Themen 3. Teil:

- Shell-Skripten
- Das grafische Fenstersystem X-Windows
- Einführung in einige Fenstermanager

Shell-Skripten

- Ein Shell-Skript ist die Zusammenfassung von Kommandos in einer Textdatei.
- Diese Textdatei wird wie ein ablaufbares Programm behandelt.
- Es werden von einer Shell auch programmierähnliche Konstrukte, wie Schleifen, Bedingungen, Parameter und Variablen zur Verfügung gestellt.

Ein Shell-Skript wird mit einem normalen ASCII-Editor (z.B. `emacs`) erstellt. Das Skript *muss* in der ersten Zeile den Hinweis auf die verwendete Shell in folgender Form enthalten:

```
#!/bin/sh
```

Dabei teilen die beiden ersten Zeichen dem Betriebssystem mit, dass es sich um ein Shell-Skript handelt, und der folgende Pfad inklusive Shell teilen dem System mit, für welche Shell das Skript erstellt wurde.

Kennt man den Pfad zur Shell nicht, so kann man zwei Kommandos ausprobieren, die einem den Ort der Shell im Dateibaum anzeigen:

```
type <Shell-Name>
```

```
locate <Shell-Name>
```

```
which <Shell-Name>
```

Alle Befehle lassen sich im Übrigen auch auf alle anderen ausführbaren Programme. die auf einem System installiert sind, anwenden (dennoch muessen beide Befehle nicht zwingend auf einem System vorhanden sein)!

Da Unix unter den Zugriffsrechten auch ein Ausführungsrecht besitzt, das normalerweise beim Anlegen von Textdateien (nichts anderes ist ein erstelltes Shell-Skript zunächst) nicht automatisch gesetzt wird, muss dieses Recht erst noch aktiviert werden:

```
chmod u+x <Shell-Skript>
```


Einfaches Beispiel

```
#!/bin/sh

echo Ich bin ein einfaches Shell-Skript!

exit 0
```

Die eingefügten Leerzeilen sind nicht verpflichtend. Ebenso kann die letzte Anweisung weggelassen werden. Bei komplexen Skripten kann der `exit`-Status aber nützlich sein, um Informationen darüber zu erhalten, ob das Skript korrekt abgearbeitet wurde.

Kommandos werden entweder durch Zeilenwechsel oder durch Semikolon getrennt.

Parameterübergabe beim Aufruf

Man kann einem Skript beim Aufruf (beliebig) viele Parameter übergeben, die innerhalb des Skriptes verarbeitet werden können.

```
skript [p1] [p2] ... [pn]
```

Diese Parameter werden als Text interpretiert, der innerhals des Skriptes in den Variablen

```
$1, $2, ... $9
```

gespeichert wird. Auch hier ist das Dollarzeichen $ wieder die Dereferenzierung auf den Inhalt einer Variablen (wie auch bei den Umgebungsvariablen).

Das Shell-Skript selbst kann nur die Parameter 1-9 ansprechen.

Das Kommando `shift`

Mittels der Angabe von

```
shift n
```

Kann der Zugriffsbereich innerhalb des Skriptes um den Betrag `n` vershoben werden, sodass auch Parameter jenseits von 9 angesprochen werden können. Der Parameter `$1` entspricht dann dem `n+1`-ten beim Aufruf angegebenen Parameter.

Beispiel

```
#!bin/sh

echo $2 $1
shift 9
echo $1

exit 0
```

Dabei spielt es keine Rolle bzw. es führt zu keinem Fehler, wenn mehr oder weniger Parameter übergeben werden als innerhalb des Skriptes verarbeitet werden.

Umdefinieren von Parametern innerhalb des Skriptes

Mit dem Kommando

```
set Parameter1 Parameter2 ...
```

können die übergebenen Parameter neu definiert werden. Dabei entspricht die Reihenfolge nach dem `set`-Kommando der Reihenfolge bei der Übergabe.

Beispiel

```
#!/bin/sh

echo $1 $2
set Hallo Welt!
echo $1 $2

exit 0
```


Besondere Parameter innerhalb des Skriptes

$0 Name des Skriptes

$# Anzahl der übergebenen Parameter

$* Die übergebenen Parameter werden als eine Zeichenkette interpretiert.

$$ PID-Nummer des aktuellen Prozesses.

Variablen in Shell-Skripten

Durch den Zuweisungsoperator = können auch Variablen definiert und mit Werten besetzt werden:

```
#!/bin/sh

variable="Hallo Welt!"
echo $variable

exit 0
```

Wichtig sind die Anführungszeichen, da sonst durch das Leerzeichen `Welt!` als nicht zu `Hallo` gehörend betrachtet würde. Eine Fehlermeldung wäre die Folge.

Variablen mit der Ausgabe eines Kommandos besetzen

Soll beispielsweise eine Variable `Datum` mit dem aktuellen Datum des Systems besetzt werden, so kann `Datum` die Ausgabe des Kommandos `date` zugewiesen werden. Dazu muss das auszuführende Kommando innerhalb der Shell in sogenannte Backticks gesetzt werden:

```
Datum=‘date‘
```

For-Schleife (1)

Um den Teil eines Skriptes mehr als einmal, abhängig von einer gewissen Bedingung, durchlaufen zu können, gibt es die `for`-Anweisung:

```
for <variable>
do
        Kommandos ...
done
```

Hierbei wird die Schleife sooft durchlaufen, wie Argumente beim Aufruf des Skriptes übergeben wurden. Mit jedem Durchlauf wird die `variable` auf den Wert des nächstes Argumentes gesetzt.

Beispiel

```
#!/bin/sh

for parameter
do
        echo $parameter
done

exit 0
```

For-Schleife (2)

Man kann auch ein konkretes Ziel angeben, mit dem die `variable` in der Schleife verglichen bzw. gesetzt werden soll:

```
for <variable> in <argumente>
do
        Kommandos ...
done
```


Beispiel

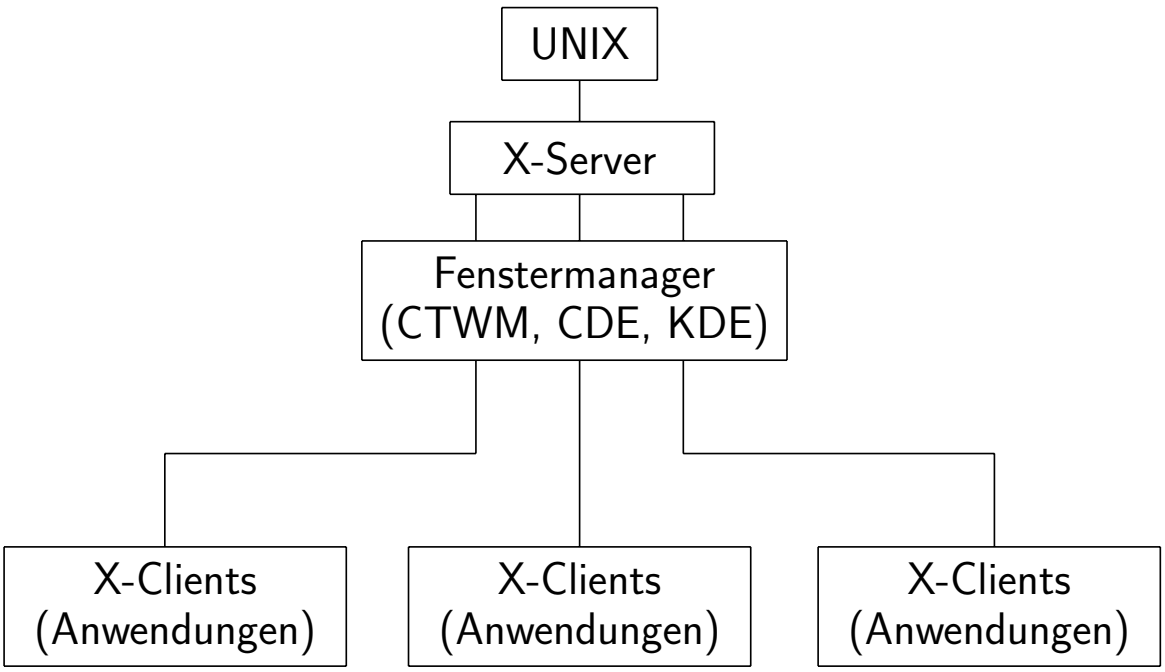
```
#!/bin/sh

for parameter in 9 8 7 6 5 4 3 2 1
do
         echo $parameter
done

exit 0
```

Das grafische Fenstersystem X-Windows

- Architekturunabhängige grafische Benutzer-Schittstelle zum Betriebssystem
- Client-Server-Konzept
- Auf X setzt der sogenannte Fenstermanager auf, der das Aussehen und die Handhabung von X festlegt.



Der X-Server

- Der X-Server ist von der Hardware abhängig, ist aber für fast alle Plattformen erhältlich.
- Der X-Server kommuniziert mit den X-Clients über das standardisierte X-Protokoll.
- Er steuert die Farbtiefe und die Auflösung.
- Er erzeugt die Fenster, in denen die gestarteten Applikationen laufen.
- Er kann auf entfernten Rechnern gestartete Applikationen auf dem lokalen Rechner darstellen.
- Er ist **nicht** für die Darstellung von Menüleisten, Knöpfen, Kontextmenüs etc. zuständig.
- Ein X-Windows kann nicht ohne ihn gestartet werden.

Vorteile von X-Windows

- Durch die Netzwerkfähigkeit können beispielsweise rechenintensive Programme auf entsprechend leistungssarken Rechnern laufen, während die grafische Ausgabe dieser Programme auf dem lokalen Rechner stattfindet.

- Programme können über systemspezifische Grenzen hinweg (Windows,Apple,Unix/Linux) verwendet werden.

- Sind Programme nicht auf dem lokalen Rechner verfügbar, aber auf anderen Rechnern vorhanden, so können sie dort gestartet werden.

Der Fenstermanager

- Der Fenstermanager besteht aus zwei Elementen:
  - Eine Bibliothek, die die Bedienelemente des Oberfläche zur Verfügung stellt (QT, Motif, GNOME).
  - Die Oberfläche, die die Bedienelemente verwaltet und auf Maus- und Tastaturereignisse reagiert (CTWM, CDE, KDE ...).
- Der Fenstermanager ist für das sogenannte „Look-and-Feel“ der Oberfläche verantwortlich.
- Es gibt eine verwirrende Vielzahl von Fenstermanagern.
- Die meisten Fenstermanager benutzen „freie“ Bibliotheken, d. h. diese Bibliotheken sind kostenlos, ihre Quellcodes sind frei zugänglich und können nach belieben verändert werden. Lediglich QT ist eine lizensierte Bibliothek, die vom Fenstermanager KDE genutzt wird.
- KDE und Gnome sind die neuesten Fenstermanager, die versuchen, dem Bedienungsstandard von Windows möglichst nahe zu kommen.

Der X-Client

- Der X-Client ist die Anwendung (Programm), die gestartet wird.
- Über das X-Protokoll kommuniziert dieser Client mit dem X-Server, der wiederum Ein- und Ausgaben des Clienten verwaltet und an das System weiterreicht.
- Der X-Client muss nicht unbedingt auf dem lokalen Rechner, auf dem der X-Server läuft, gestartet werden. Er kann auch auf einem entferneten Rechner gestartet werden, und seine Ein- und Ausgaben werden wiederum über das X-Protokoll verwaltet.
- Der X-Client muss wissen, auf welchem Rechner mit entsprechendem X-Server er dargestellt werden soll. Die Umgebungsvariable `DISPLAY` ist dafür zuständig.

Farben und Schriftsätze von X-Clients beim Start festlegen:

Um eine Applikation mit speziellen Farben und Schriftsätzen zu starten, muss man die Applikation in einem XTerminal wie bisher aufrufen. Drei zusätzliche Optionen ermöglichen die Auswahl der Vorder- bzw. Hintergrundfarbe und des Zeichensatzes:

```
-fg <Farbe>  (Vordergrund)
-bg <Farbe>  (Hintergrund)
-fn <Font>   (Zeichensatz)
```

Der Befehl

```
showrgb
```

gibt alle möglichen im System vorhandenen Farben aus, und der Befehl

```
xlsfonts
```

gibt alle möglichen Zeichensätze aus.

Zeichensatz-Format unter X:

Das Zeichensatzformat unter X hat die Form:

```
-adobe-courier-bold-o-normal-10-100-75-75-m-60-iso88591-1
   1       2       3   4    5    6   7   8   9  10 11      12
```

1	Hersteller	7	Schriftgröße in Pixel
2	Schriftfamiliefamilie	8	Schriftgröße in zehntel Punkt
3	Schriftstärke	9	Auflösung in dpi
4	Schriftschnitt	10	fester oder proportionaler Zeichenabstand
5	Zeichenabstand	11	Durchschnittl. Zeichenabstand in zehntel Punkt
6	Schriftstil (z. B. Sans Serif)	12	Zeichensatz

Zeichensätze ausprobieren:

Mit dem Programm

```
xfontsel
```

kann man verschiedene Schriftsätze interaktiv durchprobieren. Klickt man auf den `select`-Knopf, so kann man mit der Maus das Zeichensatz-Format durch Druck auf die mittlere Maustaste auf Konsole-Ebene (oder in einem Editor) einfügen.

Beipiel für Farben und Fonts:

```
xterm -fg red -bg green -fn -dec-terminal-
bold-r-normal-*-30-400-75-*-c-140-dec-dectech
```

X-Konfigurationsdateien

- `.X11initrc` Initialisierung des X-Servers X11R6
- `.Xdefaults` Voreinstellungen für die Fenstergeometrie
- `.Xresources` Voreinstellungen für einige Anwendungen

Initialisierung am RRZN durchführen:

Das Kommando

```
gut [Option]
```

initialisiert abhängig von der gewählten `Option` die entsprechenden Konfigurationsdateien. Mittels

```
ungut
```

wird bei Bedarf der vorhergehende Zustand wieder hergestellt.

Die wichtigsten Optionen von gut sind:

`all`	alle Konfigurationsdateien
`X11`	`.X11initrc .Xresources`
`XDM`	`.xsession .Xresources`
`X11+XDM`	X11- und XDM-Dateien
`Windowmanager`	`.ctwmrc .mwmrc`
`OpenWindows`	`.Xdefaults .openwin-menu .xinitrc openwin-init openwin-menu-communications openwin-menu-programs openwin-menuutilities`
`System`	`.epifile .kshrc .profile`
`Sysbull`	`.rrznnews`
`Netscape`	`MCOM-preferences`
`FTP`	`.netrc`
`Others`	`.FORTPROF .mykshrc .myprofile .nqsexample .nqsexample@T3E .nqsexample@VPP`

Alle Optionen werden auch angezeigt, wenn man gut ohne Optionen aufruft.

X-Clients auf einem entfernten Rechner starten:

- Der Client muss die Erlaubnis haben, etwas auf dem lokalen Bildschirm darstellen zu dürfen. Die Erlaubnis erteilt der X-Server.
- Der Client muss wissen, auf welchem Bildschirm er die Ausgabe umlenken soll. Das geschieht mit der Umgebungsvariable `DISPLAY`. Standardmäßig ist sie auf den lokalen Bildschirm eingestellt.
- Start des Clienten.

Erteilen der Darstellungserlaubnis:

Mit dem Befehl

```
xhost [Hostname / +]
```

kann die Erlaubnis für einen entfernten Rechner mit dem Namen `Hostname` erteilt werden, die Ausgabe an den lokalen X-Server weiterzuleiten. Die Möglichkeit + erlaubt allen fremden Rechnern den Zugriff auf den X-Server.

Importieren eines X-Servers

Man kann auch den X-Server eines entfernten Rechners auf den eigenen umleiten. Der dazu nötige Befehl lautet:

```
Xnest [Optionen]
```

Man beachte die Großschreibung! `Xnest` besitzt eine Vielzahl von Optionen. Die wichtigsten sind:

`:Nr`	Die `Nr` steht für den letzten Teil der `DISPLAY`-Variable, die der X-Server erhalten soll. Sie darf nicht identisch mit der lokalen Variable sein.
`-query Hostname`	Der `Hostname` ist der Rechner, dessen X-Windows importiert werden soll.

Setzen der DISPLAY-Variable:

Die Display-Variable besitzt standardmäßig den Wert 0:0. Die erste Null steht dabei für das erste physische Ausgabemedium (den Rechner selbst) und die zweite Null für die erste logische Ausgabeinheit (den X-Server).

Man stellt zunächst eine Verbindung zum entfernten Rechner her, d. h. man loggt sich dort ein. Dann setzt man dort die DISPLAY-Variable:

```
export DISPLAY=EigenerHostname:0
```

Danach kann die Applikation (der Client) gestartet werden.