

corewire

Linux

Shell

Folien-Hinweis

- `Space`, `Page down`: Nächste Folie
- `Page up`: Vorherige Folie
- `ESC`, `o`: Übersicht

[Zur Kapitelübersicht](#)

Pipes und Datenströme

Standard-Datenströme

Linux arbeitet mit drei Standard-Datenströmen:

- **STDIN (0)** - Standard Input (Eingabe)
- **STDOUT (1)** - Standard Output (Ausgabe)
- **STDERR (2)** - Standard Error (Fehlerausgabe)

```
$ command < input_file      # STDIN umleiten  
$ command > output_file     # STDOUT umleiten  
$ command 2> error_file     # STDERR umleiten
```

Pipes - Befehle verknüpfen

Grundlegendes Konzept

```
$ command1 | command2          # Output von command1 als Input für command2
```

Praktische Beispiele

```
$ ls -la | grep ".txt"          # Dateien auflisten und nach .txt filtern  
$ ps aux | grep firefox        # Prozesse anzeigen und nach firefox suchen  
$ cat datei.txt | wc -l        # Zeilen in einer Datei zählen  
$ history | grep git           # Git-Befehle in der Historie suchen
```

Erweiterte Pipe-Kombinationen

Mehrstufige Pipes

```
$ ps aux | grep python | grep -v grep          # Python-Prozesse ohne grep-Zeile  
$ cat /var/log/auth.log | grep "Failed" | wc -l  # Gescheiterte Login-Versuche zählen  
$ ls -la | sort -k5 -n | head -10              # 10 kleinste Dateien anzeigen
```

Umleitung von Datenströmen

Output-Umleitung

```
$ ls > dateiliste.txt          # STDOUT in Datei (überschreibt)
$ ls >> dateiliste.txt         # STDOUT an Datei anhängen
$ command 2> errors.log        # STDERR in Datei
$ command &> all_output.txt     # STDOUT und STDERR in Datei
```

Input-Umleitung

```
$ sort < unsortiert.txt       # Datei als Input verwenden
$ mysql db_name < backup.sql  # SQL-Backup einspielen
```

Exit Codes

Exit Codes und Fehlerbehandlung

- 0 - Erfolg
- 1-255 - Verschiedene Fehlercodes

```
$ cp ordner/ kopie/          # Kopiere Ordner ohne -r
cp: -r not specified; omitting directory 'ordner/'
$ echo $?                   # Exit Code des letzten Befehls anzeigen
1
```

Umgebungsvariablen

Was sind Umgebungsvariablen?

Umgebungsvariablen speichern Informationen über die Shell-Umgebung:

- **Systemkonfiguration** (PATH, HOME, USER)
- **Programmkonfiguration** (EDITOR, BROWSER)
- **Benutzerdefinierte Variablen**

```
$ echo $HOME          # Home-Verzeichnis anzeigen
$ echo $USER          # Aktueller Benutzer
$ echo $PATH           # Suchpfad für Programme
```

Wichtige System-Variablen

Standard-Umgebungsvariablen

```
$ echo $HOME          # /home/username
$ echo $PWD           # Aktuelles Verzeichnis
$ echo $SHELL         # /bin/bash
$ echo $USER          # Benutzername
$ echo $UID           # Benutzer-ID
$ echo $TERM          # Terminal-Typ
```

Variablen setzen und verwenden

Lokale Variablen

```
$ NAME="Max Mustermann"      # Variable setzen
$ echo $NAME                  # Variable verwenden
Max Mustermann
$ echo "Hallo $NAME"          # In Strings verwenden
Hallo Max Mustermann
```

Umgebungsvariablen exportieren

```
$ export EDITOR="nano"        # Für alle Programme verfügbar machen
$ crontab -e                  # Öffnet crontab mit nano
$ env                         # Alle Umgebungsvariablen anzeigen
```

Variablen dauerhaft speichern

In .bashrc für den aktuellen Benutzer

```
$ nano ~/.bashrc
# Am Ende hinzufügen:
export EDITOR="nano"
export JAVA_HOME="/usr/lib/jvm/java-11"
export PATH="$PATH:$HOME/bin"

$ source ~/.bashrc           # Änderungen laden
```

Systemweite Variablen

```
$ sudo nano /etc/environment  # Systemweite Umgebungsvariablen
$ sudo nano /etc/profile      # Für alle Benutzer
```

Command History

History-Grundlagen

Die Shell speichert alle eingegebenen Befehle in der History:

```
$ history          # Alle Befehle anzeigen
$ history 20       # Letzten 20 Befehle
$ history | grep ssh # SSH-Befehle in History suchen
$ !123             # Befehl mit Nummer 123 ausführen
$ !!              # Letzten Befehl wiederholen
```


History-Navigation

- `Pfeil hoch/runter` - Durch History navigieren
- `Ctrl+R` - Rückwärtssuche in History
- `Ctrl+S` - Vorwärtssuche in History
- `Ctrl+G` - Suche abbrechen

Automatische Vervollständigung

Tab-Completion Grundlagen

Die Shell bietet automatische Vervollständigung:

Grundlegende Vervollständigung

- `Tab` - Einmal drücken für Vervollständigung
- `Tab Tab` - Zweimal für alle Möglichkeiten

```
$ ls Do[Tab]           # Vervollständigt zu "Documents"  
$ cd /ho[Tab]/us[Tab]/Do[Tab] # Pfad schrittweise vervollständigen
```

Erweiterte Tab-Completion

```
$ git [Tab][Tab]      # Git-Befehle  
$ ssh [Tab][Tab]      # Hosts aus ~/.ssh/known_hosts  
$ kill [Tab][Tab]     # Laufende Prozess-IDs
```

Bash-Completion installieren

```
# Ubuntu/Debian  
$ sudo apt install bash-completion  
  
# CentOS/RHEL  
$ sudo yum install bash-completion
```

Auf vielen Systemen ist bash-completion bereits vorinstalliert.

Bash-Completion konfigurieren

```
$ nano ~/.bashrc

# Bash completion aktivieren
if [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi

# Oder für neuere Systeme:
if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
fi
```

.bashrc konfigurieren

Beispiel .bashrc Konfiguration

```
$ nano ~/.bashrc

# Aliases
alias ll='ls -la'
alias la='ls -A'
alias grep='grep --color=auto'

# Prompt anpassen
export PS1='\u@\h:\w\$ '

# History
export HISTSIZE=10000
export HISTCONTROL=ignoredups

# Editor
export EDITOR=nano
```

Completion für Aliase

```
# Aliase für Git-Befehle  
alias g='git'  
  
# Autocompletion für Alias 'g'  
complete -F _git g
```


Zur Kapitelübersicht

- Vorheriges Kapitel: [Navigation & Textverarbeitung](#)
- Nächstes Kapitel: [Paketmanager](#)