Grundlagen der Befehlszeile

TECH STARTER

Agenda

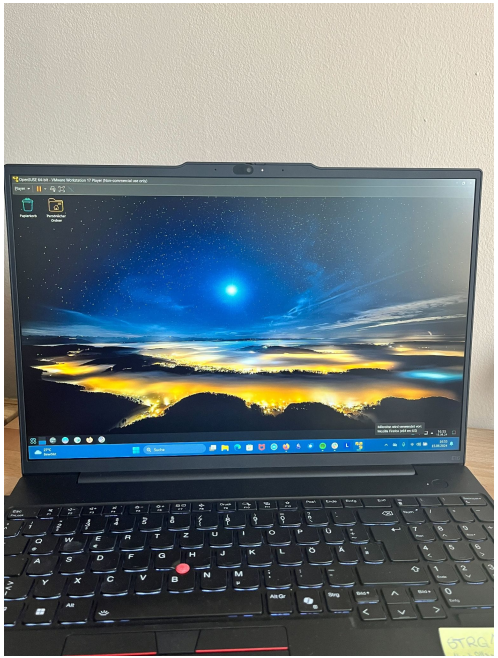
1. Shell-Grundlagen
2. Befehlszeilen-Syntax
3. Variablen, Quoting
4. Hilfe suchen über die Befehlszeile

Grundlagen der Befehlszeile

Shell-Grundlagen

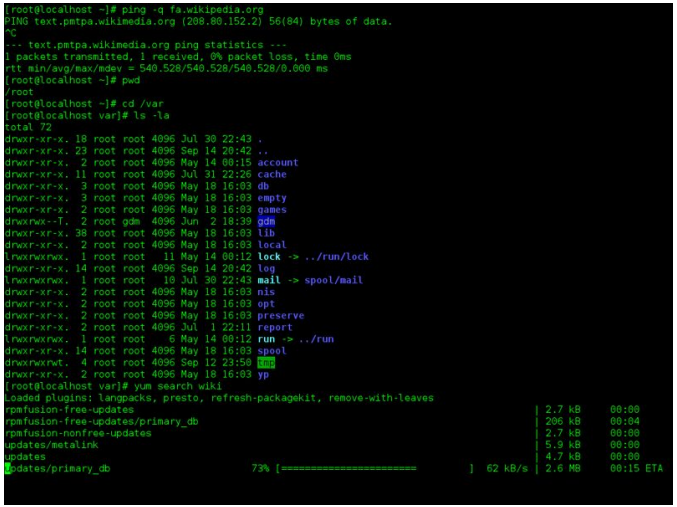
TECH
STARTER

Einführung



- Moderne Linux-Distributionen stellen grafische Benutzeroberfläche bereit
- Administratoren → Befehlszeile (Shell)
- Shell = Programm für textbasierte Kommunikation zwischen Benutzer und Betriebssystem

Shell

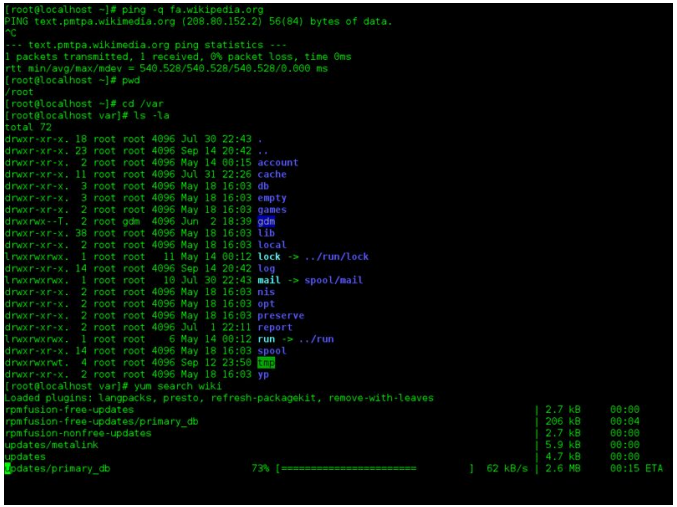


Es gibt verschiedene Shells unter Linux:

- Bourne-again shell (bash)
- C shell (csh, tcsh)
- Korn shell (ksh)
- Z shell (zsh)

TECH STARTER

Shell



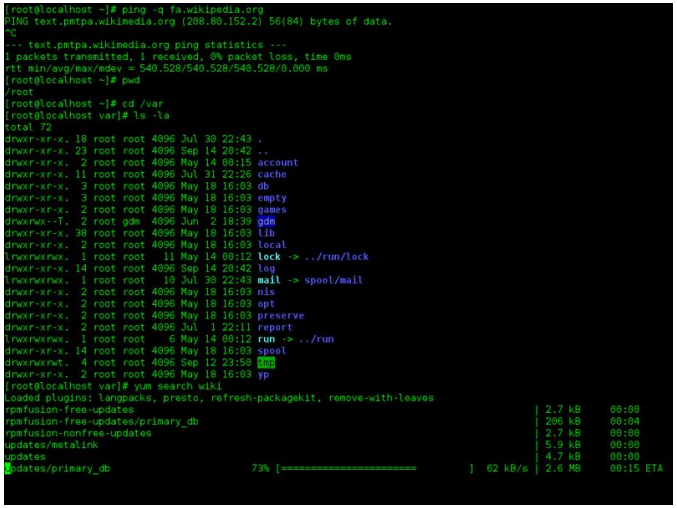
Wie funktioniert die Arbeit mit der Shell?

- Benutzer gibt Befehle an der sogenannten Eingabeaufforderung (Prompt) ein
- z.B.

```
username@hostname aktuelles_verzeichnis shell_typ
```

TECH STARTER

Shell



Wie funktioniert die Arbeit mit der Shell?

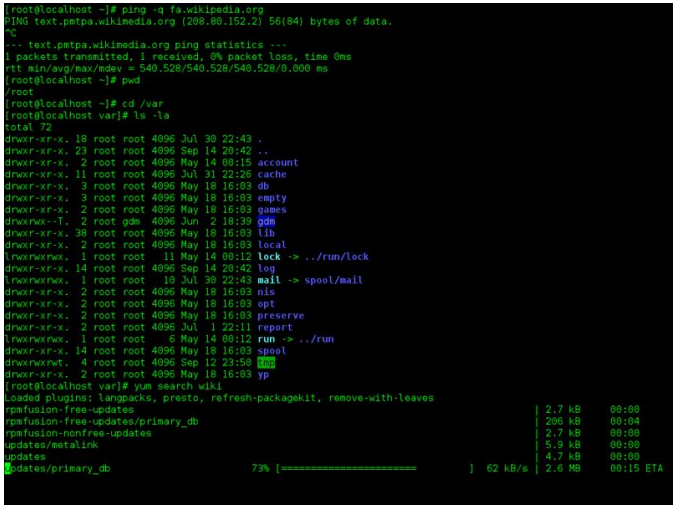
- Benutzer gibt Befehle an der
- ng (Prompt)
- Z.

username = Name des Benutzers, der die Shell ausführt

```
username@hostname aktuelles_verzeichnis shell_typ
```

TECH STARTER

Shell



Wie funktioniert die Arbeit mit der Shell?

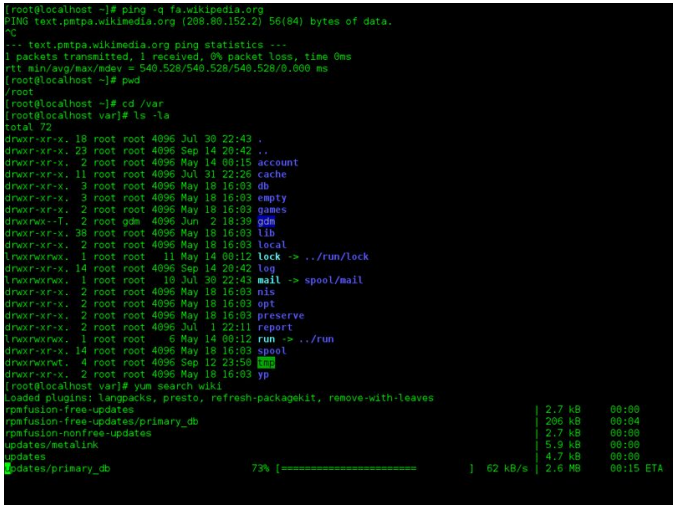
- Benutzer gibt Befehle an der so
  Ein rompt) ein
- z.B.

hostname = Name des Hosts, auf dem die Shell läuft.
[Spoiler: Es gibt auch einen Befehl hostname, mit dem man sich den Hostnamen eines Systems anzeigt

```
username@hostname aktuelles_verzeichnis shell_typ
```

TECH STARTER

Shell



Wie funktioniert die Arbeit mit der Shell?

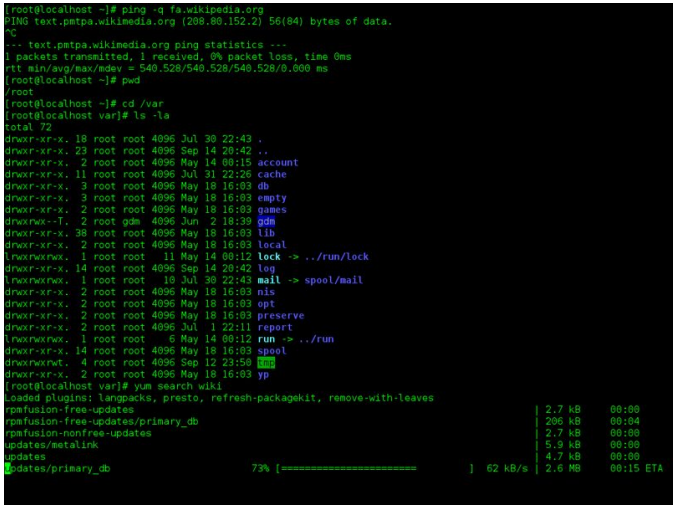
- Benutzer gibt Befehle an der sogenannte Eingabeauff ein
- z.B.

aktuelles_verzeichnis = Das Verzeichnis, in dem sich die Shell gerade befindet. Die Tilde (~) bedeutet, dass sich die Shell im Homeverzeichnis des aktuellen Benutzers befindet.

```
username@hostname aktuelles_verzeichnis shell_typ
```

TECH STARTER

Shell



Wie funktioniert die Arbeit mit der Shell?

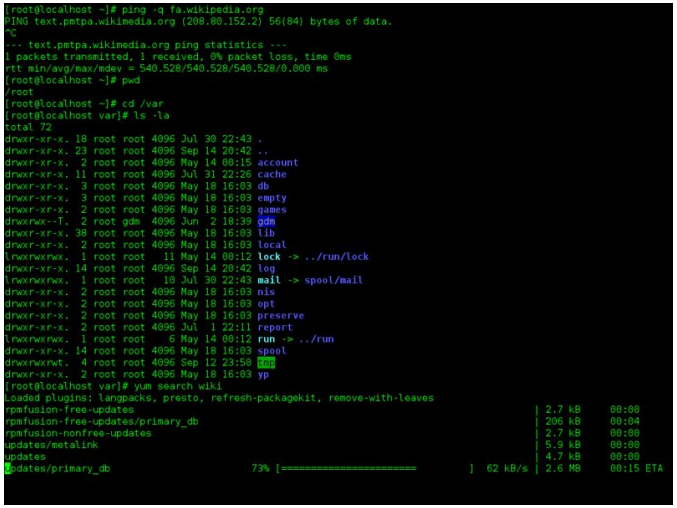
- Benutzer gibt Befehle an der sogenannten Eingabeaufforderung (ein
- z.B.

```
username@hostname aktuelles_verzeichnis shell_typ
```

shell_typ = $ zeigt an, dass die Shell von einem normalen Benutzer ausgeführt wird. # zeigt an, dass die Shell vom Superuser root ausgeführt wird

TECH STARTER

Shell



Wie funktioniert die Arbeit mit der Shell?

- Benutzer gibt Befehle an der sogenannten Eingabeaufforderung (ein
- z.B.

shell_typ = $ zeigt an, dass die Shell von einem normalen Benutzer ausgeführt wird. # zeigt an, dass die Shell vom Superuser root ausgeführt wird

```
username@hostname aktuelles_verzeichnis shell_typ
```

```
carol@mycomputer:~$
```

```
root@mycomputer:~#
```

TECH STARTER

Aufbau der Befehlszeile

```
$ ls -l /home
```

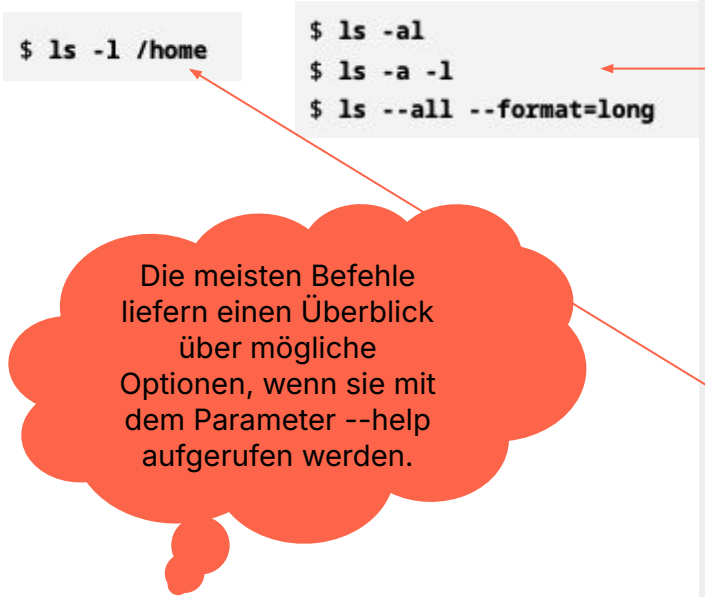
```
$ ls -al
$ ls -a -l
$ ls --all --format=long
```

- Grundstruktur:

```
Befehl  [Option(en)/Parameter...]  [Argument(e)...]
```

- **Befehl** = Programm, das der Benutzer ausführt
- **Option(en)/Parameter** = Ein “Schalter”, der das Verhalten des Befehls in irgendeiner Weise verändert. Mehrere Optionen lassen sich kombinieren, und für die Kurzform können die Buchstaben in der Regel zusammengeschrieben werden.
- **Argumente** = Zusätzliche Angaben, die vom Programm benötigt werden, etwa Dateiname oder Pfad

Aufbau der Befehlszeile



- Grundstruktur:

```
Befehl  [Option(en)/Parameter...]  [Argument(e)...]
```

- **Befehl** = Programm, das der Benutzer ausführt
- **Option(en)/Parameter** = Ein “Schalter”, der das Verhalten des Befehls in irgendeiner Weise verändert. Mehrere Optionen lassen sich kombinieren, und für die Kurzform können die Buchstaben in der Regel zusammengeschrieben werden.
- **Argumente** = Zusätzliche Angaben, die vom Programm benötigt werden, etwa Dateiname oder Pfad

Aufgabe

Teile die folgenden Zeilen in die Bestandteile Befehl, Option(en)/Parameter und Argument(e) auf

Befehl:	`cat`
Option:	`-n`
Argument:	`/etc/passwd`

1. ls -l /etc
2. ls -l -a
3. cd /home/user

Lösung

Teile die folgenden Zeilen in die Bestandteile Befehl, Option(en)/Parameter und Argument(e) auf

Befehl:	`cat`
Option:	`-n`
Argument:	`/etc/passwd`

1. ls -l /etc
   a. Befehl: ls
   b. Option: -l
   c. Argument: /etc
2. ls -l -a
   a. Befehl: ls
   b. Option: -l, -a
   c. Argument: (keines, der aktuelle Verzeichnisinhalt wird angezeigt)
3. cd /home/user
   a. Befehl: cd
   b. Option: (keine)
   c. Argument: /home/user

Befehlstypen

Der Befehl type zeigt, welchen Typs ein bestimmter Befehl ist:

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

- **Interne Befehle (Builtins)** = Diese sind Teil der Shell selbst und keine eigenständigen Programme. Es gibt etwa 30 solcher Befehle, deren Hauptzweck es ist, Aufgaben innerhalb der Shell auszuführen (z.B. cd, set, export).
- **Externe Befehle** = Diese befinden sich in einzelnen Dateien. In der Regel sind es binäre Programme oder Skripte. Wird ein Befehl ausgeführt, der kein Builtin ist, sucht die Shell mit der Variablen PATH nach einer ausführbaren Datei mit dem Namen des Befehls.

Aufgabe

Bestimme den Befehlstyp

`pwd`	Shell-Builtin
`mv`	Externer Befehl

1. cd
2. cat
3. exit
4. ls
5. echo
6. touch

Lösung

Bestimme den Befehlstyp

pwd	Shell-Builtin
mv	Externer Befehl

1. cd
   a. Typ: Builtin
   b. Erklärung: cd ist ein Shell-Builtin, was bedeutet, dass es direkt in die Shell integriert ist und nicht als eigenständiges Programm existiert.
2. cat
   a. Typ: Externer Befehl
   b. Erklärung: cat ist ein eigenständiges Programm, das sich in einem der Verzeichnisse befindet, die im $PATH definiert sind (z.B. /bin/cat).
3. exit
   a. Typ: Builtin
   b. Erklärung: exit ist ein Shell-Builtin, das verwendet wird, um die Shell zu beenden oder den Statuscode eines Skripts zu setzen.
4. ls
   a. Typ: Externer Befehl
   b. Erklärung: ls ist ein eigenständiges Programm, das in einem der Verzeichnisse im $PATH liegt (z.B. /bin/ls).
5. echo
   a. Typ: Builtin
   b. Erklärung: echo ist ein Shell-Builtin, obwohl es auch als externes Programm existieren kann (z.B. /bin/echo). In den meisten Shells wird jedoch das eingebaute echo verwendet.
6. touch
   a. Typ: Externer Befehl
   b. Erklärung: touch ist ein eigenständiges Programm, das in einem der Verzeichnisse im $PATH liegt (z.B. /usr/bin/touch).

Quoting

TWOWORDS ist eine Bash-Variable, die wir selber erstellt haben

```
$ TWOWORDS="two words"
$ touch $TWOWORDS
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 two
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 words
$ touch "$TWOWORDS"
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
$ touch '$TWOWORDS'
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 15:00 '$TWOWORDS'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
```

- Arbeit mit Dateien/Variablen → Namen mit Leerzeichen, Sonderzeichen und Variablen …
- Quoting = Kapselt Daten mit verschiedenen Arten von Anführungszeichen. Drei Arten:
  - “
  - ‘
  - \

Quoting (Doppelte Anführungszeichen)

```
$ echo I am $USER
I am tom
$ echo "I am $USER"
I am tom
```

```
$ touch new file
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 file
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 new
$ touch "new file"
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:19 new file
```

- Weisen die Shell an, den Text zwischen den Anführungszeichen als reguläre Zeichen zu übernehmen, alle Sonderzeichen verlieren ihre Bedeutung (Ausnahme: $, \, `)

Quoting (Einfaches Anführungszeichen)

```
$ echo I am $USER
I am tom
```

```
$ echo 'I am $USER'
I am $USER
```

- Kennen keine Ausnahmen wir die doppelten Anführungszeichen: Sie widerrufen jede spezielle Bedeutung für jedes Zeichen.

Quoting (Escape-Zeichen)

```
$ echo $USER
carol
```

```
$ echo \$USER
$USER
```

- Nutzen, um spezielle Bedeutungen von Zeichen aus der Bash zu entfernen.
- Anders als bei einfachen Anführungszeichen: Löst nur die Bedeutung des nachfolgenden Zeichens auf.

Aufgabe

Löse die folgenden Befehle mit Anführungszeichen auf:

1. touch “$USER”
2. touch ‘touch’

`echo "$HOME is my home directory"`	`echo /home/user is my home directory`

Lösung

Löse die folgenden Befehle mit Anführungszeichen auf:

`echo "$HOME is my home directory"`	`echo /home/user is my home directory`

1. touch “$USER”
   a. Der Befehl touch erstellt eine leere Datei oder ändert den Zeitstempel einer bestehenden Datei.
   b. "$USER" ist eine Umgebungsvariable, die den aktuellen Benutzernamen enthält. Wenn du touch "$USER" ausführst, wird eine Datei mit dem Namen deines Benutzers erstellt. Beispielsweise, wenn der Benutzername ubuntu ist, wird eine Datei mit dem Namen ubuntu erstellt.
2. touch ‘touch’
   a. Hier wird touch verwendet, um eine Datei mit dem Namen touch zu erstellen. Die einfachen Anführungszeichen ‘’ schützen den Text vor einer möglichen Interpretation als Befehl. Der Befehl erstellt also einfach eine Datei mit dem Namen touch.

Zusammenfassung Befehle

- **bash** Die beliebteste Shell auf Linux-Rechnern.
- **echo** Gibt Text im Terminal aus.
- **ls** Listet den Inhalt eines Verzeichnisses auf.
- **type** Zeigt an, wie ein bestimmter Befehl ausgeführt wird.
- **touch** Erstellt eine leere Datei oder aktualisiert das Änderungsdatum einer bestehenden Datei.
- **hostname** Zeigt oder ändert den Hostnamen eines Systems.

Übung

1. Lege mit einem Befehl und unter Verwendung der Klammer-Erweiterung (Brace Expansion) in der Bash 5 von 1 bis 5 nummerierte Dateien mit dem Präfix game an (game1, game2, ...)
2. Lösche alle 5 Dateien, die du gerade mit nur einem Befehl erstellt hast, unter Verwendung eines anderen Sonderzeichens (siehe Pathname Expansion)
3. Gibt es andere Möglichkeiten, zwei Befehle miteinander interagieren zu lassen? Welche sind das?

Variablen

- ➔ Variable = Speicher für Daten (z.B. Text, Zahlen), auf die wir später zugreifen wollen
- ➔ Haben einen Namen, über den man auf sie zugreift.
- ➔ Zwei Arten in Linux-Shells:
  - ◆ Lokale Variablen
  - ◆ Umgebungsvariablen

Variablen

- ➔ Variable = Speicher für Daten (z.B. Text, Zahlen), auf die wir später zugreifen wollen
- ➔ Haben einen Namen, über den man auf sie zugreift.
- ➔ Zwei Arten in Linux-Shells:
  - ◆ Lokale Variablen
  - ◆ Umgebungsvariablen

Variablen

Lokale Variablen	Umgebungsvariablen
● Steht nur in der **aktuellen** Shell-Sitzung zur Verfügung, d.h. Wenn wir eine lokale Variable erstellen und dann ein anderes Programm von dieser Shell aus starten, ist die Variable nicht mehr da.	● Steht in der aktuellen Shell-Sitzung **und** in Unterprozessen zur Verfügung, die aus der Sitzung hervorgegangen sind. ● Werden genutzt, um **Konfigurationsdaten** an Befehlen zu übergeben ● Da Programme auf die Variablen zugreifen können → “Umgebungsvariable” ● Meist in **Großbuchstaben** geschrieben (PATH, DATE, USER)

Variablen

Lokale Variablen	Umgebungsvariablen
• Steht nur in der **aktuellen** Shell-Sitzung zur Verfügung, d.h. Wenn wir eine lokale Variable erstellen und dann ein anderes Programm von dieser Shell aus starten, ist die Variable nicht mehr da.	• Steht in der aktuellen Shell-Sitzung **und** in Unterprozessen zur Verfügung, die aus der Sitzung hervorgegangen sind. • Werden genutzt, um **Konfigurationsdaten** an Befehlen zu übergeben • Da Programme auf die Variablen zugreifen können → “Umgebungsvariable” • Meist in **Großbuchstaben** geschrieben (PATH, DATE,

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.

Arbeit mit lokalen Variablen

- ➔ Wir setzen eine lokale Variable mit dem Operator =
- ➔ Einfache Zuweisung erstellt eine lokale Variable:
```
$ greeting=hello
```
- ➔ Wir können jede Variable mit dem Befehl echo anzeigen, der normalerweise den Text im Argumentenabschnitt anzeigt:
```
$ echo $greeting
hello
```
- ➔ Um auf den Wert der Variablen zuzugreifen, verwenden wir vor dem Variablennamen ein $
- ➔ Um eine Variable zu entfernen, nutzen wir den Befehl unset:

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

Arbeiten mit Umgebungsvariablen

- ➔ Variable für Unterprozesse verfügbar machen → Verwandle eine lokale in eine globale bzw. Umgebungsvariable mit dem Befehl **export**

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

- ➔ Einfacher: `$ export greeting=hey`
- ➔ Mit dem Befehl **env** zeigen wie alle Umgebungsvariablen an

Aufgabe

1. Erzeuge die lokale Variable number
2. Erzeuge die Umgebungsvariable ORDER mit Hilfe von zwei verschiedenen Methoden
3. Lasse dir den Namen und den Inhalt der Variablen anzeigen
4. Welche Reichweiten (Scope) haben die zuvor erzeugten Variablen?

Lösung

1. Erzeuge die lokale Variable number
   a. number=5
2. Erzeuge die Umgebungsvariable ORDER mit Hilfe von zwei verschiedenen Methoden
   a. export ORDER=12345
   b. ORDER=12345
      export ORDER
3. Lasse dir den Namen und den Inhalt der Variablen anzeigen
   a. echo "number: $number"
      echo "ORDER: $ORDER"
4. Welche Reichweiten (Scope) haben die zuvor erzeugten Variablen?
   a. Lokale Variable (number): Die lokale Variable number ist nur innerhalb der aktuellen Shell-Sitzung oder des aktuellen Skripts verfügbar. Sie ist nicht in anderen Shell-Sitzungen oder Kindprozessen verfügbar.
   b. Umgebungsvariable (ORDER): Die Umgebungsvariable ORDER ist global in der aktuellen Shell-Sitzung verfügbar und wird auch an alle Kindprozesse vererbt. Sie ist jedoch nicht in anderen parallelen Shell-Sitzungen verfügbar.

Variable PATH

- ➔ Einer der wichtigsten Umgebungsvariablen
- ➔ Sie speichert eine Liste von durch : getrennten Verzeichnissen mit ausführbaren PRogrammen, die als Befehle aus der Shell aufrufbar sind
- ➔ Um ein neues Verzeichnis anzuhängen:

```
$ 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
```

Variable PATH

- ➔ Mit Vorsicht behandeln, da wichtig für die Arbeit mit der Shell
- ➔ Um herauszufinden, wie die Shell einen bestimmten Befehl aufruft, führen wir which mit dem Namen des Befehls als Argument auf:

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

Variable PATH

- ➔ Mit Vorsicht behandeln, da wichtig für die Arbeit mit der Shell
- ➔ Um herauszufinden, wie die Shell einen bestimmten Befehl aufruft, führen wir which mit dem Namen des Befehls als Argument auf:

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

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.

Zusammenfassung Befehle

- **env** Zeigt die aktuelle Umgebung an.
- **echo** Gibt Text aus.
- **export** Macht lokale Variablen für Unterprozesse verfügbar.
- **unset** Entfernt eine Variable.

Übung

1. Erzeuge die lokale Variable nr_files und weise ihr die Anzahl der Zeilen in der Datei /etc/passwd zu. (Schau dir den Befehl wc nochmal an und die Befehlsersetzung und denk an die Anführungszeichen)
2. Erzeuge eine Umgebungsvariable ME. Weise ihr den USER als Wert zu
3. Füge den Wert der Variablen HOME an ME mit dem Trennzeichen ; an und zeige den Inhalt der Variablen ME an.
4. Erstelle eine Variable namens today und weise ihr das Datum für eine Zeitzone zu.
5. Erzeuge eine weitere Variable namens today1 und weise ihr das Systemdatum zu.

Hilfe suchen über die Befehlszeile

- ➔ Kommandozeile super komplexes Werkzeug
- ➔ Jeder Befehl hat seine eigenen Optionen → brauchen Dokumentation
- ➔ /usr/share/doc = Hier liegt die meiste Dokumentation
- ➔ Es gibt noch weitere Tools für Informationen zur Verwendung von Linux-Befehlen: z.B. man, help, info
- ➔ Zum Auffinden von bestimmten Dateien: locate

Eingebaute Hilfe

- ➔ Parameter –help
- ➔ Übersicht zur Nutzung des jew. Befehls
- ➔ Nicht alle Befehle haben diese Option!
- ➔ Im Vergleich zu anderen Dokumentationsquellen eher knapp gehalten

Manpages

```
$ man mkdir
```

- ➔ Die meisten Befehle: “Manual Page” (kurz: “Manpage”)
- ➔ Dokumentation wird mit Software installiert
- ➔ Wird mit Befehl man aufgerufen
- ➔ Navigation:
  - ◆ Pfeiltasten nach oben und unten oder Leertaste
  - ◆ mit Q schließen
- ➔ Jede Manpage hat max. 11 Abschnitte
- ➔ Manpages sind in 8 Kategorien organisiert

Manpages

Abschnitt	Beschreibung
NAME	Name des Befehls und kurze Beschreibung
SYNOPSIS	Beschreibung der Befehlssyntax
DESCRIPTION	Beschreibung der Wirkung des Befehls
OPTIONS	Verfügbare Optionen
ARGUMENTS	Verfügbare Argumente
FILES	Hilfsdateien
EXAMPLES	Ein Beispiel für den Einsatz des Befehls
SEE ALSO	Querverweise zu verwandten Themen
DIAGNOSTICS	Warn- und Fehlermeldungen
COPYRIGHT	Autor(en) des Befehls
BUGS	Bekannte Fehler und Beschränkungen des Befehls

➔ Jede Manpage hat maximal 11 Abschnitte (viele sind optionale)

Manpages

Kategorie	Beschreibung
1	Benutzerbefehle
2	Systemaufrufe
3	Funktionen der C-Bibliothek
4	Treiber und Gerätedateien
5	Konfigurationsdateien und Dateiformate
6	Spiele
7	Verschiedenes
8	Systemadministrator-Befehle
9	Kernel-Funktionen (nicht Standard)

- ➔ Manpages sind in 8 Kategorien organisiert (nummeriert von 1-8)
- ➔ Jede Manpage gehört zu genau einer Kategorie, aber mehrere Kategorien können Manpages mit gleichen Namen enthalten, z.B.
  - ◆ passwd = Passwort eines Benutzers ändern
  - ◆ Zum Einen ein Benutzerbefehl, daher Kategorie 1
  - ◆ Passwortdatenbank /etc/passwd hat auch eine Manpage mit dem Namen passwd → Kategorie 5 (Konfigurationsdatei)
  - ◆ Daher haben wir beim Verweis auf die Manpage auch den Hinweis auf entsprechende Kategorie (passwd(1) bzw. passwd(5))
  - ◆ Standardmäßig wird die erste verfügbare Manpage gezeigt, ansonsten nimmt man man 1 passwd bzw. man 5 passwd

Manpages

Kategorie	Beschreibung
1	Benutzerbefehle
2	Systemaufrufe
3	Funktionen der C-Bibliothek
4	Treiber und Gerätedateien
5	Konfigurationsdateien und Dateiformate
6	Spiele
7	Verschiedenes
8	Systemadministrator-Befehle
9	Kernel-Funktionen (nicht Standard)

- ➔ Navigation:
  - ◆ Pfeiltasten/Leertaste
  - ◆ Q zum Beenden
- ➔ Intern verwendet man den Befehl less, um den Inhalt der Manpage anzuzeigen
  - ◆ Mit less können wir nach Text innerhalb einer Manpage suchen
  - ◆ Mit /linux starten wir eine Vorwärtssuche (Mit ?linux eine Rückwärtssuche)
  - ◆ Mit N zum nächsten Treffer
  - ◆ Mit H Informationen über weitere Features

Aufgabe

Nutze den Befehl man, um herauszufinden, was die folgenden Befehle bewirken:

Befehl	Beschreibung
`ls`	Zeigt den Inhalt eines Verzeichnisses.
`cat`	
`cut`	
`cd`	
`cp`	
`mv`	
`mkdir`	
`touch`	
`wc`	
`passwd`	
`rm`	
`rmdir`	
`more`	
`less`	
`whereis`	
`head`	
`tail`	
`sort`	
`tr`	
`chmod`	
`grep`	

Lösung

Nutze den Befehl man, um herauszufinden, was die folgenden Befehle bewirken:

Befehl	Beschreibung
`ls`	Zeigt den Inhalts eines Verzeichnisses
`cat`	Verkettet Textdateien oder zeigt sie an
`cut`	Entfernt Abschnitte aus einer Textdatei
`cd`	Wechselt in ein anderes Verzeichnis
`cp`	Kopiert eine Datei
`mv`	Verschiebt eine Datei (kann auch zum Umbenennen verwendet werden)
`mkdir`	Erstellt ein neuen Verzeichnis
`touch`	Erstellt eine Datei oder ändert die Angabe zum Zeitpunkt der letzten Änderung einer bestehenden Datei
`wc`	Zählt die Anzahl der Wörter, Zeilen oder Bytes einer Datei
`passwd`	Ändert das Passwort eines Benutzers
`rm`	Löscht eine Datei
`rmdir`	Löscht ein Verzeichnis
`more`	Zeigt Textdateien Bildschirm für Bildschirm an
`less`	Zeigt Textdateien an, erlaubt Blättern zeilen- oder bildschirmweise
`whereis`	Zeigt den Pfad zu einem angegebenen Programm und den zugehörigen Dokumenationsdateien
`head`	Zeigt die ersten Zeilen einer Datei an
`tail`	Zeigt die letzten Zeilen einer Datei an
`sort`	Sortiert Dateien numerisch oder alphabetisch

Befehl	Beschreibung
`tr`	Wandelt Zeichen(folgen) in einer Datei um oder löscht sie
`chmod`	Ändert die Zugriffsrechte einer Datei
`grep`	Sucht innerhalb einer Datei

Info-Seiten

```
$ info mkdir
```

- ➔ Detaillierter als Manpages und in Hypertext formatiert
- ➔ Ähnlich wie Webseiten im Internet
- ➔ Für jede Info-Seite liest eine Info-Datei, die in einzelne Knoten innerhalb eines Baums strukturiert ist.
  - ◆ Jeder Knoten umfasst ein einfaches Thema
  - ◆ Befehl info enthält Hyperlinks, über die wir navigieren können
- ➔ Wie man hat auch info eine Seitennavigation
  - ◆ Mehr erfahren auf einer Seite über ?

Aufgabe

Öffne die info-Seite von ls und finde das MENU

1. Welche Optionen haben wir?
2. Finde die Option, mit der die Ausgabe nach dem Zeitpunkt der letzten Änderung sortiert werden kann.
3. Zeige den Pfad zu den ersten 3 README-Dateien. Verwende den Befehl man, um die richtige Option für locate zu ermitteln.

Lösung

Öffne die info-Seite von ls und finde das MENU

1. Welche Optionen haben wir?
    a. Which files are listed (Welche Dateien werden angezeigt)
    b. What information is listed (Welche Informationen werden angezeigt)
    c. Sorting the output (Ausgabe sortieren)
    d. Details about version sort (Details zur Sortierung nach Version)
    e. General output formatting (Allgemeines Ausgabeformat)
    f. Formatting file timestamps (Format von Zeitstempeln)
    g. Formatting the file names (Format von Dateinamen)
2. Finde die Option, mit der die Ausgabe nach dem Zeitpunkt der letzten Änderung sortiert werden kann.
    a. -t oder --sort=time
3. Zeige den Pfad zu den ersten 3 README-Dateien. Verwende den Befehl man, um die richtige Option für locate zu ermitteln.

```
$ locate -l 3 README
/etc/alternatives/README
/etc/init.d/README
/etc/rc0.d/README
```

/usr/share/doc/

- ➔ Verzeichnis enthält umfangreichste Dokumentation der Befehle, die das System verwendet bzw. Je ein Verzeichnis für die meisten installierten Pakete
- ➔ Name = Paketname + Versionsnummer
- ➔ README bzw. readme.txt (changelog)

Dateien suchen (locate)

```
$ locate note
/lib/udev/keymaps/zepto-znote
/usr/bin/zipnote
/usr/share/doc/initramfs-tools/maintainer-notes.html
/usr/share/man/man1/zipnote.1.gz
```

- ➔ Linux-System hat viele Verzeichnisse/Dateien
- ➔ Wir wollen eine bestimmte Datei finden mit dem Befehl locate
- ➔ locate durchsucht eine Datenbank und gibt dann jeden Namen aus, der mit der Zeichenkette übereinstimmt.
- ➔ Können auch Wildcards/reguläre Ausdrücke verwenden

Dateien suchen (locate)

```
$ locate note
/lib/udev/keymaps/zepto-znote
/usr/bin/zipnote
/usr/share/doc/initramfs-tools/maintainer-notes.html
/usr/share/man/man1/zipnote.1.gz
```

- ➔ Linux-System hat viele Verzeichnisse/Dateien
- ➔ Wir wollen eine bestimmte

Achtung: locate liest aus einer Datenbank. Die Datenbank muss natürlich aktualisiert werden, d.h. Kürzlich installierte Programme tauchen vllt nicht direkt auf. Manuell kann die Datenbank mit dem Befehl updatedb aktualisiert werden (root-Rechte)

ver iden