den werden, ob sie eine Shell- oder eine Environment-Variable sein soll (siehe Kapitel 2.3.2 auf Seite 58).

2.1.6 Bearbeitung der Kommandozeile

Vor der eigentlichen Ausführung eines vom Benutzer eingegebenen Kommandos durch die Shell, wertet diese die gesamte Kommandozeile aus. Dabei werden beispielsweise Sonderzeichen wie der `*` durch Dateinamen oder in der Kommandozeile stehende Variablennamen durch ihre Werte ersetzt. Da man die Ergebnisse solcher Ersetzungen nicht direkt beobachten kann, scheinen die gestarteten Programme oftmals nicht das zu tun, was man beabsichtigt hatte. Um solche Probleme erkennen und lösen zu können, ist ein genaueres Verständnis der Vorgänge erforderlich, die bei der Bearbeitung der Kommandozeile durch die Shell unsichtbar ablaufen. Daher sollen die wesentlichen Schritte hier kurz beschrieben werden:

1. Die Shell liest bis zum ersten Kommandotrenner (`& && || ; < >`) und stellt fest, ob Zuweisungen an Variablen erfolgen sollen oder die Ein- oder Ausgabe umgelenkt werden soll.

2. Die Shell zerlegt die Kommandozeile in einzelne Argumente. Sie trennt die Argumente durch Trennzeichen, normalerweise das Leer- und Tabulator-Zeichen.

3. Variablen (mit vorangestelltem `$`), die in der Kommandozeile stehen, werden durch ihre Werte ersetzt.

4. Kommandos, die in Apostrophen (`` ` ` ``) stehen, werden ausgeführt und durch ihre Ausgabe ersetzt.

5. `stdin`, `stdout` und `stderr` werden gemäß den Angaben auf der Kommandozeile umgelenkt.

6. Falls in der Kommandozeile Zuweisungen *an* Variablen stehen, werden diese ausgeführt.

7. Die Shell sucht nach Sonderzeichen (`*`, `?`, `[ ]`, ...) und ersetzt diese, sofern sie nicht in quote-Zeichen (`" "`, `' '`, `\`) gefaßt sind, durch passende Dateinamen.

8. Die Shell führt das Kommando aus.

Ein Beispiel soll diesen Ablauf verdeutlichen. Da es in diesem Fall wesentlich ist, welche Dateien im aktuellen Verzeichnis des Benutzers liegen, gelte die Annahme, daß sich lediglich die Verzeichnisse `tmp und etc` dort befinden. Weiterhin soll das Verzeichnis `tmp` die Dateien `titel` und `text.bak` enthalten. Das aktuelle Verzeichnis des Benutzers sei sein Home-Verzeichnis. Die Abfolge der einzelnen Aktionen, die die Shell auf der unten stehenden Kommandozeile ausführt, werden entsprechend der oben dargestellten Numerie-

rung dargestellt. Falls in einem der Schritte keine Aktion erfolgt, steht an dieser Stelle ein „—“

```
mr@hamlet> ls -l `pwd` $HOME/etc tmp/* > ls.txt
```

1. Die Shell liest bis zum `>`-Zeichen und bearbeitet zunächst den links davon stehenden Teil der Kommandozeile weiter.

2. Die Kommandozeile wird in die Argumente `ls -l`, `` `pwd` ``, `$HOME/etc` und `tmp/*` zerlegt.

3. Die Variable `$HOME` wird durch ihren Wert, also das Home-Verzeichnis des Benutzers, das beispielsweise `/home/mr` lauten könnte, ersetzt. Die Kommandozeile lautet jetzt:
   `ls -l` `` `pwd` `` `/home/mr/etc tmp/*`

4. Das in `` ` ` `` stehende Kommando `pwd` wird ausgeführt. Die Ausgabe des Kommandos (`/home/mr`) wird anstelle von `` `pwd` `` in die Kommandozeile eingefügt, so daß die Kommandozeile jetzt wie folgt aussieht:
   `ls -l /home/mr /home/mr/etc tmp/*`

5. `stdout` wird auf die Datei `ls.txt` umgelenkt. Falls diese Datei noch nicht existiert hat, wird sie angelegt, ansonsten wird der bisherige Inhalt gelöscht.

6. —

7. Das Sonderzeichen `*` in dem Argument `tmp/*` wird, da der Stern nicht durch Quote-Zeichen geschützt ist, durch Dateinamen der im Verzeichnis `tmp` liegenden Dateien ersetzt (da `*` für beliebige und beliebig viele Zeichen von Dateinamen steht). Die Kommandozeile lautet schließlich:
   `ls -l /home/mr /home/mr/etc tmp/titel tmp/text.bak`

8. Das Kommando wird ausgeführt.

2.1.7 Gruppieren von Kommandos

Normalerweise gibt der Benutzer ein Kommando ein, beendet die Eingabe durch Drücken der `Return`-Taste und wartet, bis das Kommando ausgeführt wurde und die Shell wieder den Prompt anzeigt, um ihre Eingabebereitschaft zu signalisieren.

Unter Umständen ist es jedoch sinnvoll, mehrere Kommandos in einem Zug auszuführen, Kommandos gleichzeitig abarbeiten zu lassen oder ein Kommando in Abhängigkeit vom Erfolg oder Mißerfolg eines vorangegangenen Befehls auszuführen. Zu diesem Zweck wird der Rückgabewert, den jedes Kommando liefert, ausgewertet. Ein Rückgabewert von `0` bedeutet, daß ein Kommando erfolgreich ausgeführt werden konnte, ein anderer Wert bedeutet einen Fehler. Der Rückgabewert eines Kommandos kann *direkt nach* dessen Ausführung über die Shell-Variable `?` mit dem Kommando `echo $?` ausgegeben werden. Zur

Gruppierung von Kommandos bietet die Shell verschiedene Möglichkeiten, die in folgender Tabelle aufgelistet sind:

Tabelle 2.4: Symbole zur Gruppierung von Kommandos

Symbol	Bedeutung
`;`	Bewirkt, daß die einzelnen Kommandos, die durch dieses Zeichen getrennt sind, nacheinander ausgeführt werden. Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem des zuletzt ausgeführten.
`&`	Das Anhängen dieses Zeichens an ein Kommando bewirkt, daß die Anweisung im Hintergrund ausgeführt wird; D.h. die Shell meldet sich sofort wieder mit dem Shell-Prompt eingabebereit, so daß weitere Kommandos gestartet werden können, noch während das mit `&` gestartete Kommando weiter ausgeführt wird.
`k1 && k2`	Diese Anweisung fragt den Status des Kommandos `k1` ab und arbeitet in Abhängigkeit des Ergebnisses das nächste Kommando `k2` ab oder überspringt es. Das nächste Kommando `k2` wird nur dann bearbeitet, wenn das vorhergehende einen Rückgabewert von `0` (OK) hatte. Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem des zuletzt ausgeführten.
`k1 || k2`	Wie `&&` fragt auch diese Anweisung den Rückgabewert des Kommandos `k1` ab. Das nächste Kommando `k2` wird jedoch nur dann bearbeitet, wenn das vorangegangene einen Rückgabewert ungleich `0` (Fehler) hatte. Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem des zuletzt ausgeführten.
`(kommando)`	Das Kommando wird in einer Sub-Shell ausgeführt. Eine neue Shell wird gestartet, in der die Ausführung des Kommandos erfolgt. Anschließend wird diese Shell wieder beendet. Dadurch, daß eine neue Shell gestartet wird, wirken sich Veränderungen auf Shell- und Environment-Variablen durch `kommando` nur auf die neu gestartete Shell und nicht auf die Shell aus, in der die Anweisung `(kommando)` eingegeben wurde!

Die Wirkungsweise der Gruppierungsanweisungen läßt sich am besten anhand einiger Beispiele beschreiben:

- `mr@hamlet>` **`ls; cd ..; ls; cd`**
  In diesem Beispiel werden die Kommandos von links nach rechts nacheinander ausgeführt. Das jeweils folgende Kommando kann erst ausgeführt werden, wenn das vorangegangene beendet ist.

- `mr@hamlet>` **`sleep 100&`**
  `mr@hamlet>` **`ls`**
  In diesem Beispiel wird an das Kommando `sleep`, das nichts anderes macht als die in Sekunden angegebene Zeit zu warten, ein `&` angehängt. Dadurch erscheint sofort wieder der Shell-Prompt und ein weiteres Kommando (hier `ls`) kann eingegeben werden. Beide Kommandos laufen jetzt parallel.

- `mr@hamlet> ls && echo "Hat geklappt !"`
  `Mail etc tmp`
  `Hat geklappt !`
  In diesem Beispiel erscheinen als Ausgabe zunächst die Dateien im aktuellen Verzeichnis des Benutzers. Da das Kommando `ls`, im eigenen Home-Verzeichnis aufgerufen, erfolgreich ablaufen kann, ist der Rückgabestatus `0`. Daher wird anschließend das Kommando `echo` ausgeführt.

- `mr@hamlet> ls /abcde && echo "Hat geklappt !"`
  `/abcde: No such file or directory`
  Dadurch, daß bei der Ausführung von `ls` hier ein Fehler provoziert wurde, indem ein nicht existierendes Verzeichnis aufgelistet werden sollte, ist der Rückgabestatus von `ls` ungleich `0`. Daher wird das hinter `&&` stehende Kommando *nicht* ausgeführt. Als Ausgabe erscheint daher lediglich die Fehlermeldung von `ls`.

2.1.8 Arbeiten mit Sub-shells

Sub-Shells sind immer dann von Nutzen, wenn man einige Kommandos in einer anderen Umgebung (in einem anderen Arbeitsverzeichnis oder mit anderen Werten für Shell- oder Environment-Variablen) ablaufen lassen möchte, ohne dabei die Umgebung der Shell, aus der heraus das Kommando aufgerufen wird, zu verändern.

Eine Sub-Shell wird dadurch gestartet, daß man in der Kommandozeile runde Klammern verwendet:

```
mr@hamlet> pwd; (cd /tmp; pwd); pwd
/home/mr
/tmp
/home/mr
```

In diesem Beispiel werden die Kommandos `cd /tmp; pwd` in einer Sub-Shell ausgeführt. Die öffnende runde Klammer bewirkt, daß eine neue Shell gestartet wird, bevor die rechts davon stehenden Kommandos gestartet werden. An der schließenden Klammer wird die Shell wieder beendet.

Betrachtet man sich die Ausgaben des Beispiels einmal genauer, sieht man die Auswirkung der Sub-Shell. Die erste Ausgabe des Kommandos `pwd` liefert das aktuelle Verzeichnis des Benutzers `/home/mr`. Nach Ausführung des Kommandos `cd /tmp` in der Sub-Shell ist das Arbeitsverzeichnis dieser Sub-Shell jetzt `/tmp`, wie der zweite Aufruf des Kommandos `pwd` bestätigt. Obwohl jetzt kein `cd`-Kommando mehr ausgeführt wird, liefert das dritte `pwd`-Kommando wieder den Pfad `/home/mr`. Der Grund dafür liegt darin, daß die Kommandos, die in der Sub-Shell ausgeführt werden, zwar das Arbeitsverzeichnis dieser neu gestarteten Shell verändern, jedoch nicht das Arbeitsverzeichnis der Shell beeinflussen können, in der das erste und das letzte `pwd`-Kommando aufgerufen werden. Obige Kommandozeile ist vergleichbar mit folgenden Anweisungen (Text der in einer