Alias-Definition in der Bash

MATTHIAS LEH

Der Begriff »alias« wird in der EDV in mehrfacher Hinsicht verwendet. Dieser Beitrag dreht sich ausschließlich um die Möglichkeit, Befehle oder Befehlsketten durch ein Schlüsselwort zu verkürzen. Es geht damit um die immer wieder aktuelle Frage: »Wie kann ich mir das Leben leichter machen?«

Ein »Alias« ist bei den Kommandozeileninterpretern der Unix-Derivate ein sogenanntes Shell-Builtin-Kommando. Der Befehl *alias* ist damit ein integraler Bestandteil der Shell und kein externes Kommando. Ein Programm *alias* sucht man deshalb im Dateisystem vergebens. Wird der Befehl aufgerufen, arbeitet beispielsweise die Bash – um die Verwendung des Befehls in dieser Shell handelt dieser Beitrag – die dem Befehl zugeordneten Kommandos ab. Alias-Befehle muß man sich selbst definieren, einige Kürzel sind bei verschiedenen Linux-Distributionen bereits vordefiniert. Grundsätzlich erfolgt die Einrichtung nach dem Muster

```
alias Schlüsselwort='Befehl / Befehle'
```

Wobei der Teil rechts vom Schlüsselwort, wenn der Leerzeichen enthält, gequotet werden muß. Ein einfaches Beispiel soll dies verdeutlichen. Ein

```
alias dir='ls -l'
```

an der Kommandozeile eingegeben, ergibt als Ergebnis das Auflisten des Verzeichnisinhalts. Der Befehl *dir* ist jedem, der aus der DOS-Welt nach Linux umgestiegen ist, hinlänglich bekannt. Das Beispiel wird in der obigen Form vorkonfiguriert beispielsweise auf meinem Linux-Rechner (SuSE 7.3) realisiert, um Umsteigern das Leben etwas bequemer zu machen. Unter FreeBSD ist*dir* ebenfalls vorkonfiguriert, allerdings in der folgenden Form:

```
dir ()
{
  ls -la | more
}
```

Hier handelt es sich also um die Implementation in Form einer Funktion. Im Ablauf ergibt der neue dir-Befehl unter Linux eine nicht unterbrochene Auflistung des Verzeichnisinhalts, während unter FreeBSD aufgrund der Pipeline mit *more* eine seitenweise Anzeige erfolgt. Beide Versionen sind funktional eigentlich gleichwertig, auf jeden Fall dann, wenn man das erste Beispiel um den Umleitung nach *more* erweitert:

```
alias dir='ls -la | more'
```

Da es Shell-Varianten gibt, die den Befehl *alias* nicht kennen, beispielsweise die Bourne Shell und die ash, ist das Ausweichen auf Funktionen eine Möglichkeit zur Problemlösung. In diesem Beitrag werden nur Muster nach der ersten Variante eingesetzt.

Alias löschen

Aliase können jederzeit nach dem oben beschriebenen Muster eingegeben werden. Sie wirken dann allerdings nur zur Laufzeit des Logins und auch nur in der Login-Shell, nicht in Subshells. Vorhandene Aliase lassen sich mit dem Kommandozeilenbefehl *alias* ohne weitere Parameter anzeigen. Das hat allerdings aus verwaltungstechnischer Sicht den Nachteil, daß alle Alias-Befehle in alphabetischer Reihenfolge ausgegeben werden – unabhängig von ihrem definierten Speicherort.
Um temporär vorhandene Alias-Definitionen zu löschen, wird der Befehl *unalias* eingesetzt:

```
unalias dir
```

würde den weiter oben definierten dir-Befehl löschen.
Es lassen sich aber natürlich auch »richtige« Programme über Alias-Definitionen aufrufen und steuern. Der Befehl

```
alias doku='lynx -case -localhost \
./freebsd_dokus/buch/book.html'
```

startet die Anzeige der FreeBSD-Online-Dokumentation. Ausgehend vom aktuellen Verzeichnis wird dabei die Dokumentation in den Textmodus-Browser *lynx* geladen. Der Parameter *-case* schaltet die Unterscheidung von Groß- und Kleinschreibung ein und *-localhost* verhindert, daß bei der Anwahl von Hyperlinks Internetverbindungen aufgebaut werden.
Dieses kleine Beispiel zeigt, daß Alias-Definitionen mehr Möglichkeiten bieten als die Abkürzung von Verzeichniswechseln beispielweise mit *alias log='cd /var/log* oder die Auflistung von Verzeichnissen.
Der eigentliche Vorteil von *alias* liegt nämlich in der Möglichkeit, sich das ganze Shell-Instrumentarium zunutze machen zu können. Wenn man in das Verzeichnis */var/log* wechselt, wird in der Regel der Grund dafür sein, daß man sich eine Logdatei ansehen will. Mit einer Alias-Zuweisung läßt sich das komfortabel realisieren. Oben wurde bereits die Verknüpfung mit der Pipeline auf *more*

gezeigt. Nun folgt eine Befehlsverknüpfung:

```
alias wichtig='cd /var/log && \
tail -20 messages'
```

Diese Anweisung wechselt in das Verzeichnis */var/log* und zeigt dann die letzten zwanzig Zeilen der zentralen Datei *messages* an. Dieses Beispiel setzt strenggenommen keinen Wechsel in das Verzeichnis voraus: *tail -20 /var/log/messages* wäre ebenfalls korrekt und würde sich so auch in den Alias einbauen lassen. Ein weiteres Beispiel wäre

```
alias ueberfall=' cd /var/log && \
 less security | grep Deny | more'
```

Dieser Befehl wechselt in das Log-Verzeichnis, zeigt die Logdatei *security* an, wobei der Inhalt durch den Begriff »Deny« gefiltert wird. Damit werden unzulässige Zugriffe, die durch die Firewall protokolliert wurden, anzeigt. Abschließend wird wieder der Pager *more* aufgerufen, der dafür sorgt, daß die kritischen Zeilen seitenweise angezeigt werden.
Das Beispiel

```
alias doku='lynx -case -localhost \
 ./freebsd_dokus/buch/book.html > \
 /dev/ttyv2 &'
```

zeigt wiederum die FreeBSD-Dokumentation an, dieses Mal wird aber die Ausgabe auf die dritte Konsole (in FreeBSD-Notation) umgeleitet und durch die Angabe von »&« als Hinter-grundprozeß gestartet, damit die den Befehl auslösende Konsole sofort wieder nutzbar ist. Die Konsolenbezeichnung muß bei solchen Anweisungen natürlich dem benutzten System angepaßt werden. Bei Linux müßte »ttyX« angegeben sein, wobei das »X« durch die Konsolennummer zu ersetzen ist.
Bei solchen recht langen Eingaben kann durchaus über das Zeilenende hinaus weitergeschrieben werden, ohne allerdings reflexartig am Zeilenende die Enter-Taste zu betätigen.
Ein letztes Beispiel zeigt, daß auch Variablen verarbeitet werden können:

```
alias ueberfall2='cd /var/log && \
less messages && cd $HOME'
```

Hier wird erneut die Datei *messages* angezeigt. Anschließend wird ins Heimatverzeichnis gewechselt. Auch hier gilt wieder, daß ein Verzeichniswechsel eigentlich nicht notwendig ist und daher ein Rücksprung über *cd $HOME* vermieden werden kann. Hier soll nur demonstriert werden, daß der Einbau von Variablen möglich ist.

Dauerhaftes

Das Schlüsselwort für den Alias sollte sorgfältig gewählt werden und den Verwendungszweck angemessen umschreiben. Daß dabei die erste Idee nicht immer die beste ist, zeigt die Anweisung

```
alias mysql='less mysql_freebsd_wichtig'
```

Sie startet bei der Eingabe von *mysql* auf der Kommandozeile die Ausgabe des angegebenen Texts. Dabei bleibt dann der MySQL-Client, der auf die gleiche Weise aufgerufen wird, auf der Strecke. Dieses Verhalten hängt mit der Abarbeitungsreihenfolge der Bash zusammen. Die Manpage der Bash sagt dazu folgendes: »The first word of each command, if unquotet, is checked to see if it has an alias. If so, that word is replaced by the text of the alias.«
Ein weiteres Beispiel zeigt, daß nicht alles, was gewollt ist, auch funktioniert. So schlägt beispielsweise der folgende Versuch fehl:

```
alias text='vi irgendeinedatei.txt && \
vi nezweitedatei.txt > /dev/ttyv1'
```

Hier wird versucht, auf der aktuellen Konsole mit dem Editor *vi* eine Textdatei zu öffnen und per Umleitung auf die zweite Konsole (die Zählung beginnt wie so häufig bei 0) eine zweite Textdatei zu öffnen. Der Grund: Die Öffnung der ersten Textdatei mit *vi* liefert keinen Exit-Status von 0 zurück, weil der *vi* läuft, daher wird die zweite Textdatei nicht geöffnet, solange die erste nicht beendet ist. Wird der mit *vi* geöffnete Text auf der aktuellen Konsole geschlossen, erscheint prompt auf ttyv1 der zweite angeforderte Text. Verständlich wird das Verhalten mit dem folgendem erfolgreichen Versuch:

```
alias text='cat irgendeinedatei.txt && \
cat nezweitedatei.txt > /dev/ttyv1'
```

Diese Befehlszeile funktioniert, weil sich *cat* nach der erfolgreichen Anzeige des gewollten Textes beendet und so die Anzeige des zweiten Textes per Umleitung ermöglicht. Wer also mit dem Editor seiner Wahl zwei Texte öffnen will, wird dies auf zwei Alias-Anweisungen verteilen müssen.
Grundsätzlich wirken die Befehle *alias* und *unalias* nur temporär. Sie sind nach dem nächsten Einloggen verloren. Man trägt deshalb die gewünschten Definitionen in eine der Konfigurationsdateien der Shell ein.
In der Voreinstellung arbeitet die Bash die folgenden Dateien in der hier angegebenen Reihenfolge ab:

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile

Ist eine *~/.bash_profile* vorhanden, wird nur diese Datei eingelesen, die beiden anschließenden nicht mehr! Einträge in die */etc/profile* wirken benutzerunabhängig, die Einträge in den Dateien im jeweiligen Heimatverzeichnis regeln benutzerdefinierte Konfigurationen. Diese Dateien nehmen somit auch die dauerhaften Alias-Definitionen auf, wobei die Einträge mit einem Texteditor in der bereits vorgestellten Form vorgenommen werden. Wer in Subshells Alias-Definitionen benötigt, sollte die Datei *~/.bashrc* anlegen und die Einträge dort vornehmen. Ein erneutes Login nach dem Neueintrag ist überflüssig, wenn nach einer Bearbeitung an der Kommandozeile der Befehl

```
. .bash_profile
```

eingegeben wird. Allerdings wirkt dies nach meinen Versuchen nicht

besonders zuverlässig. Am sichersten funktioniert immer die Methode des Aus- und wieder Einloggens, wobei die *~/.bash_profile* neu eingelesen wird und die Alias-Einträge sofort verfügbar sind.
Bei intensivem Gebrauch zahlreicher Alias-Definitionen geht gern der Überblick verloren. Zur Verwaltung wurde deshalb ein kleines Skript geschrieben, mit dem vorhandene Alias-Einträge angezeigt, interaktiv neue Aliase hinzugefügt und nicht mehr benötigte Aliase entfernt werden können. Die Befehle *alias* und *unalias* sind dafür nicht geeignet, da sie flüchtig wirken und nur in der entsprechenden Subshell gelten würden. Vor der Ausführung sollte die vorhandene *~/.bash_profile* gesichert werden. Außerdem muß das Skript an die Position der bash im Dateisystem angepaßt werden. Die auf der CD-ROM befindliche Variante verweist auf die Datei */usr/local/bin/bash* von FreeBSD. Bei Linux befindet sich das Programm im Verzeichnis */bin*, bei NetBSD in */usr/pkg/bin*.

Eingabezeile

Zuerst wird beim Start geprüft, ob die zu bearbeitende Datei, in diesem Beispiel *~/.bash_profile*, überhaupt vorhanden ist. Wenn nicht, wird ein Hinweis darauf gegeben, wie sie erstellt werden kann. Alternativ können hier natürlich auch andere, weiter oben angegebene Dateien genannt werden.

```
PS3="Wähle die passende Aktion aus: "
select ZAHL in anzeigen loeschen \
              hinzufügen Ende; do
```

Der PS3-Prompt soll eine etwas angenehmere Eingabezeile zur Verfügung stellen. Das anschließende *select* bietet vier Aktionen an und stellt die Aktionen wie eine numerierte Liste dar. Erwartet wird nicht die Eingabe der Aktion per Wort, wie beispielsweise »hinzufügen«, sondern die der Aktion zugeordneten Zahl. Dementsprechend wertet die nachfolgende *case*-Anweisung die Zahl aus und vergleicht sie mit dem Text der Möglichkeiten:

```
case $ZAHL in
  anzeigen) clear
    grep ^alias ~/.bash_profile ;;
```

Die Aktion ist selbsterklärend. Angezeigt werden alle Alias-Definitionen in der genannten Datei. Das Zei-chen »^« vor dem Begriff »alias« weist *grep* an, nur die Zeilen auszugeben, bei denen der Begriff am Zeilenanfang steht. Damit werden Kommentarzeilen, die den Begriff enthalten, nicht angezeigt. Vorher wird mit *clear* noch der Bildschirm gelöscht, was die Übersichtlichkeit der nachfolgenden Anzeige erhöht. Hier läßt sich natürlich auch jede andere Konfigurationsdatei angeben. Die globale Datei */etc/profile* wurde hier bewußt nicht aufgenommen, weil der »normale« User sie in der Voreinstellung zwar lesen, aber nicht verändern kann. Mit den folgenden Zeilen

```
loeschen)
  clear
  echo "Welcher Alias soll gelöscht \
 werden?"
  read NAME
  sed /$NAME/d < ~/.bash_profile \
               > ~/.temp_profile
  mv ~/.temp_profile ~/.bash_profile
  clear
  ;;
```

wird ein vorhandener *alias* gelöscht. Der Stream-Editor *sed* liest dafür die komplette *.bash_profile* ein, der in der Variable angegebene Alias-Wert wird dabei ausgefiltert. Da sed seine Ausgabe nur auf *stdout* vornimmt und dabei zeilenweise arbeitet, also auch zeilenweise löscht, wird das Ergebnis aufgefangen und in die temporäre Datei *.temp_profile* umgeleitet. Als Ergebnis bleibt also die komplette Datei ohne den zu löschenden Alias. Die Datei wird anschließend wieder in *.bash_profile* umkopiert. Negativ kann sich hier eine ungünstige Namensauswahl bei Alias-Definitionen bemerkbar machen. Heißt der zu löschende Alias »text« und ein weiterer »text1«, wird dieser ebenfalls gelöscht. Der sed-Befehl wird hier übrigens nicht gequotet, weil das Quoten die Variablenauswertung verhindern würde. Die alternative Idee

```
hinzufuegen)
  echo "Wie soll der neue Alias heißen?"
  read NAME
  echo "Wie lautet der Alias-Befehl?"
  read BEFEHL
  echo "alias $NAME='$BEFEHL'" \
       >> ~/.bash_profile
  clear
  ;;
```

Das Hinzufügen einer neuen Alias-Definition erfordert den Befehlsnamen, der in die Variable *NAME* eingelesen wird und zusätzlich in *BEFEHL* die Befehlssequenz. Sie wird hier ohne Quoting-Zeichen als Befehl nur mit Leerzeichen aufgeführt. Der Operator >> sorgt dafür, daß die Zuweisung an das Ende der Datei angehängt wird. Um die Übersichtlichkeit einer Konfigurationsdatei zu erhöhen, ist es auf jeden Fall besser, alle Alias-Definitionen am Ende der Datei zu integrieren.

```
  Ende) echo "Bis demnächst Mal ..."
    break
    ;;
  *)
  echo "Nur eine Zahl zwischen 1 und 4!"
  esac
done
```

Das auf die Eingabe von »4« folgende *break* beendet das Script und **)* stellt eine minimale Form der Fehlerbehandlung dar, mit der alle Eingaben, die sich außerhalb der Zahlen 1 bis 4 bewegen, abgefangen werden. Das Skript ist wie üblich mit *./scriptname* zu starten. Erfolgreich getestet wurde es unter FreeBSD und SuSE-Linux. ◆

Literatur:

Die Manpage zur Bash

Alexander Mayer: Shell-Programmierung in Unix. C&L-Verlag 2001, ISBN 3-932311-78-7

Red. freeX: FreeBSD 4. C&L-Verlag 2001. ISBN 3-932311-88-4