Oliver Kreipl, Christian Eichhorn, Dorian Karnbaum, Heiko Marr †

Einführung in Unix/Linux

Ein Webmasters Press Lernbuch

Version 1.3.0 vom 03.06.2019

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

Inhaltsverzeichnis

8 Die Shell (Bash)

In dieser Lektion lernen Sie

- welche Aufgaben Sie mit der Shell erledigen können.
- was Systemvariablen sind und wie Sie mit ihnen umgehen.
- was Aliasse sind und wie sie eingerichtet werden.
- wie Sie Ihre Arbeitsumgebung einrichten.

8.1 Was tut eine Shell?

Sobald Sie sich als Benutzer an einem Unix-System anmelden, benötigen Sie zum Arbeiten eine *Shell*. Ein Benutzer, der keine Shell hat — auch das ist möglich, wie Sie noch lernen werden — wird sich auch nicht am System anmelden können. Shell kann als »Hülle« oder »Schale« übersetzt werden. Als Benutzer sind Sie in eine solche Schale eingebettet und können eine komplette Arbeitsumgebung nutzen.

Es gibt verschiedene Shells, wobei die Standardshell unter Linux die sog. ***Bash*** (**B**ourne-**a**gain-**sh**ell) ist. Aus Sicht Ihres Betriebssystems (Linux) ist die Bash einfach das erste Programm, das ausgeführt wird, sobald sich der Benutzer anmeldet. Wie alles unter Linux, können Sie selbstverständlich auch dies nach eigenen Wünschen konfigurieren.

Die *Bash* zeigt sich Ihnen als Konsole und Kommandointerpreter. Es können also nicht nur Programme gestartet werden, sondern Kommandos direkt am Prompt (dt. Eingabeaufforderung) eingeben werden, die auch »prompt« verarbeitet werden.

Die *Bash* verfügt zudem über eine eigene Shell-Skript-Sprache, mit der Sie wiederkehrende Administrationsaufgaben leicht in ein kleines Programm, ein sog. ***Shell-Skript,*** packen können.

8.2 Verschachtelte Shells (Subshells)

Sobald Sie sich am System anmelden, wird Ihnen Ihre sog. ***Login-Shell*** zugewiesen, die dem Eintrag in der Datei */etc/passwd* entspricht. Linux beschränkt sich aber nicht nur auf eine Shell, sondern Sie können jederzeit eine neue, beliebige Shell starten, sofern diese installiert ist. Bei modernen Linux-Systemen wie *Debian* verweist die ursprüngliche Shell *sh* z. B. auf die *Dash*, eine schlanke und schnelle Shell, die für die Ausführung von Shellskripten gedacht ist. Für die tägliche Arbeit im ***Interaktiven Modus***[13] ist sie aber eher ungeeignet, da es ihr schlicht an Komfort fehlt. Um Ihnen dennoch zu zeigen, dass Sie sich innerhalb einer anderen Shell und nicht der Login-Shell befinden, wenn Sie innerhalb einer Shell eine neue Shell öffnen, möchte ich Ihnen kurz den Unterschied zwischen den Shell-Befehlen `exit` und `logout` erklären.

logout

Mit dem Befehl `logout` können Sie sich von der Login-Shell abmelden, und zwar nur von der Login-Shell. Befinden Sie sich innerhalb einer weiteren, verschachtelten Shell, werden Sie vom System gebeten, `exit` einzugeben, um die aktuelle Shell zu verlassen. Verlassen Sie die Login-Shell mit `exit`, wird automatisch `logout` aufgerufen.

13. Der Modus, in dem Sie Befehle auf der Kommandozeile eingeben können und in dem die Shell Ihnen Rückmeldung über Erfolg oder Misserfolg bei der Ausführung des Befehls gibt

exit

Mit `exit` können Sie jede Shell verlassen, egal ob es sich um eine verschachtelte oder die Login-Shell handelt.

Zurück zu unseren verschachtelten Shells. Da Sie jetzt den Unterschied zwischen `exit` und `logout` kennen, können Sie auch testen, in welcher Shell Sie sich befinden, indem Sie in einer verschachtelten Shell `logout` eingeben. Sehen wir uns ein Beispiel an:

Beispiel

```
oliver@gollum:~$ bash

oliver@gollum:~$ logout
bash: logout: not login shell: use 'exit'
```

```
oliver@gollum:~$ exit

oliver@gollum:~$
```

Nachdem wir den Befehl `bash` abgesetzt haben, befinden wir uns in einer weiteren Shell, und es ist uns nicht mehr möglich, diese mit `logout` zu verlassen, weil es sich nicht um die Login-Shell handelt. Mit `exit` verlassen wir die zweite Shell und befinden uns wieder in unserer Login-Shell.

8.3 Kommandoeingabe

Sie haben bereits mit der Shell gearbeitet und Befehle eingegeben. Auch mit der Autovervollständigung gehen Sie mittlerweile wie selbstverständlich um. Einige nützliche Tastenkürzel kennen Sie allerdings noch nicht. Diese möchte ich Ihnen an dieser Stelle zeigen.

8.3.1 History

Die ***History***-Funktion der Shell werden Sie häufig nutzen. Die Autovervollständigung ist bereits ein großer Schritt in Richtung Effektivität, jedoch übertrifft die History diese um Vieles. Hier die wichtigsten Befehle der History-Funktion innerhalb der Shell:

- Mit den Cursortasten **<↑>** und **<↓>** können Sie in der History zurück oder nach vorne gehen. Es werden jeweil die zuletzt verwendeten Befehle aufgerufen. Sie müssen lediglich noch mit **<ENTER>** bestätigen.
- Mit der Tastenkombination **<CTRL><R>** können Sie innerhalb der History nach Suchmustern suchen. Es wird jeweils der neueste Eintrag in der History angezeigt. Drücken Sie die Tastenkombination **<CTRL><R>** erneut, wird der nächstältere angezeigt, und so weiter.

Wenn Sie sich fragen, wo sich Linux Ihre letzten Befehle merkt, dann möchte ich Sie auf eine versteckte Datei in Ihrem Homeverzeichnis aufmerksam machen: *~/.bash_history*. Wie deren Name schon sagt, wird hier eine bestimmte Anzahl von Eingaben gespeichert.

Jede Shell, die Sie zusätzlich öffnen, hat eine eigene History. Es wird Ihnen also nicht möglich sein, einen Befehl zu finden, den Sie in einer geschlossenen Shell abgesetzt hatten.

Übung 26:

1 Sehen Sie sich die Datei *~/.bash_history* an. Verwenden Sie nicht *vi*. Welchen Befehl können Sie verwenden?

8.3.2 Autovervollständigen

Wie Sie bereits gelernt haben, können Sie mit der Tabulatortaste **<TAB>** versuchen, den eingegebenen Befehl oder Dateinamen zu vervollständigen. Ist eine Möglichkeit eindeutig, wird die Vervollständigung durchgeführt, ansonsten nur soweit, bis sie nicht mehr eindeutig ist. Drücken Sie die Taste **<TAB>** zweimal, bekommen Sie alle Möglichkeiten angezeigt.

8.3.3 Die Kommandozeile benutzen

Auch für die Kommandozeile stehen Ihnen einige Tastenkürzel zur Verfügung, die Ihnen den Umgang mit der Shell erleichtern. Wie oft kommt es vor, dass man bereits einen Pfad eingibt, jedoch den Befehl am Anfang vergessen hat. Oder ein längerer Befehl aus der History muss nur geringfügig verändert werden. Dann sind solche Tastenkürzel unerlässliche Helfer.

- Mit **<POS1>** und **<ENDE>** können Sie den Cursor an den Anfang bzw. das Ende der Zeile setzen.
- Die Tasten **<BACKSPACE>** bzw. **<ENTF>** entfernen jeweils ein Zeichen vor oder nach dem Cursor.
- Mit **<ALT><D>** können Sie ein ganzes Wort löschen, wenn der Cursor auf dem ersten Buchstaben steht.
- Die Tastenkombination **<STRG><K>** löscht alles vom Cursor bis zum Ende der Zeile.
- Eine der wichtigsten Tastenkombinationen ist **<STRG><L>**. Damit können Sie den Bildschirm löschen.[14] Sie hat die gleiche Funktion wie der Befehl `clear`.
- Mit den Tastenkombinationen ***<UMSCHALT><BildAuf>*** und ***<UMSCHALT><BildAb>*** können Sie den Bildschirminhalt seitenweise nach oben bzw. unten scrollen.

8.4 Systemvariablen

Wie auch bei *Microsoft Windows* gibt es unter Linux sog. Systemvariablen. Diese werden als Shellvariablen bzw. Umgebungsvariablen bezeichnet. Sie können solche Variablen auch selbst definieren oder auch verändern. Den Wert einer solchen Variable können Sie sich ausgeben lassen, indem Sie folgenden Befehl eingeben:

```
echo $<Variable>
```

Beispiel

```
oliver@gollum:~$ echo $SHELL
/bin/bash
```

Dem eigentlichen Variablennamen wird ein Dollarzeichen (`$`) vorangestellt, um auf den Wert der Variablen zuzugreifen.

Shellvariablen vs. Umgebungsvariablen

Bei ***Shellvariablen*** handelt es sich um sog. lokale Variablen, die nur innerhalb einer Shell gültig sind. Verlassen Sie diese Shell, verlieren diese Variablen ihre Gültigkeit und werden gegebenenfalls durch entsprechende Konfigurationseinstellungen der neuen Shell neu belegt.

14. Bildschrim löschen stimmt nicht ganz: Es werden nur so viele leere Zeilen eingefügt, dass der Prompt wieder in der ersten Zeile oben erscheint. Das optische Ergebnis ist, dass die Konsole wieder leer ist!

Umgebungsvariablen sind globale Variablen, die in allen Shells vorhanden sind, also auch in Subshells. Um eine Shellvariable zu einer Umgebungsvariablen zu machen, müssen Sie die Shellvariable exportieren. Dies geschieht mit folgendem Befehl:

```
export <Variable>
```

Durch diesen Befehl vererbt sich die Variable auf sämtliche Subshells.

Einer Variable können Sie folgendermaßen einen Wert zuweisen:

```
<Variable>=<Wert>
```

Beispiel

Nehmen wir an, Sie belegen eine Variable:

```
oliver@gollum:~$ name=marc
```

Jetzt können Sie sich die Variable ausgeben lassen, indem Sie sie mit dem Befehl `echo` aufrufen:

```
oliver@gollum:~$ echo $name
marc
```

Wechseln Sie jetzt in eine Subshell, ist die Variable *$name* nicht mehr vorhanden:

```
oliver@gollum:~$ bash

oliver@gollum:~$ echo $name
```

Wechseln wir also zurück in die Login-Shell, in der wir die Variable deklariert haben, und exportieren die Variable.

```
oliver@gollum:~$ exit

oliver@gollum:~$ export name
```

Nachdem wir die Variable *$name* exportiert haben, ist sie auch in jeder Subshell sichtbar:

```
oliver@gollum:~$ bash

oliver@gollum:~$ echo $name
marc
```

Shellvariablen anzeigen lassen

Welche Shellvariablen bereits belegt sind, können Sie sich mit dem Befehl `set` anzeigen lassen. Diese Liste kann sehr lang sein und Sie werden wohl nicht alle Variablen auf einer Bildschirmseite sehen.

```
oliver@gollum:~$ set
BASH=/bin/bash
...
HISTFILE=/home/oliver/.bash_history
HISTFILESIZE=2000
HISTSIZE=1000
HOME=/home/oliver
HOSTNAME=gollum
```

```
...
LOGNAME=oliver
...
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
...
PS1='\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
...
PWD=/home/oliver
SHELL=/bin/bash
...
USER=oliver
...
```

Sie sehen in dieser Ausgabe, einen Auszug der Systemvariablen auf meinem Server, von denen wir uns einige auf den nächsten Seiten noch genauer ansehen werden.

Um sich die Umgebungsvariablen anzeigen zu lassen, also die vererblichen (globalen) Variablen, können Sie den Befehl `export` ohne Parameter verwenden:

```
oliver@gollum:~$ export
declare -x DISPLAY="localhost:10.0"
declare -x HOME="/home/oliver"
declare -x LANG="en_US.UTF-8"
declare -x LOGNAME="oliver"
declare -x LS_COLORS="rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35 ... "
declare -x MAIL="/var/mail/oliver"
declare -x OLDPWD
declare -x PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
declare -x PWD="/home/oliver"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_CLIENT="10.0.2.2 51383 22"
declare -x SSH_CONNECTION="10.0.2.2 51383 10.0.2.15 22"
declare -x SSH_TTY="/dev/pts/0"
declare -x TERM="xterm"
declare -x USER="oliver"
declare -x name="marc"
```

8.5 Die wichtigsten Systemvariablen

8.5.1 $PATH

Die wohl wichtigste Shellvariable kennen Sie vielleicht bereits von Microsoft Windows. Es handelt sich um die Variable *$PATH*. Diese Variable enthält Pfadangaben, in denen die Shell nach Befehlseingaben sucht. Die einzelnen Pfadangaben werden durch einen Doppelpunkt (`:`) getrennt.

Beispiel

`PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games`

Sie können in jedem beliebigen Verzeichnis Befehle, die sich innerhalb der Pfadangaben von *$PATH* befinden, direkt ausführen, ohne den kompletten Pfad anzugeben. Sie können also jede ausführbare Datei, die sich z. B. innerhalb von */usr/bin/* befindet, direkt ohne Pfad ausführen.

Beispiel

```
oliver@gollum:~$ whoami
oliver
```

Möchten Sie wissen, wo genau der Befehl `whoami` liegt, können Sie auch das leicht herausfinden, indem Sie den Befehl `which` verwenden. Dieser Befehl gibt Ihnen den kompletten Pfad zur Datei aus.

Beispiel

```
oliver@gollum:~$ which whoami
/usr/bin/whoami
```

Eigenen Pfad zu $PATH hinzufügen

Sie können auch ein eigenes Verzeichnis in die Variable mit aufnehmen. Wenn Sie sich demnächst mit Shellskripting beschäftigen werden, ist es vielleicht nützlich, Ihre eigenen Skripte innerhalb eines Verzeichnisses zu speichern, das sich in *$PATH* befindet, um nicht ständig den kompletten Pfad zur Datei angeben zu müssen.

Sie können natürlich Ihre Skripte innerhalb von */usr/local/bin/* speichern, aber zu Übungszwecken legen wir ein Verzeichnis in unserem Homeverzeichnis an, das wir *$PATH* hinzufügen wollen.

Übung 27:

1. Erstellen Sie das Verzeichnis *~/bin*.

Nachdem wir das Verzeichnis *bin* in unserem Homeverzeichnis erstellt haben, können wir es wie folgt dem Pfad hinzufügen.

```
oliver@gollum:~$ PATH=$PATH':~/bin'
```

2. Fügen Sie *~/bin* der Systemvariablen *$PATH* hinzu.
3. Lassen Sie sich die Systemvariable *$PATH* ausgeben.

Nach erfolgreicher Übung haben Sie das Verzeichnis *~/bin* der Systemvariablen *$PATH* hinzugefügt und können jetzt ausführbare Dateien, die sich innerhalb dieses Verzeichnisses befinden, aus jedem Verzeichnis Ihres Systems heraus ohne Pfadangabe aufrufen.

Die Methode, mit der Sie den Wert der Variablen *$PATH* hinzugefügt haben, speichert die Variable allerdings nur für Ihre aktuelle Shell. Möchten Sie den Pfad dauerhaft zu *$PATH* hinzufügen, müssen Sie den Wert in der Datei *~/.profile* ändern, da er dort gesetzt wird. Bevor Sie nun beginnen die Datei ~/.profile mit *vi* zu bearbeiten, kann ich Ihnen die erfreuliche Mittteilung machen, dass dort schon eine Prüfung erfolgt, die dafür sorgt, dass ~/bin zu Ihrem Pfad hinzugefügt wird, falls das Verzeichnis existiert:

```
...
19 # set PATH so it includes user's private bin if it exists
20 if [ -d "$HOME/bin" ] ; then
21     PATH="$HOME/bin:$PATH"
22 fi
```

Codebeispiel 12 *Ausszug aus ~/.profile*

Die Datei ~./profile ist eine von mehreren Konfigurationsdateien Ihrer Shell, die ich Ihnen im nächsten Abschnitt noch genauer vorstellen werde. Wenn Sie die vier Zeilen Shellskript zum jetzigen Zeitpunkt noch nicht verstehen, machen Sie sich keine Sorgen. Im einem weiteren Kurs dieser Reihe lernen Sie, solche Skripte selbst zu schreiben.

8.5.2 Der Prompt - $PS1

In der Systemvariablen *$PS1* werden die Einstellungen für den Prompt gespeichert. Auch diese Variable können Sie sich mit `echo` ausgeben lassen:

Beispiel

```
oliver@gollum:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$
```

Diese Variable hat eine ganz eigene Syntax, die wir uns kurz genauer ansehen. Der Standardprompt von Debian ist dabei in drei Abschnitte unterteilt:

1. \[\e]0;\u@\h: \w\a\]: hierbei handelt es sich um eine *Escape-Sequenz*, die den Titel Ihres Terminalfensters festlegt, wenn Ihr Terminalprogramm dies unterstützt. Wechseln Sie z. B. auf der Kommandozeile das Verzeichnis, so wird das neue Verzeichnis auch in der Titelleiste angezeigt.[15]
2. ${debian_chroot:+($debian_chroot)}: dieses Konstrukt sorgt dafür, dass der Inhalt der Variablen debian_chroot in runden Klammern ausgegeben wird, allerdings nur dann, wenn die Variable debian_chroot auch gesetzt ist. Ist die Variable leer, wird nichts ausgegeben. Diese Variable kann für Change-Root-Umgebungen[16] genutzt werden.
3. \u@\h:\w\$: hier wird nun das eigentliche Aussehen Ihres Prompts festgelegt. Bei den Buchstaben handelt es sich um Kürzel von Variablen, wie sie für den Prompt verwendet werden können. Dieser Teil reicht für einen einfachen Prompt eigentlich schon aus. Die ersten beiden Teile sind also nicht zwingend nötig.

In Tabelle 8.1 sehen Sie einen Ausschnitt von Kürzeln, die verwendet werden können und was ihre Bedeutung ist.

Variable	Bedeutung
\h	Der Hostname
\w	Das aktuelle Verzeichnis
\W	Die letzte Angabe des aktuellen Verzeichnisses (*bin* bei */usr/bin*)

Tabelle 8.1 *Mögliche Variablen im Prompt*

15. Wenn Sie an der Serverkonsole angemeldet sind und nicht über ein Teminalprogramm wie z. B. *Putty*, wird dieser Teil der PS1-Systemvariablen fehlen. Er wird nur hinzugefügt, wenn die Shell erkennt, dass es sich um ein Terminalprogramm handelt, über das die Anmeldung erfolgt ist.
16. Wir werden uns im Zusammenhang mit der FTP-Server Konfiguration noch genauer mit dem Thema Change-Root-Umgebung auseinandersetzen.

Variable	Bedeutung
`\u`	Der aktuelle Benutzer
`\$`	Das Promptzeichen (`$` bei normalen Benutzern und `#` bei root)
`\t`	Aktuelle Zeit
`\d`	Aktuelles Datum
`\[`	Beginn einer Sequenz von nicht darstellbaren Zeichen. Wird zu Beginn einer Terminal-Kontrollsequenz verwendet.
`\]`	Ende einer Sequenz von nicht darstellbaren Zeichen. Wird am Ende einer Terminal-Kontrollsequenz verwendet.

Tabelle 8.1 *Mögliche Variablen im Prompt*

Eine vollständige Liste der möglichen Variablen für den Prompt finden Sie übrigens in den Man-Pages der Bash, wenn Sie nach dem Begriff PS1 suchen.

8.5.3 $HOME

Diese Variable enthält das Homeverzeichnis des aktuellen Benutzers. Sind Sie als Benutzer *oliver* angemeldet, ist Ihr Homeverzeichnis */home/oliver*. Die Information wird aus der Datei */etc/passwd* gelesen, in der das jeweilige Homeverzeichnis der Benutzer gespeichert ist.

8.5.4 $USER

In *$USER* wird der aktuelle Benutzer gespeichert.

8.5.5 $PWD

$PWD enthält den Pfad des aktuellen Verzeichnisses. Der Befehl `pwd` ist Ihnen bereits bekannt, dieser hat mit der Systemvariablen *$PWD* jedoch nichts zu tun. Testen Sie es:

Übung 28:

1. Weisen Sie der Systemvariablen *$PWD* einen neuen Wert zu.
2. Beobachten Sie die Veränderung an Ihrem Prompt.
3. Führen Sie den Befehl `pwd` aus, und vergleichen Sie diese Ausgabe mit dem angezeigten Prompt.
4. Wechseln Sie in Ihr Homeverzeichnis und beobachten Sie, was passiert.

8.6 Shell einrichten

8.6.1 Die Bash-Konfigurationsdateien

Auch bei den Konfigurationsdateien der *Bash* verfolgt *Debian* ein Schema, das immer wieder bei Konfigurationen auftaucht. Der Sinn dieses Schemas ist es, Konfigurationen für Benutzer individuell zu halten, denn jeder Benutzer soll sich seine Shell so konfigurieren können, wie er möchte.

Erreicht wird dies durch den Einsatz mehrerer Konfigurationsdateien. Es gibt eine »globale« Konfigurationsdatei, die sich meistens im Verzeichnis */etc* befindet, und eine »persönliche« Konfigurationsdatei, die versteckt im Homeverzeichnis des Benutzers liegt (z. B. *~/.vimrc*).

Diese Dateien werden nacheinander von »global« nach »persönlich« abgearbeitet, wobei Variablen jederzeit von der nächsten Datei neu zugewiesen werden können.

Beim Anmelden eines Benutzers werden folgende Konfigurationsdateien aufgerufen:

/etc/profile

Dies ist die Hauptkonfigurationsdatei für die Shell und gilt systemweit, d. h. es handelt sich um Einstellungen, die zunächst für jeden Benutzer des Systems gleich sind. In der Datei werden folgende wichtige Einstellungen vorgenommen:

- Die Konfiguration des Prompts für alle Benutzer.
- Die Konfiguration der PATH-Variablen für *root* oder andere Benutzer.
- Globlaisierung der PATH-Variablen.

~/.profile

Die lokale Konfigurationsdatei für die Shell. Sie gilt für den Benutzer, in dessen Homeverzeichnis sie sich befindet. Diese Datei enthält unter Debian einen Verweis auf die Datei *~./bashrc*. Es wird also die Konfigurationsdatei *~/.bashrc* aus *~/.profile* heraus ausgeführt.

/etc/bash.bashrc

Dies ist die globale Konfigurationsdatei der *Bash*. Allerdings enthält diese Datei Einträge, die nur im interaktiven Modus einer Nicht-Login-Shell aufgerufen werden, d. h. die Datei ist nur aktiv, wenn Sie sich innerhalb einer Shell als ein anderer Benutzer anmelden oder eine Subshell aufrufen. Andernfalls hat diese Datei keinerlei Auswirkung. Unter Debian wird diese Datei allerdings aus */etc/profile* heraus aufgerufen. Sie wird damit unter Debian sowohl für Login-Shells als auch für Nicht-Login-Shells ausgeführt. Andere Linux-Distributionen können hier durchaus anders konfiguriert sein.

~/.bashrc

Dies ist die persönliche Konfigurationsdatei der *Bash*. Allerdings enthält diese Datei Einträge, die nur im interaktiven Modus einer Nicht-Login-Shell aufgerufen werden, d. h. die Datei ist nur aktiv, wenn Sie sich innerhalb einer Shell als ein anderer Benutzer anmelden oder eine Subshell aufrufen. Da diese Datei aber in ~/.profile aufgerufen wird, wird dieses Skript in der Standardkonfiguration auch für die Login-Shell ausgeführt.

Weitere Konfigurationsdateien

Die Dateien *~/.bash_profile* und *~/.bash_login* finden in der Standardkonfiguration von Debian keine Verwendung mehr, Sie können Ihnen aber auf älteren Systemen oder anderen Linux-Distributionen begegnen. Ist eine dieser beiden Dateien in Ihrem Home-Verzeichnis vorhanden wird die Datei ~/.profile **nicht** aufgerufen!

~/.bash_profile

Wird in jedem Fall für Login-Shells ausgeführt, falls sie im Home-Verzeichnis vorhanden ist.

~/.bash_login

Wird für Login-Shells nur dann ausgeführt, wenn keine ~/.bash_profile vorhanden ist.

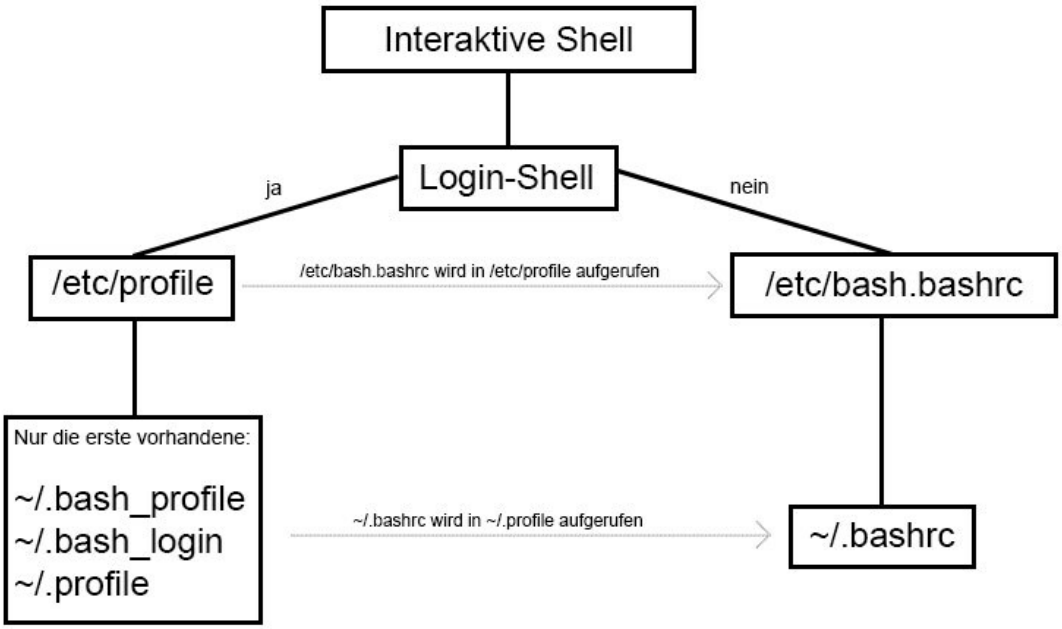

Abb. 62 *Reihenfolge der Abarbeitung der Bash-Konfigurationsdateien*

8.6.2 Aliasse

Mit Aliassen können Sie Zeichenketten neu zuweisen oder neue Zeichenketten erstellen. Geben Sie an der Eingabeaufforderung einen Befehl ein, geben Sie im Prinzip nur eine Zeichenkette ein, die nach Bestätigung durch die Taste **<ENTER>** innerhalb des aktuellen Verzeichnisses oder des Pfades gesucht wird. Wird diese Zeichenkette gefunden und sie stimmt mit dem Namen eines ausführbaren Programms überein, wird das Programm gestartet.

Aliasse stellen Ihnen eine Möglichkeit zur Verfügung, dem System die Eingabe einer anderen Zeichenkette vorzugaukeln. Möchten Sie z. B., dass bei `rm` das System `rm -i` ausführt, müssen Sie einen Alias setzen.

Beispiel

```
oliver@gollum:~$ alias rm='rm -i'
```

Nachdem Sie diesen Alias zugewiesen haben, wird die Zeichenkette `rm` durch die Zeichenkette `rm -i` ersetzt.

Übung 29:

1. Erstellen Sie einen Alias für den Befehl `ls`, sodass auch versteckte Dateien mit angezeigt werden.
2. Erstellen Sie zwei weitere Aliasse, die Sie für sinnvoll erachten. Sie können dabei auch selbst Befehle erfinden.

Haben Sie keine Angst, Ihr Befehl `ls` oder `rm` wird nicht überschrieben. Das System interpretiert nur Ihre Eingabe anders.

Natürlich kommt es vor, dass ein Alias nicht mehr erwünscht ist, genauso wie Sie sich schnell bei der Eingabe des Alias vertippt haben. Dann haben Sie vielleicht `rm` mit `echo 'hallo'` überschrieben und können keine Datei mehr löschen.

Anstatt sich vom System abzumelden und wieder anzumelden, können Sie den Alias einfach wieder löschen. Gelöscht wird ein Alias mit dem Befehl `unalias`.

Beispiel

```
oliver@gollum:~$ unalias rm
```

3 Löschen Sie die von Ihnen erstellten Aliasse von Ihrem System.

Wie alle Variablen, die Sie an der Kommandozeile zuweisen, so sind auch Aliasse nicht von ewiger Dauer, sondern gelten nur so lange, wie die aktuelle Shell läuft. Um Aliasse dauerhaft zu speichern, können Sie diese einfach in die Konfiguration Ihrer Bash einfügen. Selbstverständlich steht es Ihnen frei, die Aliasse auch global zu setzen, also für alle Benutzer, jedoch sollten Sie davon absehen, denn wie soll ein anderer Benutzer Ihre Aliasse kennen? Und vielleicht sind sie nicht nach seinem Geschmack?

Sie könnten Ihre persönlichen Aliasse nun einfach in die Datei *~/.bashrc* eintragen. Wenn Sie aber einen Blick in die Datei ~/.bashrc werfen, dann sehen Sie in den Zeilen 97-99, dass Ihnen dort noch eine weitere Möglichkeit angeboten wird, die Sie auch nutzen sollten. Diese drei Zeilen Shellskript überprüfen, ob in Ihrem Homeverzeichnis eine Datei mit dem Namen *~/.bash_aliases* existiert. Is dies der Fall, wird die Datei ausgeführt.

```
...
87 # some more ls aliases
88 #alias ll='ls -l'
89 #alias la='ls -A'
90 #alias l='ls -CF'
91
92 # Alias definitions.
93 # You may want to put all your additions into a separate file like
94 # ~/.bash_aliases, instead of adding them here directly.
95 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
96
97 if [ -f ~/.bash_aliases ]; then
98     . ~/.bash_aliases
99 fi
...
```

Codebeispiel 13 *~/.bashrc*

Sie haben richtig erkannt, dass sich auch bereits vordefinierte Aliasse in der Datei *~/.bashrc* befinden. Sie können diese aktivieren, indem Sie die Raute (`#`) am Zeilenanfang entfernen.

Übung 30:

1 Erstellen Sie die Datei *~/.bash_aliases*.

2 Legen Sie mindestens einen Alias in der Datei *~/.bash_aliases* an.

3 Melden Sie sich ab und erneut an, damit Ihre neuen Aliasse auch aktiv werden.

8.7 type

Wenn Sie sich nicht sicher sind, um was es sich bei einem Befehl handelt, den Sie an der Kommandozeile eingeben, können Sie dies mit dem Befehl `type` herausfinden. Mit `type` wird Ihnen angezeigt, ob es sich bei der von Ihnen eingegebenen Zeichenfolge um einen Systembefehl, einen normalen Befehl, einen symbolischen Link oder einen Alias handelt.

Beispiel

```
oliver@gollum:~$ type ls
ls is aliased to `ls --color=auto'
```

Hier sehen Sie das Ergebnis des Alias in der Datei *~/.bashrc*, Zeile 78.

Übung 31:

1 Wenden Sie `type` auf folgende Befehle an und vergleichen Sie die Ausgabe:
- rm
- ls
- exit
- test

8.8 Testen Sie Ihr Wissen

1. Wo wird gespeichert, welche Login-Shell ein Benutzer hat?

   Bitte ankreuzen:

   ☐ Im Home-Verzeichnis

   ☐ In der Datei */etc/profile*

   ☐ In der Datei */etc/passwd*

   ☐ In der Datei *~/.bash_login*

2. Wie können Sie am Prompt den letzten Befehl wiederholen?

   Bitte ankreuzen:

   ☐ Durch Drücken der Taste **<↑>** wird der vorherige Befehl an den Prompt geschrieben, der dann mit **<ENTER>** bestätigt werden kann.

   ☐ Mit der Tastenkombination **<CTRL><L>** wird der vorherige Befehl an den Prompt geschrieben, der dann mit **<ENTER>** bestätigt werden kann.

   ☐ Mit der Tabulator-Taste wird der vorherige Befehl an den Prompt geschrieben, der dann mit **<ENTER>** bestätigt werden kann.

   ☐ Es gibt keine einfache Möglichkeit den letzten Befehl zu wiederholen. Jeder Befehl muss neu am Prompt eingegeben werden.

3. Mit welcher Tastenkombination leeren Sie Ihren Bildschirm?

   Bitte ankreuzen:

   ☐ **<CTRL><ESC>**

   ☐ **<CTRL><L>**

- ☐ **<CTRL><E>**
- ☐ **<CTRL><D>**

4. Mit welchem Befehl können Sie feststellen in welchem Verzeichnis Sie sich befinden?

   Bitte ankreuzen:

   - ☐ `whoami`
   - ☐ `pwd`
   - ☐ `echo $PATH`
   - ☐ `passwd`
   - ☐ `echo $PS1`

5. In welche Dateien können Sie Aliasse einfügen, um sie dauerhaft zu speichern?

   Bitte ankreuzen:

   - ☐ Global in die Datei */etc/bash.bashrc*
   - ☐ Global in die Datei */etc/passwd*
   - ☐ Für einen einzelnen Benutzer in die Datei *~/.bashrc*
   - ☐ Für einen einzelnen Benutzer in die Datei *~/.bash_aliases*
   - ☐ Für einen einzelnen Benutzer in die Datei *~/.vimrc*