

UNIX-Shell als User-Interface

Allgemeines

Die Shell ist das ursprüngliche User-Interface zum Unix-Betriebssystem. In die Shell können Befehle über die Tastatur eingegeben werden, die Ein- und Ausgabe erfolgt in einem Terminalfenster (Konsole). Die Shell kapselt die Kern-Funktionen des Betriebssystems und stellt eine Alternative zu graphischen Benutzeroberflächen (z.B. Gnome für Unix-Systeme).

Für Unix gibt es eine Reihe von Shells, die sich funktional leicht unterscheiden:

- Bourne Shell (sh): Ist die Standard-Shell von Unix, die schon auf den ersten Unix-Systemen zur Verfügung stand. Sie ist auf nahezu allen Unix-Derivaten verfügbar.
- Korn Shell (ksh): Obermenge der Bourne Shell.
- C Shell (csh): Hierbei handelt es sich um die heute am weitesten verbreiteten Shell.
- Turbo C Shell (tcsh): Obermenge der C Shell.
- Bourne Again Shell (bash): Wird standardmäßig auf vielen Linux-Rechnern oder unter cygwin gestartet. Die bash ist unter GNU Lizenz und auf Linux System die Standard Shell. Die bash ist eine Obermenge der Bourne shell und erweitert diese u.a. um:
 - Command Line Editing: Kommandos können aus der Shell wie bei der Turbo C Shell heraus komfortabel editiert werden.
 - Job Control (Jobs können z.B. im Hintergrund gestartet werden, übernommen aus der csh)

Im Praktikum wird nur auf die bash Shell eingegangen. Eine Shell kann aus einer anderen Shell (daher auch der Begriff) heraus gestartet werden. So kann die bash-Shell aus der csh-Shell mit dem Kommando bash gestartet werden. Man spricht dann auch von dem Starten einer Sub Shell.

Hinweis für Windows-Nutzer ohne Linux Installation: Mit dem cygwin-Paket kann eine Linux-ähnliche Umgebung unter Windows aufgesetzt werden. Alle Praktikumsaufgaben sind auch auf einem cygwin System zu lösen. Entscheidend für das Bestehen des Praktikums ist allerdings, dass Ihre Lösungen auf den Rechnern in den Praktikumsräumen laufen. Vergewissern Sie sich, dass Ihr Code portierbar ist! Link zum Download: <http://www.cygwin.com>

Konfiguration der bash

Beim Start der bash Shell wird von dieser nach Initialisierungsdateien im Home-Verzeichnis gesucht. Die gefundenen Dateien werden dann ausgeführt. Hinsichtlich der Funktionsweisen und Abhängigkeiten dieser Dateien ist folgendes zu beachten:

- **.bash_profile** Wird ausgeführt, wenn die Shell eine Login Shell (also keine Sub Shell) ist. Alternativ wird nach `.bash_login` oder `.profile` Heimverzeichnis gesucht. Es wird aber nur eine dieser drei Dateien (in der Suchreihenfolge `.bash_profile` -> `.bash_login` -> `.profile`) ausgeführt.
- **.bashrc** Diese Datei wird ausgeführt, wenn eine neue Shell gestartet wird (Sub Shell). Hier können alle benutzerspezifischen Einstellungen für die bash Shell vorgenommen werden (z.B. Konfiguration von Kommando-Aliasen).
- **.bash_logout** Wird ausgeführt, wenn eine Login Shell verlassen wird. Hier können z.B. temporäre Dateien gelöscht werden oder ähnliches.

Variablen der bash

In der Shell sind die folgenden Arten von Variablen zu unterscheiden:

- Globale Variablen: Stehen auch allen Unterskripten zur Verfügung. Die globalen Variablen (Environment Variablen) können mit dem Kommando `env` (für Environment) angezeigt werden.
- Lokale Variablen: Nur gültig in der aktuellen Shell. Lokale Variablen können mit `set` angeschaut werden. (`set` listet auch die globalen Variablen.)
- Systemvariablen: Bestimmte Variablen haben in der bash eine besondere Bedeutung. z.B.:
 - **PATH**: Durch „:“ getrennte Liste von Suchpfaden. Hier kann auch ein eigener Pfad angefügt werden. z.B.:

```
export PATH=\$PATH:\$HOME/scripts
```

Aus Sicherheitsgründen sollten eigene Erweiterungen des Suchpfades immer ans Ende des Suchpfades gestellt werden! (Warum? Was passiert, wenn ein „böser“ User eine ausführbare Datei namens „ls“ im Verzeichnis „scripts“ ablegt?) Auch das aktuelle Verzeichnis (Arbeitsverzeichnis) „.:“ kann – muss aber nicht – in der PATH Variable stehen. (Warum ist es aus Sicherheitsgründen besser, wenn das aktuelle Verzeichnis nicht im Suchpfad steht?)

- **HOME**: Das HOME-Verzeichnis eines jeden Benutzers. Nach dem erfolgreichen Login in einer interaktiven Shell befindet sich der Nutzer in diesem Verzeichnis (was man mit `pwd` kontrollieren kann).
- **PS1**: Der primäre Prompt String. Der Inhalt bestimmt, wie der user Prompt aussieht.

Eine komplette Liste der Systemvariablen findet sich z.B. im (Bash Beginners Guide) im Kapitel 3.2. Wie werden Variablen erzeugt?

- `export VARNAME=value`
Erzeugt eine globale (Environment) Variable.
- `VARNAME=value`
Erzeugt eine lokale Variable.
- `unset VARNAME`
Löscht eine lokale oder globale Variable.

Als Konvention sollten alle Variablen in Großbuchstaben geschrieben. Auf den Inhalt einer Variable kann über `$VARNAME` zugegriffen werden. Z.B.:

- `echo $VARNAME`
Schreibt den Inhalt der Variable `VARNAME` auf die Standardausgabe (stdout).
- `ls $HOME`
Gibt unabhängig vom Arbeitsverzeichnis den Inhalt des Home-Verzeichnisses auf stdout aus.

Die Übergabeparameter für Skripte werden innerhalb des Skripts als String-Variablen behandelt. Die Variablennamen der Übergabeparameter im Skript können referenziert werden durch `$1`, `$2` ...

Arbeiten mit der bash

Mit Hilfe des man Kommandos können zu jedem externen Kommando Hilfestellungen geholt werden (so genannte Man Pages). Es gibt Hilfeseiten sowohl über die externen Befehle der bash wie auch über C Standardfunktionen. Neben externen Kommandos enthält die bash auch eine Reihe interner Kommandos. Mit dem `help` Befehl kann Hilfestellung zu diesen Befehlen aufgerufen werden.

Beispiele:

- `man ls` : Zeigt Hilfe über den `ls` Befehl an
- `help` : Listet alle internen Befehle der bash
- `help cd` : Zeigt Hilfe über den internen `cd` Befehl

Namen, Wildcards, Metazeichen: In Dateinamen können Zeichen durch Metazeichen (Wildcards) ersetzt werden. Auf diese Weise können reguläre Ausdrücke gebildet werden, die Mengen z.B. von Dateien beschreiben, die ein spezielles Namensformat aufweisen. Die Metazeichen haben folgende Bedeutung:

Metazeichen und Ihre Bedeutung:

`?` : Einzelnes Zeichen

`*` : Folge von beliebigen Zeichen

`[set]` : Jedes Zeichen in set

`[!set]` : Jedes Zeichen, das nicht in set vorkommt

Beispiele:

- [abc] a, b oder c
- [a-c] a, b oder c
- [a-z] alle kleinen Buchstaben
- [!0-9] alle Zeichen, die keine Ziffern sind

Elementare Befehle der bash

Unix	Windows	Erläuterung
ls -l	dir	Liste aller Dateien im aktuellen Verzeichnis
ls -cf	dir /w	Kurze Liste aller Dateien
pwd	cd	Anzeige des aktuellen Verzeichnisses
cd	cd	Wechseln der aktuellen Verzeichnisses
cat	type	Ausgabe einer Datei auf der Standard-Ausgabe
more	more	Alternativ kann auch der Befehl ‚less‘ verwendet werden.
echo	echo	Gibt Argument auf der Standard-Ausgabe aus.
touch	N/A	Erzeugen einer leeren Datei oder Aktualisierung des Zeitstempels
type	N/A	Finden einer ausführbaren Datei
cp	copy	Kopieren einer Datei
ln	N/A	Verknüpfung einer Datei oder eines Verzeichnisses
mv	move, ren	Umbenennen einer Datei
mkdir	md	Erstellen eines Ordners
rmdir	rd	Löschen einer leeren Ordners
rm	del	Löschen einer Datei
rm -r	deltree	Rekursives Löschen eines Verzeichnisses und der Unterverzeichnisse
man	help	Hilfe zu Kommando
chmod	attrib	Ändern der Dateiattribute
chown	N/A	Ändern des Besitzers einer Datei
nano	editor	Starten des Systemeditors
df	(chkdsk)	Anzeige des freien Plattenplatzes
ps -edf	N/A	Anzeige aller laufenden Prozesse
kill	N/A	Beenden eines Prozesses
passwd	N/A	Ändern des Passwortes

Eine genaue Beschreibung zu den Befehlen wird mit dem Befehl

man <Befehl>

Grundlegende I/O Konzepte Die Datei-basierte Verarbeitung von Daten ist eines der Grundkonzepte von Unix. Alle Ein- und Ausgabe-Operationen (auch das Drucken oder Beschreiben von externen Datenträgern) wird über das Dateisystem abgewickelt. Man spricht in diesem Zusammenhang auch von File Mapped I/O.

Je nach Art der Nutzung werden die folgenden Dateitypen unterschieden:

- reguläre Dateien: Text-/Binärdateien, allgemein eine Datei mit wahlfreiem Zugriff auf einem Datenträger.
- Verzeichnis: Enthält weitere Dateien.
- Spezielle Datei (sog. Special File): z.B block- oder zeichenorientierte Gerätedatei. Die serielle Schnittstelle, die Tastatur oder der Bildschirm sind als Special Files im Dateisystem eingebunden.

Jeder Prozess besitzt drei besondere Ausgabekanäle. Diese sind vordefinierte Dateien, die immer geöffnet sind.

- stdin (0) : Standard Eingabekanal, i.a. das Terminal / Tastatur.
- stdout (1) : Standard Ausgabekanal, i.a. das Terminal / Bildschirm (printf oder echo benutzen diesen Kanal)
- stderr (2) : Fehler-Ausgabe Kanal. Standardmäßig wird stdout benutzt.

Die Ziffern entsprechen den sog. File Handles, welche die unterschiedlichen Dateisysteme indizieren.

Die Ein- und Ausgabe kann umgelenkt werden.

- Eingabe: Ein „<“ Zeichen lenkt die Eingabe von stdin um.

Beispiel:

sort : erwartet Eingabe von stdin (der Tastatur).

sort < filename : liest Daten aus der Datei filename

- Ausgabe: Ein „>“ Zeichen leitet die Ausgabe von stdout um.

Beispiel:

ls -al : Ausgabe auf stdout (Bildschirm).

ls -al > filename : Ausgabe in die Datei filename.

- Wird „>>“ benutzt, wird die Ausgabe an eine existierende Datei angehängt, ansonsten wird die Datei überschrieben.
- „>&“ lenkt stderr und stdout um.
- „File handle>“ lenkt einen bestimmten File handle um, also „1>“ lenkt stdout um, „2>“ lenkt stderr um.

- Pipeline: Die Ausgabe (stdout) eines Kommandos wird in die Eingabe (stdin) eines anderen Kommandos umgeleitet. Das Konstrukt wird als Pipe „|“ bezeichnet.

Beispiel:

ls -al | more : Die Ausgabe des ls Befehls wird dem more Kommando übergeben.

Weitere Informationen zur bash sind in der Dokumentation von Garrels zu finden: Bash Beginners Guide.