Erste Schritte der Shell-Programmierung

Der Einstieg ist gar nicht so schwer

DIPL.-ING. THOMAS DRILLING

Auch Linux-Einsteiger können von etwas Basiswissen im Shell-Handling profitieren. Virtuosität muß nicht sein, aber Sicherheit im Umgang mit der bash öffnet so manche Tür.

Auszug

Man kann ihnen gar nicht entkommen. Sicher haben auch Sie als Einsteiger bereits bemerkt, daß Shell-Skripte in der Linux-Welt und insbesondere in der Systemadministration eine zentrale Rolle spielen. Eine beträchtliche Anzahl von Routine-Aufgaben im Anwender-Alltag und darüber hinaus beinahe sämtliche Systemadministrations-Aufgaben können und werden bei Linux nach wie vor über die Shell-Steuerung erledigt. Da die verschiedenen für Linux einsetzbaren Kommando-Shells über eine eigene Steuersprache zur Ablaufsteuerung mehrerer Shell-Befehle verfügen (sogenannte Shell-Skripte), läßt sich damit ein Großteil der Systemwartungsaufgaben elegant automatisieren. Shell-Skripte begegnen Ihnen außerdem bei fast allen Installations-Vorhaben.

Shell-Skripte und Shell-Guide

Installations-Programme bestehen bei Linux fast ausnahmslos aus Shell-Skripten. Im Grunde genommen verschwimmen bei Linux die Grenzen zwischen echten, also Kommando-Interpreter-internen Befehlen, und externen Befehlen (Programmen), die ihrerseits sehr oft aber nicht immer aus Shell-Skripten bestehen. Keine Angst. Dies ist kein Porgrammier-Crashkurs. Wer heutzutage als Anwender oder Sysad derartige Ambitionen wirklich noch hegt, hat zwar bei einer einfache Skript-Sprache durchaus die größten Aussichten auf Erfolgserlebnisse, allerdings kann Ihnen dabei ein Artikel auch nicht helfen. Hierbei brauchen Sie in der Tat ein praktisches Betätigungsfeld. Ich möchte Ihnen mit diesem Artikel lediglich einen Einblick in Objekte und Verfahren eröffnen, die auch dem Einsteiger zwangsläufig im Umgang mit Linux begegnen. Mit etwas Verständnis für die Funktionsweise der Shell, werden Ihnen viele Dinge im Linux-Alltag etwas durchsichtiger erscheinen. Vielleicht bekommen sie aber auch Appetit auf mehr.

Was bedeutet eigentlich bash?

Bei Linux ist der Anwender nicht an die Verwendung einer bestimmten Shell oder Shellsprache, wie etwa der COMMAND.COM für MS-DOS, gebunden. Es gibt verschiedene populäre Kommando-Interpreter (Shells) für Unix/Linux mit jeweils spezifischen Vor- und Nachteilen. Bei den gängigen Linux-Distributionen wird die bash (»bourne again shell«) favorisiert. Die bash wird bei jedem Anmeldeprozeß (Login) für den Anmelder spezifisch geladen (/bin/bash). Die Shell (bei Linux die bash), wird für folgende Aufgaben benötigt:

Als textbasierte Benutzer-Schnittstelle zwischen Linux-Anwender und Betriebssystem. Sie ist in dieser Funktion für den Start von Linux-Programmen und/oder Linux-Kommandos zuständig.

Zur Bereitstellung einer einfachen, interpretierten Programmiersprache mit leistungsfähigen Befehlen zur Ablaufsteuerung (bedingte Ausführung, Schleifen, Verzweigungen und Variablen) und Automatisierung mehrerer Shell-Kommandos.

Bereitstellung shell-interner Befehle und Mechanismen zur Filterung von Daten und Steuerung von Datenströmen (Filter, Ausdrücke, Expansionen etc.)

Die bash wird beim Benutzer zu dessen Default-Shell durch einen Eintrag in der Benutzerdatenbank /etc/passwd. Unter /bin finden sich noch weitere Shells, wie

/bin/sh, /bin/tcsh, /bin/csh, /bin/bash. Der Eintrag /bin/sh ist immer ein symbolischer Link auf die tatsächlich verwendete Shell, meist bash.

Befehlseingabe und Befehlsexpansion der bash

Jeder Anwender kommt automatisch mit der bash in Kontakt, sobald Befehle eingegeben werden müssen Um dem Benutzer Tipparbeit zu ersparen, beherrscht die bash einige Mechanismen zur Eingabe-Erleichterung wie beispielsweise die Befehls-Expansion und einen Befehlsspeicher. Mit den Cursortasten können die jeweils zuletzt benutzen Befehle innerhalb einer bash-Sitzung zurück geholt werden. Der bash-Befehlsspeicher ist in einem Ringpuffer organisiert. Die Tastenbelegung der bash ist außerdem weitgehend benutzerspezifisch konfigurierbar. Die entsprechende Konfigurationsdatei findet sich unter ~/.inputrc. Normalerweise korrespondiert die Tastenbelegung der bash mit dem bekannten Emacs-Modus. Mit der oben genannte Konfigurationsdatei kann auch in den vi-Modus umgeschaltet werden.

Besonderheit der bash

Insbesondere die letzte Zeile der Tabelle ist zu beachten. Die Expansion von Dateinamen ist eine Funktion, die nicht alle anderen Shells zu bieten haben. Das Funktionsprinzip:

Sobald Sie die ersten Zeichen eines gewünschten Dateinamens eingetippt haben, können Sie durch einen Druck auf die Tab-Taste erreichen, daß der Dateiname soweit möglich automatisch vervollständigt wird. Sollten mehrere Übereinstimmungen mit den der bash bekannten Befehle existieren, expandiert die bash soweit, wie es der vordere Namensteil noch eindeutig zuläßt. Testen Sie es. Die Eingabe der Zeichenfolge

```
les [tab]
```

bringt beispielsweise das erwartete Ergebnis less hervor.

```
em [tab]
```

expandiert zu `emacs`.

Eine andere Möglichkeit, Tipparbeit bei der Befehlseingabe zu sparen, ist die Zuweisung eines sogenannten Alias. Mit dem Befehl

```
alias aliasname <befehlspfad>
```

läßt sich für ein beliebiges Kommando eine Abkürzung definieren, wie zum Beispiel

```
alias em /bin/emacs
```

Emacs-Modus	Beschreibung
Cursor up/down	Den Puffer der zuletzt verwendeten Befehle durchsuchen
Cursor left/right	Cursor links / rechts auf der Befehlszeile bewegen
[Alt] + [B], [Alt] + [F]	Cursor wortweise links/rechts bewegen
[Alt] + [D]	Wort löschen
[Pos1]. [Ende]	Cursor zum Anfang, bzw. Ende der Eingabezeile bewegen
[Backspace}, [Entf}	Zeichen löschen, wie in emacs und anderen Editoren
[Tab]	Expansion des Befehls-/ Dateinamens

Tabelle 1

Kommandos im Hintergrund:

Linux ist ein Multitasking-Betriebssystem. Daher muß es auch Steuermechanismen geben, die Ausführung von mehreren Programmen zur gleichen Zeit zu veranlassen. Mit der Kommandozeilen-Option »&« (kaufmännisches UND) wird das betreffende Programm im Hintergrund ausgeführt, das heißt das Programm läuft zwar weiter, produziert aber keine Meldungen auf der aktuellen Shell, sondern gibt diese zur sofortigen Weiterverarbeitung mit einem anderen Kommando frei. Diese Option macht beispielsweise bei nicht interaktiven oder sehr zeitaufwendigen Kommandos Sinn, wie etwa einem Such-Kommando:

```
find /mnt/daten –name finanzamt98
```

Da die Ausgaben eines im Hintergrund laufenden Kommandos nicht mehr sichtbar und damit nutzlos sind, muß man die Befehlsausgabe in eine Datei umlenken.

```
find /mnt/daten –name finanzamt98 > ergebnis &
```

Man kann ein bereits laufendes Programm durch `[Strg]+[Z]` anhalten (nicht abbrechen) und dann mit dem Kommando »bg« in den Hintergrund schicken. Mit `[Strg]+[C]` bricht man ein laufendes Kommando ab.

Umleitung der Ein- und Ausgabe:

Die bash unterscheidet bei der Ausführung von Programmen und Kommandos die sogenannte Verarbeitungsrichtung von Ein- und Ausgabe betreffende Kanäle, wie

- **die Standardeingabe:** Aus der Standardeingabe lesen alle Linux-Kommandos ihre benötigten Eingabedaten. Default ist die Tastatur.
- **die Standardausgabe:** Alle Linuxprogramme

schreiben ihre Ausgabe auf den Standardausgabe-Kanal. Default ist der Bildschirm.

- **die Standardfehlerausgabe:** Die Fehlerausgabe wird per Default an das gleiche Terminal geschickt, wie die Nutz-Ausgaben. Um sich mehr Übersicht zu verschaffen, kann man den Fehleroutput eines Programms auch an eine andere Ausgabe, zum Beispiel zu einem anderen Terminal oder in eine Datei, schicken.

Die Operatoren zur Umlenkung der Ein- beziehungsweise Ausgabe sind »<« und »>«, beziehungsweise zur Umlenkung des Fehlerkanals »2>«. Beispiele:

Standardausgabe auf eine Datei umlenken:

```
find / ´*.doc´> /root/alle-word-texte.txt
```

findet alle Word-Dateien auf Ihrer Festplatte und sichert das Ergebnis in einer Datei.

Standardeingabe aus einer Datei lesen:

```
mail -s ´Bitte prüfen Sie diese Datei auf ↵
Fehler´ root < .fvwm95rc
```

sendet die Datei ».fvwm95rc« aus dem aktuellen Verzeichnis als Email an den Systemverwalter (root).

Fehlerausgabe auf ein anderes Terminal schicken:

```
Quatschbefehl 2> /dev/tty2
```

schickt die Fehlermeldungen des Kommandos Quatschbefehl auf das dritte Terminal.

Pipes:

Pipes dienen der Kombination mehrerer Linux-Kommandos zu einem Befehl, eine einfache Art der Automatisierung also. Mit Pipes können die Ausgaben eines Linux-Kommandos (Standardausgabe) an die Eingabe (Standardeingabe) des nachfolgenden Linux-Kommandos weitergereicht werden und so weiter. Als Verbindungsoperator dient das Zeichen »|«. Folgende Kommando-Kombination ermittelt die Prozeßnummer (PID) eines beliebigen Prozesses, beispielsweise die der bash:

```
ps aux |grep bash
```

Ein weiteres Beispiel:

Sehr gerne werden Pipes im Zusammenhang mit Textbetrachtern eingesetzt, weil beispielsweise die normale Bildschirmausgabe der bash oft nicht ausreicht. Stellen Sie sich vor, Sie wollen einen längeren Verzeichnisinhalt anzeigen. Mit less läßt es sich viel komfortabler scrollen, als mit der bash selbst (Bild auf / Bild ab)

```
ls -l | less
```

Ausführen mehrerer Kommandos gleichzeitig:

Man kann auch direkt zwei Programme gleichzeitig im Vordergrund starten. Dazu dienst das »;«

```
ls -l; whereis find
```

Kommando–Substitutionen

Bei der Kommando-Substitution handelt es sich ebenfalls um eine Methode der Kombination mehrerer Kommandos zu einem Konstrukt. Hierbei werden einzelne Kommandos allerdings nicht serialisiert, sondern geschachtelt. Kommandosubstitution bedeutet hierbei, daß innerhalb einer Befehlseingabezeile Konstruktionen vorkommen können, die vor der eigentlichen Ausführung des Linux-Kommandos ausgewertet, das heißt durch ihre »Ergebnisse« ersetzt und an das aufrufende Kommando übergeben werden. Solche Befehle im Befehl werden in innerhalb der Kommandozeile in einfache Ausrufungszeichen (Backquotes) eingefaßt.

Beispiel:

Die Konstruktion

```
rm -f ´find . -name *tmp´
```

sucht zunächst ausgehend vom aktuellen Verzeichnis (.) alle Dateien, die auf »tmp« enden und übergibt das Ergebnis an den rm-Befehl, der jede dieser Dateien löscht. Eine äquivalente Schreibweise, die außerdem noch die Schachtelung von mehr als zwei Ebenen erlaubt, lautet:

```
rm -f $( find . -name *tmp)
```

Joker einsetzen

Die sogenannten Jokerzeichen gehören eigentlich ebenfalls zur Kategorie der Kommandosubstitution, wie oben bereits angemerkt. Die bash verfügt über den bekannten Allerweltsjoker (*) hinaus, noch über eine ganze Reihe weiterer Sonderzeichen.

Beispiele:

Zeige alle Dateien, die mit a anfangen:

```
ls -l a*
```

Zeige alle Dateien, die mit a Anfangen und mindestens fünf Zeichen lang sind:

```
ls -l e????
```

Zeige alle Dateien, die mit einem Kleinbuchstaben anfangen:

```
ls -l [a-z]*
```

Sonderzeichen	Substitutions-Mechnismen
Jokerzeichen für Dateinamen-Erfassung	
*	beliebige viele beliebige Zeichen
?	genau ein beliebiges Zeichen an genau dieser Position
[x,y,z]	genau eines der Zeichen in den eckigen Klammern
[0-9] oder [f-m]	genau eines der Zeichen aus dem angegebenen Bereich
[!x,y,z]	genau keines der Zeichen in der Liste
~	Alias für das Heimatverzeichnis
.	Alias für das aktuelle Verzeichnis
..	Alias für das übergeordnete Verzeichnis

Tabelle 2

Zeige alle Dateien, die nicht mit einem Punkt beginnen:

```
ls -l [!.]*
```

Zeige alle Dateien, die mit einer Zahl enden:

```
ls -l *[0-9]
```

Suchmuster – »Regular Expressions«

Die Bildung von Suchmustern (reguläre Ausdrücke) stellt eine weitere Eigenschaft der bash dar, die für alle Linux-Kommandos gebraucht wird, die mit Zeichenketten operieren (wie grep, awk, cut). Suchmuster kann man sich als Bildungs-Schablonen für Zeichenketten vorstellen. Aus dem Vergleich einer solche Schablone mit einer realen Zeichenketten entsteht eine Zielmenge von Zeichen, auf die der betreffende Befehl wirken soll. Was sich kompliziert anhört ist ganz einfach zu handhaben. Im Ausdruck

```
grep EineZeichenkette <datei>
```

findet sich der reguläre Ausdruck »EineZeichenkette«. Dabei ist »EineZeichenkette« ein regulärer Ausdruck aus 16 Jokerzeichen (nicht zu verwechseln mit den Jokerzeichen der bash-Filterfunktionen), wobei jedes einzelne für sich selbst steht. Der reguläre Ausdruck »Ein.Zeichenkette« besteht ebenso aus 16 Jokerzeichen, wobei aber das Jokerzeichen ».« nicht für sich selbst, sondern für genau ein beliebiges Zeichen steht. Wenn Sie erreichen wollen, daß ein Jokerzeichen innerhalb von regulären Ausdrücken nicht ausgewertet (evaluiert) wird, das heißt dieses Sonderzeichen soll als normales Textzeichen gelten, muß das betreffenden Zeichen durch einen vorangestellten Backslash »\« vor der Auswertung geschützt werden.

Der Punkt ist eines der häufigste Jokerzeichen und steht also für genau ein beliebiges Zeichen. Der Stern »*« beispielsweise ist ein Jokerzeichen, daß als Wiederholungsoperator vorgesehen ist, das heißt der Ausdruck vor oder dem Stern kann beliebig oft oder gar nicht vorkommen. Wenn Sie prüfen müssen, ob die zu suchenden Zeichen an einer ganz bestimmten Stelle im Suchmuster stehen, so kann man mit den eckigen Klammern »[« »]« zur Zusammenfassung arbeiten. Das folgende Beispiel findet alle Zeilen aus der Datei <Datei>, die mit dem Buchstaben »XF86« beginnen, dann entweder den Buchstaben »S« oder »s« enthalten und mit der Zeichenkombination »etup« enden:

```
egrep ´XF86[Ss]etup´ Datei
```

Folgendes Suchmuster findet alle Zeilen in der Datei <Datei>, die mit einem Kleinbuchstaben beginnen

```
grep ´[a-z]*´ Datei
```

Variablen

Außer der Expansion von Dateinamen und Kommando-Substitutionen kann die bash mit Variablen umgehen. Variablen werden meist in Shell-Skripten oder innerhalb der bash-Umgebung selbst definiert. Durch ein vorangestelltes »$«-Zeichen greift man auf Variablen-Inhalte zu. Neben dynamisch (beispielsweise in Shell-Skripten) zugewiesenen Variablen, sind folgende Variablen per Default vorbelegt.

Durch Kenntnis des Umgangs mit Variablen kann man bereits aus dem Vorhandensein der Umgebungsvariablen nützliche Informationen gewinnen:

Variable	Verwendung
PATH	Such-Pfad für Programme, Befehle und Kommandos (meist belegt mit /bin, /usr/bin ..)
HOME	Das Homeverzeichnis des aktuellen Benutzers
USER	Der aktuelle (eigene) Benutzername
SHELL	Die aktuell verwendete Shell, meist /bin/sh, wobei bei allen gängigen Linux-Distributiionen sh ein symbolischer Link auf /bin/bash ist. In vielen Shellskripten, wird auf /bin/sh bezug genommen
LOGNAME	Der aktuelle Login-Name
PS1	Definition des PROMPT-Erscheinungsbildes

Tabelle 3