2.1 Grundlagen der Befehlszeile – Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.1 Grundlagen der Befehlszeile
Lektion:	2 von 2

Einführung

Alle Shells verwalten während der **Shell-Sitzungen einen Satz von Statusinformationen**, die sich während der Sitzung **ändern können** und das Verhalten der Shell beeinflussen. Programme nutzen diese Daten, um Aspekte der Systemkonfiguration zu bestimmen. Die meisten dieser Daten werden in sogenannten ***Variablen* gespeichert**, die wir in dieser Lektion behandeln.

Variablen

Variablen sind Speicher für Daten, beispielsweise Text oder Zahlen, auf die später zugegriffen werden kann. Variablen haben einen Namen, über den man auf sie zugreift, auch wenn sich ihr Inhalt ändert. Sie sind zudem ein wichtiges Werkzeug in den meisten Programmiersprachen.

In den meisten Linux-Shells gibt es **zwei Arten von Variablen**:

Lokale Variablen

Diese Variablen stehen **nur in der jeweiligen Shell-Sitzung zur Verfügung**. Wenn Sie eine lokale Variable erstellen und dann ein anderes Programm von dieser Shell aus starten, ist die Variable nicht mehr zugänglich. Da sie **nicht an Subprozesse vererbt** werden, nennt man diese Variablen *lokale Variablen* (***local variables***).

Umgebungsvariablen

Diese Variablen stehen sowohl in einer **Shell-Sitzung als auch in Unterprozessen zur Verfügung**, die aus dieser Shell-Sitzung hervorgegangen sind. Diese Variablen werden etwa genutzt, um Konfigurationsdaten an Befehle zu übergeben. Da Programme auf diese Variablen zugreifen können, heißen sie ***Umgebungsvariablen* (*environment variables*)**. Die Mehrheit der Umgebungsvariablen ist in **Großbuchstaben** geschrieben (**beispielsweise `PATH`, `DATE`, `USER`**). Ein Satz von Standardumgebungsvariablen liefert zum Beispiel Informationen über das Home-Verzeichnis oder den Terminaltyp des Benutzers.

Diese Arten von Variablen werden auch als *Gültigkeitsbereich von Variablen* (*variable scope*) bezeichnet.

Note **Variablen sind nicht persistent**. Wird die **Shell**, in der sie gesetzt wurden, geschlossen, **gehen alle Variablen und deren Inhalt verloren**. Die meisten Shells stellen Konfigurationsdateien mit Variablen bereit, die beim Start einer neuen Shell gesetzt werden. **Variablen, die dauerhaft gesetzt werden sollen, müssen zu einer dieser Konfigurationsdateien hinzugefügt werden**.

Manipulation von Variablen

Als Systemadministrator müssen Sie sowohl lokale als auch Umgebungsvariablen erzeugen, ändern oder entfernen.

Arbeiten mit lokalen Variablen

Sie setzen eine **lokale Variable** mit dem Operator `=` (Gleichheitszeichen). Eine einfache Zuweisung erstellt eine lokale Variable. Achtung, **keine Leerzeichen** beim Operator =.

```
$ greeting=hello
```

Sie können jede Variable mit dem Befehl `echo` anzeigen, der normalerweise den Text im Argumentabschnitt anzeigt:

```
$ echo greeting
greeting
```

Um auf den **Wert der Variablen zuzugreifen**, müssen Sie ein `$` **(Dollarzeichen)** vor den Variablennamen setzen.

```
$ echo $greeting
hello
```

Wie man sieht, wurde die Variable erzeugt. Öffnen Sie nun eine weitere Shell und versuchen Sie, den Inhalt der Variable anzuzeigen:

```
$ echo $greeting
```

Es wird nichts angezeigt. Hier wird deutlich, dass Variablen immer nur in einer bestimmten Shell existieren. Da die **neue Shell** in einem neuen Prozess läuft, **erbt** sie **keine lokalen Variablen** von ihrem übergeordneten Prozess:

Um eine Variable zu entfernen, nutzen Sie den Befehl `unset`:

```
$ echo $greeting
hey
$ unset greeting
$ echo $greeting
```

Note **`unset`** verlangt den Namen der Variablen als Argument. Daher dürfen Sie dem Namen **kein `$` voranstellen**, da dies die Variable auflösen und den Wert der Variablen an `unset` anstelle des Namens der Variablen übergeben würde.

Arbeiten mit Umgebungsvariablen

Um eine **Variable für Unterprozesse verfügbar zu machen**, verwandeln Sie sie von einer lokalen in eine globale oder Umgebungsvariable, und zwar mit dem Befehl `export`. Wird sie über den Variablennamen aufgerufen, wird diese Variable der Umgebung der Shell hinzugefügt:

```
$ greeting=hello
$ export greeting
```

Note Noch einmal: Stellen Sie sicher, dass Sie kein `$` beim Aufruf von `export` verwenden, da Sie den Variablennamen, nicht den Inhalt der Variablen übergeben wollen.

Eine **einfachere Möglichkeit**, die Umgebungsvariable zu erstellen, besteht darin, beide der oben genannten Methoden zu kombinieren, indem man den **Variablenwert im Argumentteil des Befehls zuweist**.

```
$ export greeting=hey
```

Lassen Sie uns noch einmal überprüfen, ob die Variable für Unterprozesse zugänglich ist:

```
$ export greeting=hey
$ echo $greeting world
hey world
$ bash -c 'echo $greeting world'
hey world
```

Eine weitere Einsatzmöglichkeit von **Umgebungsvariablen** besteht darin, sie **vor Befehle zu setzen**. Testen wir das mit der Umgebungsvariablen `TZ`, die die Zeitzone enthält. Wir nutzen die Variable, um dem Befehl `date` anzuzeigen, welche Zeitzone er nutzen soll:

```
$ TZ=EST date
Thu 31 Jan 10:07:35 EST 2019
$ TZ=GMT date
Thu 31 Jan 15:07:35 GMT 2019
```

Mit dem Befehl `env` zeigen Sie alle Umgebungsvariablen an.

Die Variable `PATH`

Die Variable `PATH` ist eine der wichtigsten Umgebungsvariablen in einem Linux-System. Sie speichert eine **Liste von durch Doppelpunkt getrennten Verzeichnissen** mit ausführbaren Programmen, die als Befehle aus der Linux-Shell aufrufbar sind.

```
$ echo $PATH
/home/user/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games
```

Um ein neues Verzeichnis an die Variable anzuhängen, setzen Sie einen **Doppelpunkt** (`:`):

```
$ PATH=$PATH:new_directory
```

Hier ein Beispiel:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ PATH=$PATH:/home/user/bin
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin
```

Wie Sie sehen, wird `$PATH` mit dem neuen Wert verwendet, der `PATH` zugewiesen wurde. Die Variable wird während der Befehlsausführung aufgelöst und sorgt dafür, dass der ursprüngliche Inhalt der Variablen erhalten bleibt. Sie können diese Zuweisung natürlich auch bei anderen Variablen durchführen.

```
$ mybin=/opt/bin
$ PATH=$PATH:$mybin
$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/user/bin
:/opt/bin
```

Die Variable `PATH` ist mit Vorsicht zu behandeln, da sie für die Arbeit auf der Kommandozeile entscheidend ist. Betrachten wir die folgende Variable `PATH`:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Um herauszufinden, wie die Shell einen bestimmten Befehl aufruft, führen wir `which` mit dem Namen des Befehls als Argument aus. So ermitteln wir z.B., wo `nano` gespeichert ist:

```
$ which nano
/usr/bin/nano
```

Offenkundig befindet sich die ausführbare Datei `nano` im Verzeichnis `/usr/bin`. Entfernen wir das Verzeichnis aus der Variablen und überprüfen wir, ob der Befehl noch funktioniert:

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games
```

Schauen wir noch einmal nach dem Befehl `nano`:

```
$ which nano
which: no nano in
(/usr/local/sbin:/usr/local/bin:/usr/sbin:/sbin:/bin:/usr/games)
```

Offensichtlich wird der Befehl nicht gefunden, also nicht ausgeführt. Die Fehlermeldung nennt auch den Grund, warum der Befehl nicht gefunden und an welchen Stellen er gesucht wurde.

Fügen wir die Verzeichnisse wieder hinzu und versuchen, den Befehl erneut auszuführen.

```
$ PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
$ which nano
/usr/bin/nano
```

Jetzt funktioniert unser Befehl wieder.

Tip Die Reihenfolge der Elemente in `PATH` definiert auch die Reihenfolge der Suche: Die erste passende ausführbare Datei, die beim Durchlaufen der Pfade gefunden wird, wird ausgeführt.

Geführte Übungen

1. Erzeugen Sie die lokale Variable `number`.
2. Erzeugen Sie die Umgebungsvariable `ORDER` mit Hilfe der beiden oben genannten Methoden.
3. Lassen Sie sowohl Namen als auch Inhalt der Variablen anzeigen.
4. Welche Reichweiten (Scope) haben die zuvor erzeugten Variablen?

Offene Übungen

1. Erzeugen Sie eine lokale Variable `nr_files` und weisen Sie die Anzahl der Zeilen in der Datei `/etc/passwd` zu. Hinweis: Schauen Sie sich den Befehl `wc` und die Befehlsersetzung an und vergessen Sie nicht die Anführungszeichen.
2. Erzeugen Sie eine Umgebungsvariable `ME`. Weisen Sie `USERNAME` als Wert zu.
3. Fügen Sie den Wert der Variablen `HOME` an `ME` mit dem Trennzeichen `:` an und zeigen Sie den Inhalt der Variablen `ME` an.
4. Erstellen Sie unter Verwendung des obigen Datumsbeispiels eine Variable namens `today` und weisen Sie das Datum für eine Zeitzone zu.
5. Erzeugen Sie eine weitere Variable namens `today1` und weisen Sie ihr das Systemdatum zu.

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Arten von Variablen
- Wie man Variablen erzeugt
- Wie man Variablen manipuliert

Befehle, die in den Übungen verwendet werden:

`env`
: Zeigt die aktuelle Umgebung an.

`echo`
: Gibt Text aus.

`export`
: Macht lokale Variablen für Unterprozesse verfügbar.

`unset`
: Entfernt eine Variable.