab:

```
sed -n -e 'y/abcd/ABCD/ ; p' test
ändert für jede Zeile jedes Zeichen a zu einem A, jedes b zu einem B, ...
und druckt dann die Zeile aus
```

## 6.4 Anwendung

Die vorgestellten Techniken eignen sich besonders gut zur dynamischen Erzeugung von Konfigurationsdateien. Beispielsweise läßt sich auf diese Art und Weise für jeden Benutzer des System automatisch und ohne manuelle Eingriffe eine Profil-Datei für den Internetzugang erstellen. Dies soll am Beispiel des Kommunikationspaketes Opera kurz erörtert werden.

Opera läßt sich vollständig über Textdateien konfigurieren. Nimmt man also *einmal* die erforderlichen Einstellungen vor und speichert diese ab, können die so hergestellten Konfigurationsdateien als Vorlagen dienen. Manuell werden nun alle Stellen herausgesucht, an denen wirklich benutzerabhängige Informationen stehen (i.d.R. sind dies eMail-Adresse, Benutzername, ...) und durch festzulegende Mustertexte ersetzt, bswp. lautet die eMail-Adresse nach der Ersetzung einfach `__EMAIL__`. Bei der (erstmaligen) Anmeldung jeden Nutzers kann dann vollautomatisch ein Skript aufgerufen werden, das folgende Schritte unternimmt:

- suche die benötigten Benutzerinformationen zusammen
- konstruiere die erforderlichen Bestandteile in Variablen

- ersetze mit Hilfe von `sed` und ggf. anderen geeigneten Werkzeugen die einfach zu identifizierenden Mustertexte durch die konstruierten Werte
- speichere die so entstandene individuelle Konfigurationsdatei im Benutzerheimatverzeichnis